

# Triangles

1.0.1

Generated by Doxygen 1.8.17



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
4.1 geom Namespace Reference	7
4.1.1 Detailed Description	9
4.1.2 Typedef Documentation	9
4.1.2.1 Vec2D	10
4.1.2.2 Vec2F	10
4.1.2.3 Vec3D	10
4.1.2.4 Vec3F	10
4.1.3 Function Documentation	10
4.1.3.1 distance()	10
4.1.3.2 isIntersect()	11
4.1.3.3 intersect()	12
4.1.3.4 operator<<() [1/5]	13
4.1.3.5 operator==( ) [1/4]	13
4.1.3.6 operator==( ) [2/4]	14
4.1.3.7 operator<<() [2/5]	15
4.1.3.8 operator<<() [3/5]	16
4.1.3.9 operator+() [1/2]	16
4.1.3.10 operator-() [1/2]	17
4.1.3.11 operator*() [1/4]	18
4.1.3.12 operator*() [2/4]	18
4.1.3.13 operator/() [1/2]	19
4.1.3.14 dot() [1/2]	19
4.1.3.15 operator==( ) [3/4]	20
4.1.3.16 operator!=( ) [1/2]	20
4.1.3.17 operator<<() [4/5]	21
4.1.3.18 operator+() [2/2]	22
4.1.3.19 operator-() [2/2]	22
4.1.3.20 operator*() [3/4]	23
4.1.3.21 operator*() [4/4]	23
4.1.3.22 operator/() [2/2]	24
4.1.3.23 dot() [2/2]	24
4.1.3.24 cross()	25
4.1.3.25 operator==( ) [4/4]	25

4.1.3.26 operator!=(()) [2/2]	26
4.1.3.27 operator<<() [5/5]	27
4.1.4 Variable Documentation	27
4.1.4.1 Number	27
4.2 geom::detail Namespace Reference	28
4.2.1 Typedef Documentation	28
4.2.1.1 Segment	28
4.2.1.2 Trian2	28
4.2.2 Function Documentation	29
4.2.2.1 isIntersect2D()	29
4.2.2.2 isIntersectMollerHaines()	29
4.2.2.3 helperMollerHaines()	29
4.2.2.4 isIntersectBothInvalid()	30
4.2.2.5 isIntersectValidInvalid()	30
4.2.2.6 isOverlap()	30
4.2.2.7 isSameSign()	30
4.2.2.8 isOnOneSide()	31
4.2.2.9 getTrian2()	31
4.2.2.10 isCounterClockwise()	31
4.2.2.11 computeInterval()	31
<b>5 Class Documentation</b>	<b>33</b>
5.1 geom::Line< T > Class Template Reference	33
5.1.1 Detailed Description	33
5.1.2 Constructor & Destructor Documentation	34
5.1.2.1 Line()	34
5.1.3 Member Function Documentation	34
5.1.3.1 org()	34
5.1.3.2 dir()	35
5.1.3.3 belongs()	35
5.1.3.4 isEqual()	35
5.1.3.5 getBy2Points()	36
5.2 geom::Plane< T > Class Template Reference	36
5.2.1 Detailed Description	37
5.2.2 Member Function Documentation	37
5.2.2.1 dist()	37
5.2.2.2 norm()	38
5.2.2.3 belongs() [1/2]	38
5.2.2.4 belongs() [2/2]	38
5.2.2.5 isEqual()	39
5.2.2.6 isPar()	39
5.2.2.7 getBy3Points()	40

5.2.2.8	getParametric()	40
5.2.2.9	getNormalPoint()	41
5.2.2.10	getNormalDist()	41
5.3	geom::Triangle< T > Class Template Reference	42
5.3.1	Detailed Description	42
5.3.2	Constructor & Destructor Documentation	42
5.3.2.1	Triangle()	43
5.3.3	Member Function Documentation	43
5.3.3.1	operator[]()	43
5.3.3.2	getPlane()	43
5.3.3.3	isValid()	44
5.4	geom::Vec2< T > Class Template Reference	44
5.4.1	Detailed Description	46
5.4.2	Constructor & Destructor Documentation	46
5.4.2.1	Vec2() [1/2]	46
5.4.2.2	Vec2() [2/2]	46
5.4.3	Member Function Documentation	47
5.4.3.1	operator+=(())	47
5.4.3.2	operator-=(())	47
5.4.3.3	operator-()	48
5.4.3.4	operator*=(()) [1/2]	48
5.4.3.5	operator/=(()) [1/2]	48
5.4.3.6	dot()	50
5.4.3.7	length2()	50
5.4.3.8	length()	51
5.4.3.9	getPerp()	51
5.4.3.10	normalized()	51
5.4.3.11	normalize()	52
5.4.3.12	operator[]() [1/2]	52
5.4.3.13	operator[]() [2/2]	52
5.4.3.14	isPar()	53
5.4.3.15	isPerp()	53
5.4.3.16	isEqual()	54
5.4.3.17	isNumEq()	54
5.4.3.18	setThreshold()	55
5.4.3.19	getThreshold()	55
5.4.3.20	setDefThreshold()	56
5.4.3.21	operator*=(()) [2/2]	56
5.4.3.22	operator/=(()) [2/2]	56
5.4.4	Member Data Documentation	56
5.4.4.1	x	56
5.4.4.2	y	57

5.5 geom::Vec3< T > Class Template Reference	57
5.5.1 Detailed Description	58
5.5.2 Constructor & Destructor Documentation	59
5.5.2.1 Vec3() [1/2]	59
5.5.2.2 Vec3() [2/2]	59
5.5.3 Member Function Documentation	59
5.5.3.1 operator+=( )	59
5.5.3.2 operator-=( )	60
5.5.3.3 operator-( )	60
5.5.3.4 operator*=( ) [1/2]	61
5.5.3.5 operator/=( ) [1/2]	61
5.5.3.6 dot( )	62
5.5.3.7 cross( )	62
5.5.3.8 length2( )	63
5.5.3.9 length( )	63
5.5.3.10 normalized( )	63
5.5.3.11 normalize( )	64
5.5.3.12 operator[]( ) [1/2]	64
5.5.3.13 operator[]( ) [2/2]	64
5.5.3.14 isPar( )	65
5.5.3.15 isPerp( )	65
5.5.3.16 isEqual( )	66
5.5.3.17 isNumEq( )	66
5.5.3.18 setThreshold( )	67
5.5.3.19 getThreshold( )	67
5.5.3.20 setDefThreshold( )	68
5.5.3.21 operator*=( ) [2/2]	68
5.5.3.22 operator/=( ) [2/2]	68
5.5.4 Member Data Documentation	68
5.5.4.1 x	68
5.5.4.2 y	69
5.5.4.3 z	69
<b>6 File Documentation</b>	<b>71</b>
6.1 include/distance/distance.hh File Reference	71
6.2 distance.hh	72
6.3 include/intersection/intersection.hh File Reference	73
6.4 intersection.hh	74
6.5 include/primitives/common.hh File Reference	79
6.6 common.hh	81
6.7 include/primitives/line.hh File Reference	81
6.8 line.hh	83

---

6.9 include/primitives/plane.hh File Reference . . . . .	84
6.10 plane.hh . . . . .	87
6.11 include/primitives/primitives.hh File Reference . . . . .	89
6.12 primitives.hh . . . . .	90
6.13 include/primitives/triangle.hh File Reference . . . . .	91
6.14 triangle.hh . . . . .	92
6.15 include/primitives/vec2.hh File Reference . . . . .	94
6.15.1 Detailed Description . . . . .	96
6.16 vec2.hh . . . . .	96
6.17 include/primitives/vec3.hh File Reference . . . . .	102
6.17.1 Detailed Description . . . . .	104
6.18 vec3.hh . . . . .	105





# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">geom</a>	<a href="#">Line.hh</a> <a href="#">Line</a> class implementation . . . . .	<a href="#">7</a>
<a href="#">geom::detail</a>	. . . . .	<a href="#">28</a>



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">geom::Line&lt; T &gt;</a>	
<a href="#">Line</a> class implementation . . . . .	<a href="#">33</a>
<a href="#">geom::Plane&lt; T &gt;</a>	
<a href="#">Plane</a> class realization . . . . .	<a href="#">36</a>
<a href="#">geom::Triangle&lt; T &gt;</a>	
<a href="#">Triangle</a> class implementation . . . . .	<a href="#">42</a>
<a href="#">geom::Vec2&lt; T &gt;</a>	
<a href="#">Vec2</a> class realization . . . . .	<a href="#">44</a>
<a href="#">geom::Vec3&lt; T &gt;</a>	
<a href="#">Vec3</a> class realization . . . . .	<a href="#">57</a>



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/distance/ <a href="#">distance.hh</a> . . . . .	71
include/intersection/ <a href="#">intersection.hh</a> . . . . .	73
include/primitives/ <a href="#">common.hh</a> . . . . .	79
include/primitives/ <a href="#">line.hh</a> . . . . .	81
include/primitives/ <a href="#">plane.hh</a> . . . . .	84
include/primitives/ <a href="#">primitives.hh</a> . . . . .	89
include/primitives/ <a href="#">triangle.hh</a> . . . . .	91
include/primitives/ <a href="#">vec2.hh</a> . . . . .	94
include/primitives/ <a href="#">vec3.hh</a> . . . . .	102



## 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 [Vec2](#)  
*[Vec2](#) class realization.*
- class [Vec3](#)  
*[Vec3](#) class realization.*

#### Typedefs

- using [Vec2D](#) = [Vec2](#)< double >
- using [Vec2F](#) = [Vec2](#)< float >
- using [Vec3D](#) = [Vec3](#)< double >
- using [Vec3F](#) = [Vec3](#)< float >

## Functions

- `template<std::floating_point T>`  
`T distance (const Plane< T > &pl, const Vec3< T > &pt)`  
*Calculates signed distance between point and plane.*
- `template<std::floating_point T>`  
`bool isIntersect (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>`  
`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>`  
`Vec2< T > operator+ (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Overloaded + operator.*
- `template<std::floating_point T>`  
`Vec2< T > operator- (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Overloaded - operator.*
- `template<Number nT, std::floating_point T>`  
`Vec2< T > operator* (const nT &val, const Vec2< T > &rhs)`  
*Overloaded multiple by value operator.*
- `template<Number nT, std::floating_point T>`  
`Vec2< T > operator* (const Vec2< T > &lhs, const nT &val)`  
*Overloaded multiple by value operator.*
- `template<Number nT, std::floating_point T>`  
`Vec2< T > operator/ (const Vec2< T > &lhs, const nT &val)`  
*Overloaded divide by value operator.*
- `template<std::floating_point T>`  
`T dot (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Dot product function.*
- `template<std::floating_point T>`  
`bool operator== (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Vec2 equality operator.*
- `template<std::floating_point T>`  
`bool operator!= (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Vec2 inequality operator.*
- `template<std::floating_point T>`  
`std::ostream & operator<< (std::ostream &ost, const Vec2< T > &vec)`



- Vec2 print operator.*
- template<std::floating\_point T>  
**Vec3**< T > **operator+** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Overloaded + operator.*
- template<std::floating\_point T>  
**Vec3**< T > **operator-** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Overloaded - operator.*
- template<Number nT, std::floating\_point T>  
**Vec3**< T > **operator\*** (const nT &val, const **Vec3**< T > &rhs)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
**Vec3**< T > **operator\*** (const **Vec3**< T > &lhs, const nT &val)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
**Vec3**< T > **operator/** (const **Vec3**< T > &lhs, const nT &val)  
*Overloaded divide by value operator.*
- template<std::floating\_point T>  
T **dot** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Dot product function.*
- template<std::floating\_point T>  
**Vec3**< T > **cross** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Cross product function.*
- template<std::floating\_point T>  
bool **operator==** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Vec3 equality operator.*
- template<std::floating\_point T>  
bool **operator!=** (const **Vec3**< T > &lhs, const **Vec3**< T > &rhs)  
*Vec3 inequality operator.*
- template<std::floating\_point T>  
std::ostream & **operator<<** (std::ostream &ost, const **Vec3**< T > &vec)  
*Vec3 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 Vec2D

```
using geom::Vec2D = typedef Vec2<double>
```

Definition at line 368 of file [vec2.hh](#).

#### 4.1.2.2 Vec2F

```
using geom::Vec2F = typedef Vec2<float>
```

Definition at line 369 of file [vec2.hh](#).

#### 4.1.2.3 Vec3D

```
using geom::Vec3D = typedef Vec3<double>
```

Definition at line 384 of file [vec3.hh](#).

#### 4.1.2.4 Vec3F

```
using geom::Vec3F = typedef Vec3<float>
```

Definition at line 385 of file [vec3.hh](#).

### 4.1.3 Function Documentation

#### 4.1.3.1 distance()

```
template<std::floating_point T>
T geom::distance (
    const Plane< T > & pl,
    const Vec3< 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::helperMollerHaines\(\)](#), and [geom::detail::isOnOneSide\(\)](#).

## 4.1.3.2 isIntersect()

```
template<std::floating_point T>
bool geom::isIntersect (
    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 150 of file [intersection.hh](#).

References [geom::Triangle< T >::getPlane\(\)](#), [geom::detail::isIntersect2D\(\)](#), [geom::detail::isIntersectBothInvalid\(\)](#), [geom::detail::isIntersectMollerHaines\(\)](#), [geom::detail::isIntersectValidInvalid\(\)](#), [geom::detail::isOnOneSide\(\)](#), and [geom::Triangle< T >::isValid\(\)](#).

#### 4.1.3.3 intersect()

```
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.

Common intersection case (parallel planes case is trivial):

Let  $\vec{P}$  - point in space

$$pl_1 \text{ equation: } \vec{n}_1 \cdot \vec{P} = d_1$$

$$pl_2 \text{ equation: } \vec{n}_2 \cdot \vec{P} = d_2$$

$$\text{Intersection line direction: } \vec{dir} = \vec{n}_1 \times \vec{n}_2$$

Let origin of intersection line be a linear combination of  $\vec{n}_1$  and  $\vec{n}_2$ :

$$\vec{P} = a \cdot \vec{n}_1 + b \cdot \vec{n}_2$$

$\vec{P}$  must satisfy both  $pl_1$  and  $pl_2$  equations:

$$\vec{n}_1 \cdot \vec{P} = d_1 \Leftrightarrow \vec{n}_1 \cdot (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) = d_1 \Leftrightarrow a + b \cdot \vec{n}_1 \cdot \vec{n}_2 = d_1$$

$$\vec{n}_2 \cdot \vec{P} = d_2 \Leftrightarrow \vec{n}_2 \cdot (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) = d_2 \Leftrightarrow a \cdot \vec{n}_1 \cdot \vec{n}_2 + b = d_2$$

Let's find  $a$  and  $b$ :

$$a = \frac{d_2 \cdot \vec{n}_1 \cdot \vec{n}_2 - d_1}{(\vec{n}_1 \cdot \vec{n}_2)^2 - 1}$$

$$b = \frac{d_1 \cdot \vec{n}_1 \cdot \vec{n}_2 - d_2}{(\vec{n}_1 \cdot \vec{n}_2)^2 - 1}$$

Intersection line equation:

$$\vec{r}(t) = \vec{P} + t \cdot \vec{n}_1 \times \vec{n}_2 = (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) + t \cdot \vec{n}_1 \times \vec{n}_2$$

#### Template Parameters

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

#### Parameters

$pl1$	first plane
$pl2$	second plane

#### Returns

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

Definition at line 183 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 operator<<() [1/5]

```
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>	<a href="#">Line</a> 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\(\)](#).

#### 4.1.3.5 operator==( ) [1/4]

```
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\(\)](#).

**4.1.3.6 operator==( ) [2/4]**

```
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 146 of file [plane.hh](#).

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

#### 4.1.3.7 operator<<() [2/5]

```
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 160 of file [plane.hh](#).

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

## 4.1.3.8 operator&lt;&lt;() [3/5]

```
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>	<a href="#">Triangle</a> to print

## Returns

std::ostream& modified ostream instance

Definition at line 76 of file [triangle.hh](#).

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

```
template<std::floating_point T>
Vec2<T> geom::operator+ (
```



```
const Vec2< T > & lhs,
const Vec2< 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

Vec2<T> sum of two vectors

Definition at line 235 of file [vec2.hh](#).

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

```
template<std::floating_point T>
Vec2<T> geom::operator- (
    const Vec2< T > & lhs,
    const Vec2< 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

Vec2<T> res of two vectors

Definition at line 251 of file [vec2.hh](#).

**4.1.3.11 operator\*() [1/4]**

```
template<Number nT, std::floating_point T>
Vec2<T> geom::operator* (
    const nT & val,
    const Vec2< 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**

Vec2<T> result vector

Definition at line 268 of file [vec2.hh](#).

**4.1.3.12 operator\*() [2/4]**

```
template<Number nT, std::floating_point T>
Vec2<T> geom::operator* (
    const Vec2< 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**

Vec2<T> result vector

Definition at line 285 of file [vec2.hh](#).

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

```
template<Number nT, std::floating_point T>
Vec2<T> geom::operator/ (
    const Vec2< 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

Vec2<T> result vector

Definition at line 302 of file [vec2.hh](#).

#### 4.1.3.14 dot() [1/2]

```
template<std::floating_point T>
T geom::dot (
    const Vec2< T > & lhs,
    const Vec2< 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 318 of file [vec2.hh](#).

References [geom::Vec2< T >::dot\(\)](#).

Referenced by [geom::detail::computeInterval\(\)](#), [distance\(\)](#), [geom::detail::helperMollerHaines\(\)](#), [intersect\(\)](#), [geom::Vec2< T >::isPerp\(\)](#), [geom::Vec3< T >::isPerp\(\)](#), [geom::Vec2< T >::length2\(\)](#), and [geom::Vec3< T >::length2\(\)](#).

**4.1.3.15 operator==( ) [3/4]**

```
template<std::floating_point T>
bool geom::operator==(
    const Vec2< T > & lhs,
    const Vec2< T > & rhs )
```

[Vec2](#) equality operator.

**Template Parameters**

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

**Parameters**

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

**Returns**

true if vectors are equal  
false otherwise

Definition at line 333 of file [vec2.hh](#).

References [geom::Vec2< T >::isEqual\(\)](#).

**4.1.3.16 operator!=( ) [1/2]**

```
template<std::floating_point T>
bool geom::operator!=(
    const Vec2< T > & lhs,
    const Vec2< T > & rhs )
```

[Vec2](#) 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 348 of file [vec2.hh](#).

4.1.3.17 `operator<<()` [4/5]

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Vec2< T > & vec )
```

[Vec2](#) 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 362 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

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

```
template<std::floating_point T>
Vec3<T> geom::operator+ (
    const Vec3< T > & lhs,
    const Vec3< 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**

Vec3<T> sum of two vectors

Definition at line 237 of file [vec3.hh](#).

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

```
template<std::floating_point T>
Vec3<T> geom::operator- (
    const Vec3< T > & lhs,
    const Vec3< 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**

Vec3<T> res of two vectors

Definition at line 253 of file [vec3.hh](#).

**4.1.3.20 operator\*() [3/4]**

```
template<Number nT, std::floating_point T>
Vec3<T> geom::operator* (
    const nT & val,
    const Vec3< 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**

Vec3<T> result vector

Definition at line 270 of file [vec3.hh](#).

**4.1.3.21 operator\*() [4/4]**

```
template<Number nT, std::floating_point T>
Vec3<T> geom::operator* (
    const Vec3< 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**

Vec3<T> result vector

Definition at line 287 of file [vec3.hh](#).

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

```
template<Number nT, std::floating_point T>
Vec3<T> geom::operator/ (
    const Vec3< 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

Vec3<T> result vector

Definition at line 304 of file [vec3.hh](#).

#### 4.1.3.23 dot() [2/2]

```
template<std::floating_point T>
T geom::dot (
    const Vec3< T > & lhs,
    const Vec3< 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 320 of file [vec3.hh](#).

References [geom::Vec3< T >::dot\(\)](#).

**4.1.3.24 cross()**

```
template<std::floating_point T>
Vec3<T> geom::cross (
    const Vec3< T > & lhs,
    const Vec3< 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 334 of file [vec3.hh](#).

References [geom::Vec3< T >::cross\(\)](#).

Referenced by [intersect\(\)](#), [geom::Vec3< T >::isPar\(\)](#), and [geom::Triangle< T >::isValid\(\)](#).

**4.1.3.25 operator==( ) [4/4]**

```
template<std::floating_point T>
bool geom::operator== (
    const Vec3< T > & lhs,
    const Vec3< T > & rhs )
```

[Vec3](#) 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 349 of file [vec3.hh](#).

References [geom::Vec3< T >::isEqual\(\)](#).

## 4.1.3.26 operator"!=()" [2/2]

```
template<std::floating_point T>
bool geom::operator!= (
    const Vec3< T > & lhs,
    const Vec3< T > & rhs )
```

[Vec3](#) 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 364 of file [vec3.hh](#).

**4.1.3.27 operator<<() [5/5]**

```
template<std::floating_point T>
std::ostream& geom::operator<< (
    std::ostream & ost,
    const Vec3< T > & vec )
```

Vec3 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 378 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< 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 15 of file [common.hh](#).

## 4.2 geom::detail Namespace Reference

### Typedefs

- `template<typename T >`  
`using Segment = std::pair< T, T >`
- `template<std::floating_point T>`  
`using Trian2 = std::array< Vec2< T >, 3 >`

### Functions

- `template<std::floating_point T>`  
`bool isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`bool isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`Segment< T > helperMollerHaines (const Triangle< T > &tr, const Plane< T > &pl, const Line< T > &l)`
- `template<std::floating_point T>`  
`bool isIntersectBothInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`bool isIntersectValidInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`bool isOverlap (Segment< T > &segm1, Segment< T > &segm2)`
- `template<std::forward_iterator It>`  
`bool isSameSign (It begin, It end)`
- `template<std::floating_point T>`  
`bool isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)`
- `template<std::floating_point T>`  
`Trian2< T > getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)`
- `template<std::floating_point T>`  
`bool isCounterClockwise (Trian2< T > &tr)`
- `template<std::floating_point T>`  
`Segment< T > computeInterval (const Trian2< T > &tr, const Vec2< T > &d)`

### 4.2.1 Typedef Documentation

#### 4.2.1.1 Segment

```
template<typename T >
using geom::detail::Segment = typedef std::pair<T, T>
```

Definition at line 104 of file [intersection.hh](#).

#### 4.2.1.2 Trian2

```
template<std::floating_point T>
using geom::detail::Trian2 = typedef std::array<Vec2<T>, 3>
```

Definition at line 107 of file [intersection.hh](#).

## 4.2.2 Function Documentation

### 4.2.2.1 isIntersect2D()

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

Definition at line 214 of file [intersection.hh](#).

References [computeInterval\(\)](#), [geom::Triangle< T >::getPlane\(\)](#), and [getTrian2\(\)](#).

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

### 4.2.2.2 isIntersectMollerHaines()

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

Definition at line 239 of file [intersection.hh](#).

References [geom::Triangle< T >::getPlane\(\)](#), [helperMollerHaines\(\)](#), [geom::intersect\(\)](#), and [isOverlap\(\)](#).

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

### 4.2.2.3 helperMollerHaines()

```
template<std::floating_point T>
Segment< T > geom::detail::helperMollerHaines (
    const Triangle< T > & tr,
    const Plane< T > & pl,
    const Line< T > & l )
```

Definition at line 253 of file [intersection.hh](#).

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

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

#### 4.2.2.4 isIntersectBothInvalid()

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

Definition at line 289 of file [intersection.hh](#).

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

#### 4.2.2.5 isIntersectValidInvalid()

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

Definition at line 295 of file [intersection.hh](#).

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

#### 4.2.2.6 isOverlap()

```
template<std::floating_point T>
bool geom::detail::isOverlap (
    Segment< T > & segm1,
    Segment< T > & segm2 )
```

Definition at line 301 of file [intersection.hh](#).

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

#### 4.2.2.7 isSameSign()

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

Definition at line 307 of file [intersection.hh](#).

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

#### 4.2.2.8 isOnOneSide()

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

Definition at line 320 of file [intersection.hh](#).

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

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

#### 4.2.2.9 getTrian2()

```
template<std::floating_point T>
Trian2< T > geom::detail::getTrian2 (
    const Plane< T > & pl,
    const Triangle< T > & tr )
```

Definition at line 333 of file [intersection.hh](#).

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

Referenced by [isIntersect2D\(\)](#).

#### 4.2.2.10 isCounterClockwise()

```
template<std::floating_point T>
bool geom::detail::isCounterClockwise (
    Trian2< T > & tr )
```

Definition at line 367 of file [intersection.hh](#).

Referenced by [getTrian2\(\)](#).

#### 4.2.2.11 computeInterval()

```
template<std::floating_point T>
Segment< T > geom::detail::computeInterval (
    const Trian2< T > & tr,
    const Vec2< T > & d )
```

Definition at line 387 of file [intersection.hh](#).

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

Referenced by [isIntersect2D\(\)](#).





## Chapter 5

# Class Documentation

### 5.1 geom::Line< T > Class Template Reference

[Line](#) class implementation.

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

#### Public Member Functions

- [Line](#) (const [Vec3](#)< T > &org, const [Vec3](#)< T > &dir)  
*Construct a new [Line](#) object.*
- const [Vec3](#)< T > & [org](#) () const  
*Getter for origin vector.*
- const [Vec3](#)< T > & [dir](#) () const  
*Getter for direction vector.*
- bool [belongs](#) (const [Vec3](#)< 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 [Vec3](#)< T > &p1, const [Vec3](#)< 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 Vec3< T > & org,
    const Vec3< 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 Vec3< T > & geom::Line< T >::org
```

Getter for origin vector.

#### Returns

const Vec3<T>& const reference to origin vector

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

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

## 5.1.3.2 dir()

```
template<std::floating_point T>
const Vec3< T > & geom::Line< T >::dir
```

Getter for direction vector.

## Returns

const Vec3<T>& const reference to direction vector

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

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

## 5.1.3.3 belongs()

```
template<std::floating_point T>
bool geom::Line< T >::belongs (
    const Vec3< 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 Vec3< T > & p1,
    const Vec3< 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 [Vec3](#)< T > & [norm](#) () const  
*Getter for normal vector.*
- bool [belongs](#) (const [Vec3](#)< 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 [Vec3](#)< T > &pt1, const [Vec3](#)< T > &pt2, const [Vec3](#)< T > &pt3)  
*Get plane by 3 points.*
- static [Plane getParametric](#) (const [Vec3](#)< T > &org, const [Vec3](#)< T > &dir1, const [Vec3](#)< T > &dir2)  
*Get plane from parametric plane equation.*
- static [Plane getNormalPoint](#) (const [Vec3](#)< T > &norm, const [Vec3](#)< T > &point)  
*Get plane from normal point plane equation.*
- static [Plane getNormalDist](#) (const [Vec3](#)< T > &norm, T constant)  
*Get plane form normal const plane equation.*

### 5.2.1 Detailed Description

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

[Plane](#) class realization.

Template Parameters

<a href="#">T</a>	- 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 174 of file [plane.hh](#).

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

### 5.2.2.2 norm()

```
template<std::floating_point T>
const Vec3< T > & geom::Plane< T >::norm
```

Getter for normal vector.

#### Returns

const Vec3<T>& const reference to normal vector

Definition at line 180 of file [plane.hh](#).

Referenced by [geom::distance\(\)](#), [geom::detail::getTrian2\(\)](#), [geom::intersect\(\)](#), and [geom::operator<<\(\)](#).

### 5.2.2.3 belongs() [1/2]

```
template<std::floating_point T>
bool geom::Plane< T >::belongs (
    const Vec3< 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 186 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 192 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 198 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 204 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 Vec3< T > & pt1,
    const Vec3< T > & pt2,
    const Vec3< 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 210 of file [plane.hh](#).

Referenced by [geom::Triangle< T >::getPlane\(\)](#).

### 5.2.2.8 getParametric()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getParametric (
    const Vec3< T > & org,
    const Vec3< T > & dir1,
    const Vec3< 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 217 of file [plane.hh](#).References [geom::Vec3< T >::cross\(\)](#).**5.2.2.9 getNormalPoint()**

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalPoint (
    const Vec3< T > & norm,
    const Vec3< 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 225 of file [plane.hh](#).References [geom::Vec3< T >::normalized\(\)](#).**5.2.2.10 getNormalDist()**

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalDist (
    const Vec3< 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 232 of file [plane.hh](#).

References [geom::Vec3< 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 [Vec3< T >](#) &p1, const [Vec3< T >](#) &p2, const [Vec3< T >](#) &p3)  
*Construct a new [Triangle](#) object from 3 points.*
- const [Vec3< T >](#) & [operator\[\]](#) (std::size\_t idx) const  
*Overloaded operator[] to get access to vertices.*
- [Plane< T >](#) [getPlane](#) () const  
*Get triangle's plane.*
- bool [isValid](#) () const  
*Check is triangle valid.*

### 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 25 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 Vec3< T > & p1,
    const Vec3< T > & p2,
    const Vec3< 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 88 of file [triangle.hh](#).

## 5.3.3 Member Function Documentation

### 5.3.3.1 operator[]()

```
template<std::floating_point T>
const Vec3< 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 Vec3<T>& const reference to vertex

Definition at line 93 of file [triangle.hh](#).

### 5.3.3.2 getPlane()

```
template<std::floating_point T>
Plane< T > geom::Triangle< T >::getPlane
```

Get triangle's plane.

**Returns**

Plane<T>

Definition at line 99 of file [triangle.hh](#).

References [geom::Plane< T >::getBy3Points\(\)](#).

Referenced by [geom::isIntersect\(\)](#), [geom::detail::isIntersect2D\(\)](#), and [geom::detail::isIntersectMollerHaines\(\)](#).

**5.3.3.3 isValid()**

```
template<std::floating_point T>
bool geom::Triangle< T >::isValid
```

Check is triangle valid.

**Returns**

true if triangle is valid  
false if triangle is invalid

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

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

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

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

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

**5.4 geom::Vec2< T > Class Template Reference**

[Vec2](#) class realization.

```
#include <vec2.hh>
```

## Public Member Functions

- [Vec2](#) (T coordX, T coordY)  
*Construct a new [Vec2](#) object from 3 coordinates.*
- [Vec2](#) (T coordX={})  
*Construct a new [Vec2](#) object with equals coordinates.*
- [Vec2](#) & [operator+=](#) (const [Vec2](#) &vec)  
*Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.*
- [Vec2](#) & [operator-=](#) (const [Vec2](#) &vec)  
*Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.*
- [Vec2](#) [operator-](#) () const  
*Unary - operator.*
- template<Number nType>  
[Vec2](#) & [operator\\*=](#) (nType val)  
*Overloaded \*= by number operator.*
- template<Number nType>  
[Vec2](#) & [operator/=](#) (nType val)  
*Overloaded /= by number operator.*
- T [dot](#) (const [Vec2](#) &rhs) const  
*Dot product function.*
- T [length2](#) () const  
*Calculate squared length of a vector function.*
- T [length](#) () const  
*Calculate length of a vector function.*
- [Vec2](#) [getPerp](#) () const  
*Get the perpendicular to this vector.*
- [Vec2](#) [normalized](#) () const  
*Get normalized vector function.*
- [Vec2](#) & [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 [Vec2](#) &rhs) const  
*Check if vector is parallel to another.*
- bool [isPerp](#) (const [Vec2](#) &rhs) const  
*Check if vector is perpendicular to another.*
- bool [isEqual](#) (const [Vec2](#) &rhs) const  
*Check if vector is equal to another.*
- template<Number nType>  
[Vec2](#)< T > & [operator\\*=](#) (nType val)
- template<Number nType>  
[Vec2](#)< 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 T [getThreshold](#) ()  
*Get current threshold value.*
- static void [setDefThreshold](#) ()  
*Set threshold to default value.*

## Public Attributes

- `T x {}`  
*Vec2* coordinates.
- `T y {}`

### 5.4.1 Detailed Description

```
template<std::floating_point T>
class geom::Vec2< T >
```

*Vec2* class realization.

Template Parameters

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

Definition at line 27 of file [vec2.hh](#).

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 *Vec2()* [1/2]

```
template<std::floating_point T>
geom::Vec2< T >::Vec2 (
    T coordX,
    T coordY ) [inline]
```

Construct a new *Vec2* object from 3 coordinates.

Parameters

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

Definition at line 47 of file [vec2.hh](#).

#### 5.4.2.2 *Vec2()* [2/2]

```
template<std::floating_point T>
geom::Vec2< T >::Vec2 (
    T coordX = {} ) [inline], [explicit]
```

Construct a new *Vec2* object with equals coordinates.

## Parameters

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

Definition at line 55 of file [vec2.hh](#).

### 5.4.3 Member Function Documentation

#### 5.4.3.1 operator+=()

```
template<std::floating_point T>
Vec2< T > & geom::Vec2< T >::operator+= (
    const Vec2< T > & vec )
```

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

## Parameters

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

## Returns

[Vec2](#)& reference to current instance

Definition at line 372 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

#### 5.4.3.2 operator-=()

```
template<std::floating_point T>
Vec2< T > & geom::Vec2< T >::operator-= (
    const Vec2< T > & vec )
```

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

## Parameters

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

## Returns

[Vec2](#)& reference to current instance

Definition at line 381 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

#### 5.4.3.3 operator-()

```
template<std::floating_point T>
Vec2< T > geom::Vec2< T >::operator-
```

Unary - operator.

##### Returns

[Vec2](#) negated [Vec2](#) instance

Definition at line 390 of file [vec2.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec2& geom::Vec2< 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

[Vec2&](#) reference to vector instance

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

```
template<std::floating_point T>
template<Number nType>
```



```
Vec2& geom::Vec2< T >::operator/= (
    nType val )
```

Overloaded /= by number operator.

**Template Parameters**

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

**Parameters**

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

**Returns**

[Vec2](#)& reference to vector instance

**Warning**

Does not check if *val* equals 0

**5.4.3.6 dot()**

```
template<std::floating_point T>
T geom::Vec2< T >::dot (
    const Vec2< 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 416 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

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

**5.4.3.7 length2()**

```
template<std::floating_point T>
T geom::Vec2< T >::length2
```

Calculate squared length of a vector function.

**Returns**

$T \text{ length}^2$

Definition at line 422 of file [vec2.hh](#).

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

**5.4.3.8 length()**

```
template<std::floating_point T>
T geom::Vec2< T >::length
```

Calculate length of a vector function.

**Returns**

$T \text{ length}$

Definition at line 428 of file [vec2.hh](#).

**5.4.3.9 getPerp()**

```
template<std::floating_point T>
Vec2< T > geom::Vec2< T >::getPerp
```

Get the perpendicular to this vector.

**Returns**

[Vec2](#) perpendicular vector

Definition at line 434 of file [vec2.hh](#).

**5.4.3.10 normalized()**

```
template<std::floating_point T>
Vec2< T > geom::Vec2< T >::normalized
```

Get normalized vector function.

**Returns**

[Vec2](#) normalized vector

Definition at line 440 of file [vec2.hh](#).

References [geom::Vec2< T >::normalize\(\)](#).

#### 5.4.3.11 normalize()

```
template<std::floating_point T>
Vec2< T > & geom::Vec2< T >::normalize
```

Normalize vector function.

##### Returns

Vec2& reference to instance

Definition at line 448 of file [vec2.hh](#).

Referenced by [geom::Vec2< T >::normalized\(\)](#).

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

```
template<std::floating_point T>
T & geom::Vec2< 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)
----------	------------------------------------

##### Returns

T& reference to coordinate value

##### Note

Coordinates calculated by mod 2

Definition at line 457 of file [vec2.hh](#).

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

```
template<std::floating_point T>
T geom::Vec2< 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)
----------	------------------------------------

## Returns

T coordinate value

## Note

Coordinates calculated by mod 2

Definition at line 471 of file [vec2.hh](#).

## 5.4.3.14 isPar()

```
template<std::floating_point T>
bool geom::Vec2< T >::isPar (
    const Vec2< 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 485 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

## 5.4.3.15 isPerp()

```
template<std::floating_point T>
bool geom::Vec2< T >::isPerp (
    const Vec2< 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 492 of file [vec2.hh](#).

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

**5.4.3.16 isEqual()**

```
template<std::floating_point T>
bool geom::Vec2< T >::isEqual (
    const Vec2< 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 498 of file [vec2.hh](#).

References [geom::Vec2< T >::x](#), and [geom::Vec2< T >::y](#).

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

**5.4.3.17 isNumEq()**

```
template<std::floating_point T>
bool geom::Vec2< 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 ( $|lhs - rhs| < threshold$ )  
false otherwise

## Note

Threshold defined by `threshold_ static` member

Definition at line 504 of file [vec2.hh](#).

**5.4.3.18 setThreshold()**

```
template<std::floating_point T>
void geom::Vec2< T >::setThreshold (
    T thres ) [static]
```

Set new threshold value.

## Parameters

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

Definition at line 510 of file [vec2.hh](#).

**5.4.3.19 getThreshold()**

```
template<std::floating_point T>
T geom::Vec2< T >::getThreshold [static]
```

Get current threshold value.

Definition at line 516 of file [vec2.hh](#).

#### 5.4.3.20 setDefThreshold()

```
template<std::floating_point T>
void geom::Vec2< T >::setDefThreshold [static]
```

Set threshold to default value.

##### Note

default value equals float point epsilon

Definition at line 522 of file [vec2.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec2<T>& geom::Vec2< T >::operator*= (
    nType val )
```

Definition at line 397 of file [vec2.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec2<T>& geom::Vec2< T >::operator/= (
    nType val )
```

Definition at line 407 of file [vec2.hh](#).

### 5.4.4 Member Data Documentation

#### 5.4.4.1 x

```
template<std::floating_point T>
T geom::Vec2< T >::x {}
```

[Vec2](#) coordinates.

Definition at line 39 of file [vec2.hh](#).

Referenced by [geom::Vec2< T >::dot\(\)](#), [geom::Vec2< T >::isEqual\(\)](#), [geom::Vec2< T >::isPar\(\)](#), [geom::Vec2< T >::operator+=\( \)](#), [geom::Vec2< T >::operator-=\( \)](#), and [geom::operator<<\(\)](#).



## 5.4.4.2 y

```
template<std::floating_point T>
T geom::Vec2< T >::y {}
```

Definition at line 39 of file [vec2.hh](#).

Referenced by [geom::Vec2< T >::dot\(\)](#), [geom::Vec2< T >::isEqual\(\)](#), [geom::Vec2< T >::isPar\(\)](#), [geom::Vec2< T >::operator+=\(\)](#), [geom::Vec2< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

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

- include/primitives/[vec2.hh](#)

## 5.5 geom::Vec3&lt; T &gt; Class Template Reference

[Vec3](#) class realization.

```
#include <vec3.hh>
```

## Public Member Functions

- [Vec3](#) (T coordX, T coordY, T coordZ)  
*Construct a new [Vec3](#) object from 3 coordinates.*
- [Vec3](#) (T coordX={})  
*Construct a new [Vec3](#) object with equals coordinates.*
- [Vec3](#) & [operator+=](#) (const [Vec3](#) &vec)  
*Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.*
- [Vec3](#) & [operator-=](#) (const [Vec3](#) &vec)  
*Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.*
- [Vec3](#) [operator-](#) () const  
*Unary - operator.*
- template<Number nType>  
[Vec3](#) & [operator\\*=](#) (nType val)  
*Overloaded \*= by number operator.*
- template<Number nType>  
[Vec3](#) & [operator/=](#) (nType val)  
*Overloaded /= by number operator.*
- T [dot](#) (const [Vec3](#) &rhs) const  
*Dot product function.*
- [Vec3](#) [cross](#) (const [Vec3](#) &rhs) const  
*Cross product function.*
- T [length2](#) () const  
*Calculate squared length of a vector function.*
- T [length](#) () const  
*Calculate length of a vector function.*
- [Vec3](#) [normalized](#) () const  
*Get normalized vector function.*
- [Vec3](#) & [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 Vec3 &rhs) const`  
*Check if vector is parallel to another.*
- `bool isPerp (const Vec3 &rhs) const`  
*Check if vector is perpendicular to another.*
- `bool isEqual (const Vec3 &rhs) const`  
*Check if vector is equal to another.*
- `template<Number nType>`  
`Vec3< T > & operator*=( nType val)`
- `template<Number nType>`  
`Vec3< 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 T getThreshold ()`  
*Get current threshold value.*
- `static void setDefThreshold ()`  
*Set threshold to default value.*

## Public Attributes

- `T x {}`  
*Vec3 coordinates.*
- `T y {}`
- `T z {}`

### 5.5.1 Detailed Description

```
template<std::floating_point T>
class geom::Vec3< T >
```

`Vec3` class realization.

#### Template Parameters

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

Definition at line 27 of file `vec3.hh`.

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 Vec3() [1/2]

```
template<std::floating_point T>
geom::Vec3< T >::Vec3 (
    T coordX,
    T coordY,
    T coordZ ) [inline]
```

Construct a new [Vec3](#) 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 48 of file [vec3.hh](#).

### 5.5.2.2 Vec3() [2/2]

```
template<std::floating_point T>
geom::Vec3< T >::Vec3 (
    T coordX = {} ) [inline], [explicit]
```

Construct a new [Vec3](#) object with equals coordinates.

#### Parameters

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

Definition at line 56 of file [vec3.hh](#).

## 5.5.3 Member Function Documentation

### 5.5.3.1 operator+=()

```
template<std::floating_point T>
Vec3< T > & geom::Vec3< T >::operator+= (
    const Vec3< T > & vec )
```

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

## Parameters

in	vec	vector to incremented with
----	-----	----------------------------

## Returns

[Vec3](#)& reference to current instance

Definition at line 388 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< T >::z](#).

### 5.5.3.2 operator-=()

```
template<std::floating_point T>
Vec3< T > & geom::Vec3< T >::operator-= (
    const Vec3< T > & vec )
```

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

## Parameters

in	vec	vector to decremented with
----	-----	----------------------------

## Returns

[Vec3](#)& reference to current instance

Definition at line 398 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< T >::z](#).

### 5.5.3.3 operator-()

```
template<std::floating_point T>
Vec3< T > geom::Vec3< T >::operator-
```

Unary - operator.

## Returns

[Vec3](#) negated [Vec3](#) instance

Definition at line 408 of file [vec3.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec3& geom::Vec3< 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**

[Vec3&](#) reference to vector instance

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

```
template<std::floating_point T>
template<Number nType>
Vec3& geom::Vec3< 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**

[Vec3&](#) reference to vector instance

**Warning**

Does not check if val equals 0

### 5.5.3.6 dot()

```
template<std::floating_point T>
T geom::Vec3< T >::dot (
    const Vec3< 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 436 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< T >::z](#).

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

### 5.5.3.7 cross()

```
template<std::floating_point T>
Vec3< T > geom::Vec3< T >::cross (
    const Vec3< T > & rhs ) const
```

Cross product function.

#### Parameters

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

#### Returns

[Vec3](#) cross product of two vectors

Definition at line 442 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< T >::z](#).

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

#### 5.5.3.8 length2()

```
template<std::floating_point T>
T geom::Vec3< T >::length2
```

Calculate squared length of a vector function.

##### Returns

$T \text{ length}^2$

Definition at line 448 of file [vec3.hh](#).

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

#### 5.5.3.9 length()

```
template<std::floating_point T>
T geom::Vec3< T >::length
```

Calculate length of a vector function.

##### Returns

$T \text{ length}$

Definition at line 454 of file [vec3.hh](#).

#### 5.5.3.10 normalized()

```
template<std::floating_point T>
Vec3< T > geom::Vec3< T >::normalized
```

Get normalized vector function.

##### Returns

[Vec3](#) normalized vector

Definition at line 460 of file [vec3.hh](#).

References [geom::Vec3< T >::normalize\(\)](#).

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

### 5.5.3.11 normalize()

```
template<std::floating_point T>
Vec3< T > & geom::Vec3< T >::normalize
```

Normalize vector function.

#### Returns

Vec3& reference to instance

Definition at line 468 of file [vec3.hh](#).

Referenced by [geom::Vec3< T >::normalized\(\)](#).

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

```
template<std::floating_point T>
T & geom::Vec3< 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 477 of file [vec3.hh](#).

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

```
template<std::floating_point T>
T geom::Vec3< 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 493 of file [vec3.hh](#).

## 5.5.3.14 isPar()

```
template<std::floating_point T>
bool geom::Vec3< T >::isPar (
    const Vec3< 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 509 of file [vec3.hh](#).

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

## 5.5.3.15 isPerp()

```
template<std::floating_point T>
bool geom::Vec3< T >::isPerp (
    const Vec3< T > & rhs ) const
```

Check if vector is perpendicular to another.

**Parameters**

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

**Returns**

true if vector is perpendicular  
false otherwise

Definition at line 515 of file [vec3.hh](#).

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

**5.5.3.16 isEqual()**

```
template<std::floating_point T>
bool geom::Vec3< T >::isEqual (
    const Vec3< T > & rhs ) const
```

Check if vector is equal to another.

**Parameters**

<code>in</code>	<code>rhs</code>	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 521 of file [vec3.hh](#).

References [geom::Vec3< T >::x](#), [geom::Vec3< T >::y](#), and [geom::Vec3< T >::z](#).

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

**5.5.3.17 isNumEq()**

```
template<std::floating_point T>
bool geom::Vec3< 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 ( $|lhs - rhs| < threshold$ )  
false otherwise

## Note

Threshold defined by `threshold_` static member

Definition at line 527 of file [vec3.hh](#).

### 5.5.3.18 setThreshold()

```
template<std::floating_point T>
void geom::Vec3< T >::setThreshold (
    T thres ) [static]
```

Set new threshold value.

## Parameters

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

Definition at line 533 of file [vec3.hh](#).

### 5.5.3.19 getThreshold()

```
template<std::floating_point T>
T geom::Vec3< T >::getThreshold [static]
```

Get current threshold value.

Definition at line 539 of file [vec3.hh](#).

#### 5.5.3.20 setDefThreshold()

```
template<std::floating_point T>
void geom::Vec3< T >::setDefThreshold [static]
```

Set threshold to default value.

##### Note

default value equals float point epsilon

Definition at line 545 of file [vec3.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec3<T>& geom::Vec3< T >::operator*= (
    nType val )
```

Definition at line 415 of file [vec3.hh](#).

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

```
template<std::floating_point T>
template<Number nType>
Vec3<T>& geom::Vec3< T >::operator/= (
    nType val )
```

Definition at line 426 of file [vec3.hh](#).

### 5.5.4 Member Data Documentation

#### 5.5.4.1 x

```
template<std::floating_point T>
T geom::Vec3< T >::x {}
```

[Vec3](#) coordinates.

Definition at line 39 of file [vec3.hh](#).

Referenced by [geom::Vec3< T >::cross\(\)](#), [geom::Vec3< T >::dot\(\)](#), [geom::Vec3< T >::isEqual\(\)](#), [geom::Vec3< T >::operator+=\( \)](#), [geom::Vec3< T >::operator-=\( \)](#), and [geom::operator<<\(\)](#).

#### 5.5.4.2 y

```
template<std::floating_point T>
T geom::Vec3< T >::y {}
```

Definition at line 39 of file [vec3.hh](#).

Referenced by [geom::Vec3< T >::cross\(\)](#), [geom::Vec3< T >::dot\(\)](#), [geom::Vec3< T >::isEqual\(\)](#), [geom::Vec3< T >::operator+=\(\)](#), [geom::Vec3< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

#### 5.5.4.3 z

```
template<std::floating_point T>
T geom::Vec3< T >::z {}
```

Definition at line 39 of file [vec3.hh](#).

Referenced by [geom::Vec3< T >::cross\(\)](#), [geom::Vec3< T >::dot\(\)](#), [geom::Vec3< T >::isEqual\(\)](#), [geom::Vec3< T >::operator+=\(\)](#), [geom::Vec3< T >::operator-=\(\)](#), and [geom::operator<<\(\)](#).

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

- [include/primitives/vec3.hh](#)



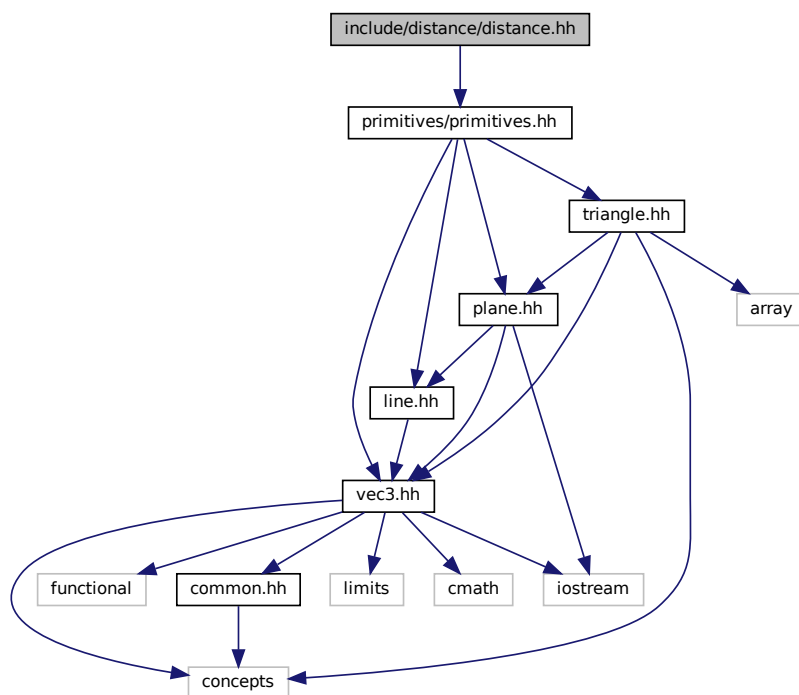
## Chapter 6

# File Documentation

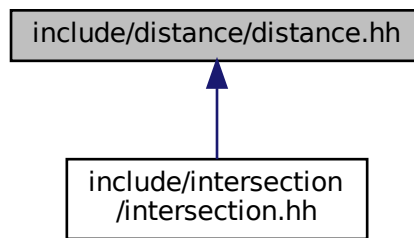
### 6.1 include/distance/distance.hh File Reference

```
#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 Vec3< T > &pt)`  
*Calculates signed distance between point and plane.*

## 6.2 distance.hh

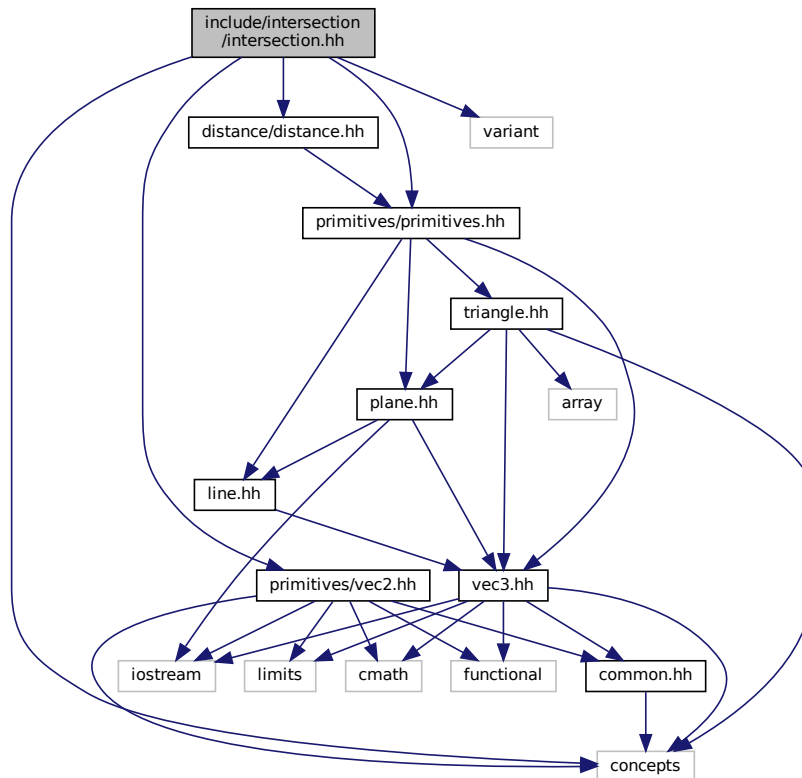
```

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 Vec3<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 Vec3<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 "primitives/vec2.hh"
Include dependency graph for intersection.hh:
```



## Namespaces

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

## Typedefs

- `template<typename T>`  
   using [geom::detail::Segment](#) = `std::pair< T, T >`
- `template<std::floating_point T>`  
   using [geom::detail::Trian2](#) = `std::array< Vec2< T >, 3 >`

## Functions

- `template<std::floating_point T>`  
`bool geom::isIntersect (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)`
- `template<std::floating_point T>`  
`bool geom::detail::isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`Segment< T > geom::detail::helperMollerHaines (const Triangle< T > &tr, const Plane< T > &pl, const Line< T > &l)`
- `template<std::floating_point T>`  
`bool geom::detail::isIntersectBothInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`bool geom::detail::isIntersectValidInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)`
- `template<std::floating_point T>`  
`bool geom::detail::isOverlap (Segment< T > &segm1, Segment< T > &segm2)`
- `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>`  
`Trian2< T > geom::detail::getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)`
- `template<std::floating_point T>`  
`bool geom::detail::isCounterClockwise (Trian2< T > &tr)`
- `template<std::floating_point T>`  
`Segment< T > geom::detail::computeInterval (const Trian2< T > &tr, const Vec2< T > &d)`

## 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 #include "primitives/vec2.hh"
00010
00011 namespace geom
00012 {
00013
00014 /**
00015  * @brief Checks intersection of 2 triangles
00016  *
00017  * @tparam T - floating point type of coordinates
00018  * @param tr1 first triangle
00019  * @param tr2 second triangle
00020  * @return true if triangles are intersect
00021  * @return false if triangles are not intersect
00022  */
00023 template <std::floating_point T>
00024 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2);
00025
00026 /**
00027  * @brief Intersect 2 planes and return result of intersection
00028  * @details
00029  * Common intersection case (parallel planes case is trivial):
00030  *
00031  * Let  $\vec{P}$  - point in space

```

```

00032 *
00033 * \f$ pl_1 \f$ equation: \f$ \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1 \f$
00034 *
00035 * \f$ pl_2 \f$ equation: \f$ \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2 \f$
00036 *
00037 * Intersection line direction: \f$ \overrightarrow{dir} = \overrightarrow{n}_1 \times
00038 * \overrightarrow{n}_2 \f$
00039 *
00040 * Let origin of intersection line be a linear combination of \f$ \overrightarrow{n}_1 \f$
00041 * and \f$ \overrightarrow{n}_2 \f$: \f[ \overrightarrow{P} = a \cdot \overrightarrow{n}_1
00042 * + b \cdot \overrightarrow{n}_2 \f]
00043 *
00044 * \f$ \overrightarrow{P} \f$ must satisfy both \f$ pl_1 \f$ and \f$ pl_2 \f$ equations:
00045 * \f[
00046 * \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1
00047 * \Leftrightarrow
00048 * \overrightarrow{n}_1
00049 * \cdot
00050 * \left(
00051 * a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00052 * \right)
00053 * = d_1
00054 * \Leftrightarrow
00055 * a + b \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 = d_1
00056 * \f]
00057 * \f[
00058 * \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2
00059 * \Leftrightarrow
00060 * \overrightarrow{n}_2
00061 * \cdot
00062 * \left(
00063 * a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00064 * \right) = d_2
00065 * \Leftrightarrow
00066 * a \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 + b = d_2
00067 * \f]
00068 *
00069 * Let's find \f$a\f$ and \f$b\f$:
00070 * \f[
00071 * a = \frac{
00072 * d_2 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_1
00073 * }{
00074 * \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 \right)^2 - 1
00075 * }
00076 * \f]
00077 * \f[
00078 * b = \frac{
00079 * d_1 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_2
00080 * }{
00081 * \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 \right)^2 - 1
00082 * }
00083 * \f]
00084 *
00085 * Intersection line equation:
00086 * \f[
00087 * \overrightarrow{r}(t) = \overrightarrow{P} + t \cdot \overrightarrow{n}_1 \times
00088 * \overrightarrow{n}_2 = (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) +
00089 * t \cdot \overrightarrow{n}_1 \times \overrightarrow{n}_2 \f]
00090 *
00091 * @tparam T - floating point type of coordinates
00092 * @param pl1 first plane
00093 * @param pl2 second plane
00094 * @return std::variant<std::monostate, Line<T>, Plane<T>
00095 */
00096 template <std::floating_point T>
00097 std::variant<std::monostate, Line<T>, Plane<T> intersect(const Plane<T> &pl1,
00098                                                         const Plane<T> &pl2);
00099
00100 namespace detail
00101 {
00102
00103 template <typename T>
00104 using Segment = std::pair<T, T>;
00105
00106 template <std::floating_point T>
00107 using Trian2 = std::array<Vec2<T>, 3>;
00108
00109 template <std::floating_point T>
00110 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2);
00111
00112 template <std::floating_point T>
00113 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2);
00114
00115 template <std::floating_point T>
00116 Segment<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl,
00117                               const Line<T> &l);
00118

```

```

00119 template <std::floating_point T>
00120 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2);
00121
00122 template <std::floating_point T>
00123 bool isIntersectValidInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2);
00124
00125 template <std::floating_point T>
00126 bool isOverlap(Segment<T> &segm1, Segment<T> &segm2);
00127
00128 template <std::forward_iterator It>
00129 bool isSameSign(It begin, It end);
00130
00131 template <std::floating_point T>
00132 bool isOnOneSide(const Plane<T> &p1, const Triangle<T> &tr);
00133
00134 template <std::floating_point T>
00135 Trian2<T> getTrian2(const Plane<T> &p1, const Triangle<T> &tr);
00136
00137 template <std::floating_point T>
00138 bool isCounterClockwise(Trian2<T> &tr);
00139
00140 template <std::floating_point T>
00141 Segment<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d);
00142
00143 } // namespace detail
00144 } // namespace geom
00145
00146 namespace geom
00147 {
00148
00149 template <std::floating_point T>
00150 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
00151 {
00152     /* TODO: handle invalid triangles case */
00153     auto isInv1 = !tr1.isValid();
00154     auto isInv2 = !tr2.isValid();
00155
00156     if (isInv1 && isInv2)
00157         return detail::isIntersectBothInvalid(tr1, tr2);
00158
00159     if (isInv1)
00160         return detail::isIntersectValidInvalid(tr2, tr1);
00161
00162     if (isInv2)
00163         return detail::isIntersectValidInvalid(tr1, tr2);
00164
00165     auto p11 = tr1.getPlane();
00166     if (detail::isOnOneSide(p11, tr2))
00167         return false;
00168
00169     auto p12 = tr2.getPlane();
00170     if (p11 == p12)
00171         return detail::isIntersect2D(tr1, tr2);
00172
00173     if (p11.isPar(p12))
00174         return false;
00175
00176     if (detail::isOnOneSide(p12, tr1))
00177         return false;
00178
00179     return detail::isIntersectMollerHaines(tr1, tr2);
00180 }
00181
00182 template <std::floating_point T>
00183 std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &p11,
00184                                                         const Plane<T> &p12)
00185 {
00186     const auto &n1 = p11.norm();
00187     const auto &n2 = p12.norm();
00188
00189     auto dir = cross(n1, n2);
00190
00191     /* if planes are parallel */
00192     if (Vec3<T>{0} == dir)
00193     {
00194         if (p11 == p12)
00195             return p11;
00196
00197         return std::monostate{};
00198     }
00199
00200     auto n1n2 = dot(n1, n2);
00201     auto d1 = p11.dist();
00202     auto d2 = p12.dist();
00203
00204     auto a = (d2 * n1n2 - d1) / (n1n2 * n1n2 - 1);
00205     auto b = (d1 * n1n2 - d2) / (n1n2 * n1n2 - 1);

```

```

00206
00207     return Line<T>{(a * n1) + (b * n2), dir};
00208 }
00209
00210 namespace detail
00211 {
00212
00213 template <std::floating_point T>
00214 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
00215 {
00216     auto pl = tr1.getPlane();
00217
00218     auto trian1 = getTrian2(pl, tr1);
00219     auto trian2 = getTrian2(pl, tr2);
00220
00221     for (auto trian : {trian1, trian2})
00222     {
00223         for (size_t i0 = 0, i1 = 2; i0 < 3; i1 = i0, ++i0)
00224         {
00225             auto d = (trian[i0] - trian[i1]).getPerp();
00226
00227             auto s1 = computeInterval(trian1, d);
00228             auto s2 = computeInterval(trian2, d);
00229
00230             if (s2.second < s1.first || s1.second < s2.first)
00231                 return false;
00232         }
00233     }
00234
00235     return true;
00236 }
00237
00238 template <std::floating_point T>
00239 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
00240 {
00241     auto pl1 = tr1.getPlane();
00242     auto pl2 = tr2.getPlane();
00243
00244     auto l = std::get<Line<T>>(intersect(pl1, pl2));
00245
00246     auto params1 = helperMollerHaines(tr1, pl2, l);
00247     auto params2 = helperMollerHaines(tr2, pl1, l);
00248
00249     return isOverlap(params1, params2);
00250 }
00251
00252 template <std::floating_point T>
00253 Segment<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
00254 {
00255     /* Project the triangle vertices onto line */
00256     std::array<T, 3> vert{};
00257     for (size_t i = 0; i < 3; ++i)
00258         vert[i] = dot(l.dir(), tr[i] - l.org());
00259
00260     std::array<T, 3> sdist{};
00261     for (size_t i = 0; i < 3; ++i)
00262         sdist[i] = distance(pl, tr[i]);
00263
00264     std::array<bool, 3> isOneSide{};
00265     for (size_t i = 0; i < 3; ++i)
00266         isOneSide[i] = (sdist[i] * sdist[(i + 1) % 3] > 0);
00267
00268     /* Looking for vertex which is alone on it's side */
00269     size_t rogue = 0;
00270     for (size_t i = 0; i < 3; ++i)
00271         if (isOneSide[i])
00272             rogue = (i + 2) % 3;
00273
00274     std::vector<T> segm{};
00275     std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};
00276
00277     for (size_t i : arr)
00278         segm.push_back(vert[i] +
00279             (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]));
00280
00281     /* Sort segment's ends */
00282     if (segm[0] > segm[1])
00283         std::swap(segm[0], segm[1]);
00284
00285     return {segm[0], segm[1]};
00286 }
00287
00288 template <std::floating_point T>
00289 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
00290 {
00291     return false;
00292 }

```

```

00293
00294 template <std::floating_point T>
00295 bool isIntersectValidInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
00296 {
00297     return false;
00298 }
00299
00300 template <std::floating_point T>
00301 bool isOverlap(Segment<T> &segm1, Segment<T> &segm2)
00302 {
00303     return (segm2.first <= segm1.second) && (segm2.second >= segm1.first);
00304 }
00305
00306 template <std::forward_iterator It>
00307 bool isSameSign(It begin, It end)
00308 {
00309     auto cur = begin;
00310     auto prev = begin;
00311
00312     for (++cur; cur != end; ++cur)
00313         if ((*cur) * (*prev) <= 0)
00314             return false;
00315
00316     return true;
00317 }
00318
00319 template <std::floating_point T>
00320 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
00321 {
00322     std::array<T, 3> sdist{};
00323     for (size_t i = 0; i < 3; ++i)
00324         sdist[i] = distance(pl, tr[i]);
00325
00326     if (detail::isSameSign(sdist.begin(), sdist.end()))
00327         return true;
00328
00329     return false;
00330 }
00331
00332 template <std::floating_point T>
00333 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr)
00334 {
00335     auto norm = pl.norm();
00336
00337     const Vec3<T> x{1, 0, 0};
00338     const Vec3<T> y{0, 1, 0};
00339     const Vec3<T> z{0, 0, 1};
00340
00341     std::array<Vec3<T>, 3> xyz{x, y, z};
00342     std::array<T, 3> xyzDot;
00343
00344     std::transform(xyz.begin(), xyz.end(), xyzDot.begin(),
00345         [&norm](const auto &axis) { return std::abs(dot(axis, norm)); });
00346
00347     auto maxIt = std::max_element(xyzDot.begin(), xyzDot.end());
00348     auto maxIdx = static_cast<size_t>(maxIt - xyzDot.begin());
00349
00350     Trian2<T> res;
00351     for (size_t i = 0; i < 3; ++i)
00352         for (size_t j = 0, k = 0; j < 2; ++j, ++k)
00353             {
00354                 if (k == maxIdx)
00355                     ++k;
00356
00357                 res[i][j] = tr[i][k];
00358             }
00359
00360     if (!isCounterClockwise(res))
00361         std::swap(res[0], res[1]);
00362
00363     return res;
00364 }
00365
00366 template <std::floating_point T>
00367 bool isCounterClockwise(Trian2<T> &tr)
00368 {
00369     /**
00370      * The triangle is counterclockwise ordered if  $\Delta > 0$ 
00371      * and clockwise ordered if  $\Delta < 0$ .
00372      *
00373      *  $\Delta = \det \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \end{vmatrix} = (x_1 * y_2 - x_2 * y_1) - (x_0 * y_2 - x_2 * y_0) + (x_0 * y_1 - x_1 * y_0)$ 
00374      *
00375      */
00376
00377     auto x0 = tr[0][0], x1 = tr[1][0], x2 = tr[2][0];

```

```

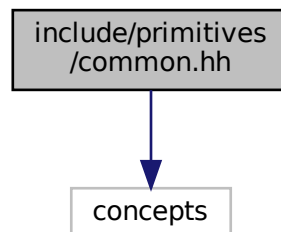
00380     auto y0 = tr[0][1], y1 = tr[1][1], y2 = tr[2][1];
00381
00382     auto delta = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0) + (x0 * y1 - x1 * y0);
00383     return (delta > 0);
00384 }
00385
00386 template <std::floating_point T>
00387 Segment<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
00388 {
00389     auto init = dot(d, tr[0]);
00390     auto min = init;
00391     auto max = init;
00392
00393     for (size_t i = 1; i < 3; ++i)
00394         if (auto val = dot(d, tr[i]); val < min)
00395             min = val;
00396         else if (val > max)
00397             max = val;
00398
00399     return {min, max};
00400 }
00401
00402 } // namespace detail
00403 } // namespace geom
00404
00405 #endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__

```

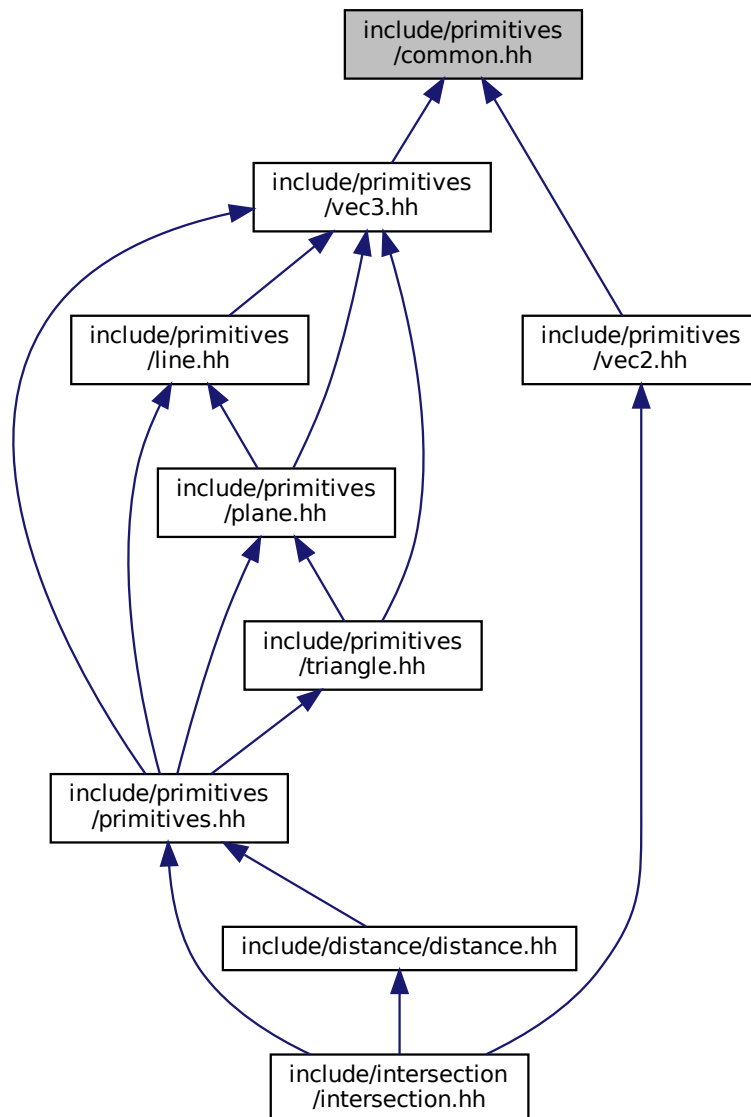
## 6.5 include/primitives/common.hh File Reference

#include <concepts>

Include dependency graph for common.hh:



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



## Namespaces

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

## 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.*



## 6.6 common.hh

```

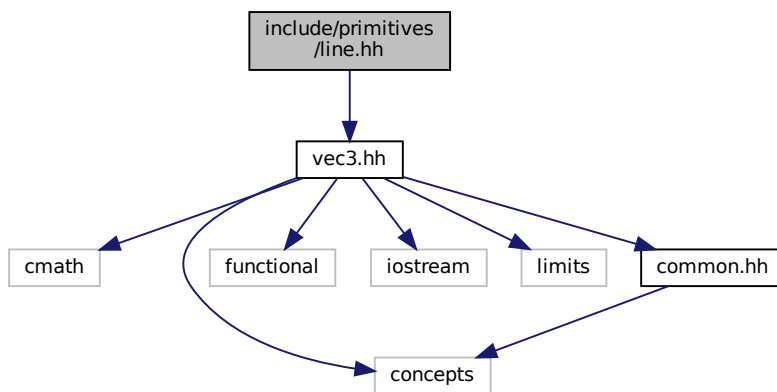
00001 #ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
00002 #define __INCLUDE_PRIMITIVES_COMMON_HH__
00003
00004 #include <concepts>
00005
00006 namespace geom
00007 {
00008 /**
00009  * @concept Number
00010  * @brief Useful concept which represents floating point and integral types
00011  *
00012  * @tparam T
00013  */
00014 template <class T>
00015 concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>;
00016
00017 } // namespace geom
00018
00019 #endif // __INCLUDE_PRIMITIVES_COMMON_HH__

```

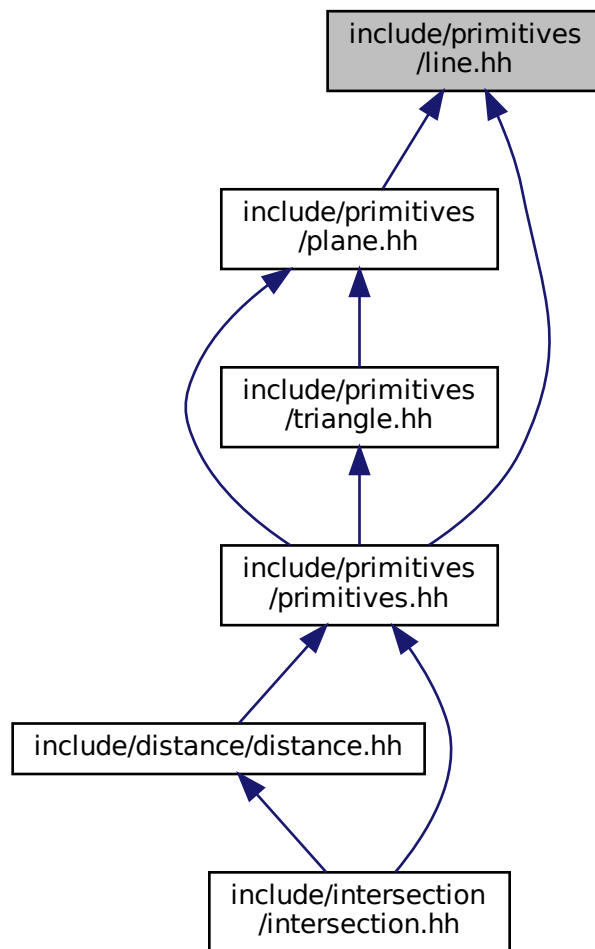
## 6.7 include/primitives/line.hh File Reference

```
#include "vec3.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.*

## 6.8 line.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_LINE_HH__
00002 #define __INCLUDE_PRIMITIVES_LINE_HH__
00003
00004 #include "vec3.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 Vec3<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 Vec3<T> &org, const Vec3<T> &dir);
00037
00038 /**
00039  * @brief Getter for origin vector
00040  *
00041  * @return const Vec3<T>& const reference to origin vector
00042  */
00043 const Vec3<T> &org() const;
00044
00045 /**
00046  * @brief Getter for direction vector
00047  *
00048  * @return const Vec3<T>& const reference to direction vector
00049  */
00050 const Vec3<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 Vec3<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 Vec3<T> &p1, const Vec3<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 Vec3<T> &org, const Vec3<T> &dir) : org_{org}, dir_{dir}
00112 {
00113     if (dir_ == Vec3<T>{0})
00114         throw std::logic_error{"Direction vector equals zero."};
00115 }
00116
00117 template <std::floating_point T>
00118 const Vec3<T> &Line<T>::org() const
00119 {
00120     return org_;
00121 }
00122
00123 template <std::floating_point T>
00124 const Vec3<T> &Line<T>::dir() const
00125 {
00126     return dir_;
00127 }
00128
00129 template <std::floating_point T>
00130 bool Line<T>::belongs(const Vec3<T> &point) const
00131 {
00132     return dir_.cross(point - org_) == Vec3<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 Vec3<T> &p1, const Vec3<T> &p2)
00143 {
00144     return Line<T>{p1, p2 - p1};
00145 }
00146
00147 } // namespace geom
00148
00149 #endif // __INCLUDE_PRIMITIVES_LINE_HH__

```

## 6.9 include/primitives/plane.hh File Reference

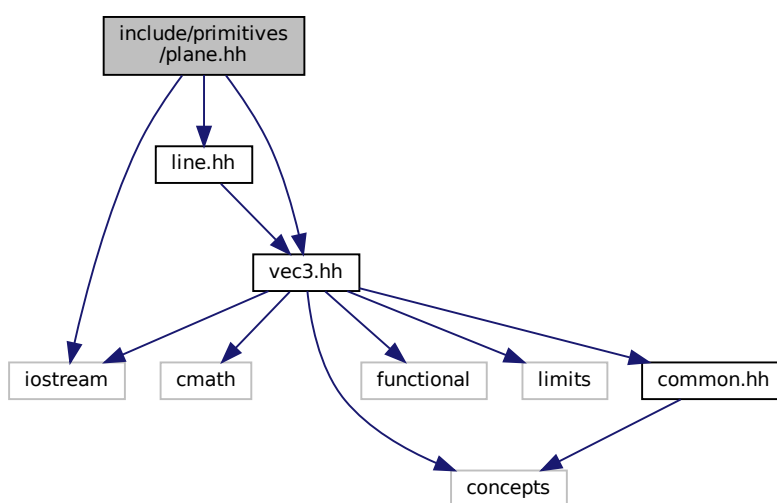
```

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

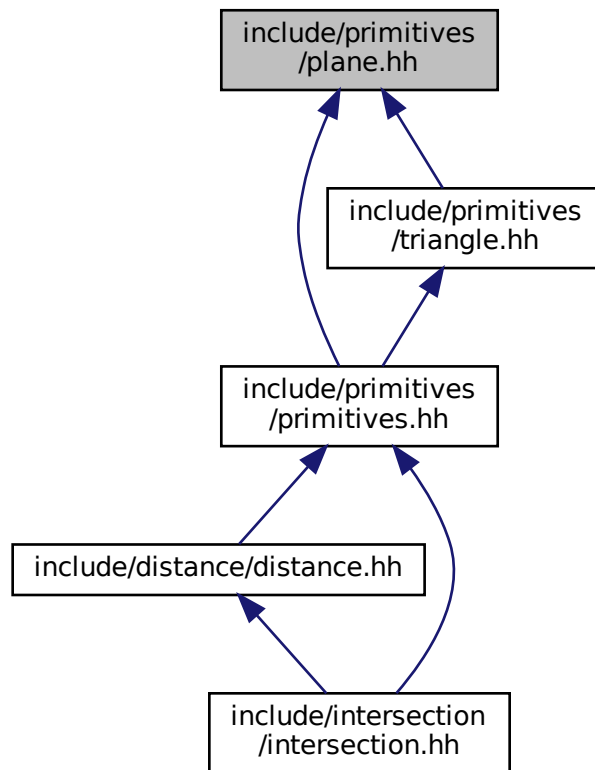
```

```
#include "vec3.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.

## 6.10 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 "vec3.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     Vec3<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 Vec3<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 Vec3<T>& const reference to normal vector
00057      */
00058     const Vec3<T> &norm() const;
00059
00060     /**
00061      * @brief Checks if point belongs to plane
00062      *
00063      * @param[in] point const referene 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 Vec3<T> &point) const;
00068
00069     /**
00070      * @brief Checks if line belongs to plane
00071      *
00072      * @param[in] line const referene 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 is *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 Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3);
00105
00106 /**
00107  * @brief Get plane from parametric plane equation
00108  *
00109  * @param[in] org origin vector
00110  * @param[in] dir1 1st direction vector
00111  * @param[in] dir2 2nd direction vector
00112  * @return Plane
00113  */
00114 static Plane getParametric(const Vec3<T> &org, const Vec3<T> &dir1,
00115                             const Vec3<T> &dir2);
00116
00117 /**
00118  * @brief Get plane from normal point plane equation
00119  *
00120  * @param[in] norm normal vector
00121  * @param[in] point point lying on the plane
00122  * @return Plane
00123  */
00124 static Plane getNormalPoint(const Vec3<T> &norm, const Vec3<T> &point);
00125
00126 /**
00127  * @brief Get plane from normal const plane equation
00128  *
00129  * @param[in] norm normal vector
00130  * @param[in] constant distance
00131  * @return Plane
00132  */
00133 static Plane getNormalDist(const Vec3<T> &norm, T constant);
00134 };
00135
00136 /**
00137  * @brief Plane equality operator
00138  *
00139  * @tparam T - floating point type of coordinates
00140  * @param[in] lhs 1st plane
00141  * @param[in] rhs 2nd plane
00142  * @return true if planes are equal
00143  * @return false if planes are not equal
00144  */
00145 template <std::floating_point T>
00146 bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
00147 {
00148     return lhs.isEqual(rhs);
00149 }
00150
00151 /**
00152  * @brief Plane print operator
00153  *
00154  * @tparam T - floating point type of coordinates
00155  * @param[in, out] ost output stream
00156  * @param[in] pl plane to print
00157  * @return std::ostream& modified ostream instance
00158  */
00159 template <std::floating_point T>
00160 std::ostream &operator<<(std::ostream &ost, const Plane<T> &pl)
00161 {
00162     ost << pl.norm() << " * X = " << pl.dist();
00163     return ost;
00164 }
00165
00166 template <std::floating_point T>
00167 Plane<T>::Plane(const Vec3<T> &norm, T dist) : norm_(norm), dist_(dist)
00168 {
00169     if (norm == Vec3<T>{0})
00170         throw std::logic_error{"normal vector equals to zero"};
00171 }
00172

```



```

00173 template <std::floating_point T>
00174 T Plane<T>::dist() const
00175 {
00176     return dist_;
00177 }
00178
00179 template <std::floating_point T>
00180 const Vec3<T> &Plane<T>::norm() const
00181 {
00182     return norm_;
00183 }
00184
00185 template <std::floating_point T>
00186 bool Plane<T>::belongs(const Vec3<T> &pt) const
00187 {
00188     return Vec3<T>::isNumEq(norm_.dot(pt), dist_);
00189 }
00190
00191 template <std::floating_point T>
00192 bool Plane<T>::belongs(const Line<T> &line) const
00193 {
00194     return norm_.isPerp(line.dir()) && belongs(line.org());
00195 }
00196
00197 template <std::floating_point T>
00198 bool Plane<T>::isEqual(const Plane &rhs) const
00199 {
00200     return (norm_ * dist_ == rhs.norm_ * rhs.dist_) && (norm_.isPar(rhs.norm_));
00201 }
00202
00203 template <std::floating_point T>
00204 bool Plane<T>::isPar(const Plane &rhs) const
00205 {
00206     return norm_.isPar(rhs.norm_);
00207 }
00208
00209 template <std::floating_point T>
00210 Plane<T> Plane<T>::getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2,
00211                               const Vec3<T> &pt3)
00212 {
00213     return getParametric(pt1, pt2 - pt1, pt3 - pt1);
00214 }
00215
00216 template <std::floating_point T>
00217 Plane<T> Plane<T>::getParametric(const Vec3<T> &org, const Vec3<T> &dir1,
00218                                 const Vec3<T> &dir2)
00219 {
00220     auto norm = dir1.cross(dir2);
00221     return getNormalPoint(norm, org);
00222 }
00223
00224 template <std::floating_point T>
00225 Plane<T> Plane<T>::getNormalPoint(const Vec3<T> &norm, const Vec3<T> &pt)
00226 {
00227     auto normalized = norm.normalized();
00228     return Plane(normalized, normalized.dot(pt));
00229 }
00230
00231 template <std::floating_point T>
00232 Plane<T> Plane<T>::getNormalDist(const Vec3<T> &norm, T dist)
00233 {
00234     auto normalized = norm.normalized();
00235     return Plane(normalized, dist);
00236 }
00237
00238 } // namespace geom
00239
00240 #endif // __INCLUDE_PRIMITIVES_PLANE_HH__

```

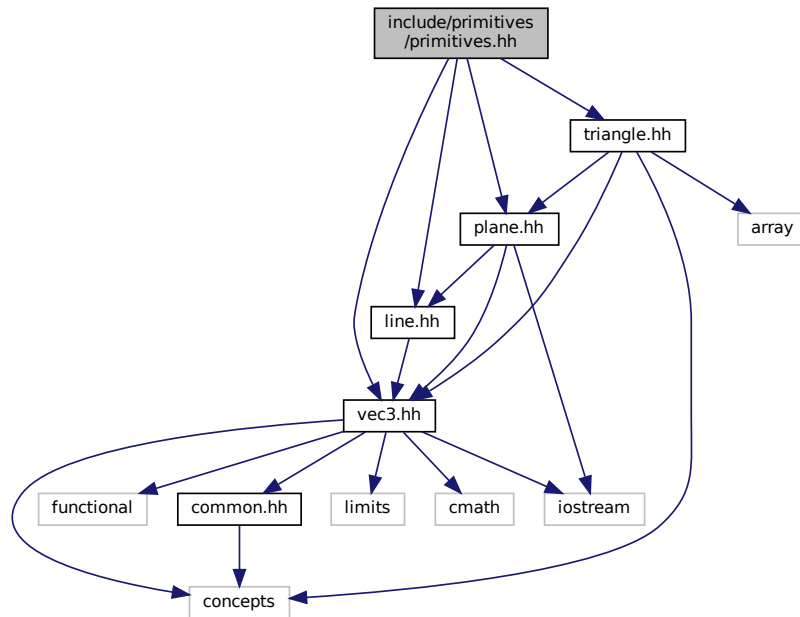
## 6.11 include/primitives/primitives.hh File Reference

```

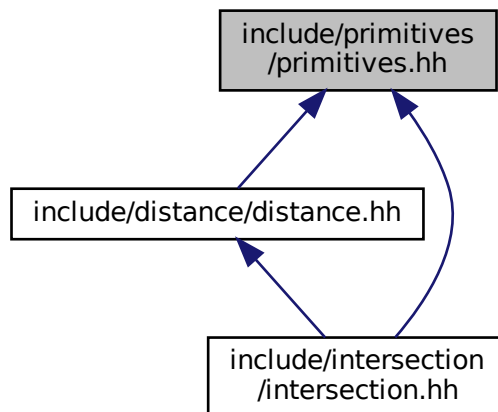
#include "line.hh"
#include "plane.hh"
#include "triangle.hh"
#include "vec3.hh"

```

Include dependency graph for primitives.hh:



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



## 6.12 primitives.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00002 #define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00003
00004 #include "line.hh"

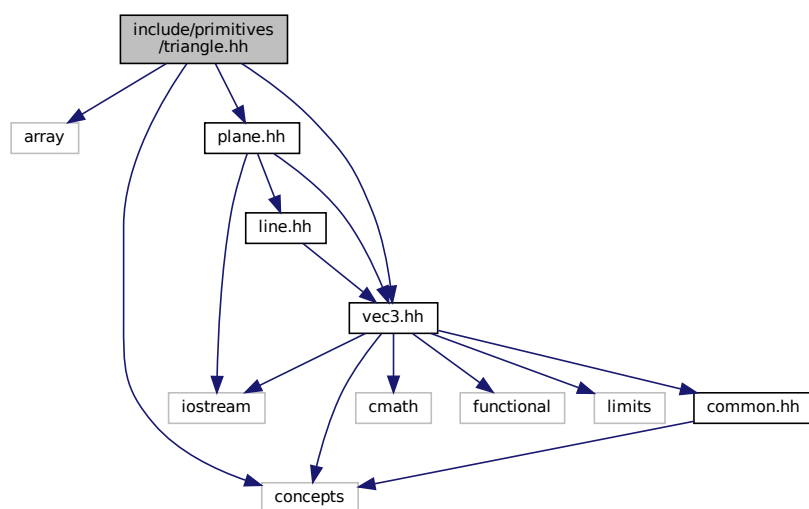
```

```
00005 #include "plane.hh"
00006 #include "triangle.hh"
00007 #include "vec3.hh"
00008
00009 #endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

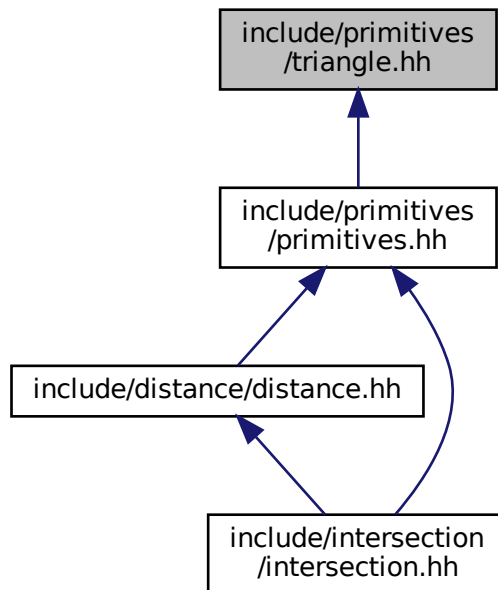
## 6.13 include/primitives/triangle.hh File Reference

```
#include <array>
#include <concepts>
#include "plane.hh"
#include "vec3.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.*

## 6.14 triangle.hh

```

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

```

```

00008 #include "vec3.hh"
00009
00010 /**
00011  * @brief triangle.hh
00012  * Triangle class implementation
00013  */
00014
00015 namespace geom
00016 {
00017
00018 /**
00019  * @class Triangle
00020  * @brief Triangle class implementation
00021  *
00022  * @tparam T - floating point type of coordinates
00023  */
00024 template <std::floating_point T>
00025 class Triangle final
00026 {
00027 private:
00028     /**
00029      * @brief Vertices of triangle
00030      */
00031     std::array<Vec3<T>, 3> vertices_;
00032
00033 public:
00034     /**
00035      * @brief Construct a new Triangle object from 3 points
00036      *
00037      * @param[in] p1 1st point
00038      * @param[in] p2 2nd point
00039      * @param[in] p3 3rd point
00040      */
00041     Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3);
00042
00043     /**
00044      * @brief Overloaded operator[] to get access to vertices
00045      *
00046      * @param[in] idx index of vertex
00047      * @return const Vec3<T>& const reference to vertex
00048      */
00049     const Vec3<T> &operator[](std::size_t idx) const;
00050
00051     /**
00052      * @brief Get triangle's plane
00053      *
00054      * @return Plane<T>
00055      */
00056     Plane<T> getPlane() const;
00057
00058     /**
00059      * @brief Check is triangle valid
00060      *
00061      * @return true if triangle is valid
00062      * @return false if triangle is invalid
00063      */
00064     bool isValid() const;
00065 };
00066
00067 /**
00068  * @brief Triangle print operator
00069  *
00070  * @tparam T - floating point type of coordinates
00071  * @param[in, out] ost output stream
00072  * @param[in] tr Triangle to print
00073  * @return std::ostream& modified ostream instance
00074  */
00075 template <std::floating_point T>
00076 std::ostream &operator<<(std::ostream &ost, const Triangle<T> &tr)
00077 {
00078     ost << "Triangle: {";
00079     for (size_t i : {0, 1, 2})
00080         ost << tr[i] << (i == 2 ? " " : ", ");
00081
00082     ost << "}";
00083
00084     return ost;
00085 }
00086
00087 template <std::floating_point T>
00088 Triangle<T>::Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3)
00089     : vertices_{p1, p2, p3}
00090 {}
00091
00092 template <std::floating_point T>
00093 const Vec3<T> &Triangle<T>::operator[](std::size_t idx) const
00094 {

```

```

00095     return vertices_[idx % 3];
00096 }
00097
00098 template <std::floating_point T>
00099 Plane<T> Triangle<T>::getPlane() const
00100 {
00101     return Plane<T>::getBy3Points(vertices_[0], vertices_[1], vertices_[2]);
00102 }
00103
00104 template <std::floating_point T>
00105 bool Triangle<T>::isValid() const
00106 {
00107     auto edge1 = vertices_[1] - vertices_[0];
00108     auto edge2 = vertices_[2] - vertices_[0];
00109
00110     auto cross12 = cross(edge1, edge2);
00111     return (cross12 != Vec3<T>{});
00112 }
00113
00114 } // namespace geom
00115
00116 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__

```

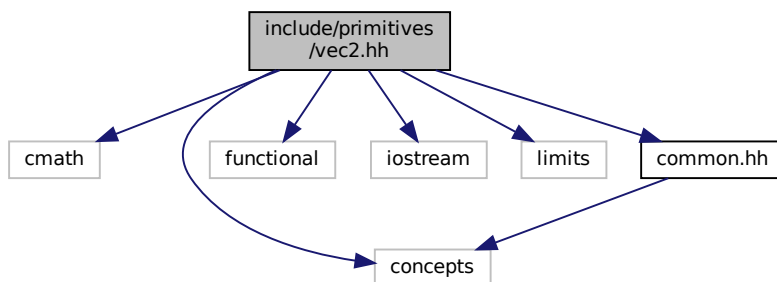
## 6.15 include/primitives/vec2.hh File Reference

```

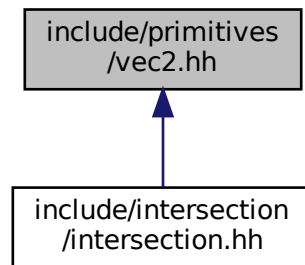
#include <cmath>
#include <concepts>
#include <functional>
#include <iostream>
#include <limits>
#include "common.hh"

```

Include dependency graph for vec2.hh:



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



## Classes

- class [geom::Vec2< T >](#)  
*Vec2 class realization.*

## Namespaces

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

## Typedefs

- using [geom::Vec2D](#) = Vec2< double >
- using [geom::Vec2F](#) = Vec2< float >

## Functions

- template<std::floating\_point T>  
Vec2< T > [geom::operator+](#) (const Vec2< T > &lhs, const Vec2< T > &rhs)  
*Overloaded + operator.*
- template<std::floating\_point T>  
Vec2< T > [geom::operator-](#) (const Vec2< T > &lhs, const Vec2< T > &rhs)  
*Overloaded - operator.*
- template<Number nT, std::floating\_point T>  
Vec2< T > [geom::operator\\*](#) (const nT &val, const Vec2< T > &rhs)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
Vec2< T > [geom::operator\\*](#) (const Vec2< T > &lhs, const nT &val)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
Vec2< T > [geom::operator/](#) (const Vec2< T > &lhs, const nT &val)  
*Overloaded divide by value operator.*

- `template<std::floating_point T>`  
`T geom::dot (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Dot product function.*
- `template<std::floating_point T>`  
`bool geom::operator== (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Vec2 equality operator.*
- `template<std::floating_point T>`  
`bool geom::operator!= (const Vec2< T > &lhs, const Vec2< T > &rhs)`  
*Vec2 inequality operator.*
- `template<std::floating_point T>`  
`std::ostream & geom::operator<< (std::ostream &ost, const Vec2< T > &vec)`  
*Vec2 print operator.*

### 6.15.1 Detailed Description

Vec2 class implementation

Definition in file [vec2.hh](#).

## 6.16 vec2.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_VEC2_HH__
00002 #define __INCLUDE_PRIMITIVES_VEC2_HH__
00003
00004 #include <cmath>
00005 #include <concepts>
00006 #include <functional>
00007 #include <iostream>
00008 #include <limits>
00009
00010 #include "common.hh"
00011
00012 /**
00013  * @file vec2.hh
00014  * Vec2 class implementation
00015  */
00016
00017 namespace geom
00018 {
00019
00020 /**
00021  * @class Vec2
00022  * @brief Vec2 class realization
00023  *
00024  * @tparam T - floating point type of coordinates
00025  */
00026 template <std::floating_point T>
00027 struct Vec2 final
00028 {
00029 private:
00030     /**
00031      * @brief Threshold static variable for numbers comparision
00032      */
00033     static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00034
00035 public:
00036     /**
00037      * @brief Vec2 coordinates
00038      */
00039     T x{}, y{};
00040
00041     /**
00042      * @brief Construct a new Vec2 object from 3 coordinates
00043      *
00044      * @param[in] coordX x coordinate
00045      * @param[in] coordY y coordinate
00046      */
00047     Vec2(T coordX, T coordY) : x(coordX), y(coordY)
00048     {}
00049

```



```

00050  /**
00051   * @brief Construct a new Vec2 object with equals coordinates
00052   *
00053   * @param[in] coordX coordinate (default to {})
00054   */
00055  explicit Vec2(T coordX = {}) : Vec2(coordX, coordX)
00056  {}
00057
00058  /**
00059   * @brief Overloaded += operator
00060   * Increments vector coordinates by corresponding coordinates of vec
00061   * @param[in] vec vector to incremented with
00062   * @return Vec2& reference to current instance
00063   */
00064  Vec2 &operator+=(const Vec2 &vec);
00065
00066  /**
00067   * @brief Overloaded -= operator
00068   * Decrements vector coordinates by corresponding coordinates of vec
00069   * @param[in] vec vector to decremented with
00070   * @return Vec2& reference to current instance
00071   */
00072  Vec2 &operator-=(const Vec2 &vec);
00073
00074  /**
00075   * @brief Unary - operator
00076   *
00077   * @return Vec2 negated Vec2 instance
00078   */
00079  Vec2 operator-() const;
00080
00081  /**
00082   * @brief Overloaded *= by number operator
00083   *
00084   * @tparam nType numeric type of value to multiply by
00085   * @param[in] val value to multiply by
00086   * @return Vec2& reference to vector instance
00087   */
00088  template <Number nType>
00089  Vec2 &operator*=(nType val);
00090
00091  /**
00092   * @brief Overloaded /= by number operator
00093   *
00094   * @tparam nType numeric type of value to divide by
00095   * @param[in] val value to divide by
00096   * @return Vec2& reference to vector instance
00097   *
00098   * @warning Does not check if val equals 0
00099   */
00100  template <Number nType>
00101  Vec2 &operator/=(nType val);
00102
00103  /**
00104   * @brief Dot product function
00105   *
00106   * @param rhs vector to dot product with
00107   * @return T dot product of two vectors
00108   */
00109  T dot(const Vec2 &rhs) const;
00110
00111  /**
00112   * @brief Calculate squared length of a vector function
00113   *
00114   * @return T length^2
00115   */
00116  T length2() const;
00117
00118  /**
00119   * @brief Calculate length of a vector function
00120   *
00121   * @return T length
00122   */
00123  T length() const;
00124
00125  /**
00126   * @brief Get the perpendicular to this vector
00127   *
00128   * @return Vec2 perpendicular vector
00129   */
00130  Vec2 getPerp() const;
00131
00132  /**
00133   * @brief Get normalized vector function
00134   *
00135   * @return Vec2 normalized vector
00136   */

```

```

00137 Vec2 normalized() const;
00138
00139 /**
00140  * @brief Normalize vector function
00141  *
00142  * @return Vec2& reference to instance
00143  */
00144 Vec2 &normalize();
00145
00146 /**
00147  * @brief Overloaded operator [] (non-const version)
00148  * To get access to coordinates
00149  * @param i index of coordinate (0 - x, 1 - y)
00150  * @return T& reference to coordinate value
00151  *
00152  * @note Coordinates calculated by mod 2
00153  */
00154 T &operator[](size_t i);
00155
00156 /**
00157  * @brief Overloaded operator [] (const version)
00158  * To get access to coordinates
00159  * @param i index of coordinate (0 - x, 1 - y)
00160  * @return T coordinate value
00161  *
00162  * @note Coordinates calculated by mod 2
00163  */
00164 T operator[](size_t i) const;
00165
00166 /**
00167  * @brief Check if vector is parallel to another
00168  *
00169  * @param[in] rhs vector to check parallelism with
00170  * @return true if vector is parallel
00171  * @return false otherwise
00172  */
00173 bool isPar(const Vec2 &rhs) const;
00174
00175 /**
00176  * @brief Check if vector is perpendicular to another
00177  *
00178  * @param[in] rhs vector to check perpendicularity with
00179  * @return true if vector is perpendicular
00180  * @return false otherwise
00181  */
00182 bool isPerp(const Vec2 &rhs) const;
00183
00184 /**
00185  * @brief Check if vector is equal to another
00186  *
00187  * @param[in] rhs vector to check equality with
00188  * @return true if vector is equal
00189  * @return false otherwise
00190  *
00191  * @note Equality check performs using isNumEq(T lhs, T rhs) function
00192  */
00193 bool isEqual(const Vec2 &rhs) const;
00194
00195 /**
00196  * @brief Check equality (with threshold) of two floating point numbers function
00197  *
00198  * @param[in] lhs first number
00199  * @param[in] rhs second number
00200  * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
00201  * @return false otherwise
00202  *
00203  * @note Threshold defined by threshold_ static member
00204  */
00205 static bool isNumEq(T lhs, T rhs);
00206
00207 /**
00208  * @brief Set new threshold value
00209  *
00210  * @param[in] thres value to set
00211  */
00212 static void setThreshold(T thres);
00213
00214 /**
00215  * @brief Get current threshold value
00216  */
00217 static T getThreshold();
00218
00219 /**
00220  * @brief Set threshold to default value
00221  * @note default value equals float point epsilon
00222  */
00223 static void setDefThreshold();

```

```

00224 };
00225
00226 /**
00227  * @brief Overloaded + operator
00228  *
00229  * @tparam T vector template parameter
00230  * @param[in] lhs first vector
00231  * @param[in] rhs second vector
00232  * @return Vec2<T> sum of two vectors
00233  */
00234 template <std::floating_point T>
00235 Vec2<T> operator+(const Vec2<T> &lhs, const Vec2<T> &rhs)
00236 {
00237     Vec2<T> res{lhs};
00238     res += rhs;
00239     return res;
00240 }
00241
00242 /**
00243  * @brief Overloaded - operator
00244  *
00245  * @tparam T vector template parameter
00246  * @param[in] lhs first vector
00247  * @param[in] rhs second vector
00248  * @return Vec2<T> res of two vectors
00249  */
00250 template <std::floating_point T>
00251 Vec2<T> operator-(const Vec2<T> &lhs, const Vec2<T> &rhs)
00252 {
00253     Vec2<T> res{lhs};
00254     res -= rhs;
00255     return res;
00256 }
00257
00258 /**
00259  * @brief Overloaded multiple by value operator
00260  *
00261  * @tparam nT type of value to multiply by
00262  * @tparam T vector template parameter
00263  * @param[in] val value to multiply by
00264  * @param[in] rhs vector to multiply by value
00265  * @return Vec2<T> result vector
00266  */
00267 template <Number nT, std::floating_point T>
00268 Vec2<T> operator*(const nT &val, const Vec2<T> &rhs)
00269 {
00270     Vec2<T> res{rhs};
00271     res *= val;
00272     return res;
00273 }
00274
00275 /**
00276  * @brief Overloaded multiple by value operator
00277  *
00278  * @tparam nT type of value to multiply by
00279  * @tparam T vector template parameter
00280  * @param[in] val value to multiply by
00281  * @param[in] lhs vector to multiply by value
00282  * @return Vec2<T> result vector
00283  */
00284 template <Number nT, std::floating_point T>
00285 Vec2<T> operator*(const Vec2<T> &lhs, const nT &val)
00286 {
00287     Vec2<T> res{lhs};
00288     res *= val;
00289     return res;
00290 }
00291
00292 /**
00293  * @brief Overloaded divide by value operator
00294  *
00295  * @tparam nT type of value to divide by
00296  * @tparam T vector template parameter
00297  * @param[in] val value to divide by
00298  * @param[in] lhs vector to divide by value
00299  * @return Vec2<T> result vector
00300  */
00301 template <Number nT, std::floating_point T>
00302 Vec2<T> operator/(const Vec2<T> &lhs, const nT &val)
00303 {
00304     Vec2<T> res{lhs};
00305     res /= val;
00306     return res;
00307 }
00308
00309 /**
00310  * @brief Dot product function

```

```

00311 *
00312 * @tparam T vector template parameter
00313 * @param[in] lhs first vector
00314 * @param[in] rhs second vector
00315 * @return T dot production
00316 */
00317 template <std::floating_point T>
00318 T dot(const Vec2<T> &lhs, const Vec2<T> &rhs)
00319 {
00320     return lhs.dot(rhs);
00321 }
00322
00323 /**
00324 * @brief Vec2 equality operator
00325 *
00326 * @tparam T vector template parameter
00327 * @param[in] lhs first vector
00328 * @param[in] rhs second vector
00329 * @return true if vectors are equal
00330 * @return false otherwise
00331 */
00332 template <std::floating_point T>
00333 bool operator==(const Vec2<T> &lhs, const Vec2<T> &rhs)
00334 {
00335     return lhs.isEqual(rhs);
00336 }
00337
00338 /**
00339 * @brief Vec2 inequality operator
00340 *
00341 * @tparam T vector template parameter
00342 * @param[in] lhs first vector
00343 * @param[in] rhs second vector
00344 * @return true if vectors are not equal
00345 * @return false otherwise
00346 */
00347 template <std::floating_point T>
00348 bool operator!=(const Vec2<T> &lhs, const Vec2<T> &rhs)
00349 {
00350     return !(lhs == rhs);
00351 }
00352
00353 /**
00354 * @brief Vec2 print operator
00355 *
00356 * @tparam T vector template parameter
00357 * @param[in, out] ost output stream
00358 * @param[in] vec vector to print
00359 * @return std::ostream& modified stream instance
00360 */
00361 template <std::floating_point T>
00362 std::ostream &operator<<(std::ostream &ost, const Vec2<T> &vec)
00363 {
00364     ost << "(" << vec.x << ", " << vec.y << ")";
00365     return ost;
00366 }
00367
00368 using Vec2D = Vec2<double>;
00369 using Vec2F = Vec2<float>;
00370
00371 template <std::floating_point T>
00372 Vec2<T> &Vec2<T>::operator+=(const Vec2 &vec)
00373 {
00374     x += vec.x;
00375     y += vec.y;
00376     return *this;
00377 }
00378
00379 template <std::floating_point T>
00380 Vec2<T> &Vec2<T>::operator-=(const Vec2 &vec)
00381 {
00382     x -= vec.x;
00383     y -= vec.y;
00384     return *this;
00385 }
00386
00387 template <std::floating_point T>
00388 Vec2<T> Vec2<T>::operator-() const
00389 {
00390     return Vec2{-x, -y};
00391 }
00392
00393 template <std::floating_point T>
00394 Vec2<T> &Vec2<T>::operator*=(nType val)

```

```

00398 {
00399     x *= val;
00400     y *= val;
00401
00402     return *this;
00403 }
00404
00405 template <std::floating_point T>
00406 template <Number nType>
00407 Vec2<T> &Vec2<T>::operator/=(nType val)
00408 {
00409     x /= static_cast<T>(val);
00410     y /= static_cast<T>(val);
00411
00412     return *this;
00413 }
00414
00415 template <std::floating_point T>
00416 T Vec2<T>::dot(const Vec2 &rhs) const
00417 {
00418     return x * rhs.x + y * rhs.y;
00419 }
00420
00421 template <std::floating_point T>
00422 T Vec2<T>::length2() const
00423 {
00424     return dot(*this);
00425 }
00426
00427 template <std::floating_point T>
00428 T Vec2<T>::length() const
00429 {
00430     return std::sqrt(length2());
00431 }
00432
00433 template <std::floating_point T>
00434 Vec2<T> Vec2<T>::getPerp() const
00435 {
00436     return {y, -x};
00437 }
00438
00439 template <std::floating_point T>
00440 Vec2<T> Vec2<T>::normalized() const
00441 {
00442     Vec2 res{*this};
00443     res.normalize();
00444     return res;
00445 }
00446
00447 template <std::floating_point T>
00448 Vec2<T> &Vec2<T>::normalize()
00449 {
00450     T len2 = length2();
00451     if (isNumEq(len2, 0) || isNumEq(len2, 1))
00452         return *this;
00453     return *this /= std::sqrt(len2);
00454 }
00455
00456 template <std::floating_point T>
00457 T &Vec2<T>::operator[](size_t i)
00458 {
00459     switch (i % 3)
00460     {
00461     case 0:
00462         return x;
00463     case 1:
00464         return y;
00465     default:
00466         throw std::logic_error{"Impossible case in operator[]\n"};
00467     }
00468 }
00469
00470 template <std::floating_point T>
00471 T Vec2<T>::operator[](size_t i) const
00472 {
00473     switch (i % 3)
00474     {
00475     case 0:
00476         return x;
00477     case 1:
00478         return y;
00479     default:
00480         throw std::logic_error{"Impossible case in operator[]\n"};
00481     }
00482 }
00483
00484 template <std::floating_point T>

```

```

00485 bool Vec2<T>::isPar(const Vec2 &rhs) const
00486 {
00487     auto det = x * rhs.y - rhs.x * y;
00488     return isNumEq(det, 0);
00489 }
00490
00491 template <std::floating_point T>
00492 bool Vec2<T>::isPerp(const Vec2 &rhs) const
00493 {
00494     return isNumEq(dot(rhs), 0);
00495 }
00496
00497 template <std::floating_point T>
00498 bool Vec2<T>::isEqual(const Vec2 &rhs) const
00499 {
00500     return isNumEq(x, rhs.x) && isNumEq(y, rhs.y);
00501 }
00502
00503 template <std::floating_point T>
00504 bool Vec2<T>::isNumEq(T lhs, T rhs)
00505 {
00506     return std::abs(rhs - lhs) < threshold_;
00507 }
00508
00509 template <std::floating_point T>
00510 void Vec2<T>::setThreshold(T thres)
00511 {
00512     threshold_ = thres;
00513 }
00514
00515 template <std::floating_point T>
00516 T Vec2<T>::getThreshold()
00517 {
00518     return threshold_;
00519 }
00520
00521 template <std::floating_point T>
00522 void Vec2<T>::setDefThreshold()
00523 {
00524     threshold_ = std::numeric_limits<T>::epsilon();
00525 }
00526
00527 } // namespace geom
00528
00529 #endif // __INCLUDE_PRIMITIVES_VEC2_HH__

```

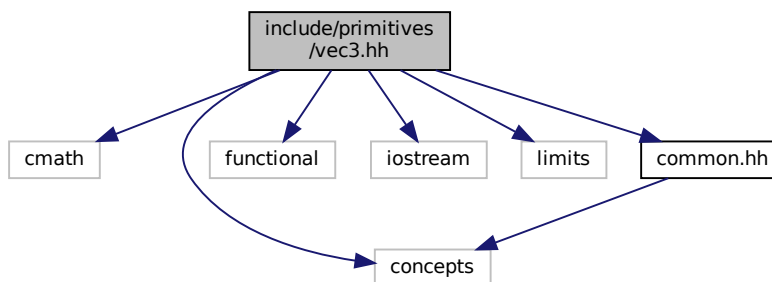
## 6.17 include/primitives/vec3.hh File Reference

```

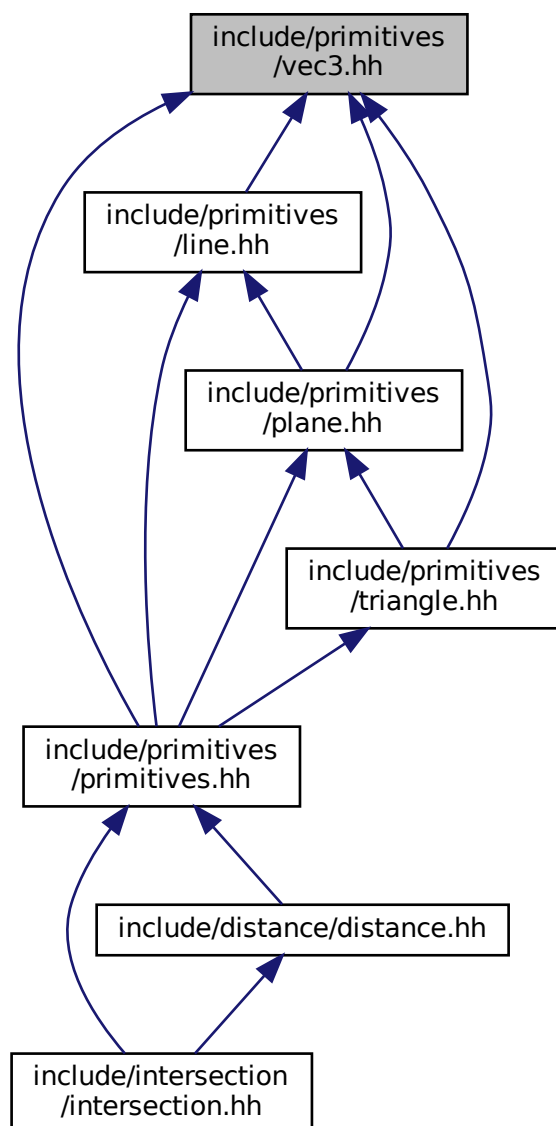
#include <cmath>
#include <concepts>
#include <functional>
#include <iostream>
#include <limits>
#include "common.hh"

```

Include dependency graph for vec3.hh:



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



## Classes

- class [geom::Vec3< T >](#)  
*Vec3 class realization.*

## Namespaces

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

## Typedefs

- using [geom::Vec3D](#) = Vec3< double >
- using [geom::Vec3F](#) = Vec3< float >

## Functions

- template<std::floating\_point T>  
Vec3< T > [geom::operator+](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Overloaded + operator.*
- template<std::floating\_point T>  
Vec3< T > [geom::operator-](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Overloaded - operator.*
- template<Number nT, std::floating\_point T>  
Vec3< T > [geom::operator\\*](#) (const nT &val, const Vec3< T > &rhs)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
Vec3< T > [geom::operator\\*](#) (const Vec3< T > &lhs, const nT &val)  
*Overloaded multiple by value operator.*
- template<Number nT, std::floating\_point T>  
Vec3< T > [geom::operator/](#) (const Vec3< T > &lhs, const nT &val)  
*Overloaded divide by value operator.*
- template<std::floating\_point T>  
T [geom::dot](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Dot product function.*
- template<std::floating\_point T>  
Vec3< T > [geom::cross](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Cross product function.*
- template<std::floating\_point T>  
bool [geom::operator==](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Vec3 equality operator.*
- template<std::floating\_point T>  
bool [geom::operator!=](#) (const Vec3< T > &lhs, const Vec3< T > &rhs)  
*Vec3 inequality operator.*
- template<std::floating\_point T>  
std::ostream & [geom::operator<<](#) (std::ostream &ost, const Vec3< T > &vec)  
*Vec3 print operator.*

### 6.17.1 Detailed Description

Vec3 class implementation

Definition in file [vec3.hh](#).



## 6.18 vec3.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_VEC3_HH__
00002 #define __INCLUDE_PRIMITIVES_VEC3_HH__
00003
00004 #include <cmath>
00005 #include <concepts>
00006 #include <functional>
00007 #include <iostream>
00008 #include <limits>
00009
00010 #include "common.hh"
00011
00012 /**
00013  * @file vec3.hh
00014  * Vec3 class implementation
00015  */
00016
00017 namespace geom
00018 {
00019
00020 /**
00021  * @class Vec3
00022  * @brief Vec3 class realization
00023  *
00024  * @tparam T - floating point type of coordinates
00025  */
00026 template <std::floating_point T>
00027 struct Vec3 final
00028 {
00029 private:
00030     /**
00031      * @brief Threshold static variable for numbers comparison
00032      */
00033     static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00034
00035 public:
00036     /**
00037      * @brief Vec3 coordinates
00038      */
00039     T x{}, y{}, z{};
00040
00041     /**
00042      * @brief Construct a new Vec3 object from 3 coordinates
00043      *
00044      * @param[in] coordX x coordinate
00045      * @param[in] coordY y coordinate
00046      * @param[in] coordZ z coordinate
00047      */
00048     Vec3(T coordX, T coordY, T coordZ) : x(coordX), y(coordY), z(coordZ)
00049     {}
00050
00051     /**
00052      * @brief Construct a new Vec3 object with equals coordinates
00053      *
00054      * @param[in] coordX coordinate (default to {})
00055      */
00056     explicit Vec3(T coordX = {}) : Vec3(coordX, coordX, coordX)
00057     {}
00058
00059     /**
00060      * @brief Overloaded += operator
00061      * Increments vector coordinates by corresponding coordinates of vec
00062      * @param[in] vec vector to incremented with
00063      * @return Vec3& reference to current instance
00064      */
00065     Vec3 &operator+=(const Vec3 &vec);
00066
00067     /**
00068      * @brief Overloaded -= operator
00069      * Decrements vector coordinates by corresponding coordinates of vec
00070      * @param[in] vec vector to decremented with
00071      * @return Vec3& reference to current instance
00072      */
00073     Vec3 &operator-=(const Vec3 &vec);
00074
00075     /**
00076      * @brief Unary - operator
00077      *
00078      * @return Vec3 negated Vec3 instance
00079      */
00080     Vec3 operator-() const;
00081
00082     /**
00083      * @brief Overloaded *= by number operator
00084      *
00085      * @tparam nType numeric type of value to multiply by

```

```

00086     * @param[in] val value to multiply by
00087     * @return Vec3& reference to vector instance
00088     */
00089     template <Number nType>
00090     Vec3 &operator*=(nType val);
00091
00092     /**
00093     * @brief Overloaded /= by number operator
00094     *
00095     * @tparam nType numeric type of value to divide by
00096     * @param[in] val value to divide by
00097     * @return Vec3& reference to vector instance
00098     *
00099     * @warning Does not check if val equals 0
00100     */
00101     template <Number nType>
00102     Vec3 &operator/=(nType val);
00103
00104     /**
00105     * @brief Dot product function
00106     *
00107     * @param rhs vector to dot product with
00108     * @return T dot product of two vectors
00109     */
00110     T dot(const Vec3 &rhs) const;
00111
00112     /**
00113     * @brief Cross product function
00114     *
00115     * @param rhs vector to cross product with
00116     * @return Vec3 cross product of two vectors
00117     */
00118     Vec3 cross(const Vec3 &rhs) const;
00119
00120     /**
00121     * @brief Calculate squared length of a vector function
00122     *
00123     * @return T length^2
00124     */
00125     T length2() const;
00126
00127     /**
00128     * @brief Calculate length of a vector function
00129     *
00130     * @return T length
00131     */
00132     T length() const;
00133
00134     /**
00135     * @brief Get normalized vector function
00136     *
00137     * @return Vec3 normalized vector
00138     */
00139     Vec3 normalized() const;
00140
00141     /**
00142     * @brief Normalize vector function
00143     *
00144     * @return Vec3& reference to instance
00145     */
00146     Vec3 &normalize();
00147
00148     /**
00149     * @brief Overloaded operator [] (non-const version)
00150     * To get access to coordinates
00151     * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00152     * @return T& reference to coordinate value
00153     *
00154     * @note Coordinates calculated by mod 3
00155     */
00156     T &operator[](size_t i);
00157
00158     /**
00159     * @brief Overloaded operator [] (const version)
00160     * To get access to coordinates
00161     * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00162     * @return T coordinate value
00163     *
00164     * @note Coordinates calculated by mod 3
00165     */
00166     T operator[](size_t i) const;
00167
00168     /**
00169     * @brief Check if vector is parallel to another
00170     *
00171     * @param[in] rhs vector to check parallelism with
00172     * @return true if vector is parallel

```

```

00173     * @return false otherwise
00174     */
00175     bool isPar(const Vec3 &rhs) const;
00176
00177     /**
00178     * @brief Check if vector is perpendicular to another
00179     *
00180     * @param[in] rhs vector to check perpendicularity with
00181     * @return true if vector is perpendicular
00182     * @return false otherwise
00183     */
00184     bool isPerp(const Vec3 &rhs) const;
00185
00186     /**
00187     * @brief Check if vector is equal to another
00188     *
00189     * @param[in] rhs vector to check equality with
00190     * @return true if vector is equal
00191     * @return false otherwise
00192     *
00193     * @note Equality check performs using isNumEq(T lhs, T rhs) function
00194     */
00195     bool isEqual(const Vec3 &rhs) const;
00196
00197     /**
00198     * @brief Check equality (with threshold) of two floating point numbers function
00199     *
00200     * @param[in] lhs first number
00201     * @param[in] rhs second number
00202     * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
00203     * @return false otherwise
00204     *
00205     * @note Threshold defined by threshold_ static member
00206     */
00207     static bool isNumEq(T lhs, T rhs);
00208
00209     /**
00210     * @brief Set new threshold value
00211     *
00212     * @param[in] thres value to set
00213     */
00214     static void setThreshold(T thres);
00215
00216     /**
00217     * @brief Get current threshold value
00218     */
00219     static T getThreshold();
00220
00221     /**
00222     * @brief Set threshold to default value
00223     * @note default value equals float point epsilon
00224     */
00225     static void setDefThreshold();
00226 };
00227
00228 /**
00229 * @brief Overloaded + operator
00230 *
00231 * @tparam T vector template parameter
00232 * @param[in] lhs first vector
00233 * @param[in] rhs second vector
00234 * @return Vec3<T> sum of two vectors
00235 */
00236 template <std::floating_point T>
00237 Vec3<T> operator+(const Vec3<T> &lhs, const Vec3<T> &rhs)
00238 {
00239     Vec3<T> res{lhs};
00240     res += rhs;
00241     return res;
00242 }
00243
00244 /**
00245 * @brief Overloaded - operator
00246 *
00247 * @tparam T vector template parameter
00248 * @param[in] lhs first vector
00249 * @param[in] rhs second vector
00250 * @return Vec3<T> res of two vectors
00251 */
00252 template <std::floating_point T>
00253 Vec3<T> operator-(const Vec3<T> &lhs, const Vec3<T> &rhs)
00254 {
00255     Vec3<T> res{lhs};
00256     res -= rhs;
00257     return res;
00258 }
00259

```

```

00260 /**
00261  * @brief Overloaded multiple by value operator
00262  *
00263  * @tparam nT type of value to multiply by
00264  * @tparam T vector template parameter
00265  * @param[in] val value to multiply by
00266  * @param[in] rhs vector to multiply by value
00267  * @return Vec3<T> result vector
00268  */
00269 template <Number nT, std::floating_point T>
00270 Vec3<T> operator*(const nT &val, const Vec3<T> &rhs)
00271 {
00272     Vec3<T> res{rhs};
00273     res *= val;
00274     return res;
00275 }
00276
00277 /**
00278  * @brief Overloaded multiple by value operator
00279  *
00280  * @tparam nT type of value to multiply by
00281  * @tparam T vector template parameter
00282  * @param[in] val value to multiply by
00283  * @param[in] lhs vector to multiply by value
00284  * @return Vec3<T> result vector
00285  */
00286 template <Number nT, std::floating_point T>
00287 Vec3<T> operator*(const Vec3<T> &lhs, const nT &val)
00288 {
00289     Vec3<T> res{lhs};
00290     res *= val;
00291     return res;
00292 }
00293
00294 /**
00295  * @brief Overloaded divide by value operator
00296  *
00297  * @tparam nT type of value to divide by
00298  * @tparam T vector template parameter
00299  * @param[in] val value to divide by
00300  * @param[in] lhs vector to divide by value
00301  * @return Vec3<T> result vector
00302  */
00303 template <Number nT, std::floating_point T>
00304 Vec3<T> operator/(const Vec3<T> &lhs, const nT &val)
00305 {
00306     Vec3<T> res{lhs};
00307     res /= val;
00308     return res;
00309 }
00310
00311 /**
00312  * @brief Dot product function
00313  *
00314  * @tparam T vector template parameter
00315  * @param[in] lhs first vector
00316  * @param[in] rhs second vector
00317  * @return T dot production
00318  */
00319 template <std::floating_point T>
00320 T dot(const Vec3<T> &lhs, const Vec3<T> &rhs)
00321 {
00322     return lhs.dot(rhs);
00323 }
00324
00325 /**
00326  * @brief Cross product function
00327  *
00328  * @tparam T vector template parameter
00329  * @param[in] lhs first vector
00330  * @param[in] rhs second vector
00331  * @return T cross production
00332  */
00333 template <std::floating_point T>
00334 Vec3<T> cross(const Vec3<T> &lhs, const Vec3<T> &rhs)
00335 {
00336     return lhs.cross(rhs);
00337 }
00338
00339 /**
00340  * @brief Vec3 equality operator
00341  *
00342  * @tparam T vector template parameter
00343  * @param[in] lhs first vector
00344  * @param[in] rhs second vector
00345  * @return true if vectors are equal
00346  * @return false otherwise

```

```

00347  */
00348  template <std::floating_point T>
00349  bool operator==(const Vec3<T> &lhs, const Vec3<T> &rhs)
00350  {
00351      return lhs.isEqual(rhs);
00352  }
00353
00354  /**
00355   * @brief Vec3 inequality operator
00356   *
00357   * @tparam T vector template parameter
00358   * @param[in] lhs first vector
00359   * @param[in] rhs second vector
00360   * @return true if vectors are not equal
00361   * @return false otherwise
00362   */
00363  template <std::floating_point T>
00364  bool operator!=(const Vec3<T> &lhs, const Vec3<T> &rhs)
00365  {
00366      return !(lhs == rhs);
00367  }
00368
00369  /**
00370   * @brief Vec3 print operator
00371   *
00372   * @tparam T vector template parameter
00373   * @param[in, out] ost output stream
00374   * @param[in] vec vector to print
00375   * @return std::ostream& modified stream instance
00376   */
00377  template <std::floating_point T>
00378  std::ostream &operator<<(std::ostream &ost, const Vec3<T> &vec)
00379  {
00380      ost << "(" << vec.x << ", " << vec.y << ", " << vec.z << ")";
00381      return ost;
00382  }
00383
00384  using Vec3D = Vec3<double>;
00385  using Vec3F = Vec3<float>;
00386
00387  template <std::floating_point T>
00388  Vec3<T> &Vec3<T>::operator+=(const Vec3 &vec)
00389  {
00390      x += vec.x;
00391      y += vec.y;
00392      z += vec.z;
00393
00394      return *this;
00395  }
00396
00397  template <std::floating_point T>
00398  Vec3<T> &Vec3<T>::operator-=(const Vec3 &vec)
00399  {
00400      x -= vec.x;
00401      y -= vec.y;
00402      z -= vec.z;
00403
00404      return *this;
00405  }
00406
00407  template <std::floating_point T>
00408  Vec3<T> Vec3<T>::operator-() const
00409  {
00410      return Vec3{-x, -y, -z};
00411  }
00412
00413  template <std::floating_point T>
00414  template <Number nType>
00415  Vec3<T> &Vec3<T>::operator*=(nType val)
00416  {
00417      x *= val;
00418      y *= val;
00419      z *= val;
00420
00421      return *this;
00422  }
00423
00424  template <std::floating_point T>
00425  template <Number nType>
00426  Vec3<T> &Vec3<T>::operator/=(nType val)
00427  {
00428      x /= static_cast<T>(val);
00429      y /= static_cast<T>(val);
00430      z /= static_cast<T>(val);
00431
00432      return *this;
00433  }

```

```

00434
00435 template <std::floating_point T>
00436 T Vec3<T>::dot(const Vec3 &rhs) const
00437 {
00438     return x * rhs.x + y * rhs.y + z * rhs.z;
00439 }
00440
00441 template <std::floating_point T>
00442 Vec3<T> Vec3<T>::cross(const Vec3 &rhs) const
00443 {
00444     return Vec3{y * rhs.z - z * rhs.y, z * rhs.x - x * rhs.z, x * rhs.y - y * rhs.x};
00445 }
00446
00447 template <std::floating_point T>
00448 T Vec3<T>::length2() const
00449 {
00450     return dot(*this);
00451 }
00452
00453 template <std::floating_point T>
00454 T Vec3<T>::length() const
00455 {
00456     return std::sqrt(length2());
00457 }
00458
00459 template <std::floating_point T>
00460 Vec3<T> Vec3<T>::normalized() const
00461 {
00462     Vec3 res{*this};
00463     res.normalize();
00464     return res;
00465 }
00466
00467 template <std::floating_point T>
00468 Vec3<T> &Vec3<T>::normalize()
00469 {
00470     T len2 = length2();
00471     if (isNumEq(len2, 0) || isNumEq(len2, 1))
00472         return *this;
00473     return *this /= std::sqrt(len2);
00474 }
00475
00476 template <std::floating_point T>
00477 T &Vec3<T>::operator[](size_t i)
00478 {
00479     switch (i % 3)
00480     {
00481     case 0:
00482         return x;
00483     case 1:
00484         return y;
00485     case 2:
00486         return z;
00487     default:
00488         throw std::logic_error{"Impossible case in operator[]\n"};
00489     }
00490 }
00491
00492 template <std::floating_point T>
00493 T Vec3<T>::operator[](size_t i) const
00494 {
00495     switch (i % 3)
00496     {
00497     case 0:
00498         return x;
00499     case 1:
00500         return y;
00501     case 2:
00502         return z;
00503     default:
00504         throw std::logic_error{"Impossible case in operator[]\n"};
00505     }
00506 }
00507
00508 template <std::floating_point T>
00509 bool Vec3<T>::isPar(const Vec3 &rhs) const
00510 {
00511     return cross(rhs).isEqual(Vec3<T>{0});
00512 }
00513
00514 template <std::floating_point T>
00515 bool Vec3<T>::isPerp(const Vec3 &rhs) const
00516 {
00517     return isNumEq(dot(rhs), 0);
00518 }
00519
00520 template <std::floating_point T>

```

```
00521 bool Vec3<T>::isEqual(const Vec3 &rhs) const
00522 {
00523     return isNumEq(x, rhs.x) && isNumEq(y, rhs.y) && isNumEq(z, rhs.z);
00524 }
00525
00526 template <std::floating_point T>
00527 bool Vec3<T>::isNumEq(T lhs, T rhs)
00528 {
00529     return std::abs(rhs - lhs) < threshold_;
00530 }
00531
00532 template <std::floating_point T>
00533 void Vec3<T>::setThreshold(T thres)
00534 {
00535     threshold_ = thres;
00536 }
00537
00538 template <std::floating_point T>
00539 T Vec3<T>::getThreshold()
00540 {
00541     return threshold_;
00542 }
00543
00544 template <std::floating_point T>
00545 void Vec3<T>::setDefThreshold()
00546 {
00547     threshold_ = std::numeric_limits<T>::epsilon();
00548 }
00549
00550 } // namespace geom
00551
00552 #endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```

