

# 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	10
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-() [1/2]	17
4.1.3.12 operator*() [1/4]	18
4.1.3.13 operator*() [2/4]	18
4.1.3.14 operator/() [1/2]	19
4.1.3.15 dot() [1/2]	19
4.1.3.16 operator==( [3/4]	20
4.1.3.17 operator!=( [1/2]	21
4.1.3.18 operator<<() [4/5]	21
4.1.3.19 operator+() [2/2]	22
4.1.3.20 operator-() [2/2]	22
4.1.3.21 operator*() [3/4]	23
4.1.3.22 operator*() [4/4]	23
4.1.3.23 operator/() [2/2]	24
4.1.3.24 dot() [2/2]	24
4.1.3.25 cross()	25

4.1.3.26 operator==( ) [ 4 / 4 ] . . . . .	26
4.1.3.27 operator!=( ) [ 2 / 2 ] . . . . .	26
4.1.3.28 operator<<( ) [ 5 / 5 ] . . . . .	27
4.1.3.29 operator>>( ) [ 2 / 2 ] . . . . .	27
4.1.4 Variable Documentation . . . . .	28
4.1.4.1 Number . . . . .	28
4.2 geom::detail Namespace Reference . . . . .	28
4.2.1 Typedef Documentation . . . . .	29
4.2.1.1 Segment2D . . . . .	29
4.2.1.2 Trian2 . . . . .	29
4.2.1.3 Segment3D . . . . .	30
4.2.2 Function Documentation . . . . .	30
4.2.2.1 isIntersect2D( ) . . . . .	30
4.2.2.2 isIntersectMollerHaines( ) . . . . .	30
4.2.2.3 helperMollerHaines( ) . . . . .	30
4.2.2.4 isIntersectBothInvalid( ) . . . . .	31
4.2.2.5 isIntersectValidInvalid( ) . . . . .	31
4.2.2.6 isIntersectPointTriangle( ) . . . . .	31
4.2.2.7 isPoint( ) . . . . .	31
4.2.2.8 isOverlap( ) . . . . .	32
4.2.2.9 isSameSign( ) . . . . .	32
4.2.2.10 isOnOneSide( ) . . . . .	32
4.2.2.11 getTrian2( ) . . . . .	32
4.2.2.12 isCounterClockwise( ) . . . . .	33
4.2.2.13 computeInterval( ) . . . . .	33
4.2.2.14 getSegment( ) . . . . .	33
<b>5 Class Documentation</b> . . . . .	<b>35</b>
5.1 geom::Line< T > Class Template Reference . . . . .	35
5.1.1 Detailed Description . . . . .	35
5.1.2 Constructor & Destructor Documentation . . . . .	36
5.1.2.1 Line( ) . . . . .	36
5.1.3 Member Function Documentation . . . . .	36
5.1.3.1 org( ) . . . . .	36
5.1.3.2 dir( ) . . . . .	37
5.1.3.3 belongs( ) . . . . .	37
5.1.3.4 isEqual( ) . . . . .	37
5.1.3.5 getBy2Points( ) . . . . .	38
5.2 geom::Plane< T > Class Template Reference . . . . .	38
5.2.1 Detailed Description . . . . .	39
5.2.2 Member Function Documentation . . . . .	39
5.2.2.1 dist( ) . . . . .	39

5.2.2.2 norm()	40
5.2.2.3 belongs() [1/2]	40
5.2.2.4 belongs() [2/2]	40
5.2.2.5 isEqual()	41
5.2.2.6 isPar()	41
5.2.2.7 getBy3Points()	42
5.2.2.8 getParametric()	42
5.2.2.9 getNormalPoint()	43
5.2.2.10 getNormalDist()	43
5.3 geom::Triangle< T > Class Template Reference	44
5.3.1 Detailed Description	44
5.3.2 Constructor & Destructor Documentation	45
5.3.2.1 Triangle() [1/2]	45
5.3.2.2 Triangle() [2/2]	45
5.3.3 Member Function Documentation	45
5.3.3.1 operator[]() [1/2]	45
5.3.3.2 operator[]() [2/2]	46
5.3.3.3 getPlane()	46
5.3.3.4 isValid()	47
5.4 geom::Vec2< T > Class Template Reference	47
5.4.1 Detailed Description	48
5.4.2 Constructor & Destructor Documentation	49
5.4.2.1 Vec2() [1/2]	49
5.4.2.2 Vec2() [2/2]	49
5.4.3 Member Function Documentation	49
5.4.3.1 operator+=( )	50
5.4.3.2 operator-=( )	50
5.4.3.3 operator-( )	50
5.4.3.4 operator*=( ) [1/2]	51
5.4.3.5 operator/=( ) [1/2]	51
5.4.3.6 dot()	52
5.4.3.7 length2()	52
5.4.3.8 length()	53
5.4.3.9 getPerp()	53
5.4.3.10 normalized()	53
5.4.3.11 normalize()	54
5.4.3.12 operator[]() [1/2]	54
5.4.3.13 operator[]() [2/2]	54
5.4.3.14 isPar()	55
5.4.3.15 isPerp()	55
5.4.3.16 isEqual()	56
5.4.3.17 isNumEq()	56

5.4.3.18 setThreshold()	57
5.4.3.19 getThreshold()	57
5.4.3.20 setDefThreshold()	58
5.4.3.21 operator*=( ) [2/2]	58
5.4.3.22 operator/=( ) [2/2]	58
5.4.4 Member Data Documentation	58
5.4.4.1 x	58
5.4.4.2 y	59
5.5 geom::Vec3< T > Class Template Reference	59
5.5.1 Detailed Description	60
5.5.2 Constructor & Destructor Documentation	61
5.5.2.1 Vec3() [1/2]	61
5.5.2.2 Vec3() [2/2]	61
5.5.3 Member Function Documentation	61
5.5.3.1 operator+=( )	61
5.5.3.2 operator-=( )	62
5.5.3.3 operator-( )	62
5.5.3.4 operator*=( ) [1/2]	63
5.5.3.5 operator/=( ) [1/2]	63
5.5.3.6 dot()	64
5.5.3.7 cross()	64
5.5.3.8 length2()	65
5.5.3.9 length()	65
5.5.3.10 normalized()	65
5.5.3.11 normalize()	66
5.5.3.12 operator[]() [1/2]	66
5.5.3.13 operator[]() [2/2]	66
5.5.3.14 isPar()	67
5.5.3.15 isPerp()	67
5.5.3.16 isEqual()	68
5.5.3.17 isNumEq()	68
5.5.3.18 setThreshold()	69
5.5.3.19 getThreshold()	69
5.5.3.20 setDefThreshold()	70
5.5.3.21 operator*=( ) [2/2]	70
5.5.3.22 operator/=( ) [2/2]	70
5.5.4 Member Data Documentation	70
5.5.4.1 x	70
5.5.4.2 y	71
5.5.4.3 z	71

---

6.1 include/distance/distance.hh File Reference . . . . .	73
6.2 distance.hh . . . . .	74
6.3 include/intersection/intersection.hh File Reference . . . . .	75
6.4 intersection.hh . . . . .	76
6.5 include/primitives/common.hh File Reference . . . . .	82
6.6 common.hh . . . . .	84
6.7 include/primitives/line.hh File Reference . . . . .	84
6.8 line.hh . . . . .	86
6.9 include/primitives/plane.hh File Reference . . . . .	87
6.10 plane.hh . . . . .	90
6.11 include/primitives/primitives.hh File Reference . . . . .	92
6.12 primitives.hh . . . . .	94
6.13 include/primitives/triangle.hh File Reference . . . . .	94
6.14 triangle.hh . . . . .	96
6.15 include/primitives/vec2.hh File Reference . . . . .	97
6.15.1 Detailed Description . . . . .	99
6.16 vec2.hh . . . . .	99
6.17 include/primitives/vec3.hh File Reference . . . . .	105
6.17.1 Detailed Description . . . . .	108
6.18 vec3.hh . . . . .	109





# 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="#">35</a>
<a href="#">geom::Plane&lt; T &gt;</a>	
<a href="#">Plane</a> class realization . . . . .	<a href="#">38</a>
<a href="#">geom::Triangle&lt; T &gt;</a>	
<a href="#">Triangle</a> class implementation . . . . .	<a href="#">44</a>
<a href="#">geom::Vec2&lt; T &gt;</a>	
<a href="#">Vec2</a> class realization . . . . .	<a href="#">47</a>
<a href="#">geom::Vec3&lt; T &gt;</a>	
<a href="#">Vec3</a> class realization . . . . .	<a href="#">59</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>	73
include/intersection/ <a href="#">intersection.hh</a>	75
include/primitives/ <a href="#">common.hh</a>	82
include/primitives/ <a href="#">line.hh</a>	84
include/primitives/ <a href="#">plane.hh</a>	87
include/primitives/ <a href="#">primitives.hh</a>	92
include/primitives/ <a href="#">triangle.hh</a>	94
include/primitives/ <a href="#">vec2.hh</a>	97
include/primitives/ <a href="#">vec3.hh</a>	105



## 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>`  
`std::istream & operator>> (std::istream &ist, Triangle< T > &tr)`
- `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.*
- `template<std::floating_point T>`  
`std::istream & operator>> (std::istream &ist, Vec3< T > &vec)`  
*Vec3 scan 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 367 of file [vec2.hh](#).

### 4.1.2.2 Vec2F

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

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

### 4.1.2.3 Vec3D

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

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

### 4.1.2.4 Vec3F

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

Definition at line 399 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\(\)](#), [geom::detail::isIntersectValidInvalid\(\)](#), 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 160 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 193 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 143 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 157 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 88 of file [triangle.hh](#).

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

```
template<std::floating_point T>
std::istream& geom::operator>> (
```



```
std::istream & ist,
Triangle< T > & tr )
```

Definition at line 100 of file [triangle.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> sum of two vectors

Definition at line 234 of file [vec2.hh](#).

#### 4.1.3.11 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 250 of file [vec2.hh](#).

**4.1.3.12 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 267 of file [vec2.hh](#).

**4.1.3.13 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 284 of file [vec2.hh](#).

**4.1.3.14 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 301 of file [vec2.hh](#).

**4.1.3.15 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 317 of file [vec2.hh](#).

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

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

4.1.3.16 **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 332 of file [vec2.hh](#).

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

**4.1.3.17 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**

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

**Returns**

true if vectors are not equal  
false otherwise

Definition at line 347 of file [vec2.hh](#).

**4.1.3.18 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**

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

**Returns**

std::ostream& modified stream instance

Definition at line 361 of file [vec2.hh](#).

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

#### 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> sum of two vectors

Definition at line 236 of file [vec3.hh](#).

#### 4.1.3.20 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 252 of file [vec3.hh](#).

**4.1.3.21 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 269 of file [vec3.hh](#).

**4.1.3.22 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 286 of file [vec3.hh](#).

**4.1.3.23 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 303 of file [vec3.hh](#).

**4.1.3.24 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 319 of file [vec3.hh](#).

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

## 4.1.3.25 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 333 of file [vec3.hh](#).

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

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

**4.1.3.26 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**

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

**Returns**

true if vectors are equal

false otherwise

Definition at line 348 of file [vec3.hh](#).

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

**4.1.3.27 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**

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

**Returns**

true if vectors are not equal  
false otherwise

Definition at line 363 of file [vec3.hh](#).

**4.1.3.28 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 377 of file [vec3.hh](#).

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

**4.1.3.29 operator>>() [2/2]**

```
template<std::floating_point T>
std::istream& geom::operator>> (
    std::istream & ist,
    Vec3< T > & vec )
```

[Vec3](#) scan operator.

**Template Parameters**

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

**Parameters**

<i>in, out</i>	<i>ist</i>	input stram
<i>in, out</i>	<i>vec</i>	vector to scan

**Returns**

`std::istream&` modified stream instance

Definition at line 392 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 [Segment2D](#) = `std::pair< T, T >`
- `template<std::floating_point T>`  
using [Trian2](#) = `std::array< Vec2< T >, 3 >`
- `template<std::floating_point T>`  
using [Segment3D](#) = `std::pair< Vec3< T >, Vec3< T > >`

## 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>`  
`Segment2D< 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 > &valid, const Triangle< T > &invalid)`
- `template<std::floating_point T>`  
`bool isIntersectPointTriangle (const Vec3< T > &pt, const Triangle< T > &tr)`
- `template<std::floating_point T>`  
`bool isPoint (const Triangle< T > &tr)`
- `template<std::floating_point T>`  
`bool isOverlap (Segment2D< T > &segm1, Segment2D< 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>`  
`Segment2D< T > computeInterval (const Trian2< T > &tr, const Vec2< T > &d)`
- `template<std::floating_point T>`  
`Segment3D< T > getSegment (const Triangle< T > &tr)`

### 4.2.1 Typedef Documentation

#### 4.2.1.1 Segment2D

```
template<typename T >
using geom::detail::Segment2D = typedef std::pair<T, T>
```

Definition at line 103 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 106 of file [intersection.hh](#).

#### 4.2.1.3 Segment3D

```
template<std::floating_point T>
using geom::detail::Segment3D = typedef std::pair<Vec3<T>, Vec3<T> >
```

Definition at line 109 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 223 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 248 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>
Segment2D< T > geom::detail::helperMollerHaines (
    const Triangle< T > & tr,
    const Plane< T > & pl,
    const Line< T > & l )
```

Definition at line 262 of file [intersection.hh](#).

References [geom::Line< T >::dir\(\)](#), [geom::distance\(\)](#), [geom::dot\(\)](#), [geom::Vec3< T >::isNumEq\(\)](#), [isSameSign\(\)](#), 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 303 of file [intersection.hh](#).

References [isPoint\(\)](#).

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

#### 4.2.2.5 isIntersectValidInvalid()

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

Definition at line 324 of file [intersection.hh](#).

References [geom::distance\(\)](#), [geom::Triangle< T >::getPlane\(\)](#), [getSegment\(\)](#), [isIntersectPointTriangle\(\)](#), and [isPoint\(\)](#).

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

#### 4.2.2.6 isIntersectPointTriangle()

```
template<std::floating_point T>
bool geom::detail::isIntersectPointTriangle (
    const Vec3< T > & pt,
    const Triangle< T > & tr )
```

Definition at line 349 of file [intersection.hh](#).

References [geom::dot\(\)](#), [geom::Triangle< T >::getPlane\(\)](#), and [geom::Vec3< T >::getThreshold\(\)](#).

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

#### 4.2.2.7 isPoint()

```
template<std::floating_point T>
bool geom::detail::isPoint (
    const Triangle< T > & tr )
```

Definition at line 377 of file [intersection.hh](#).

Referenced by [isIntersectBothInvalid\(\)](#), and [isIntersectValidInvalid\(\)](#).

#### 4.2.2.8 isOverlap()

```
template<std::floating_point T>
bool geom::detail::isOverlap (
    Segment2D< T > & segm1,
    Segment2D< T > & segm2 )
```

Definition at line 383 of file [intersection.hh](#).

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

#### 4.2.2.9 isSameSign()

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

Definition at line 389 of file [intersection.hh](#).

Referenced by [helperMollerHaines\(\)](#), and [isOnOneSide\(\)](#).

#### 4.2.2.10 isOnOneSide()

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

Definition at line 402 of file [intersection.hh](#).

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

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

#### 4.2.2.11 getTrian2()

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

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

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

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



#### 4.2.2.12 isCounterClockwise()

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

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

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

#### 4.2.2.13 computeInterval()

```
template<std::floating_point T>
Segment2D< T > geom::detail::computeInterval (
    const Trian2< T > & tr,
    const Vec2< T > & d )
```

Definition at line [469](#) of file [intersection.hh](#).

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

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

#### 4.2.2.14 getSegment()

```
template<std::floating_point T>
Segment3D< T > geom::detail::getSegment (
    const Triangle< T > & tr )
```

Definition at line [485](#) of file [intersection.hh](#).

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



## 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 from 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 22 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 171 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 177 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 183 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 189 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 195 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 201 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 207 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 213 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 220 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 227 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](#) ()  
*Construct a new [Triangle](#) object.*
- [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.*
- [Vec3](#)< T > & [operator\[\]](#) (std::size\_t idx)  
*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

<a href="#">T</a>	- floating point type of coordinates
-------------------	--------------------------------------

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

## 5.3.2 Constructor & Destructor Documentation

### 5.3.2.1 Triangle() [1/2]

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

Construct a new [Triangle](#) object.

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

### 5.3.2.2 Triangle() [2/2]

```
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 111 of file [triangle.hh](#).

## 5.3.3 Member Function Documentation

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

```
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 115 of file [triangle.hh](#).

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

```
template<std::floating_point T>
Vec3< T > & geom::Triangle< T >::operator[] (
    std::size_t idx )
```

Overloaded operator[] to get access to vertices.

**Parameters**

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

**Returns**

Vec3<T>& reference to vertex

Definition at line 121 of file [triangle.hh](#).

**5.3.3.3 getPlane()**

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

Get triangle's plane.

**Returns**

Plane<T>

Definition at line 127 of file [triangle.hh](#).

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

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

## 5.3.3.4 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 133 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&lt; T &gt; 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 26 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 46 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 54 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	vec	vector to incremented with
----	-----	----------------------------

##### Returns

Vec2& reference to current instance

Definition at line 371 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	vec	vector to decremented with
----	-----	----------------------------

##### Returns

Vec2& reference to current instance

Definition at line 380 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 389 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**

in	<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 415 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 length<sup>2</sup>

Definition at line 421 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 length

Definition at line 427 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 433 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 439 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 447 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

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

##### Returns

`T&` reference to coordinate value

##### Note

Coordinates calculated by mod 2

Definition at line 456 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 470 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 484 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 491 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 497 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 ( $|\text{lhs} - \text{rhs}| < \text{threshold}$ )  
false otherwise

## Note

Threshold defined by `threshold_` static member

Definition at line 503 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 509 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 515 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 521 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 396 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 406 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 38 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 38 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 26 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 47 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 55 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 402 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 412 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 422 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 450 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 456 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 462 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 468 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 474 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 482 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 491 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 507 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 523 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 529 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 535 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 541 of file [vec3.hh](#).

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

**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 547 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 553 of file [vec3.hh](#).

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

### 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 559 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 429 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 440 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 38 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-=\( \)](#), [geom::operator<<\(\)](#), and [geom::operator>>\(\)](#).

### 5.5.4.2 y

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

Definition at line 38 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-=\(\)](#), [geom::operator<<\(\)](#), and [geom::operator>>\(\)](#).

### 5.5.4.3 z

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

Definition at line 38 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-=\(\)](#), [geom::operator<<\(\)](#), and [geom::operator>>\(\)](#).

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

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





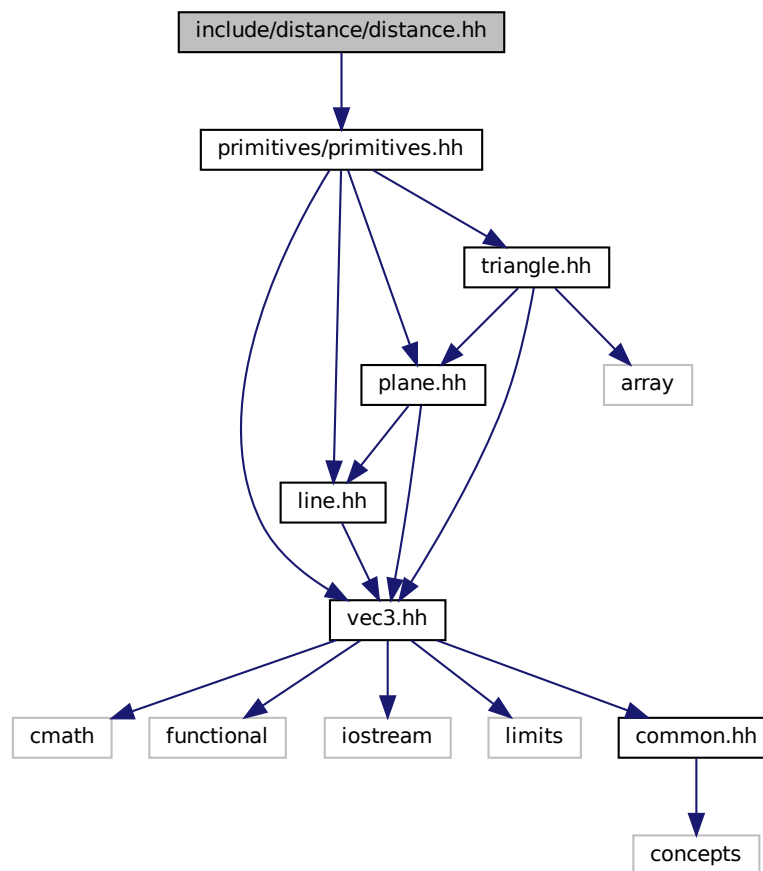
## Chapter 6

# File Documentation

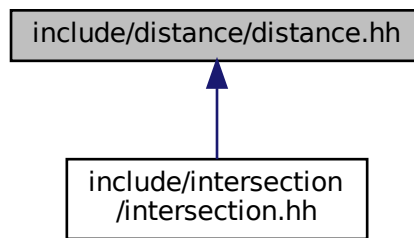
### 6.1 include/distance/distance.hh File Reference

```
#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

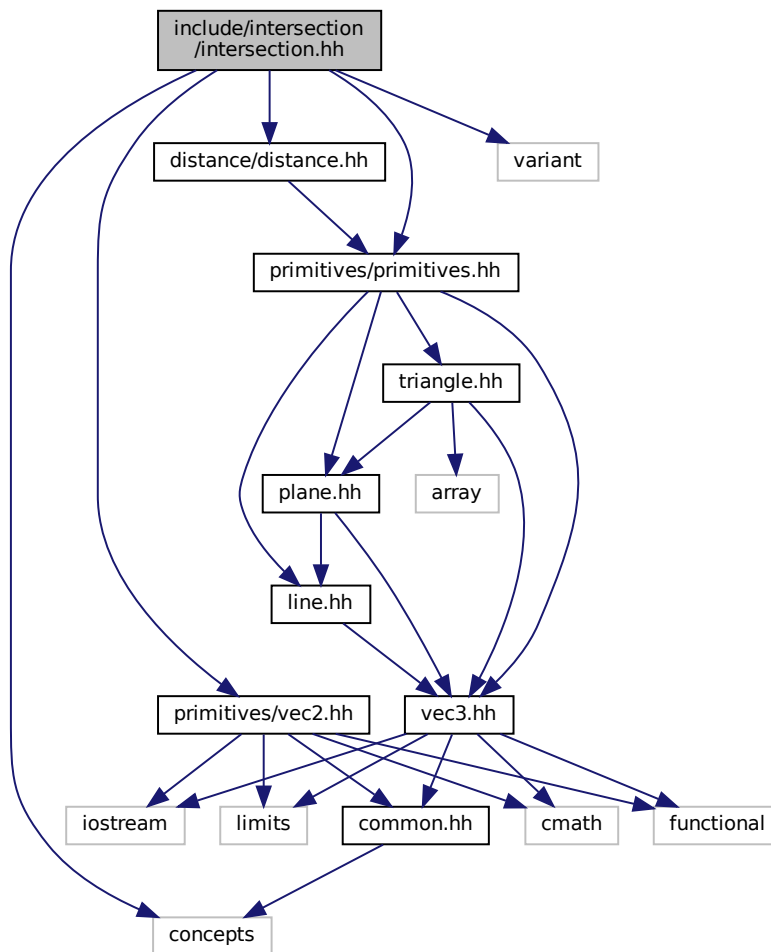
```

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::Segment2D](#) = `std::pair< T, T >`

- `template<std::floating_point T>`  
using `geom::detail::Trian2` = `std::array< Vec2< T >, 3 >`
- `template<std::floating_point T>`  
using `geom::detail::Segment3D` = `std::pair< Vec3< T >, Vec3< T >`

## 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>`  
`Segment2D< 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 > &valid, const Triangle< T > &invalid)
- `template<std::floating_point T>`  
bool `geom::detail::isIntersectPointTriangle` (const Vec3< T > &pt, const Triangle< T > &tr)
- `template<std::floating_point T>`  
bool `geom::detail::isPoint` (const Triangle< T > &tr)
- `template<std::floating_point T>`  
bool `geom::detail::isOverlap` (Segment2D< T > &segm1, Segment2D< 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>`  
`Segment2D< T >` `geom::detail::computeInterval` (const Trian2< T > &tr, const Vec2< T > &d)
- `template<std::floating_point T>`  
`Segment3D< T >` `geom::detail::getSegment` (const Triangle< T > &tr)

## 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  *  $\vec{pl}_1$  equation:  $\vec{n}_1 \cdot \vec{P} = d_1$ 
00034  *
00035  *  $\vec{pl}_2$  equation:  $\vec{n}_2 \cdot \vec{P} = d_2$ 
00036  *
00037  * Intersection line direction:  $\vec{dir} = \vec{n}_1 \times \vec{n}_2$ 
00038  *
00039  *
00040  * Let origin of intersection line be a linear combination of  $\vec{n}_1$ 
00041  * and  $\vec{n}_2$ :  $\vec{P} = a \cdot \vec{n}_1 + b \cdot \vec{n}_2$ 
00042  *
00043  *
00044  *  $\vec{P}$  must satisfy both  $\vec{pl}_1$  and  $\vec{pl}_2$  equations:
00045  *
00046  *  $\vec{n}_1 \cdot \vec{P} = d_1$ 
00047  *
00048  *  $\vec{n}_1 \cdot (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) = d_1$ 
00049  *
00050  *  $a + b \cdot \vec{n}_1 \cdot \vec{n}_2 = d_1$ 
00051  *
00052  *  $a = \frac{d_1 - b \cdot \vec{n}_1 \cdot \vec{n}_2}{\vec{n}_1 \cdot \vec{n}_1}$ 
00053  *
00054  *  $\vec{n}_2 \cdot \vec{P} = d_2$ 
00055  *
00056  *  $\vec{n}_2 \cdot (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) = d_2$ 
00057  *
00058  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00059  *
00060  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00061  *
00062  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00063  *
00064  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00065  *
00066  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00067  *
00068  *
00069  * Let's find  $a$  and  $b$ :
00070  *
00071  *  $a = \frac{d_1 - b \cdot \vec{n}_1 \cdot \vec{n}_2}{\vec{n}_1 \cdot \vec{n}_1}$ 
00072  *
00073  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00074  *
00075  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00076  *
00077  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00078  *
00079  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00080  *
00081  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00082  *
00083  *  $a \cdot \vec{n}_1 \cdot \vec{n}_2 + b \cdot \vec{n}_2 \cdot \vec{n}_2 = d_2$ 
00084  *
00085  * Intersection line equation:
00086  *
00087  *  $\vec{r}(t) = \vec{P} + t \cdot \vec{dir}$ 
00088  *
00089  *  $\vec{r}(t) = (a \cdot \vec{n}_1 + b \cdot \vec{n}_2) + t \cdot \vec{dir}$ 
00090  *
00091  * @tparam T - floating point type of coordinates
00092  * @param p11 first plane
00093  * @param p12 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> &p11, const Plane<T> &p12);
00098
00099 namespace detail

```

```

00100 {
00101
00102 template <typename T>
00103 using Segment2D = std::pair<T, T>;
00104
00105 template <std::floating_point T>
00106 using Trian2 = std::array<Vec2<T>, 3>;
00107
00108 template <std::floating_point T>
00109 using Segment3D = std::pair<Vec3<T>, Vec3<T>>;
00110
00111 template <std::floating_point T>
00112 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2);
00113
00114 template <std::floating_point T>
00115 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2);
00116
00117 template <std::floating_point T>
00118 Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l);
00119
00120 template <std::floating_point T>
00121 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2);
00122
00123 template <std::floating_point T>
00124 bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid);
00125
00126 template <std::floating_point T>
00127 bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr);
00128
00129 template <std::floating_point T>
00130 bool isPoint(const Triangle<T> &tr);
00131
00132 template <std::floating_point T>
00133 bool isOverlap(Segment2D<T> &segm1, Segment2D<T> &segm2);
00134
00135 template <std::forward_iterator It>
00136 bool isSameSign(It begin, It end);
00137
00138 template <std::floating_point T>
00139 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr);
00140
00141 template <std::floating_point T>
00142 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr);
00143
00144 template <std::floating_point T>
00145 bool isCounterClockwise(Trian2<T> &tr);
00146
00147 template <std::floating_point T>
00148 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d);
00149
00150 template <std::floating_point T>
00151 Segment3D<T> getSegment(const Triangle<T> &tr);
00152
00153 } // namespace detail
00154 } // namespace geom
00155
00156 namespace geom
00157 {
00158
00159 template <std::floating_point T>
00160 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
00161 {
00162     /* TODO: handle invalid triangles case */
00163     auto isInv1 = !tr1.isValid();
00164     auto isInv2 = !tr2.isValid();
00165
00166     if (isInv1 && isInv2)
00167         return detail::isIntersectBothInvalid(tr1, tr2);
00168
00169     if (isInv1)
00170         return detail::isIntersectValidInvalid(tr2, tr1);
00171
00172     if (isInv2)
00173         return detail::isIntersectValidInvalid(tr1, tr2);
00174
00175     auto pl1 = tr1.getPlane();
00176     if (detail::isOnOneSide(pl1, tr2))
00177         return false;
00178
00179     auto pl2 = tr2.getPlane();
00180     if (pl1 == pl2)
00181         return detail::isIntersect2D(tr1, tr2);
00182
00183     if (pl1.isPar(pl2))
00184         return false;
00185
00186     if (detail::isOnOneSide(pl2, tr1))

```

```

00187     return false;
00188
00189     return detail::isIntersectMollerHaines(tr1, tr2);
00190 }
00191
00192 template <std::floating_point T>
00193 std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &p1, const Plane<T> &p2)
00194 {
00195     const auto &n1 = p1.norm();
00196     const auto &n2 = p2.norm();
00197
00198     auto dir = cross(n1, n2);
00199
00200     /* if planes are parallel */
00201     if (Vec3<T>{0} == dir)
00202     {
00203         if (p1 == p2)
00204             return p1;
00205
00206         return std::monostate{};
00207     }
00208
00209     auto nln2 = dot(n1, n2);
00210     auto d1 = p1.dist();
00211     auto d2 = p2.dist();
00212
00213     auto a = (d2 * nln2 - d1) / (nln2 * nln2 - 1);
00214     auto b = (d1 * nln2 - d2) / (nln2 * nln2 - 1);
00215
00216     return Line<T>{(a * n1) + (b * n2), dir};
00217 }
00218
00219 namespace detail
00220 {
00221
00222 template <std::floating_point T>
00223 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
00224 {
00225     auto pl = tr1.getPlane();
00226
00227     auto trian1 = getTrian2(pl, tr1);
00228     auto trian2 = getTrian2(pl, tr2);
00229
00230     for (auto trian : {trian1, trian2})
00231     {
00232         for (size_t i0 = 0, i1 = 2; i0 < 3; i1 = i0, ++i0)
00233         {
00234             auto d = (trian[i0] - trian[i1]).getPerp();
00235
00236             auto s1 = computeInterval(trian1, d);
00237             auto s2 = computeInterval(trian2, d);
00238
00239             if (s2.second < s1.first || s1.second < s2.first)
00240                 return false;
00241         }
00242     }
00243
00244     return true;
00245 }
00246
00247 template <std::floating_point T>
00248 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
00249 {
00250     auto pl1 = tr1.getPlane();
00251     auto pl2 = tr2.getPlane();
00252
00253     auto l = std::get<Line<T>>(intersect(pl1, pl2));
00254
00255     auto params1 = helperMollerHaines(tr1, pl2, l);
00256     auto params2 = helperMollerHaines(tr2, pl1, l);
00257
00258     return isOverlap(params1, params2);
00259 }
00260
00261 template <std::floating_point T>
00262 Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
00263 {
00264     /* Project the triangle vertices onto line */
00265     std::array<T, 3> vert{};
00266     for (size_t i = 0; i < 3; ++i)
00267         vert[i] = dot(l.dir(), tr[i] - l.org());
00268
00269     std::array<T, 3> sdist{};
00270     for (size_t i = 0; i < 3; ++i)
00271         sdist[i] = distance(pl, tr[i]);
00272
00273     auto isSameSign = [] (const auto &num1, const auto &num2) {

```

```

00274     if (num1 * num2 > Vec3<T>::getThreshold())
00275         return true;
00276     return Vec3<T>::isNumEq(num1, 0) && Vec3<T>::isNumEq(num2, 0);
00277 };
00278
00279 std::array<bool, 3> isOneSide{};
00280 for (size_t i = 0; i < 3; ++i)
00281     isOneSide[i] = isSameSign(sdist[i], sdist[(i + 1) % 3]);
00282
00283 /* Looking for vertex which is alone on it's side */
00284 size_t rogue = 0;
00285 for (size_t i = 0; i < 3; ++i)
00286     if (isOneSide[i])
00287         rogue = (i + 2) % 3;
00288
00289 std::vector<T> segm{};
00290 std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};
00291
00292 for (size_t i : arr)
00293     segm.push_back(vert[i] + (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]));
00294
00295 /* Sort segment's ends */
00296 if (segm[0] > segm[1])
00297     std::swap(segm[0], segm[1]);
00298
00299 return {segm[0], segm[1]};
00300 }
00301
00302 template <std::floating_point T>
00303 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
00304 {
00305     auto isPoint1 = isPoint(tr1);
00306     auto isPoint2 = isPoint(tr2);
00307
00308     if (isPoint1 && isPoint2)
00309         return tr1[0] == tr2[0];
00310
00311     if (isPoint1)
00312         return false; // TODO: handle
00313
00314     if (isPoint2)
00315         return false; // TODO: handle
00316
00317     std::cout << "both invalid" << std::endl;
00318     std::cout << "tr1: " << tr1 << std::endl;
00319     std::cout << "tr2: " << tr2 << std::endl;
00320     return false;
00321 }
00322
00323 template <std::floating_point T>
00324 bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid)
00325 {
00326     if (isPoint(invalid))
00327         return isIntersectPointTriangle(invalid[0], valid);
00328
00329     auto segm = getSegment(invalid);
00330     auto pl = valid.getPlane();
00331
00332     auto dst1 = distance(pl, segm.first);
00333     auto dst2 = distance(pl, segm.second);
00334
00335     if (dst1 * dst2 > 0)
00336         return false;
00337
00338     if (Vec3<T>::isNumEq(dst1, 0) && Vec3<T>::isNumEq(dst2, 0))
00339         return false; // TODO: handle
00340
00341     dst1 = std::abs(dst1);
00342     dst2 = std::abs(dst2);
00343
00344     auto pt = segm.first + (segm.second - segm.first) * dst1 / (dst1 + dst2);
00345     return isIntersectPointTriangle(pt, valid);
00346 }
00347
00348 template <std::floating_point T>
00349 bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr)
00350 {
00351     if (!tr.getPlane().belongs(pt))
00352         return false;
00353
00354     /* TODO: comment better */
00355     /* pt = point + u * edge1 + v * edge2 */
00356     auto point = pt - tr[0];
00357     auto edge1 = tr[1] - tr[0];
00358     auto edge2 = tr[2] - tr[0];
00359
00360     auto dotE1E1 = dot(edge1, edge1);

```



```

00361     auto dotE1E2 = dot(edge1, edge2);
00362     auto dotE1PT = dot(edge1, point);
00363
00364     auto dotE2E2 = dot(edge2, edge2);
00365     auto dotE2PT = dot(edge2, point);
00366
00367     auto denom = dotE1E1 * dotE2E2 - dotE1E2 * dotE1E2;
00368     auto u = (dotE2E2 * dotE1PT - dotE1E2 * dotE2PT) / denom;
00369     auto v = (dotE1E1 * dotE2PT - dotE1E2 * dotE1PT) / denom;
00370
00371     /* Point belongs to triangle if: (u >= 0) && (v >= 0) && (u + v <= 1) */
00372     auto eps = Vec3<T>::getThreshold();
00373     return (u > -eps) && (v > -eps) && (u + v < 1 + eps);
00374 }
00375
00376 template <std::floating_point T>
00377 bool isPoint(const Triangle<T> &tr)
00378 {
00379     return (tr[0] == tr[1]) && (tr[0] == tr[2]);
00380 }
00381
00382 template <std::floating_point T>
00383 bool isOverlap(Segment2D<T> &segm1, Segment2D<T> &segm2)
00384 {
00385     return (segm2.first <= segm1.second) && (segm2.second >= segm1.first);
00386 }
00387
00388 template <std::forward_iterator It>
00389 bool isSameSign(It begin, It end)
00390 {
00391     auto cur = begin;
00392     auto prev = begin;
00393
00394     for (++cur; cur != end; ++cur)
00395         if ((*cur) * (*prev) <= 0)
00396             return false;
00397
00398     return true;
00399 }
00400
00401 template <std::floating_point T>
00402 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
00403 {
00404     std::array<T, 3> sdist{};
00405     for (size_t i = 0; i < 3; ++i)
00406         sdist[i] = distance(pl, tr[i]);
00407
00408     if (detail::isSameSign(sdist.begin(), sdist.end()))
00409         return true;
00410
00411     return false;
00412 }
00413
00414 template <std::floating_point T>
00415 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr)
00416 {
00417     auto norm = pl.norm();
00418
00419     const Vec3<T> x{1, 0, 0};
00420     const Vec3<T> y{0, 1, 0};
00421     const Vec3<T> z{0, 0, 1};
00422
00423     std::array<Vec3<T>, 3> xyz{x, y, z};
00424     std::array<T, 3> xyzDot;
00425
00426     std::transform(xyz.begin(), xyz.end(), xyzDot.begin(),
00427         [&norm](const auto &axis) { return std::abs(dot(axis, norm)); });
00428
00429     auto maxIt = std::max_element(xyzDot.begin(), xyzDot.end());
00430     auto maxIdx = static_cast<size_t>(maxIt - xyzDot.begin());
00431
00432     Trian2<T> res;
00433     for (size_t i = 0; i < 3; ++i)
00434         for (size_t j = 0, k = 0; j < 2; ++j, ++k)
00435             if (k == maxIdx)
00436                 ++k;
00437
00438         res[i][j] = tr[i][k];
00439     }
00440
00441     if (!isCounterClockwise(res))
00442         std::swap(res[0], res[1]);
00443
00444     return res;
00445 }
00446 }
00447

```

```

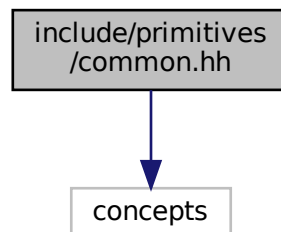
00448 template <std::floating_point T>
00449 bool isCounterClockwise(Trian2<T> &tr)
00450 {
00451     /**
00452      * The triangle is counterclockwise ordered if \delta > 0
00453      * and clockwise ordered if \delta < 0.
00454      *
00455      *      + 1 1 1 +
00456      * \delta = det | x0 x1 x2 | = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0)
00457      *      + y0 y1 y2 +      + (x0 * y1 - x1 * y0)
00458      *
00459      */
00460
00461     auto x0 = tr[0][0], x1 = tr[1][0], x2 = tr[2][0];
00462     auto y0 = tr[0][1], y1 = tr[1][1], y2 = tr[2][1];
00463
00464     auto delta = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0) + (x0 * y1 - x1 * y0);
00465     return (delta > 0);
00466 }
00467
00468 template <std::floating_point T>
00469 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
00470 {
00471     auto init = dot(d, tr[0]);
00472     auto min = init;
00473     auto max = init;
00474
00475     for (size_t i = 1; i < 3; ++i)
00476         if (auto val = dot(d, tr[i]); val < min)
00477             min = val;
00478         else if (val > max)
00479             max = val;
00480
00481     return {min, max};
00482 }
00483
00484 template <std::floating_point T>
00485 Segment3D<T> getSegment(const Triangle<T> &tr)
00486 {
00487     std::array<T, 3> lenArr{};
00488     for (size_t i = 0; i < 3; ++i)
00489         lenArr[i] = (tr[i] - tr[i + 1]).length2();
00490
00491     auto maxIt = std::max_element(lenArr.begin(), lenArr.end());
00492     auto maxIdx = static_cast<size_t>(maxIt - lenArr.begin());
00493
00494     return {tr[maxIdx], tr[maxIdx + 1]};
00495 }
00496
00497 } // namespace detail
00498 } // namespace geom
00499
00500 #endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__

```

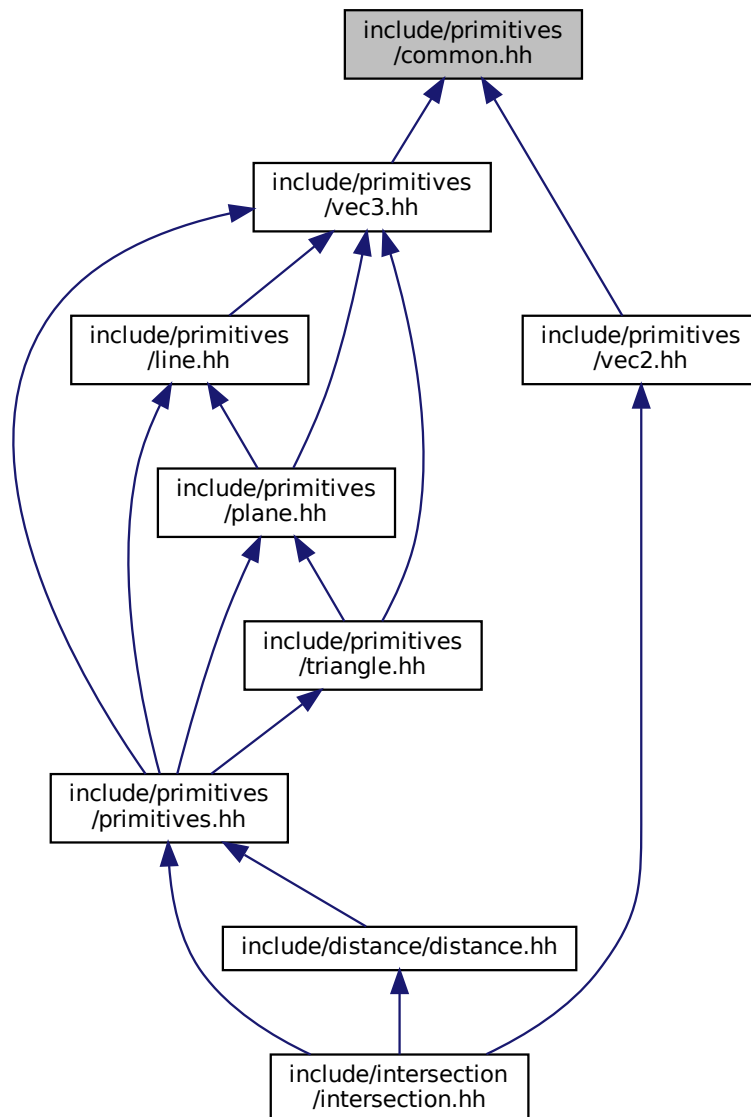
## 6.5 include/primitives/common.hh File Reference

#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

```

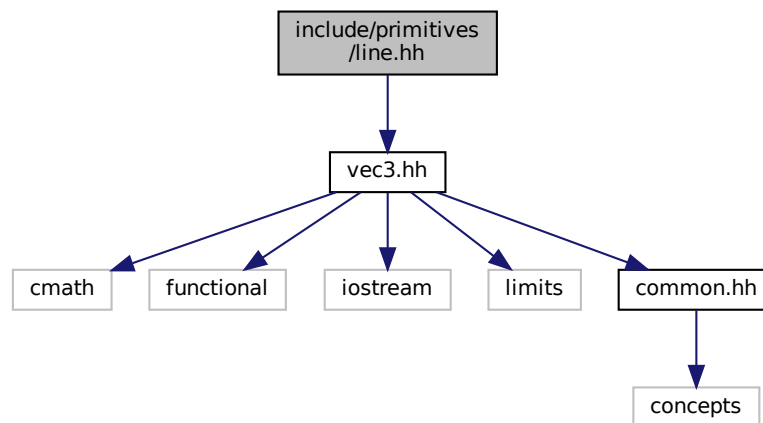
00001 #ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
00002 #define __INCLUDE_PRIMITIVES_COMMON_HH__
00003
00004 #include <concepts>
00005
00006 namespace geom
00007 {
00008 /**
00009  * @concept Number
00010  * @brief Useful concept which represents floating point and integral types
00011  *
00012  * @tparam 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__

```

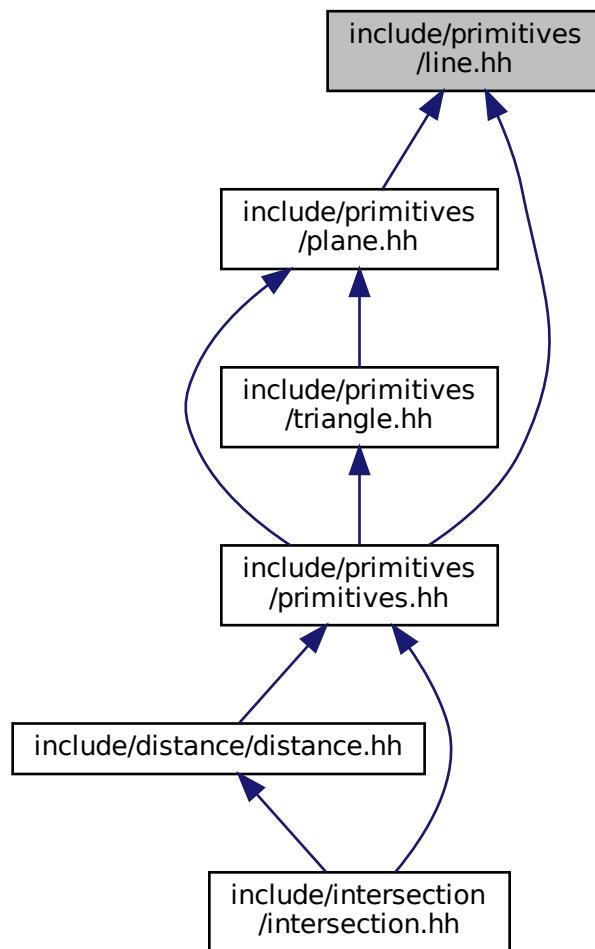
## 6.7 include/primitives/line.hh File Reference

```
#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__

```

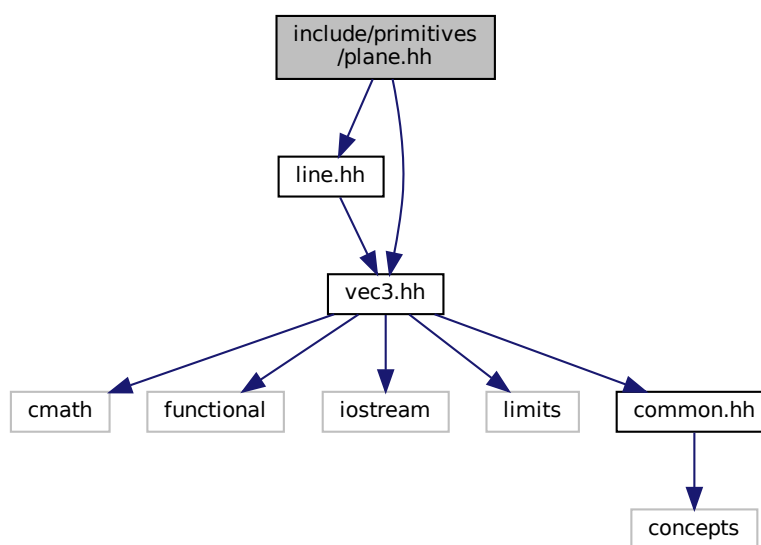
## 6.9 include/primitives/plane.hh File Reference

```

#include "line.hh"
#include "vec3.hh"

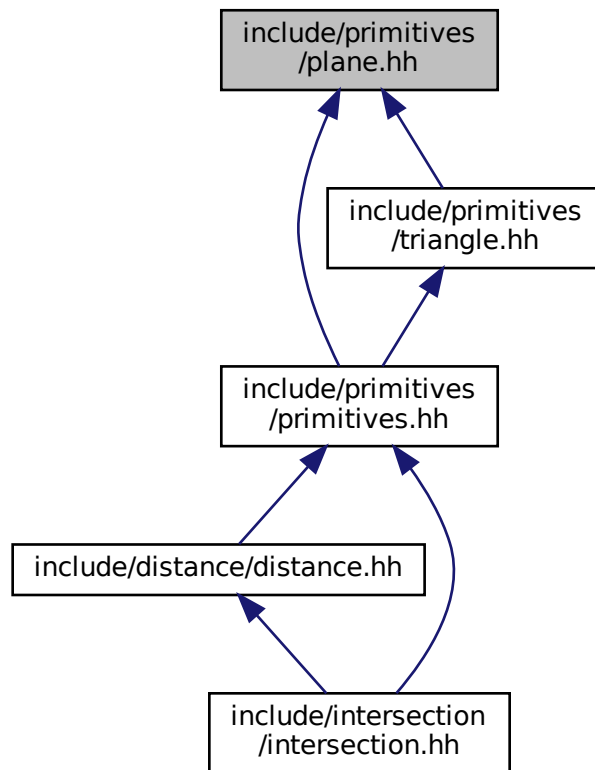
```

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 "line.hh"
00005 #include "vec3.hh"
00006
00007 /**
00008  * @brief
00009  * Plane class implementation
00010  */
00011
00012 namespace geom
00013 {
00014
00015 /**
00016  * @class Plane
00017  * @brief Plane class realization
00018  *
00019  * @tparam T - floating point type of coordinates
00020  */
00021 template <std::floating_point T>
00022 class Plane final
00023 {
00024 private:
00025     /**
00026      * @brief Normal vector, length equals to 1
00027      */
00028     Vec3<T> norm_{};
00029
00030     /**
00031      * @brief Distance from zero to plane
00032      */
00033     T dist_{};
00034
00035     /**
00036      * @brief Construct a new Plane object from normal vector and distance
00037      *
00038      * @param[in] norm normal vector
00039      * @param[in] dist distance from plane to zero
00040      */
00041     Plane(const Vec3<T> &norm, T dist);
00042
00043 public:
00044     /**
00045      * @brief Getter for distance
00046      *
00047      * @return T value of distance
00048      */
00049     T dist() const;
00050
00051     /**
00052      * @brief Getter for normal vector
00053      *
00054      * @return const Vec3<T>& const reference to normal vector
00055      */
00056     const Vec3<T> &norm() const;
00057
00058     /**
00059      * @brief Checks if point belongs to plane
00060      *
00061      * @param[in] point const referene to point vector
00062      * @return true if point belongs to plane
00063      * @return false if point doesn't belong to plane
00064      */
00065     bool belongs(const Vec3<T> &point) const;
00066
00067     /**
00068      * @brief Checks if line belongs to plane
00069      *
00070      * @param[in] line const referene to line
00071      * @return true if line belongs to plane
00072      * @return false if line doesn't belong to plane
00073      */
00074     bool belongs(const Line<T> &line) const;
00075
00076     /**
00077      * @brief Checks is *this equals to another plane
00078      *
00079      * @param[in] rhs const reference to another plane
00080      * @return true if planes are equal
00081      * @return false if planes are not equal
00082      */
00083     bool isEqual(const Plane &rhs) const;
00084
00085     /**

```

```

00086     * @brief Checks is *this is parallel to another plane
00087     *
00088     * @param[in] rhs const reference to another plane
00089     * @return true if planes are parallel
00090     * @return false if planes are not parallel
00091     */
00092     bool isPar(const Plane &rhs) const;
00093
00094     /**
00095     * @brief Get plane by 3 points
00096     *
00097     * @param[in] pt1 1st point
00098     * @param[in] pt2 2nd point
00099     * @param[in] pt3 3rd point
00100     * @return Plane passing through three points
00101     */
00102     static Plane getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3);
00103
00104     /**
00105     * @brief Get plane from parametric plane equation
00106     *
00107     * @param[in] org origin vector
00108     * @param[in] dir1 1st direction vector
00109     * @param[in] dir2 2nd direction vector
00110     * @return Plane
00111     */
00112     static Plane getParametric(const Vec3<T> &org, const Vec3<T> &dir1, const Vec3<T> &dir2);
00113
00114     /**
00115     * @brief Get plane from normal point plane equation
00116     *
00117     * @param[in] norm normal vector
00118     * @param[in] point point lying on the plane
00119     * @return Plane
00120     */
00121     static Plane getNormalPoint(const Vec3<T> &norm, const Vec3<T> &point);
00122
00123     /**
00124     * @brief Get plane form normal const plane equation
00125     *
00126     * @param[in] norm normal vector
00127     * @param[in] constant distance
00128     * @return Plane
00129     */
00130     static Plane getNormalDist(const Vec3<T> &norm, T constant);
00131 };
00132
00133 /**
00134 * @brief Plane equality operator
00135 *
00136 * @tparam T - floating point type of coordinates
00137 * @param[in] lhs 1st plane
00138 * @param[in] rhs 2nd plane
00139 * @return true if planes are equal
00140 * @return false if planes are not equal
00141 */
00142 template <std::floating_point T>
00143 bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
00144 {
00145     return lhs.isEqual(rhs);
00146 }
00147
00148 /**
00149 * @brief Plane print operator
00150 *
00151 * @tparam T - floating point type of coordinates
00152 * @param[in, out] ost output stream
00153 * @param[in] pl plane to print
00154 * @return std::ostream& modified ostream instance
00155 */
00156 template <std::floating_point T>
00157 std::ostream &operator<<(std::ostream &ost, const Plane<T> &pl)
00158 {
00159     ost << pl.norm() << " * X = " << pl.dist();
00160     return ost;
00161 }
00162
00163 template <std::floating_point T>
00164 Plane<T>::Plane(const Vec3<T> &norm, T dist) : norm_(norm), dist_(dist)
00165 {
00166     if (norm == Vec3<T>{0})
00167         throw std::logic_error{"normal vector equals to zero"};
00168 }
00169
00170 template <std::floating_point T>
00171 T Plane<T>::dist() const
00172 {

```

```

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

```

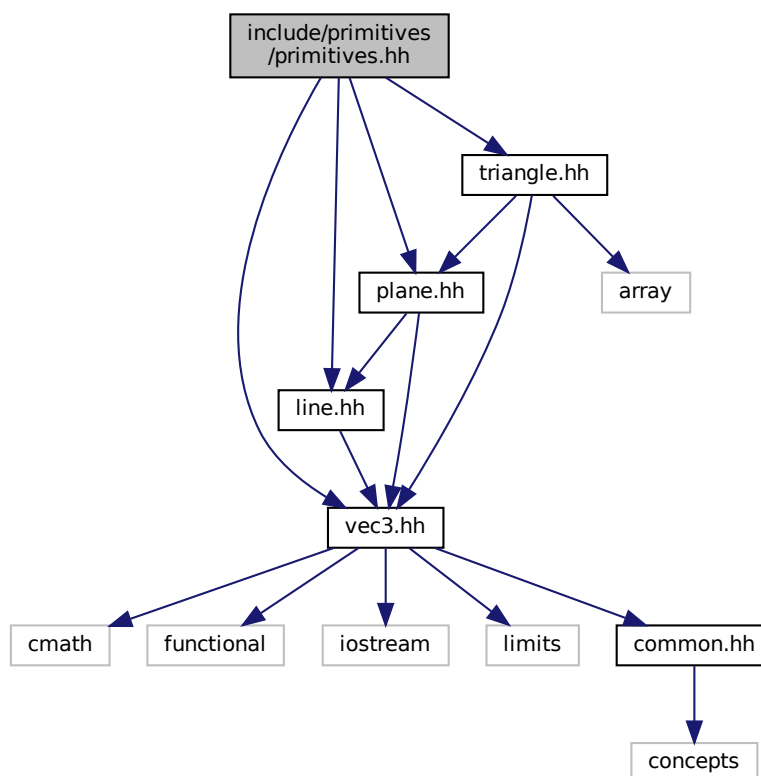
## 6.11 include/primitives/primitives.hh File Reference

```

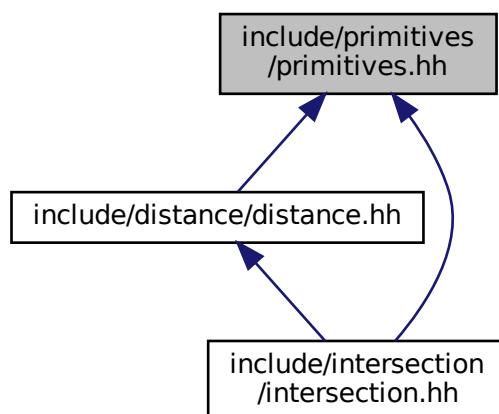
#include "line.hh"
#include "plane.hh"
#include "triangle.hh"
#include "vec3.hh"

```

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__

```

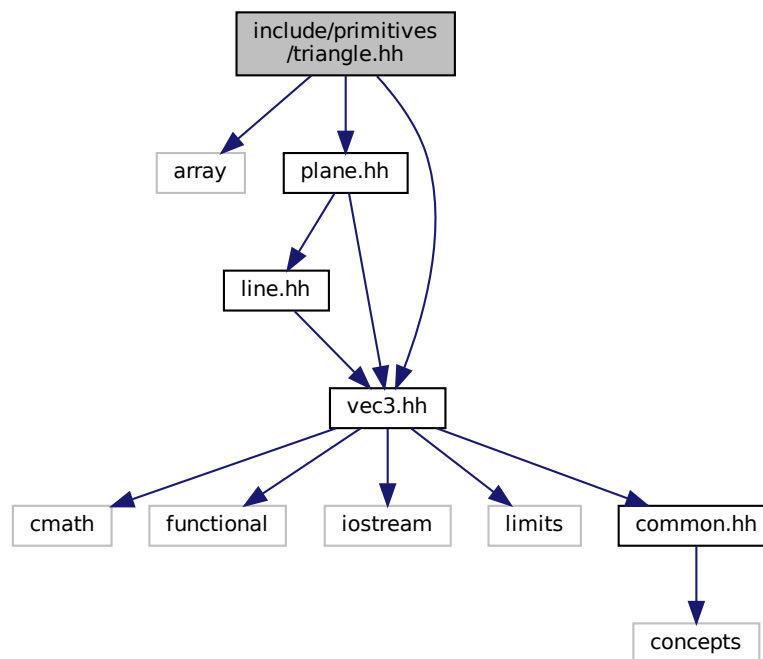
## 6.13 include/primitives/triangle.hh File Reference

```

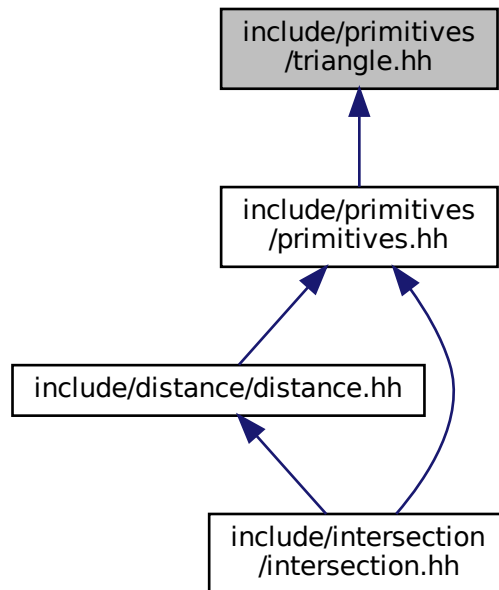
#include <array>
#include "plane.hh"
#include "vec3.hh"

```

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.
- template<std::floating\_point T>  
std::istream & [geom::operator>>](#) (std::istream &ist, Triangle< T > &tr)

## 6.14 triangle.hh

```

00001 #ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00002 #define __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00003
00004 #include <array>
00005
00006 #include "plane.hh"
00007 #include "vec3.hh"
00008
00009 /**
00010  * @brief triangle.hh
00011  * Triangle class implementation
00012  */
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @class Triangle
00019  * @brief Triangle class implementation
00020  *
00021  * @tparam T - floating point type of coordinates
00022  */
00023 template <std::floating_point T>
00024 class Triangle final
00025 {
00026 private:
00027     /**
00028      * @brief Vertices of triangle
00029      */
00030     std::array<Vec3<T>, 3> vertices_;
00031
00032 public:
00033     /**
00034      * @brief Construct a new Triangle object
00035      */
00036     Triangle();
00037
00038     /**
00039      * @brief Construct a new Triangle object from 3 points
00040      *
00041      * @param[in] p1 1st point
00042      * @param[in] p2 2nd point
00043      * @param[in] p3 3rd point
00044      */
00045     Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3);
00046
00047     /**
00048      * @brief Overloaded operator[] to get access to vertices
00049      *
00050      * @param[in] idx index of vertex
00051      * @return const Vec3<T>& const reference to vertex
00052      */
00053     const Vec3<T> &operator[](std::size_t idx) const;
00054
00055     /**
00056      * @brief Overloaded operator[] to get access to vertices
00057      *
00058      * @param[in] idx index of vertex
00059      * @return Vec3<T>& reference to vertex
00060      */
00061     Vec3<T> &operator[](std::size_t idx);
00062
00063     /**
00064      * @brief Get triangle's plane
00065      *
00066      * @return Plane<T>
00067      */
00068     Plane<T> getPlane() const;
00069
00070     /**
00071      * @brief Check is triangle valid
00072      *
00073      * @return true if triangle is valid
00074      * @return false if triangle is invalid
00075      */
00076     bool isValid() const;
00077 };
00078
00079 /**
00080  * @brief Triangle print operator
00081  *
00082  * @tparam T - floating point type of coordinates
00083  * @param[in, out] ost output stream
00084  * @param[in] tr Triangle to print
00085  * @return std::ostream& modified ostream instance

```



```

00086  */
00087  template <std::floating_point T>
00088  std::ostream &operator<<(std::ostream &ost, const Triangle<T> &tr)
00089  {
00090      ost << "Triangle: {";
00091      for (size_t i = 0; i < 3; ++i)
00092          ost << tr[i] << (i == 2 ? " : " : ", ");
00093
00094      ost << "}";
00095
00096      return ost;
00097  }
00098
00099  template <std::floating_point T>
00100  std::istream &operator>>(std::istream &ist, Triangle<T> &tr)
00101  {
00102      ist >> tr[0] >> tr[1] >> tr[2];
00103      return ist;
00104  }
00105
00106  template <std::floating_point T>
00107  Triangle<T>::Triangle() : vertices_()
00108  {}
00109
00110  template <std::floating_point T>
00111  Triangle<T>::Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3) : vertices_{p1, p2, p3}
00112  {}
00113
00114  template <std::floating_point T>
00115  const Vec3<T> &Triangle<T>::operator[](std::size_t idx) const
00116  {
00117      return vertices_[idx % 3];
00118  }
00119
00120  template <std::floating_point T>
00121  Vec3<T> &Triangle<T>::operator[](std::size_t idx)
00122  {
00123      return vertices_[idx % 3];
00124  }
00125
00126  template <std::floating_point T>
00127  Plane<T> Triangle<T>::getPlane() const
00128  {
00129      return Plane<T>::getBy3Points(vertices_[0], vertices_[1], vertices_[2]);
00130  }
00131
00132  template <std::floating_point T>
00133  bool Triangle<T>::isValid() const
00134  {
00135      auto edge1 = vertices_[1] - vertices_[0];
00136      auto edge2 = vertices_[2] - vertices_[0];
00137
00138      auto cross12 = cross(edge1, edge2);
00139      return (cross12 != Vec3<T>{});
00140  }
00141
00142 } // namespace geom
00143
00144 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__

```

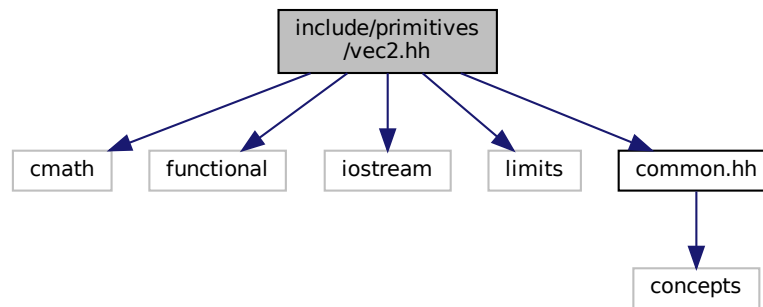
## 6.15 include/primitives/vec2.hh File Reference

```

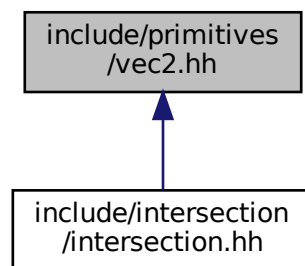
#include <cmath>
#include <functional>
#include <iostream>
#include <limits>
#include "common.hh"

```

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 <functional>
00006 #include <iostream>
00007 #include <limits>
00008
00009 #include "common.hh"
00010
00011 /**
00012  * @file vec2.hh
00013  * Vec2 class implementation
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Vec2
00021  * @brief Vec2 class realization
```

```

00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025  template <std::floating_point T>
00026  struct Vec2 final
00027  {
00028  private:
00029      /**
00030       * @brief Threshold static variable for numbers comparision
00031       */
00032       static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00033
00034  public:
00035      /**
00036       * @brief Vec2 coordinates
00037       */
00038       T x{}, y{};
00039
00040      /**
00041       * @brief Construct a new Vec2 object from 3 coordinates
00042       *
00043       * @param[in] coordX x coordinate
00044       * @param[in] coordY y coordinate
00045       */
00046       Vec2(T coordX, T coordY) : x(coordX), y(coordY)
00047       {}
00048
00049      /**
00050       * @brief Construct a new Vec2 object with equals coordinates
00051       *
00052       * @param[in] coordX coordinate (default to {})
00053       */
00054       explicit Vec2(T coordX = {}) : Vec2(coordX, coordX)
00055       {}
00056
00057      /**
00058       * @brief Overloaded += operator
00059       * Increments vector coordinates by corresponding coordinates of vec
00060       * @param[in] vec vector to incremented with
00061       * @return Vec2& reference to current instance
00062       */
00063       Vec2 &operator+=(const Vec2 &vec);
00064
00065      /**
00066       * @brief Overloaded -= operator
00067       * Decrements vector coordinates by corresponding coordinates of vec
00068       * @param[in] vec vector to decremented with
00069       * @return Vec2& reference to current instance
00070       */
00071       Vec2 &operator-=(const Vec2 &vec);
00072
00073      /**
00074       * @brief Unary - operator
00075       *
00076       * @return Vec2 negated Vec2 instance
00077       */
00078       Vec2 operator-() const;
00079
00080      /**
00081       * @brief Overloaded *= by number operator
00082       *
00083       * @tparam nType numeric type of value to multiply by
00084       * @param[in] val value to multiply by
00085       * @return Vec2& reference to vector instance
00086       */
00087       template <Number nType>
00088       Vec2 &operator*=(nType val);
00089
00090      /**
00091       * @brief Overloaded /= by number operator
00092       *
00093       * @tparam nType numeric type of value to divide by
00094       * @param[in] val value to divide by
00095       * @return Vec2& reference to vector instance
00096       *
00097       * @warning Does not check if val equals 0
00098       */
00099       template <Number nType>
00100       Vec2 &operator/=(nType val);
00101
00102      /**
00103       * @brief Dot product function
00104       *
00105       * @param rhs vector to dot product with
00106       * @return T dot product of two vectors
00107       */
00108       T dot(const Vec2 &rhs) const;

```

```

00109
00110 /**
00111  * @brief Calculate squared length of a vector function
00112  *
00113  * @return T length^2
00114  */
00115 T length2() const;
00116
00117 /**
00118  * @brief Calculate length of a vector function
00119  *
00120  * @return T length
00121  */
00122 T length() const;
00123
00124 /**
00125  * @brief Get the perpendicular to this vector
00126  *
00127  * @return Vec2 perpendicular vector
00128  */
00129 Vec2 getPerp() const;
00130
00131 /**
00132  * @brief Get normalized vector function
00133  *
00134  * @return Vec2 normalized vector
00135  */
00136 Vec2 normalized() const;
00137
00138 /**
00139  * @brief Normalize vector function
00140  *
00141  * @return Vec2& reference to instance
00142  */
00143 Vec2 &normalize();
00144
00145 /**
00146  * @brief Overloaded operator [] (non-const version)
00147  * To get access to coordinates
00148  * @param i index of coordinate (0 - x, 1 - y)
00149  * @return T& reference to coordinate value
00150  *
00151  * @note Coordinates calculated by mod 2
00152  */
00153 T &operator[](size_t i);
00154
00155 /**
00156  * @brief Overloaded operator [] (const version)
00157  * To get access to coordinates
00158  * @param i index of coordinate (0 - x, 1 - y)
00159  * @return T coordinate value
00160  *
00161  * @note Coordinates calculated by mod 2
00162  */
00163 T operator[](size_t i) const;
00164
00165 /**
00166  * @brief Check if vector is parallel to another
00167  *
00168  * @param[in] rhs vector to check parallelism with
00169  * @return true if vector is parallel
00170  * @return false otherwise
00171  */
00172 bool isPar(const Vec2 &rhs) const;
00173
00174 /**
00175  * @brief Check if vector is perpendicular to another
00176  *
00177  * @param[in] rhs vector to check perpendicularity with
00178  * @return true if vector is perpendicular
00179  * @return false otherwise
00180  */
00181 bool isPerp(const Vec2 &rhs) const;
00182
00183 /**
00184  * @brief Check if vector is equal to another
00185  *
00186  * @param[in] rhs vector to check equality with
00187  * @return true if vector is equal
00188  * @return false otherwise
00189  *
00190  * @note Equality check performs using isNumEq(T lhs, T rhs) function
00191  */
00192 bool isEqual(const Vec2 &rhs) const;
00193
00194 /**
00195  * @brief Check equality (with threshold) of two floating point numbers function

```

```

00196     *
00197     * @param[in] lhs first number
00198     * @param[in] rhs second number
00199     * @return true if numbers equals with threshold ( $|lhs - rhs| < threshold$ )
00200     * @return false otherwise
00201     *
00202     * @note Threshold defined by threshold_ static member
00203     */
00204     static bool isNumEq(T lhs, T rhs);
00205
00206     /**
00207     * @brief Set new threshold value
00208     *
00209     * @param[in] thres value to set
00210     */
00211     static void setThreshold(T thres);
00212
00213     /**
00214     * @brief Get current threshold value
00215     */
00216     static T getThreshold();
00217
00218     /**
00219     * @brief Set threshold to default value
00220     * @note default value equals float point epsilon
00221     */
00222     static void setDefThreshold();
00223 };
00224
00225 /**
00226 * @brief Overloaded + operator
00227 *
00228 * @tparam T vector template parameter
00229 * @param[in] lhs first vector
00230 * @param[in] rhs second vector
00231 * @return Vec2<T> sum of two vectors
00232 */
00233 template <std::floating_point T>
00234 Vec2<T> operator+(const Vec2<T> &lhs, const Vec2<T> &rhs)
00235 {
00236     Vec2<T> res{lhs};
00237     res += rhs;
00238     return res;
00239 }
00240
00241 /**
00242 * @brief Overloaded - operator
00243 *
00244 * @tparam T vector template parameter
00245 * @param[in] lhs first vector
00246 * @param[in] rhs second vector
00247 * @return Vec2<T> res of two vectors
00248 */
00249 template <std::floating_point T>
00250 Vec2<T> operator-(const Vec2<T> &lhs, const Vec2<T> &rhs)
00251 {
00252     Vec2<T> res{lhs};
00253     res -= rhs;
00254     return res;
00255 }
00256
00257 /**
00258 * @brief Overloaded multiple by value operator
00259 *
00260 * @tparam nT type of value to multiply by
00261 * @tparam T vector template parameter
00262 * @param[in] val value to multiply by
00263 * @param[in] rhs vector to multiply by value
00264 * @return Vec2<T> result vector
00265 */
00266 template <Number nT, std::floating_point T>
00267 Vec2<T> operator*(const nT &val, const Vec2<T> &rhs)
00268 {
00269     Vec2<T> res{rhs};
00270     res *= val;
00271     return res;
00272 }
00273
00274 /**
00275 * @brief Overloaded multiple by value operator
00276 *
00277 * @tparam nT type of value to multiply by
00278 * @tparam T vector template parameter
00279 * @param[in] val value to multiply by
00280 * @param[in] lhs vector to multiply by value
00281 * @return Vec2<T> result vector
00282 */

```

```

00283 template <Number nT, std::floating_point T>
00284 Vec2<T> operator*(const Vec2<T> &lhs, const nT &val)
00285 {
00286     Vec2<T> res{lhs};
00287     res *= val;
00288     return res;
00289 }
00290
00291 /**
00292  * @brief Overloaded divide by value operator
00293  *
00294  * @tparam nT type of value to divide by
00295  * @tparam T vector template parameter
00296  * @param[in] val value to divide by
00297  * @param[in] lhs vector to divide by value
00298  * @return Vec2<T> result vector
00299  */
00300 template <Number nT, std::floating_point T>
00301 Vec2<T> operator/(const Vec2<T> &lhs, const nT &val)
00302 {
00303     Vec2<T> res{lhs};
00304     res /= val;
00305     return res;
00306 }
00307
00308 /**
00309  * @brief Dot product function
00310  *
00311  * @tparam T vector template parameter
00312  * @param[in] lhs first vector
00313  * @param[in] rhs second vector
00314  * @return T dot production
00315  */
00316 template <std::floating_point T>
00317 T dot(const Vec2<T> &lhs, const Vec2<T> &rhs)
00318 {
00319     return lhs.dot(rhs);
00320 }
00321
00322 /**
00323  * @brief Vec2 equality operator
00324  *
00325  * @tparam T vector template parameter
00326  * @param[in] lhs first vector
00327  * @param[in] rhs second vector
00328  * @return true if vectors are equal
00329  * @return false otherwise
00330  */
00331 template <std::floating_point T>
00332 bool operator==(const Vec2<T> &lhs, const Vec2<T> &rhs)
00333 {
00334     return lhs.isEqual(rhs);
00335 }
00336
00337 /**
00338  * @brief Vec2 inequality operator
00339  *
00340  * @tparam T vector template parameter
00341  * @param[in] lhs first vector
00342  * @param[in] rhs second vector
00343  * @return true if vectors are not equal
00344  * @return false otherwise
00345  */
00346 template <std::floating_point T>
00347 bool operator!=(const Vec2<T> &lhs, const Vec2<T> &rhs)
00348 {
00349     return !(lhs == rhs);
00350 }
00351
00352 /**
00353  * @brief Vec2 print operator
00354  *
00355  * @tparam T vector template parameter
00356  * @param[in, out] ost output stream
00357  * @param[in] vec vector to print
00358  * @return std::ostream& modified stream instance
00359  */
00360 template <std::floating_point T>
00361 std::ostream &operator<<(std::ostream &ost, const Vec2<T> &vec)
00362 {
00363     ost << "(" << vec.x << ", " << vec.y << ")";
00364     return ost;
00365 }
00366
00367 using Vec2D = Vec2<double>;
00368 using Vec2F = Vec2<float>;
00369

```

```

00370 template <std::floating_point T>
00371 Vec2<T> &Vec2<T>::operator+=(const Vec2 &vec)
00372 {
00373     x += vec.x;
00374     y += vec.y;
00375     return *this;
00376 }
00377
00378 template <std::floating_point T>
00379 Vec2<T> &Vec2<T>::operator-=(const Vec2 &vec)
00380 {
00381     x -= vec.x;
00382     y -= vec.y;
00383     return *this;
00384 }
00385
00386 template <std::floating_point T>
00387 Vec2<T> Vec2<T>::operator-() const
00388 {
00389     return Vec2{-x, -y};
00390 }
00391
00392 template <std::floating_point T>
00393 template <Number nType>
00394 Vec2<T> &Vec2<T>::operator*=(nType val)
00395 {
00396     x *= val;
00397     y *= val;
00398     return *this;
00399 }
00400
00401 template <std::floating_point T>
00402 template <Number nType>
00403 Vec2<T> &Vec2<T>::operator/=(nType val)
00404 {
00405     x /= static_cast<T>(val);
00406     y /= static_cast<T>(val);
00407     return *this;
00408 }
00409
00410 template <std::floating_point T>
00411 T Vec2<T>::dot(const Vec2 &rhs) const
00412 {
00413     return x * rhs.x + y * rhs.y;
00414 }
00415
00416 template <std::floating_point T>
00417 T Vec2<T>::length2() const
00418 {
00419     return dot(*this);
00420 }
00421
00422 template <std::floating_point T>
00423 T Vec2<T>::length() const
00424 {
00425     return std::sqrt(length2());
00426 }
00427
00428 template <std::floating_point T>
00429 Vec2<T> Vec2<T>::getPerp() const
00430 {
00431     return {y, -x};
00432 }
00433
00434 template <std::floating_point T>
00435 Vec2<T> Vec2<T>::normalized() const
00436 {
00437     Vec2 res{*this};
00438     res.normalize();
00439     return res;
00440 }
00441
00442 template <std::floating_point T>
00443 Vec2<T> &Vec2<T>::normalize()
00444 {
00445     T len2 = length2();
00446     if (isNumEq(len2, 0) || isNumEq(len2, 1))
00447         return *this;
00448     return *this /= std::sqrt(len2);
00449 }
00450
00451 template <std::floating_point T>
00452 T &Vec2<T>::operator[](size_t i)

```



```

00457 {
00458     switch (i % 3)
00459     {
00460     case 0:
00461         return x;
00462     case 1:
00463         return y;
00464     default:
00465         throw std::logic_error{"Impossible case in operator[]\n"};
00466     }
00467 }
00468
00469 template <std::floating_point T>
00470 T Vec2<T>::operator[](size_t i) const
00471 {
00472     switch (i % 3)
00473     {
00474     case 0:
00475         return x;
00476     case 1:
00477         return y;
00478     default:
00479         throw std::logic_error{"Impossible case in operator[]\n"};
00480     }
00481 }
00482
00483 template <std::floating_point T>
00484 bool Vec2<T>::isPar(const Vec2 &rhs) const
00485 {
00486     auto det = x * rhs.y - rhs.x * y;
00487     return isNumEq(det, 0);
00488 }
00489
00490 template <std::floating_point T>
00491 bool Vec2<T>::isPerp(const Vec2 &rhs) const
00492 {
00493     return isNumEq(dot(rhs), 0);
00494 }
00495
00496 template <std::floating_point T>
00497 bool Vec2<T>::isEqual(const Vec2 &rhs) const
00498 {
00499     return isNumEq(x, rhs.x) && isNumEq(y, rhs.y);
00500 }
00501
00502 template <std::floating_point T>
00503 bool Vec2<T>::isNumEq(T lhs, T rhs)
00504 {
00505     return std::abs(rhs - lhs) < threshold_;
00506 }
00507
00508 template <std::floating_point T>
00509 void Vec2<T>::setThreshold(T thres)
00510 {
00511     threshold_ = thres;
00512 }
00513
00514 template <std::floating_point T>
00515 T Vec2<T>::getThreshold()
00516 {
00517     return threshold_;
00518 }
00519
00520 template <std::floating_point T>
00521 void Vec2<T>::setDefThreshold()
00522 {
00523     threshold_ = std::numeric_limits<T>::epsilon();
00524 }
00525
00526 } // namespace geom
00527
00528 #endif // __INCLUDE_PRIMITIVES_VEC2_HH__

```

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

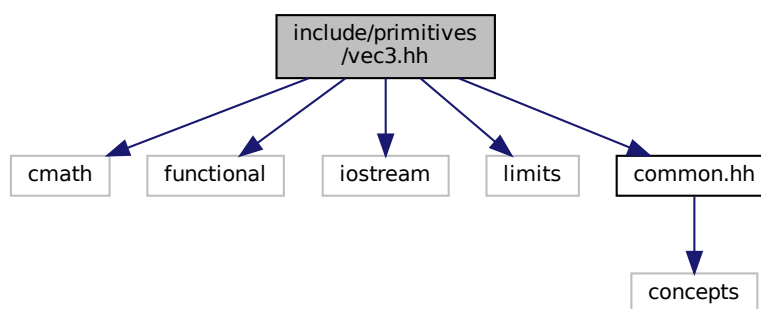
```

#include <cmath>
#include <functional>
#include <iostream>
#include <limits>

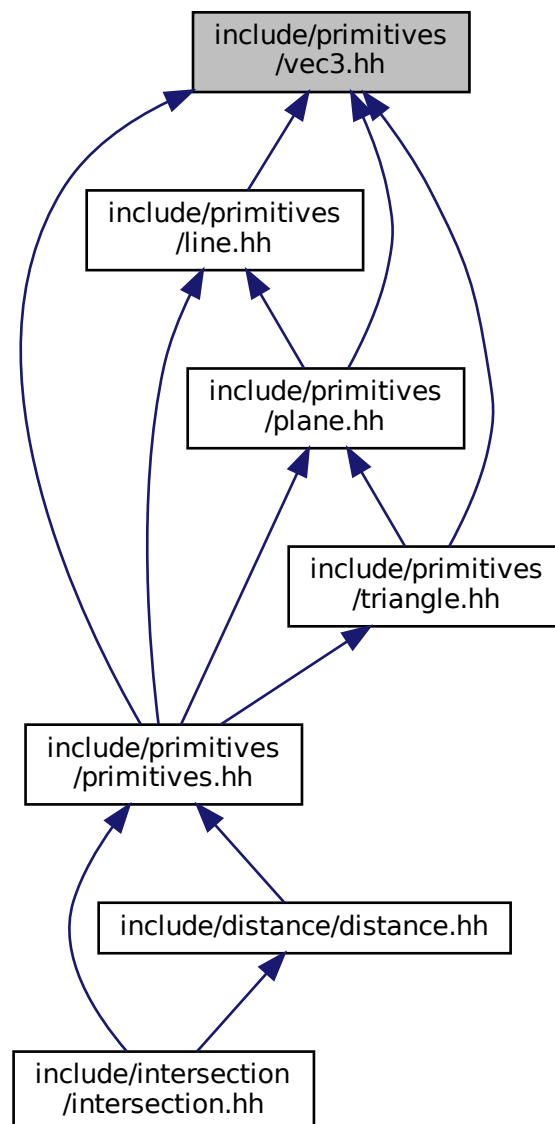
```

```
#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.*
- template<std::floating\_point T>  
std::istream & [geom::operator>>](#) (std::istream &ist, Vec3< T > &vec)  
*Vec3 scan 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 <functional>
00006 #include <iostream>
00007 #include <limits>
00008
00009 #include "common.hh"
00010
00011 /**
00012  * @file vec3.hh
00013  * Vec3 class implementation
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Vec3
00021  * @brief Vec3 class realization
00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025 template <std::floating_point T>
00026 struct Vec3 final
00027 {
00028 private:
00029     /**
00030      * @brief Threshold static variable for numbers comparision
00031      */
00032     static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00033
00034 public:
00035     /**
00036      * @brief Vec3 coordinates
00037      */
00038     T x{}, y{}, z{};
00039
00040     /**
00041      * @brief Construct a new Vec3 object from 3 coordinates
00042      *
00043      * @param[in] coordX x coordinate
00044      * @param[in] coordY y coordinate
00045      * @param[in] coordZ z coordinate
00046      */
00047     Vec3(T coordX, T coordY, T coordZ) : x(coordX), y(coordY), z(coordZ)
00048     {}
00049
00050     /**
00051      * @brief Construct a new Vec3 object with equals coordinates
00052      *
00053      * @param[in] coordX coordinate (default to {})
00054      */
00055     explicit Vec3(T coordX = {}) : Vec3(coordX, 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 Vec3& reference to current instance
00063      */
00064     Vec3 &operator+=(const Vec3 &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 Vec3& reference to current instance
00071      */
00072     Vec3 &operator-=(const Vec3 &vec);
00073
00074     /**
00075      * @brief Unary - operator
00076      *
00077      * @return Vec3 negated Vec3 instance
00078      */
00079     Vec3 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 Vec3& reference to vector instance
00087     */
00088     template <Number nType>
00089     Vec3 &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 Vec3& reference to vector instance
00097     *
00098     * @warning Does not check if val equals 0
00099     */
00100     template <Number nType>
00101     Vec3 &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 Vec3 &rhs) const;
00110
00111     /**
00112     * @brief Cross product function
00113     *
00114     * @param rhs vector to cross product with
00115     * @return Vec3 cross product of two vectors
00116     */
00117     Vec3 cross(const Vec3 &rhs) const;
00118
00119     /**
00120     * @brief Calculate squared length of a vector function
00121     *
00122     * @return T length^2
00123     */
00124     T length2() const;
00125
00126     /**
00127     * @brief Calculate length of a vector function
00128     *
00129     * @return T length
00130     */
00131     T length() const;
00132
00133     /**
00134     * @brief Get normalized vector function
00135     *
00136     * @return Vec3 normalized vector
00137     */
00138     Vec3 normalized() const;
00139
00140     /**
00141     * @brief Normalize vector function
00142     *
00143     * @return Vec3& reference to instance
00144     */
00145     Vec3 &normalize();
00146
00147     /**
00148     * @brief Overloaded operator [] (non-const version)
00149     * To get access to coordinates
00150     * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00151     * @return T& reference to coordinate value
00152     *
00153     * @note Coordinates calculated by mod 3
00154     */
00155     T &operator[](size_t i);
00156
00157     /**
00158     * @brief Overloaded operator [] (const version)
00159     * To get access to coordinates
00160     * @param i index of coordinate (0 - x, 1 - y, 2 - z)
00161     * @return T coordinate value
00162     *
00163     * @note Coordinates calculated by mod 3
00164     */
00165     T operator[](size_t i) const;
00166
00167     /**
00168     * @brief Check if vector is parallel to another
00169     *
00170     * @param[in] rhs vector to check parallelism with
00171     * @return true if vector is parallel
00172     * @return false otherwise

```

```

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

```

```

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

```



```

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

```

```

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

```

```
00521
00522 template <std::floating_point T>
00523 bool Vec3<T>::isPar(const Vec3 &rhs) const
00524 {
00525     return cross(rhs).isEqual(Vec3<T>{0});
00526 }
00527
00528 template <std::floating_point T>
00529 bool Vec3<T>::isPerp(const Vec3 &rhs) const
00530 {
00531     return isNumEq(dot(rhs), 0);
00532 }
00533
00534 template <std::floating_point T>
00535 bool Vec3<T>::isEqual(const Vec3 &rhs) const
00536 {
00537     return isNumEq(x, rhs.x) && isNumEq(y, rhs.y) && isNumEq(z, rhs.z);
00538 }
00539
00540 template <std::floating_point T>
00541 bool Vec3<T>::isNumEq(T lhs, T rhs)
00542 {
00543     return std::abs(rhs - lhs) < threshold_;
00544 }
00545
00546 template <std::floating_point T>
00547 void Vec3<T>::setThreshold(T thres)
00548 {
00549     threshold_ = thres;
00550 }
00551
00552 template <std::floating_point T>
00553 T Vec3<T>::getThreshold()
00554 {
00555     return threshold_;
00556 }
00557
00558 template <std::floating_point T>
00559 void Vec3<T>::setDefThreshold()
00560 {
00561     threshold_ = std::numeric_limits<T>::epsilon();
00562 }
00563
00564 } // namespace geom
00565
00566 #endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```

