Triangles

1.0.1

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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 368 of file vec2.hh.

4.1.2.2 Vec2F

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

Definition at line 369 of file vec2.hh.

4.1.2.3 Vec3D

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

Definition at line 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

T	- floating point type of coordinates

Parameters

pl	plane
pt	point

Returns

T signed distance between point and plane

Definition at line 26 of file distance.hh.

References geom::Plane< T >::dist(), dot(), and geom::Plane< T >::norm().

Referenced by geom::detail::helperMollerHaines(), and geom::detail::isOnOneSide().

4.1.3.2 isIntersect()

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

Checks intersection of 2 triangles.

Template Parameters

T	- floating point type of coordinates

Parameters

tr1	first triangle
tr2	second triangle

Returns

true if triangles are intersect

false if triangles are not intersect

Definition at line 150 of file intersection.hh.

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

4.1.3.3 intersect()

```
template<std::floating_point T>
std::variant< std::monostate, Line< T >, Plane< T > > geom::intersect (
            const Plane< T > & pl1,
            const Plane< T > & pl2 )
```

Intersect 2 planes and return result of intersection.

Common intersection case (parallel planes case is trivial):

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

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

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

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

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

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

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

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

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

Let's find a and b:

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

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

Intersection line equation:

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

Template Parameters

T	- floating point type of coordinates

Parameters

pl1	first plane
pl2	second plane

Returns

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

Definition at line 183 of file intersection.hh.

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

Referenced by geom::detail::isIntersectMollerHaines().

4.1.3.4 operator<<() [1/5]

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

Line print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in,out	*ost*	output stream
in	*line*	Line to print

Returns

std::ostream& modified ostream instance

Definition at line 89 of file line.hh.

References geom::Line< T >::dir(), and geom::Line< T >::org().

4.1.3.5 operator==() [1/4]

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

Line equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in	*lhs*	1st line
in	*rhs*	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

T	- floating point type of coordinates

Parameters

in	*lhs*	1st plane
in	*rhs*	2nd plane

Returns

true if planes are equal

false if planes are not equal

Definition at line 144 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

T	- floating point type of coordinates

Parameters

in,out	*ost*	output stream
in	*pl*	plane to print

Returns

std::ostream& modified ostream instance

Definition at line 158 of file plane.hh.

References geom::Plane< T >::dist(), and geom::Plane< T >::norm().

4.1.3.8 operator<<() [3/5]

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

Triangle print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in,out	*ost*	output stream
in	*tr*	Triangle to print

Returns

std::ostream& modified ostream instance

Definition at line 88 of file triangle.hh.

4.1.3.9 operator>>() [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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	second vector

Returns

Vec2<T> sum of two vectors

Definition at line 235 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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	second vector

Returns

Vec2<T> res of two vectors

Definition at line 251 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

nT	type of value to multiply by
T	vector template parameter

Parameters

in	*val*	value to multiply by
in	*rhs*	vector to multiply by value

Returns

Vec2<T> result vector

Definition at line 268 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

nT	type of value to multiply by
T	vector template parameter

Parameters

`in`	*val*	value to multiply by
`in`	*lhs*	vector to multiply by value

Returns

Vec2<T> result vector

Definition at line 285 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

nT	type of value to divide by
T	vector template parameter

Parameters

`in`	*val*	value to divide by
`in`	*lhs*	vector to divide by value

Returns

Vec2<T> result vector

Definition at line 302 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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	second vector

Returns

T dot production

Definition at line 318 of file vec2.hh.

References geom::Vec2< T >::dot().

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

4.1.3.16 operator==() [3/4]

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

Vec2 equality operator.

Template Parameters

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	second vector

Returns

true if vectors are equal

false otherwise

Definition at line 333 of file vec2.hh.

References geom::Vec2< T >::isEqual().

4.1.3.17 operator"!=() [1/2]

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

Vec2 inequality operator.

Template Parameters

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	second vector

Returns

true if vectors are not equal

false otherwise

Definition at line 348 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

T	vector template parameter

Parameters

in,out	*ost*	output stream
in	*vec*	vector to print

Returns

std::ostream& modified stream instance

Definition at line 362 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

4.1.3.19 operator+() [2/2]

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

Overloaded + operator.

Template Parameters

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

nT	type of value to multiply by
T	vector template parameter

Parameters

in	*val*	value to multiply by
in	*rhs*	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

nT	type of value to multiply by
T	vector template parameter

Parameters

in	*val*	value to multiply by
in	*lhs*	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

nT	type of value to divide by
T	vector template parameter

Parameters

in	*val*	value to divide by
in	*lhs*	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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

T	vector template parameter

Parameters

in	*lhs*	first vector
in	*rhs*	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

T	vector template parameter

Parameters

in,out	*ost*	output stream
in	*vec*	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

T	vector template parameter

Parameters

in,out	*ist*	input stram
in,out	*vec*	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

T	

Definition at line 15 of file common.hh.

4.2 geom::detail Namespace Reference

Typedefs

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

Functions

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

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

4.2.1 Typedef Documentation

4.2.1.1 Segment

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

Definition at line 104 of file intersection.hh.

4.2.1.2 Trian2

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

Definition at line 107 of file intersection.hh.

4.2.2 Function Documentation

4.2.2.1 isIntersect2D()

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

Definition at line 214 of file intersection.hh.

References computeInterval(), geom::Triangle< T >::getPlane(), and getTrian2().

Referenced by geom::isIntersect().

4.2.2.2 isIntersectMollerHaines()

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

Definition at line 239 of file intersection.hh.

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

Referenced by geom::isIntersect().

4.2.2.3 helperMollerHaines()

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

Definition at line 253 of file intersection.hh.

References geom::Line< T >::dir(), geom::distance(), geom::dot(), 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 295 of file intersection.hh.

Referenced by geom::isIntersect().

4.2.2.5 isIntersectValidInvalid()

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

Definition at line 304 of file intersection.hh.

Referenced by geom::isIntersect().

4.2.2.6 isOverlap()

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

Definition at line 313 of file intersection.hh.

Referenced by isIntersectMollerHaines().

4.2.2.7 isSameSign()

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

Definition at line 319 of file intersection.hh.

Referenced by helperMollerHaines(), and isOnOneSide().

4.2.2.8 isOnOneSide()

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

Definition at line 332 of file intersection.hh.

References geom::distance(), and isSameSign().

Referenced by geom::isIntersect().

4.2.2.9 getTrian2()

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

Definition at line 345 of file intersection.hh.

References isCounterClockwise(), and geom::Plane< T >::norm().

Referenced by isIntersect2D().

4.2.2.10 isCounterClockwise()

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

Definition at line 379 of file intersection.hh.

Referenced by getTrian2().

4.2.2.11 computeInterval()

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

Definition at line 399 of file intersection.hh.

References geom::dot().

Referenced by isIntersect2D().

Chapter 5

Class Documentation

5.1 geom::Line< T > Class Template Reference

Line class implementation.

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

Public Member Functions

- Line (const Vec3< T > &org, const Vec3< T > &dir)

  Construct a new Line object.
- const Vec3< T > & org () const

  Getter for origin vector.
- const Vec3< T > & dir () const

  Getter for direction vector.
- bool belongs (const Vec3< T > &point) const

  Checks is point belongs to line.
- bool isEqual (const Line &line) const

  Checks is ∗this equals to another line.

Static Public Member Functions

- static Line getBy2Points (const Vec3< T > &p1, const Vec3< T > &p2)

  Get line by 2 points.

5.1.1 Detailed Description

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

Line class implementation.

Template Parameters

T	- 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	*org*	origin vector
in	*dir*	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	*point*	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	*line*	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	*p1*	1st point
in	*p2*	2nd point

Returns

Line passing through two points

Definition at line 142 of file line.hh.

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

- include/primitives/line.hh

5.2 geom::Plane< T > Class Template Reference

Plane class realization.

`#include <plane.hh>`

Public Member Functions

- T dist () const

  Getter for distance.
- const Vec3< T > & norm () const

  Getter for normal vector.
- bool belongs (const Vec3< T > &point) const

  Checks if point belongs to plane.
- bool belongs (const Line< T > &line) const

  Checks if line belongs to plane.
- bool isEqual (const Plane &rhs) const

  Checks is ∗this equals to another plane.
- bool isPar (const Plane &rhs) const

  Checks is ∗this is parallel to another plane.

Static Public Member Functions

- static Plane getBy3Points (const Vec3< T > &pt1, const Vec3< T > &pt2, const Vec3< T > &pt3)

  Get plane by 3 points.
- static Plane getParametric (const Vec3< T > &org, const Vec3< T > &dir1, const Vec3< T > &dir2)

  Get plane from parametric plane equation.
- static Plane getNormalPoint (const Vec3< T > &norm, const Vec3< T > &point)

  Get plane from normal point plane equation.
- static Plane getNormalDist (const Vec3< T > &norm, T constant)

  Get plane form normal const plane equation.

5.2.1 Detailed Description

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

Plane class realization.

Template Parameters

T	- 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 172 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 178 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	*point*	const referene to point vector

Returns

true if point belongs to plane

false if point doesn't belong to plane

Definition at line 184 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	*line*	const referene to line

Returns

true if line belongs to plane

false if line doesn't belong to plane

Definition at line 190 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

in	*rhs*	const reference to another plane

Returns

true if planes are equal

false if planes are not equal

Definition at line 196 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

in	*rhs*	const reference to another plane

Returns

true if planes are parallel

false if planes are not parallel

Definition at line 202 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	*pt1*	1st point
in	*pt2*	2nd point
in	*pt3*	3rd point

Returns

Plane passing through three points

Definition at line 208 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	*org*	origin vector
in	*dir1*	1st direction vector
in	*dir2*	2nd direction vector

Returns

Plane

Definition at line 215 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	*norm*	normal vector
in	*point*	point lying on the plane

Returns

Plane

Definition at line 223 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	*norm*	normal vector
in	*constant*	distance

Returns

Plane

Definition at line 230 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

T	- 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	*p1*	1st point
in	*p2*	2nd point
in	*p3*	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	*idx*	index of vertex

Returns

const Vec3<T>& const reference to vertex

Definition at line 116 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

in	*idx*	index of vertex

Returns

Vec3<T>& reference to vertex

Definition at line 122 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 128 of file triangle.hh.

References geom::Plane< T >::getBy3Points().

Referenced by geom::isIntersect(), geom::detail::isIntersect2D(), and geom::detail::isIntersectMollerHaines().

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 134 of file triangle.hh.

References geom::cross().

Referenced by geom::isIntersect().

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

- include/primitives/triangle.hh

5.4 geom::Vec2< T > Class Template Reference

Vec2 class realization.

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

Public Member Functions

- Vec2 (T coordX, T coordY)

  Construct a new Vec2 object from 3 coordinates.
- Vec2 (T coordX={})

  Construct a new Vec2 object with equals coordinates.
- Vec2 & operator+= (const Vec2 &vec)

  Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.
- Vec2 & operator-= (const Vec2 &vec)

  Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.
- Vec2 operator- () const

  Unary - operator.
- template<Number nType>
  Vec2 & operator*= (nType val)

  *Overloaded *= by number operator.*
- template<Number nType>
  Vec2 & operator/= (nType val)

  Overloaded /= by number operator.
- T dot (const Vec2 &rhs) const

  Dot product function.
- T length2 () const

  Calculate squared length of a vector function.

- T length () const

    Calculate length of a vector function.
- Vec2 getPerp () const

    Get the perpendicular to this vector.
- Vec2 normalized () const

    Get normalized vector function.
- Vec2 & normalize ()

    Normalize vector function.
- T & operator[ ] (size_t i)

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

    Overloaded operator [] (const version) To get access to coordinates.
- bool isPar (const Vec2 &rhs) const

    Check if vector is parallel to another.
- bool isPerp (const Vec2 &rhs) const

    Check if vector is perpendicular to another.
- bool isEqual (const Vec2 &rhs) const

    Check if vector is equal to another.
- template<Number nType>
  Vec2< T > & operator*= (nType val)
- template<Number nType>
  Vec2< T > & operator/= (nType val)

Static Public Member Functions

- static bool isNumEq (T lhs, T rhs)

    Check equality (with threshold) of two floating point numbers function.
- static void setThreshold (T thres)

    Set new threshold value.
- static T getThreshold ()

    Get current threshold value.
- static void setDefThreshold ()

    Set threshold to default value.

Public Attributes

- T x {}

    Vec2 coordinates.
- T y {}

5.4.1 Detailed Description

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

Vec2 class realization.

Template Parameters

T	- floating point type of coordinates

Definition at line 27 of file vec2.hh.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Vec2() [1/2]

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

Construct a new Vec2 object from 3 coordinates.

Parameters

in	*coordX*	x coordinate
in	*coordY*	y coordinate

Definition at line 47 of file vec2.hh.

5.4.2.2 Vec2() [2/2]

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

Construct a new Vec2 object with equals coordinates.

Parameters

in	*coordX*	coordinate (default to {})

Definition at line 55 of file vec2.hh.

5.4.3 Member Function Documentation

5.4.3.1 operator+=()

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

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

Parameters

in	*vec*	vector to incremented with

Returns

Vec2& reference to current instance

Definition at line 372 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

5.4.3.2 operator-=()

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

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

Parameters

in	*vec*	vector to decremented with

Returns

Vec2& reference to current instance

Definition at line 381 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

5.4.3.3 operator-()

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

Unary - operator.

Returns

Vec2 negated Vec2 instance

Definition at line 390 of file vec2.hh.

5.4.3.4 operator*=() [1/2]

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

Overloaded *= by number operator.

Template Parameters

nType	numeric type of value to multiply by

Parameters

in	*val*	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

nType	numeric type of value to divide by

Parameters

in	*val*	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

rhs	vector to dot product with

Returns

T dot product of two vectors

Definition at line 416 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

Referenced by geom::dot().

5.4.3.7 length2()

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

Calculate squared length of a vector function.

Returns

T length^2

Definition at line 422 of file vec2.hh.

References geom::dot().

5.4.3.8 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 428 of file vec2.hh.

5.4.3.9 getPerp()

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

Get the perpendicular to this vector.

Returns

Vec2 perpendicular vector

Definition at line 434 of file vec2.hh.

5.4.3.10 normalized()

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

Get normalized vector function.

Returns

Vec2 normalized vector

Definition at line 440 of file vec2.hh.

References geom::Vec2< T >::normalize().

5.4.3.11 normalize()

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

Normalize vector function.

Returns

Vec2& reference to instance

Definition at line 448 of file vec2.hh.

Referenced by geom::Vec2< T >::normalized().

5.4.3.12 operator[]() [1/2]

```
template<std::floating_point T>
T & geom::Vec2< T >::operator[] (
             size_t i )
```

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

Parameters

i	index of coordinate (0 - x, 1 - y)

Returns

T& reference to coordinate value

Note

Coordinates calculated by mod 2

Definition at line 457 of file vec2.hh.

5.4.3.13 operator[]() [2/2]

```
template<std::floating_point T>
T geom::Vec2< T >::operator[] (
             size_t i ) const
```

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

Parameters

i	index of coordinate (0 - x, 1 - y)

Returns

T coordinate value

Note

Coordinates calculated by mod 2

Definition at line 471 of file vec2.hh.

5.4.3.14 isPar()

```
template<std::floating_point T>
bool geom::Vec2< T >::isPar (
             const Vec2< T > & rhs ) const
```

Check if vector is parallel to another.

Parameters

in	*rhs*	vector to check parallelism with

Returns

true if vector is parallel

false otherwise

Definition at line 485 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

5.4.3.15 isPerp()

```
template<std::floating_point T>
bool geom::Vec2< T >::isPerp (
             const Vec2< T > & rhs ) const
```

Check if vector is perpendicular to another.

Parameters

in	*rhs*	vector to check perpendicularity with

Returns

true if vector is perpendicular

false otherwise

Definition at line 492 of file vec2.hh.

References geom::dot().

5.4.3.16 isEqual()

```
template<std::floating_point T>
bool geom::Vec2< T >::isEqual (
             const Vec2< T > & rhs ) const
```

Check if vector is equal to another.

Parameters

in	*rhs*	vector to check equality with

Returns

true if vector is equal

false otherwise

Note

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

Definition at line 498 of file vec2.hh.

References geom::Vec2< T >::x, and geom::Vec2< T >::y.

Referenced by geom::operator==().

5.4.3.17 isNumEq()

```
template<std::floating_point T>
bool geom::Vec2< T >::isNumEq (
             T lhs,
             T rhs ) [static]
```

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

Parameters

in	*lhs*	first number
in	*rhs*	second number

Returns

true if numbers equals with threshold (|lhs - rhs| < threshold)

false otherwise

Note

Threshold defined by threshold_ static member

Definition at line 504 of file vec2.hh.

5.4.3.18 setThreshold()

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

Set new threshold value.

Parameters

in	*thres*	value to set

Definition at line 510 of file vec2.hh.

5.4.3.19 getThreshold()

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

Get current threshold value.

Definition at line 516 of file vec2.hh.

5.4.3.20 setDefThreshold()

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

Set threshold to default value.

Note

default value equals float point epsilon

Definition at line 522 of file vec2.hh.

5.4.3.21 operator*=() [2/2]

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

Definition at line 397 of file vec2.hh.

5.4.3.22 operator/=() [2/2]

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

Definition at line 407 of file vec2.hh.

5.4.4 Member Data Documentation

5.4.4.1 x

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

Vec2 coordinates.

Definition at line 39 of file vec2.hh.

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

5.4.4.2 y

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

Definition at line 39 of file vec2.hh.

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

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

- include/primitives/vec2.hh

5.5 geom::Vec3< T > 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

T	- 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	*coordX*	x coordinate
in	*coordY*	y coordinate
in	*coordZ*	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	*coordX*	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

nType	numeric type of value to multiply by

Parameters

in	*val*	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

nType	numeric type of value to divide by

Parameters

in	*val*	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

rhs	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

rhs	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 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 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	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	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

in	*rhs*	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

in	*rhs*	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

in	*rhs*	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	*lhs*	first number
in	*rhs*	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	*thres*	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.

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

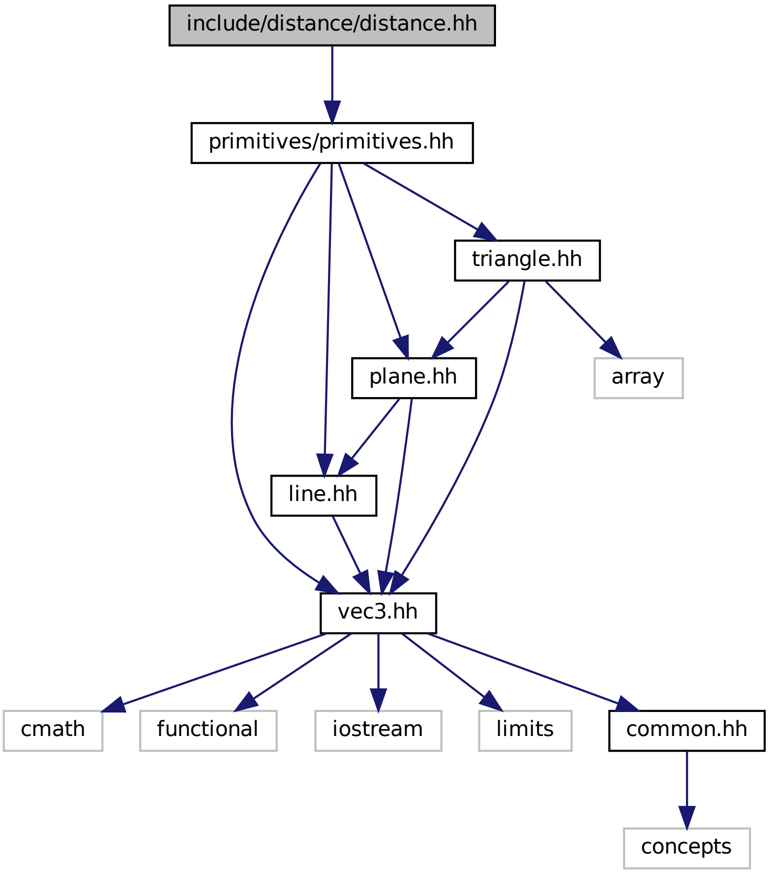
Chapter 6

File Documentation

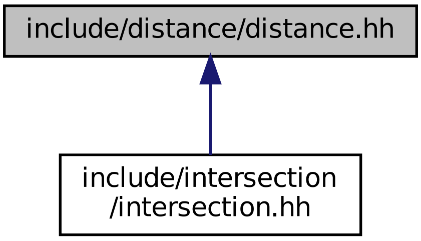
6.1 include/distance/distance.hh File Reference

```
#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.

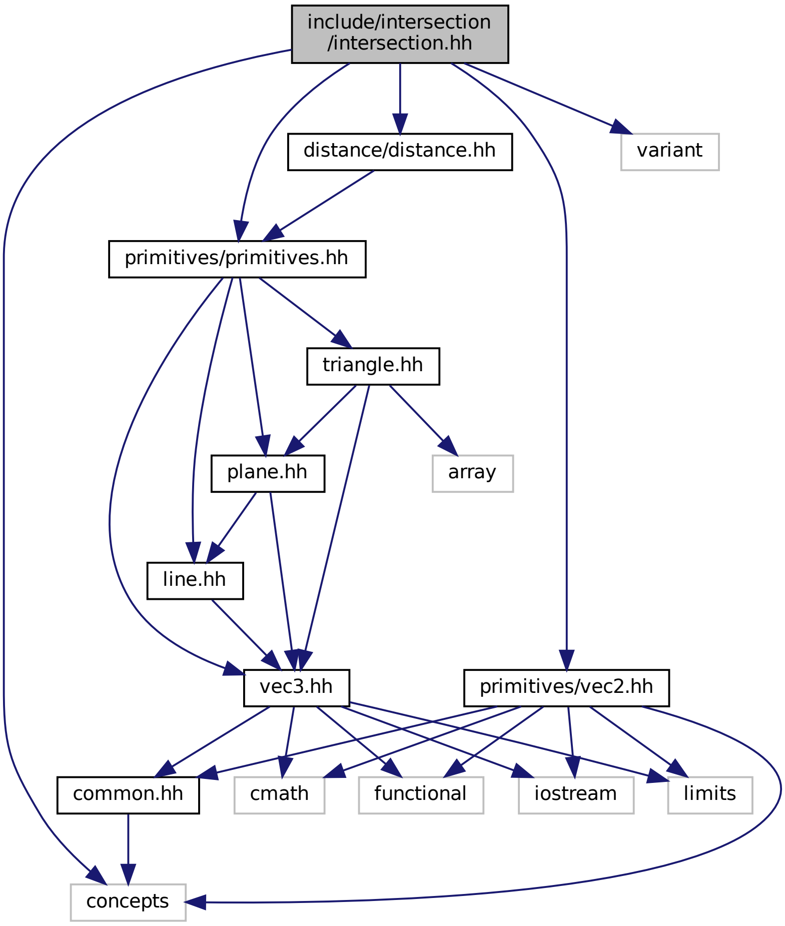
6.2 distance.hh

```
#ifndef __INCLUDE_DISTANCE_DISTANCE_HH__
#define __INCLUDE_DISTANCE_DISTANCE_HH__

#include "primitives/primitives.hh"

namespace geom
{

/**
 * @brief Calculates signed distance between point and plane
 *
 * @tparam T - floating point type of coordinates
 * @param pl plane
 * @param pt point
 * @return T signed distance between point and plane
 */
template <std::floating_point T>
T distance(const Plane<T> &pl, const Vec3<T> &pt);

} // namespace geom

namespace geom
{

template <std::floating_point T>
T distance(const Plane<T> &pl, const Vec3<T> &pt)
{
  return dot(pt, pl.norm()) - pl.dist();
}

} // namespace geom

#endif // __INCLUDE_DISTANCE_DISTANCE_HH__
```

6.3 include/intersection/intersection.hh File Reference

```
#include <concepts>
#include <variant>
#include "distance/distance.hh"
#include "primitives/primitives.hh"
#include "primitives/vec2.hh"
```

Include dependency graph for intersection.hh:



Namespaces

- geom

  line.hh Line class implementation
- geom::detail

Typedefs

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

Functions

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

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

  Intersect 2 planes and return result of intersection.
- template<std::floating_point T>
  bool geom::detail::isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool geom::detail::isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  Segment< T > geom::detail::helperMollerHaines (const Triangle< T > &tr, const Plane< T > &pl, const Line< T > &l)
- template<std::floating_point T>
  bool geom::detail::isIntersectBothInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool geom::detail::isIntersectValidInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool geom::detail::isOverlap (Segment< T > &segm1, Segment< T > &segm2)
- template<std::forward_iterator It>
  bool geom::detail::isSameSign (It begin, It end)
- template<std::floating_point T>
  bool geom::detail::isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  Trian2< T > geom::detail::getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  bool geom::detail::isCounterClockwise (Trian2< T > &tr)
- template<std::floating_point T>
  Segment< T > geom::detail::computeInterval (const Trian2< T > &tr, const Vec2< T > &d)

6.4 intersection.hh

```
#ifndef __INCLUDE_INTERSECTION_INTERSECTION_HH__
#define __INCLUDE_INTERSECTION_INTERSECTION_HH__

#include <concepts>
#include <variant>

#include "distance/distance.hh"
#include "primitives/primitives.hh"
#include "primitives/vec2.hh"

namespace geom
{

/**
 * @brief Checks intersection of 2 triangles
 *
 * @tparam T - floating point type of coordinates
 * @param tr1 first triangle
 * @param tr2 second triangle
 * @return true if triangles are intersect
 * @return false if triangles are not intersect
 */
template <std::floating_point T>
bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2);

/**
 * @brief Intersect 2 planes and return result of intersection
 * @details
 * Common intersection case (parallel planes case is trivial):
 *
 * Let \f$ \overrightarrow{P} \f$ - point in space
 *
 * \f$ pl_1 \f$ equation: \f$ \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1 \f$
 *
 * \f$ pl_2 \f$ equation: \f$ \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2 \f$
 *
 * Intersection line direction: \f$ \overrightarrow{dir} = \overrightarrow{n}_1 \times
 * \overrightarrow{n}_2 \f$
 *
 * Let origin of intersection line be a linear combination of \f$ \overrightarrow{n}_1 \f$
 * and \f$ \overrightarrow{n}_2 \f$: \f[ \overrightarrow{P} = a \cdot \overrightarrow{n}_1
 * + b \cdot \overrightarrow{n}_2 \f]
 *
 * \f$ \overrightarrow{P} \f$ must satisfy both \f$ pl_1 \f$ and \f$ pl_1 \f$ equations:
 * \f[
 * \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1
 * \Leftrightarrow
 * \overrightarrow{n}_1
 * \cdot
 * \left(
 *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
 *  \right)
 *  = d_1
 * \Leftrightarrow
 * a + b \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2  = d_1
 * \f]
 * \f[
 * \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2
 * \Leftrightarrow
 * \overrightarrow{n}_2
 * \cdot
 * \left(
 *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
 *  \right) = d_2
 * \Leftrightarrow
 * a \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 + b = d_2
 * \f]
 *
 * Let's find \f$a\f$ and \f$b\f$:
 * \f[
 * a = \frac{
 *  d_2 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_1
 * }{
 *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
 * }
 * \f]
 * \f[
 * b = \frac{
 * d_1 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_2
 * }{
 *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
 * }
 * \f]
 *
 * Intersection line equation:
 * \f[
 * \overrightarrow{r}(t) = \overrightarrow{P} + t \cdot \overrightarrow{n}_1 \times
 * \overrightarrow{n}_2 = (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) +
 * t \cdot \overrightarrow{n}_1 \times \overrightarrow{n}_2 \f]
 *
 * @tparam T - floating point type of coordinates
 * @param pl1 first plane
 * @param pl2 second plane
 * @return std::variant<std::monostate, Line<T>, Plane<T>»
 */
template <std::floating_point T>
std::variant<std::monostate, Line<T>, Plane<T>» intersect(const Plane<T> &pl1,
                                                          const Plane<T> &pl2);

namespace detail
{

template <typename T>
using Segment = std::pair<T, T>;

template <std::floating_point T>
using Trian2 = std::array<Vec2<T>, 3>;

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

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

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

```

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

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

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

template <std::forward_iterator It>
bool isSameSign(It begin, It end);

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

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

template <std::floating_point T>
bool isCounterClockwise(Trian2<T> &tr);

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

} // namespace detail
} // namespace geom

namespace geom
{

template <std::floating_point T>
bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  /* TODO: handle invalid triangles case */
  auto isInv1 = !tr1.isValid();
  auto isInv2 = !tr2.isValid();

  if (isInv1 && isInv2)
    return detail::isIntersectBothInvalid(tr1, tr2);

  if (isInv1)
    return detail::isIntersectValidInvalid(tr2, tr1);

  if (isInv2)
    return detail::isIntersectValidInvalid(tr1, tr2);

  auto pl1 = tr1.getPlane();
  if (detail::isOnOneSide(pl1, tr2))
    return false;

  auto pl2 = tr2.getPlane();
  if (pl1 == pl2)
    return detail::isIntersect2D(tr1, tr2);

  if (pl1.isPar(pl2))
    return false;

  if (detail::isOnOneSide(pl2, tr1))
    return false;

  return detail::isIntersectMollerHaines(tr1, tr2);
}

template <std::floating_point T>
std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &pl1,
                                                          const Plane<T> &pl2)
{
  const auto &n1 = pl1.norm();
  const auto &n2 = pl2.norm();

  auto dir = cross(n1, n2);

  /* if planes are parallel */
  if (Vec3<T>{0} == dir)
  {
    if (pl1 == pl2)
      return pl1;

    return std::monostate{};
  }

  auto n1n2 = dot(n1, n2);
  auto d1 = pl1.dist();
  auto d2 = pl2.dist();

  auto a = (d2 * n1n2 - d1) / (n1n2 * n1n2 - 1);
  auto b = (d1 * n1n2 - d2) / (n1n2 * n1n2 - 1);

  return Line<T>{(a * n1) + (b * n2), dir};
}

namespace detail
{

template <std::floating_point T>
bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  auto pl = tr1.getPlane();

  auto trian1 = getTrian2(pl, tr1);
  auto trian2 = getTrian2(pl, tr2);

  for (auto trian : {trian1, trian2})
  {
    for (size_t i0 = 0, i1 = 2; i0 < 3; i1 = i0, ++i0)
    {
      auto d = (trian[i0] - trian[i1]).getPerp();

      auto s1 = computeInterval(trian1, d);
      auto s2 = computeInterval(trian2, d);

      if (s2.second < s1.first || s1.second < s2.first)
        return false;
    }
  }

  return true;
}

template <std::floating_point T>
bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  auto pl1 = tr1.getPlane();
  auto pl2 = tr2.getPlane();

  auto l = std::get<Line<T>»(intersect(pl1, pl2));

  auto params1 = helperMollerHaines(tr1, pl2, l);
  auto params2 = helperMollerHaines(tr2, pl1, l);

  return isOverlap(params1, params2);
}

template <std::floating_point T>
Segment<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
{
  /* Project the triangle vertices onto line */
  std::array<T, 3> vert{};
  for (size_t i = 0; i < 3; ++i)
    vert[i] = dot(l.dir(), tr[i] - l.org());

  std::array<T, 3> sdist{};
  for (size_t i = 0; i < 3; ++i)
    sdist[i] = distance(pl, tr[i]);

  auto isSameSign = [](const auto &num1, const auto &num2) {
    if (num1 * num2 > Vec3<T>::getThreshold())
      return true;
    return Vec3<T>::isNumEq(num1, 0) && Vec3<T>::isNumEq(num2, 0);
  };

  std::array<bool, 3> isOneSide{};
  for (size_t i = 0; i < 3; ++i)
    isOneSide[i] = isSameSign(sdist[i], sdist[(i + 1) % 3]);

  /* Looking for vertex which is alone on it's side */
  size_t rogue = 0;
  for (size_t i = 0; i < 3; ++i)
    if (isOneSide[i])
      rogue = (i + 2) % 3;

  std::vector<T> segm{};
  std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};

  for (size_t i : arr)
    segm.push_back(vert[i] +
                   (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]));

  /* Sort segment's ends */
  if (segm[0] > segm[1])
    std::swap(segm[0], segm[1]);

  return {segm[0], segm[1]};
}
```

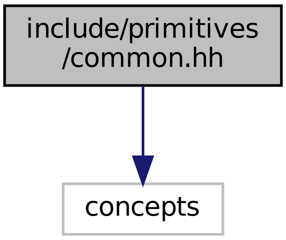
```

template <std::floating_point T>
bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  std::cout « "both invalid" « std::endl;
  std::cout « "tr1: " « tr1 « std::endl;
  std::cout « "tr2: " « tr2 « std::endl;
  return false;
}

template <std::floating_point T>
bool isIntersectValidInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  std::cout « "one invalid" « std::endl;
  std::cout « "tr1: " « tr1 « std::endl;
  std::cout « "tr2: " « tr2 « std::endl;
  return false;
}

template <std::floating_point T>
bool isOverlap(Segment<T> &segm1, Segment<T> &segm2)
{
  return (segm2.first <= segm1.second) && (segm2.second >= segm1.first);
}

template <std::forward_iterator It>
bool isSameSign(It begin, It end)
{
  auto cur = begin;
  auto prev = begin;

  for (++cur; cur != end; ++cur)
    if ((*cur) * (*prev) <= 0)
      return false;

  return true;
}

template <std::floating_point T>
bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
{
  std::array<T, 3> sdist{};
  for (size_t i = 0; i < 3; ++i)
    sdist[i] = distance(pl, tr[i]);

  if (detail::isSameSign(sdist.begin(), sdist.end()))
    return true;

  return false;
}

template <std::floating_point T>
Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr)
{
  auto norm = pl.norm();

  const Vec3<T> x{1, 0, 0};
  const Vec3<T> y{0, 1, 0};
  const Vec3<T> z{0, 0, 1};

  std::array<Vec3<T>, 3> xyz{x, y, z};
  std::array<T, 3> xyzDot;

  std::transform(xyz.begin(), xyz.end(), xyzDot.begin(),
                 [&norm](const auto &axis) { return std::abs(dot(axis, norm)); });

  auto maxIt = std::max_element(xyzDot.begin(), xyzDot.end());
  auto maxIdx = static_cast<size_t>(maxIt - xyzDot.begin());

  Trian2<T> res;
  for (size_t i = 0; i < 3; ++i)
    for (size_t j = 0, k = 0; j < 2; ++j, ++k)
    {
      if (k == maxIdx)
        ++k;

      res[i][j] = tr[i][k];
    }

  if (!isCounterClockwise(res))
    std::swap(res[0], res[1]);

  return res;
}

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

```
{
  /**
   * The triangle is counterclockwise ordered if \delta > 0
   * and clockwise ordered if \delta < 0.
   *
   *              +  1  1  1 +
   * \delta = det | x0 x1 x2 | = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0)
   *              + y0 y1 y2 +                        + (x0 * y1 - x1 * y0)
   *
   */

  auto x0 = tr[0][0], x1 = tr[1][0], x2 = tr[2][0];
  auto y0 = tr[0][1], y1 = tr[1][1], y2 = tr[2][1];

  auto delta = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0) + (x0 * y1 - x1 * y0);
  return (delta > 0);
}

template <std::floating_point T>
Segment<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
{
  auto init = dot(d, tr[0]);
  auto min = init;
  auto max = init;

  for (size_t i = 1; i < 3; ++i)
    if (auto val = dot(d, tr[i]); val < min)
      min = val;
    else if (val > max)
      max = val;

  return {min, max};
}

} // namespace detail
} // namespace geom

#endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__
```

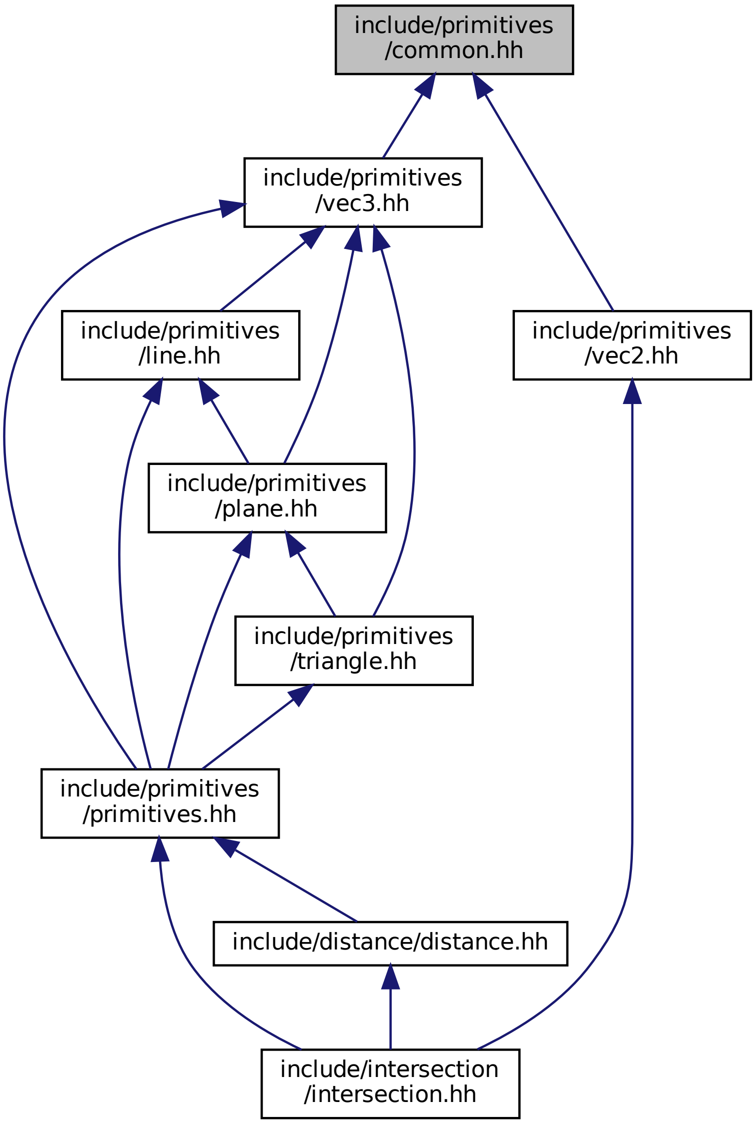
6.5 include/primitives/common.hh File Reference

`#include <concepts>`

Include dependency graph for common.hh:



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


include/primitives /common.hh
include/primitives /vec3.hh
include/primitives /line.hh
include/primitives /vec2.hh
include/primitives /plane.hh
include/primitives /triangle.hh
include/primitives /primitives.hh
include/distance/distance.hh
include/intersection /intersection.hh

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.

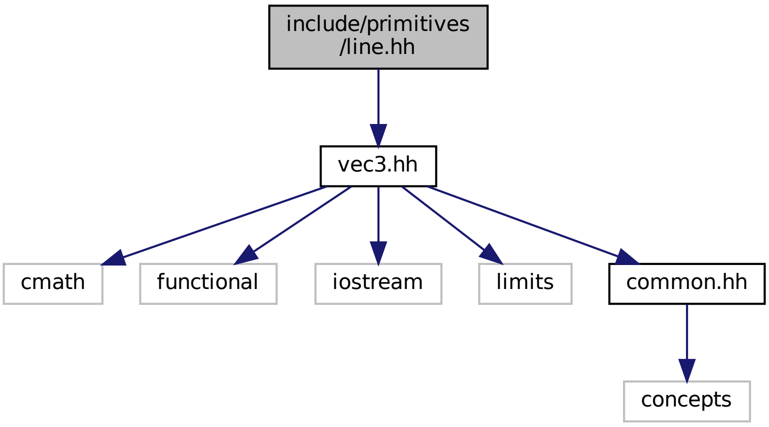
6.6 common.hh

```
#ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
#define __INCLUDE_PRIMITIVES_COMMON_HH__

#include <concepts>

namespace geom
{
/**
 * @concept Number
 * @brief Useful concept which represents floating point and integral types
 *
 * @tparam T
 */
template <class T>
concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>;

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_COMMON_HH__
```

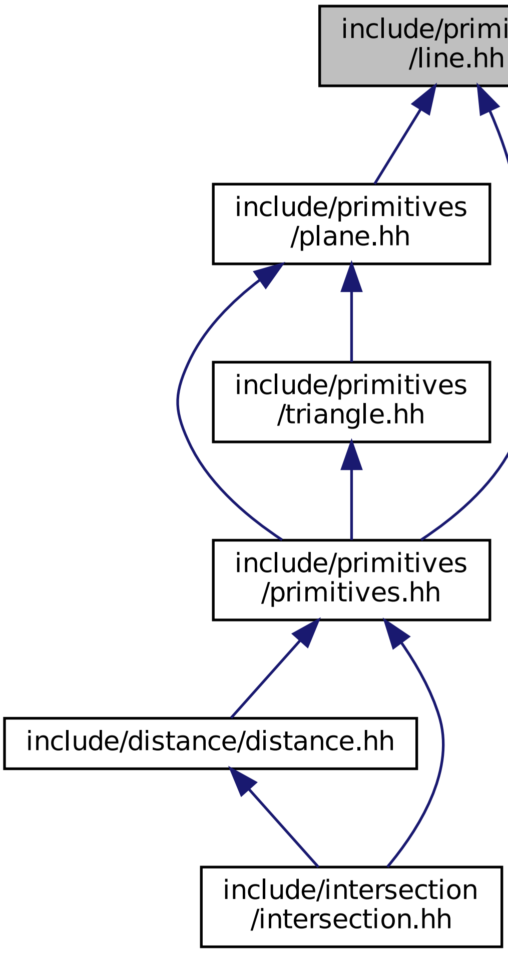
6.7 include/primitives/line.hh File Reference

`#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.

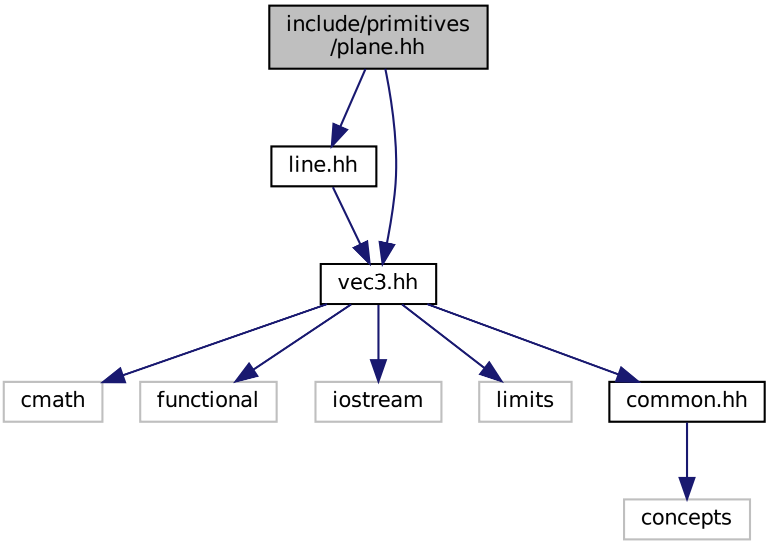
6.8 line.hh

```
#ifndef __INCLUDE_PRIMITIVES_LINE_HH__
#define __INCLUDE_PRIMITIVES_LINE_HH__

#include "vec3.hh"

/**
 * @brief line.hh
 * Line class implementation
 */

namespace geom
{

/**
 * @class Line
 * @brief Line class implementation
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
class Line final
{
private:
  /**
   * @brief Origin and direction vectors
   */
  Vec3<T> org_{}, dir_{};

public:
  /**
   * @brief Construct a new Line object
   *
   * @param[in] org origin vector
   * @param[in] dir direction vector
   */
  Line(const Vec3<T> &org, const Vec3<T> &dir);

  /**
   * @brief Getter for origin vector
   *
   * @return const Vec3<T>& const reference to origin vector
   */
  const Vec3<T> &org() const;

  /**
   * @brief Getter for direction vector
   *
   * @return const Vec3<T>& const reference to direction vector
   */
  const Vec3<T> &dir() const;

  /**
   * @brief Checks is point belongs to line
   *
   * @param[in] point const reference to point vector
   * @return true if point belongs to line
   * @return false if point doesn't belong to line
   */
  bool belongs(const Vec3<T> &point) const;

  /**
   * @brief Checks is *this equals to another line
   *
   * @param[in] line const reference to another line
   * @return true if lines are equal
   * @return false if lines are not equal
   */
  bool isEqual(const Line &line) const;

  /**
   * @brief Get line by 2 points
   *
   * @param[in] p1 1st point
   * @param[in] p2 2nd point
   * @return Line passing through two points
   */
  static Line getBy2Points(const Vec3<T> &p1, const Vec3<T> &p2);
};

/**
 * @brief Line print operator
 *
 * @tparam T - floating point type of coordinates
 * @param[in, out] ost output stream
 * @param[in] line Line to print
 * @return std::ostream& modified ostream instance
 */
template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const Line<T> &line)
{
  ost « line.org() « " + " « line.dir() « " * t";
  return ost;
}

/**
 * @brief Line equality operator
 *
 * @tparam T - floating point type of coordinates
 * @param[in] lhs 1st line
 * @param[in] rhs 2nd line
 * @return true if lines are equal
 * @return false if lines are not equal
 */
template <std::floating_point T>
bool operator==(const Line<T> &lhs, const Line<T> &rhs)
{
  return lhs.isEqual(rhs);
}

template <std::floating_point T>
Line<T>::Line(const Vec3<T> &org, const Vec3<T> &dir) : org_{org}, dir_{dir}
{
  if (dir_ == Vec3<T>{0})
    throw std::logic_error{"Direction vector equals zero."};
}

template <std::floating_point T>
const Vec3<T> &Line<T>::org() const
{
  return org_;
}

template <std::floating_point T>
const Vec3<T> &Line<T>::dir() const
{
  return dir_;
}

template <std::floating_point T>
bool Line<T>::belongs(const Vec3<T> &point) const
{
  return dir_.cross(point - org_) == Vec3<T>{0};
}

template <std::floating_point T>
bool Line<T>::isEqual(const Line<T> &line) const
{
  return belongs(line.org_) && dir_.isPar(line.dir_);
}

template <std::floating_point T>
Line<T> Line<T>::getBy2Points(const Vec3<T> &p1, const Vec3<T> &p2)
{
  return Line<T>{p1, p2 - p1};
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_LINE_HH__
```

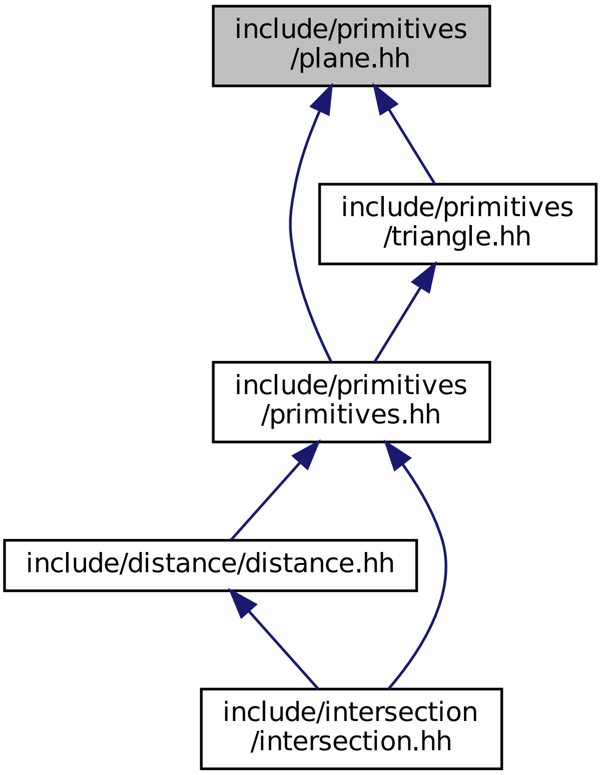
6.9 include/primitives/plane.hh File Reference

```
#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

```
#ifndef __INCLUDE_PRIMITIVES_PLANE_HH__
#define __INCLUDE_PRIMITIVES_PLANE_HH__

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

/**
 * @brief
 * Plane class implementation
 */

namespace geom
{

/**
 * @class Plane
 * @brief Plane class realization
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
class Plane final
{
private:
  /**
   * @brief Normal vector, length equals to 1
   */
  Vec3<T> norm_{};

  /**
   * @brief Distance from zero to plane
   */
  T dist_{};

  /**
   * @brief Construct a new Plane object from normal vector and distance
   *
   * @param[in] norm normal vector
   * @param[in] dist distance from plane to zero
   */
  Plane(const Vec3<T> &norm, T dist);

public:
  /**
   * @brief Getter for distance
   *
   * @return T value of distance
   */
  T dist() const;

  /**
   * @brief Getter for normal vector
   *
   * @return const Vec3<T>& const reference to normal vector
   */
  const Vec3<T> &norm() const;

  /**
   * @brief Checks if point belongs to plane
   *
   * @param[in] point const referene to point vector
   * @return true if point belongs to plane
   * @return false if point doesn’t belong to plane
   */
  bool belongs(const Vec3<T> &point) const;

  /**
   * @brief Checks if line belongs to plane
   *
   * @param[in] line const referene to line
   * @return true if line belongs to plane
   * @return false if line doesn’t belong to plane
   */
  bool belongs(const Line<T> &line) const;

  /**
   * @brief Checks is *this equals to another plane
   *
   * @param[in] rhs const reference to another plane
   * @return true if planes are equal
   * @return false if planes are not equal
   */
  bool isEqual(const Plane &rhs) const;

  /**
```

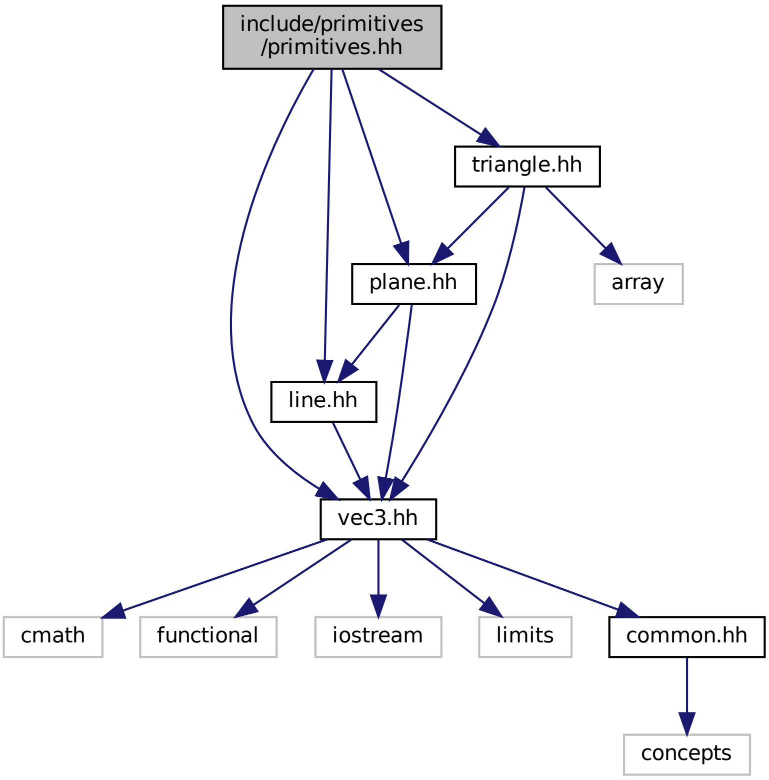
```
   * @brief Checks is *this is parallel to another plane
   *
   * @param[in] rhs const reference to another plane
   * @return true if planes are parallel
   * @return false if planes are not parallel
   */
  bool isPar(const Plane &rhs) const;

  /**
   * @brief Get plane by 3 points
   *
   * @param[in] pt1 1st point
   * @param[in] pt2 2nd point
   * @param[in] pt3 3rd point
   * @return Plane passing through three points
   */
  static Plane getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3);

  /**
   * @brief Get plane from parametric plane equation
   *
   * @param[in] org origin vector
   * @param[in] dir1 1st direction vector
   * @param[in] dir2 2nd direction vector
   * @return Plane
   */
  static Plane getParametric(const Vec3<T> &org, const Vec3<T> &dir1,
                             const Vec3<T> &dir2);

  /**
   * @brief Get plane from normal point plane equation
   *
   * @param[in] norm normal vector
   * @param[in] point point lying on the plane
   * @return Plane
   */
  static Plane getNormalPoint(const Vec3<T> &norm, const Vec3<T> &point);

  /**
   * @brief Get plane form normal const plane equation
   *
   * @param[in] norm normal vector
   * @param[in] constant distance
   * @return Plane
   */
  static Plane getNormalDist(const Vec3<T> &norm, T constant);
};

/**
 * @brief Plane equality operator
 *
 * @tparam T - floating point type of coordinates
 * @param[in] lhs 1st plane
 * @param[in] rhs 2nd plane
 * @return true if planes are equal
 * @return false if planes are not equal
 */
template <std::floating_point T>
bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
{
  return lhs.isEqual(rhs);
}

/**
 * @brief Plane print operator
 *
 * @tparam T - floating point type of coordinates
 * @param[in, out] ost output stream
 * @param[in] pl plane to print
 * @return std::ostream& modified ostream instance
 */
template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const Plane<T> &pl)
{
  ost « pl.norm() « " * X = " « pl.dist();
  return ost;
}

template <std::floating_point T>
Plane<T>::Plane(const Vec3<T> &norm, T dist) : norm_(norm), dist_(dist)
{
  if (norm == Vec3<T>{0})
    throw std::logic_error{"normal vector equals to zero"};
}

template <std::floating_point T>
T Plane<T>::dist() const
```

```
{
  return dist_;
}

template <std::floating_point T>
const Vec3<T> &Plane<T>::norm() const
{
  return norm_;
}

template <std::floating_point T>
bool Plane<T>::belongs(const Vec3<T> &pt) const
{
  return Vec3<T>::isNumEq(norm_.dot(pt), dist_);
}

template <std::floating_point T>
bool Plane<T>::belongs(const Line<T> &line) const
{
  return norm_.isPerp(line.dir()) && belongs(line.org());
}

template <std::floating_point T>
bool Plane<T>::isEqual(const Plane &rhs) const
{
  return (norm_ * dist_ == rhs.norm_ * rhs.dist_) && (norm_.isPar(rhs.norm_));
}

template <std::floating_point T>
bool Plane<T>::isPar(const Plane &rhs) const
{
  return norm_.isPar(rhs.norm_);
}

template <std::floating_point T>
Plane<T> Plane<T>::getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2,
                                const Vec3<T> &pt3)
{
  return getParametric(pt1, pt2 - pt1, pt3 - pt1);
}

template <std::floating_point T>
Plane<T> Plane<T>::getParametric(const Vec3<T> &org, const Vec3<T> &dir1,
                                 const Vec3<T> &dir2)
{
  auto norm = dir1.cross(dir2);
  return getNormalPoint(norm, org);
}

template <std::floating_point T>
Plane<T> Plane<T>::getNormalPoint(const Vec3<T> &norm, const Vec3<T> &pt)
{
  auto normalized = norm.normalized();
  return Plane{normalized, normalized.dot(pt)};
}

template <std::floating_point T>
Plane<T> Plane<T>::getNormalDist(const Vec3<T> &norm, T dist)
{
  auto normalized = norm.normalized();
  return Plane{normalized, dist};
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_PLANE_HH__
```

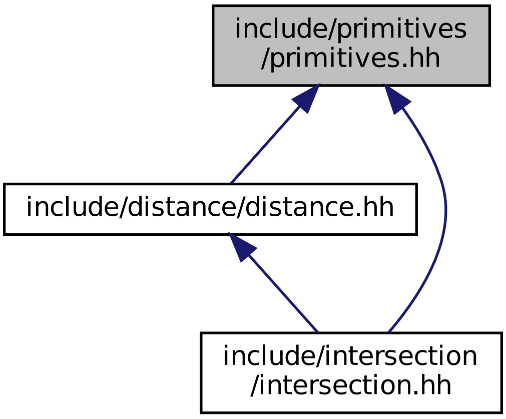
6.11 include/primitives/primitives.hh File Reference

```
#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:



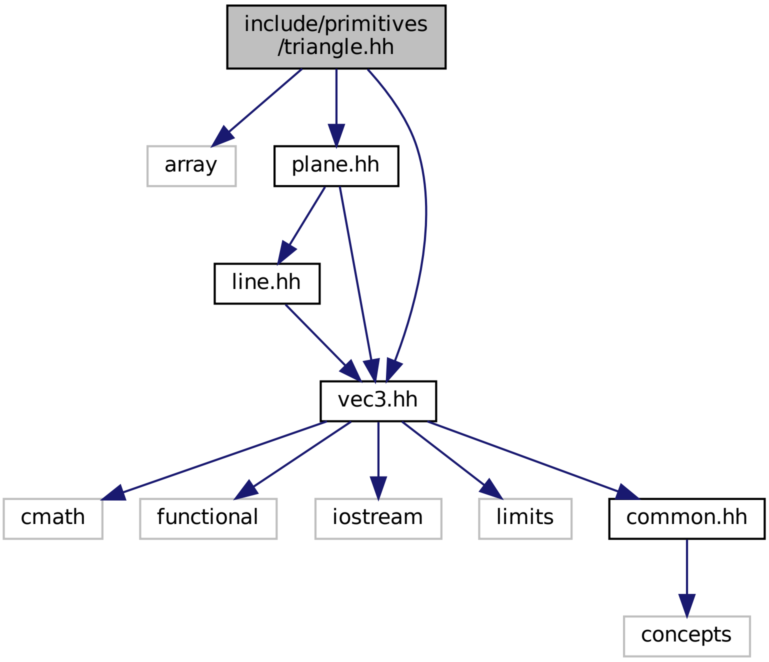
6.12 primitives.hh

```
#ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
#define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__

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

#endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

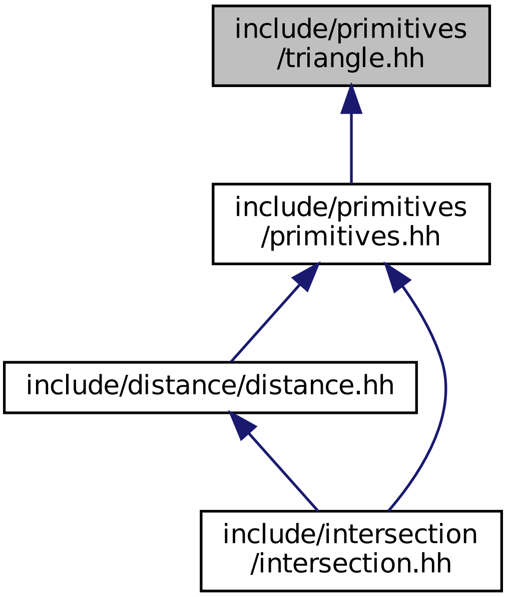
6.13 include/primitives/triangle.hh File Reference

```
#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:


include/primitives /triangle.hh
include/primitives /primitives.hh
include/distance/distance.hh
include/intersection /intersection.hh

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)

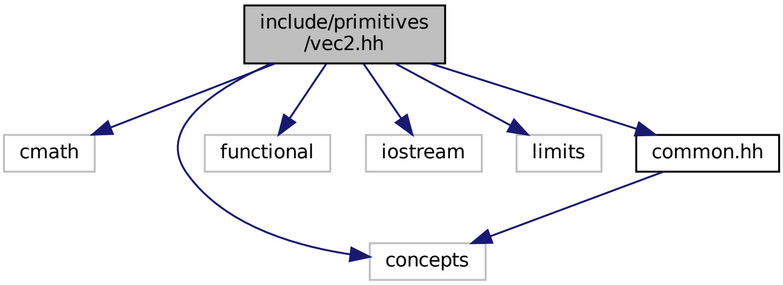
6.14 triangle.hh

```
#ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
#define __INCLUDE_PRIMITIVES_TRIANGLE_HH__

#include <array>

#include "plane.hh"
#include "vec3.hh"

/**
 * @brief triangle.hh
 * Triangle class implementation
 */

namespace geom
{

/**
 * @class Triangle
 * @brief Triangle class implementation
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
class Triangle final
{
private:
  /**
   * @brief Vertices of triangle
   */
  std::array<Vec3<T>, 3> vertices_;

public:
  /**
   * @brief Construct a new Triangle object
   */
  Triangle();

  /**
   * @brief Construct a new Triangle object from 3 points
   *
   * @param[in] p1 1st point
   * @param[in] p2 2nd point
   * @param[in] p3 3rd point
   */
  Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3);

  /**
   * @brief Overloaded operator[] to get access to vertices
   *
   * @param[in] idx index of vertex
   * @return const Vec3<T>& const reference to vertex
   */
  const Vec3<T> &operator[](std::size_t idx) const;

  /**
   * @brief Overloaded operator[] to get access to vertices
   *
   * @param[in] idx index of vertex
   * @return Vec3<T>& reference to vertex
   */
  Vec3<T> &operator[](std::size_t idx);

  /**
   * @brief Get triangle's plane
   *
   * @return Plane<T>
   */
  Plane<T> getPlane() const;

  /**
   * @brief Check is triangle valid
   *
   * @return true if triangle is valid
   * @return false if triangle is invalid
   */
  bool isValid() const;
};

/**
 * @brief Triangle print operator
 *
 * @tparam T - floating point type of coordinates
 * @param[in, out] ost output stream
 * @param[in] tr Triangle to print
 * @return std::ostream& modified ostream instance
 */
template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const Triangle<T> &tr)
{
  ost « "Triangle: {";
  for (size_t i = 0; i < 3; ++i)
    ost « tr[i] « (i == 2 ? "" : ", ");

  ost « "}";

  return ost;
}

template <std::floating_point T>
std::istream &operator»(std::istream &ist, Triangle<T> &tr)
{
  ist » tr[0] » tr[1] » tr[2];
  return ist;
}

template <std::floating_point T>
Triangle<T>::Triangle() : vertices_()
{}

template <std::floating_point T>
Triangle<T>::Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3)
  : vertices_{p1, p2, p3}
{}

template <std::floating_point T>
const Vec3<T> &Triangle<T>::operator[](std::size_t idx) const
{
  return vertices_[idx % 3];
}

template <std::floating_point T>
Vec3<T> &Triangle<T>::operator[](std::size_t idx)
{
  return vertices_[idx % 3];
}

template <std::floating_point T>
Plane<T> Triangle<T>::getPlane() const
{
  return Plane<T>::getBy3Points(vertices_[0], vertices_[1], vertices_[2]);
}

template <std::floating_point T>
bool Triangle<T>::isValid() const
{
  auto edge1 = vertices_[1] - vertices_[0];
  auto edge2 = vertices_[2] - vertices_[0];

  auto cross12 = cross(edge1, edge2);
  return (cross12 != Vec3<T>{});
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__
```

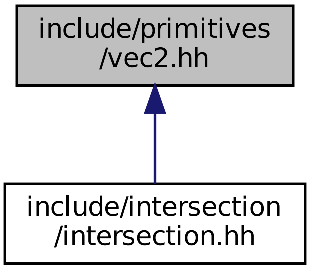
6.15 include/primitives/vec2.hh File Reference

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

Include dependency graph for vec2.hh:



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



Classes

- class geom::Vec2< T >

    Vec2 class realization.

Namespaces

- geom

    line.hh Line class implementation

Typedefs

- using geom::Vec2D = Vec2< double >
- using geom::Vec2F = Vec2< float >

Functions

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

  Overloaded + operator.
- template<std::floating_point T>
  Vec2< T > geom::operator- (const Vec2< T > &lhs, const Vec2< T > &rhs)

  Overloaded - operator.
- template<Number nT, std::floating_point T>
  Vec2< T > geom::operator* (const nT &val, const Vec2< T > &rhs)

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec2< T > geom::operator* (const Vec2< T > &lhs, const nT &val)

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec2< T > geom::operator/ (const Vec2< T > &lhs, const nT &val)

  Overloaded divide by value operator.
- template<std::floating_point T>
  T geom::dot (const Vec2< T > &lhs, const Vec2< T > &rhs)

  Dot product function.
- template<std::floating_point T>
  bool geom::operator== (const Vec2< T > &lhs, const Vec2< T > &rhs)

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

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

  Vec2 print operator.

6.15.1 Detailed Description

Vec2 class implementation

Definition in file vec2.hh.

6.16 vec2.hh

```
#ifndef __INCLUDE_PRIMITIVES_VEC2_HH__
#define __INCLUDE_PRIMITIVES_VEC2_HH__

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

#include "common.hh"

/**
 * @file vec2.hh
 * Vec2 class implementation
 */

namespace geom
{

/**
 * @class Vec2
 * @brief Vec2 class realization
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
struct Vec2 final
{
private:
  /**
   * @brief Threshold static variable for numbers comparision
   */
  static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();

public:
  /**
   * @brief Vec2 coordinates
   */
  T x{}, y{};

  /**
   * @brief Construct a new Vec2 object from 3 coordinates
   *
   * @param[in] coordX x coordinate
   * @param[in] coordY y coordinate
   */
  Vec2(T coordX, T coordY) : x(coordX), y(coordY)
  {}

  /**
   * @brief Construct a new Vec2 object with equals coordinates
   *
   * @param[in] coordX coordinate (default to {})
   */
  explicit Vec2(T coordX = {}) : Vec2(coordX, coordX)
  {}

  /**
   * @brief Overloaded += operator
   * Increments vector coordinates by corresponding coordinates of vec
   * @param[in] vec vector to incremented with
   * @return Vec2& reference to current instance
   */
  Vec2 &operator+=(const Vec2 &vec);

  /**
   * @brief Overloaded -= operator
   * Decrements vector coordinates by corresponding coordinates of vec
   * @param[in] vec vector to decremented with
   * @return Vec2& reference to current instance
   */
  Vec2 &operator-=(const Vec2 &vec);

  /**
   * @brief Unary - operator
   *
   * @return Vec2 negated Vec2 instance
   */
  Vec2 operator-() const;

  /**
   * @brief Overloaded *= by number operator
   *
   * @tparam nType numeric type of value to multiply by
   * @param[in] val value to multiply by
   * @return Vec2& reference to vector instance
   */
  template <Number nType>
  Vec2 &operator*=(nType val);

  /**
   * @brief Overloaded /= by number operator
   *
   * @tparam nType numeric type of value to divide by
   * @param[in] val value to divide by
   * @return Vec2& reference to vector instance
   *
   * @warning Does not check if val equals 0
   */
  template <Number nType>
  Vec2 &operator/=(nType val);

  /**
   * @brief Dot product function
   *
   * @param rhs vector to dot product with
   * @return T dot product of two vectors
   */
```

```
T dot(const Vec2 &rhs) const;

/**
 * @brief Calculate squared length of a vector function
 *
 * @return T length^2
 */
T length2() const;

/**
 * @brief Calculate length of a vector function
 *
 * @return T length
 */
T length() const;

/**
 * @brief Get the perpendicular to this vector
 *
 * @return Vec2 perpendicular vector
 */
Vec2 getPerp() const;

/**
 * @brief Get normalized vector function
 *
 * @return Vec2 normalized vector
 */
Vec2 normalized() const;

/**
 * @brief Normalize vector function
 *
 * @return Vec2& reference to instance
 */
Vec2 &normalize();

/**
 * @brief Overloaded operator [] (non-const version)
 * To get access to coordinates
 * @param i index of coordinate (0 - x, 1 - y)
 * @return T& reference to coordinate value
 *
 * @note Coordinates calculated by mod 2
 */
T &operator[](size_t i);

/**
 * @brief Overloaded operator [] (const version)
 * To get access to coordinates
 * @param i index of coordinate (0 - x, 1 - y)
 * @return T coordinate value
 *
 * @note Coordinates calculated by mod 2
 */
T operator[](size_t i) const;

/**
 * @brief Check if vector is parallel to another
 *
 * @param[in] rhs vector to check parallelism with
 * @return true if vector is parallel
 * @return false otherwise
 */
bool isPar(const Vec2 &rhs) const;

/**
 * @brief Check if vector is perpendicular to another
 *
 * @param[in] rhs vector to check perpendicularity with
 * @return true if vector is perpendicular
 * @return false otherwise
 */
bool isPerp(const Vec2 &rhs) const;

/**
 * @brief Check if vector is equal to another
 *
 * @param[in] rhs vector to check equality with
 * @return true if vector is equal
 * @return false otherwise
 *
 * @note Equality check performs using isNumEq(T lhs, T rhs) function
 */
bool isEqual(const Vec2 &rhs) const;

/**
```

```
   * @brief Check equality (with threshold) of two floating point numbers function
   *
   * @param[in] lhs first number
   * @param[in] rhs second number
   * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
   * @return false otherwise
   *
   * @note Threshold defined by threshold_ static member
   */
  static bool isNumEq(T lhs, T rhs);

  /**
   * @brief Set new threshold value
   *
   * @param[in] thres value to set
   */
  static void setThreshold(T thres);

  /**
   * @brief Get current threshold value
   */
  static T getThreshold();

  /**
   * @brief Set threshold to default value
   * @note default value equals float point epsilon
   */
  static void setDefThreshold();
};

/**
 * @brief Overloaded + operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return Vec2<T> sum of two vectors
 */
template <std::floating_point T>
Vec2<T> operator+(const Vec2<T> &lhs, const Vec2<T> &rhs)
{
  Vec2<T> res{lhs};
  res += rhs;
  return res;
}

/**
 * @brief Overloaded - operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return Vec2<T> res of two vectors
 */
template <std::floating_point T>
Vec2<T> operator-(const Vec2<T> &lhs, const Vec2<T> &rhs)
{
  Vec2<T> res{lhs};
  res -= rhs;
  return res;
}

/**
 * @brief Overloaded multiple by value operator
 *
 * @tparam nT type of value to multiply by
 * @tparam T vector template parameter
 * @param[in] val value to multiply by
 * @param[in] rhs vector to multiply by value
 * @return Vec2<T> result vector
 */
template <Number nT, std::floating_point T>
Vec2<T> operator*(const nT &val, const Vec2<T> &rhs)
{
  Vec2<T> res{rhs};
  res *= val;
  return res;
}

/**
 * @brief Overloaded multiple by value operator
 *
 * @tparam nT type of value to multiply by
 * @tparam T vector template parameter
 * @param[in] val value to multiply by
 * @param[in] lhs vector to multiply by value
 * @return Vec2<T> result vector
 */
template <Number nT, std::floating_point T>
Vec2<T> operator*(const Vec2<T> &lhs, const nT &val)
{
  Vec2<T> res{lhs};
  res *= val;
  return res;
}

/**
 * @brief Overloaded divide by value operator
 *
 * @tparam nT type of value to divide by
 * @tparam T vector template parameter
 * @param[in] val value to divide by
 * @param[in] lhs vector to divide by value
 * @return Vec2<T> result vector
 */
template <Number nT, std::floating_point T>
Vec2<T> operator/(const Vec2<T> &lhs, const nT &val)
{
  Vec2<T> res{lhs};
  res /= val;
  return res;
}

/**
 * @brief Dot product function
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return T dot production
 */
template <std::floating_point T>
T dot(const Vec2<T> &lhs, const Vec2<T> &rhs)
{
  return lhs.dot(rhs);
}

/**
 * @brief Vec2 equality operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return true if vectors are equal
 * @return false otherwise
 */
template <std::floating_point T>
bool operator==(const Vec2<T> &lhs, const Vec2<T> &rhs)
{
  return lhs.isEqual(rhs);
}

/**
 * @brief Vec2 inequality operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return true if vectors are not equal
 * @return false otherwise
 */
template <std::floating_point T>
bool operator!=(const Vec2<T> &lhs, const Vec2<T> &rhs)
{
  return !(lhs == rhs);
}

/**
 * @brief Vec2 print operator
 *
 * @tparam T vector template parameter
 * @param[in, out] ost output stream
 * @param[in] vec vector to print
 * @return std::ostream& modified stream instance
 */
template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const Vec2<T> &vec)
{
  ost « "(" « vec.x « ", " « vec.y « ")";
  return ost;
}

using Vec2D = Vec2<double>;
using Vec2F = Vec2<float>;
```

```

template <std::floating_point T>
Vec2<T> &Vec2<T>::operator+=(const Vec2 &vec)
{
  x += vec.x;
  y += vec.y;

  return *this;
}

template <std::floating_point T>
Vec2<T> &Vec2<T>::operator-=(const Vec2 &vec)
{
  x -= vec.x;
  y -= vec.y;

  return *this;
}

template <std::floating_point T>
Vec2<T> Vec2<T>::operator-() const
{
  return Vec2{-x, -y};
}

template <std::floating_point T>
template <Number nType>
Vec2<T> &Vec2<T>::operator*=(nType val)
{
  x *= val;
  y *= val;

  return *this;
}

template <std::floating_point T>
template <Number nType>
Vec2<T> &Vec2<T>::operator/=(nType val)
{
  x /= static_cast<T>(val);
  y /= static_cast<T>(val);

  return *this;
}

template <std::floating_point T>
T Vec2<T>::dot(const Vec2 &rhs) const
{
  return x * rhs.x + y * rhs.y;
}

template <std::floating_point T>
T Vec2<T>::length2() const
{
  return dot(*this);
}

template <std::floating_point T>
T Vec2<T>::length() const
{
  return std::sqrt(length2());
}

template <std::floating_point T>
Vec2<T> Vec2<T>::getPerp() const
{
  return {y, -x};
}

template <std::floating_point T>
Vec2<T> Vec2<T>::normalized() const
{
  Vec2 res{*this};
  res.normalize();
  return res;
}

template <std::floating_point T>
Vec2<T> &Vec2<T>::normalize()
{
  T len2 = length2();
  if (isNumEq(len2, 0) || isNumEq(len2, 1))
    return *this;
  return *this /= std::sqrt(len2);
}

template <std::floating_point T>
```

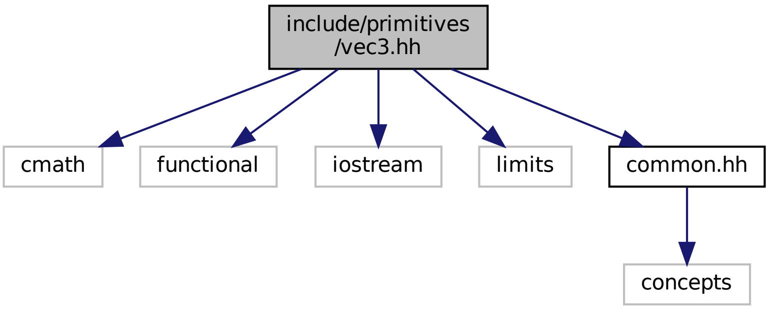
```
T &Vec2<T>::operator[](size_t i)
{
  switch (i % 3)
  {
  case 0:
    return x;
  case 1:
    return y;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

template <std::floating_point T>
T Vec2<T>::operator[](size_t i) const
{
  switch (i % 3)
  {
  case 0:
    return x;
  case 1:
    return y;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

template <std::floating_point T>
bool Vec2<T>::isPar(const Vec2 &rhs) const
{
  auto det = x * rhs.y - rhs.x * y;
  return isNumEq(det, 0);
}

template <std::floating_point T>
bool Vec2<T>::isPerp(const Vec2 &rhs) const
{
  return isNumEq(dot(rhs), 0);
}

template <std::floating_point T>
bool Vec2<T>::isEqual(const Vec2 &rhs) const
{
  return isNumEq(x, rhs.x) && isNumEq(y, rhs.y);
}

template <std::floating_point T>
bool Vec2<T>::isNumEq(T lhs, T rhs)
{
  return std::abs(rhs - lhs) < threshold_;
}

template <std::floating_point T>
void Vec2<T>::setThreshold(T thres)
{
  threshold_ = thres;
}

template <std::floating_point T>
T Vec2<T>::getThreshold()
{
  return threshold_;
}

template <std::floating_point T>
void Vec2<T>::setDefThreshold()
{
  threshold_ = std::numeric_limits<T>::epsilon();
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_VEC2_HH__
```

6.17 include/primitives/vec3.hh File Reference

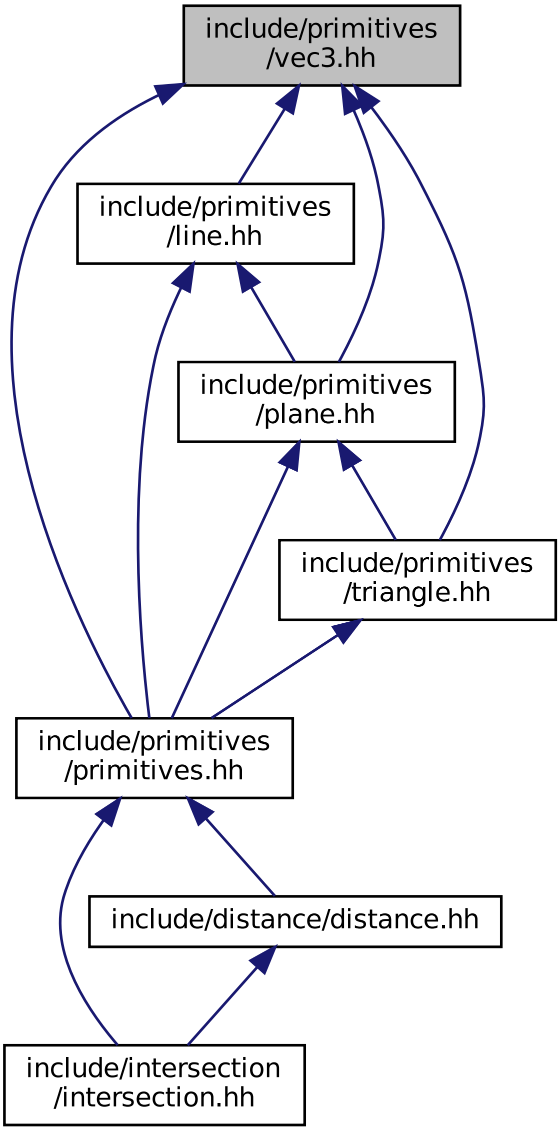
```
#include <cmath>
#include <functional>
#include <iostream>
#include <limits>
```

```
#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

```
#ifndef __INCLUDE_PRIMITIVES_VEC3_HH__
#define __INCLUDE_PRIMITIVES_VEC3_HH__

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

#include "common.hh"

/**
 * @file vec3.hh
 * Vec3 class implementation
 */

namespace geom
{

/**
 * @class Vec3
 * @brief Vec3 class realization
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
struct Vec3 final
{
private:
  /**
   * @brief Threshold static variable for numbers comparision
   */
  static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();

public:
  /**
   * @brief Vec3 coordinates
   */
  T x{}, y{}, z{};

  /**
   * @brief Construct a new Vec3 object from 3 coordinates
   *
   * @param[in] coordX x coordinate
   * @param[in] coordY y coordinate
   * @param[in] coordZ z coordinate
   */
  Vec3(T coordX, T coordY, T coordZ) : x(coordX), y(coordY), z(coordZ)
  {}

  /**
   * @brief Construct a new Vec3 object with equals coordinates
   *
   * @param[in] coordX coordinate (default to {})
   */
  explicit Vec3(T coordX = {}) : Vec3(coordX, coordX, coordX)
  {}

  /**
   * @brief Overloaded += operator
   * Increments vector coordinates by corresponding coordinates of vec
   * @param[in] vec vector to incremented with
   * @return Vec3& reference to current instance
   */
  Vec3 &operator+=(const Vec3 &vec);

  /**
   * @brief Overloaded -= operator
   * Decrements vector coordinates by corresponding coordinates of vec
   * @param[in] vec vector to decremented with
   * @return Vec3& reference to current instance
   */
  Vec3 &operator-=(const Vec3 &vec);

  /**
   * @brief Unary - operator
   *
   * @return Vec3 negated Vec3 instance
   */
  Vec3 operator-() const;

  /**
   * @brief Overloaded *= by number operator
   *
   * @tparam nType numeric type of value to multiply by
   * @param[in] val value to multiply by
   * @return Vec3& reference to vector instance
   */
  template <Number nType>
  Vec3 &operator*=(nType val);

  /**
   * @brief Overloaded /= by number operator
   *
   * @tparam nType numeric type of value to divide by
   * @param[in] val value to divide by
   * @return Vec3& reference to vector instance
   *
   * @warning Does not check if val equals 0
   */
  template <Number nType>
  Vec3 &operator/=(nType val);

  /**
   * @brief Dot product function
   *
   * @param rhs vector to dot product with
   * @return T dot product of two vectors
   */
  T dot(const Vec3 &rhs) const;

  /**
   * @brief Cross product function
   *
   * @param rhs vector to cross product with
   * @return Vec3 cross product of two vectors
   */
  Vec3 cross(const Vec3 &rhs) const;

  /**
   * @brief Calculate squared length of a vector function
   *
   * @return T length^2
   */
  T length2() const;

  /**
   * @brief Calculate length of a vector function
   *
   * @return T length
   */
  T length() const;

  /**
   * @brief Get normalized vector function
   *
   * @return Vec3 normalized vector
   */
  Vec3 normalized() const;

  /**
   * @brief Normalize vector function
   *
   * @return Vec3& reference to instance
   */
  Vec3 &normalize();

  /**
   * @brief Overloaded operator [] (non-const version)
   * To get access to coordinates
   * @param i index of coordinate (0 - x, 1 - y, 2 - z)
   * @return T& reference to coordinate value
   *
   * @note Coordinates calculated by mod 3
   */
  T &operator[](size_t i);

  /**
   * @brief Overloaded operator [] (const version)
   * To get access to coordinates
   * @param i index of coordinate (0 - x, 1 - y, 2 - z)
   * @return T coordinate value
   *
   * @note Coordinates calculated by mod 3
   */
  T operator[](size_t i) const;

  /**
   * @brief Check if vector is parallel to another
   *
   * @param[in] rhs vector to check parallelism with
   * @return true if vector is parallel
   * @return false otherwise
```

```
   */
  bool isPar(const Vec3 &rhs) const;

  /**
   * @brief Check if vector is perpendicular to another
   *
   * @param[in] rhs vector to check perpendicularity with
   * @return true if vector is perpendicular
   * @return false otherwise
   */
  bool isPerp(const Vec3 &rhs) const;

  /**
   * @brief Check if vector is equal to another
   *
   * @param[in] rhs vector to check equality with
   * @return true if vector is equal
   * @return false otherwise
   *
   * @note Equality check performs using isNumEq(T lhs, T rhs) function
   */
  bool isEqual(const Vec3 &rhs) const;

  /**
   * @brief Check equality (with threshold) of two floating point numbers function
   *
   * @param[in] lhs first number
   * @param[in] rhs second number
   * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
   * @return false otherwise
   *
   * @note Threshold defined by threshold_ static member
   */
  static bool isNumEq(T lhs, T rhs);

  /**
   * @brief Set new threshold value
   *
   * @param[in] thres value to set
   */
  static void setThreshold(T thres);

  /**
   * @brief Get current threshold value
   */
  static T getThreshold();

  /**
   * @brief Set threshold to default value
   * @note default value equals float point epsilon
   */
  static void setDefThreshold();
};

/**
 * @brief Overloaded + operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return Vec3<T> sum of two vectors
 */
template <std::floating_point T>
Vec3<T> operator+(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  Vec3<T> res{lhs};
  res += rhs;
  return res;
}

/**
 * @brief Overloaded - operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return Vec3<T> res of two vectors
 */
template <std::floating_point T>
Vec3<T> operator-(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  Vec3<T> res{lhs};
  res -= rhs;
  return res;
}

/**
```

```
 * @brief Overloaded multiple by value operator
 *
 * @tparam nT type of value to multiply by
 * @tparam T vector template parameter
 * @param[in] val value to multiply by
 * @param[in] rhs vector to multiply by value
 * @return Vec3<T> result vector
 */
template <Number nT, std::floating_point T>
Vec3<T> operator*(const nT &val, const Vec3<T> &rhs)
{
  Vec3<T> res{rhs};
  res *= val;
  return res;
}

/**
 * @brief Overloaded multiple by value operator
 *
 * @tparam nT type of value to multiply by
 * @tparam T vector template parameter
 * @param[in] val value to multiply by
 * @param[in] lhs vector to multiply by value
 * @return Vec3<T> result vector
 */
template <Number nT, std::floating_point T>
Vec3<T> operator*(const Vec3<T> &lhs, const nT &val)
{
  Vec3<T> res{lhs};
  res *= val;
  return res;
}

/**
 * @brief Overloaded divide by value operator
 *
 * @tparam nT type of value to divide by
 * @tparam T vector template parameter
 * @param[in] val value to divide by
 * @param[in] lhs vector to divide by value
 * @return Vec3<T> result vector
 */
template <Number nT, std::floating_point T>
Vec3<T> operator/(const Vec3<T> &lhs, const nT &val)
{
  Vec3<T> res{lhs};
  res /= val;
  return res;
}

/**
 * @brief Dot product function
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return T dot production
 */
template <std::floating_point T>
T dot(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  return lhs.dot(rhs);
}

/**
 * @brief Cross product function
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return T cross production
 */
template <std::floating_point T>
Vec3<T> cross(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  return lhs.cross(rhs);
}

/**
 * @brief Vec3 equality operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return true if vectors are equal
 * @return false otherwise
 */
```

```
template <std::floating_point T>
bool operator==(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  return lhs.isEqual(rhs);
}

/**
 * @brief Vec3 inequality operator
 *
 * @tparam T vector template parameter
 * @param[in] lhs first vector
 * @param[in] rhs second vector
 * @return true if vectors are not equal
 * @return false otherwise
 */
template <std::floating_point T>
bool operator!=(const Vec3<T> &lhs, const Vec3<T> &rhs)
{
  return !(lhs == rhs);
}

/**
 * @brief Vec3 print operator
 *
 * @tparam T vector template parameter
 * @param[in, out] ost output stream
 * @param[in] vec vector to print
 * @return std::ostream& modified stream instance
 */
template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const Vec3<T> &vec)
{
  ost « "(" « vec.x « ", " « vec.y « ", " « vec.z « ")";
  return ost;
}

/**
 * @brief Vec3 scan operator
 *
 * @tparam T vector template parameter
 * @param[in, out] ist input stram
 * @param[in, out] vec vector to scan
 * @return std::istream& modified stream instance
 */
template <std::floating_point T>
std::istream &operator»(std::istream &ist, Vec3<T> &vec)
{
  ist » vec.x » vec.y » vec.z;
  return ist;
}

using Vec3D = Vec3<double>;
using Vec3F = Vec3<float>;

template <std::floating_point T>
Vec3<T> &Vec3<T>::operator+=(const Vec3 &vec)
{
  x += vec.x;
  y += vec.y;
  z += vec.z;

  return *this;
}

template <std::floating_point T>
Vec3<T> &Vec3<T>::operator-=(const Vec3 &vec)
{
  x -= vec.x;
  y -= vec.y;
  z -= vec.z;

  return *this;
}

template <std::floating_point T>
Vec3<T> Vec3<T>::operator-() const
{
  return Vec3{-x, -y, -z};
}

template <std::floating_point T>
template <Number nType>
Vec3<T> &Vec3<T>::operator*=(nType val)
{
  x *= val;
  y *= val;
  z *= val;

  return *this;
}

template <std::floating_point T>
template <Number nType>
Vec3<T> &Vec3<T>::operator/=(nType val)
{
  x /= static_cast<T>(val);
  y /= static_cast<T>(val);
  z /= static_cast<T>(val);

  return *this;
}

template <std::floating_point T>
T Vec3<T>::dot(const Vec3 &rhs) const
{
  return x * rhs.x + y * rhs.y + z * rhs.z;
}

template <std::floating_point T>
Vec3<T> Vec3<T>::cross(const Vec3 &rhs) const
{
  return Vec3{y * rhs.z - z * rhs.y, z * rhs.x - x * rhs.z, x * rhs.y - y * rhs.x};
}

template <std::floating_point T>
T Vec3<T>::length2() const
{
  return dot(*this);
}

template <std::floating_point T>
T Vec3<T>::length() const
{
  return std::sqrt(length2());
}

template <std::floating_point T>
Vec3<T> Vec3<T>::normalized() const
{
  Vec3 res{*this};
  res.normalize();
  return res;
}

template <std::floating_point T>
Vec3<T> &Vec3<T>::normalize()
{
  T len2 = length2();
  if (isNumEq(len2, 0) || isNumEq(len2, 1))
    return *this;
  return *this /= std::sqrt(len2);
}

template <std::floating_point T>
T &Vec3<T>::operator[](size_t i)
{
  switch (i % 3)
  {
  case 0:
    return x;
  case 1:
    return y;
  case 2:
    return z;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

template <std::floating_point T>
T Vec3<T>::operator[](size_t i) const
{
  switch (i % 3)
  {
  case 0:
    return x;
  case 1:
    return y;
  case 2:
    return z;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}
```

```

template <std::floating_point T>
bool Vec3<T>::isPar(const Vec3 &rhs) const
{
  return cross(rhs).isEqual(Vec3<T>{0});
}

template <std::floating_point T>
bool Vec3<T>::isPerp(const Vec3 &rhs) const
{
  return isNumEq(dot(rhs), 0);
}

template <std::floating_point T>
bool Vec3<T>::isEqual(const Vec3 &rhs) const
{
  return isNumEq(x, rhs.x) && isNumEq(y, rhs.y) && isNumEq(z, rhs.z);
}

template <std::floating_point T>
bool Vec3<T>::isNumEq(T lhs, T rhs)
{
  return std::abs(rhs - lhs) < threshold_;
}

template <std::floating_point T>
void Vec3<T>::setThreshold(T thres)
{
  threshold_ = thres;
}

template <std::floating_point T>
T Vec3<T>::getThreshold()
{
  return threshold_;
}

template <std::floating_point T>
void Vec3<T>::setDefThreshold()
{
  threshold_ = std::numeric_limits<T>::epsilon();
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```