

**vmath**  
vmath-0.13

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Intro</b>	<b>1</b>
1.1	Features . . . . .	1
1.2	types . . . . .	1
<b>2</b>	<b>License</b>	<b>3</b>
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	Aabb3< T > Class Template Reference . . . . .	9
5.1.1	Detailed Description . . . . .	11
5.1.2	Constructor & Destructor Documentation . . . . .	11
5.1.2.1	Aabb3() [1/5] . . . . .	11
5.1.2.2	Aabb3() [2/5] . . . . .	11
5.1.2.3	Aabb3() [3/5] . . . . .	12
5.1.2.4	Aabb3() [4/5] . . . . .	12
5.1.2.5	Aabb3() [5/5] . . . . .	13
5.1.3	Member Function Documentation . . . . .	13
5.1.3.1	center() . . . . .	13
5.1.3.2	extend() [1/2] . . . . .	13
5.1.3.3	extend() [2/2] . . . . .	13

5.1.3.4	<a href="#">extended()</a> [1/2]	14
5.1.3.5	<a href="#">extended()</a> [2/2]	14
5.1.3.6	<a href="#">extent()</a>	15
5.1.3.7	<a href="#">intersection()</a>	15
5.1.3.8	<a href="#">intersects()</a> [1/2]	15
5.1.3.9	<a href="#">intersects()</a> [2/2]	16
5.1.3.10	<a href="#">invalidate()</a>	16
5.1.3.11	<a href="#">operator &amp;()</a>	16
5.1.3.12	<a href="#">operator !=()</a>	17
5.1.3.13	<a href="#">operator*()</a>	17
5.1.3.14	<a href="#">operator*==()</a>	18
5.1.3.15	<a href="#">operator&lt;&lt;()</a> [1/2]	18
5.1.3.16	<a href="#">operator&lt;&lt;()</a> [2/2]	18
5.1.3.17	<a href="#">operator=()</a>	19
5.1.3.18	<a href="#">operator==()</a>	19
5.1.3.19	<a href="#">operator"   ()</a>	19
5.1.3.20	<a href="#">point()</a>	21
5.1.3.21	<a href="#">size()</a>	21
5.1.3.22	<a href="#">transformed()</a>	22
5.1.3.23	<a href="#">valid()</a>	22
5.1.4	<a href="#">Friends And Related Function Documentation</a>	22
5.1.4.1	<a href="#">operator&lt;&lt;</a>	22
5.1.5	<a href="#">Member Data Documentation</a>	23
5.1.5.1	<a href="#">max</a>	23
5.1.5.2	<a href="#">min</a>	23
5.2	<a href="#">Matrix3&lt; T &gt; Class Template Reference</a>	23
5.2.1	<a href="#">Detailed Description</a>	25
5.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	26
5.2.2.1	<a href="#">Matrix3()</a> [1/4]	26
5.2.2.2	<a href="#">Matrix3()</a> [2/4]	26

5.2.2.3	<a href="#">Matrix3()</a> [3/4]	26
5.2.2.4	<a href="#">Matrix3()</a> [4/4]	26
5.2.3	<a href="#">Member Function Documentation</a>	27
5.2.3.1	<a href="#">at()</a> [1/2]	27
5.2.3.2	<a href="#">at()</a> [2/2]	27
5.2.3.3	<a href="#">createRotationAroundAxis()</a>	27
5.2.3.4	<a href="#">det()</a>	28
5.2.3.5	<a href="#">fromColumnMajorArray()</a>	28
5.2.3.6	<a href="#">fromOde()</a>	28
5.2.3.7	<a href="#">fromRowMajorArray()</a>	28
5.2.3.8	<a href="#">identity()</a>	29
5.2.3.9	<a href="#">inverse()</a>	29
5.2.3.10	<a href="#">lerp()</a>	29
5.2.3.11	<a href="#">operator const T*()</a>	30
5.2.3.12	<a href="#">operator T*()</a>	30
5.2.3.13	<a href="#">operator"!=(())</a>	30
5.2.3.14	<a href="#">operator()()</a> [1/2]	31
5.2.3.15	<a href="#">operator()()</a> [2/2]	31
5.2.3.16	<a href="#">operator*()</a> [1/3]	31
5.2.3.17	<a href="#">operator*()</a> [2/3]	32
5.2.3.18	<a href="#">operator*()</a> [3/3]	32
5.2.3.19	<a href="#">operator+()</a> [1/2]	32
5.2.3.20	<a href="#">operator+()</a> [2/2]	32
5.2.3.21	<a href="#">operator-()</a> [1/2]	33
5.2.3.22	<a href="#">operator-()</a> [2/2]	33
5.2.3.23	<a href="#">operator/()</a>	33
5.2.3.24	<a href="#">operator=()</a> [1/3]	34
5.2.3.25	<a href="#">operator=()</a> [2/3]	34
5.2.3.26	<a href="#">operator=()</a> [3/3]	34
5.2.3.27	<a href="#">operator==(())</a>	35

5.2.3.28	<a href="#">toString()</a>	35
5.2.3.29	<a href="#">transpose()</a>	35
5.2.4	<a href="#">Friends And Related Function Documentation</a>	35
5.2.4.1	<a href="#">operator&lt;&lt;</a>	35
5.2.5	<a href="#">Member Data Documentation</a>	36
5.2.5.1	<a href="#">data</a>	36
5.3	<a href="#">Matrix4&lt; T &gt; Class Template Reference</a>	36
5.3.1	<a href="#">Detailed Description</a>	39
5.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	39
5.3.2.1	<a href="#">Matrix4() [1/4]</a>	39
5.3.2.2	<a href="#">Matrix4() [2/4]</a>	39
5.3.2.3	<a href="#">Matrix4() [3/4]</a>	39
5.3.2.4	<a href="#">Matrix4() [4/4]</a>	40
5.3.3	<a href="#">Member Function Documentation</a>	40
5.3.3.1	<a href="#">at() [1/2]</a>	40
5.3.3.2	<a href="#">at() [2/2]</a>	40
5.3.3.3	<a href="#">createFrustum()</a>	41
5.3.3.4	<a href="#">createLookAt()</a>	41
5.3.3.5	<a href="#">createOrtho()</a>	42
5.3.3.6	<a href="#">createRotationAroundAxis()</a>	42
5.3.3.7	<a href="#">createScale()</a>	43
5.3.3.8	<a href="#">createTranslation()</a>	43
5.3.3.9	<a href="#">det()</a>	43
5.3.3.10	<a href="#">fromColumnMajorArray()</a>	44
5.3.3.11	<a href="#">fromRowMajorArray()</a>	44
5.3.3.12	<a href="#">getScale()</a>	45
5.3.3.13	<a href="#">getTranslation()</a>	45
5.3.3.14	<a href="#">identity()</a>	45
5.3.3.15	<a href="#">inverse()</a>	45
5.3.3.16	<a href="#">lerp()</a>	45

5.3.3.17	<a href="#">operator const T *()</a>	46
5.3.3.18	<a href="#">operator T*()</a>	46
5.3.3.19	<a href="#">operator"!=( )</a>	46
5.3.3.20	<a href="#">operator()( )</a> [1/2]	47
5.3.3.21	<a href="#">operator()( )</a> [2/2]	47
5.3.3.22	<a href="#">operator*( )</a> [1/4]	47
5.3.3.23	<a href="#">operator*( )</a> [2/4]	48
5.3.3.24	<a href="#">operator*( )</a> [3/4]	48
5.3.3.25	<a href="#">operator*( )</a> [4/4]	48
5.3.3.26	<a href="#">operator+( )</a> [1/2]	49
5.3.3.27	<a href="#">operator+( )</a> [2/2]	49
5.3.3.28	<a href="#">operator-( )</a> [1/2]	49
5.3.3.29	<a href="#">operator-( )</a> [2/2]	49
5.3.3.30	<a href="#">operator/( )</a>	50
5.3.3.31	<a href="#">operator=( )</a> [1/3]	50
5.3.3.32	<a href="#">operator=( )</a> [2/3]	50
5.3.3.33	<a href="#">operator=( )</a> [3/3]	51
5.3.3.34	<a href="#">operator==( )</a>	51
5.3.3.35	<a href="#">setRotation()</a>	51
5.3.3.36	<a href="#">setScale()</a> [1/3]	52
5.3.3.37	<a href="#">setScale()</a> [2/3]	52
5.3.3.38	<a href="#">setScale()</a> [3/3]	52
5.3.3.39	<a href="#">setTranslation()</a>	52
5.3.3.40	<a href="#">toString()</a>	53
5.3.3.41	<a href="#">transpose()</a>	53
5.3.4	<a href="#">Friends And Related Function Documentation</a>	53
5.3.4.1	<a href="#">operator&lt;&lt;</a>	53
5.3.5	<a href="#">Member Data Documentation</a>	54
5.3.5.1	<a href="#">data</a>	54
5.4	<a href="#">Quaternion&lt; T &gt; Class Template Reference</a>	54

5.4.1	Detailed Description	56
5.4.2	Constructor & Destructor Documentation	56
5.4.2.1	Quaternion() [1/5]	57
5.4.2.2	Quaternion() [2/5]	57
5.4.2.3	Quaternion() [3/5]	57
5.4.2.4	Quaternion() [4/5]	57
5.4.2.5	Quaternion() [5/5]	58
5.4.3	Member Function Documentation	58
5.4.3.1	fromAxisRot()	58
5.4.3.2	fromEulerAngles()	58
5.4.3.3	fromMatrix() [1/2]	59
5.4.3.4	fromMatrix() [2/2]	59
5.4.3.5	length()	60
5.4.3.6	lengthSq()	60
5.4.3.7	lerp()	60
5.4.3.8	normalize()	61
5.4.3.9	operator"!="()	61
5.4.3.10	operator*() [1/2]	61
5.4.3.11	operator*() [2/2]	61
5.4.3.12	operator*=( ) [1/2]	62
5.4.3.13	operator*=( ) [2/2]	62
5.4.3.14	operator+()	62
5.4.3.15	operator+=( )	63
5.4.3.16	operator-( ) [1/2]	63
5.4.3.17	operator-( ) [2/2]	63
5.4.3.18	operator-=( )	63
5.4.3.19	operator=( ) [1/2]	64
5.4.3.20	operator=( ) [2/2]	64
5.4.3.21	operator==( )	64
5.4.3.22	operator~()	65

5.4.3.23	<a href="#">rotMatrix()</a>	65
5.4.3.24	<a href="#">slerp()</a>	65
5.4.3.25	<a href="#">toString()</a>	66
5.4.3.26	<a href="#">transform()</a>	66
5.4.4	<a href="#">Friends And Related Function Documentation</a>	66
5.4.4.1	<a href="#">operator&lt;&lt;</a>	66
5.4.5	<a href="#">Member Data Documentation</a>	66
5.4.5.1	<a href="#">v</a>	67
5.4.5.2	<a href="#">w</a>	67
5.5	<a href="#">Vector2&lt; T &gt; Class Template Reference</a>	67
5.5.1	<a href="#">Detailed Description</a>	69
5.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	70
5.5.2.1	<a href="#">Vector2() [1/4]</a>	70
5.5.2.2	<a href="#">Vector2() [2/4]</a>	70
5.5.2.3	<a href="#">Vector2() [3/4]</a>	70
5.5.2.4	<a href="#">Vector2() [4/4]</a>	70
5.5.3	<a href="#">Member Function Documentation</a>	71
5.5.3.1	<a href="#">length()</a>	71
5.5.3.2	<a href="#">lengthSq()</a>	71
5.5.3.3	<a href="#">lerp()</a>	71
5.5.3.4	<a href="#">normalize()</a>	72
5.5.3.5	<a href="#">operator const T *()</a>	72
5.5.3.6	<a href="#">operator T*()</a>	72
5.5.3.7	<a href="#">operator"!=(</a>	72
5.5.3.8	<a href="#">operator*() [1/2]</a>	73
5.5.3.9	<a href="#">operator*() [2/2]</a>	73
5.5.3.10	<a href="#">operator*=(</a>	73
5.5.3.11	<a href="#">operator*=(</a>	74
5.5.3.12	<a href="#">operator+() [1/2]</a>	74
5.5.3.13	<a href="#">operator+() [2/2]</a>	74

5.5.3.14	<a href="#">operator+=()</a> [1/2]	75
5.5.3.15	<a href="#">operator+=()</a> [2/2]	75
5.5.3.16	<a href="#">operator-()</a> [1/3]	75
5.5.3.17	<a href="#">operator-()</a> [2/3]	75
5.5.3.18	<a href="#">operator-()</a> [3/3]	76
5.5.3.19	<a href="#">operator-=()</a> [1/2]	76
5.5.3.20	<a href="#">operator-=()</a> [2/2]	76
5.5.3.21	<a href="#">operator/()</a> [1/2]	77
5.5.3.22	<a href="#">operator/()</a> [2/2]	77
5.5.3.23	<a href="#">operator/=()</a> [1/2]	77
5.5.3.24	<a href="#">operator/=()</a> [2/2]	77
5.5.3.25	<a href="#">operator=()</a> [1/2]	78
5.5.3.26	<a href="#">operator=()</a> [2/2]	78
5.5.3.27	<a href="#">operator==()</a>	78
5.5.3.28	<a href="#">operator[]()</a> [1/2]	79
5.5.3.29	<a href="#">operator[]()</a> [2/2]	79
5.5.3.30	<a href="#">toString()</a>	79
5.5.4	<a href="#">Friends And Related Function Documentation</a>	80
5.5.4.1	<a href="#">operator&lt;&lt;</a>	80
5.5.5	<a href="#">Member Data Documentation</a>	80
5.5.5.1	<a href="#">"@1</a>	80
5.5.5.2	<a href="#">"@3</a>	80
5.5.5.3	<a href="#">s</a>	80
5.5.5.4	<a href="#">t</a>	81
5.5.5.5	<a href="#">x</a>	81
5.5.5.6	<a href="#">y</a>	81
5.6	<a href="#">Vector3&lt; T &gt; Class Template Reference</a>	81
5.6.1	<a href="#">Detailed Description</a>	84
5.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	84
5.6.2.1	<a href="#">Vector3()</a> [1/4]	84

5.6.2.2	<a href="#">Vector3()</a> [2/4]	84
5.6.2.3	<a href="#">Vector3()</a> [3/4]	85
5.6.2.4	<a href="#">Vector3()</a> [4/4]	85
5.6.3	<a href="#">Member Function Documentation</a>	85
5.6.3.1	<a href="#">crossProduct()</a>	85
5.6.3.2	<a href="#">dotProduct()</a>	86
5.6.3.3	<a href="#">length()</a>	86
5.6.3.4	<a href="#">lengthSq()</a>	86
5.6.3.5	<a href="#">lerp()</a>	87
5.6.3.6	<a href="#">normalize()</a>	87
5.6.3.7	<a href="#">operator const T *()</a>	87
5.6.3.8	<a href="#">operator T*()</a>	88
5.6.3.9	<a href="#">operator"!=(())</a>	88
5.6.3.10	<a href="#">operator*()</a> [1/2]	88
5.6.3.11	<a href="#">operator*()</a> [2/2]	88
5.6.3.12	<a href="#">operator*==(())</a> [1/2]	89
5.6.3.13	<a href="#">operator*==(())</a> [2/2]	89
5.6.3.14	<a href="#">operator+()</a> [1/2]	89
5.6.3.15	<a href="#">operator+()</a> [2/2]	90
5.6.3.16	<a href="#">operator+==(())</a> [1/2]	90
5.6.3.17	<a href="#">operator+==(())</a> [2/2]	90
5.6.3.18	<a href="#">operator-()</a> [1/3]	91
5.6.3.19	<a href="#">operator-()</a> [2/3]	91
5.6.3.20	<a href="#">operator-()</a> [3/3]	91
5.6.3.21	<a href="#">operator-==(())</a> [1/2]	91
5.6.3.22	<a href="#">operator-==(())</a> [2/2]	92
5.6.3.23	<a href="#">operator/()</a> [1/2]	92
5.6.3.24	<a href="#">operator/()</a> [2/2]	92
5.6.3.25	<a href="#">operator/==(())</a> [1/2]	93
5.6.3.26	<a href="#">operator/==(())</a> [2/2]	93

5.6.3.27	<a href="#">operator=()</a> [1/2]	93
5.6.3.28	<a href="#">operator=()</a> [2/2]	93
5.6.3.29	<a href="#">operator==()</a>	95
5.6.3.30	<a href="#">operator[]()</a> [1/2]	95
5.6.3.31	<a href="#">operator[]()</a> [2/2]	95
5.6.3.32	<a href="#">rotate()</a>	96
5.6.3.33	<a href="#">toString()</a>	96
5.6.4	<a href="#">Friends And Related Function Documentation</a>	96
5.6.4.1	<a href="#">operator&lt;&lt;</a>	96
5.6.5	<a href="#">Member Data Documentation</a>	97
5.6.5.1	<a href="#">"@5</a>	97
5.6.5.2	<a href="#">"@7</a>	97
5.6.5.3	<a href="#">"@9</a>	97
5.6.5.4	<a href="#">b</a>	97
5.6.5.5	<a href="#">g</a>	97
5.6.5.6	<a href="#">r</a>	98
5.6.5.7	<a href="#">s</a>	98
5.6.5.8	<a href="#">t</a>	98
5.6.5.9	<a href="#">u</a>	98
5.6.5.10	<a href="#">x</a>	98
5.6.5.11	<a href="#">y</a>	98
5.6.5.12	<a href="#">z</a>	99
5.7	<a href="#">Vector4&lt; T &gt; Class Template Reference</a>	99
5.7.1	<a href="#">Detailed Description</a>	102
5.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	102
5.7.2.1	<a href="#">Vector4()</a> [1/6]	102
5.7.2.2	<a href="#">Vector4()</a> [2/6]	102
5.7.2.3	<a href="#">Vector4()</a> [3/6]	103
5.7.2.4	<a href="#">Vector4()</a> [4/6]	103
5.7.2.5	<a href="#">Vector4()</a> [5/6]	103

5.7.2.6	<a href="#">Vector4()</a> [6/6]	103
5.7.3	<a href="#">Member Function Documentation</a>	103
5.7.3.1	<a href="#">length()</a>	104
5.7.3.2	<a href="#">lengthSq()</a>	104
5.7.3.3	<a href="#">lerp()</a>	104
5.7.3.4	<a href="#">normalize()</a>	105
5.7.3.5	<a href="#">operator const T *()</a>	105
5.7.3.6	<a href="#">operator T*()</a>	105
5.7.3.7	<a href="#">operator"! =()</a>	105
5.7.3.8	<a href="#">operator*()</a> [1/2]	106
5.7.3.9	<a href="#">operator*()</a> [2/2]	106
5.7.3.10	<a href="#">operator*==()</a> [1/2]	106
5.7.3.11	<a href="#">operator*==()</a> [2/2]	107
5.7.3.12	<a href="#">operator+()</a> [1/2]	107
5.7.3.13	<a href="#">operator+()</a> [2/2]	107
5.7.3.14	<a href="#">operator+=()</a> [1/2]	108
5.7.3.15	<a href="#">operator+=()</a> [2/2]	108
5.7.3.16	<a href="#">operator-()</a> [1/3]	108
5.7.3.17	<a href="#">operator-()</a> [2/3]	108
5.7.3.18	<a href="#">operator-()</a> [3/3]	109
5.7.3.19	<a href="#">operator-=()</a> [1/2]	109
5.7.3.20	<a href="#">operator-=()</a> [2/2]	109
5.7.3.21	<a href="#">operator/()</a> [1/2]	110
5.7.3.22	<a href="#">operator/()</a> [2/2]	110
5.7.3.23	<a href="#">operator/=()</a> [1/2]	110
5.7.3.24	<a href="#">operator/=()</a> [2/2]	110
5.7.3.25	<a href="#">operator=()</a> [1/2]	111
5.7.3.26	<a href="#">operator=()</a> [2/2]	111
5.7.3.27	<a href="#">operator==()</a>	111
5.7.3.28	<a href="#">operator[]()</a> [1/2]	112

5.7.3.29	<code>operator[]()</code> [2/2]	112
5.7.3.30	<code>toString()</code>	112
5.7.3.31	<code>xyz()</code>	113
5.7.4	Friends And Related Function Documentation	113
5.7.4.1	<code>operator&lt;&lt;</code>	113
5.7.5	Member Data Documentation	113
5.7.5.1	<code>"@11</code>	113
5.7.5.2	<code>"@13</code>	114
5.7.5.3	<code>"@15</code>	114
5.7.5.4	<code>"@17</code>	114
5.7.5.5	<code>a</code>	114
5.7.5.6	<code>b</code>	114
5.7.5.7	<code>g</code>	114
5.7.5.8	<code>r</code>	115
5.7.5.9	<code>w</code>	115
5.7.5.10	<code>x</code>	115
5.7.5.11	<code>y</code>	115
5.7.5.12	<code>z</code>	115
<b>6</b>	<b>File Documentation</b>	<b>117</b>
6.1	<code>src/vmath.cpp</code> File Reference	117
6.2	<code>src/vmath.h</code> File Reference	117
6.2.1	Macro Definition Documentation	120
6.2.1.1	<code>DEG2RAD</code>	120
6.2.1.2	<code>EPSILON</code>	120
6.2.1.3	<code>M_PI</code>	120
6.2.1.4	<code>VEC2</code>	120
6.2.1.5	<code>VEC3</code>	120
6.2.1.6	<code>VEC4</code>	121
6.2.2	Typedef Documentation	121
6.2.2.1	<code>Aabb3d</code>	121

6.2.2.2	Aabb3f	121
6.2.2.3	Matrix3d	121
6.2.2.4	Matrix3f	121
6.2.2.5	Matrix3i	121
6.2.2.6	Matrix4d	122
6.2.2.7	Matrix4f	122
6.2.2.8	Matrix4i	122
6.2.2.9	Quatd	122
6.2.2.10	Quatf	122
6.2.2.11	Vector2d	122
6.2.2.12	Vector2f	123
6.2.2.13	Vector2i	123
6.2.2.14	Vector3d	123
6.2.2.15	Vector3f	123
6.2.2.16	Vector3i	123
6.2.2.17	Vector4d	123
6.2.2.18	Vector4f	124
6.2.2.19	Vector4i	124
6.2.3	Function Documentation	124
6.2.3.1	max() [1/3]	124
6.2.3.2	max() [2/3]	124
6.2.3.3	max() [3/3]	125
6.2.3.4	min() [1/3]	125
6.2.3.5	min() [2/3]	125
6.2.3.6	min() [3/3]	126
6.2.4	Variable Documentation	126
6.2.4.1	epsilon	126



# Chapter 1

## Intro

Vector mathematics for computer graphics

### 1.1 Features

- basic arithmetic operations - using operators
- basic linear algebra operations - such as transpose, dot product, etc.
- aliases for vertex coordinates - it means:

```
Vector3f v;  
// use vertex coordinates  
v.x = 1; v.y = 2; v.z = -1;  
  
// use texture coordinates  
v.s = 0; v.t = 1; v.u = 0.5;  
// use color coordinates  
v.r = 1; v.g = 0.5; v.b = 0;
```

- conversion constructor and assign operators - so you can assign a value of `Vector3<T1>` type to a variable of `Vector3<T2>` type for any convertible T1, T2 type pairs. In other words, you can do this:

```
Vector3f f3; Vector3d d3 = f3;  
...  
f3 = d3;
```

### 1.2 types

- `Vector2` Two dimensional vector
  - float — `Vector2f`
  - double — `Vector2d`
  - int — `Vector2i`

- [Vector3](#) Three dimensional vector
  - float — Vector3f
  - double — Vector3d
  - int — Vector3i
- [Vector4](#) Four dimensional vector
  - float — Vector4f
  - double — Vector4d
  - int — Vector4i
- [Matrix3](#) Matrix 3x3
  - float — Matrix3f
  - double — Matrix3d
  - int — Matrix3i
- [Matrix4](#) Matrix 4x4
  - float — Matrix4f
  - double — Matrix4d
  - int — Matrix4i
- [Quaternion](#)
  - float — Quatf
  - double — Quatd
- [Aabb3](#) axes-aligned bounding-box
  - float — Aabb3f
  - double — Aabb3d

## Chapter 2

# License

vmath, set of classes for computer graphics mathematics.

Copyright (c) 2005-2018, Jan Bartipan < barzto at gmail dot com > All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Aabb3&lt; T &gt;</a>	Axes-aligned bounding-box (aka AABB) class . . . . .	9
<a href="#">Matrix3&lt; T &gt;</a>	Class for matrix 3x3 . . . . .	23
<a href="#">Matrix4&lt; T &gt;</a>	Class for matrix 4x4 . . . . .	36
<a href="#">Quaternion&lt; T &gt;</a>	<a href="#">Quaternion</a> class implementing some quaternion algebra operations . . . . .	54
<a href="#">Vector2&lt; T &gt;</a>	Class for two dimensional vector . . . . .	67
<a href="#">Vector3&lt; T &gt;</a>	Class for three dimensional vector . . . . .	81
<a href="#">Vector4&lt; T &gt;</a>	Class for four dimensional vector . . . . .	99



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">vmath.cpp</a> . . . . .	117
src/ <a href="#">vmath.h</a> . . . . .	117



## Chapter 5

# Class Documentation

### 5.1 Aabb3< T > Class Template Reference

Axes-aligned bounding-box (aka AABB) class.

```
#include <vmath.h>
```

#### Public Member Functions

- [Aabb3](#) ()  
*Constructs invalid axes-aligned bounding-box.*
- template<typename SrcT >  
[Aabb3](#) (const [Vector3](#)< SrcT > &point)  
*Constructs axes-aligned bound-box containing one point point.*
- template<typename SrcT >  
[Aabb3](#) (SrcT x0, SrcT y0, SrcT z0, SrcT x1, SrcT y1, SrcT z1)  
*Constructs axes-aligned bounding-box form two corner points (x0, y0, z0) and (x1, y1, z1)*
- template<typename SrcT >  
[Aabb3](#) (SrcT x, SrcT y, SrcT z)  
*Constructs axes-aligned bounding-box containing point (x, y, z)*
- template<typename SrcT >  
[Aabb3](#) (const [Aabb3](#)< SrcT > &src)  
*Creates copy of axis-aligned bounding-box.*
- template<typename SrcT >  
[Aabb3](#)< T > & [operator=](#) (const [Aabb3](#)< SrcT > &rhs)  
*Assign operator.*
- bool [valid](#) () const  
*Checks if bounding-box is valid.*
- void [invalidate](#) ()  
*Makes this bounding-box invalid.*
- template<typename SrcT >  
void [extend](#) (const [Vector3](#)< SrcT > &point)  
*Extends this bounding-box by a point point.*
- template<typename SrcT >  
void [extend](#) (const [Aabb3](#)< SrcT > &box)  
*Extends this bounding-box by a box box.*

- `template<typename SrcT >`  
`Aabb3< T > extended (const Vector3< SrcT > &point) const`  
*Gets a copy of this bounding-box extend by a point point.*
- `template<typename SrcT >`  
`Aabb3< T > extended (const Aabb3< SrcT > &box) const`  
*Gets a copy of this bounding-box extended by box box.*
- `template<typename SrcT >`  
`bool intersects (const Vector3< SrcT > &point) const`  
*Tests if the point point is within this bounding-box.*
- `template<typename SrcT >`  
`bool intersects (const Aabb3< SrcT > &box) const`  
*Tests if other bounding-box box intersects (even partially) with this bounding-box.*
- `template<typename SrcT >`  
`Aabb3< T > intersection (const Aabb3< SrcT > &other) const`  
*Gets result of intersection of this bounding-box with other bounding-box.*
- `Vector3< T > center () const`  
*Gets center point of bounding-box.*
- `Vector3< T > extent () const`  
*Gets extent of bounding-box.*
- `Vector3< T > size () const`  
*Gets diagonal size of bounding-box.*
- `Vector3< T > point (size_t i) const`  
*Gets all 8 corner-points of bounding box.*
- `Aabb3< T > transformed (const Matrix4< T > &t) const`  
*Gets transformed bounding-box by transform t.*
- `template<typename RhsT >`  
`bool operator== (const Aabb3< RhsT > &rhs) const`  
*Tests if rhs is equal to this bounding-box.*
- `template<typename RhsT >`  
`bool operator!= (const Aabb3< RhsT > &rhs) const`  
*Tests if rhs is not equal to this bounding-box.*
- `Aabb3< T > operator* (const Matrix4< T > &rhs) const`  
*Gets transformed bounding-box by transform rhs.*
- `Aabb3< T > & operator*= (const Matrix4< T > &rhs)`  
*Apply transform rhs to this bounding-box.*
- `template<typename SrcT >`  
`Aabb3< T > & operator<< (const Vector3< SrcT > &rhs)`  
*Extends this bounding-box by point rhs.*
- `template<typename SrcT >`  
`Aabb3< T > & operator<< (const Aabb3< SrcT > &rhs)`  
*Extends this bounding-box by box rhs.*
- `template<typename RhsT >`  
`Aabb3< T > operator| (const Aabb3< RhsT > &rhs) const`  
*Union of this and rhs bounding-boxes.*
- `template<typename RhsT >`  
`Aabb3< T > operator & (const Aabb3< RhsT > &rhs) const`  
*Intersection of this and rhs bounding-boxes.*

## Public Attributes

- `Vector3< T > min`  
*Position of Min corner of bounding box.*
- `Vector3< T > max`  
*Position of Max corner of bounding box.*

## Friends

- `std::ostream & operator<< (std::ostream &lhs, const Aabb3< T > &rhs)`  
*Outputs string representation of bounding-box rhs to output stream lhs.*

### 5.1.1 Detailed Description

```
template<typename T>
class Aabb3< T >
```

Axes-aligned bounding-box (aka AABB) class.

This class provides functionality for:

- creating AABB from point, or other AABB,
- testing if point of other AABB intersects with,
- getting result of intersection with other AABB,
- transforming AABB with 4x4 matrix.

There are also overloaded couple of operators to shorten common operation. For instance you can use `operator<<` on AABB to extend it with passed point or other AABB.

```
Aabb3f aabb;
aabb << Vector3f(1, 1, 2) << Aabb3f(-3,-3,-3, 2, 2, 2);
```

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 Aabb3() [1/5]

```
template<typename T>
Aabb3< T >::Aabb3 ( ) [inline]
```

Constructs invalid axes-aligned bounding-box.

See also

[valid\(\)](#) for explanation of invalid bounding-box usage.

#### 5.1.2.2 Aabb3() [2/5]

```
template<typename T>
template<typename SrcT >
Aabb3< T >::Aabb3 (
    const Vector3< SrcT > & point ) [inline]
```

Constructs axes-aligned bound-box containing one point *point*.

## Parameters

<i>point</i>	
--------------	--

5.1.2.3 **Aabb3()** [3/5]

```
template<typename T>
template<typename SrcT >
Aabb3< T >::Aabb3 (
    SrcT x0,
    SrcT y0,
    SrcT z0,
    SrcT x1,
    SrcT y1,
    SrcT z1 ) [inline]
```

Constructs axes-aligned bounding-box form two corner points ( $x0, y0, z0$ ) and ( $x1, y1, z1$ )

## Parameters

<i>x0</i>	X-coordinate of first point
<i>y0</i>	Y-coordinate of first point
<i>z0</i>	Z-coordinate of first point
<i>x1</i>	X-coordinate of second point
<i>y1</i>	Y-coordinate of second point
<i>z1</i>	Z-coordinate of second point

5.1.2.4 **Aabb3()** [4/5]

```
template<typename T>
template<typename SrcT >
Aabb3< T >::Aabb3 (
    SrcT x,
    SrcT y,
    SrcT z ) [inline]
```

Constructs axes-aligned bounding-box containing point ( $x, y, z$ )

## Parameters

<i>x</i>	X-coordinate of point
<i>y</i>	Y-coordinate of point
<i>z</i>	Z-coordinate of point

## 5.1.2.5 Aabb3() [5/5]

```
template<typename T>
template<typename SrcT >
Aabb3< T >::Aabb3 (
    const Aabb3< SrcT > & src ) [inline]
```

Creates copy of axis-aligned bounding-box.

## Parameters

<i>src</i>	Source bounding-box
------------	---------------------

## 5.1.3 Member Function Documentation

## 5.1.3.1 center()

```
template<typename T>
Vector3<T> Aabb3< T >::center ( ) const [inline]
```

Gets center point of bounding-box.

## Returns

A center point of bounding-box.

## 5.1.3.2 extend() [1/2]

```
template<typename T>
template<typename SrcT >
void Aabb3< T >::extend (
    const Vector3< SrcT > & point ) [inline]
```

Extends this bounding-box by a point *point*.

## Parameters

<i>point</i>	A point to extend bounding-box by.
--------------	------------------------------------

## 5.1.3.3 extend() [2/2]

```
template<typename T>
```

```
template<typename SrcT >
void Aabb3< T >::extend (
    const Aabb3< SrcT > & box ) [inline]
```

Extends this bounding-box by a box *box*.

#### Parameters

<i>box</i>	A box to extend this bounding-box by.
------------	---------------------------------------

#### 5.1.3.4 extended() [1/2]

```
template<typename T>
template<typename SrcT >
Aabb3<T> Aabb3< T >::extended (
    const Vector3< SrcT > & point ) const [inline]
```

Gets a copy of this bounding-box extend by a point *point*.

#### Parameters

<i>point</i>	A point to extend the box by
--------------	------------------------------

#### Returns

Copy of extended bounding-box

#### 5.1.3.5 extended() [2/2]

```
template<typename T>
template<typename SrcT >
Aabb3<T> Aabb3< T >::extended (
    const Aabb3< SrcT > & box ) const [inline]
```

Gets a copy of this bounding-box extnended by box *box*.

#### Parameters

<i>box</i>	A box to extend the copy be.
------------	------------------------------

#### Returns

Copy of extended bounding-box

## 5.1.3.6 extent()

```
template<typename T>
Vector3<T> Aabb3< T >::extent ( ) const [inline]
```

Gets extent of bounding-box.

## Returns

Extent of bounding-box.

## 5.1.3.7 intersection()

```
template<typename T>
template<typename SrcT >
Aabb3<T> Aabb3< T >::intersection (
    const Aabb3< SrcT > & other ) const [inline]
```

Gets result of intersection of this bounding-box with *other* bounding-box.

In case the boxes don't intersect, the returned bounding-box is invalid.

## Parameters

<i>other</i>	Box to be tested
--------------	------------------

## Returns

Result of intersection.

## See also

[valid\(\)](#) method for more information on invalid bounding-boxes.

## 5.1.3.8 intersects() [1/2]

```
template<typename T>
template<typename SrcT >
bool Aabb3< T >::intersects (
    const Vector3< SrcT > & point ) const [inline]
```

Tests if the point *point* is within this bounding-box.

## Parameters

<i>point</i>	A point to be tested
--------------	----------------------

**Returns**

True if point *point* lies within bounding-box, otherwise false.

**5.1.3.9 intersects()** [2/2]

```
template<typename T>
template<typename SrcT >
bool Aabb3< T >::intersects (
    const Aabb3< SrcT > & box ) const [inline]
```

Tests if other bounding-box *box* intersects (even partially) with this bounding-box.

**Parameters**

<i>box</i>	A box to be tested for intersection.
------------	--------------------------------------

**Returns**

True if there's intersection between boxes, otherwise false.

**5.1.3.10 invalidate()**

```
template<typename T>
void Aabb3< T >::invalidate ( ) [inline]
```

Makes this bounding-box invalid.

So calling [valid\(\)](#) gets false.

**See also**

[valid\(\)](#) method for more info on usage of invalid bounding-boxes.

**5.1.3.11 operator &()**

```
template<typename T>
template<typename RhsT >
Aabb3<T> Aabb3< T >::operator& (
    const Aabb3< RhsT > & rhs ) const [inline]
```

Intersection of this and *rhs* bounding-boxed.

## Parameters

<i>rhs</i>	Right-hand side
------------	-----------------

## Returns

Resulting bounding-box representing the intersection.

## 5.1.3.12 operator!=()

```
template<typename T>
template<typename RhsT >
bool Aabb3< T >::operator!= (
    const Aabb3< RhsT > & rhs ) const [inline]
```

Tests if *rhs* is not equal to this bounding-box.

## Parameters

<i>rhs</i>	Right-hand side
------------	-----------------

## Returns

True if *rhs* and this bounding-boxes are not equal, otherwise false

## 5.1.3.13 operator\*()

```
template<typename T>
Aabb3<T> Aabb3< T >::operator* (
    const Matrix4< T > & rhs ) const [inline]
```

Gets transformed bounding-box by transform *rhs*.

## Parameters

<i>rhs</i>	Matrix 4x4 representing the transform
------------	---------------------------------------

## Returns

Transformed bounding-box

#### 5.1.3.14 operator\*=( )

```
template<typename T>
Aabb3<T>& Aabb3< T >::operator*= (
    const Matrix4< T > & rhs ) [inline]
```

Apply transform *rhs* to this bounding-box.

##### Parameters

<i>rhs</i>	A transform to be applied
------------	---------------------------

##### Returns

Reference to this

#### 5.1.3.15 operator<<() [1/2]

```
template<typename T>
template<typename SrcT >
Aabb3<T>& Aabb3< T >::operator<< (
    const Vector3< SrcT > & rhs ) [inline]
```

Extends this bounding-box by point *rhs*.

##### Parameters

<i>rhs</i>	A point to extend this bounding-box by
------------	--

##### Returns

Reference to this

#### 5.1.3.16 operator<<() [2/2]

```
template<typename T>
template<typename SrcT >
Aabb3<T>& Aabb3< T >::operator<< (
    const Aabb3< SrcT > & rhs ) [inline]
```

Extends this bounding-box by box *rhs*.

##### Parameters

<i>rhs</i>	A box to extend this bounding-box by
------------	--------------------------------------

**Returns**

Reference to this

**5.1.3.17 operator=()**

```
template<typename T>
template<typename SrcT >
Aabb3<T>& Aabb3< T >::operator= (
    const Aabb3< SrcT > & rhs ) [inline]
```

Assign operator.

**Parameters**

<i>rhs</i>	source bounding-box
------------	---------------------

**Returns**

reference to this

**5.1.3.18 operator==( )**

```
template<typename T>
template<typename RhsT >
bool Aabb3< T >::operator==(
    const Aabb3< RhsT > & rhs ) const [inline]
```

Tests if *rhs* is equal to this bounding-box.

**Parameters**

<i>rhs</i>	Right-hand side
------------	-----------------

**Returns**

True if *rhs* and this bounding-boxes are equal, otherwise false

**5.1.3.19 operator" | ()**

```
template<typename T>
template<typename RhsT >
```

```
Aabb3<T> Aabb3< T >::operator| (  
    const Aabb3< RhsT > & rhs ) const [inline]
```

Union of this and *rhs* bounding-boxes.

## Parameters

<i>rhs</i>	Right-hand side of union
------------	--------------------------

## Returns

A resulting bounding-box representing union

## 5.1.3.20 point()

```
template<typename T>
Vector3<T> Aabb3< T >::point (
    size_t i ) const [inline]
```

Gets all 8 corner-points of bounding box.

## Parameters

<i>i</i>	An index of bounding-box corner point. Valid values are 0 .. 7.
----------	---

## Returns

A position of *i-th* corner-point.

## Note

The order of points is as follows (where + denotes max-point and – min-point):

1. (+ + +)
2. (– + +)
3. (+ – +)
4. (– – +)
5. (+ + –)
6. (– + –)
7. (+ – –)
8. (– – –)

## 5.1.3.21 size()

```
template<typename T>
Vector3<T> Aabb3< T >::size ( ) const [inline]
```

Gets diagonal size of bounding-box.

## Returns

Sizes for particular dimensions.

### 5.1.3.22 transformed()

```
template<typename T>
Aabb3<T> Aabb3< T >::transformed (
    const Matrix4< T > & t ) const [inline]
```

Gets transformed bounding-box by transform *t*.

#### Parameters

<i>t</i>	A transform matrix
----------	--------------------

#### Returns

Transformed bounding-box

### 5.1.3.23 valid()

```
template<typename T>
bool Aabb3< T >::valid ( ) const [inline]
```

Checks if bounding-box is valid.

Valid bounding-box has non-negative size. If an invalid bounding-box is extended by point or another bounding-box, the target bounding box becomes valid and contains solely the source point or bounding-box respectively.

#### Returns

True if box is valid, otherwise false

## 5.1.4 Friends And Related Function Documentation

### 5.1.4.1 operator<<

```
template<typename T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Aabb3< T > & rhs ) [friend]
```

Outputs string representation of bounding-box *rhs* to output stream *lhs*.

#### Parameters

<i>lhs</i>	Output stream to write to
<i>rhs</i>	Bounding-box to write to output stream.

### Returns

Reference to output stream *lhs*

## 5.1.5 Member Data Documentation

### 5.1.5.1 max

```
template<typename T>  
Vector3<T> Aabb3< T >::max
```

Position of Max corner of bounding box.

### 5.1.5.2 min

```
template<typename T>  
Vector3<T> Aabb3< T >::min
```

Position of Min corner of bounding box.

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

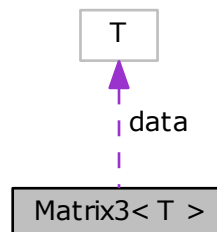
- [src/vmath.h](#)

## 5.2 Matrix3< T > Class Template Reference

Class for matrix 3x3.

```
#include <vmath.h>
```

Collaboration diagram for Matrix3< T >:



## Public Member Functions

- [Matrix3](#) ()  
*Creates identity matrix.*
- [Matrix3](#) (const T \*dt)  
*Copy matrix values from array (these data must be in column major order!)*
- [Matrix3](#) (const [Matrix3](#)< T > &src)  
*Copy constructor.*
- template<class FromT >  
[Matrix3](#) (const [Matrix3](#)< FromT > &src)  
*Copy casting constructor.*
- void [identity](#) ()  
*Resets matrix to be identity matrix.*
- bool [operator==](#) (const [Matrix3](#)< T > &rhs) const  
*Equality test operator.*
- bool [operator!=](#) (const [Matrix3](#)< T > &rhs) const  
*Inequality test operator.*
- T & [at](#) (int x, int y)  
*Get reference to element at position (x,y).*
- const T & [at](#) (int x, int y) const  
*Get constant reference to element at position (x,y).*
- T & [operator\(\)](#) (int i, int j)  
*Get reference to element at position (i,j), with math matrix notation.*
- const T & [operator\(\)](#) (int i, int j) const  
*Get constant reference to element at position (i,j), with math matrix notation.*
- [Matrix3](#)< T > & [operator=](#) (const [Matrix3](#)< T > &rhs)  
*Copy operator.*
- template<class FromT >  
[Matrix3](#)< T > & [operator=](#) (const [Matrix3](#)< FromT > &rhs)  
*Copy casting operator.*
- [Matrix3](#)< T > & [operator=](#) (const T \*rhs)  
*Copy operator.*
- [Matrix3](#)< T > [operator+](#) (const [Matrix3](#)< T > &rhs) const  
*Addition operator.*
- [Matrix3](#)< T > [operator-](#) (const [Matrix3](#)< T > &rhs) const  
*Subtraction operator.*
- [Matrix3](#)< T > [operator+](#) (T rhs) const  
*Addition operator.*
- [Matrix3](#)< T > [operator-](#) (T rhs) const  
*Subtraction operator.*
- [Matrix3](#)< T > [operator\\*](#) (T rhs) const  
*Multiplication operator.*
- [Matrix3](#)< T > [operator/](#) (T rhs) const  
*Division operator.*
- [Vector3](#)< T > [operator\\*](#) (const [Vector3](#)< T > &rhs) const  
*Multiplication operator.*
- [Matrix3](#)< T > [operator\\*](#) ([Matrix3](#)< T > rhs) const  
*Multiplication operator.*
- [Matrix3](#)< T > [transpose](#) ()  
*Transpose matrix.*
- [Matrix3](#)< T > [lerp](#) (T fact, const [Matrix3](#)< T > &rhs) const

- *Linear interpolation of two matrices.*
- `T det ()`
- `Matrix3< T > inverse ()`  
*Computes inverse matrix.*
- `operator T* ()`  
*Conversion to pointer operator.*
- `operator const T * () const`  
*Conversion to pointer operator.*
- `std::string toString () const`  
*Gets string representation.*

## Static Public Member Functions

- static `Matrix3< T > createRotationAroundAxis (T xDeg, T yDeg, T zDeg)`  
*Creates rotation matrix by rotation around axis.*
- `template<class It >`  
static `Matrix3< T > fromOde (const It *mat)`  
*Creates rotation matrix from ODE Matrix.*
- `template<class FromT >`  
static `Matrix3< T > fromRowMajorArray (const FromT *arr)`  
*Creates new matrix 3x3 from array that represents such matrix 3x3 as array of tightly packed elements in row major order.*
- `template<class FromT >`  
static `Matrix3< T > fromColumnMajorArray (const FromT *arr)`  
*Creates new matrix 3x3 from array that represents such matrix 3x3 as array of tightly packed elements in column major order.*

## Public Attributes

- `T data [9]`  
*Data stored in column major order.*

## Friends

- `std::ostream & operator<< (std::ostream &lhs, const Matrix3< T > &rhs)`  
*Output to stream operator.*

### 5.2.1 Detailed Description

```
template<class T>
class Matrix3< T >
```

Class for matrix 3x3.

#### Note

Data stored in this matrix are in column major order. This arrangement suits OpenGL. If you're using row major matrix, consider using `fromRowMajorArray` as way for construction `Matrix3<T>` instance.

## 5.2.2 Constructor & Destructor Documentation

### 5.2.2.1 `Matrix3()` [1/4]

```
template<class T>
Matrix3< T >::Matrix3 ( ) [inline]
```

Creates identity matrix.

### 5.2.2.2 `Matrix3()` [2/4]

```
template<class T>
Matrix3< T >::Matrix3 (
    const T * dt ) [inline]
```

Copy matrix values from array (these data must be in column major order!)

### 5.2.2.3 `Matrix3()` [3/4]

```
template<class T>
Matrix3< T >::Matrix3 (
    const Matrix3< T > & src ) [inline]
```

Copy constructor.

#### Parameters

<code>src</code>	Data source for new created instance of <a href="#">Matrix3</a>
------------------	---

### 5.2.2.4 `Matrix3()` [4/4]

```
template<class T>
template<class FromT >
Matrix3< T >::Matrix3 (
    const Matrix3< FromT > & src ) [inline]
```

Copy casting constructor.

#### Parameters

<code>src</code>	Data source for new created instance of <a href="#">Matrix3</a>
------------------	---

### 5.2.3 Member Function Documentation

#### 5.2.3.1 at() [1/2]

```
template<class T>
T& Matrix3< T >::at (
    int x,
    int y ) [inline]
```

Get reference to element at position (x,y).

##### Parameters

<i>x</i>	Number of column (0..2)
<i>y</i>	Number of row (0..2)

#### 5.2.3.2 at() [2/2]

```
template<class T>
const T& Matrix3< T >::at (
    int x,
    int y ) const [inline]
```

Get constant reference to element at position (x,y).

##### Parameters

<i>x</i>	Number of column (0..2)
<i>y</i>	Number of row (0..2)

#### 5.2.3.3 createRotationAroundAxis()

```
template<class T>
static Matrix3<T> Matrix3< T >::createRotationAroundAxis (
    T xDeg,
    T yDeg,
    T zDeg ) [inline], [static]
```

Creates rotation matrix by rotation around axis.

##### Parameters

<i>xDeg</i>	Angle (in degrees) of rotation around axis X.
<i>yDeg</i>	Angle (in degrees) of rotation around axis Y.
<i>zDeg</i>	Angle (in degrees) of rotation around axis Z.

#### 5.2.3.4 det()

```
template<class T>
T Matrix3< T >::det ( ) [inline]
```

#### 5.2.3.5 fromColumnMajorArray()

```
template<class T>
template<class FromT >
static Matrix3<T> Matrix3< T >::fromColumnMajorArray (
    const FromT * arr ) [inline], [static]
```

Creates new matrix 3x3 from array that represents such matrix 3x3 as array of tightly packed elements in column major order.

##### Parameters

<i>arr</i>	An array of elements for 3x3 matrix in column major order.
------------	--

##### Returns

An instance of Matrix3<T> representing *arr*

#### 5.2.3.6 fromOde()

```
template<class T>
template<class It >
static Matrix3<T> Matrix3< T >::fromOde (
    const It * mat ) [inline], [static]
```

Creates rotation matrix from ODE Matrix.

#### 5.2.3.7 fromRowMajorArray()

```
template<class T>
template<class FromT >
static Matrix3<T> Matrix3< T >::fromRowMajorArray (
    const FromT * arr ) [inline], [static]
```

Creates new matrix 3x3 from array that represents such matrix 3x3 as array of tightly packed elements in row major order.

**Parameters**

<i>arr</i>	An array of elements for 3x3 matrix in row major order.
------------	---

**Returns**

An instance of Matrix3<T> representing *arr*

**5.2.3.8 identity()**

```
template<class T>
void Matrix3< T >::identity ( ) [inline]
```

Resets matrix to be identity matrix.

**5.2.3.9 inverse()**

```
template<class T>
Matrix3<T> Matrix3< T >::inverse ( ) [inline]
```

Computes inverse matrix.

**Returns**

Inverse matrix of this matrix.

**5.2.3.10 lerp()**

```
template<class T>
Matrix3<T> Matrix3< T >::lerp (
    T fact,
    const Matrix3< T > & rhs ) const [inline]
```

Linear interpolation of two matrices.

**Parameters**

<i>fact</i>	Factor of interpolation. For translation from position of this matrix (lhs) to matrix rhs, values of factor goes from 0.0 to 1.0.
<i>rhs</i>	Second Matrix for interpolation

**Note**

However values of fact parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

**5.2.3.11 operator const T \*()**

```
template<class T>
Matrix3< T >::operator const T * ( ) const [inline]
```

Conversion to pointer operator.

**Returns**

Constant Pointer to internally stored (in management of class Matrix3<T>) used for passing Matrix3<T> values to gl\*[fd]v functions.

**5.2.3.12 operator T\*()**

```
template<class T>
Matrix3< T >::operator T* ( ) [inline]
```

Conversion to pointer operator.

**Returns**

Pointer to internally stored (in management of class Matrix3<T>) used for passing Matrix3<T> values to gl\*[fd]v functions.

**5.2.3.13 operator"!=()**

```
template<class T>
bool Matrix3< T >::operator!= (
    const Matrix3< T > & rhs ) const [inline]
```

Inequality test operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Returns

not (lhs == rhs) :-P

## 5.2.3.14 operator() [1/2]

```
template<class T>
T& Matrix3< T >::operator() (
    int i,
    int j ) [inline]
```

Get reference to element at position (i,j), with math matrix notation.

## Parameters

<i>i</i>	Number of row (1..3)
<i>j</i>	Number of column (1..3)

## 5.2.3.15 operator() [2/2]

```
template<class T>
const T& Matrix3< T >::operator() (
    int i,
    int j ) const [inline]
```

Get constant reference to element at position (i,j), with math matrix notation.

## Parameters

<i>i</i>	Number of row (1..3)
<i>j</i>	Number of column (1..3)

## 5.2.3.16 operator\*() [1/3]

```
template<class T>
Matrix3<T> Matrix3< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.17 `operator*()` [2/3]

```
template<class T>
Vector3<T> Matrix3< T >::operator* (
    const Vector3< T > & rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.18 `operator*()` [3/3]

```
template<class T>
Matrix3<T> Matrix3< T >::operator* (
    Matrix3< T > rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.19 `operator+()` [1/2]

```
template<class T>
Matrix3<T> Matrix3< T >::operator+ (
    const Matrix3< T > & rhs ) const [inline]
```

Addition operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.20 `operator+()` [2/2]

```
template<class T>
```

```
Matrix3<T> Matrix3< T >::operator+ (
    T rhs ) const [inline]
```

Addition operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.2.3.21 operator-() [1/2]

```
template<class T>
Matrix3<T> Matrix3< T >::operator- (
    const Matrix3< T > & rhs ) const [inline]
```

Subtraction operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.2.3.22 operator-() [2/2]

```
template<class T>
Matrix3<T> Matrix3< T >::operator- (
    T rhs ) const [inline]
```

Subtraction operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.2.3.23 operator/()

```
template<class T>
Matrix3<T> Matrix3< T >::operator/ (
    T rhs ) const [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.24 `operator=()` [1/3]

```
template<class T>
Matrix3<T>& Matrix3< T >::operator= (
    const Matrix3< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.25 `operator=()` [2/3]

```
template<class T>
template<class FromT >
Matrix3<T>& Matrix3< T >::operator= (
    const Matrix3< FromT > & rhs ) [inline]
```

Copy casting operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.2.3.26 `operator=()` [3/3]

```
template<class T>
Matrix3<T>& Matrix3< T >::operator= (
    const T * rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.2.3.27 operator==( )

```
template<class T>
bool Matrix3< T >::operator== (
    const Matrix3< T > & rhs ) const [inline]
```

Equality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Note

Test of equality is based on threshold EPSILON value. To be two values equal, must satisfy this condition all elements of matrix  $|lhs[i] - rhs[i]| < EPSILON$ , same for y-coordinate, z-coordinate, and w-coordinate.

## 5.2.3.28 toString()

```
template<class T>
std::string Matrix3< T >::toString ( ) const [inline]
```

Gets string representation.

## 5.2.3.29 transpose()

```
template<class T>
Matrix3<T> Matrix3< T >::transpose ( ) [inline]
```

Transpose matrix.

## 5.2.4 Friends And Related Function Documentation

## 5.2.4.1 operator&lt;&lt;

```
template<class T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Matrix3< T > & rhs ) [friend]
```

Output to stream operator.

**Parameters**

<i>lhs</i>	Left hand side argument of operator (commonly ostream instance).
<i>rhs</i>	Right hand side argument of operator.

**Returns**

Left hand side argument - the ostream object passed to operator.

**5.2.5 Member Data Documentation****5.2.5.1 data**

```
template<class T>
T Matrix3< T >::data[9]
```

Data stored in column major order.

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

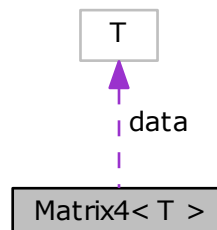
- [src/vmath.h](#)

**5.3 Matrix4< T > Class Template Reference**

Class for matrix 4x4.

```
#include <vmath.h>
```

Collaboration diagram for Matrix4< T >:



## Public Member Functions

- [Matrix4](#) ()  
*Creates identity matrix.*
- [Matrix4](#) (const T \*dt)  
*Copy matrix values from array (these data must be in column major order!)*
- [Matrix4](#) (const [Matrix4](#)< T > &src)  
*Copy constructor.*
- template<class FromT >  
[Matrix4](#) (const [Matrix4](#)< FromT > &src)  
*Copy casting constructor.*
- void [identity](#) ()  
*Resets matrix to be identity matrix.*
- bool [operator==](#) (const [Matrix4](#)< T > &rhs) const  
*Equality test operator.*
- bool [operator!=](#) (const [Matrix4](#)< T > &rhs) const  
*Inequality test operator.*
- T & [at](#) (int x, int y)  
*Get reference to element at position (x,y).*
- const T & [at](#) (int x, int y) const  
*Get constant reference to element at position (x,y).*
- T & [operator\(\)](#) (int i, int j)  
*Get reference to element at position (i,j), with math matrix notation.*
- const T & [operator\(\)](#) (int i, int j) const  
*Get constant reference to element at position (i,j), with math matrix notation.*
- void [setTranslation](#) (const [Vector3](#)< T > &v)  
*Sets translation part of matrix.*
- [Vector3](#)< T > [getTranslation](#) () const
- void [setRotation](#) (const [Matrix3](#)< T > &m)  
*Sets rotation part (matrix 3x3) of matrix.*
- [Vector3](#)< T > [getScale](#) () const  
*Gets matrix scale.*
- void [setScale](#) (T s)  
*Sets matrix uniform scale values.*
- void [setScale](#) (T sx, T sy, T sz)  
*Sets matrix scale for all axes.*
- void [setScale](#) (const [Vector3](#)< T > &s)  
*Sets matrix scale for all axes.*
- [Matrix4](#)< T > & [operator=](#) (const [Matrix4](#)< T > &rhs)  
*Copy operator.*
- template<class FromT >  
[Matrix4](#)< T > & [operator=](#) (const [Matrix4](#)< FromT > &rhs)  
*Copy casting operator.*
- [Matrix4](#)< T > & [operator=](#) (const T \*rhs)  
*Copy operator.*
- [Matrix4](#)< T > [operator+](#) (const [Matrix4](#)< T > &rhs) const  
*Addition operator.*
- [Matrix4](#)< T > [operator-](#) (const [Matrix4](#)< T > &rhs) const  
*Subtraction operator.*
- [Matrix4](#)< T > [operator+](#) (T rhs) const  
*Addition operator.*

- [Matrix4< T > operator-](#) (T rhs) const  
*Subtraction operator.*
- [Matrix4< T > operator\\*](#) (T rhs) const  
*Multiplication operator.*
- [Matrix4< T > operator/](#) (T rhs) const  
*Division operator.*
- [Vector4< T > operator\\*](#) (const [Vector4< T > &rhs](#)) const  
*Multiplication operator.*
- [Vector3< T > operator\\*](#) (const [Vector3< T > &rhs](#)) const  
*Multiplication operator.*
- [Matrix4< T > operator\\*](#) ([Matrix4< T > rhs](#)) const  
*Multiplication operator.*
- [T det](#) ()  
*Computes determinant of matrix.*
- [Matrix4< T > inverse](#) ()  
*Computes inverse matrix.*
- [Matrix4< T > transpose](#) ()  
*Transpose matrix.*
- [Matrix4< T > lerp](#) (T fact, const [Matrix4< T > &rhs](#)) const  
*Linear interpolation of two matrices.*
- [operator T\\*](#) ()  
*Conversion to pointer operator.*
- [operator const T \\*](#) () const  
*Conversion to pointer operator.*
- [std::string toString](#) () const  
*Gets string representation.*

## Static Public Member Functions

- static [Matrix4< T > createRotationAroundAxis](#) (T xDeg, T yDeg, T zDeg)  
*Creates rotation matrix by rotation around axis.*
- static [Matrix4< T > createTranslation](#) (T x, T y, T z, T w=1)  
*Creates translation matrix.*
- static [Matrix4< T > createScale](#) (T sx, T sy, T sz)  
*Create scale matrix with sx, sy, and sz being values of matrix main diagonal.*
- static [Matrix4< T > createLookAt](#) (const [Vector3< T > &eyePos](#), const [Vector3< T > &centerPos](#), const [Vector3< T > &upDir](#))  
*Creates new view matrix to look from specified position eyePos to specified position centerPos.*
- static [Matrix4< T > createFrustum](#) (T left, T right, T bottom, T top, T zNear, T zFar)  
*Creates OpenGL compatible perspective projection according specified frustum parameters.*
- static [Matrix4< T > createOrtho](#) (T left, T right, T bottom, T top, T zNear, T zFar)  
*Creates OpenGL compatible orthographic projection matrix.*
- template<class FromT >  
static [Matrix4< T > fromRowMajorArray](#) (const FromT \*arr)  
*Creates new matrix 4x4 from array that represents such matrix 4x4 as array of tightly packed elements in row major order.*
- template<class FromT >  
static [Matrix4< T > fromColumnMajorArray](#) (const FromT \*arr)  
*Creates new matrix 4x4 from array that represents such matrix 4x4 as array of tightly packed elements in column major order.*

## Public Attributes

- T `data` [16]  
*Data stored in column major order.*

## Friends

- std::ostream & `operator<<` (std::ostream &lhs, const Matrix4< T > &rhs)  
*Output to stream operator.*

### 5.3.1 Detailed Description

```
template<class T>
class Matrix4< T >
```

Class for matrix 4x4.

#### Note

Data stored in this matrix are in column major order. This arrangement suits OpenGL. If you're using row major matrix, consider using `fromRowMajorArray` as way for construction Matrix4<T> instance.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Matrix4() [1/4]

```
template<class T>
Matrix4< T >::Matrix4 ( ) [inline]
```

Creates identity matrix.

#### 5.3.2.2 Matrix4() [2/4]

```
template<class T>
Matrix4< T >::Matrix4 (
    const T * dt ) [inline]
```

Copy matrix values from array (these data must be in column major order!)

#### 5.3.2.3 Matrix4() [3/4]

```
template<class T>
Matrix4< T >::Matrix4 (
    const Matrix4< T > & src ) [inline]
```

Copy constructor.

## Parameters

<code>src</code>	Data source for new created instance of <a href="#">Matrix4</a> .
------------------	---

5.3.2.4 `Matrix4()` [4/4]

```
template<class T>
template<class FromT >
Matrix4< T >::Matrix4 (
    const Matrix4< FromT > & src ) [inline]
```

Copy casting constructor.

## Parameters

<code>src</code>	Data source for new created instance of <a href="#">Matrix4</a> .
------------------	---

## 5.3.3 Member Function Documentation

5.3.3.1 `at()` [1/2]

```
template<class T>
T& Matrix4< T >::at (
    int x,
    int y ) [inline]
```

Get reference to element at postion (x,y).

## Parameters

<code>x</code>	Number of column (0..3)
<code>y</code>	Number of row (0..3)

5.3.3.2 `at()` [2/2]

```
template<class T>
const T& Matrix4< T >::at (
    int x,
    int y ) const [inline]
```

Get constant reference to element at position (x,y).

## Parameters

<i>x</i>	Number of column (0..3)
<i>y</i>	Number of row (0..3)

## 5.3.3.3 createFrustum()

```
template<class T>
static Matrix4<T> Matrix4< T >::createFrustum (
    T left,
    T right,
    T bottom,
    T top,
    T zNear,
    T zFar ) [inline], [static]
```

Creates OpenGL compatible perspective projection according specified frustum parameters.

## Parameters

<i>left</i>	Specify the coordinate for the left vertical clipping plane,
<i>right</i>	Specify the coordinate for the right vertical clipping plane.
<i>bottom</i>	Specify the coordinate for the bottom horizontal clipping plane,
<i>top</i>	Specify the coordinate for the top horizontal clipping plane.
<i>zNear</i>	Specify the distance to the near clipping plane. Distance must be positive.
<i>zFar</i>	Specify the distance to the far depth clipping plane. Distance must be positive.

## Returns

Projection matrix for specified frustum.

## 5.3.3.4 createLookAt()

```
template<class T>
static Matrix4<T> Matrix4< T >::createLookAt (
    const Vector3< T > & eyePos,
    const Vector3< T > & centerPos,
    const Vector3< T > & upDir ) [inline], [static]
```

Creates new view matrix to look from specified position *eyePos* to specified position *centerPos*.

## Parameters

<i>eyePos</i>	A position of camera
<i>centerPos</i>	A position where camera looks-at
<i>upDir</i>	Direction of up vector

**Returns**

Resulting view matrix that looks from and at specific position.

**5.3.3.5 createOrtho()**

```
template<class T>
static Matrix4<T> Matrix4< T >::createOrtho (
    T left,
    T right,
    T bottom,
    T top,
    T zNear,
    T zFar ) [inline], [static]
```

Creates OpenGL compatible orthographic projection matrix.

**Parameters**

<i>left</i>	Specify the coordinate for the left vertical clipping plane,
<i>right</i>	Specify the coordinate for the right vertical clipping plane.
<i>bottom</i>	Specify the coordinate for the bottom horizontal clipping plane,
<i>top</i>	Specify the coordinate for the top horizontal clipping plane.
<i>zNear</i>	Specify the distance to the nearer depth clipping plane. This value is negative if the plane is to be behind the viewer,
<i>zFar</i>	Specify the distance to the farther depth clipping plane. This value is negative if the plane is to be behind the viewer.

**Returns**

Othrographic projection matrix.

**5.3.3.6 createRotationAroundAxis()**

```
template<class T>
static Matrix4<T> Matrix4< T >::createRotationAroundAxis (
    T xDeg,
    T yDeg,
    T zDeg ) [inline], [static]
```

Creates rotation matrix by rotation around axis.

**Parameters**

<i>xDeg</i>	Angle (in degrees) of rotation around axis X.
<i>yDeg</i>	Angle (in degrees) of rotation around axis Y.
<i>zDeg</i>	Angle (in degrees) of rotation around axis Z.

## 5.3.3.7 createScale()

```
template<class T>
static Matrix4<T> Matrix4< T >::createScale (
    T sx,
    T sy,
    T sz ) [inline], [static]
```

Create scale matrix with *sx*, *sy*, and *sz* being values of matrix main diagonal.

## Parameters

<i>sx</i>	Scale in X-axis
<i>sy</i>	Scale in Y-axis
<i>sz</i>	Scale in Z-axis

## Returns

Transform matrix 4x4 with scale transformation.

## 5.3.3.8 createTranslation()

```
template<class T>
static Matrix4<T> Matrix4< T >::createTranslation (
    T x,
    T y,
    T z,
    T w = 1 ) [inline], [static]
```

Creates translation matrix.

Creates translation matrix.

## Parameters

<i>x</i>	X-direction translation
<i>y</i>	Y-direction translation
<i>z</i>	Z-direction translation
<i>w</i>	for W-coordinate translation (implicitly set to 1)

## 5.3.3.9 det()

```
template<class T>
T Matrix4< T >::det ( ) [inline]
```

Computes determinant of matrix.

#### Returns

Determinant of matrix

#### Note

This function does  $3 * 4 * 6$  muls,  $3 * 6$  adds.

#### 5.3.3.10 fromColumnMajorArray()

```
template<class T>
template<class FromT >
static Matrix4<T> Matrix4< T >::fromColumnMajorArray (
    const FromT * arr ) [inline], [static]
```

Creates new matrix 4x4 from array that represents such matrix 4x4 as array of tightly packed elements in column major order.

#### Parameters

<i>arr</i>	An array of elements for 4x4 matrix in column major order.
------------	--

#### Returns

An instance of Matrix4<T> representing *arr*

#### 5.3.3.11 fromRowMajorArray()

```
template<class T>
template<class FromT >
static Matrix4<T> Matrix4< T >::fromRowMajorArray (
    const FromT * arr ) [inline], [static]
```

Creates new matrix 4x4 from array that represents such matrix 4x4 as array of tightly packed elements in row major order.

#### Parameters

<i>arr</i>	An array of elements for 4x4 matrix in row major order.
------------	---

#### Returns

An instance of Matrix4<T> representing *arr*

#### 5.3.3.12 getScale()

```
template<class T>
Vector3<T> Matrix4< T >::getScale ( ) const [inline]
```

Gets matrix scale.

##### Returns

Scales (i.e. first three values from matrix diagonal).

#### 5.3.3.13 getTranslation()

```
template<class T>
Vector3<T> Matrix4< T >::getTranslation ( ) const [inline]
```

#### 5.3.3.14 identity()

```
template<class T>
void Matrix4< T >::identity ( ) [inline]
```

Resets matrix to be identity matrix.

#### 5.3.3.15 inverse()

```
template<class T>
Matrix4<T> Matrix4< T >::inverse ( ) [inline]
```

Computes inverse matrix.

##### Returns

Inverse matrix of this matrix.

##### Note

This is a little bit time consuming operation (16 \* 6 \* 3 muls, 16 \* 5 adds + [det\(\)](#) + mul() functions)

#### 5.3.3.16 lerp()

```
template<class T>
Matrix4<T> Matrix4< T >::lerp (
    T fact,
    const Matrix4< T > & rhs ) const [inline]
```

Linear interpolation of two matrices.

## Parameters

<i>fact</i>	Factor of interpolation. For translation from position of this matrix (lhs) to matrix rhs, values of factor goes from 0.0 to 1.0.
<i>rhs</i>	Second Matrix for interpolation

## Note

However values of fact parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

## 5.3.3.17 operator const T \*()

```
template<class T>
Matrix4< T >::operator const T * ( ) const [inline]
```

Conversion to pointer operator.

## Returns

Constant Pointer to internally stored (in management of class Matrix4<T>) used for passing Matrix4<T> values to gl\*[fd]v functions.

## 5.3.3.18 operator T\*()

```
template<class T>
Matrix4< T >::operator T* ( ) [inline]
```

Conversion to pointer operator.

## Returns

Pointer to internally stored (in management of class Matrix4<T>) used for passing Matrix4<T> values to gl\*[fd]v functions.

## 5.3.3.19 operator!=(())

```
template<class T>
bool Matrix4< T >::operator!= (
    const Matrix4< T > & rhs ) const [inline]
```

Inequality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Returns

not (lhs == rhs) :-P

## 5.3.3.20 operator&gt;() [1/2]

```
template<class T>
T& Matrix4< T >::operator() (
    int i,
    int j ) [inline]
```

Get reference to element at position (i,j), with math matrix notation.

## Parameters

<i>i</i>	Number of row (1..4)
<i>j</i>	Number of column (1..4)

## 5.3.3.21 operator&gt;() [2/2]

```
template<class T>
const T& Matrix4< T >::operator() (
    int i,
    int j ) const [inline]
```

Get constant reference to element at position (i,j), with math matrix notation.

## Parameters

<i>i</i>	Number of row (1..4)
<i>j</i>	Number of column (1..4)

## 5.3.3.22 operator\*() [1/4]

```
template<class T>
Matrix4<T> Matrix4< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.3.3.23 `operator*()` [2/4]

```
template<class T>
Vector4<T> Matrix4< T >::operator* (
    const Vector4< T > & rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.3.3.24 `operator*()` [3/4]

```
template<class T>
Vector3<T> Matrix4< T >::operator* (
    const Vector3< T > & rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.3.3.25 `operator*()` [4/4]

```
template<class T>
Matrix4<T> Matrix4< T >::operator* (
    Matrix4< T > rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.3.3.26 operator+() [1/2]

```
template<class T>
Matrix4<T> Matrix4< T >::operator+ (
    const Matrix4< T > & rhs ) const [inline]
```

Addition operator.

##### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.3.3.27 operator+() [2/2]

```
template<class T>
Matrix4<T> Matrix4< T >::operator+ (
    T rhs ) const [inline]
```

Addition operator.

##### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.3.3.28 operator-() [1/2]

```
template<class T>
Matrix4<T> Matrix4< T >::operator- (
    const Matrix4< T > & rhs ) const [inline]
```

Subtraction operator.

##### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.3.3.29 operator-() [2/2]

```
template<class T>
Matrix4<T> Matrix4< T >::operator- (
    T rhs ) const [inline]
```

Subtraction operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.3.3.30 operator/()

```
template<class T>
Matrix4<T> Matrix4< T >::operator/ (
    T rhs ) const [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.3.3.31 operator=() [1/3]

```
template<class T>
Matrix4<T>& Matrix4< T >::operator= (
    const Matrix4< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.3.3.32 operator=() [2/3]

```
template<class T>
template<class FromT >
Matrix4<T>& Matrix4< T >::operator= (
    const Matrix4< FromT > & rhs ) [inline]
```

Copy casting operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.3.3.33 operator=()** [3/3]

```
template<class T>
Matrix4<T>& Matrix4< T >::operator= (
    const T * rhs ) [inline]
```

Copy operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.3.3.34 operator==()**

```
template<class T>
bool Matrix4< T >::operator== (
    const Matrix4< T > & rhs ) const [inline]
```

Equality test operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Note**

Test of equality is based of threshold EPSILON value. To be two values equal, must satisfy this condition all elements of matrix  $|lhs[i] - rhs[i]| < EPSILON$ , same for y-coordinate, z-coordinate, and w-coordinate.

**5.3.3.35 setRotation()**

```
template<class T>
void Matrix4< T >::setRotation (
    const Matrix3< T > & m ) [inline]
```

Sets rotation part (matrix 3x3) of matrix.

**Parameters**

<i>m</i>	Rotation part of matrix
----------	-------------------------

**5.3.3.36 setScale()** [1/3]

```
template<class T>
void Matrix4< T >::setScale (
    T s ) [inline]
```

Sets matrix uniform scale values.

**Parameters**

<b>s</b>	Uniform scale value
----------	---------------------

**5.3.3.37 setScale()** [2/3]

```
template<class T>
void Matrix4< T >::setScale (
    T sx,
    T sy,
    T sz ) [inline]
```

Sets matrix scale for all axes.

**Parameters**

<b>sx</b>	X-axis scale factor
<b>sy</b>	Y-axis scale factor
<b>sz</b>	Z-axis scale factor

**5.3.3.38 setScale()** [3/3]

```
template<class T>
void Matrix4< T >::setScale (
    const Vector3< T > & s ) [inline]
```

Sets matrix scale for all axes.

**Parameters**

<b>s</b>	Scale factors for X, Y, and Z coordinate.
----------	---

**5.3.3.39 setTranslation()**

```
template<class T>
```

```
void Matrix4< T >::setTranslation (
    const Vector3< T > & v ) [inline]
```

Sets translation part of matrix.

#### Parameters

<i>v</i>	Vector of translation to be set.
----------	----------------------------------

#### 5.3.3.40 toString()

```
template<class T>
std::string Matrix4< T >::toString ( ) const [inline]
```

Gets string representation.

#### 5.3.3.41 transpose()

```
template<class T>
Matrix4<T> Matrix4< T >::transpose ( ) [inline]
```

Transpose matrix.

### 5.3.4 Friends And Related Function Documentation

#### 5.3.4.1 operator<<

```
template<class T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Matrix4< T > & rhs ) [friend]
```

Output to stream operator.

#### Parameters

<i>lhs</i>	Left hand side argument of operator (commonly ostream instance).
<i>rhs</i>	Right hand side argument of operator.

#### Returns

Left hand side argument - the ostream object passed to operator.

### 5.3.5 Member Data Documentation

#### 5.3.5.1 data

```
template<class T>
T Matrix4< T >::data[16]
```

Data stored in column major order.

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

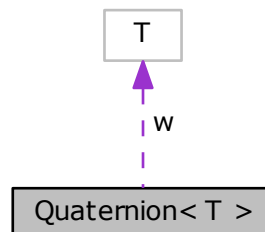
- [src/vmath.h](#)

## 5.4 Quaternion< T > Class Template Reference

[Quaternion](#) class implementing some quaternion algebra operations.

```
#include <vmath.h>
```

Collaboration diagram for Quaternion< T >:



## Public Member Functions

- [Quaternion](#) ()  
*Quaternion constructor, sets quaternion to  $(0 + 0i + 0j + 0k)$ .*
- [Quaternion](#) (const [Quaternion](#)< T > &q)  
*Copy constructor.*
- template<class FromT >  
[Quaternion](#) (const [Quaternion](#)< FromT > &q)  
*Copy casting constructor.*
- [Quaternion](#) (T w\_, const [Vector3](#)< T > &v\_)  
*Creates quaternion object from real part w\_ and complex part v\_.*
- [Quaternion](#) (T w\_, T x, T y, T z)  
*Creates quaternion object from value  $(w_ + xi + yj + zk)$ .*
- [Quaternion](#)< T > & [operator=](#) (const [Quaternion](#)< T > &rhs)  
*Copy operator.*
- template<class FromT >  
[Quaternion](#)< T > & [operator=](#) (const [Quaternion](#)< FromT > &rhs)  
*Copy convert operator.*
- [Quaternion](#)< T > [operator+](#) (const [Quaternion](#)< T > &rhs) const  
*Addition operator.*
- [Quaternion](#)< T > [operator\\*](#) (const [Quaternion](#)< T > &rhs) const  
*Multiplication operator.*
- [Quaternion](#)< T > [operator\\*](#) (T rhs) const  
*Multiplication operator.*
- [Quaternion](#)< T > [operator-](#) (const [Quaternion](#)< T > &rhs) const  
*Subtraction operator.*
- [Quaternion](#)< T > & [operator+=](#) (const [Quaternion](#)< T > &rhs)  
*Addition operator.*
- [Quaternion](#)< T > & [operator-=](#) (const [Quaternion](#)< T > &rhs)  
*Subtraction operator.*
- [Quaternion](#)< T > & [operator\\*=](#) (const [Quaternion](#)< T > &rhs)  
*Multiplication operator.*
- [Quaternion](#)< T > & [operator\\*=](#) (T rhs)  
*Multiplication operator.*
- bool [operator==](#) (const [Quaternion](#)< T > &rhs) const  
*Equality test operator.*
- bool [operator!=](#) (const [Quaternion](#)< T > &rhs) const  
*Inequality test operator.*
- [Quaternion](#)< T > [operator-](#) () const  
*Unary negate operator.*
- [Quaternion](#)< T > [operator~](#) () const  
*Unary conjugate operator.*
- T [length](#) () const  
*Get lenght of quaternion.*
- T [lengthSq](#) () const  
*Return square of length.*
- void [normalize](#) ()  
*Normalize quaternion.*
- [Matrix3](#)< T > [rotMatrix](#) ()  
*Converts quaternion into rotation matrix.*
- [Matrix4](#)< T > [transform](#) () const

- Converts quaternion into transformation matrix.*
  - `Quaternion< T > lerp (T fact, const Quaternion< T > &rhs) const`  
*Linear interpolation of two quaternions.*
  - `std::string toString () const`  
*Gets string representation.*
  - `Quaternion< T > slerp (T r, const Quaternion< T > &q2) const`  
*Computes spherical interpolation between quaternions (this, q2) using coefficient of interpolation r (in [0, 1]).*

## Static Public Member Functions

- static `Quaternion< T > fromEulerAngles (T x, T y, T z)`  
*Creates quaternion for eulers angles.*
- static `Quaternion< T > fromAxisRot (Vector3< T > axis, float angleDeg)`  
*Creates quaternion as rotation around axis.*
- static `Quaternion< T > fromMatrix (const Matrix4< T > &m)`  
*Creates quaternion from transform matrix.*
- static `Quaternion< T > fromMatrix (const Matrix3< T > &m)`  
*Creates quaternion from rotation matrix.*

## Public Attributes

- `T w`  
*Real part of quaternion.*
- `Vector3< T > v`  
*Imaginary part of quaternion.*

## Friends

- `std::ostream & operator<< (std::ostream &oss, const Quaternion< T > &q)`  
*Provides output to standard output stream.*

### 5.4.1 Detailed Description

```
template<class T>
class Quaternion< T >
```

`Quaternion` class implementing some quaternion algebra operations.

`Quaternion` is kind of complex number it consists of its real part (w) and its complex part v. This complex part has three elements, so we can express it as  $xi + yj + zk$ . Note that coordinates of (x,y,z) are hold inside v field.

### 5.4.2 Constructor & Destructor Documentation

## 5.4.2.1 Quaternion() [1/5]

```
template<class T>
Quaternion< T >::Quaternion ( ) [inline]
```

Quaternion constructor, sets quaternion to  $(0 + 0i + 0j + 0k)$ .

## 5.4.2.2 Quaternion() [2/5]

```
template<class T>
Quaternion< T >::Quaternion (
    const Quaternion< T > & q ) [inline]
```

Copy constructor.

## 5.4.2.3 Quaternion() [3/5]

```
template<class T>
template<class FromT >
Quaternion< T >::Quaternion (
    const Quaternion< FromT > & q ) [inline]
```

Copy casting constructor.

## 5.4.2.4 Quaternion() [4/5]

```
template<class T>
Quaternion< T >::Quaternion (
    T w_,
    const Vector3< T > & v_ ) [inline]
```

Creates quaternion object from real part  $w_$  and complex part  $v_$ .

## Parameters

$w_$ ↔ _	Real part of quaternion.
$v_$ ↔ _	Complex part of quaternion $(xi + yj + zk)$ .

#### 5.4.2.5 Quaternion() [5/5]

```
template<class T>
Quaternion< T >::Quaternion (
    T w_,
    T x,
    T y,
    T z ) [inline]
```

Creates quaternion object from value ( $w_ + xi + yj + zk$ ).

##### Parameters

$w_ \leftrightarrow$ $_ \leftrightarrow$	Real part of quaternion.
$x$	Complex coefficient for i complex constant.
$y$	Complex coefficient for j complex constant.
$z$	Complex coefficient for k complex constant.

### 5.4.3 Member Function Documentation

#### 5.4.3.1 fromAxisRot()

```
template<class T>
static Quaternion<T> Quaternion< T >::fromAxisRot (
    Vector3< T > axis,
    float angleDeg ) [inline], [static]
```

Creates quaternion as rotation around axis.

##### Parameters

$axis$	Unit vector expressing axis of rotation.
$angleDeg$	Angle of rotation around axis (in degrees).

#### 5.4.3.2 fromEulerAngles()

```
template<class T>
static Quaternion<T> Quaternion< T >::fromEulerAngles (
    T x,
    T y,
    T z ) [inline], [static]
```

Creates quaternion for eulers angles.

## Parameters

<i>x</i>	Rotation around x axis (in degrees).
<i>y</i>	Rotation around y axis (in degrees).
<i>z</i>	Rotation around z axis (in degrees).

## Returns

[Quaternion](#) object representing transformation.

## 5.4.3.3 fromMatrix() [1/2]

```
template<class T>
static Quaternion<T> Quaternion< T >::fromMatrix (
    const Matrix4< T > & m ) [inline], [static]
```

Creates quaternion from transform matrix.

## Parameters

<i>m</i>	Transform matrix used to compute quaternion.
----------	--

## Returns

[Quaternion](#) representing rotation of matrix m.

## 5.4.3.4 fromMatrix() [2/2]

```
template<class T>
static Quaternion<T> Quaternion< T >::fromMatrix (
    const Matrix3< T > & m ) [inline], [static]
```

Creates quaternion from rotation matrix.

## Parameters

<i>m</i>	Rotation matrix used to compute quaternion.
----------	---

## Returns

[Quaternion](#) representing rotation of matrix m.

#### 5.4.3.5 length()

```
template<class T>
T Quaternion< T >::length ( ) const [inline]
```

Get lenght of quaternion.

##### Returns

Length of quaternion.

#### 5.4.3.6 lengthSq()

```
template<class T>
T Quaternion< T >::lengthSq ( ) const [inline]
```

Return square of length.

##### Returns

$\text{length}^2$

##### Note

This method is faster then [length\(\)](#). For comparison of length of two quaternion can be used just this value, instead of more expensive [length\(\)](#) method.

#### 5.4.3.7 lerp()

```
template<class T>
Quaternion<T> Quaternion< T >::lerp (
    T fact,
    const Quaternion< T > & rhs ) const [inline]
```

Linear interpolation of two quaternions.

##### Parameters

<i>fact</i>	Factor of interpolation. For translation from position of this vector to quaternion rhs, values of factor goes from 0.0 to 1.0.
<i>rhs</i>	Second <a href="#">Quaternion</a> for interpolation

##### Note

However values of fact parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

## 5.4.3.8 normalize()

```
template<class T>
void Quaternion< T >::normalize ( ) [inline]
```

Normalize quaternion.

## 5.4.3.9 operator!=(())

```
template<class T>
bool Quaternion< T >::operator!= (
    const Quaternion< T > & rhs ) const [inline]
```

Inequality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Returns

not (lhs == rhs) :-P

## 5.4.3.10 operator\*() [1/2]

```
template<class T>
Quaternion<T> Quaternion< T >::operator* (
    const Quaternion< T > & rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.4.3.11 operator\*() [2/2]

```
template<class T>
Quaternion<T> Quaternion< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.4.3.12 `operator*=( )` [1/2]

```
template<class T>
Quaternion<T>& Quaternion< T >::operator*= (
    const Quaternion< T > & rhs ) [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.4.3.13 `operator*=( )` [2/2]

```
template<class T>
Quaternion<T>& Quaternion< T >::operator*= (
    T rhs ) [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.4.3.14 `operator+( )`

```
template<class T>
Quaternion<T> Quaternion< T >::operator+ (
    const Quaternion< T > & rhs ) const [inline]
```

Addition operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.4.3.15 operator+=( )**

```
template<class T>
Quaternion<T>& Quaternion< T >::operator+= (
    const Quaternion< T > & rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.4.3.16 operator-( )** [1/2]

```
template<class T>
Quaternion<T> Quaternion< T >::operator- (
    const Quaternion< T > & rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.4.3.17 operator-( )** [2/2]

```
template<class T>
Quaternion<T> Quaternion< T >::operator- ( ) const [inline]
```

Unary negate operator.

**Returns**

negated quaternion

**5.4.3.18 operator-=( )**

```
template<class T>
Quaternion<T>& Quaternion< T >::operator-= (
    const Quaternion< T > & rhs ) [inline]
```

Subtraction operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.4.3.19 `operator=()` [1/2]

```
template<class T>
Quaternion<T>& Quaternion< T >::operator= (
    const Quaternion< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.4.3.20 `operator=()` [2/2]

```
template<class T>
template<class FromT >
Quaternion<T>& Quaternion< T >::operator= (
    const Quaternion< FromT > & rhs ) [inline]
```

Copy convert operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.4.3.21 `operator==()`

```
template<class T>
bool Quaternion< T >::operator== (
    const Quaternion< T > & rhs ) const [inline]
```

Equality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Note**

Test of equality is based of threshold EPSILON value. To be two values equal, must satisfy this condition  $|lhs - rhs| < EPSILON$ , for all quaternion coordinates.

**5.4.3.22 operator~()**

```
template<class T>
Quaternion<T> Quaternion< T >::operator~ ( ) const [inline]
```

Unary conjugate operator.

**Returns**

conjugated quaternion

**5.4.3.23 rotMatrix()**

```
template<class T>
Matrix3<T> Quaternion< T >::rotMatrix ( ) [inline]
```

Converts quaternion into rotation matrix.

**Returns**

Rotation matrix expressing this quaternion.

**5.4.3.24 slerp()**

```
template<class T>
Quaternion<T> Quaternion< T >::slerp (
    T r,
    const Quaternion< T > & q2 ) const [inline]
```

Computes spherical interpolation between quaternions (this, q2) using coefficient of interpolation r (in [0, 1]).

**Parameters**

<i>r</i>	The ratio of interpolation form this ( $r = 0$ ) to $q2$ ( $r = 1$ ).
<i>q2</i>	Second quaternion for interpolation.

### Returns

Result of interpolation.

#### 5.4.3.25 toString()

```
template<class T>
std::string Quaternion< T >::toString ( ) const [inline]
```

Gets string representation.

#### 5.4.3.26 transform()

```
template<class T>
Matrix4<T> Quaternion< T >::transform ( ) const [inline]
```

Converts quaternion into transformation matrix.

### Note

This method performs same operation as [rotMatrix\(\)](#) conversion method. But returns Matrix of 4x4 elements.

### Returns

Transformation matrix expressing this quaternion.

## 5.4.4 Friends And Related Function Documentation

#### 5.4.4.1 operator<<

```
template<class T>
std::ostream& operator<< (
    std::ostream & oss,
    const Quaternion< T > & q ) [friend]
```

Provides output to standard output stream.

#### 5.4.5 Member Data Documentation

## 5.4.5.1 v

```
template<class T>
Vector3<T> Quaternion< T >::v
```

Imaginary part of quaternion.

## 5.4.5.2 w

```
template<class T>
T Quaternion< T >::w
```

Real part of quaternion.

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

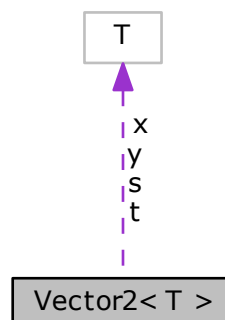
- [src/vmath.h](#)

## 5.5 Vector2< T > Class Template Reference

Class for two dimensional vector.

```
#include <vmath.h>
```

Collaboration diagram for Vector2< T >:



## Public Member Functions

- [Vector2](#) ()  
*Creates and sets to (0,0)*
- [Vector2](#) (T nx, T ny)  
*Creates and sets to (x,y)*
- [Vector2](#) (const [Vector2](#)< T > &src)  
*Copy constructor.*
- template<class FromT >  
[Vector2](#) (const [Vector2](#)< FromT > &src)  
*Copy casting constructor.*
- template<class FromT >  
[Vector2](#)< T > & [operator=](#) (const [Vector2](#)< FromT > &rhs)  
*Copy casting operator.*
- [Vector2](#)< T > & [operator=](#) (const [Vector2](#)< T > &rhs)  
*Copy operator.*
- T & [operator\[\]](#) (int n)  
*Array access operator.*
- const T & [operator\[\]](#) (int n) const  
*Constant array access operator.*
- [Vector2](#)< T > [operator+](#) (const [Vector2](#)< T > &rhs) const  
*Addition operator.*
- [Vector2](#)< T > [operator-](#) (const [Vector2](#)< T > &rhs) const  
*Subtraction operator.*
- [Vector2](#)< T > [operator\\*](#) (const [Vector2](#)< T > &rhs) const  
*Multiplication operator.*
- [Vector2](#)< T > [operator/](#) (const [Vector2](#)< T > &rhs) const  
*Division operator.*
- [Vector2](#)< T > & [operator+=](#) (const [Vector2](#)< T > &rhs)  
*Addition operator.*
- [Vector2](#)< T > & [operator-=](#) (const [Vector2](#)< T > &rhs)  
*Subtraction operator.*
- [Vector2](#)< T > & [operator\\*=](#) (const [Vector2](#)< T > &rhs)  
*Multiplication operator.*
- [Vector2](#)< T > & [operator/=](#) (const [Vector2](#)< T > &rhs)  
*Division operator.*
- [Vector2](#)< T > [operator+](#) (T rhs) const  
*Addition operator.*
- [Vector2](#)< T > [operator-](#) (T rhs) const  
*Subtraction operator.*
- [Vector2](#)< T > [operator\\*](#) (T rhs) const  
*Multiplication operator.*
- [Vector2](#)< T > [operator/](#) (T rhs) const  
*Division operator.*
- [Vector2](#)< T > & [operator+=](#) (T rhs)  
*Addition operator.*
- [Vector2](#)< T > & [operator-=](#) (T rhs)  
*Subtraction operator.*
- [Vector2](#)< T > & [operator\\*=](#) (T rhs)  
*Multiplication operator.*
- [Vector2](#)< T > & [operator/=](#) (T rhs)  
*Division operator.*

- Division operator.*
- bool `operator==` (const `Vector2< T >` &rhs) const
- Equality test operator.*
- bool `operator!=` (const `Vector2< T >` &rhs) const
- Inequality test operator.*
- `Vector2< T >` `operator-` () const
- Unary negate operator.*
- T `length` () const
- Get length of vector.*
- void `normalize` ()
- Normalize vector.*
- T `lengthSq` () const
- Return square of length.*
- `Vector2< T >` `lerp` (T fact, const `Vector2< T >` &r) const
- Linear interpolation of two vectors.*
- `operator T*` ()
- Conversion to pointer operator.*
- `operator const T*` () const
- Conversion to pointer operator.*
- std::string `toString` () const
- Gets string representation.*

### Public Attributes

- union {
  - T `x`  
*First element of vector, alias for X-coordinate.*
  - T `s`  
*First element of vector, alias for S-coordinate.*
- };
- union {
  - T `y`  
*Second element of vector, alias for Y-coordinate.*
  - T `t`  
*Second element of vector, alias for T-coordinate.*
- };

### Friends

- std::ostream & `operator<<` (std::ostream &lhs, const `Vector2< T >` &rhs)
- Output to stream operator.*

#### 5.5.1 Detailed Description

```
template<class T>
class Vector2< T >
```

Class for two dimensional vector.

There are three ways of accessing vector components. Let's have `Vector2f v`, you can either:

- access as position(x,y) — `v.x = v.y = 3;`
- access as texture coordinate (s,t) — `v.s = v.t = 3;`
- access via operator[] — `v[0] = v[1] = 3;`

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 `Vector2()` [1/4]

```
template<class T>
Vector2< T >::Vector2 ( ) [inline]
```

Creates and sets to (0,0)

### 5.5.2.2 `Vector2()` [2/4]

```
template<class T>
Vector2< T >::Vector2 (
    T nx,
    T ny ) [inline]
```

Creates and sets to (x,y)

#### Parameters

<i>nx</i>	initial x-coordinate value
<i>ny</i>	initial y-coordinate value

### 5.5.2.3 `Vector2()` [3/4]

```
template<class T>
Vector2< T >::Vector2 (
    const Vector2< T > & src ) [inline]
```

Copy constructor.

#### Parameters

<i>src</i>	Source of data for new created instance.
------------	--

### 5.5.2.4 `Vector2()` [4/4]

```
template<class T>
template<class FromT >
```

```
Vector2< T >::Vector2 (
    const Vector2< FromT > & src ) [inline]
```

Copy casting constructor.

#### Parameters

<i>src</i>	Source of data for new created instance.
------------	--

### 5.5.3 Member Function Documentation

#### 5.5.3.1 length()

```
template<class T>
T Vector2< T >::length ( ) const [inline]
```

Get length of vector.

#### Returns

length of vector

#### 5.5.3.2 lengthSq()

```
template<class T>
T Vector2< T >::lengthSq ( ) const [inline]
```

Return square of length.

#### Returns

$\text{length}^2$

#### Note

This method is faster then [length\(\)](#). For comparison of length of two vector can be used just this value, instead of more expensive [length\(\)](#) method.

#### 5.5.3.3 lerp()

```
template<class T>
Vector2<T> Vector2< T >::lerp (
    T fact,
    const Vector2< T > & r ) const [inline]
```

Linear interpolation of two vectors.

**Parameters**

<i>fact</i>	Factor of interpolation. For translation from position of this vector to vector <i>r</i> , values of factor goes from 0.0 to 1.0.
<i>r</i>	Second Vector for interpolation

**Note**

However values of *fact* parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

**5.5.3.4 normalize()**

```
template<class T>
void Vector2< T >::normalize ( ) [inline]
```

Normalize vector.

**5.5.3.5 operator const T\*()**

```
template<class T>
Vector2< T >::operator const T * ( ) const [inline]
```

Conversion to pointer operator.

**Returns**

Constant Pointer to internally stored (in management of class Vector2<T>) used for passing Vector2<T> values to gl\*2[fd] functions.

**5.5.3.6 operator T\*()**

```
template<class T>
Vector2< T >::operator T* ( ) [inline]
```

Conversion to pointer operator.

**Returns**

Pointer to internally stored (in management of class Vector2<T>) used for passing Vector2<T> values to gl\*2[fd] functions.

**5.5.3.7 operator!=(())**

```
template<class T>
bool Vector2< T >::operator!= (
    const Vector2< T > & rhs ) const [inline]
```

Inequality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Returns

not (lhs == rhs) :-P

## 5.5.3.8 operator\*() [1/2]

```
template<class T>
Vector2<T> Vector2< T >::operator* (
    const Vector2< T > & rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.5.3.9 operator\*() [2/2]

```
template<class T>
Vector2<T> Vector2< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.5.3.10 operator\*=( ) [1/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator*= (
    const Vector2< T > & rhs ) [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.11 `operator*=( )` [2/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator*= (
    T rhs ) [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.12 `operator+( )` [1/2]

```
template<class T>
Vector2<T> Vector2< T >::operator+ (
    const Vector2< T > & rhs ) const [inline]
```

Addition operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.13 `operator+( )` [2/2]

```
template<class T>
Vector2<T> Vector2< T >::operator+ (
    T rhs ) const [inline]
```

Addition operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.14 operator+=( )** [1/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator+= (
    const Vector2< T > & rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.15 operator+=( )** [2/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator+= (
    T rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.16 operator-( )** [1/3]

```
template<class T>
Vector2<T> Vector2< T >::operator- (
    const Vector2< T > & rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.17 operator-( )** [2/3]

```
template<class T>
Vector2<T> Vector2< T >::operator- (
    T rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.18 operator-()** [3/3]

```
template<class T>
Vector2<T> Vector2< T >::operator- ( ) const [inline]
```

Unary negate operator.

**Returns**

negated vector

**5.5.3.19 operator-=()** [1/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator-= (
    const Vector2< T > & rhs ) [inline]
```

Substraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.20 operator-=()** [2/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator-= (
    T rhs ) [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.21 operator/()** [1/2]

```
template<class T>
Vector2<T> Vector2< T >::operator/ (
    const Vector2< T > & rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.22 operator/()** [2/2]

```
template<class T>
Vector2<T> Vector2< T >::operator/ (
    T rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.23 operator/=( )** [1/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator/= (
    const Vector2< T > & rhs ) [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.5.3.24 operator/=( )** [2/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator/= (
    T rhs ) [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.25 `operator=()` [1/2]

```
template<class T>
template<class FromT >
Vector2<T>& Vector2< T >::operator= (
    const Vector2< FromT > & rhs ) [inline]
```

Copy casting operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.26 `operator=()` [2/2]

```
template<class T>
Vector2<T>& Vector2< T >::operator= (
    const Vector2< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

5.5.3.27 `operator==()`

```
template<class T>
bool Vector2< T >::operator== (
    const Vector2< T > & rhs ) const [inline]
```

Equality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Note**

Test of equality is based of threshold EPSILON value. To be two values equal, must satisfy this condition  $|lhs.x - rhs.y| < EPSILON$ , same for y-coordinate.

**5.5.3.28 operator[]()** [1/2]

```
template<class T>
T& Vector2< T >::operator[] (
    int n ) [inline]
```

Array access operator.

**Parameters**

<i>n</i>	Array index
----------	-------------

**Returns**

For  $n = 0$ , reference to x coordinate, else reference to y y coordinate.

**5.5.3.29 operator[]()** [2/2]

```
template<class T>
const T& Vector2< T >::operator[] (
    int n ) const [inline]
```

Constant array access operator.

**Parameters**

<i>n</i>	Array index
----------	-------------

**Returns**

For  $n = 0$ , reference to x coordinate, else reference to y y coordinate.

**5.5.3.30 toString()**

```
template<class T>
std::string Vector2< T >::toString ( ) const [inline]
```

Gets string representation.

## 5.5.4 Friends And Related Function Documentation

### 5.5.4.1 operator<<

```
template<class T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Vector2< T > & rhs ) [friend]
```

Output to stream operator.

#### Parameters

<i>lhs</i>	Left hand side argument of operator (commonly ostream instance).
<i>rhs</i>	Right hand side argument of operator.

#### Returns

Left hand side argument - the ostream object passed to operator.

## 5.5.5 Member Data Documentation

### 5.5.5.1 "@1

```
union { ... }
```

### 5.5.5.2 "@3

```
union { ... }
```

### 5.5.5.3 s

```
template<class T>
T Vector2< T >::s
```

First element of vector, alias for S-coordinate.

For textures notation.

## 5.5.5.4 t

```
template<class T>
T Vector2< T >::t
```

Second element of vector, alias for T-coordinate.

For textures notation.

## 5.5.5.5 x

```
template<class T>
T Vector2< T >::x
```

First element of vector, alias for X-coordinate.

## 5.5.5.6 y

```
template<class T>
T Vector2< T >::y
```

Second element of vector, alias for Y-coordinate.

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

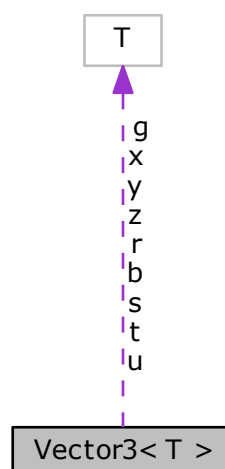
- [src/vmath.h](#)

## 5.6 Vector3&lt; T &gt; Class Template Reference

Class for three dimensional vector.

```
#include <vmath.h>
```

Collaboration diagram for Vector3< T >:



## Public Member Functions

- [Vector3](#) ()  
*Creates and sets to (0,0,0)*
- [Vector3](#) (T nx, T ny, T nz)  
*Creates and sets to (x,y,z)*
- [Vector3](#) (const [Vector3](#)< T > &src)  
*Copy constructor.*
- template<class FromT >  
[Vector3](#) (const [Vector3](#)< FromT > &src)  
*Copy casting constructor.*
- [Vector3](#)< T > [operator=](#) (const [Vector3](#)< T > &rhs)  
*Copy operator.*
- template<class FromT >  
[Vector3](#)< T > [operator=](#) (const [Vector3](#)< FromT > &rhs)  
*Copy casting operator.*
- T & [operator\[\]](#) (int n)  
*Array access operator.*
- const T & [operator\[\]](#) (int n) const  
*Constant array access operator.*
- [Vector3](#)< T > [operator+](#) (const [Vector3](#)< T > &rhs) const  
*Addition operator.*
- [Vector3](#)< T > [operator-](#) (const [Vector3](#)< T > &rhs) const  
*Subtraction operator.*
- [Vector3](#)< T > [operator\\*](#) (const [Vector3](#)< T > &rhs) const  
*Multiplication operator.*
- [Vector3](#)< T > [operator/](#) (const [Vector3](#)< T > &rhs) const  
*Division operator.*
- [Vector3](#)< T > & [operator+=](#) (const [Vector3](#)< T > &rhs)  
*Addition operator.*
- [Vector3](#)< T > & [operator-=](#) (const [Vector3](#)< T > &rhs)  
*Subtraction operator.*
- [Vector3](#)< T > & [operator\\*=](#) (const [Vector3](#)< T > &rhs)  
*Multiplication operator.*
- [Vector3](#)< T > & [operator/=](#) (const [Vector3](#)< T > &rhs)  
*Division operator.*
- T [dotProduct](#) (const [Vector3](#)< T > &rhs) const  
*Dot product of two vectors.*
- [Vector3](#)< T > [crossProduct](#) (const [Vector3](#)< T > &rhs) const  
*Cross product operator.*
- [Vector3](#)< T > [operator+](#) (T rhs) const  
*Addition operator.*
- [Vector3](#)< T > [operator-](#) (T rhs) const  
*Subtraction operator.*
- [Vector3](#)< T > [operator\\*](#) (T rhs) const  
*Multiplication operator.*
- [Vector3](#)< T > [operator/](#) (T rhs) const  
*Division operator.*
- [Vector3](#)< T > & [operator+=](#) (T rhs)  
*Addition operator.*
- [Vector3](#)< T > & [operator-=](#) (T rhs)

- Subtraction operator.*
- `Vector3< T > & operator*=( T rhs)`
- Multiplication operator.*
- `Vector3< T > & operator/=( T rhs)`
- Division operator.*
- `bool operator==( const Vector3< T > &rhs) const`
- Equality test operator.*
- `bool operator!=( const Vector3< T > &rhs) const`
- Inequality test operator.*
- `Vector3< T > operator- () const`
- Unary negate operator.*
- `T length () const`
- Get length of vector.*
- `T lengthSq () const`
- Return square of length.*
- `void normalize ()`
- Normalize vector.*
- `void rotate (T ax, T ay, T az)`
- Rotate vector around three axis.*
- `Vector3< T > lerp (T fact, const Vector3< T > &r) const`
- Linear interpolation of two vectors.*
- `operator T* ()`
- Conversion to pointer operator.*
- `operator const T * () const`
- Conversion to pointer operator.*
- `std::string toString () const`
- Gets string representation.*

## Public Attributes

- union {
  - `T x`  
*First element of vector, alias for X-coordinate.*
  - `T s`  
*First element of vector, alias for S-coordinate.*
  - `T r`  
*First element of vector, alias for R-coordinate.*
- };
- union {
  - `T y`  
*Second element of vector, alias for Y-coordinate.*
  - `T t`  
*Second element of vector, alias for T-coordinate.*
  - `T g`  
*Second element of vector, alias for G-coordinate.*
- };

- union {
    - T **z**  
Third element of vector, alias for Z-coordinate.
    - T **u**  
Third element of vector, alias for U-coordinate.
    - T **b**  
Third element of vector, alias for B-coordinate.
- };

## Friends

- std::ostream & **operator**<< (std::ostream &lhs, const **Vector3**< T > rhs)  
Output to stream operator.

### 5.6.1 Detailed Description

```
template<class T>
class Vector3< T >
```

Class for three dimensional vector.

There are four ways of accessing vector components. Let's have **Vector3f** v, you can either:

- access as position (x,y,z) — **v.x** = **v.y** = **v.z** = 1;
- access as texture coordinate (s,t,u) — **v.s** = **v.t** = **v.u** = 1;
- access as color (r,g,b) — **v.r** = **v.g** = **v.b** = 1;
- access via operator[] — **v[0]** = **v[1]** = **v[2]** = 1;

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 **Vector3()** [1/4]

```
template<class T>
Vector3< T >::Vector3 ( ) [inline]
```

Creates and sets to (0,0,0)

#### 5.6.2.2 **Vector3()** [2/4]

```
template<class T>
Vector3< T >::Vector3 (
    T nx,
    T ny,
    T nz ) [inline]
```

Creates and sets to (x,y,z)

## Parameters

<i>nx</i>	initial x-coordinate value
<i>ny</i>	initial y-coordinate value
<i>nz</i>	initial z-coordinate value

## 5.6.2.3 Vector3() [3/4]

```
template<class T>
Vector3< T >::Vector3 (
    const Vector3< T > & src ) [inline]
```

Copy constructor.

## Parameters

<i>src</i>	Source of data for new created <a href="#">Vector3</a> instance.
------------	--

## 5.6.2.4 Vector3() [4/4]

```
template<class T>
template<class FromT >
Vector3< T >::Vector3 (
    const Vector3< FromT > & src ) [inline]
```

Copy casting constructor.

## Parameters

<i>src</i>	Source of data for new created <a href="#">Vector3</a> instance.
------------	--

## 5.6.3 Member Function Documentation

## 5.6.3.1 crossProduct()

```
template<class T>
Vector3<T> Vector3< T >::crossProduct (
    const Vector3< T > & rhs ) const [inline]
```

Cross product operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.2 dotProduct()**

```
template<class T>
T Vector3< T >::dotProduct (
    const Vector3< T > & rhs ) const [inline]
```

Dot product of two vectors.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.3 length()**

```
template<class T>
T Vector3< T >::length ( ) const [inline]
```

Get length of vector.

**Returns**

length of vector

**5.6.3.4 lengthSq()**

```
template<class T>
T Vector3< T >::lengthSq ( ) const [inline]
```

Return square of length.

**Returns**

$\text{length}^2$

**Note**

This method is faster then [length\(\)](#). For comparison of length of two vector can be used just this value, instead of more expensive [length\(\)](#) method.

## 5.6.3.5 lerp()

```
template<class T>
Vector3<T> Vector3< T >::lerp (
    T fact,
    const Vector3< T > & r ) const [inline]
```

Linear interpolation of two vectors.

## Parameters

<i>fact</i>	Factor of interpolation. For translation from position of this vector to vector <i>r</i> , values of factor goes from 0.0 to 1.0.
<i>r</i>	Second Vector for interpolation

## Note

However values of *fact* parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

## 5.6.3.6 normalize()

```
template<class T>
void Vector3< T >::normalize ( ) [inline]
```

Normalize vector.

## 5.6.3.7 operator const T\*()

```
template<class T>
Vector3< T >::operator const T * ( ) const [inline]
```

Conversion to pointer operator.

## Returns

Constant Pointer to internally stored (in management of class Vector3<T>) used for passing Vector3<T> values to gl\*3[fd] functions.

**5.6.3.8 operator T\*()**

```
template<class T>
Vector3< T >::operator T* ( ) [inline]
```

Conversion to pointer operator.

**Returns**

Pointer to internally stored (in management of class Vector3<T>) used for passing Vector3<T> values to gl\*3[fd] functions.

**5.6.3.9 operator!=(())**

```
template<class T>
bool Vector3< T >::operator!= (
    const Vector3< T > & rhs ) const [inline]
```

Inequality test operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Returns**

not (lhs == rhs) :-P

**5.6.3.10 operator\*() [1/2]**

```
template<class T>
Vector3<T> Vector3< T >::operator* (
    const Vector3< T > & rhs ) const [inline]
```

Multiplication operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.11 operator\*() [2/2]**

```
template<class T>
```

```
Vector3<T> Vector3< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.6.3.12 operator\*=( ) [1/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator*= (
    const Vector3< T > & rhs ) [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.6.3.13 operator\*=( ) [2/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator*= (
    T rhs ) [inline]
```

Multiplication operator.

#### Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

#### 5.6.3.14 operator+( ) [1/2]

```
template<class T>
Vector3<T> Vector3< T >::operator+ (
    const Vector3< T > & rhs ) const [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.15 operator+()** [2/2]

```
template<class T>
Vector3<T> Vector3< T >::operator+ (
    T rhs ) const [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.16 operator+=()** [1/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator+= (
    const Vector3< T > & rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.17 operator+=()** [2/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator+=(
    T rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.18 operator-()** [1/3]

```
template<class T>
Vector3<T> Vector3< T >::operator- (
    const Vector3< T > & rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.19 operator-()** [2/3]

```
template<class T>
Vector3<T> Vector3< T >::operator- (
    T rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.20 operator-()** [3/3]

```
template<class T>
Vector3<T> Vector3< T >::operator- ( ) const [inline]
```

Unary negate operator.

**Returns**

negated vector

**5.6.3.21 operator-=( )** [1/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator-= (
    const Vector3< T > & rhs ) [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.22 operator-=()** [2/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator-= (
    T rhs ) [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.23 operator/()** [1/2]

```
template<class T>
Vector3<T> Vector3< T >::operator/ (
    const Vector3< T > & rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.6.3.24 operator/()** [2/2]

```
template<class T>
Vector3<T> Vector3< T >::operator/ (
    T rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.6.3.25 operator/=( ) [1/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator/= (
    const Vector3< T > & rhs ) [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.6.3.26 operator/=( ) [2/2]

```
template<class T>
Vector3<T>& Vector3< T >::operator/= (
    T rhs ) [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.6.3.27 operator=( ) [1/2]

```
template<class T>
Vector3<T> Vector3< T >::operator= (
    const Vector3< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.6.3.28 operator=( ) [2/2]

```
template<class T>
template<class FromT >
Vector3<T> Vector3< T >::operator= (
    const Vector3< FromT > & rhs ) [inline]
```

Copy casting operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.6.3.29 operator==( )

```
template<class T>
bool Vector3< T >::operator== (
    const Vector3< T > & rhs ) const [inline]
```

Equality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## Note

Test of equality is based of threshold EPSILON value. To be two values equal, must satisfy this condition  $|lhs.x - rhs.y| < EPSILON$ , same for y-coordinate, and z-coordinate.

## 5.6.3.30 operator[]() [1/2]

```
template<class T>
T& Vector3< T >::operator[] (
    int n ) [inline]
```

Array access operator.

## Parameters

<i>n</i>	Array index
----------	-------------

## Returns

For  $n = 0$ , reference to x coordinate,  $n = 1$  reference to y, else reference to z y coordinate.

## 5.6.3.31 operator[]() [2/2]

```
template<class T>
const T& Vector3< T >::operator[] (
    int n ) const [inline]
```

Constant array access operator.

**Parameters**

<i>n</i>	Array index
----------	-------------

**Returns**

For  $n = 0$ , reference to x coordinate,  $n = 1$  reference to y, else reference to z y coordinate.

**5.6.3.32 rotate()**

```
template<class T>
void Vector3< T >::rotate (
    T ax,
    T ay,
    T az ) [inline]
```

Rotate vector around three axis.

**Parameters**

<i>ax</i>	Angle (in degrees) to be rotated around X-axis.
<i>ay</i>	Angle (in degrees) to be rotated around Y-axis.
<i>az</i>	Angle (in degrees) to be rotated around Z-axis.

**5.6.3.33 toString()**

```
template<class T>
std::string Vector3< T >::toString ( ) const [inline]
```

Gets string representation.

**5.6.4 Friends And Related Function Documentation****5.6.4.1 operator<<**

```
template<class T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Vector3< T > rhs ) [friend]
```

Output to stream operator.

**Parameters**

<i>lhs</i>	Left hand side argument of operator (commonly ostream instance).
<i>rhs</i>	Right hand side argument of operator.

**Returns**

Left hand side argument - the ostream object passed to operator.

**5.6.5 Member Data Documentation****5.6.5.1 "@5**

```
union { ... }
```

**5.6.5.2 "@7**

```
union { ... }
```

**5.6.5.3 "@9**

```
union { ... }
```

**5.6.5.4 b**

```
template<class T>  
T Vector3< T >::b
```

Third element of vector, alias for B-coordinate.

For color notation.

**5.6.5.5 g**

```
template<class T>  
T Vector3< T >::g
```

Second element of vector, alias for G-coordinate.

For color notation.

#### 5.6.5.6 r

```
template<class T>  
T Vector3< T >::r
```

First element of vector, alias for R-coordinate.

For color notation.

#### 5.6.5.7 s

```
template<class T>  
T Vector3< T >::s
```

First element of vector, alias for S-coordinate.

For textures notation.

#### 5.6.5.8 t

```
template<class T>  
T Vector3< T >::t
```

Second element of vector, alias for T-coordinate.

For textures notation.

#### 5.6.5.9 u

```
template<class T>  
T Vector3< T >::u
```

Third element of vector, alias for U-coordinate.

For textures notation.

#### 5.6.5.10 x

```
template<class T>  
T Vector3< T >::x
```

First element of vector, alias for X-coordinate.

#### 5.6.5.11 y

```
template<class T>  
T Vector3< T >::y
```

Second element of vector, alias for Y-coordinate.

## 5.6.5.12 z

```
template<class T>
T Vector3< T >::z
```

Third element of vector, alias for Z-coordinate.

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

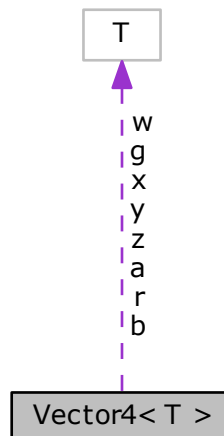
- src/vmath.h

## 5.7 Vector4< T > Class Template Reference

Class for four dimensional vector.

```
#include <vmath.h>
```

Collaboration diagram for Vector4< T >:



### Public Member Functions

- **Vector4** ()  
*Creates and sets to (0,0,0,0)*
- **Vector4** (T nx, T ny, T nz, T nw)  
*Creates and sets to (x,y,z,z)*
- **Vector4** (const **Vector4**< T > &src)  
*Copy constructor.*
- template<class FromT >  
**Vector4** (const **Vector4**< FromT > &src)  
*Copy casting constructor.*

- `Vector4` (const `Vector3`< T > &src, T w)
- template<typename FromT >  
`Vector4` (const `Vector3`< FromT > &src, FromT w)
- `Vector4`< T > `operator=` (const `Vector4`< T > &rhs)  
*Copy operator.*
- template<class FromT >  
`Vector4`< T > `operator=` (const `Vector4`< FromT > &rhs)  
*Copy casting operator.*
- T & `operator[]` (int n)  
*Array access operator.*
- const T & `operator[]` (int n) const  
*Array access operator.*
- `Vector4`< T > `operator+` (const `Vector4`< T > &rhs) const  
*Addition operator.*
- `Vector4`< T > `operator-` (const `Vector4`< T > &rhs) const  
*Subtraction operator.*
- `Vector4`< T > `operator*` (const `Vector4`< T > rhs) const  
*Multiplication operator.*
- `Vector4`< T > `operator/` (const `Vector4`< T > &rhs) const  
*Division operator.*
- `Vector4`< T > & `operator+=` (const `Vector4`< T > &rhs)  
*Addition operator.*
- `Vector4`< T > & `operator-=` (const `Vector4`< T > &rhs)  
*Subtraction operator.*
- `Vector4`< T > & `operator*=` (const `Vector4`< T > &rhs)  
*Multiplication operator.*
- `Vector4`< T > & `operator/=` (const `Vector4`< T > &rhs)  
*Division operator.*
- bool `operator==` (const `Vector4`< T > &rhs) const  
*Equality test operator.*
- bool `operator!=` (const `Vector4`< T > &rhs) const  
*Inequality test operator.*
- `Vector4`< T > `operator-` () const  
*Unary negate operator.*
- `Vector4`< T > `operator+` (T rhs) const  
*Addition operator.*
- `Vector4`< T > `operator-` (T rhs) const  
*Subtraction operator.*
- `Vector4`< T > `operator*` (T rhs) const  
*Multiplication operator.*
- `Vector4`< T > `operator/` (T rhs) const  
*Division operator.*
- `Vector4`< T > & `operator+=` (T rhs)  
*Addition operator.*
- `Vector4`< T > & `operator-=` (T rhs)  
*Subtraction operator.*
- `Vector4`< T > & `operator*=` (T rhs)  
*Multiplication operator.*
- `Vector4`< T > & `operator/=` (T rhs)  
*Division operator.*
- T `length` () const

- Get length of vector.*
  - void `normalize` ()
- Normalize vector.*
  - T `lengthSq` () const
- Return square of length.*
  - `Vector4`< T > `lerp` (T fact, const `Vector4`< T > &r) const
- Linear interpolation of two vectors.*
  - `operator T*` ()
- Conversion to pointer operator.*
  - `operator const T *` () const
- Conversion to pointer operator.*
  - `Vector3`< T > `xyz` () const
- Gets 3D vector.*
  - std::string `toString` () const
- Gets string representation.*

## Public Attributes

- union {
  - T `r`  
*First element of vector, alias for R-coordinate.*
  - T `x`
 };
- union {
  - T `g`  
*Second element of vector, alias for G-coordinate.*
  - T `y`  
*Second element of vector, alias for Y-coordinate.*
 };
- union {
  - T `b`  
*Third element of vector, alias for B-coordinate.*
  - T `z`  
*Third element of vector, alias for Z-coordinate.*
 };
- union {
  - T `a`  
*Fourth element of vector, alias for A-coordinate.*
  - T `w`  
*First element of vector, alias for W-coordinate.*
 };

## Friends

- std::ostream & `operator<<` (std::ostream &lhs, const `Vector4`< T > &rhs)
  - Output to stream operator.*

### 5.7.1 Detailed Description

```
template<class T>
class Vector4< T >
```

Class for four dimensional vector.

There are four ways of accessing vector components. Let's have `Vector4f v`, you can either:

- access as position in projective space (x,y,z,w) — `v.x = v.y = v.z = v.w = 1;`
- access as texture coordinate (s,t,u,v) — `v.s = v.t = v.u = v.v = 1;`
- access as color (r,g,b,a) — `v.r = v.g = v.b = v.a = 1;`
- access via operator[] — `v[0] = v[1] = v[2] = v[3] = 1;`

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 Vector4() [1/6]

```
template<class T>
Vector4< T >::Vector4 ( ) [inline]
```

Creates and sets to (0,0,0,0)

#### 5.7.2.2 Vector4() [2/6]

```
template<class T>
Vector4< T >::Vector4 (
    T nx,
    T ny,
    T nz,
    T nw ) [inline]
```

Creates and sets to (x,y,z,z)

#### Parameters

<i>nx</i>	initial x-coordinate value (R)
<i>ny</i>	initial y-coordinate value (G)
<i>nz</i>	initial z-coordinate value (B)
<i>nw</i>	initial w-coordinate value (Alpha)

**5.7.2.3 Vector4()** [3/6]

```
template<class T>
Vector4< T >::Vector4 (
    const Vector4< T > & src ) [inline]
```

Copy constructor.

**Parameters**

<i>src</i>	Source of data for new created <a href="#">Vector4</a> instance.
------------	--

**5.7.2.4 Vector4()** [4/6]

```
template<class T>
template<class FromT >
Vector4< T >::Vector4 (
    const Vector4< FromT > & src ) [inline]
```

Copy casting constructor.

**Parameters**

<i>src</i>	Source of data for new created <a href="#">Vector4</a> instance.
------------	--

**5.7.2.5 Vector4()** [5/6]

```
template<class T>
Vector4< T >::Vector4 (
    const Vector3< T > & src,
    T w ) [inline]
```

**5.7.2.6 Vector4()** [6/6]

```
template<class T>
template<typename FromT >
Vector4< T >::Vector4 (
    const Vector3< FromT > & src,
    FromT w ) [inline]
```

**5.7.3 Member Function Documentation**

### 5.7.3.1 length()

```
template<class T>
T Vector4< T >::length ( ) const [inline]
```

Get length of vector.

#### Returns

length of vector

### 5.7.3.2 lengthSq()

```
template<class T>
T Vector4< T >::lengthSq ( ) const [inline]
```

Return square of length.

#### Returns

$\text{length}^2$

#### Note

This method is faster then [length\(\)](#). For comparison of length of two vector can be used just this value, instead of more expensive [length\(\)](#) method.

### 5.7.3.3 lerp()

```
template<class T>
Vector4<T> Vector4< T >::lerp (
    T fact,
    const Vector4< T > & r ) const [inline]
```

Linear interpolation of two vectors.

#### Parameters

<i>fact</i>	Factor of interpolation. For translation from position of this vector to vector r, values of factor goes from 0.0 to 1.0.
<i>r</i>	Second Vector for interpolation

#### Note

However values of fact parameter are reasonable only in interval [0.0 , 1.0], you can pass also values outside of this interval and you can get result (extrapolation?)

#### 5.7.3.4 normalize()

```
template<class T>
void Vector4< T >::normalize ( ) [inline]
```

Normalize vector.

#### 5.7.3.5 operator const T \*()

```
template<class T>
Vector4< T >::operator const T * ( ) const [inline]
```

Conversion to pointer operator.

##### Returns

Constant Pointer to internally stored (in management of class Vector4<T>) used for passing Vector4<T> values to gl\*4[fd] functions.

#### 5.7.3.6 operator T\*()

```
template<class T>
Vector4< T >::operator T* ( ) [inline]
```

Conversion to pointer operator.

##### Returns

Pointer to internally stored (in management of class Vector4<T>) used for passing Vector4<T> values to gl\*4[fd] functions.

#### 5.7.3.7 operator!=(())

```
template<class T>
bool Vector4< T >::operator!= (
    const Vector4< T > & rhs ) const [inline]
```

Inequality test operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Returns**

not (lhs == rhs) :-P

**5.7.3.8 operator\*()** [1/2]

```
template<class T>
Vector4<T> Vector4< T >::operator* (
    const Vector4< T > rhs ) const [inline]
```

Multiplication operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.9 operator\*()** [2/2]

```
template<class T>
Vector4<T> Vector4< T >::operator* (
    T rhs ) const [inline]
```

Multiplication operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.10 operator\*=( )** [1/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator*= (
    const Vector4< T > & rhs ) [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.11 operator\*=( ) [ 2 / 2 ]

```
template<class T>
Vector4<T>& Vector4< T >::operator*= (
    T rhs ) [inline]
```

Multiplication operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.12 operator+( ) [ 1 / 2 ]

```
template<class T>
Vector4<T> Vector4< T >::operator+ (
    const Vector4< T > & rhs ) const [inline]
```

Addition operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.13 operator+( ) [ 2 / 2 ]

```
template<class T>
Vector4<T> Vector4< T >::operator+ (
    T rhs ) const [inline]
```

Addition operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.14** `operator+=()` [1/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator+= (
    const Vector4< T > & rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.15** `operator+=()` [2/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator+= (
    T rhs ) [inline]
```

Addition operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.16** `operator-()` [1/3]

```
template<class T>
Vector4<T> Vector4< T >::operator- (
    const Vector4< T > & rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.17** `operator-()` [2/3]

```
template<class T>
Vector4<T> Vector4< T >::operator- ( ) const [inline]
```

Unary negate operator.

**Returns**

negated vector

**5.7.3.18 operator-()** [3/3]

```
template<class T>
Vector4<T> Vector4< T >::operator- (
    T rhs ) const [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.19 operator-=()** [1/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator-= (
    const Vector4< T > & rhs ) [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.20 operator-=( )** [2/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator-=(
    T rhs ) [inline]
```

Subtraction operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.21** `operator/()` [1/2]

```
template<class T>
Vector4<T> Vector4< T >::operator/ (
    const Vector4< T > & rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.22** `operator/()` [2/2]

```
template<class T>
Vector4<T> Vector4< T >::operator/ (
    T rhs ) const [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.23** `operator/=( )` [1/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator/= (
    const Vector4< T > & rhs ) [inline]
```

Division operator.

**Parameters**

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**5.7.3.24** `operator/=( )` [2/2]

```
template<class T>
Vector4<T>& Vector4< T >::operator/= (
    T rhs ) [inline]
```

Division operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.25 operator=() [1/2]

```
template<class T>
Vector4<T> Vector4< T >::operator= (
    const Vector4< T > & rhs ) [inline]
```

Copy operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.26 operator=() [2/2]

```
template<class T>
template<class FromT >
Vector4<T> Vector4< T >::operator= (
    const Vector4< FromT > & rhs ) [inline]
```

Copy casting operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

## 5.7.3.27 operator==( )

```
template<class T>
bool Vector4< T >::operator== (
    const Vector4< T > & rhs ) const [inline]
```

Equality test operator.

## Parameters

<i>rhs</i>	Right hand side argument of binary operator.
------------	--

**Note**

Test of equality is based of threshold EPSILON value. To be two values equal, must satisfy this condition  $|lhs.x - rhs.y| < EPSILON$ , same for y-coordinate, z-coordinate, and w-coordinate.

**5.7.3.28 operator[]()** [1/2]

```
template<class T>
T& Vector4< T >::operator[] (
    int n ) [inline]
```

Array access operator.

**Parameters**

<i>n</i>	Array index
----------	-------------

**Returns**

For n = 0, reference to x coordinate, n = 1 reference to y coordinate, n = 2 reference to z, else reference to w coordinate.

**5.7.3.29 operator[]()** [2/2]

```
template<class T>
const T& Vector4< T >::operator[] (
    int n ) const [inline]
```

Array access operator.

**Parameters**

<i>n</i>	Array index
----------	-------------

**Returns**

For n = 0, reference to x coordinate, n = 1 reference to y coordinate, n = 2 reference to z, else reference to w coordinate.

**5.7.3.30 toString()**

```
template<class T>
std::string Vector4< T >::toString ( ) const [inline]
```

Gets string representation.

## 5.7.3.31 xyz()

```
template<class T>
Vector3<T> Vector4< T >::xyz ( ) const [inline]
```

Gets 3D vector.

Note that the output is divided by w coordinate to apply projection transform. If the w coordinate is equal to zero, the result is not divided.

## Returns

(x/w, y/w, z/w) iff w != 0 otherwise (x,y,z)

## 5.7.4 Friends And Related Function Documentation

## 5.7.4.1 operator&lt;&lt;

```
template<class T>
std::ostream& operator<< (
    std::ostream & lhs,
    const Vector4< T > & rhs ) [friend]
```

Output to stream operator.

## Parameters

<i>lhs</i>	Left hand side argument of operator (commonly ostream instance).
<i>rhs</i>	Right hand side argument of operator.

## Returns

Left hand side argument - the ostream object passed to operator.

## 5.7.5 Member Data Documentation

## 5.7.5.1 "@11

```
union { ... }
```

#### 5.7.5.2 "@13

```
union { ... }
```

#### 5.7.5.3 "@15

```
union { ... }
```

#### 5.7.5.4 "@17

```
union { ... }
```

#### 5.7.5.5 a

```
template<class T>  
T Vector4< T >::a
```

Fourth element of vector, alias for A-coordinate.

For color notation. This represnt aplha chanell

#### 5.7.5.6 b

```
template<class T>  
T Vector4< T >::b
```

Third element of vector, alias for B-coordinate.

For color notation.

#### 5.7.5.7 g

```
template<class T>  
T Vector4< T >::g
```

Second element of vector, alias for G-coordinate.

For color notation.

#### 5.7.5.8 r

```
template<class T>
T Vector4< T >::r
```

First element of vector, alias for R-coordinate.

For color notation. First element of vector, alias for X-coordinate.

#### 5.7.5.9 w

```
template<class T>
T Vector4< T >::w
```

First element of vector, alias for W-coordinate.

#### Note

For vectors (such as normals) should be set to 0.0 For vertices should be set to 1.0

#### 5.7.5.10 x

```
template<class T>
T Vector4< T >::x
```

#### 5.7.5.11 y

```
template<class T>
T Vector4< T >::y
```

Second element of vector, alias for Y-coordinate.

#### 5.7.5.12 z

```
template<class T>
T Vector4< T >::z
```

Third element of vector, alias for Z-coordinate.

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

- [src/vmath.h](#)



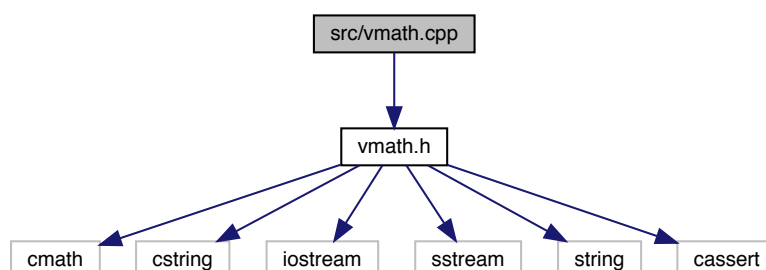
## Chapter 6

# File Documentation

### 6.1 src/vmath.cpp File Reference

```
#include "vmath.h"
```

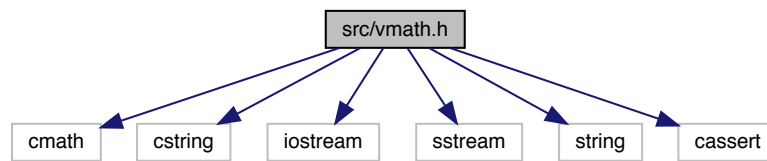
Include dependency graph for vmath.cpp:



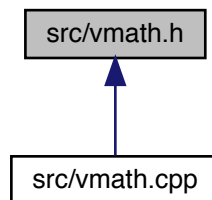
### 6.2 src/vmath.h File Reference

```
#include <cmath>
#include <cstring>
#include <iostream>
#include <sstream>
#include <string>
#include <cassert>
```

Include dependency graph for vmath.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Vector2< T >](#)  
*Class for two dimensional vector.*
- class [Vector3< T >](#)  
*Class for three dimensional vector.*
- class [Vector4< T >](#)  
*Class for four dimensional vector.*
- class [Matrix3< T >](#)  
*Class for matrix 3x3.*
- class [Matrix4< T >](#)  
*Class for matrix 4x4.*
- class [Quaternion< T >](#)  
*[Quaternion](#) class implementing some quaternion algebra operations.*
- class [Aabb3< T >](#)  
*Axes-aligned bounding-box (aka AABB) class.*

## Macros

- `#define M\_PI 3.14159265358979323846 /* pi */`
- `#define DEG2RAD(x) ((x * M\_PI) / 180.0)`
- `#define EPSILON epsilon`
- `#define VEC2 Vector2`
- `#define VEC3 Vector3`
- `#define VEC4 Vector4`

## Typedefs

- typedef class [Vector2](#)< float > [Vector2f](#)  
*Two dimensional Vector of floats.*
- typedef class [Vector2](#)< double > [Vector2d](#)  
*Two dimensional Vector of doubles.*
- typedef class [Vector2](#)< int > [Vector2i](#)  
*Two dimensional Vector of ints.*
- typedef [Vector3](#)< float > [Vector3f](#)  
*Three dimensional Vector of floats.*
- typedef [Vector3](#)< double > [Vector3d](#)  
*Three dimensional Vector of doubles.*
- typedef [Vector3](#)< int > [Vector3i](#)  
*Three dimensional Vector of ints.*
- typedef [Vector4](#)< float > [Vector4f](#)  
*Three dimensional Vector of floats.*
- typedef [Vector4](#)< double > [Vector4d](#)  
*Three dimensional Vector of doubles.*
- typedef [Vector4](#)< int > [Vector4i](#)  
*Three dimensional Vector of ints.*
- typedef [Matrix3](#)< float > [Matrix3f](#)  
*Matrix 3x3 of floats.*
- typedef [Matrix3](#)< double > [Matrix3d](#)  
*Matrix 3x3 of doubles.*
- typedef [Matrix3](#)< int > [Matrix3i](#)  
*Matrix 3x3 of int.*
- typedef [Matrix4](#)< float > [Matrix4f](#)  
*Matrix 4x4 of floats.*
- typedef [Matrix4](#)< double > [Matrix4d](#)  
*Matrix 4x4 of doubles.*
- typedef [Matrix4](#)< int > [Matrix4i](#)  
*Matrix 4x4 of int.*
- typedef [Quaternion](#)< float > [Quatf](#)
- typedef [Quaternion](#)< double > [Quatd](#)
- typedef [Aabb3](#)< float > [Aabb3f](#)
- typedef [Aabb3](#)< double > [Aabb3d](#)

## Functions

- template<typename T >  
[VEC2](#)< T > [std::min](#) (const [VEC2](#)< T > &a, const [VEC2](#)< T > &b)  
*Gets vector containing minimal values of a and b coordinates.*
- template<typename T >  
[VEC3](#)< T > [std::min](#) (const [VEC3](#)< T > &a, const [VEC3](#)< T > &b)  
*Gets vector containing minimal values of a and b coordinates.*
- template<typename T >  
[VEC4](#)< T > [std::min](#) (const [VEC4](#)< T > &a, const [VEC4](#)< T > &b)  
*Gets vector containing minimal values of a and b coordinates.*
- template<typename T >  
[VEC2](#)< T > [std::max](#) (const [VEC2](#)< T > &a, const [VEC2](#)< T > &b)  
*Gets vector containing maximal values of a and b coordinates.*

- `template<typename T>`  
`VEC3< T > std::max (const VEC3< T > &a, const VEC3< T > &b)`  
*Gets vector containing maximal values of a and b coordinates.*
- `template<typename T>`  
`VEC4< T > std::max (const VEC4< T > &a, const VEC4< T > &b)`  
*Gets vector containing maximal values of a and b coordinates.*

## Variables

- `const double epsilon = 4.37114e-05`

## 6.2.1 Macro Definition Documentation

### 6.2.1.1 DEG2RAD

```
#define DEG2RAD(  
    x ) ((x * M_PI) / 180.0)
```

### 6.2.1.2 EPSILON

```
#define EPSILON epsilon
```

### 6.2.1.3 M\_PI

```
#define M_PI 3.14159265358979323846 /* pi */
```

### 6.2.1.4 VEC2

```
#define VEC2 Vector2
```

### 6.2.1.5 VEC3

```
#define VEC3 Vector3
```

### 6.2.1.6 VEC4

```
#define VEC4 Vector4
```

## 6.2.2 Typedef Documentation

### 6.2.2.1 Aabb3d

```
typedef Aabb3<double> Aabb3d
```

### 6.2.2.2 Aabb3f

```
typedef Aabb3<float> Aabb3f
```

### 6.2.2.3 Matrix3d

```
typedef Matrix3<double> Matrix3d
```

Matrix 3x3 of doubles.

### 6.2.2.4 Matrix3f

```
typedef Matrix3<float> Matrix3f
```

Matrix 3x3 of floats.

### 6.2.2.5 Matrix3i

```
typedef Matrix3<int> Matrix3i
```

Matrix 3x3 of int.

#### 6.2.2.6 Matrix4d

```
typedef Matrix4<double> Matrix4d
```

Matrix 4x4 of doubles.

#### 6.2.2.7 Matrix4f

```
typedef Matrix4<float> Matrix4f
```

Matrix 4x4 of floats.

#### 6.2.2.8 Matrix4i

```
typedef Matrix4<int> Matrix4i
```

Matrix 4x4 of int.

#### 6.2.2.9 Quatd

```
typedef Quaternion<double> Quatd
```

#### 6.2.2.10 Quatf

```
typedef Quaternion<float> Quatf
```

#### 6.2.2.11 Vector2d

```
typedef class Vector2< double > Vector2d
```

Two dimensional Vector of doubles.

#### 6.2.2.12 Vector2f

```
typedef class Vector2< float > Vector2f
```

Two dimensional Vector of floats.

#### 6.2.2.13 Vector2i

```
typedef class Vector2< int > Vector2i
```

Two dimensional Vector of ints.

#### 6.2.2.14 Vector3d

```
typedef Vector3<double> Vector3d
```

Three dimensional Vector of doubles.

#### 6.2.2.15 Vector3f

```
typedef Vector3<float> Vector3f
```

Three dimensional Vector of floats.

#### 6.2.2.16 Vector3i

```
typedef Vector3<int> Vector3i
```

Three dimensional Vector of ints.

#### 6.2.2.17 Vector4d

```
typedef Vector4<double> Vector4d
```

Three dimensional Vector of doubles.

#### 6.2.2.18 Vector4f

```
typedef Vector4<float> Vector4f
```

Three dimensional Vector of floats.

#### 6.2.2.19 Vector4i

```
typedef Vector4<int> Vector4i
```

Three dimensional Vector of ints.

### 6.2.3 Function Documentation

#### 6.2.3.1 max() [1/3]

```
template<typename T >
VEC2<T> std::max (
    const VEC2< T > & a,
    const VEC2< T > & b )
```

Gets vector containing maximal values of *a* and *b* coordinates.

##### Returns

Vector of maximal coordinates.

#### 6.2.3.2 max() [2/3]

```
template<typename T >
VEC3<T> std::max (
    const VEC3< T > & a,
    const VEC3< T > & b )
```

Gets vector containing maximal values of *a* and *b* coordinates.

##### Returns

Vector of maximal coordinates.

### 6.2.3.3 max() [3/3]

```
template<typename T >
VEC4<T> std::max (
    const VEC4< T > & a,
    const VEC4< T > & b )
```

Gets vector containing maximal values of *a* and *b* coordinates.

#### Returns

Vector of maximal coordinates.

### 6.2.3.4 min() [1/3]

```
template<typename T >
VEC2<T> std::min (
    const VEC2< T > & a,
    const VEC2< T > & b )
```

Gets vector containing minimal values of *a* and *b* coordinates.

#### Returns

Vector of minimal coordinates.

### 6.2.3.5 min() [2/3]

```
template<typename T >
VEC3<T> std::min (
    const VEC3< T > & a,
    const VEC3< T > & b )
```

Gets vector containing minimal values of *a* and *b* coordinates.

#### Returns

Vector of minimal coordinates.

#### 6.2.3.6 min() [3/3]

```
template<typename T >
VEC4<T> std::min (
    const VEC4< T > & a,
    const VEC4< T > & b )
```

Gets vector containing minimal values of *a* and *b* coordinates.

##### Returns

Vector of minimal coordinates.

### 6.2.4 Variable Documentation

#### 6.2.4.1 epsilon

```
const double epsilon = 4.37114e-05
```

# Index

## a

Vector4, [114](#)

## Aabb3

Aabb3, [11](#), [12](#)  
center, [13](#)  
extend, [13](#)  
extended, [14](#)  
extent, [14](#)  
intersection, [15](#)  
intersects, [15](#), [16](#)  
invalidate, [16](#)  
max, [23](#)  
min, [23](#)  
operator &, [16](#)  
operator !=, [17](#)  
operator <<, [18](#), [22](#)  
operator \*, [17](#)  
operator \*=, [17](#)  
operator =, [19](#)  
operator ==, [19](#)  
operator |, [19](#)  
point, [21](#)  
size, [21](#)  
transformed, [21](#)  
valid, [22](#)

Aabb3 < T >, [9](#)

## Aabb3d

vmath.h, [121](#)

## Aabb3f

vmath.h, [121](#)

## at

Matrix3, [27](#)  
Matrix4, [40](#)

## b

Vector3, [97](#)  
Vector4, [114](#)

## center

Aabb3, [13](#)

## createFrustum

Matrix4, [41](#)

## createLookAt

Matrix4, [41](#)

## createOrtho

Matrix4, [42](#)

## createRotationAroundAxis

Matrix3, [27](#)

Matrix4, [42](#)

## createScale

Matrix4, [43](#)

## createTranslation

Matrix4, [43](#)

## crossProduct

Vector3, [85](#)

## DEG2RAD

vmath.h, [120](#)

## data

Matrix3, [36](#)

Matrix4, [54](#)

## det

Matrix3, [28](#)

Matrix4, [43](#)

## dotProduct

Vector3, [86](#)

## EPSILON

vmath.h, [120](#)

## epsilon

vmath.h, [126](#)

## extend

Aabb3, [13](#)

## extended

Aabb3, [14](#)

## extent

Aabb3, [14](#)

## fromAxisRot

Quaternion, [58](#)

## fromColumnMajorArray

Matrix3, [28](#)

Matrix4, [44](#)

## fromEulerAngles

Quaternion, [58](#)

## fromMatrix

Quaternion, [59](#)

## fromOde

Matrix3, [28](#)

## fromRowMajorArray

Matrix3, [28](#)

Matrix4, [44](#)

## g

Vector3, [97](#)

Vector4, [114](#)

## getScale

Matrix4, [45](#)

## getTranslation

Matrix4, [45](#)

- identity
  - Matrix3, 29
  - Matrix4, 45
- intersection
  - Aabb3, 15
- intersects
  - Aabb3, 15, 16
- invalidate
  - Aabb3, 16
- inverse
  - Matrix3, 29
  - Matrix4, 45
- length
  - Quaternion, 59
  - Vector2, 71
  - Vector3, 86
  - Vector4, 103
- lengthSq
  - Quaternion, 60
  - Vector2, 71
  - Vector3, 86
  - Vector4, 104
- lerp
  - Matrix3, 29
  - Matrix4, 45
  - Quaternion, 60
  - Vector2, 71
  - Vector3, 86
  - Vector4, 104
- M\_PI
  - vmath.h, 120
- Matrix3
  - at, 27
  - createRotationAroundAxis, 27
  - data, 36
  - det, 28
  - fromColumnMajorArray, 28
  - fromOde, 28
  - fromRowMajorArray, 28
  - identity, 29
  - inverse, 29
  - lerp, 29
  - Matrix3, 26
  - operator const T \*, 30
  - operator T \*, 30
  - operator!=, 30
  - operator<<, 35
  - operator\*, 31, 32
  - operator(), 31
  - operator+, 32
  - operator-, 33
  - operator/, 33
  - operator=, 34
  - operator==, 35
  - toString, 35
  - transpose, 35
- Matrix3< T >, 23
- Matrix3d
  - vmath.h, 121
- Matrix3f
  - vmath.h, 121
- Matrix3i
  - vmath.h, 121
- Matrix4
  - at, 40
  - createFrustum, 41
  - createLookAt, 41
  - createOrtho, 42
  - createRotationAroundAxis, 42
  - createScale, 43
  - createTranslation, 43
  - data, 54
  - det, 43
  - fromColumnMajorArray, 44
  - fromRowMajorArray, 44
  - getScale, 45
  - getTranslation, 45
  - identity, 45
  - inverse, 45
  - lerp, 45
  - Matrix4, 39, 40
  - operator const T \*, 46
  - operator T \*, 46
  - operator!=, 46
  - operator<<, 53
  - operator\*, 47, 48
  - operator(), 47
  - operator+, 48, 49
  - operator-, 49
  - operator/, 50
  - operator=, 50, 51
  - operator==, 51
  - setRotation, 51
  - setScale, 51, 52
  - setTranslation, 52
  - toString, 53
  - transpose, 53
- Matrix4< T >, 36
- Matrix4d
  - vmath.h, 121
- Matrix4f
  - vmath.h, 122
- Matrix4i
  - vmath.h, 122
- max
  - Aabb3, 23
  - vmath.h, 124
- min
  - Aabb3, 23
  - vmath.h, 125
- normalize
  - Quaternion, 61
  - Vector2, 72
  - Vector3, 87
  - Vector4, 105

- operator &
  - Aabb3, [16](#)
- operator const T \*
  - Matrix3, [30](#)
  - Matrix4, [46](#)
  - Vector2, [72](#)
  - Vector3, [87](#)
  - Vector4, [105](#)
- operator T\*
  - Matrix3, [30](#)
  - Matrix4, [46](#)
  - Vector2, [72](#)
  - Vector3, [87](#)
  - Vector4, [105](#)
- operator!=
  - Aabb3, [17](#)
  - Matrix3, [30](#)
  - Matrix4, [46](#)
  - Quaternion, [61](#)
  - Vector2, [72](#)
  - Vector3, [88](#)
  - Vector4, [105](#)
- operator<<
  - Aabb3, [18](#), [22](#)
  - Matrix3, [35](#)
  - Matrix4, [53](#)
  - Quaternion, [66](#)
  - Vector2, [80](#)
  - Vector3, [96](#)
  - Vector4, [113](#)
- operator\*
  - Aabb3, [17](#)
  - Matrix3, [31](#), [32](#)
  - Matrix4, [47](#), [48](#)
  - Quaternion, [61](#)
  - Vector2, [73](#)
  - Vector3, [88](#)
  - Vector4, [106](#)
- operator\*=
  - Aabb3, [17](#)
  - Quaternion, [62](#)
  - Vector2, [73](#), [74](#)
  - Vector3, [89](#)
  - Vector4, [106](#), [107](#)
- operator~
  - Quaternion, [65](#)
- operator()
  - Matrix3, [31](#)
  - Matrix4, [47](#)
- operator+
  - Matrix3, [32](#)
  - Matrix4, [48](#), [49](#)
  - Quaternion, [62](#)
  - Vector2, [74](#)
  - Vector3, [89](#), [90](#)
  - Vector4, [107](#)
- operator+=
  - Quaternion, [63](#)
- Vector2, [74](#), [75](#)
- Vector3, [90](#)
- Vector4, [107](#), [108](#)
- operator-
  - Matrix3, [33](#)
  - Matrix4, [49](#)
  - Quaternion, [63](#)
  - Vector2, [75](#), [76](#)
  - Vector3, [90](#), [91](#)
  - Vector4, [108](#), [109](#)
- operator-=
  - Quaternion, [63](#)
  - Vector2, [76](#)
  - Vector3, [91](#), [92](#)
  - Vector4, [109](#)
- operator/
  - Matrix3, [33](#)
  - Matrix4, [50](#)
  - Vector2, [76](#), [77](#)
  - Vector3, [92](#)
  - Vector4, [109](#), [110](#)
- operator/=
  - Vector2, [77](#)
  - Vector3, [92](#), [93](#)
  - Vector4, [110](#)
- operator=
  - Aabb3, [19](#)
  - Matrix3, [34](#)
  - Matrix4, [50](#), [51](#)
  - Quaternion, [64](#)
  - Vector2, [78](#)
  - Vector3, [93](#)
  - Vector4, [111](#)
- operator==
  - Aabb3, [19](#)
  - Matrix3, [35](#)
  - Matrix4, [51](#)
  - Quaternion, [64](#)
  - Vector2, [78](#)
  - Vector3, [95](#)
  - Vector4, [111](#)
- operator[]
  - Vector2, [79](#)
  - Vector3, [95](#)
  - Vector4, [112](#)
- operator |
  - Aabb3, [19](#)
- point
  - Aabb3, [21](#)
- Quatd
  - vmath.h, [122](#)
- Quaternion
  - fromAxisRot, [58](#)
  - fromEulerAngles, [58](#)
  - fromMatrix, [59](#)
  - length, [59](#)
  - lengthSq, [60](#)

- lerp, [60](#)
- normalize, [61](#)
- operator!=, [61](#)
- operator<<, [66](#)
- operator\*, [61](#)
- operator\*=[62](#)
- operator~, [65](#)
- operator+, [62](#)
- operator+=, [63](#)
- operator-, [63](#)
- operator-=, [63](#)
- operator=, [64](#)
- operator==, [64](#)
- Quaternion, [56](#), [57](#)
- rotMatrix, [65](#)
- slerp, [65](#)
- toString, [66](#)
- transform, [66](#)
- v, [66](#)
- w, [67](#)
- Quaternion< T >, [54](#)
- Quatf
  - vmath.h, [122](#)
- r
  - Vector3, [97](#)
  - Vector4, [114](#)
- rotMatrix
  - Quaternion, [65](#)
- rotate
  - Vector3, [96](#)
- s
  - Vector2, [80](#)
  - Vector3, [98](#)
- setRotation
  - Matrix4, [51](#)
- setScale
  - Matrix4, [51](#), [52](#)
- setTranslation
  - Matrix4, [52](#)
- size
  - Aabb3, [21](#)
- slerp
  - Quaternion, [65](#)
- src/vmath.cpp, [117](#)
- src/vmath.h, [117](#)
- t
  - Vector2, [80](#)
  - Vector3, [98](#)
- toString
  - Matrix3, [35](#)
  - Matrix4, [53](#)
  - Quaternion, [66](#)
  - Vector2, [79](#)
  - Vector3, [96](#)
  - Vector4, [112](#)
- transform
  - Quaternion, [66](#)
  - transformed
    - Aabb3, [21](#)
  - transpose
    - Matrix3, [35](#)
    - Matrix4, [53](#)
- u
  - Vector3, [98](#)
- v
  - Quaternion, [66](#)
- VEC2
  - vmath.h, [120](#)
- VEC3
  - vmath.h, [120](#)
- VEC4
  - vmath.h, [120](#)
- valid
  - Aabb3, [22](#)
- Vector2
  - length, [71](#)
  - lengthSq, [71](#)
  - lerp, [71](#)
  - normalize, [72](#)
  - operator const T \*, [72](#)
  - operator T\*, [72](#)
  - operator!=, [72](#)
  - operator<<, [80](#)
  - operator\*, [73](#)
  - operator\*=[73](#), [74](#)
  - operator+, [74](#)
  - operator+=, [74](#), [75](#)
  - operator-, [75](#), [76](#)
  - operator-=, [76](#)
  - operator/, [76](#), [77](#)
  - operator/=[77](#)
  - operator=, [78](#)
  - operator==, [78](#)
  - operator[], [79](#)
  - s, [80](#)
  - t, [80](#)
  - toString, [79](#)
  - Vector2, [70](#)
  - x, [81](#)
  - y, [81](#)
- Vector2< T >, [67](#)
- Vector2d
  - vmath.h, [122](#)
- Vector2f
  - vmath.h, [122](#)
- Vector2i
  - vmath.h, [123](#)
- Vector3
  - b, [97](#)
  - crossProduct, [85](#)
  - dotProduct, [86](#)
  - g, [97](#)
  - length, [86](#)

- lengthSq, 86
- lerp, 86
- normalize, 87
- operator const T \*, 87
- operator T\*, 87
- operator!=, 88
- operator<<, 96
- operator\*, 88
- operator\*=: 89
- operator+, 89, 90
- operator+=, 90
- operator-, 90, 91
- operator-=, 91, 92
- operator/, 92
- operator/=, 92, 93
- operator=, 93
- operator==, 95
- operator[], 95
- r, 97
- rotate, 96
- s, 98
- t, 98
- toString, 96
- u, 98
- Vector3, 84, 85
- x, 98
- y, 98
- z, 98
- Vector3< T >, 81
- Vector3d
  - vmath.h, 123
- Vector3f
  - vmath.h, 123
- Vector3i
  - vmath.h, 123
- Vector4
  - a, 114
  - b, 114
  - g, 114
  - length, 103
  - lengthSq, 104
  - lerp, 104
  - normalize, 105
  - operator const T \*, 105
  - operator T\*, 105
  - operator!=, 105
  - operator<<, 113
  - operator\*, 106
  - operator\*=: 106, 107
  - operator+, 107
  - operator+=, 107, 108
  - operator-, 108, 109
  - operator-=, 109
  - operator/, 109, 110
  - operator/=, 110
  - operator=, 111
  - operator==, 111
  - operator[], 112
  - r, 114
  - toString, 112
  - Vector4, 102, 103
  - w, 115
  - x, 115
  - xyz, 112
  - y, 115
  - z, 115
- Vector4< T >, 99
- Vector4d
  - vmath.h, 123
- Vector4f
  - vmath.h, 123
- Vector4i
  - vmath.h, 124
- vmath.h
  - Aabb3d, 121
  - Aabb3f, 121
  - DEG2RAD, 120
  - EPSILON, 120
  - epsilon, 126
  - M\_PI, 120
  - Matrix3d, 121
  - Matrix3f, 121
  - Matrix3i, 121
  - Matrix4d, 121
  - Matrix4f, 122
  - Matrix4i, 122
  - max, 124
  - min, 125
  - Quatd, 122
  - Quatf, 122
  - VEC2, 120
  - VEC3, 120
  - VEC4, 120
  - Vector2d, 122
  - Vector2f, 122
  - Vector2i, 123
  - Vector3d, 123
  - Vector3f, 123
  - Vector3i, 123
  - Vector4d, 123
  - Vector4f, 123
  - Vector4i, 124
- w
  - Quaternion, 67
  - Vector4, 115
- x
  - Vector2, 81
  - Vector3, 98
  - Vector4, 115
- xyz
  - Vector4, 112
- y
  - Vector2, 81
  - Vector3, 98

Vector4, [115](#)

z

Vector3, [98](#)

Vector4, [115](#)