

Triangles

1.0.1

Generated by Doxygen 1.8.17

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 geom Namespace Reference	7
4.1.1 Detailed Description	9
4.1.2 Typedef Documentation	9
4.1.2.1 VectorD	9
4.1.2.2 VectorF	9
4.1.3 Function Documentation	9
4.1.3.1 distance()	9
4.1.3.2 isIntersect2D()	10
4.1.3.3 intersect()	11
4.1.3.4 isIntersect()	11
4.1.3.5 operator<<() [1/4]	12
4.1.3.6 operator==() [1/3]	12
4.1.3.7 operator==() [2/3]	13
4.1.3.8 operator<<() [2/4]	14
4.1.3.9 operator<<() [3/4]	15
4.1.3.10 operator+()	15
4.1.3.11 operator-()	16
4.1.3.12 operator*() [1/2]	17
4.1.3.13 operator*() [2/2]	17
4.1.3.14 operator/()	18
4.1.3.15 dot()	18
4.1.3.16 cross()	19
4.1.3.17 operator==() [3/3]	19
4.1.3.18 operator"!=()	20
4.1.3.19 operator<<() [4/4]	21
4.1.4 Variable Documentation	21
4.1.4.1 Number	21
4.2 geom::detail Namespace Reference	22
4.2.1 Function Documentation	22
4.2.1.1 isIntersect2D()	22
4.2.1.2 isIntersectMollerHaines()	23
4.2.1.3 isOverlap()	23
4.2.1.4 isSameSign()	23

4.2.1.5 isOnOneSide()	23
5 Class Documentation	25
5.1 geom::Line< T > Class Template Reference	25
5.1.1 Detailed Description	25
5.1.2 Constructor & Destructor Documentation	26
5.1.2.1 Line()	26
5.1.3 Member Function Documentation	26
5.1.3.1 org()	26
5.1.3.2 dir()	27
5.1.3.3 belongs()	27
5.1.3.4 isEqual()	27
5.1.3.5 getBy2Points()	28
5.2 geom::Plane< T > Class Template Reference	28
5.2.1 Detailed Description	29
5.2.2 Member Function Documentation	29
5.2.2.1 dist()	29
5.2.2.2 norm()	30
5.2.2.3 belongs() [1/2]	30
5.2.2.4 belongs() [2/2]	30
5.2.2.5 isEqual()	31
5.2.2.6 isPar()	31
5.2.2.7 getBy3Points()	32
5.2.2.8 getParametric()	32
5.2.2.9 getNormalPoint()	33
5.2.2.10 getNormalDist()	33
5.3 geom::Triangle< T > Class Template Reference	34
5.3.1 Detailed Description	34
5.3.2 Constructor & Destructor Documentation	34
5.3.2.1 Triangle()	34
5.3.3 Member Function Documentation	35
5.3.3.1 operator[]()	35
5.4 geom::Vector< T > Class Template Reference	35
5.4.1 Detailed Description	37
5.4.2 Constructor & Destructor Documentation	37
5.4.2.1 Vector() [1/2]	37
5.4.2.2 Vector() [2/2]	37
5.4.3 Member Function Documentation	38
5.4.3.1 operator+=()	38
5.4.3.2 operator-=()	38
5.4.3.3 operator-()	39
5.4.3.4 operator*=() [1/2]	39

5.4.3.5 operator/=() [1/2]	40
5.4.3.6 dot()	40
5.4.3.7 cross()	41
5.4.3.8 length2()	41
5.4.3.9 length()	41
5.4.3.10 normalized()	42
5.4.3.11 normalize()	42
5.4.3.12 operator[]() [1/2]	42
5.4.3.13 operator[]() [2/2]	43
5.4.3.14 isPar()	43
5.4.3.15 isPerp()	44
5.4.3.16 isEqual()	44
5.4.3.17 isNumEq()	45
5.4.3.18 setThreshold()	45
5.4.3.19 getThreshold()	46
5.4.3.20 setDefThreshold()	46
5.4.3.21 operator*=() [2/2]	46
5.4.3.22 operator/=() [2/2]	46
5.4.4 Member Data Documentation	47
5.4.4.1 x	47
5.4.4.2 y	47
5.4.4.3 z	47
6 File Documentation	49
6.1 include/distance/distance.hh File Reference	49
6.2 distance.hh	50
6.3 include/intersection/intersection.hh File Reference	51
6.4 intersection.hh	52
6.5 include/primitives/line.hh File Reference	55
6.6 line.hh	57
6.7 include/primitives/plane.hh File Reference	58
6.8 plane.hh	60
6.9 include/primitives/primitives.hh File Reference	63
6.10 primitives.hh	64
6.11 include/primitives/triangle.hh File Reference	64
6.12 triangle.hh	65
6.13 include/primitives/vector.hh File Reference	66
6.13.1 Detailed Description	69
6.14 vector.hh	69

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

geom	Line.hh Line class implementation	7
geom::detail	22

Chapter 2

Class Index

2.1 Class List

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

geom::Line< T >	
Line class implementation	25
geom::Plane< T >	
Plane class realization	28
geom::Triangle< T >	
Triangle class implementation	34
geom::Vector< T >	
Vector class realization	35

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/distance/ distance.hh	49
include/intersection/ intersection.hh	51
include/primitives/ line.hh	55
include/primitives/ plane.hh	58
include/primitives/ primitives.hh	63
include/primitives/ triangle.hh	64
include/primitives/ vector.hh	66

Chapter 4

Namespace Documentation

4.1 geom Namespace Reference

[line.hh](#) [Line](#) class implementation

Namespaces

- [detail](#)

Classes

- class [Line](#)
[Line](#) class implementation.
- class [Plane](#)
[Plane](#) class realization.
- class [Triangle](#)
[Triangle](#) class implementation.
- class [Vector](#)
[Vector](#) class realization.

Typedefs

- using [VectorD](#) = [Vector](#)< double >
- using [VectorF](#) = [Vector](#)< float >

Functions

- `template<std::floating_point T>`
`T distance (const Plane< T > &pl, const Vector< T > &pt)`
Calculates signed distance between point and plane.
- `template<std::floating_point T>`
`bool isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)`
Checks intersection of 2 triangles.
- `template<std::floating_point T>`
`std::variant< std::monostate, Line< T >, Plane< T > > intersect (const Plane< T > &pl1, const Plane< T > &pl2)`
Intersect 2 planes and return result of intersection.
- `template<std::floating_point T>`
`bool isIntersect (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`
`std::ostream & operator<< (std::ostream &ost, const Line< T > &line)`
Line print operator.
- `template<std::floating_point T>`
`bool operator== (const Line< T > &lhs, const Line< T > &rhs)`
Line equality operator.
- `template<std::floating_point T>`
`bool operator== (const Plane< T > &lhs, const Plane< T > &rhs)`
Plane equality operator.
- `template<std::floating_point T>`
`std::ostream & operator<< (std::ostream &ost, const Plane< T > &pl)`
Plane print operator.
- `template<std::floating_point T>`
`std::ostream & operator<< (std::ostream &ost, const Triangle< T > &tr)`
Triangle print operator.
- `template<std::floating_point T>`
`Vector< T > operator+ (const Vector< T > &lhs, const Vector< T > &rhs)`
Overloaded + operator.
- `template<std::floating_point T>`
`Vector< T > operator- (const Vector< T > &lhs, const Vector< T > &rhs)`
Overloaded - operator.
- `template<Number nT, std::floating_point T>`
`Vector< T > operator* (const nT &val, const Vector< T > &rhs)`
Overloaded multiple by value operator.
- `template<Number nT, std::floating_point T>`
`Vector< T > operator* (const Vector< T > &lhs, const nT &val)`
Overloaded multiple by value operator.
- `template<Number nT, std::floating_point T>`
`Vector< T > operator/ (const Vector< T > &lhs, const nT &val)`
Overloaded divide by value operator.
- `template<std::floating_point T>`
`T dot (const Vector< T > &lhs, const Vector< T > &rhs)`
Dot product function.
- `template<std::floating_point T>`
`Vector< T > cross (const Vector< T > &lhs, const Vector< T > &rhs)`
Cross product function.
- `template<std::floating_point T>`
`bool operator== (const Vector< T > &lhs, const Vector< T > &rhs)`
Vector equality operator.

- `template<std::floating_point T>`
`bool operator!= (const Vector< T > &lhs, const Vector< T > &rhs)`
Vector inequality operator.
- `template<std::floating_point T>`
`std::ostream & operator<< (std::ostream &ost, const Vector< T > &vec)`
Vector print operator.

Variables

- `template<class T >`
`concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>`
Useful concept which represents floating point and integral types.

4.1.1 Detailed Description

[line.hh](#) [Line](#) class implementation

[triangle.hh](#) [Triangle](#) class implementation

[Plane](#) class implementation.

4.1.2 Typedef Documentation

4.1.2.1 VectorD

```
using geom::VectorD = typedef Vector<double>
```

Definition at line 391 of file [vector.hh](#).

4.1.2.2 VectorF

```
using geom::VectorF = typedef Vector<float>
```

Definition at line 392 of file [vector.hh](#).

4.1.3 Function Documentation

4.1.3.1 distance()

```
template<std::floating_point T>
T geom::distance (
    const Plane< T > & pl,
    const Vector< T > & pt )
```

Calculates signed distance between point and plane.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>pl</i>	plane
<i>pt</i>	point

Returns

T signed distance between point and plane

Definition at line 26 of file [distance.hh](#).

References [geom::Plane< T >::dist\(\)](#), [dot\(\)](#), and [geom::Plane< T >::norm\(\)](#).

Referenced by [geom::detail::isIntersectMollerHaines\(\)](#), and [geom::detail::isOnOneSide\(\)](#).

4.1.3.2 isIntersect2D()

```
template<std::floating_point T>
bool geom::isIntersect2D (
    const Triangle< T > & tr1,
    const Triangle< T > & tr2 )
```

Checks intersection of 2 triangles.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>tr1</i>	first triangle
<i>tr2</i>	second triangle

Returns

true if triangles are intersect
false if triangles are not intersect

Definition at line 141 of file [intersection.hh](#).

Referenced by [isIntersect\(\)](#).

4.1.3.3 intersect()

```
template<std::floating_point T>
std::variant< std::monostate, Line< T >, Plane< T > > geom::intersect (
    const Plane< T > & p11,
    const Plane< T > & p12 )
```

Intersect 2 planes and return result of intersection.

Common intersection case (parallel planes case is trivial):

Let P = point in space

pl1 equation: $\text{dot}(n1, P) = d1$ pl2 equation: $\text{dot}(n2, P) = d2$

Intersection line direction: $\text{dir} = \text{cross}(n1, n2)$

Let origin of intersection line be a linear combination of n1 and n2: $P = a * n1 + b * n2$

P must satisfy both pl1 and pl2 equations: $\text{dot}(n1, P) = d1 \Leftrightarrow \text{dot}(n1, a * n1 + b * n2) = d1 \Leftrightarrow a + b * \text{dot}(n1, n2) = d1$
 $\text{dot}(n2, P) = d2 \Leftrightarrow \text{dot}(n2, a * n1 + b * n2) = d2 \Leftrightarrow a * \text{dot}(n1, n2) + b = d2$

Let's find a and b: $a = (d2 * \text{dot}(n1, n2) - d1) / ((\text{dot}(n1, n2))^2 - 1)$ $b = (d1 * \text{dot}(n1, n2) - d2) / ((\text{dot}(n1, n2))^2 - 1)$

Intersection line equation: $L = P + t * \text{cross}(n1, n2) = (a * n1 + b * n2) + t * \text{cross}(n1, n2)$

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>p1</i>	first plane
<i>p2</i>	second plane

Returns

`std::variant<std::monostate, Line<T>, Plane<T>>`

Definition at line 110 of file [intersection.hh](#).

References [cross\(\)](#), [geom::Plane< T >::dist\(\)](#), [dot\(\)](#), and [geom::Plane< T >::norm\(\)](#).

Referenced by [geom::detail::isIntersectMollerHaines\(\)](#).

4.1.3.4 isIntersect()

```
template<std::floating_point T>
bool geom::isIntersect (
```

```
const Triangle< T > & tr1,
const Triangle< T > & tr2 )
```

Definition at line 86 of file [intersection.hh](#).

References [geom::Plane< T >::getBy3Points\(\)](#), [geom::detail::isIntersect2D\(\)](#), [geom::detail::isIntersectMollerHaines\(\)](#), and [geom::detail::isOnOneSide\(\)](#).

4.1.3.5 operator<<() [1/4]

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Line< T > & line )
```

[Line](#) print operator.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>in, out</i>	<i>ost</i>	output stream
<i>in</i>	<i>line</i>	Line to print

Returns

std::ostream& modified ostream instance

Definition at line 89 of file [line.hh](#).

References [geom::Line< T >::dir\(\)](#), and [geom::Line< T >::org\(\)](#).

4.1.3.6 operator==() [1/3]

```
template<std::floating_point T>
bool geom::operator==(
    const Line< T > & lhs,
    const Line< T > & rhs )
```

[Line](#) equality operator.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>in</i>	<i>lhs</i>	1st line
<i>in</i>	<i>rhs</i>	2nd line

Returns

true if lines are equal
false if lines are not equal

Definition at line 105 of file [line.hh](#).

References [geom::Line< T >::isEqual\(\)](#).

4.1.3.7 operator==() [2/3]

```
template<std::floating_point T>
bool geom::operator== (
    const Plane< T > & lhs,
    const Plane< T > & rhs )
```

[Plane](#) equality operator.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>in</i>	<i>lhs</i>	1st plane
<i>in</i>	<i>rhs</i>	2nd plane

Returns

true if planes are equal
false if planes are not equal

Definition at line 147 of file [plane.hh](#).

References [geom::Plane< T >::isEqual\(\)](#).

4.1.3.8 operator<<() [2/4]

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Plane< T > & pl )
```

Plane print operator.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>in, out</i>	<i>ost</i>	output stream
<i>in</i>	<i>pl</i>	plane to print

Returns

std::ostream& modified ostream instance

Definition at line 161 of file [plane.hh](#).

References [geom::Plane< T >::dist\(\)](#), and [geom::Plane< T >::norm\(\)](#).

4.1.3.9 operator<<() [3/4]

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Triangle< T > & tr )
```

[Triangle](#) print operator.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>in, out</i>	<i>ost</i>	output stream
<i>in</i>	<i>tr</i>	Triangle to print

Returns

std::ostream& modified ostream instance

Definition at line 60 of file [triangle.hh](#).

4.1.3.10 operator+()

```
template<std::floating_point T>
Vector<T> geom::operator+ (
```

```
const Vector< T > & lhs,
const Vector< T > & rhs )
```

Overloaded + operator.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

in	<i>lhs</i>	first vector
in	<i>rhs</i>	second vector

Returns

Vector<T> sum of two vectors

Definition at line 244 of file [vector.hh](#).

4.1.3.11 operator-()

```
template<std::floating_point T>
Vector<T> geom::operator- (
    const Vector< T > & lhs,
    const Vector< T > & rhs )
```

Overloaded - operator.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

in	<i>lhs</i>	first vector
in	<i>rhs</i>	second vector

Returns

Vector<T> res of two vectors

Definition at line 260 of file [vector.hh](#).

4.1.3.12 operator*() [1/2]

```
template<Number nT, std::floating_point T>
Vector<T> geom::operator* (
    const nT & val,
    const Vector< T > & rhs )
```

Overloaded multiple by value operator.

Template Parameters

<i>nT</i>	type of value to multiply by
<i>T</i>	vector template parameter

Parameters

in	<i>val</i>	value to multiply by
in	<i>rhs</i>	vector to multiply by value

Returns

Vector<T> result vector

Definition at line 277 of file [vector.hh](#).

4.1.3.13 operator*() [2/2]

```
template<Number nT, std::floating_point T>
Vector<T> geom::operator* (
    const Vector< T > & lhs,
    const nT & val )
```

Overloaded multiple by value operator.

Template Parameters

<i>nT</i>	type of value to multiply by
<i>T</i>	vector template parameter

Parameters

in	<i>val</i>	value to multiply by
in	<i>lhs</i>	vector to multiply by value

Returns

Vector<T> result vector

Definition at line 294 of file [vector.hh](#).

4.1.3.14 operator/()

```
template<Number nT, std::floating_point T>
Vector<T> geom::operator/ (
    const Vector< T > & lhs,
    const nT & val )
```

Overloaded divide by value operator.

Template Parameters

<i>nT</i>	type of value to divide by
<i>T</i>	vector template parameter

Parameters

in	<i>val</i>	value to divide by
in	<i>lhs</i>	vector to divide by value

Returns

Vector<T> result vector

Definition at line 311 of file [vector.hh](#).

4.1.3.15 dot()

```
template<std::floating_point T>
T geom::dot (
    const Vector< T > & lhs,
    const Vector< T > & rhs )
```

Dot product function.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

in	<i>lhs</i>	first vector
in	<i>rhs</i>	second vector

Returns

T dot production

Definition at line 327 of file [vector.hh](#).

References [geom::Vector< T >::dot\(\)](#).

Referenced by [distance\(\)](#), [intersect\(\)](#), [geom::detail::isIntersectMollerHaines\(\)](#), [geom::Vector< T >::isPerp\(\)](#), and [geom::Vector< T >::length2\(\)](#).

4.1.3.16 cross()

```
template<std::floating_point T>
Vector<T> geom::cross (
    const Vector< T > & lhs,
    const Vector< T > & rhs )
```

Cross product function.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

in	<i>lhs</i>	first vector
in	<i>rhs</i>	second vector

Returns

T cross production

Definition at line 341 of file [vector.hh](#).

References [geom::Vector< T >::cross\(\)](#).

Referenced by [intersect\(\)](#), and [geom::Vector< T >::isPar\(\)](#).

4.1.3.17 operator==() [3/3]

```
template<std::floating_point T>
bool geom::operator== (
    const Vector< T > & lhs,
    const Vector< T > & rhs )
```

[Vector](#) equality operator.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

<i>in</i>	<i>lhs</i>	first vector
<i>in</i>	<i>rhs</i>	second vector

Returns

true if vectors are equal
false otherwise

Definition at line 356 of file [vector.hh](#).

References [geom::Vector< T >::isEqual\(\)](#).

4.1.3.18 operator"!=()

```
template<std::floating_point T>
bool geom::operator!= (
    const Vector< T > & lhs,
    const Vector< T > & rhs )
```

[Vector](#) inequality operator.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

<i>in</i>	<i>lhs</i>	first vector
<i>in</i>	<i>rhs</i>	second vector

Returns

true if vectors are not equal
false otherwise

Definition at line 371 of file [vector.hh](#).

4.1.3.19 operator<<() [4/4]

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Vector< T > & vec )
```

Vector print operator.

Template Parameters

<i>T</i>	vector template parameter
----------	---------------------------

Parameters

<i>in, out</i>	<i>ost</i>	output stream
<i>in</i>	<i>vec</i>	vector to print

Returns

std::ostream& modified stream instance

Definition at line 385 of file [vector.hh](#).

References [geom::Vector< T >::x](#), [geom::Vector< T >::y](#), and [geom::Vector< T >::z](#).

4.1.4 Variable Documentation**4.1.4.1 Number**

```
template<class T >
concept geom::Number = std::is_floating_point_v<T> || std::is_integral_v<T>
```

Useful concept which represents floating point and integral types.

@concept Number

Template Parameters

<i>T</i>	
----------	--

Definition at line 25 of file [vector.hh](#).

4.2 geom::detail Namespace Reference

Functions

- `template<std::floating_point T>`
`bool isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)`
Checks intersection of 2 triangles.
- `template<std::floating_point T>`
`bool isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`
`bool isOverlap (const T ¶ms10, const T ¶ms11, const T ¶ms20, const T ¶ms21)`
- `template<std::forward_iterator It>`
`bool isSameSign (It begin, It end)`
- `template<std::floating_point T>`
`bool isOnOneSide (const Plane< T > &p1, const Triangle< T > &tr)`

4.2.1 Function Documentation

4.2.1.1 isIntersect2D()

```
template<std::floating_point T>
bool geom::detail::isIntersect2D (
    const Triangle< T > & tr1,
    const Triangle< T > & tr2 )
```

Checks intersection of 2 triangles.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Parameters

<i>tr1</i>	first triangle
<i>tr2</i>	second triangle

Returns

true if triangles are intersect
false if triangles are not intersect

Definition at line 141 of file [intersection.hh](#).

Referenced by [geom::isIntersect\(\)](#).

4.2.1.2 isIntersectMollerHaines()

```
template<std::floating_point T>
bool geom::detail::isIntersectMollerHaines (
    const Triangle< T > & tr1,
    const Triangle< T > & tr2 )
```

Definition at line 148 of file [intersection.hh](#).

References [geom::distance\(\)](#), [geom::dot\(\)](#), [geom::Plane< T >::getBy3Points\(\)](#), [geom::intersect\(\)](#), and [isOverlap\(\)](#).

Referenced by [geom::isIntersect\(\)](#).

4.2.1.3 isOverlap()

```
template<std::floating_point T>
bool geom::detail::isOverlap (
    const T & params10,
    const T & params11,
    const T & params20,
    const T & params21 )
```

Definition at line 213 of file [intersection.hh](#).

Referenced by [isIntersectMollerHaines\(\)](#).

4.2.1.4 isSameSign()

```
template<std::forward_iterator It>
bool geom::detail::isSameSign (
    It begin,
    It end )
```

Definition at line 219 of file [intersection.hh](#).

Referenced by [isOnOneSide\(\)](#).

4.2.1.5 isOnOneSide()

```
template<std::floating_point T>
bool geom::detail::isOnOneSide (
    const Plane< T > & pl,
    const Triangle< T > & tr )
```

Definition at line 232 of file [intersection.hh](#).

References [geom::distance\(\)](#), and [isSameSign\(\)](#).

Referenced by [geom::isIntersect\(\)](#).

Chapter 5

Class Documentation

5.1 geom::Line< T > Class Template Reference

[Line](#) class implementation.

```
#include <line.hh>
```

Public Member Functions

- [Line](#) (const [Vector](#)< T > &[org](#), const [Vector](#)< T > &[dir](#))
Construct a new [Line](#) object.
- const [Vector](#)< T > & [org](#) () const
Getter for origin vector.
- const [Vector](#)< T > & [dir](#) () const
Getter for direction vector.
- bool [belongs](#) (const [Vector](#)< T > &point) const
Checks is point belongs to line.
- bool [isEqual](#) (const [Line](#) &line) const
*Checks is *this equals to another line.*

Static Public Member Functions

- static [Line](#) [getBy2Points](#) (const [Vector](#)< T > &p1, const [Vector](#)< T > &p2)
Get line by 2 points.

5.1.1 Detailed Description

```
template<std::floating_point T>  
class geom::Line< T >
```

[Line](#) class implementation.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Definition at line 21 of file [line.hh](#).

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Line()

```
template<std::floating_point T>
geom::Line< T >::Line (
    const Vector< T > & org,
    const Vector< T > & dir )
```

Construct a new [Line](#) object.

Parameters

in	<i>org</i>	origin vector
in	<i>dir</i>	direction vector

Definition at line 111 of file [line.hh](#).

References [geom::Line< T >::org\(\)](#).

5.1.3 Member Function Documentation

5.1.3.1 org()

```
template<std::floating_point T>
const Vector< T > & geom::Line< T >::org
```

Getter for origin vector.

Returns

const Vector<T>& const reference to origin vector

Definition at line 118 of file [line.hh](#).

Referenced by [geom::Plane< T >::belongs\(\)](#), [geom::Line< T >::Line\(\)](#), and [geom::operator<<\(\)](#).

5.1.3.2 dir()

```
template<std::floating_point T>
const Vector< T > & geom::Line< T >::dir
```

Getter for direction vector.

Returns

const Vector<T>& const reference to direction vector

Definition at line 124 of file [line.hh](#).

Referenced by [geom::Plane< T >::belongs\(\)](#), and [geom::operator<<\(\)](#).

5.1.3.3 belongs()

```
template<std::floating_point T>
bool geom::Line< T >::belongs (
    const Vector< T > & point ) const
```

Checks is point belongs to line.

Parameters

in	<i>point</i>	const reference to point vector
----	--------------	---------------------------------

Returns

true if point belongs to line
false if point doesn't belong to line

Definition at line 130 of file [line.hh](#).

5.1.3.4 isEqual()

```
template<std::floating_point T>
bool geom::Line< T >::isEqual (
    const Line< T > & line ) const
```

Checks is *this equals to another line.

Parameters

in	<i>line</i>	const reference to another line
----	-------------	---------------------------------

Returns

true if lines are equal
false if lines are not equal

Definition at line 136 of file [line.hh](#).

Referenced by [geom::operator==\(\)](#).

5.1.3.5 getBy2Points()

```
template<std::floating_point T>
Line< T > geom::Line< T >::getBy2Points (
    const Vector< T > & p1,
    const Vector< T > & p2 ) [static]
```

Get line by 2 points.

Parameters

in	<i>p1</i>	1st point
in	<i>p2</i>	2nd point

Returns

[Line](#) passing through two points

Definition at line 142 of file [line.hh](#).

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

- include/primitives/[line.hh](#)

5.2 geom::Plane< T > Class Template Reference

[Plane](#) class realization.

```
#include <plane.hh>
```

Public Member Functions

- T [dist](#) () const
Getter for distance.
- const Vector< T > & [norm](#) () const
Getter for normal vector.
- bool [belongs](#) (const Vector< T > &point) const
Checks if point belongs to plane.
- bool [belongs](#) (const Line< T > &line) const
Checks if line belongs to plane.
- bool [isEqual](#) (const Plane &rhs) const
*Checks is *this equals to another plane.*
- bool [isPar](#) (const Plane &rhs) const
*Checks is *this is parallel to another plane.*

Static Public Member Functions

- static [Plane getBy3Points](#) (const [Vector](#)< T > &pt1, const [Vector](#)< T > &pt2, const [Vector](#)< T > &pt3)
Get plane by 3 points.
- static [Plane getParametric](#) (const [Vector](#)< T > &org, const [Vector](#)< T > &dir1, const [Vector](#)< T > &dir2)
Get plane from parametric plane equation.
- static [Plane getNormalPoint](#) (const [Vector](#)< T > &norm, const [Vector](#)< T > &point)
Get plane from normal point plane equation.
- static [Plane getNormalDist](#) (const [Vector](#)< T > &norm, T constant)
Get plane from normal const plane equation.

5.2.1 Detailed Description

```
template<std::floating_point T>
class geom::Plane< T >
```

[Plane](#) class realization.

Template Parameters

T	- floating point type of coordinates
-------------------	--------------------------------------

Definition at line 24 of file [plane.hh](#).

5.2.2 Member Function Documentation

5.2.2.1 dist()

```
template<std::floating_point T>
T geom::Plane< T >::dist
```

Getter for distance.

Returns

T value of distance

Definition at line 175 of file [plane.hh](#).

Referenced by [geom::distance\(\)](#), [geom::intersect\(\)](#), and [geom::operator<<\(\)](#).

5.2.2.2 norm()

```
template<std::floating_point T>
const Vector< T > & geom::Plane< T >::norm
```

Getter for normal vector.

Returns

const Vector<T>& const reference to normal vector

Definition at line 181 of file [plane.hh](#).

Referenced by [geom::distance\(\)](#), [geom::intersect\(\)](#), and [geom::operator<<\(\)](#).

5.2.2.3 belongs() [1/2]

```
template<std::floating_point T>
bool geom::Plane< T >::belongs (
    const Vector< T > & point ) const
```

Checks if point belongs to plane.

Parameters

in	<i>point</i>	const referene to point vector
----	--------------	--------------------------------

Returns

true if point belongs to plane

false if point doesn't belong to plane

Definition at line 187 of file [plane.hh](#).

5.2.2.4 belongs() [2/2]

```
template<std::floating_point T>
bool geom::Plane< T >::belongs (
    const Line< T > & line ) const
```

Checks if line belongs to plane.

Parameters

in	<i>line</i>	const referene to line
----	-------------	------------------------

Returns

true if line belongs to plane
false if line doesn't belong to plane

Definition at line 193 of file [plane.hh](#).

References [geom::Line< T >::dir\(\)](#), and [geom::Line< T >::org\(\)](#).

5.2.2.5 isEqual()

```
template<std::floating_point T>
bool geom::Plane< T >::isEqual (
    const Plane< T > & rhs ) const
```

Checks is *this equals to another plane.

Parameters

<code>in</code>	<code>rhs</code>	const reference to another plane
-----------------	------------------	----------------------------------

Returns

true if planes are equal
false if planes are not equal

Definition at line 199 of file [plane.hh](#).

Referenced by [geom::operator==\(\)](#).

5.2.2.6 isPar()

```
template<std::floating_point T>
bool geom::Plane< T >::isPar (
    const Plane< T > & rhs ) const
```

Checks is *this is parallel to another plane.

Parameters

<code>in</code>	<code>rhs</code>	const reference to another plane
-----------------	------------------	----------------------------------

Returns

true if planes are parallel
false if planes are not parallel

Definition at line 205 of file [plane.hh](#).

References [geom::Plane< T >::isPar\(\)](#).

Referenced by [geom::Plane< T >::isPar\(\)](#).

5.2.2.7 getBy3Points()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getBy3Points (
    const Vector< T > & pt1,
    const Vector< T > & pt2,
    const Vector< T > & pt3 ) [static]
```

Get plane by 3 points.

Parameters

in	<i>pt1</i>	1st point
in	<i>pt2</i>	2nd point
in	<i>pt3</i>	3rd point

Returns

[Plane](#) passing through three points

Definition at line 211 of file [plane.hh](#).

Referenced by [geom::isIntersect\(\)](#), and [geom::detail::isIntersectMollerHaines\(\)](#).

5.2.2.8 getParametric()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getParametric (
    const Vector< T > & org,
    const Vector< T > & dir1,
    const Vector< T > & dir2 ) [static]
```

Get plane from parametric plane equation.

Parameters

in	<i>org</i>	origin vector
in	<i>dir1</i>	1st direction vector
in	<i>dir2</i>	2nd direction vector

Returns

[Plane](#)

Definition at line 218 of file [plane.hh](#).

References [geom::Vector< T >::cross\(\)](#).

5.2.2.9 getNormalPoint()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalPoint (
    const Vector< T > & norm,
    const Vector< T > & point ) [static]
```

Get plane from normal point plane equation.

Parameters

in	<i>norm</i>	normal vector
in	<i>point</i>	point lying on the plane

Returns

[Plane](#)

Definition at line 226 of file [plane.hh](#).

References [geom::Vector< T >::normalized\(\)](#).

5.2.2.10 getNormalDist()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalDist (
    const Vector< T > & norm,
    T constant ) [static]
```

Get plane form normal const plane equation.

Parameters

in	<i>norm</i>	normal vector
in	<i>constant</i>	distance

Returns

[Plane](#)

Definition at line 233 of file [plane.hh](#).

References [geom::Vector< T >::normalized\(\)](#).

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

- [include/primitives/plane.hh](#)

5.3 [geom::Triangle< T >](#) Class Template Reference

[Triangle](#) class implementation.

```
#include <triangle.hh>
```

Public Member Functions

- [Triangle](#) (const [Vector< T >](#) &p1, const [Vector< T >](#) &p2, const [Vector< T >](#) &p3)
Construct a new [Triangle](#) object from 3 points.
- const [Vector< T >](#) & [operator\[\]](#) (std::size_t idx) const
Overloaded operator[] to get access to vertices.

5.3.1 Detailed Description

```
template<std::floating_point T>
class geom::Triangle< T >
```

[Triangle](#) class implementation.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Definition at line 24 of file [triangle.hh](#).

5.3.2 Constructor & Destructor Documentation

5.3.2.1 [Triangle\(\)](#)

```
template<std::floating_point T>
geom::Triangle< T >::Triangle (
```



```
const Vector< T > & p1,
const Vector< T > & p2,
const Vector< T > & p3 )
```

Construct a new [Triangle](#) object from 3 points.

Parameters

in	<i>p1</i>	1st point
in	<i>p2</i>	2nd point
in	<i>p3</i>	3rd point

Definition at line 72 of file [triangle.hh](#).

5.3.3 Member Function Documentation

5.3.3.1 operator[]()

```
template<std::floating_point T>
const Vector< T > & geom::Triangle< T >::operator[] (
    std::size_t idx ) const
```

Overloaded operator[] to get access to vertices.

Parameters

in	<i>idx</i>	index of vertex
----	------------	-----------------

Returns

const Vector<T>& const reference to vertex

Definition at line 77 of file [triangle.hh](#).

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

- include/primitives/[triangle.hh](#)

5.4 geom::Vector< T > Class Template Reference

[Vector](#) class realization.

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

Public Member Functions

- [Vector](#) (T coordX, T coordY, T coordZ)
Construct a new [Vector](#) object from 3 coordinates.
- [Vector](#) (T coordX={})
Construct a new [Vector](#) object with equals coordinates.
- [Vector](#) & [operator+=](#) (const [Vector](#) &vec)
Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.
- [Vector](#) & [operator-=](#) (const [Vector](#) &vec)
Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.
- [Vector](#) [operator-](#) () const
Unary - operator.
- template<Number nType>
[Vector](#) & [operator*=
Overloaded *= by number operator.](#) (nType val)
- template<Number nType>
[Vector](#) & [operator/=](#) (nType val)
Overloaded /= by number operator.
- T [dot](#) (const [Vector](#) &rhs) const
Dot product function.
- [Vector](#) [cross](#) (const [Vector](#) &rhs) const
Cross product function.
- T [length2](#) () const
Calculate squared length of a vector function.
- T [length](#) () const
Calculate length of a vector function.
- [Vector](#) [normalized](#) () const
Get normalized vector function.
- [Vector](#) & [normalize](#) ()
Normalize vector function.
- T & [operator\[\]](#) (size_t i)
Overloaded operator [] (non-const version) To get access to coordinates.
- T [operator\[\]](#) (size_t i) const
Overloaded operator [] (const version) To get access to coordinates.
- bool [isPar](#) (const [Vector](#) &rhs) const
Check if vector is parallel to another.
- bool [isPerp](#) (const [Vector](#) &rhs) const
Check if vector is perpendicular to another.
- bool [isEqual](#) (const [Vector](#) &rhs) const
Check if vector is equal to another.
- template<Number nType>
[Vector](#)< T > & [operator*=
Overloaded *= by number operator.](#) (nType val)
- template<Number nType>
[Vector](#)< T > & [operator/=](#) (nType val)

Static Public Member Functions

- static bool [isNumEq](#) (T lhs, T rhs)
Check equality (with threshold) of two floating point numbers function.
- static void [setThreshold](#) (T thres)
Set new threshold value.
- static void [getThreshold](#) ()
Get current threshold value.
- static void [setDefThreshold](#) ()
Set threshold to default value.

Public Attributes

- `T x {}`
Vector coordinates.
- `T y {}`
- `T z {}`

5.4.1 Detailed Description

```
template<std::floating_point T>
class geom::Vector< T >
```

Vector class realization.

Template Parameters

<i>T</i>	- floating point type of coordinates
----------	--------------------------------------

Definition at line 34 of file [vector.hh](#).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Vector() [1/2]

```
template<std::floating_point T>
geom::Vector< T >::Vector (
    T coordX,
    T coordY,
    T coordZ ) [inline]
```

Construct a new *Vector* object from 3 coordinates.

Parameters

in	<i>coordX</i>	x coordinate
in	<i>coordY</i>	y coordinate
in	<i>coordZ</i>	z coordinate

Definition at line 55 of file [vector.hh](#).

5.4.2.2 Vector() [2/2]

```
template<std::floating_point T>
geom::Vector< T >::Vector (
    T coordX = {} ) [inline], [explicit]
```

Construct a new [Vector](#) object with equals coordinates.

Parameters

in	<i>coordX</i>	coordinate (default to {})
----	---------------	----------------------------

Definition at line 63 of file [vector.hh](#).

5.4.3 Member Function Documentation

5.4.3.1 operator+=()

```
template<std::floating_point T>
Vector< T > & geom::Vector< T >::operator+= (
    const Vector< T > & vec )
```

Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.

Parameters

in	<i>vec</i>	vector to incremented with
----	------------	----------------------------

Returns

[Vector](#)& reference to current instance

Definition at line 395 of file [vector.hh](#).

References [geom::Vector< T >::x](#), [geom::Vector< T >::y](#), and [geom::Vector< T >::z](#).

5.4.3.2 operator-=()

```
template<std::floating_point T>
Vector< T > & geom::Vector< T >::operator-= (
    const Vector< T > & vec )
```

Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.

Parameters

in	<i>vec</i>	vector to decremented with
----	------------	----------------------------

Returns

[Vector](#)& reference to current instance

Definition at line 405 of file [vector.hh](#).

References [geom::Vector< T >::x](#), [geom::Vector< T >::y](#), and [geom::Vector< T >::z](#).

5.4.3.3 operator-()

```
template<std::floating_point T>
Vector< T > geom::Vector< T >::operator-
```

Unary - operator.

Returns

[Vector](#) negated [Vector](#) instance

Definition at line 415 of file [vector.hh](#).

5.4.3.4 operator*=() [1/2]

```
template<std::floating_point T>
template<Number nType>
Vector& geom::Vector< T >::operator*= (
    nType val )
```

Overloaded *= by number operator.

Template Parameters

<i>nType</i>	numeric type of value to multiply by
--------------	--------------------------------------

Parameters

in	<i>val</i>	value to multiply by
----	------------	----------------------

Returns

[Vector](#)& reference to vector instance

5.4.3.5 operator/=() [1/2]

```
template<std::floating_point T>
template<Number nType>
Vector& geom::Vector< T >::operator/= (
    nType val )
```

Overloaded /= by number operator.

Template Parameters

<i>nType</i>	numeric type of value to divide by
--------------	------------------------------------

Parameters

in	<i>val</i>	value to divide by
----	------------	--------------------

Returns

Vector& reference to vector instance

Warning

Does not check if val equals 0

5.4.3.6 dot()

```
template<std::floating_point T>
T geom::Vector< T >::dot (
    const Vector< T > & rhs ) const
```

Dot product function.

Parameters

<i>rhs</i>	vector to dot product with
------------	----------------------------

Returns

T dot product of two vectors

Definition at line 443 of file [vector.hh](#).

References [geom::Vector< T >::x](#), [geom::Vector< T >::y](#), and [geom::Vector< T >::z](#).

Referenced by [geom::dot\(\)](#).

5.4.3.7 cross()

```
template<std::floating_point T>
Vector< T > geom::Vector< T >::cross (
    const Vector< T > & rhs ) const
```

Cross product function.

Parameters

<i>rhs</i>	vector to cross product with
------------	------------------------------

Returns

Vector cross product of two vectors

Definition at line 449 of file [vector.hh](#).

References [geom::Vector< T >::x](#), [geom::Vector< T >::y](#), and [geom::Vector< T >::z](#).

Referenced by [geom::cross\(\)](#), and [geom::Plane< T >::getParametric\(\)](#).

5.4.3.8 length2()

```
template<std::floating_point T>
T geom::Vector< T >::length2
```

Calculate squared length of a vector function.

Returns

T length²

Definition at line 455 of file [vector.hh](#).

References [geom::dot\(\)](#).

5.4.3.9 length()

```
template<std::floating_point T>
T geom::Vector< T >::length
```

Calculate length of a vector function.

Returns

T length

Definition at line 461 of file [vector.hh](#).

5.4.3.10 normalized()

```
template<std::floating_point T>
Vector< T > geom::Vector< T >::normalized
```

Get normalized vector function.

Returns

Vector normalized vector

Definition at line 467 of file [vector.hh](#).

References [geom::Vector< T >::normalize\(\)](#).

Referenced by [geom::Plane< T >::getNormalDist\(\)](#), and [geom::Plane< T >::getNormalPoint\(\)](#).

5.4.3.11 normalize()

```
template<std::floating_point T>
Vector< T > & geom::Vector< T >::normalize
```

Normalize vector function.

Returns

Vector& reference to instance

Definition at line 475 of file [vector.hh](#).

Referenced by [geom::Vector< T >::normalized\(\)](#).

5.4.3.12 operator[]() [1/2]

```
template<std::floating_point T>
T & geom::Vector< T >::operator[] (
    size_t i )
```

Overloaded operator [] (non-const version) To get access to coordinates.

Parameters

<i>i</i>	index of coordinate (0 - x, 1 - y, 2 - z)
----------	---

Returns

T& reference to coordinate value

Note

Coordinates calculated by mod 3

Definition at line 484 of file [vector.hh](#).

5.4.3.13 operator[]() [2/2]

```
template<std::floating_point T>
T geom::Vector< T >::operator[] (
    size_t i ) const
```

Overloaded operator [] (const version) To get access to coordinates.

Parameters

<i>i</i>	index of coordinate (0 - x, 1 - y, 2 - z)
----------	---

Returns

T coordinate value

Note

Coordinates calculated by mod 3

Definition at line 500 of file [vector.hh](#).

5.4.3.14 isPar()

```
template<std::floating_point T>
bool geom::Vector< T >::isPar (
    const Vector< T > & rhs ) const
```

Check if vector is parallel to another.

Parameters

<i>in</i>	<i>rhs</i>	vector to check parallelism with
-----------	------------	----------------------------------

Returns

true if vector is parallel
false otherwise

Definition at line 516 of file [vector.hh](#).

References [geom::cross\(\)](#).

5.4.3.15 isPerp()

```
template<std::floating_point T>
bool geom::Vector< T >::isPerp (
    const Vector< T > & rhs ) const
```

Check if vector is perpendicular to another.

Parameters

<i>in</i>	<i>rhs</i>	vector to check perpendicularity with
-----------	------------	---------------------------------------

Returns

true if vector is perpendicular
false otherwise

Definition at line 522 of file [vector.hh](#).

References [geom::dot\(\)](#).

5.4.3.16 isEqual()

```
template<std::floating_point T>
bool geom::Vector< T >::isEqual (
    const Vector< T > & rhs ) const
```

Check if vector is equal to another.

Parameters

<i>in</i>	<i>rhs</i>	vector to check equality with
-----------	------------	-------------------------------

Returns

true if vector is equal
false otherwise

Note

Equality check performs using `isNumEq(T lhs, T rhs)` function

Definition at line 528 of file `vector.hh`.

References `geom::Vector< T >::x`, `geom::Vector< T >::y`, and `geom::Vector< T >::z`.

Referenced by `geom::operator==()`.

5.4.3.17 isNumEq()

```
template<std::floating_point T>
bool geom::Vector< T >::isNumEq (
    T lhs,
    T rhs ) [static]
```

Check equality (with threshold) of two floating point numbers function.

Parameters

in	<i>lhs</i>	first number
in	<i>rhs</i>	second number

Returns

true if numbers equals with threshold ($|lhs - rhs| < \text{threshold}$)
false otherwise

Note

Threshold defined by `threshold_` static member

Definition at line 534 of file `vector.hh`.

5.4.3.18 setThreshold()

```
template<std::floating_point T>
void geom::Vector< T >::setThreshold (
    T thres ) [static]
```

Set new threshold value.

Parameters

in	<i>thres</i>	value to set
----	--------------	--------------

Definition at line 540 of file [vector.hh](#).

5.4.3.19 `getThreshold()`

```
template<std::floating_point T>
void geom::Vector< T >::getThreshold [static]
```

Get current threshold value.

Definition at line 546 of file [vector.hh](#).

5.4.3.20 `setDefThreshold()`

```
template<std::floating_point T>
void geom::Vector< T >::setDefThreshold [static]
```

Set threshold to default value.

Note

default value equals float point epsilon

Definition at line 552 of file [vector.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vector<T>& geom::Vector< T >::operator*= (
    nType val )
```

Definition at line 422 of file [vector.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vector<T>& geom::Vector< T >::operator/= (
    nType val )
```

Definition at line 433 of file [vector.hh](#).

5.4.4 Member Data Documentation

5.4.4.1 x

```
template<std::floating_point T>
T geom::Vector< T >::x {}
```

Vector coordinates.

Definition at line 46 of file [vector.hh](#).

Referenced by [geom::Vector< T >::cross\(\)](#), [geom::Vector< T >::dot\(\)](#), [geom::Vector< T >::isEqual\(\)](#), [geom::Vector< T >::operator+](#), [geom::Vector< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

5.4.4.2 y

```
template<std::floating_point T>
T geom::Vector< T >::y {}
```

Definition at line 46 of file [vector.hh](#).

Referenced by [geom::Vector< T >::cross\(\)](#), [geom::Vector< T >::dot\(\)](#), [geom::Vector< T >::isEqual\(\)](#), [geom::Vector< T >::operator+](#), [geom::Vector< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

5.4.4.3 z

```
template<std::floating_point T>
T geom::Vector< T >::z {}
```

Definition at line 46 of file [vector.hh](#).

Referenced by [geom::Vector< T >::cross\(\)](#), [geom::Vector< T >::dot\(\)](#), [geom::Vector< T >::isEqual\(\)](#), [geom::Vector< T >::operator+](#), [geom::Vector< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

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

- [include/primitives/vector.hh](#)

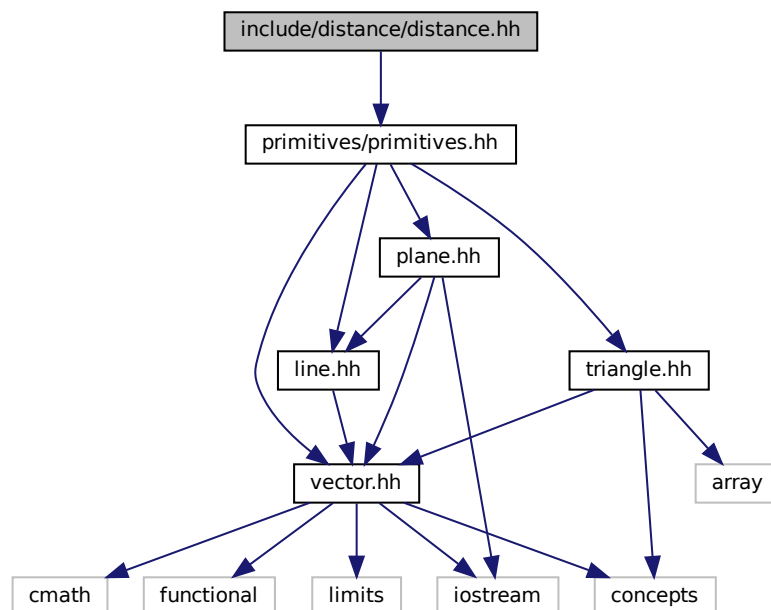
Chapter 6

File Documentation

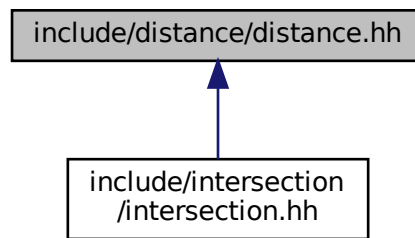
6.1 include/distance/distance.hh File Reference

```
#include "primitives/primitives.hh"
```

Include dependency graph for distance.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- [geom](#)
line.hh Line class implementation

Functions

- `template<std::floating_point T>`
`T geom::distance (const Plane< T > &pl, const Vector< T > &pt)`
Calculates signed distance between point and plane.

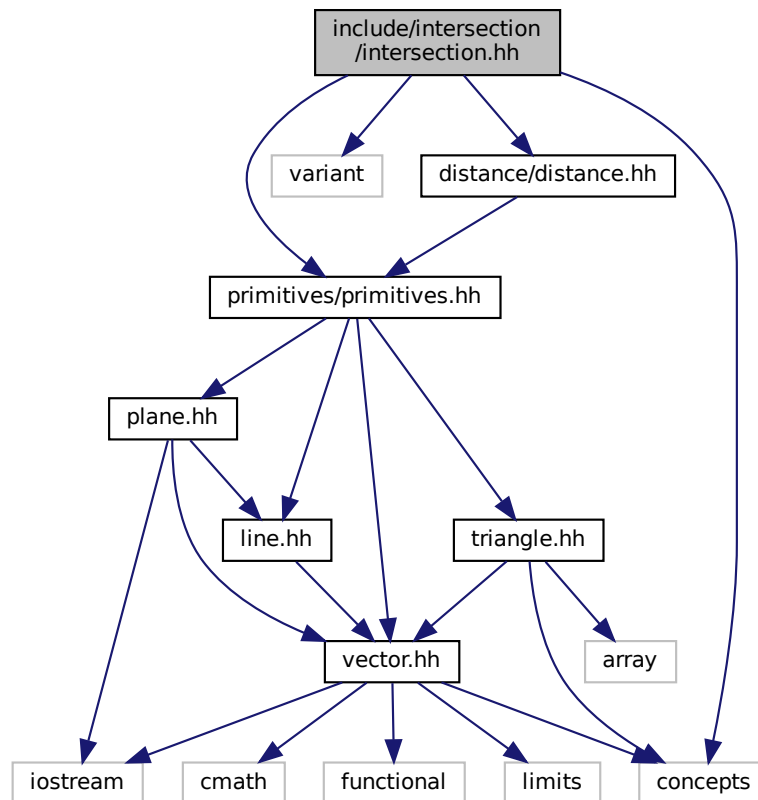
6.2 distance.hh

```

00001 #ifndef __INCLUDE_DISTANCE_DISTANCE_HH__
00002 #define __INCLUDE_DISTANCE_DISTANCE_HH__
00003
00004 #include "primitives/primitives.hh"
00005
00006 namespace geom
00007 {
00008
00009 /**
00010  * @brief Calculates signed distance between point and plane
00011  *
00012  * @tparam T - floating point type of coordinates
00013  * @param pl plane
00014  * @param pt point
00015  * @return T signed distance between point and plane
00016  */
00017 template <std::floating_point T>
00018 T distance(const Plane<T> &pl, const Vector<T> &pt);
00019
00020 } // namespace geom
00021
00022 namespace geom
00023 {
00024
00025 template <std::floating_point T>
00026 T distance(const Plane<T> &pl, const Vector<T> &pt)
00027 {
00028     return dot(pt, pl.norm()) - pl.dist();
00029 }
00030
00031 } // namespace geom
00032
00033 #endif // __INCLUDE_DISTANCE_DISTANCE_HH__
  
```


6.3 include/intersection/intersection.hh File Reference

```
#include <concepts>
#include <variant>
#include "distance/distance.hh"
#include "primitives/primitives.hh"
Include dependency graph for intersection.hh:
```



Namespaces

- [geom](#)
 [line.hh](#) *Line* class implementation
- [geom::detail](#)

Functions

- `template<std::floating_point T>`
 `bool geom::isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)`
 Checks intersection of 2 triangles.
- `template<std::floating_point T>`
 `std::variant< std::monostate, Line< T >, Plane< T > > geom::intersect (const Plane< T > &pl1, const Plane< T > &pl2)`

Intersect 2 planes and return result of intersection.

- `template<std::floating_point T>`
`bool geom::detail::isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)`

Checks intersection of 2 triangles.

- `template<std::floating_point T>`
`bool geom::detail::isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`
`bool geom::detail::isOverlap (const T ¶ms10, const T ¶ms11, const T ¶ms20, const T ¶ms21)`
- `template<std::forward_iterator It>`
`bool geom::detail::isSameSign (It begin, It end)`
- `template<std::floating_point T>`
`bool geom::detail::isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)`
- `template<std::floating_point T>`
`bool geom::isIntersect (const Triangle< T > &tr1, const Triangle< T > &tr2)`

6.4 intersection.hh

```
00001 #ifndef __INCLUDE_INTERSECTION_INTERSECTION_HH__
00002 #define __INCLUDE_INTERSECTION_INTERSECTION_HH__
00003
00004 #include <concepts>
00005 #include <variant>
00006
00007 #include "distance/distance.hh"
00008 #include "primitives/primitives.hh"
00009
00010 namespace geom
00011 {
00012
00013 /**
00014  * @brief Checks intersection of 2 triangles
00015  *
00016  * @tparam T - floating point type of coordinates
00017  * @param tr1 first triangle
00018  * @param tr2 second triangle
00019  * @return true if triangles are intersect
00020  * @return false if triangles are not intersect
00021  */
00022 template <std::floating_point T>
00023 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2);
00024
00025 /**
00026  * @brief Intersect 2 planes and return result of intersection
00027  * @details
00028  * Common intersection case (parallel planes case is trivial):
00029  *
00030  * Let P = point in space
00031  *
00032  * p11 equation: dot(n1, P) = d1
00033  * p12 equation: dot(n2, P) = d2
00034  *
00035  * Intersection line direction: dir = cross(n1, n2)
00036  *
00037  * Let origin of intersection line be a linear combination of n1 and n2:
00038  * P = a * n1 + b * n2
00039  *
00040  * P must satisfy both p11 and p12 equations:
00041  * dot(n1, P) = d1 ==> dot(n1, a * n1 + b * n2) = d1 ==> a + b * dot(n1, n2) = d1
00042  * dot(n2, P) = d2 ==> dot(n2, a * n1 + b * n2) = d2 ==> a * dot(n1, n2) + b = d2
00043  *
00044  * Let's find a and b:
00045  * a = (d2 * dot(n1, n2) - d1) / ((dot(n1, n2))^2 - 1)
00046  * b = (d1 * dot(n1, n2) - d2) / ((dot(n1, n2))^2 - 1)
00047  *
00048  * Intersection line equation:
00049  * L = P + t * cross(n1, n2) = (a * n1 + b * n2) + t * cross(n1, n2)
00050  *
00051  * @tparam T - floating point type of coordinates
00052  * @param p11 first plane
00053  * @param p12 second plane
00054  * @return std::variant<std::monostate, Line<T>, Plane<T>
00055  */
00056 template <std::floating_point T>
00057 std::variant<std::monostate, Line<T>, Plane<T> intersect(const Plane<T> &p11,
```

```

00058                                     const Plane<T> &p12);
00059
00060 namespace detail
00061 {
00062     template <std::floating_point T>
00063     bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2);
00064
00065     template <std::floating_point T>
00066     bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2);
00067
00068     template <std::floating_point T>
00069     bool isOverlap(const T &params10, const T &params11, const T &params20,
00070                   const T &params21);
00071
00072     template <std::forward_iterator It>
00073     bool isSameSign(It begin, It end);
00074
00075     template <std::floating_point T>
00076     bool isOnOneSide(const Plane<T> &p1, const Triangle<T> &tr);
00077
00078 } // namespace detail
00079 } // namespace geom
00080
00081 namespace geom
00082 {
00083     template <std::floating_point T>
00084     bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
00085     {
00086         /* TODO: handle invalid triangles case */
00087
00088         auto p11 = Plane<T>::getBy3Points(tr1[0], tr1[1], tr1[2]);
00089
00090         if (!detail::isOnOneSide(p11, tr2))
00091             return false;
00092
00093         auto p12 = Plane<T>::getBy3Points(tr2[0], tr2[1], tr2[2]);
00094
00095         if (p11 == p12)
00096             return detail::isIntersect2D(tr1, tr2);
00097
00098         if (p11.isPar(p12))
00099             return false;
00100
00101         if (!detail::isOnOneSide(p12, tr1))
00102             return false;
00103
00104         return detail::isIntersectMollerHaines(tr1, tr2);
00105     }
00106
00107     template <std::floating_point T>
00108     std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &p11,
00109                                                             const Plane<T> &p12)
00110     {
00111         const auto &n1 = p11.norm();
00112         const auto &n2 = p12.norm();
00113
00114         auto dir = cross(n1, n2);
00115
00116         /* if planes are parallel */
00117         if (Vector<T>{0} == dir)
00118         {
00119             if (p11 == p12)
00120                 return p11;
00121             return std::monostate{};
00122         }
00123
00124         auto n1n2 = dot(n1, n2);
00125         auto d1 = p11.dist();
00126         auto d2 = p12.dist();
00127
00128         auto a = (d2 * n1n2 - d1) / (n1n2 * n1n2 - 1);
00129         auto b = (d1 * n1n2 - d2) / (n1n2 * n1n2 - 1);
00130
00131         return Line<T>{(a * n1) + (b * n2), dir};
00132     }
00133
00134 namespace detail
00135 {
00136     template <std::floating_point T>
00137     bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
00138     {
00139         assert(false && "Not implemented yet");
00140         return false;
00141     }
00142

```

```

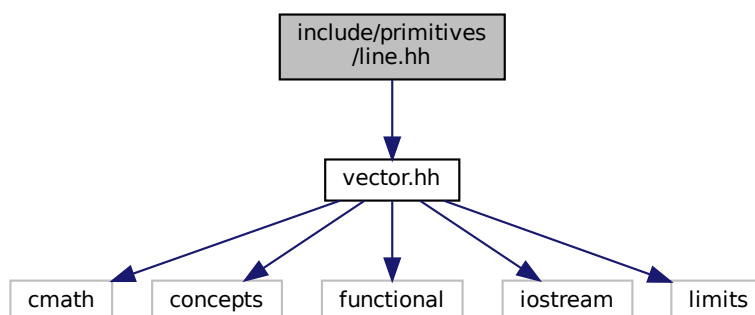
00145 }
00146
00147 template <std::floating_point T>
00148 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
00149 {
00150     // All this function is HARDCODE
00151     // TODO:
00152     // 1) make it more beautiful
00153     // 2) add handling such case: all points of triangle has signed dist to plane of another
00154     // triangle of same sign
00155
00156     auto pl1 = Plane<T>::getBy3Points(tr1[0], tr1[1], tr1[2]);
00157     auto pl2 = Plane<T>::getBy3Points(tr2[0], tr2[1], tr2[2]);
00158
00159     auto l = std::get<Line<T>>(intersect(pl1, pl2));
00160
00161     /* Project the triangle vertices onto line */
00162     std::array<T, 3> vert1, vert2;
00163     for (size_t i = 0; i < 3; ++i)
00164     {
00165         vert1[i] = dot(l.dir(), tr1[i] - l.org());
00166         vert2[i] = dot(l.dir(), tr2[i] - l.org());
00167     }
00168
00169     std::array<T, 3> sdist1, sdist2;
00170     for (size_t i = 0; i < 3; ++i)
00171     {
00172         sdist1[i] = distance(pl2, tr1[i]);
00173         sdist2[i] = distance(pl1, tr2[i]);
00174     }
00175
00176     std::array<T, 3> sign1, sign2;
00177     for (size_t i = 0; i < 3; ++i)
00178     {
00179         sign1[i] = sdist1[i] * sdist1[(i + 1) % 3];
00180         sign2[i] = sdist2[i] * sdist2[(i + 1) % 3];
00181     }
00182
00183     size_t rogue1 = 0;
00184     size_t rogue2 = 0;
00185     for (size_t i = 0; i < 3; ++i)
00186     {
00187         if (sign1[i] >= 0)
00188             rogue1 = (i + 2) % 3;
00189
00190         if (sign2[i] >= 0)
00191             rogue2 = (i + 2) % 3;
00192     }
00193
00194     std::vector<T> params1, params2;
00195     std::array<size_t, 2> arr1{(rogue1 + 1) % 3, (rogue1 + 2) % 3};
00196     std::array<size_t, 2> arr2{(rogue2 + 1) % 3, (rogue2 + 2) % 3};
00197
00198     for (size_t i : arr1)
00199         params1.push_back(vert1[i] + (vert1[rogue1] - vert1[i]) * sdist1[i] /
00200                             (sdist1[i] - sdist1[rogue1]));
00201
00202     for (size_t i : arr2)
00203         params2.push_back(vert2[i] + (vert2[rogue2] - vert2[i]) * sdist2[i] /
00204                             (sdist2[i] - sdist2[rogue2]));
00205
00206     std::sort(params1.begin(), params1.end());
00207     std::sort(params2.begin(), params2.end());
00208
00209     return isOverlap(params1[0], params1[1], params2[0], params2[1]);
00210 }
00211
00212 template <std::floating_point T>
00213 bool isOverlap(const T &params10, const T &params11, const T &params20, const T &params21)
00214 {
00215     return (params20 <= params11) && (params21 >= params10);
00216 }
00217
00218 template <std::forward_iterator It>
00219 bool isSameSign(It begin, It end)
00220 {
00221     auto cur = begin;
00222     auto prev = begin;
00223
00224     for (++cur; cur != end; ++cur)
00225         if ((*cur) * (*prev) < 0)
00226             return false;
00227
00228     return true;
00229 }
00230
00231 template <std::floating_point T>

```

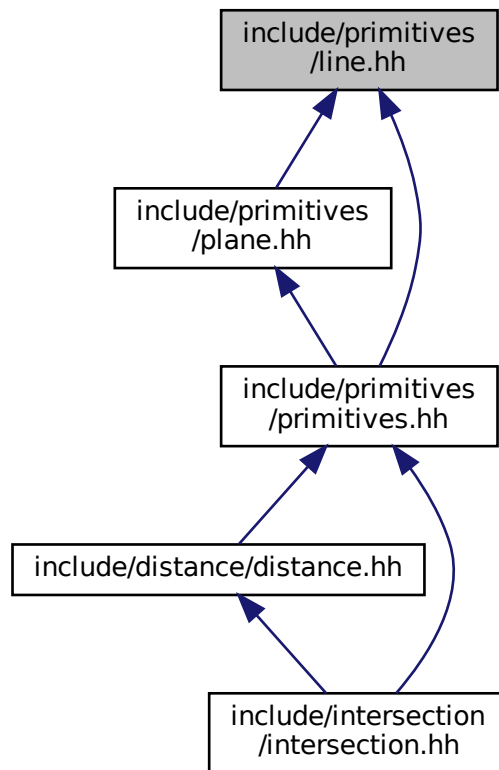
```
00232 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
00233 {
00234     std::array<T, 3> sdist{};
00235     for (size_t i = 0; i < 3; ++i)
00236         sdist[i] = distance(pl, tr[i]);
00237
00238     if (detail::isSameSign(sdist.begin(), sdist.end()))
00239         return false;
00240
00241     return true;
00242 }
00243
00244 } // namespace detail
00245 } // namespace geom
00246
00247 #endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__
```

6.5 include/primitives/line.hh File Reference

#include "vector.hh"
Include dependency graph for line.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class [geom::Line< T >](#)
[Line](#) class implementation.

Namespaces

- [geom](#)
[line.hh](#) [Line](#) class implementation

Functions

- template<std::floating_point T>
std::ostream & [geom::operator<<](#) (std::ostream &ost, const Line< T > &line)
[Line](#) print operator.
- template<std::floating_point T>
bool [geom::operator==](#) (const Line< T > &lhs, const Line< T > &rhs)
[Line](#) equality operator.

6.6 line.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_LINE_HH__
00002 #define __INCLUDE_PRIMITIVES_LINE_HH__
00003
00004 #include "vector.hh"
00005
00006 /**
00007  * @brief line.hh
00008  * Line class implementation
00009  */
00010
00011 namespace geom
00012 {
00013
00014 /**
00015  * @class Line
00016  * @brief Line class implementation
00017  *
00018  * @tparam T - floating point type of coordinates
00019  */
00020 template <std::floating_point T>
00021 class Line final
00022 {
00023 private:
00024     /**
00025      * @brief Origin and direction vectors
00026      */
00027     Vector<T> org_{}, dir_{};
00028
00029 public:
00030     /**
00031      * @brief Construct a new Line object
00032      *
00033      * @param[in] org origin vector
00034      * @param[in] dir direction vector
00035      */
00036     Line(const Vector<T> &org, const Vector<T> &dir);
00037
00038     /**
00039      * @brief Getter for origin vector
00040      *
00041      * @return const Vector<T>& const reference to origin vector
00042      */
00043     const Vector<T> &org() const;
00044
00045     /**
00046      * @brief Getter for direction vector
00047      *
00048      * @return const Vector<T>& const reference to direction vector
00049      */
00050     const Vector<T> &dir() const;
00051
00052     /**
00053      * @brief Checks is point belongs to line
00054      *
00055      * @param[in] point const reference to point vector
00056      * @return true if point belongs to line
00057      * @return false if point doesn't belong to line
00058      */
00059     bool belongs(const Vector<T> &point) const;
00060
00061     /**
00062      * @brief Checks is *this equals to another line
00063      *
00064      * @param[in] line const reference to another line
00065      * @return true if lines are equal
00066      * @return false if lines are not equal
00067      */
00068     bool isEqual(const Line &line) const;
00069
00070     /**
00071      * @brief Get line by 2 points
00072      *
00073      * @param[in] p1 1st point
00074      * @param[in] p2 2nd point
00075      * @return Line passing through two points
00076      */
00077     static Line getBy2Points(const Vector<T> &p1, const Vector<T> &p2);
00078 };
00079
00080 /**
00081  * @brief Line print operator
00082  *
00083  * @tparam T - floating point type of coordinates
00084  * @param[in, out] ost output stream
00085  * @param[in] line Line to print

```

```

00086 * @return std::ostream& modified ostream instance
00087 */
00088 template <std::floating_point T>
00089 std::ostream &operator<<(std::ostream &ost, const Line<T> &line)
00090 {
00091     ost << line.org() << " + " << line.dir() << " * t";
00092     return ost;
00093 }
00094
00095 /**
00096 * @brief Line equality operator
00097 *
00098 * @tparam T - floating point type of coordinates
00099 * @param[in] lhs 1st line
00100 * @param[in] rhs 2nd line
00101 * @return true if lines are equal
00102 * @return false if lines are not equal
00103 */
00104 template <std::floating_point T>
00105 bool operator==(const Line<T> &lhs, const Line<T> &rhs)
00106 {
00107     return lhs.isEqual(rhs);
00108 }
00109
00110 template <std::floating_point T>
00111 Line<T>::Line(const Vector<T> &org, const Vector<T> &dir) : org_{org}, dir_{dir}
00112 {
00113     if (dir_ == Vector<T>{0})
00114         throw std::logic_error{"Direction vector equals zero."};
00115 }
00116
00117 template <std::floating_point T>
00118 const Vector<T> &Line<T>::org() const
00119 {
00120     return org_;
00121 }
00122
00123 template <std::floating_point T>
00124 const Vector<T> &Line<T>::dir() const
00125 {
00126     return dir_;
00127 }
00128
00129 template <std::floating_point T>
00130 bool Line<T>::belongs(const Vector<T> &point) const
00131 {
00132     return dir_.cross(point - org_) == Vector<T>{0};
00133 }
00134
00135 template <std::floating_point T>
00136 bool Line<T>::isEqual(const Line<T> &line) const
00137 {
00138     return belongs(line.org_) && dir_.isPar(line.dir_);
00139 }
00140
00141 template <std::floating_point T>
00142 Line<T> Line<T>::getBy2Points(const Vector<T> &p1, const Vector<T> &p2)
00143 {
00144     return Line<T>{p1, p2 - p1};
00145 }
00146
00147 } // namespace geom
00148
00149 #endif // __INCLUDE_PRIMITIVES_LINE_HH__

```

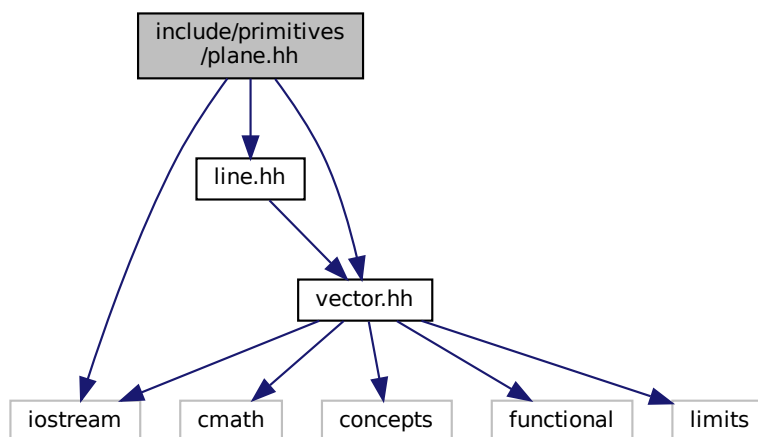
6.7 include/primitives/plane.hh File Reference

```

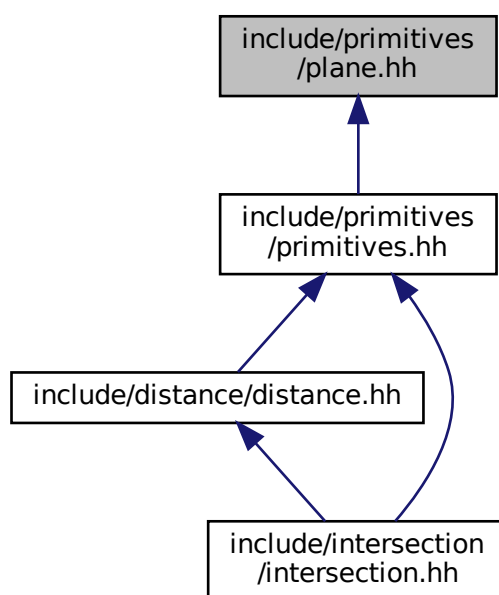
#include <iostream>
#include "line.hh"
#include "vector.hh"

```


Include dependency graph for plane.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class `geom::Plane< T >`
Plane class realization.

Namespaces

- [geom](#)
line.hh Line class implementation

Functions

- `template<std::floating_point T>`
`bool geom::operator== (const Plane< T > &lhs, const Plane< T > &rhs)`
Plane equality operator.
- `template<std::floating_point T>`
`std::ostream & geom::operator<< (std::ostream &ost, const Plane< T > &pl)`
Plane print operator.

6.8 plane.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_PLANE_HH__
00002 #define __INCLUDE_PRIMITIVES_PLANE_HH__
00003
00004 #include <iostream>
00005
00006 #include "line.hh"
00007 #include "vector.hh"
00008
00009 /**
00010  * @brief
00011  * Plane class implementation
00012  */
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @class Plane
00019  * @brief Plane class realization
00020  *
00021  * @tparam T - floating point type of coordinates
00022  */
00023 template <std::floating_point T>
00024 class Plane final
00025 {
00026 private:
00027     /**
00028      * @brief Normal vector, length equals to 1
00029      */
00030     Vector<T> norm_{};
00031
00032     /**
00033      * @brief Distance from zero to plane
00034      */
00035     T dist_{};
00036
00037     /**
00038      * @brief Construct a new Plane object from normal vector and distance
00039      *
00040      * @param[in] norm normal vector
00041      * @param[in] dist distance from plane to zero
00042      */
00043     Plane(const Vector<T> &norm, T dist);
00044
00045 public:
00046     /**
00047      * @brief Getter for distance
00048      *
00049      * @return T value of distance
00050      */
00051     T dist() const;
00052
00053     /**
00054      * @brief Getter for normal vector
00055      *
00056      * @return const Vector<T>& const reference to normal vector
00057      */
00058     const Vector<T> &norm() const;

```

```

00059
00060 /**
00061  * @brief Checks if point belongs to plane
00062  *
00063  * @param[in] point const reference to point vector
00064  * @return true if point belongs to plane
00065  * @return false if point doesn't belong to plane
00066  */
00067 bool belongs(const Vector<T> &point) const;
00068
00069 /**
00070  * @brief Checks if line belongs to plane
00071  *
00072  * @param[in] line const reference to line
00073  * @return true if line belongs to plane
00074  * @return false if line doesn't belong to plane
00075  */
00076 bool belongs(const Line<T> &line) const;
00077
00078 /**
00079  * @brief Checks if *this equals to another plane
00080  *
00081  * @param[in] rhs const reference to another plane
00082  * @return true if planes are equal
00083  * @return false if planes are not equal
00084  */
00085 bool isEqual(const Plane &rhs) const;
00086
00087 /**
00088  * @brief Checks if *this is parallel to another plane
00089  *
00090  * @param[in] rhs const reference to another plane
00091  * @return true if planes are parallel
00092  * @return false if planes are not parallel
00093  */
00094 bool isPar(const Plane &rhs) const;
00095
00096 /**
00097  * @brief Get plane by 3 points
00098  *
00099  * @param[in] pt1 1st point
00100  * @param[in] pt2 2nd point
00101  * @param[in] pt3 3rd point
00102  * @return Plane passing through three points
00103  */
00104 static Plane getBy3Points(const Vector<T> &pt1, const Vector<T> &pt2,
00105                          const Vector<T> &pt3);
00106
00107 /**
00108  * @brief Get plane from parametric plane equation
00109  *
00110  * @param[in] org origin vector
00111  * @param[in] dir1 1st direction vector
00112  * @param[in] dir2 2nd direction vector
00113  * @return Plane
00114  */
00115 static Plane getParametric(const Vector<T> &org, const Vector<T> &dir1,
00116                          const Vector<T> &dir2);
00117
00118 /**
00119  * @brief Get plane from normal point plane equation
00120  *
00121  * @param[in] norm normal vector
00122  * @param[in] point point lying on the plane
00123  * @return Plane
00124  */
00125 static Plane getNormalPoint(const Vector<T> &norm, const Vector<T> &point);
00126
00127 /**
00128  * @brief Get plane from normal const plane equation
00129  *
00130  * @param[in] norm normal vector
00131  * @param[in] constant distance
00132  * @return Plane
00133  */
00134 static Plane getNormalDist(const Vector<T> &norm, T constant);
00135 };
00136
00137 /**
00138  * @brief Plane equality operator
00139  *
00140  * @tparam T - floating point type of coordinates
00141  * @param[in] lhs 1st plane
00142  * @param[in] rhs 2nd plane
00143  * @return true if planes are equal
00144  * @return false if planes are not equal
00145  */

```

```

00146 template <std::floating_point T>
00147 bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
00148 {
00149     return lhs.isEqual(rhs);
00150 }
00151
00152 /**
00153  * @brief Plane print operator
00154  *
00155  * @tparam T - floating point type of coordinates
00156  * @param[in, out] ost output stream
00157  * @param[in] pl plane to print
00158  * @return std::ostream& modified ostream instance
00159  */
00160 template <std::floating_point T>
00161 std::ostream &operator<<(std::ostream &ost, const Plane<T> &pl)
00162 {
00163     ost << pl.norm() << " * X = " << pl.dist();
00164     return ost;
00165 }
00166
00167 template <std::floating_point T>
00168 Plane<T>::Plane(const Vector<T> &norm, T dist) : norm_(norm), dist_(dist)
00169 {
00170     if (norm == Vector<T>{0})
00171         throw std::logic_error{"normal vector equals to zero"};
00172 }
00173
00174 template <std::floating_point T>
00175 T Plane<T>::dist() const
00176 {
00177     return dist_;
00178 }
00179
00180 template <std::floating_point T>
00181 const Vector<T> &Plane<T>::norm() const
00182 {
00183     return norm_;
00184 }
00185
00186 template <std::floating_point T>
00187 bool Plane<T>::belongs(const Vector<T> &pt) const
00188 {
00189     return Vector<T>::isNumEq(norm_.dot(pt), dist_);
00190 }
00191
00192 template <std::floating_point T>
00193 bool Plane<T>::belongs(const Line<T> &line) const
00194 {
00195     return norm_.isPerp(line.dir()) && belongs(line.org());
00196 }
00197
00198 template <std::floating_point T>
00199 bool Plane<T>::isEqual(const Plane &rhs) const
00200 {
00201     return (norm_ * dist_ == rhs.norm_ * rhs.dist_) && (norm_.isPar(rhs.norm_));
00202 }
00203
00204 template <std::floating_point T>
00205 bool Plane<T>::isPar(const Plane &rhs) const
00206 {
00207     return norm_.isPar(rhs.norm_);
00208 }
00209
00210 template <std::floating_point T>
00211 Plane<T> Plane<T>::getBy3Points(const Vector<T> &pt1, const Vector<T> &pt2,
00212                                const Vector<T> &pt3)
00213 {
00214     return getParametric(pt1, pt2 - pt1, pt3 - pt1);
00215 }
00216
00217 template <std::floating_point T>
00218 Plane<T> Plane<T>::getParametric(const Vector<T> &org, const Vector<T> &dir1,
00219                                  const Vector<T> &dir2)
00220 {
00221     auto norm = dir1.cross(dir2);
00222     return getNormalPoint(norm, org);
00223 }
00224
00225 template <std::floating_point T>
00226 Plane<T> Plane<T>::getNormalPoint(const Vector<T> &norm, const Vector<T> &pt)
00227 {
00228     auto normalized = norm.normalized();
00229     return Plane{normalized, normalized.dot(pt)};
00230 }
00231
00232 template <std::floating_point T>

```

```

00233 Plane<T> Plane<T>::getNormalDist(const Vector<T> &norm, T dist)
00234 {
00235     auto normalized = norm.normalized();
00236     return Plane{normalized, dist};
00237 }
00238
00239 } // namespace geom
00240
00241 #endif // __INCLUDE_PRIMITIVES_PLANE_HH__

```

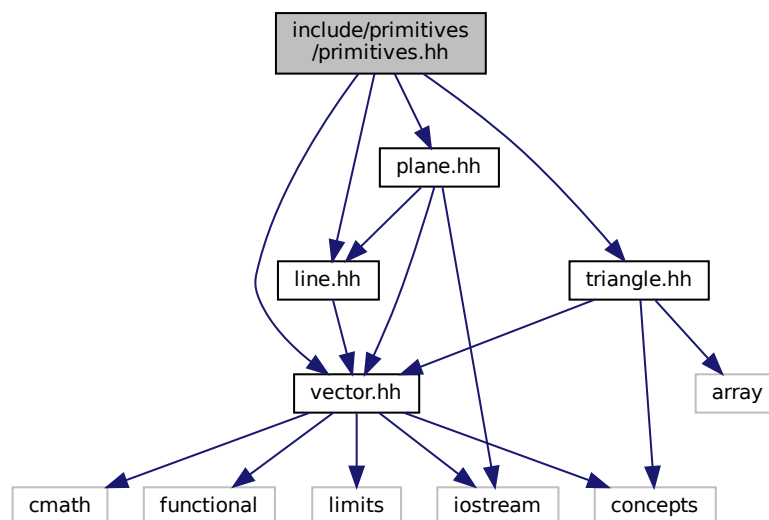
6.9 include/primitives/primitives.hh File Reference

```

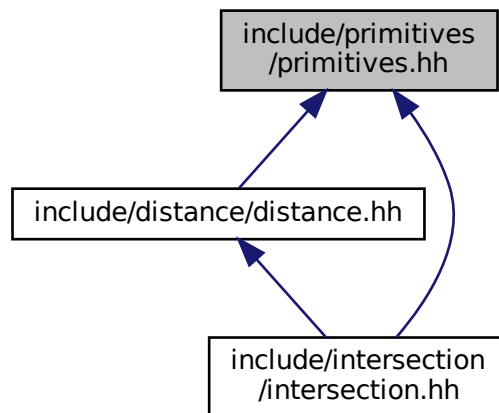
#include "line.hh"
#include "plane.hh"
#include "triangle.hh"
#include "vector.hh"

```

Include dependency graph for primitives.hh:



This graph shows which files directly or indirectly include this file:



6.10 primitives.hh

```

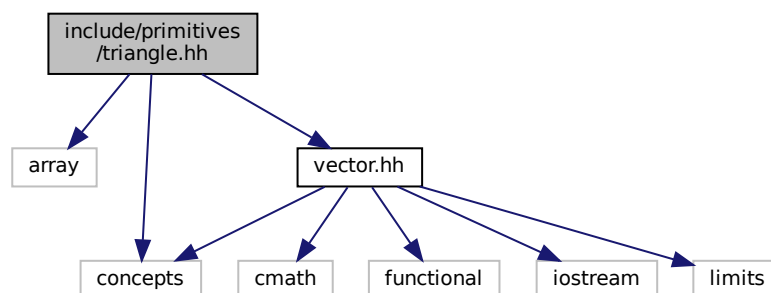
00001 #ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00002 #define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00003
00004 #include "line.hh"
00005 #include "plane.hh"
00006 #include "triangle.hh"
00007 #include "vector.hh"
00008
00009 #endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
  
```

6.11 include/primitives/triangle.hh File Reference

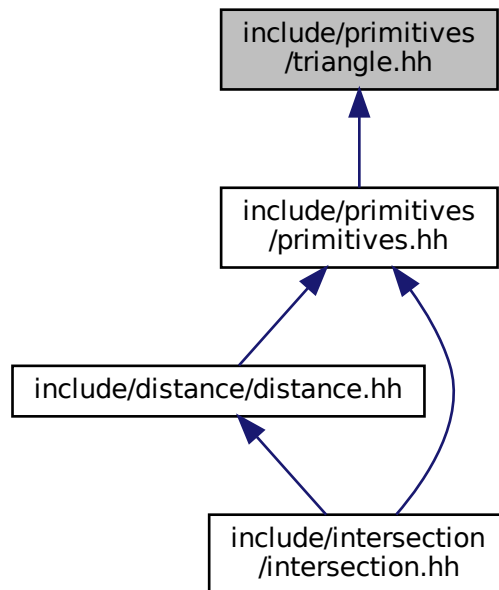
```

#include <array>
#include <concepts>
#include "vector.hh"
  
```

Include dependency graph for triangle.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class `geom::Triangle< T >`
Triangle class implementation.

Namespaces

- `geom`
line.hh *Line* class implementation

Functions

- `template<std::floating_point T>`
`std::ostream & geom::operator<< (std::ostream &ost, const Triangle< T > &tr)`
Triangle print operator.

6.12 triangle.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00002 #define __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00003
00004 #include <array>
00005 #include <concepts>
00006
00007 #include "vector.hh"

```

```

00008
00009 /**
00010  * @brief triangle.hh
00011  * Triangle class implementation
00012  */
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @class Triangle
00019  * @brief Triangle class implementation
00020  *
00021  * @tparam T - floating point type of coordinates
00022  */
00023 template <std::floating_point T>
00024 class Triangle final
00025 {
00026 private:
00027     /**
00028      * @brief Vertices of triangle
00029      */
00030     std::array<Vector<T>, 3> vertices_;
00031
00032 public:
00033     /**
00034      * @brief Construct a new Triangle object from 3 points
00035      *
00036      * @param[in] p1 1st point
00037      * @param[in] p2 2nd point
00038      * @param[in] p3 3rd point
00039      */
00040     Triangle(const Vector<T> &p1, const Vector<T> &p2, const Vector<T> &p3);
00041
00042     /**
00043      * @brief Overloaded operator[] to get access to vertices
00044      *
00045      * @param[in] idx index of vertex
00046      * @return const Vector<T>& const reference to vertex
00047      */
00048     const Vector<T> &operator[](std::size_t idx) const;
00049 };
00050
00051 /**
00052  * @brief Triangle print operator
00053  *
00054  * @tparam T - floating point type of coordinates
00055  * @param[in, out] ost output stream
00056  * @param[in] tr Triangle to print
00057  * @return std::ostream& modified ostream instance
00058  */
00059 template <std::floating_point T>
00060 std::ostream &operator<<(std::ostream &ost, const Triangle<T> &tr)
00061 {
00062     ost << "Triangle: {";
00063     for (size_t i : {0, 1, 2})
00064         ost << tr[i] << (i == 2 ? "" : ", ");
00065
00066     ost << "}";
00067
00068     return ost;
00069 }
00070
00071 template <std::floating_point T>
00072 Triangle<T>::Triangle(const Vector<T> &p1, const Vector<T> &p2, const Vector<T> &p3)
00073     : vertices_{p1, p2, p3}
00074 {}
00075
00076 template <std::floating_point T>
00077 const Vector<T> &Triangle<T>::operator[](std::size_t idx) const
00078 {
00079     return vertices_[idx % 3];
00080 }
00081
00082 } // namespace geom
00083
00084 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__

```

6.13 include/primitives/vector.hh File Reference

```

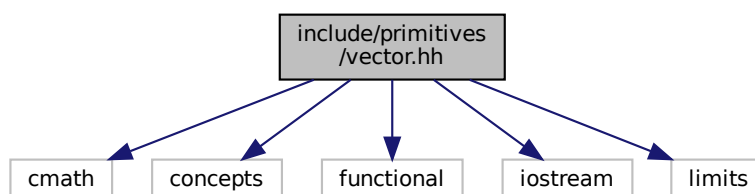
#include <cmath>
#include <concepts>

```

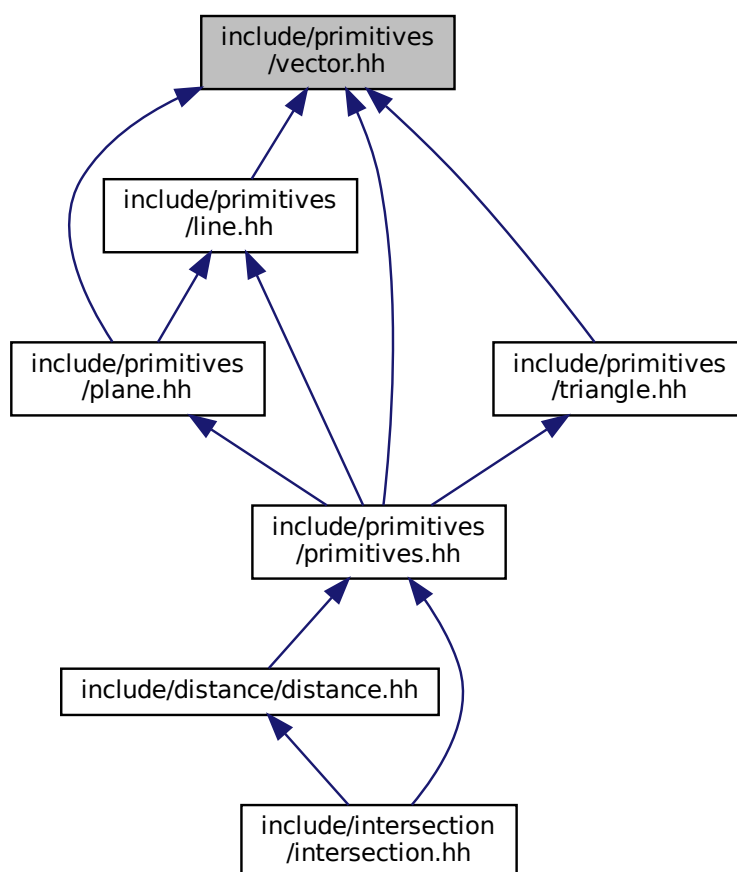


```
#include <functional>
#include <iostream>
#include <limits>
```

Include dependency graph for vector.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class `geom::Vector< T >`

Vector class realization.

Namespaces

- [geom](#)
line.hh Line class implementation

Typedefs

- using [geom::VectorD](#) = Vector< double >
- using [geom::VectorF](#) = Vector< float >

Functions

- template<std::floating_point T>
Vector< T > [geom::operator+](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Overloaded + operator.
- template<std::floating_point T>
Vector< T > [geom::operator-](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Overloaded - operator.
- template<Number nT, std::floating_point T>
Vector< T > [geom::operator*](#) (const nT &val, const Vector< T > &rhs)
Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
Vector< T > [geom::operator*](#) (const Vector< T > &lhs, const nT &val)
Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
Vector< T > [geom::operator/](#) (const Vector< T > &lhs, const nT &val)
Overloaded divide by value operator.
- template<std::floating_point T>
T [geom::dot](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Dot product function.
- template<std::floating_point T>
Vector< T > [geom::cross](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Cross product function.
- template<std::floating_point T>
bool [geom::operator==](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Vector equality operator.
- template<std::floating_point T>
bool [geom::operator!=](#) (const Vector< T > &lhs, const Vector< T > &rhs)
Vector inequality operator.
- template<std::floating_point T>
std::ostream & [geom::operator<<](#) (std::ostream &ost, const Vector< T > &vec)
Vector print operator.

Variables

- template<class T >
concept [geom::Number](#) = std::is_floating_point_v<T> || std::is_integral_v<T>
Useful concept which represents floating point and integral types.

6.13.1 Detailed Description

Vector class implementation

Definition in file [vector.hh](#).

6.14 vector.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_VECTOR_HH__
00002 #define __INCLUDE_PRIMITIVES_VECTOR_HH__
00003
00004 #include <cmath>
00005 #include <concepts>
00006 #include <functional>
00007 #include <iostream>
00008 #include <limits>
00009
00010 /**
00011  * @file vector.hh
00012  * Vector class implementation
00013  */
00014
00015 namespace geom
00016 {
00017
00018 /**
00019  * @concept Number
00020  * @brief Useful concept which represents floating point and integral types
00021  *
00022  * @tparam T
00023  */
00024 template <class T>
00025 concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>;
00026
00027 /**
00028  * @class Vector
00029  * @brief Vector class realization
00030  *
00031  * @tparam T - floating point type of coordinates
00032  */
00033 template <std::floating_point T>
00034 struct Vector final
00035 {
00036 private:
00037     /**
00038      * @brief Threshold static variable for numbers comparision
00039      */
00040     static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00041
00042 public:
00043     /**
00044      * @brief Vector coordinates
00045      */
00046     T x{}, y{}, z{};
00047
00048     /**
00049      * @brief Construct a new Vector object from 3 coordinates
00050      *
00051      * @param[in] coordX x coordinate
00052      * @param[in] coordY y coordinate
00053      * @param[in] coordZ z coordinate
00054      */
00055     Vector(T coordX, T coordY, T coordZ) : x(coordX), y(coordY), z(coordZ)
00056     {}
00057
00058     /**
00059      * @brief Construct a new Vector object with equals coordinates
00060      *
00061      * @param[in] coordX coordinate (default to {})
00062      */
00063     explicit Vector(T coordX = {}) : Vector(coordX, coordX, coordX)
00064     {}
00065
00066     /**
00067      * @brief Overloaded += operator
00068      * Increments vector coordinates by corresponding coordinates of vec
00069      * @param[in] vec vector to incremented with
00070      * @return Vector& reference to current instance
00071      */
00072     Vector &operator+=(const Vector &vec);

```

```

00073
00074 /**
00075  * @brief Overloaded -= operator
00076  * Decrements vector coordinates by corresponding coordinates of vec
00077  * @param[in] vec vector to decremented with
00078  * @return Vector& reference to current instance
00079  */
00080 Vector &operator--(const Vector &vec);
00081
00082 /**
00083  * @brief Unary - operator
00084  *
00085  * @return Vector negated Vector instance
00086  */
00087 Vector operator-() const;
00088
00089 /**
00090  * @brief Overloaded *= by number operator
00091  *
00092  * @tparam nType numeric type of value to multiply by
00093  * @param[in] val value to multiply by
00094  * @return Vector& reference to vector instance
00095  */
00096 template <Number nType>
00097 Vector &operator*=(nType val);
00098
00099 /**
00100  * @brief Overloaded /= by number operator
00101  *
00102  * @tparam nType numeric type of value to divide by
00103  * @param[in] val value to divide by
00104  * @return Vector& reference to vector instance
00105  *
00106  * @warning Does not check if val equals 0
00107  */
00108 template <Number nType>
00109 Vector &operator/=(nType val);
00110
00111 /**
00112  * @brief Dot product function
00113  *
00114  * @param rhs vector to dot product with
00115  * @return T dot product of two vectors
00116  */
00117 T dot(const Vector &rhs) const;
00118
00119 /**
00120  * @brief Cross product function
00121  *
00122  * @param rhs vector to cross product with
00123  * @return Vector cross product of two vectors
00124  */
00125 Vector cross(const Vector &rhs) const;
00126
00127 /**
00128  * @brief Calculate squared length of a vector function
00129  *
00130  * @return T length^2
00131  */
00132 T length2() const;
00133
00134 /**
00135  * @brief Calculate length of a vector function
00136  *
00137  * @return T length
00138  */
00139 T length() const;
00140
00141 /**
00142  * @brief Get normalized vector function
00143  *
00144  * @return Vector normalized vector
00145  */
00146 Vector normalized() const;
00147
00148 /**
00149  * @brief Normalize vector function
00150  *
00151  * @return Vector& reference to instance
00152  */
00153 Vector &normalize();
00154
00155 /**
00156  * @brief Overloaded operator [] (non-const version)
00157  * To get access to coordinates
00158  * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00159  * @return T& reference to coordinate value

```

```

00160     *
00161     * @note Coordinates calculated by mod 3
00162     */
00163     T &operator[](size_t i);
00164
00165     /**
00166     * @brief Overloaded operator [] (const version)
00167     * To get access to coordinates
00168     * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00169     * @return T coordinate value
00170     *
00171     * @note Coordinates calculated by mod 3
00172     */
00173     T operator[](size_t i) const;
00174
00175     /**
00176     * @brief Check if vector is parallel to another
00177     *
00178     * @param[in] rhs vector to check parallelism with
00179     * @return true if vector is parallel
00180     * @return false otherwise
00181     */
00182     bool isPar(const Vector &rhs) const;
00183
00184     /**
00185     * @brief Check if vector is perpendicular to another
00186     *
00187     * @param[in] rhs vector to check perpendicularity with
00188     * @return true if vector is perpendicular
00189     * @return false otherwise
00190     */
00191     bool isPerp(const Vector &rhs) const;
00192
00193     /**
00194     * @brief Check if vector is equal to another
00195     *
00196     * @param[in] rhs vector to check equality with
00197     * @return true if vector is equal
00198     * @return false otherwise
00199     *
00200     * @note Equality check performs using isNumEq(T lhs, T rhs) function
00201     */
00202     bool isEqual(const Vector &rhs) const;
00203
00204     /**
00205     * @brief Check equality (with threshold) of two floating point numbers function
00206     *
00207     * @param[in] lhs first number
00208     * @param[in] rhs second number
00209     * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
00210     * @return false otherwise
00211     *
00212     * @note Threshold defined by threshold_ static member
00213     */
00214     static bool isNumEq(T lhs, T rhs);
00215
00216     /**
00217     * @brief Set new threshold value
00218     *
00219     * @param[in] thres value to set
00220     */
00221     static void setThreshold(T thres);
00222
00223     /**
00224     * @brief Get current threshold value
00225     */
00226     static void getThreshold();
00227
00228     /**
00229     * @brief Set threshold to default value
00230     * @note default value equals float point epsilon
00231     */
00232     static void setDefThreshold();
00233 };
00234
00235 /**
00236 * @brief Overloaded + operator
00237 *
00238 * @tparam T vector template parameter
00239 * @param[in] lhs first vector
00240 * @param[in] rhs second vector
00241 * @return Vector<T> sum of two vectors
00242 */
00243 template <std::floating_point T>
00244 Vector<T> operator+(const Vector<T> &lhs, const Vector<T> &rhs)
00245 {
00246     Vector<T> res{lhs};

```

```

00247     res += rhs;
00248     return res;
00249 }
00250
00251 /**
00252  * @brief Overloaded - operator
00253  *
00254  * @tparam T vector template parameter
00255  * @param[in] lhs first vector
00256  * @param[in] rhs second vector
00257  * @return Vector<T> res of two vectors
00258  */
00259 template <std::floating_point T>
00260 Vector<T> operator-(const Vector<T> &lhs, const Vector<T> &rhs)
00261 {
00262     Vector<T> res{lhs};
00263     res -= rhs;
00264     return res;
00265 }
00266
00267 /**
00268  * @brief Overloaded multiple by value operator
00269  *
00270  * @tparam nT type of value to multiply by
00271  * @tparam T vector template parameter
00272  * @param[in] val value to multiply by
00273  * @param[in] rhs vector to multiply by value
00274  * @return Vector<T> result vector
00275  */
00276 template <Number nT, std::floating_point T>
00277 Vector<T> operator*(const nT &val, const Vector<T> &rhs)
00278 {
00279     Vector<T> res{rhs};
00280     res *= val;
00281     return res;
00282 }
00283
00284 /**
00285  * @brief Overloaded multiple by value operator
00286  *
00287  * @tparam nT type of value to multiply by
00288  * @tparam T vector template parameter
00289  * @param[in] val value to multiply by
00290  * @param[in] lhs vector to multiply by value
00291  * @return Vector<T> result vector
00292  */
00293 template <Number nT, std::floating_point T>
00294 Vector<T> operator*(const Vector<T> &lhs, const nT &val)
00295 {
00296     Vector<T> res{lhs};
00297     res *= val;
00298     return res;
00299 }
00300
00301 /**
00302  * @brief Overloaded divide by value operator
00303  *
00304  * @tparam nT type of value to divide by
00305  * @tparam T vector template parameter
00306  * @param[in] val value to divide by
00307  * @param[in] lhs vector to divide by value
00308  * @return Vector<T> result vector
00309  */
00310 template <Number nT, std::floating_point T>
00311 Vector<T> operator/(const Vector<T> &lhs, const nT &val)
00312 {
00313     Vector<T> res{lhs};
00314     res /= val;
00315     return res;
00316 }
00317
00318 /**
00319  * @brief Dot product function
00320  *
00321  * @tparam T vector template parameter
00322  * @param[in] lhs first vector
00323  * @param[in] rhs second vector
00324  * @return T dot production
00325  */
00326 template <std::floating_point T>
00327 T dot(const Vector<T> &lhs, const Vector<T> &rhs)
00328 {
00329     return lhs.dot(rhs);
00330 }
00331
00332 /**
00333  * @brief Cross product function

```

```

00334 *
00335 * @tparam T vector template parameter
00336 * @param[in] lhs first vector
00337 * @param[in] rhs second vector
00338 * @return T cross production
00339 */
00340 template <std::floating_point T>
00341 Vector<T> cross(const Vector<T> &lhs, const Vector<T> &rhs)
00342 {
00343     return lhs.cross(rhs);
00344 }
00345
00346 /**
00347 * @brief Vector equality operator
00348 *
00349 * @tparam T vector template parameter
00350 * @param[in] lhs first vector
00351 * @param[in] rhs second vector
00352 * @return true if vectors are equal
00353 * @return false otherwise
00354 */
00355 template <std::floating_point T>
00356 bool operator==(const Vector<T> &lhs, const Vector<T> &rhs)
00357 {
00358     return lhs.isEqual(rhs);
00359 }
00360
00361 /**
00362 * @brief Vector inequality operator
00363 *
00364 * @tparam T vector template parameter
00365 * @param[in] lhs first vector
00366 * @param[in] rhs second vector
00367 * @return true if vectors are not equal
00368 * @return false otherwise
00369 */
00370 template <std::floating_point T>
00371 bool operator!=(const Vector<T> &lhs, const Vector<T> &rhs)
00372 {
00373     return !(lhs == rhs);
00374 }
00375
00376 /**
00377 * @brief Vector print operator
00378 *
00379 * @tparam T vector template parameter
00380 * @param[in, out] ost output stream
00381 * @param[in] vec vector to print
00382 * @return std::ostream& modified stream instance
00383 */
00384 template <std::floating_point T>
00385 std::ostream &operator<<(std::ostream &ost, const Vector<T> &vec)
00386 {
00387     ost << "(" << vec.x << ", " << vec.y << ", " << vec.z << ")";
00388     return ost;
00389 }
00390
00391 using VectorD = Vector<double>;
00392 using VectorF = Vector<float>;
00393
00394 template <std::floating_point T>
00395 Vector<T> &Vector<T>::operator+=(const Vector &vec)
00396 {
00397     x += vec.x;
00398     y += vec.y;
00399     z += vec.z;
00400
00401     return *this;
00402 }
00403
00404 template <std::floating_point T>
00405 Vector<T> &Vector<T>::operator-=(const Vector &vec)
00406 {
00407     x -= vec.x;
00408     y -= vec.y;
00409     z -= vec.z;
00410
00411     return *this;
00412 }
00413
00414 template <std::floating_point T>
00415 Vector<T> Vector<T>::operator-() const
00416 {
00417     return Vector{-x, -y, -z};
00418 }
00419
00420 template <std::floating_point T>

```

```

00421 template <Number nType>
00422 Vector<T> &Vector<T>::operator*=(nType val)
00423 {
00424     x *= val;
00425     y *= val;
00426     z *= val;
00427
00428     return *this;
00429 }
00430
00431 template <std::floating_point T>
00432 template <Number nType>
00433 Vector<T> &Vector<T>::operator/=(nType val)
00434 {
00435     x /= static_cast<T>(val);
00436     y /= static_cast<T>(val);
00437     z /= static_cast<T>(val);
00438
00439     return *this;
00440 }
00441
00442 template <std::floating_point T>
00443 T Vector<T>::dot(const Vector &rhs) const
00444 {
00445     return x * rhs.x + y * rhs.y + z * rhs.z;
00446 }
00447
00448 template <std::floating_point T>
00449 Vector<T> Vector<T>::cross(const Vector &rhs) const
00450 {
00451     return Vector{y * rhs.z - z * rhs.y, z * rhs.x - x * rhs.z, x * rhs.y - y * rhs.x};
00452 }
00453
00454 template <std::floating_point T>
00455 T Vector<T>::length2() const
00456 {
00457     return dot(*this);
00458 }
00459
00460 template <std::floating_point T>
00461 T Vector<T>::length() const
00462 {
00463     return std::sqrt(length2());
00464 }
00465
00466 template <std::floating_point T>
00467 Vector<T> Vector<T>::normalized() const
00468 {
00469     Vector res{*this};
00470     res.normalize();
00471     return res;
00472 }
00473
00474 template <std::floating_point T>
00475 Vector<T> &Vector<T>::normalize()
00476 {
00477     T len2 = length2();
00478     if (isNumEq(len2, 0) || isNumEq(len2, 1))
00479         return *this;
00480     return *this /= std::sqrt(len2);
00481 }
00482
00483 template <std::floating_point T>
00484 T &Vector<T>::operator[](size_t i)
00485 {
00486     switch (i % 3)
00487     {
00488     case 0:
00489         return x;
00490     case 1:
00491         return y;
00492     case 2:
00493         return z;
00494     default:
00495         throw std::logic_error{"Impossible case in operator[]\n"};
00496     }
00497 }
00498
00499 template <std::floating_point T>
00500 T Vector<T>::operator[](size_t i) const
00501 {
00502     switch (i % 3)
00503     {
00504     case 0:
00505         return x;
00506     case 1:
00507         return y;

```



```

00508     case 2:
00509         return z;
00510     default:
00511         throw std::logic_error{"Impossible case in operator[]\n"};
00512     }
00513 }
00514
00515 template <std::floating_point T>
00516 bool Vector<T>::isPar(const Vector &rhs) const
00517 {
00518     return cross(rhs).isEqual(Vector<T>{0});
00519 }
00520
00521 template <std::floating_point T>
00522 bool Vector<T>::isPerp(const Vector &rhs) const
00523 {
00524     return isNumEq(dot(rhs), 0);
00525 }
00526
00527 template <std::floating_point T>
00528 bool Vector<T>::isEqual(const Vector &rhs) const
00529 {
00530     return isNumEq(x, rhs.x) && isNumEq(y, rhs.y) && isNumEq(z, rhs.z);
00531 }
00532
00533 template <std::floating_point T>
00534 bool Vector<T>::isNumEq(T lhs, T rhs)
00535 {
00536     return std::abs(rhs - lhs) < threshold_;
00537 }
00538
00539 template <std::floating_point T>
00540 void Vector<T>::setThreshold(T thres)
00541 {
00542     threshold_ = thres;
00543 }
00544
00545 template <std::floating_point T>
00546 void Vector<T>::getThreshold()
00547 {
00548     return threshold_;
00549 }
00550
00551 template <std::floating_point T>
00552 void Vector<T>::setDefThreshold()
00553 {
00554     threshold_ = std::numeric_limits<T>::epsilon();
00555 }
00556
00557 } // namespace geom
00558
00559 #endif // __INCLUDE_PRIMITIVES_VECTOR_HH__

```

