MathsLibrary reference document

By Leonard Andrew Spencer, 2017

# Classes

**Vector 3**

**Matrix 10**

# Vector

template<size\_t DIM>

Vectors are spatial vectors, represented as an array of the vector’s components.

Vector2, Vector3, and Vector4 are used as aliases for Vector<2>, Vector<3> and Vector<4>

## Template Parameters

DIM Dimensions of the vector.

## Member Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Type** | **Access** |  |
| m\_component | float[DIM] | protected | Components of the vector. |

## Member Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Page** |  |
| (constructor) | 4 | Initializes vector with components passed as arguments |
| (destructor) |  | Destroys the vector |
| operator[] |  | Access specified component |
| operator float\* |  | Cast vector as array of floats, allowing access to its components |
| operator Vector<D> |  | Cast vector as vector of different dimensions by creating a new vector |
| dot |  | Returns product of dot multiplication |
| cross |  | Returns product of cross multiplication for Vector<3> and Vector<4> |
| magnitudeSquared |  | Returns square of vector’s magnitude |
| magnitude |  | Returns vector’s magnitude |
| normalise |  | Scales vector to a unit vector |
| compareMagnitude |  | Compares the vector’s magnitude to a float |

## Non-member Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Page** |  |
| operator+ |  | Add vectors |
| operator- |  | Subtract vector |
| operator\* |  | Scale vector by float |
| operator> |  | Vector magnitude greater than |
| operator< |  | Vector magnitude less than |
| operator>= |  | Vector magnitude greater than or equal to |
| operator<= |  | Vector magnitude less than or equal to |
| operator== |  | Vector components are equal |
| operator!= |  | Vector components are not equal |

## Vector::Vector

Constructs a Vector object, initializing its components depending on constructor used

### Overloaded Functions

1. Vector(); *default constructor*  
   Constructs a zero vector, with all components equal to zero
2. template<typename... Args>  
   Vector(typename std::enable\_if<(sizeof...(Args)+1 == DIM), float>::type x, Args... args);  
   Initializes components with arguments passed, starting with x. Enable\_if is used so that each Vector class compiles with a constructor taking as many arguments as it has components

### Template Parameters

… Args Parameter pack allowing enable\_if to check total number of arguments and

### Function Parameters

x Float to be used as the first component of the vector.

args Function parameter pack containing additional vector components

## Vector::~Vector

~Vector();

Destroys the vector.

## Vector::operator[]

Returns a reference to the component at position n

### Overloaded Functions

1. float& operator[](size\_t n);
2. const float& operator[](size\_t n) const;  
   Returns constant reference

### Function Parameters

n Position of component in array. The first component is at position 0

### Return Value

Reference to float. If constant version is used, constant reference to float

## Vector::operator float\*

explicit operator float\*();

Casts vector as a pointer to its first component, allowing it to be treated as an array

### Return Value

float\* pointing to first component of the vector.

## Vector::Vector<D>

template<size\_t D>

explicit operator Vector<D>();

Casts vector as a vector of different dimensions. The components of this vector are copied to a newly constructed vector. If the new vector has more dimensions, the components for those dimensions are left as 0.

### Template Parameters

D Dimensions of vector class being cast to

### Return Value

Vector<D> object with components copied from this vector for all shared dimensions

## Vector::dot

float dot(const Vector<DIM>& b) const;

Calculates the dot product of this vector and vector b of same dimensions. The dot product is calculated by multiplying the corresponding components of each vector, and adding the products together.

### Function Parameters

b Reference to the other vector of the dot product

### Return Value

float equal to dot product of this vector and b;

## Vector::cross

template <std::size\_t D = DIM>

typename std::enable\_if<D == 3||D==4, Vector<D>>::type cross(const Vector<D>& b) const;

Calculates the cross product of this vector and vector b of the same dimensions. As cross products are only well defined for 3 dimensional vectors, this function only exists for Vector<3> and Vector<4> (with the Vector<4> being treated as a 3D vector in homogeneous coordinates with a w element of 0)

### Template Parameters

D Dimensions of this vector. Used by enable\_if to prevent this function from being compiled for Vector classes of invalid dimensions

### Function Parameters

b Reference to the other vector of the cross product

### Return Value

Vector<D> equal to cross product of this vector and b. If this is a Vector<4>, its final component will have a value of 0.

## Vector::magnitudeSquared

float magnitudeSquared() const

Calculates the square of the vector’s magnitude. This is equal to the sum of the square of each component. This value can be used to compare the vector’s magnitude to other values without having to calculate a square root.

### Return Value

float equal to the square of this vector’s magnitude

## Vector::magnitude

float magnitude() const;

Calculates the magnitude of this vector. This is done by taking the square root of magnitudeSquared.

### Return Value

Float equal to this vector’s magnitude

## Vector::normalise

void normalise();

Converts this vector to its normalised form, by scaling it to a magnitude of 1. The inverse of the vector’s magnitude is calculated, and all components are multiplied by this value.

## Vector::compareMagnitude

int compareMagnitude(float f) const;

Compares the vector’s magnitude with the float value passed. This is done by comparing magnitudeSquared to the square of f.

### Function Parameters

f value being compared to the vector’s magnitude

### Return Value

Returns 0 if the vector’s magnitude is equal to the float. Otherwise, returns 1 if the magnitude is greater, and -1 if the float is greater.

## operator+ (Vector)

template<size\_t DIM>

Vector<DIM> operator+(const Vector<DIM>& a, const Vector<DIM>& b);

Adds two vectors with the same dimensions together. Each component of the returned vector is equal to the sum of the corresponding components of the vectors

### Template Parameters

DIM dimensions of the vectors

### Function Parameters

a, b Vectors to be added

### Return Value

Vector equal to the sum of a and b

## operator- (Vector)

template<size\_t DIM>

Vector<DIM> operator-(const Vector<DIM>& a, const Vector<DIM>& b);

Subtracts one vector from another. Each component of the returned vector is equal to the difference between the corresponding components of the vectors.

### Template Parameters

DIM dimensions of the vectors

### Function Parameters

a Vector being subtracted from

b Vector to be subtracted from a

### Return Value

Vector equal to the difference between a and b

## operator\* (Vector)

1. template<size\_t DIM>  
   Vector<DIM> operator\*(const Vector<DIM>& v, float f)
2. template<size\_t DIM>  
   Vector<DIM> operator\*(float f, const Vector<DIM>& v)

Multiplies a vector by a float. Each component of the returned vector is equal to the corresponding component in the vector multiplied by the float.

### Template Parameters

DIM Dimensions of the vector

### Function Parameters

v Vector to be multiplied

f Value to multiply the vector by

### Return Value

Vector equal to v scaled by f

## operator>,<,>=,<= (Vector)

1. template<size\_t DIM>  
   bool operator>(const Vector<DIM>& a, const Vector<DIM>& b);
2. template<size\_t DIM>  
   bool operator<(const Vector<DIM>& a, const Vector<DIM>& b);
3. template<size\_t DIM>  
   bool operator>=(const Vector<DIM>& a, const Vector<DIM>& b);
4. template<size\_t DIM>  
   bool operator<=(const Vector<DIM>& a, const Vector<DIM>& b);

Compares the magnitude of two vectors, using the squares of their magnitudes. Note that >= and <= refer to the magnitudes being equal, while the == operator requires all components be equal. “a<b||a==b” is not the same as “a<=b”, since if the vectors have the same magnitude and different directions, the first is false but the second is true.

### Template Parameters

DIM dimensions of the vectors

### Function Parameters

a, b Vectors to be compared

### Return Value

1. True if the magnitude of a is greater than that of b, false otherwise
2. True if the magnitude of a is less than that of b, false otherwise
3. True if the magnitude of a is greater than or equal to that of b, false otherwise
4. True if the magnitude of a is less than or equal to that of b, false otherwise

## operator==,!= (Vector)

1. template<size\_t DIM>  
   bool operator==(const Vector<DIM>& a, const Vector<DIM>& b)
2. template<size\_t DIM>  
   bool operator!=(const Vector<DIM>& a, const Vector<DIM>& b)

Compares the components of two vectors. Vectors are equal if all corresponding components are equal.

### Template Parameters

DIM dimensions of the vectors

### Function Parameters

a, b Vectors to be compared

### Return Value

1. Returns true if the corresponding components of a and b are equal, false otherwise
2. Returns true if any component in a is not equal to a the corresponding component in b, false otherwise

# Matrix

template<size\_t ORDER>

(Matrix Description)

## Template Parameters

ORDER Order of the matrix

## Member Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Type** | **Access** |  |
|  |  |  |  |
|  |  |  |  |

## Member Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Page** |  |
| (constructor) |  | Initializes matrix with elements passed as arguments |
| (destructor) |  | Destroys the matrix |
| operator[] |  | Access specified column |
| operator\* |  | Multiplies vector or matrix by this matrix |
| setIdentity |  | Sets matrix as identity matrix |
| setRotate |  | Sets Matrix<2> as transformation matrix rotating by given angle |
| setRotateX |  | Sets Matrix<3> or Matrix<4> as transformation matrix rotating around X axis |
| setRotateY |  | Sets Matrix<3> or Matrix<4> as transformation matrix rotating around Y axis |
| setRotateZ |  | Sets Matrix<3> or Matrix<4> as transformation matrix rotating around Z axis |
| setEulerRotate |  | Sets Matrix<3> or Matrix<4> as transformation matrix rotating to an orientation defined by Euler angles |
| setTaitBryanRotate |  | Sets Matrix<3> or Matrix<4> as transformation matrix rotating to an orientation defined by Euler angles given in Tait-Bryan notation |
| calculateInverse |  | Calculates inverse matrix for this matrix |

## Protected Member Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Page** |  |
| invertTransform |  |  |
| areColumnsEqual |  | Checks if two matrix columns or vectors, passed as float arrays, are equal |
| swapRows |  | Swaps rows of matrix passed as float array |
| multiplyRow |  | Multiplies all elements in matrix row by a float |
| addRow |  | Adds matrix row to another matrix row |