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
- kdtree

Classes

- struct BoundBox
- class Line

  Line class implementation.
- class Plane

  Plane class realization.
- class ThresComp
- 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 >

Enumerations

- enum Axis : std::int8_t { Axis::X = 0, Axis::Y = 1, Axis::Z = 2, Axis::NONE }

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::variant< std::monostate, Vec3< T >, Line< T > > intersect (const Line< T > &l1, const Line< T > &l2)

  Intersect 2 lines and return result of intersection.
- template<std::floating_point T>
  bool operator== (const BoundBox< T > &lhs, const BoundBox< T > &rhs)
- template<std::floating_point T>
  std::ostream & operator<< (std::ostream &ost, const BoundBox< T > &bb)
- 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>
  T triple (const Vec3< T > &v1, const Vec3< T > &v2, const Vec3< T > &v3)

  Triple 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 338 of file vec2.hh.

4.1.2.2 Vec2F

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

Definition at line 339 of file vec2.hh.

4.1.2.3 Vec3D

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

Definition at line 384 of file vec3.hh.

4.1.2.4 Vec3F

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

Definition at line 385 of file vec3.hh.

4.1.3 Enumeration Type Documentation

4.1.3.1 Axis

```
enum geom::Axis :  std::int8_t  [strong]
```

Enumerator

X	
Y	
Z	
NONE	

Definition at line 19 of file common.hh.

4.1.4 Function Documentation

4.1.4.1 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::getSegment(), geom::detail::getTrian2(), geom::detail::helperMollerHaines(), and geom::detail::isIntersectValidInvalid().

4.1.4.2 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 156 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().

4.1.4.3 intersect() [1/2]

```
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

in	*pl1*	first plane
in	*pl2*	second plane

Returns

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

Definition at line 188 of file intersection.hh.

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

Referenced by geom::detail::isIntersectMollerHaines(), and geom::detail::isIntersectSegmentSegment().

4.1.4.4 intersect() [2/2]

```
template<std::floating_point T>
std::variant< std::monostate, Vec3< T >, Line< T > > geom::intersect (
            const Line< T > & l1,
            const Line< T > & l2 )
```

Intersect 2 lines and return result of intersection.

Common intersection case (parallel & skew lines cases are trivial): Let $\overrightarrow{P}$ - point in space, intersection point of two lines.

l_1 equation: $\overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 = \overrightarrow{P}$

l_2 equation: $\overrightarrow{org}_2 + \overrightarrow{dir}_2 \cdot t_2 = \overrightarrow{P}$

Let's equate left sides:

$$\overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 = \overrightarrow{org}_2 + \overrightarrow{dir}_2 \cdot t_2$$

Cross multiply both sides from right by $\overrightarrow{dir}_2$:

$$t_1 \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right) = (\overrightarrow{org}_2 - \overrightarrow{org}_1) \times \overrightarrow{dir}_2$$

Dot multiply both sides by $\frac{\overrightarrow{dir}_1 \times \overrightarrow{dir}_2}{\left|\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right|^2}$:

$$t_1 = \frac{\left((\overrightarrow{org}_2 - \overrightarrow{org}_1) \times \overrightarrow{dir}_2\right) \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right)}{\left|\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right|^2}$$

Thus we get intersection point parameter t_1 on l_1, let's substitute it to l_1 equation:

$$\overrightarrow{P} = \overrightarrow{org}_1 + \frac{\left((\overrightarrow{org}_2 - \overrightarrow{org}_1) \times \overrightarrow{dir}_2\right) \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right)}{\left|\overrightarrow{dir}_1 \times \overrightarrow{dir}_2\right|^2} \cdot \overrightarrow{dir}_1$$

Template Parameters

T	- floating point type of coordinates

Parameters

in	*l1*	first line
in	*l2*	second line

Returns

std::variant<std::monostate, Vec3<T>, Line<T>>

Definition at line 215 of file intersection.hh.

References cross(), geom::Line< T >::dir(), dot(), geom::Line< T >::getPoint(), geom::Line< T >::isEqual(), geom::Line< T >::isPar(), geom::Line< T >::isSkew(), and geom::Line< T >::org().

4.1.4.5 operator==() [1/5]

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

Definition at line 110 of file boundbox.hh.

References geom::BoundBox< T >::maxX, geom::BoundBox< T >::maxY, geom::BoundBox< T >::minX, geom::BoundBox< T >::minY, and geom::BoundBox< T >::minZ.

Referenced by geom::kdtree::Container< T >::ConstIterator::operator!=(), and geom::kdtree::KdTree< T >::ConstIterator::operator!=

4.1.4.6 operator<<() [1/6]

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

Definition at line 118 of file boundbox.hh.

References geom::BoundBox< T >::maxX, geom::BoundBox< T >::maxY, geom::BoundBox< T >::maxZ, geom::BoundBox< T >::minX, geom::BoundBox< T >::minY, and geom::BoundBox< T >::minZ.

4.1.4.7 operator<<() [2/6]

```
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 131 of file line.hh.

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

4.1.4.8 operator==() [2/5]

```
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 147 of file line.hh.

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

4.1.4.9 operator==() [3/5]

```
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 150 of file plane.hh.

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

4.1.4.10 operator<<() [3/6]

```
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 164 of file plane.hh.

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

4.1.4.11 operator<<() [4/6]

```
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 141 of file triangle.hh.

4.1.4.12 operator>>() [1/2]

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

Definition at line 153 of file triangle.hh.

4.1.4.13 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 205 of file vec2.hh.

4.1.4.14 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 221 of file vec2.hh.

4.1.4.15 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 238 of file vec2.hh.

4.1.4.16 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 255 of file vec2.hh.

4.1.4.17 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 272 of file vec2.hh.

4.1.4.18 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 288 of file vec2.hh.

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

Referenced by distance(), intersect(), geom::detail::isIntersectPointSegment(), geom::detail::isIntersectPointTriangle(), geom::detail::isIntersectSegmentSegment(), geom::Vec2< T >::isPerp(), geom::Vec3< T >::isPerp(), geom::Vec2< T >::length2(), geom::Vec3< T >::length2(), and triple().

4.1.4.19 operator==() [4/5]

```
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 303 of file vec2.hh.

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

4.1.4.20 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 318 of file vec2.hh.

4.1.4.21 operator<<() [5/6]

```
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 332 of file vec2.hh.

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

4.1.4.22 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 207 of file vec3.hh.

4.1.4.23 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 223 of file vec3.hh.

4.1.4.24 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 240 of file vec3.hh.

4.1.4.25 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 257 of file vec3.hh.

4.1.4.26 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 274 of file vec3.hh.

4.1.4.27 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 290 of file vec3.hh.

References geom::Vec3< T >::dot().

4.1.4.28 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 304 of file vec3.hh.

References geom::Vec3< T >::cross().

Referenced by intersect(), geom::Vec3< T >::isPar(), geom::Triangle< T >::isValid(), and triple().

4.1.4.29 triple()

```
template<std::floating_point T>
T geom::triple (
            const Vec3< T > & v1,
            const Vec3< T > & v2,
            const Vec3< T > & v3 )
```

Triple product function.

Template Parameters

T	vector template parameter

Parameters

in	*v1*	first vector
in	*v2*	second vector
in	*v3*	third vector

Returns

T triple production

Definition at line 319 of file vec3.hh.

References cross(), and dot().

Referenced by geom::Line< T >::isSkew().

4.1.4.30 operator==() [5/5]

```
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 334 of file vec3.hh.

References geom::Vec3< T >::isEqual().

4.1.4.31 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 349 of file vec3.hh.

4.1.4.32 operator<<() [6/6]

```
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 363 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

4.1.4.33 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 378 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

4.1.5 Variable Documentation

4.1.5.1 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 17 of file common.hh.

4.2 geom::detail Namespace Reference

Typedefs

- template<typename T >
  using Segment2D = std::pair< T, T >
- template<std::floating_point T>
  using Trian2 = std::array< Vec2< T >, 3 >
- template<std::floating_point T>
  using Segment3D = std::pair< Vec3< T >, Vec3< T > >

Functions

- template<std::floating_point T>
  bool isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  Segment2D< T > helperMollerHaines (const Triangle< T > &tr, const Plane< T > &pl, const Line< T > &l)
- template<std::floating_point T>
  bool isIntersectBothInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool isIntersectValidInvalid (const Triangle< T > &valid, const Triangle< T > &invalid)
- template<std::floating_point T>
  bool isIntersectPointTriangle (const Vec3< T > &pt, const Triangle< T > &tr)
- template<std::floating_point T>
  bool isIntersectPointSegment (const Vec3< T > &pt, const Segment3D< T > &segm)
- template<std::floating_point T>
  bool isIntersectSegmentSegment (const Segment3D< T > &segm1, const Segment3D< T > &segm2)
- template<std::floating_point T>
  bool isPoint (const Triangle< T > &tr)
- template<std::floating_point T>
  bool isOverlap (Segment2D< T > &segm1, Segment2D< T > &segm2)
- template<std::forward_iterator It>
  bool isAllPosNeg (It begin, It end)
- template<std::floating_point T>
  bool isAllPosNeg (T num1, T num2)
- template<std::floating_point T>
  bool isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  Trian2< T > getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  bool isCounterClockwise (Trian2< T > &tr)
- template<std::floating_point T>
  Segment2D< T > computeInterval (const Trian2< T > &tr, const Vec2< T > &d)
- template<std::floating_point T>
  Segment3D< T > getSegment (const Triangle< T > &tr)

4.2.1 Typedef Documentation

4.2.1.1 Segment2D

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

Definition at line 15 of file detail.hh.

4.2.1.2 Trian2

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

Definition at line 18 of file detail.hh.

4.2.1.3 Segment3D

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

Definition at line 21 of file detail.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 77 of file detail.hh.

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

Referenced by geom::isIntersect(), and isIntersectValidInvalid().

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 100 of file detail.hh.

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

Referenced by geom::isIntersect().

4.2.2.3 helperMollerHaines()

```
template<std::floating_point T>
Segment2D< T > geom::detail::helperMollerHaines (
            const Triangle< T > & tr,
            const Plane< T > & pl,
            const Line< T > & l )
```

Definition at line 114 of file detail.hh.

References geom::Triangle< T >::begin(), geom::Line< T >::dir(), geom::distance(), geom::Triangle< T >::end(), isAllPosNeg(), 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 154 of file detail.hh.

References getSegment(), isIntersectPointSegment(), isIntersectSegmentSegment(), and isPoint().

Referenced by geom::isIntersect().

4.2.2.5 isIntersectValidInvalid()

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

Definition at line 172 of file detail.hh.

References geom::distance(), geom::Triangle< T >::getPlane(), getSegment(), isIntersect2D(), isIntersectPointTriangle(), and isPoint().

Referenced by geom::isIntersect().

4.2.2.6 isIntersectPointTriangle()

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

Definition at line 197 of file detail.hh.

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

Referenced by isIntersectValidInvalid().

4.2.2.7 isIntersectPointSegment()

```
template<std::floating_point T>
bool geom::detail::isIntersectPointSegment (
            const Vec3< T > & pt,
            const Segment3D< T > & segm )
```

Definition at line 225 of file detail.hh.

References geom::dot(), and isAllPosNeg().

Referenced by isIntersectBothInvalid(), and isIntersectSegmentSegment().

4.2.2.8 isIntersectSegmentSegment()

```
template<std::floating_point T>
bool geom::detail::isIntersectSegmentSegment (
            const Segment3D< T > & segm1,
            const Segment3D< T > & segm2 )
```

Definition at line 238 of file detail.hh.

References geom::dot(), geom::intersect(), isIntersectPointSegment(), and isOverlap().

Referenced by isIntersectBothInvalid().

4.2.2.9 isPoint()

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

Definition at line 262 of file detail.hh.

Referenced by isIntersectBothInvalid(), and isIntersectValidInvalid().

4.2.2.10 isOverlap()

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

Definition at line 268 of file detail.hh.

Referenced by isIntersectMollerHaines(), and isIntersectSegmentSegment().

4.2.2.11 isAllPosNeg() [1/2]

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

Definition at line 274 of file detail.hh.

Referenced by helperMollerHaines(), isIntersectPointSegment(), and isOnOneSide().

4.2.2.12 isAllPosNeg() [2/2]

```
template<std::floating_point T>
bool geom::detail::isAllPosNeg (
            T num1,
            T num2 )
```

Definition at line 286 of file detail.hh.

4.2.2.13 isOnOneSide()

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

Definition at line 293 of file detail.hh.

References geom::Triangle< T >::begin(), geom::Triangle< T >::end(), and isAllPosNeg().

Referenced by geom::isIntersect().

4.2.2.14 getTrian2()

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

Definition at line 301 of file detail.hh.

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

Referenced by isIntersect2D().

4.2.2.15 isCounterClockwise()

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

Definition at line 335 of file detail.hh.

Referenced by getTrian2().

4.2.2.16 computeInterval()

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

Definition at line 355 of file detail.hh.

Referenced by isIntersect2D().

4.2.2.17 getSegment()

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

Definition at line 364 of file detail.hh.

References geom::distance().

Referenced by isIntersectBothInvalid(), and isIntersectValidInvalid().

4.3 geom::kdtree Namespace Reference

Classes

- class Container
- class KdTree
- struct Node

Typedefs

- using Index = std::size_t

4.3.1 Typedef Documentation

4.3.1.1 Index

```
using geom::kdtree::Index = typedef std::size_t
```

Definition at line 13 of file node.hh.

Chapter 5

Class Documentation

5.1 geom::BoundBox< T > Struct Template Reference

`#include <boundbox.hh>`

Public Member Functions

- bool belongsTo (const BoundBox< T > &bb)
- T & min (Axis axis) &
- T & max (Axis axis) &
- T min (Axis axis) &&
- T max (Axis axis) &&
- T min (Axis axis) const &
- T max (Axis axis) const &
- Axis getMaxDim () const

Public Attributes

- T minX {}
- T maxX {}
- T minY {}
- T maxY {}
- T minZ {}
- T maxZ {}

5.1.1 Detailed Description

template<std::floating_point T>
struct geom::BoundBox< T >

Definition at line 14 of file boundbox.hh.

5.1.2 Member Function Documentation

5.1.2.1 belongsTo()

```
template<std::floating_point T>
bool geom::BoundBox< T >::belongsTo (
            const BoundBox< T > & bb )
```

Definition at line 40 of file boundbox.hh.

References geom::BoundBox< T >::maxX, geom::BoundBox< T >::maxY, geom::BoundBox< T >::maxZ, geom::BoundBox< T >::minX, geom::BoundBox< T >::minY, and geom::BoundBox< T >::minZ.

5.1.2.2 min() [1/3]

```
template<std::floating_point T>
T & geom::BoundBox< T >::min (
            Axis axis ) &
```

Definition at line 64 of file boundbox.hh.

References BBFILL.

5.1.2.3 max() [1/3]

```
template<std::floating_point T>
T & geom::BoundBox< T >::max (
            Axis axis ) &
```

Definition at line 70 of file boundbox.hh.

References BBFILL.

5.1.2.4 min() [2/3]

```
template<std::floating_point T>
T geom::BoundBox< T >::min (
            Axis axis ) &&
```

Definition at line 76 of file boundbox.hh.

References BBFILL.

5.1.2.5 max() [2/3]

```
template<std::floating_point T>
T geom::BoundBox< T >::max (
            Axis axis ) &&
```

Definition at line 82 of file boundbox.hh.

References BBFILL.

5.1.2.6 min() [3/3]

```
template<std::floating_point T>
T geom::BoundBox< T >::min (
            Axis axis ) const &
```

Definition at line 88 of file boundbox.hh.

References BBFILL.

5.1.2.7 max() [3/3]

```
template<std::floating_point T>
T geom::BoundBox< T >::max (
            Axis axis ) const &
```

Definition at line 94 of file boundbox.hh.

References BBFILL.

5.1.2.8 getMaxDim()

```
template<std::floating_point T>
Axis geom::BoundBox< T >::getMaxDim
```

Definition at line 102 of file boundbox.hh.

References geom::X, geom::Y, and geom::Z.

5.1.3 Member Data Documentation

5.1.3.1 minX

```
template<std::floating_point T>
T geom::BoundBox< T >::minX {}
```

Definition at line 16 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), geom::operator<<(), and geom::operator==().

5.1.3.2 maxX

```
template<std::floating_point T>
T geom::BoundBox< T >::maxX {}
```

Definition at line 17 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), geom::operator<<(), and geom::operator==().

5.1.3.3 minY

```
template<std::floating_point T>
T geom::BoundBox< T >::minY {}
```

Definition at line 19 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), geom::operator<<(), and geom::operator==().

5.1.3.4 maxY

```
template<std::floating_point T>
T geom::BoundBox< T >::maxY {}
```

Definition at line 20 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), geom::operator<<(), and geom::operator==().

5.1.3.5 minZ

```
template<std::floating_point T>
T geom::BoundBox< T >::minZ {}
```

Definition at line 22 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), geom::operator<<(), and geom::operator==().

5.1.3.6 maxZ

```
template<std::floating_point T>
T geom::BoundBox< T >::maxZ {}
```

Definition at line 23 of file boundbox.hh.

Referenced by geom::BoundBox< T >::belongsTo(), and geom::operator<<().

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

- include/primitives/boundbox.hh

5.2 geom::kdtree::Container< T > Class Template Reference

```
#include <container.hh>
```

Classes

- class ConstIterator

Public Member Functions

- Container (const KdTree< T > *tree, const Node< T > *node)
- ConstIterator cbegin () const &
- ConstIterator cend () const &
- ConstIterator begin () const &
- ConstIterator end () const &
- Node< T >::IndexConstIterator indexBegin () const &
- Node< T >::IndexConstIterator indexEnd () const &
- T separator () const
- Axis sepAxis () const
- BoundBox< T > boundBox () const
- const Triangle< T > & triangleByIndex (Index index) const &
- Container left () const
- Container right () const
- bool isValid () const

5.2.1 Detailed Description

template<std::floating_point T>
class geom::kdtree::Container< T >

Definition at line 16 of file container.hh.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Container()

```
template<std::floating_point T>
geom::kdtree::Container< T >::Container (
            const KdTree< T > * tree,
            const Node< T > * node )
```

Definition at line 79 of file container.hh.

5.2.3 Member Function Documentation

5.2.3.1 cbegin()

```
template<std::floating_point T>
Container< T >::ConstIterator geom::kdtree::Container< T >::cbegin
```

Definition at line 83 of file container.hh.

5.2.3.2 cend()

```
template<std::floating_point T>
Container< T >::ConstIterator geom::kdtree::Container< T >::cend
```

Definition at line 89 of file container.hh.

5.2.3.3 begin()

```
template<std::floating_point T>
Container< T >::ConstIterator geom::kdtree::Container< T >::begin
```

Definition at line 95 of file container.hh.

5.2.3.4 end()

```
template<std::floating_point T>
Container< T >::ConstIterator geom::kdtree::Container< T >::end
```

Definition at line 101 of file container.hh.

5.2.3.5 indexBegin()

```
template<std::floating_point T>
Node< T >::IndexConstIterator geom::kdtree::Container< T >::indexBegin
```

Definition at line 107 of file container.hh.

Referenced by geom::kdtree::Container< T >::ConstIterator::ConstIterator().

5.2.3.6 indexEnd()

```
template<std::floating_point T>
Node< T >::IndexConstIterator geom::kdtree::Container< T >::indexEnd
```

Definition at line 113 of file container.hh.

Referenced by geom::kdtree::Container< T >::ConstIterator::ConstIterator().

5.2.3.7 separator()

```
template<std::floating_point T>
T geom::kdtree::Container< T >::separator
```

Definition at line 119 of file container.hh.

5.2.3.8 sepAxis()

```
template<std::floating_point T>
Axis geom::kdtree::Container< T >::sepAxis
```

Definition at line 125 of file container.hh.

5.2.3.9 boundBox()

```
template<std::floating_point T>
BoundBox< T > geom::kdtree::Container< T >::boundBox
```

Definition at line 131 of file container.hh.

5.2.3.10 triangleByIndex()

```
template<std::floating_point T>
const Triangle< T > & geom::kdtree::Container< T >::triangleByIndex (
            Index index ) const &
```

Definition at line 137 of file container.hh.

5.2.3.11 left()

```
template<std::floating_point T>
Container< T > geom::kdtree::Container< T >::left
```

Definition at line 143 of file container.hh.

References geom::kdtree::Container< T >::left().

Referenced by geom::kdtree::Container< T >::left().

5.2.3.12 right()

```
template<std::floating_point T>
Container< T > geom::kdtree::Container< T >::right
```

Definition at line 149 of file container.hh.

References geom::kdtree::Container< T >::right().

Referenced by geom::kdtree::Container< T >::right().

5.2.3.13 isValid()

```
template<std::floating_point T>
bool geom::kdtree::Container< T >::isValid
```

Definition at line 155 of file container.hh.

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

- include/kdtree/container.hh

5.3 geom::kdtree::Container< T >::ConstIterator Class Reference

```
#include <container.hh>
```

Public Types

- using iterator_category = std::forward_iterator_tag
- using difference_type = std::size_t
- using value_type = Triangle< T >
- using reference = const Triangle< T > &
- using pointer = const Triangle< T > *

Public Member Functions

- ConstIterator (const Container *cont, bool isEnd=false)
- Index getIndex ()
- ConstIterator & operator++ ()
- ConstIterator operator++ (int)
- reference operator* () const
- pointer operator-> () const
- bool operator== (const ConstIterator &lhs) const
- bool operator!= (const ConstIterator &lhs) const

5.3.1 Detailed Description

template<std::floating_point T>
class geom::kdtree::Container< T >::ConstIterator

Definition at line 45 of file container.hh.

5.3.2 Member Typedef Documentation

5.3.2.1 iterator_category

```
template<std::floating_point T>
using geom::kdtree::Container< T >::ConstIterator::iterator_category = std::forward_iterator↩
_tag
```

Definition at line 48 of file container.hh.

5.3.2.2 difference_type

```
template<std::floating_point T>
using geom::kdtree::Container< T >::ConstIterator::difference_type = std::size_t
```

Definition at line 49 of file container.hh.

5.3.2.3 value_type

```
template<std::floating_point T>
using geom::kdtree::Container< T >::ConstIterator::value_type = Triangle<T>
```

Definition at line 50 of file container.hh.

5.3.2.4 reference

```
template<std::floating_point T>
using geom::kdtree::Container< T >::ConstIterator::reference = const Triangle<T> &
```

Definition at line 51 of file container.hh.

5.3.2.5 pointer

```
template<std::floating_point T>
using geom::kdtree::Container< T >::ConstIterator::pointer = const Triangle<T> *
```

Definition at line 52 of file container.hh.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 ConstIterator()

```
template<std::floating_point T>
geom::kdtree::Container< T >::ConstIterator::ConstIterator (
            const Container * cont,
            bool isEnd = false )
```

Definition at line 165 of file container.hh.

References geom::kdtree::Container< T >::indexBegin(), and geom::kdtree::Container< T >::indexEnd().

5.3.4 Member Function Documentation

5.3.4.1 getIndex()

```
template<std::floating_point T>
Index geom::kdtree::Container< T >::ConstIterator::getIndex
```

Definition at line 177 of file container.hh.

5.3.4.2 operator++() [1/2]

```
template<std::floating_point T>
Container< T >::ConstIterator & geom::kdtree::Container< T >::ConstIterator::operator++
```

Definition at line 183 of file container.hh.

5.3.4.3 operator++() [2/2]

```
template<std::floating_point T>
Container< T >::ConstIterator geom::kdtree::Container< T >::ConstIterator::operator++ (
            int  )
```

Definition at line 190 of file container.hh.

5.3.4.4 operator*()

```
template<std::floating_point T>
Container< T >::ConstIterator::reference geom::kdtree::Container< T >::ConstIterator::operator*
```

Definition at line 198 of file container.hh.

5.3.4.5 operator->()

```
template<std::floating_point T>
Container< T >::ConstIterator::pointer geom::kdtree::Container< T >::ConstIterator::operator->
```

Definition at line 204 of file container.hh.

5.3.4.6 operator==()

```
template<std::floating_point T>
bool geom::kdtree::Container< T >::ConstIterator::operator== (
            const ConstIterator & lhs ) const
```

Definition at line 210 of file container.hh.

5.3.4.7 operator"!=()

```
template<std::floating_point T>
bool geom::kdtree::Container< T >::ConstIterator::operator!= (
            const ConstIterator & lhs ) const
```

Definition at line 216 of file container.hh.

References geom::operator==().

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

- include/kdtree/container.hh

5.4 geom::kdtree::KdTree< T > Class Template Reference

```
#include <container.hh>
```

Classes

- class ConstIterator
- struct ContainerPtr

Public Member Functions

- KdTree (std::initializer_list< Triangle< T >> il)
- KdTree (const KdTree &tree)
- KdTree (KdTree &&tree)=default
- KdTree ()=default
- ~KdTree ()
- KdTree & operator= (const KdTree &tree)
- KdTree & operator= (KdTree &&tree)=default
- ConstIterator cbegin () const &
- ConstIterator cend () const &
- ConstIterator begin () const &
- ConstIterator end () const &
- ConstIterator beginFrom (const ConstIterator &iter) const &
- void insert (const Triangle< T > &tr)
- void clear ()
- void setNodeCapacity (std::size_t newCap)
- bool empty () const
- std::size_t size () const
- std::size_t nodeCapacity () const
- const Triangle< T > & triangleByIndex (Index index) const &
- void dumpRecursive (std::ostream &ost=std::cout) const

Static Public Member Functions

- static bool isOnPosSide (Axis axis, T separator, const Triangle< T > &tr)
- static bool isOnNegSide (Axis axis, T separator, const Triangle< T > &tr)
- static bool isOnSide (Axis axis, T separator, const Triangle< T > &tr, std::function< bool(T, T)> comparator)

5.4.1 Detailed Description

template<std::floating_point T>
class geom::kdtree::KdTree< T >

Definition at line 13 of file container.hh.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 KdTree() [1/4]

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::KdTree (
            std::initializer_list< Triangle< T >> il )
```

Definition at line 119 of file kdtree.hh.

5.4.2.2 KdTree() [2/4]

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::KdTree (
            const KdTree< T > & tree )
```

Definition at line 126 of file kdtree.hh.

5.4.2.3 KdTree() [3/4]

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::KdTree (
            KdTree< T > && tree )  [default]
```

5.4.2.4 KdTree() [4/4]

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::KdTree ( )  [default]
```

5.4.2.5 ~KdTree()

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::~KdTree
```

Definition at line 134 of file kdtree.hh.

5.4.3 Member Function Documentation

5.4.3.1 operator=() [1/2]

```
template<std::floating_point T>
KdTree< T > & geom::kdtree::KdTree< T >::operator= (
            const KdTree< T > & tree )
```

Definition at line 140 of file kdtree.hh.

5.4.3.2 operator=() [2/2]

```
template<std::floating_point T>
KdTree& geom::kdtree::KdTree< T >::operator= (
            KdTree< T > && tree )  [default]
```

5.4.3.3 cbegin()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::cbegin
```

Definition at line 149 of file kdtree.hh.

5.4.3.4 cend()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::cend
```

Definition at line 155 of file kdtree.hh.

5.4.3.5 begin()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::begin
```

Definition at line 161 of file kdtree.hh.

5.4.3.6 end()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::end
```

Definition at line 167 of file kdtree.hh.

5.4.3.7 beginFrom()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::beginFrom (
            const ConstIterator & iter ) const &
```

Definition at line 173 of file kdtree.hh.

References geom::kdtree::KdTree< T >::ConstIterator::beginFrom().

5.4.3.8 insert()

```
template<std::floating_point T>
void geom::kdtree::KdTree< T >::insert (
            const Triangle< T > & tr )
```

Definition at line 181 of file kdtree.hh.

References geom::Triangle< T >::belongsTo(), geom::Triangle< T >::boundBox(), and geom::NONE.

5.4.3.9 clear()

```
template<std::floating_point T>
void geom::kdtree::KdTree< T >::clear
```

Definition at line 201 of file kdtree.hh.

5.4.3.10 setNodeCapacity()

```
template<std::floating_point T>
void geom::kdtree::KdTree< T >::setNodeCapacity (
            std::size_t newCap )
```

Definition at line 228 of file kdtree.hh.

5.4.3.11 empty()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::empty
```

Definition at line 235 of file kdtree.hh.

5.4.3.12 size()

```
template<std::floating_point T>
std::size_t geom::kdtree::KdTree< T >::size
```

Definition at line 241 of file kdtree.hh.

5.4.3.13 nodeCapacity()

```
template<std::floating_point T>
std::size_t geom::kdtree::KdTree< T >::nodeCapacity
```

Definition at line 247 of file kdtree.hh.

5.4.3.14 triangleByIndex()

```
template<std::floating_point T>
const Triangle< T > & geom::kdtree::KdTree< T >::triangleByIndex (
            Index index ) const &
```

Definition at line 253 of file kdtree.hh.

5.4.3.15 dumpRecursive()

```
template<std::floating_point T>
void geom::kdtree::KdTree< T >::dumpRecursive (
            std::ostream & ost = std::cout ) const
```

Definition at line 259 of file kdtree.hh.

5.4.3.16 isOnPosSide()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::isOnPosSide (
            Axis axis,
            T separator,
            const Triangle< T > & tr )  [static]
```

Definition at line 268 of file kdtree.hh.

5.4.3.17 isOnNegSide()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::isOnNegSide (
            Axis axis,
            T separator,
            const Triangle< T > & tr )  [static]
```

Definition at line 274 of file kdtree.hh.

5.4.3.18 isOnSide()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::isOnSide (
            Axis axis,
            T separator,
            const Triangle< T > & tr,
            std::function< bool(T, T)> comparator )  [static]
```

Definition at line 280 of file kdtree.hh.

References geom::Triangle< T >::begin(), geom::Triangle< T >::end(), and geom::NONE.

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

- include/kdtree/container.hh
- include/kdtree/kdtree.hh

5.5 geom::kdtree::KdTree< T >::ConstIterator Class Reference

```
#include <kdtree.hh>
```

Public Types

- using iterator_category = std::forward_iterator_tag
- using difference_type = std::size_t
- using value_type = Container< T >
- using reference = Container< T >
- using pointer = ContainerPtr

Public Member Functions

- ConstIterator (const KdTree< T > *tree, const Node< T > *node)
- ConstIterator & operator++ ()
- ConstIterator operator++ (int)
- reference operator* () const
- pointer operator-> () const
- bool operator== (const ConstIterator &lhs) const
- bool operator!= (const ConstIterator &lhs) const

Static Public Member Functions

- static ConstIterator beginFrom (const ConstIterator &iter)

5.5.1 Detailed Description

template<std::floating_point T>
class geom::kdtree::KdTree< T >::ConstIterator

Definition at line 84 of file kdtree.hh.

5.5.2 Member Typedef Documentation

5.5.2.1 iterator_category

```
template<std::floating_point T>
using geom::kdtree::KdTree< T >::ConstIterator::iterator_category = std::forward_iterator_tag
```

Definition at line 87 of file kdtree.hh.

5.5.2.2 difference_type

```
template<std::floating_point T>
using geom::kdtree::KdTree< T >::ConstIterator::difference_type = std::size_t
```

Definition at line 88 of file kdtree.hh.

5.5.2.3 value_type

```
template<std::floating_point T>
using geom::kdtree::KdTree< T >::ConstIterator::value_type = Container<T>
```

Definition at line 89 of file kdtree.hh.

5.5.2.4 reference

```
template<std::floating_point T>
using geom::kdtree::KdTree< T >::ConstIterator::reference = Container<T>
```

Definition at line 90 of file kdtree.hh.

5.5.2.5 pointer

```
template<std::floating_point T>
using geom::kdtree::KdTree< T >::ConstIterator::pointer = ContainerPtr
```

Definition at line 91 of file kdtree.hh.

5.5.3 Constructor & Destructor Documentation

5.5.3.1 ConstIterator()

```
template<std::floating_point T>
geom::kdtree::KdTree< T >::ConstIterator::ConstIterator (
            const KdTree< T > * tree,
            const Node< T > * node )
```

Definition at line 414 of file kdtree.hh.

5.5.4 Member Function Documentation

5.5.4.1 operator++() [1/2]

```
template<std::floating_point T>
KdTree< T >::ConstIterator & geom::kdtree::KdTree< T >::ConstIterator::operator++
```

Definition at line 419 of file kdtree.hh.

References geom::NONE.

5.5.4.2 operator++() [2/2]

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::ConstIterator::operator++ (
            int  )
```

Definition at line 440 of file kdtree.hh.

5.5.4.3 operator*()

```
template<std::floating_point T>
KdTree< T >::ConstIterator::reference geom::kdtree::KdTree< T >::ConstIterator::operator*
```

Definition at line 448 of file kdtree.hh.

5.5.4.4 operator->()

```
template<std::floating_point T>
KdTree< T >::ConstIterator::pointer geom::kdtree::KdTree< T >::ConstIterator::operator->
```

Definition at line 454 of file kdtree.hh.

5.5.4.5 operator==()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::ConstIterator::operator== (
            const ConstIterator & lhs ) const
```

Definition at line 460 of file kdtree.hh.

5.5.4.6 operator"!=()

```
template<std::floating_point T>
bool geom::kdtree::KdTree< T >::ConstIterator::operator!= (
            const ConstIterator & lhs ) const
```

Definition at line 466 of file kdtree.hh.

References geom::operator==().

5.5.4.7 beginFrom()

```
template<std::floating_point T>
KdTree< T >::ConstIterator geom::kdtree::KdTree< T >::ConstIterator::beginFrom (
            const ConstIterator & iter )  [static]
```

Definition at line 472 of file kdtree.hh.

Referenced by geom::kdtree::KdTree< T >::beginFrom().

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

- include/kdtree/kdtree.hh

5.6 geom::kdtree::KdTree< T >::ContainerPtr Struct Reference

```
#include <kdtree.hh>
```

Public Member Functions

- const Container< T > * operator-> () const

Public Attributes

- Container< T > cont

5.6.1 Detailed Description

template<std::floating_point T>
struct geom::kdtree::KdTree< T >::ContainerPtr

Definition at line 78 of file kdtree.hh.

5.6.2 Member Function Documentation

5.6.2.1 operator->()

```
template<std::floating_point T>
const Container< T > * geom::kdtree::KdTree< T >::ContainerPtr::operator->
```

Definition at line 404 of file kdtree.hh.

References geom::kdtree::KdTree< T >::ContainerPtr::cont.

5.6.3 Member Data Documentation

5.6.3.1 cont

```
template<std::floating_point T>
Container<T> geom::kdtree::KdTree< T >::ContainerPtr::cont
```

Definition at line 80 of file kdtree.hh.

Referenced by geom::kdtree::KdTree< T >::ContainerPtr::operator->().

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

- include/kdtree/kdtree.hh

5.7 geom::kdtree::Node< T > Struct Template Reference

`#include <node.hh>`

Public Types

- using IndexIterator = std::vector< Index >::iterator
- using IndexConstIterator = std::vector< Index >::const_iterator

Public Member Functions

- void dumpRecursive (std::ostream &ost) const

Public Attributes

- T separator {}
- Axis sepAxis {Axis::NONE}
- BoundBox< T > boundBox {}
- std::vector< Index > indicies {}
- std::unique_ptr< Node > left {nullptr}
- std::unique_ptr< Node > right {nullptr}

5.7.1 Detailed Description

template<std::floating_point T>
struct geom::kdtree::Node< T >

Definition at line 16 of file node.hh.

5.7.2 Member Typedef Documentation

5.7.2.1 IndexIterator

```
template<std::floating_point T>
using geom::kdtree::Node< T >::IndexIterator = std::vector<Index>::iterator
```

Definition at line 26 of file node.hh.

5.7.2.2 IndexConstIterator

```
template<std::floating_point T>
using geom::kdtree::Node< T >::IndexConstIterator = std::vector<Index>::const_iterator
```

Definition at line 27 of file node.hh.

5.7.3 Member Function Documentation

5.7.3.1 dumpRecursive()

```
template<std::floating_point T>
void geom::kdtree::Node< T >::dumpRecursive (
            std::ostream & ost ) const
```

Definition at line 33 of file node.hh.

5.7.4 Member Data Documentation

5.7.4.1 separator

```
template<std::floating_point T>
T geom::kdtree::Node< T >::separator {}
```

Definition at line 18 of file node.hh.

5.7.4.2 sepAxis

```
template<std::floating_point T>
Axis geom::kdtree::Node< T >::sepAxis {Axis::NONE}
```

Definition at line 19 of file node.hh.

5.7.4.3 boundBox

```
template<std::floating_point T>
BoundBox<T> geom::kdtree::Node< T >::boundBox {}
```

Definition at line 20 of file node.hh.

5.7.4.4 indicies

```
template<std::floating_point T>
std::vector<Index> geom::kdtree::Node< T >::indicies {}
```

Definition at line 21 of file node.hh.

5.7.4.5 left

```
template<std::floating_point T>
std::unique_ptr<Node> geom::kdtree::Node< T >::left {nullptr}
```

Definition at line 23 of file node.hh.

5.7.4.6 right

```
template<std::floating_point T>
std::unique_ptr<Node> geom::kdtree::Node< T >::right {nullptr}
```

Definition at line 24 of file node.hh.

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

- include/kdtree/node.hh

5.8 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.
- Vec3< T > && org () &&
  Getter for origin vector.
- Vec3< T > && dir () &&
  Getter for direction vector.
- template<Number nType>
  Vec3< T > getPoint (nType t) const
  Get point on line by parameter t.
- 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.
- bool isPar (const Line &line) const
  Checks is ∗this parallel to another line.
- bool isSkew (const Line< T > &line) const
  Checks is ∗this is skew with another line.

Static Public Member Functions

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

  Get line by 2 points.

5.8.1 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.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 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 153 of file line.hh.

References geom::Line< T >::org().

5.8.3 Member Function Documentation

5.8.3.1 org() [1/2]

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

Getter for origin vector.

Returns

const Vec3<T>& const reference to origin vector

Definition at line 160 of file line.hh.

Referenced by geom::Plane< T >::belongs(), geom::detail::helperMollerHaines(), geom::intersect(), geom::Line< T >::Line(), and geom::operator<<().

5.8.3.2 dir() [1/2]

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

Getter for direction vector.

Returns

const Vec3<T>& const reference to direction vector

Definition at line 166 of file line.hh.

Referenced by geom::Plane< T >::belongs(), geom::detail::helperMollerHaines(), geom::intersect(), and geom::operator<<().

5.8.3.3 org() [2/2]

```
template<std::floating_point T>
Vec3<T>&& geom::Line< T >::org ( ) &&
```

Getter for origin vector.

Returns

Vec3<T>&& reference to origin vector

5.8.3.4 dir() [2/2]

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

Getter for direction vector.

Returns

Vec3<T>&& reference to direction vector

5.8.3.5 getPoint()

```
template<std::floating_point T>
template<Number nType>
Vec3< T > geom::Line< T >::getPoint (
            nType t ) const
```

Get point on line by parameter t.

Template Parameters

nType	numeric type

Parameters

in	*t*	point paramater from line's equation

Returns

Vec3<T> Point related to parameter

Definition at line 185 of file line.hh.

Referenced by geom::intersect().

5.8.3.6 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 191 of file line.hh.

5.8.3.7 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 197 of file line.hh.

Referenced by geom::intersect(), and geom::operator==().

5.8.3.8 isPar()

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

Checks is *this parallel to another line.

Note

Assumes equal lines as parallel

Parameters

in	*line*	const reference to another line

Returns

true if lines are parallel

false if lines are not parallel

Definition at line 203 of file line.hh.

Referenced by geom::intersect().

5.8.3.9 isSkew()

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

Checks is ∗this is skew with another line.

Parameters

in	*line*	const reference to another line

Returns

true if lines are skew

false if lines are not skew

Definition at line 209 of file line.hh.

References geom::ThresComp< T >::isZero(), and geom::triple().

Referenced by geom::intersect().

5.8.3.10 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 216 of file line.hh.

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

- include/primitives/line.hh

5.9 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.
- Vec3< T > && norm () &&
  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.

5.9.1 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.

5.9.2 Member Function Documentation

5.9.2.1 dist()

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

Getter for distance.

Returns

T value of distance

Definition at line 178 of file plane.hh.

Referenced by geom::distance(), geom::intersect(), and geom::operator<<().

5.9.2.2 norm() [1/2]

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

Getter for normal vector.

Returns

const Vec3<T>& const reference to normal vector

Definition at line 184 of file plane.hh.

Referenced by geom::distance(), geom::detail::getTrian2(), geom::intersect(), and geom::operator<<().

5.9.2.3 norm() [2/2]

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

Getter for normal vector.

Returns

Vec3<T>&& reference to normal vector

5.9.2.4 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 196 of file plane.hh.

5.9.2.5 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 202 of file plane.hh.

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

5.9.2.6 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 208 of file plane.hh.

Referenced by geom::operator==().

5.9.2.7 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 214 of file plane.hh.

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

Referenced by geom::Plane< T >::isPar().

5.9.2.8 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 220 of file plane.hh.

Referenced by geom::Triangle< T >::getPlane().

5.9.2.9 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 226 of file plane.hh.

References geom::Vec3< T >::cross().

5.9.2.10 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 233 of file plane.hh.

References geom::Vec3< T >::normalized().

5.9.2.11 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 240 of file plane.hh.

References geom::Vec3< T >::normalized().

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

- include/primitives/plane.hh

5.10 geom::ThresComp< T > Class Template Reference

```
#include <common.hh>
```

Public Member Functions

- ThresComp ()=delete

Static Public Member Functions

- static void setThreshold (T thres) requires std
- static bool isZero (T num)

5.10.1 Detailed Description

template<Number T>
class geom::ThresComp< T >

Definition at line 28 of file common.hh.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ThresComp()

```
template<Number T>
geom::ThresComp< T >::ThresComp ( )  [delete]
```

5.10.3 Member Function Documentation

5.10.3.1 setThreshold()

```
template<Number T>
static void geom::ThresComp< T >::setThreshold (
            T thres )  [inline], [static]
```

Definition at line 36 of file common.hh.

5.10.3.2 isZero()

```
template<Number T>
static bool geom::ThresComp< T >::isZero (
            T num )  [inline], [static]
```

Definition at line 64 of file common.hh.

Referenced by geom::Vec2< T >::isPar(), geom::Vec2< T >::isPerp(), geom::Vec3< T >::isPerp(), and geom::Line< T >::isSkew().

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

- include/primitives/common.hh

5.11 geom::Triangle< T > Class Template Reference

Triangle class implementation.

`#include <triangle.hh>`

Public Types

- using Iterator = typename std::array< Vec3< T >, 3 >::iterator
- using ConstIterator = typename std::array< Vec3< T >, 3 >::const_iterator

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.
- Vec3< T > & operator[ ] (std::size_t idx) &

  Overloaded operator[] to get access to vertices.
- Iterator begin () &

  Get begin iterator.
- Iterator end () &

  Get end iterator.
- ConstIterator begin () const &

  Get begin const iterator.
- ConstIterator end () const &

  Get end const iterator.
- Plane< T > getPlane () const

  Get triangle's plane.
- bool isValid () const

  Check is triangle valid.
- BoundBox< T > boundBox () const

  Returns triangle's bound box.
- bool belongsTo (const BoundBox< T > &bb) const

  Checks if this Triangle belongs to BoundBox.

5.11.1 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 26 of file triangle.hh.

5.11.2 Member Typedef Documentation

5.11.2.1 Iterator

```
template<std::floating_point T>
using geom::Triangle< T >::Iterator = typename std::array<Vec3<T>, 3>::iterator
```

Definition at line 35 of file triangle.hh.

5.11.2.2 ConstIterator

```
template<std::floating_point T>
using geom::Triangle< T >::ConstIterator = typename std::array<Vec3<T>, 3>::const_iterator
```

Definition at line 36 of file triangle.hh.

5.11.3 Constructor & Destructor Documentation

5.11.3.1 Triangle() [1/2]

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

Construct a new Triangle object.

Definition at line 160 of file triangle.hh.

5.11.3.2 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 164 of file triangle.hh.

5.11.4 Member Function Documentation

5.11.4.1 operator[]() [1/3]

```
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 169 of file triangle.hh.

5.11.4.2 operator[]() [2/3]

```
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 175 of file triangle.hh.

5.11.4.3 operator[]() [3/3]

```
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 181 of file triangle.hh.

5.11.4.4 begin() [1/2]

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

Get begin iterator.

Returns

Iterator

Definition at line 187 of file triangle.hh.

Referenced by geom::detail::helperMollerHaines(), geom::detail::isOnOneSide(), and geom::kdtree::KdTree< T >::isOnSide().

5.11.4.5 end() [1/2]

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

Get end iterator.

Returns

Iterator

Definition at line 193 of file triangle.hh.

Referenced by geom::detail::helperMollerHaines(), geom::detail::isOnOneSide(), and geom::kdtree::KdTree< T >::isOnSide().

5.11.4.6 begin() [2/2]

```
template<std::floating_point T>
ConstIterator geom::Triangle< T >::begin ( ) const &
```

Get begin const iterator.

Returns

ConstIterator

5.11.4.7 end() [2/2]

```
template<std::floating_point T>
ConstIterator geom::Triangle< T >::end ( ) const &
```

Get end const iterator.

Returns

ConstIterator

5.11.4.8 getPlane()

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

Get triangle's plane.

Returns

Plane<T>

Definition at line 211 of file triangle.hh.

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

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

5.11.4.9 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 217 of file triangle.hh.

References geom::cross().

Referenced by geom::isIntersect().

5.11.4.10 boundBox()

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

Returns triangle's bound box.

Returns

BoundBox<T>

Definition at line 227 of file triangle.hh.

Referenced by geom::kdtree::KdTree< T >::insert().

5.11.4.11 belongsTo()

```
template<std::floating_point T>
bool geom::Triangle< T >::belongsTo (
            const BoundBox< T > & bb ) const
```

Checks if this Triangle belongs to BoundBox.

Parameters

in	*bb*	BoundBox

Returns

true if Triangle belongs to BoundBox

false if Triangle doesn't belong to BoundBox

Definition at line 240 of file triangle.hh.

Referenced by geom::kdtree::KdTree< T >::insert().

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

- include/primitives/triangle.hh

5.12 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[ ] (std::size_t i) &

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

- T operator[ ] (std::size_t i) const &

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

  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)

Public Attributes

- T x {}

  Vec2 coordinates.
- T y {}

5.12.1 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 26 of file vec2.hh.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 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 39 of file vec2.hh.

5.12.2.2 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 47 of file vec2.hh.

5.12.3 Member Function Documentation

5.12.3.1 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 342 of file vec2.hh.

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

5.12.3.2 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 351 of file vec2.hh.

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

5.12.3.3 operator-()

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

Unary - operator.

Returns

Vec2 negated Vec2 instance

Definition at line 360 of file vec2.hh.

5.12.3.4 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

5.12.3.5 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

5.12.3.6 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 386 of file vec2.hh.

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

Referenced by geom::dot().

5.12.3.7 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 392 of file vec2.hh.

References geom::dot().

5.12.3.8 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 398 of file vec2.hh.

5.12.3.9 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 404 of file vec2.hh.

5.12.3.10 normalized()

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

Get normalized vector function.

Returns

Vec2 normalized vector

Definition at line 410 of file vec2.hh.

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

5.12.3.11 normalize()

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

Normalize vector function.

Returns

Vec2& reference to instance

Definition at line 418 of file vec2.hh.

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

5.12.3.12 operator[]() [1/3]

```
template<std::floating_point T>
T & geom::Vec2< T >::operator[] (
            std::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 427 of file vec2.hh.

5.12.3.13 operator[]() [2/3]

```
template<std::floating_point T>
T geom::Vec2< T >::operator[] (
            std::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 441 of file vec2.hh.

5.12.3.14 operator[]() [3/3]

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

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 455 of file vec2.hh.

5.12.3.15 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 469 of file vec2.hh.

References geom::ThresComp< T >::isZero(), geom::Vec2< T >::x, and geom::Vec2< T >::y.

5.12.3.16 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 476 of file vec2.hh.

References geom::dot(), and geom::ThresComp< T >::isZero().

5.12.3.17 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

Definition at line 482 of file vec2.hh.

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

Referenced by geom::operator==().

5.12.3.18 operator*=() [2/2]

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

Definition at line 367 of file vec2.hh.

5.12.3.19 operator/=() [2/2]

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

Definition at line 377 of file vec2.hh.

5.12.4 Member Data Documentation

5.12.4.1 x

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

Vec2 coordinates.

Definition at line 31 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<<().

5.12.4.2 y

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

Definition at line 31 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

5.13 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[ ] (std::size_t i) &

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

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

  Overloaded operator [] (rvalue `this` 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)

Public Attributes

- T x {}

  Vec3 coordinates.
- T y {}
- T z {}

5.13.1 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.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 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 40 of file vec3.hh.

5.13.2.2 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 48 of file vec3.hh.

5.13.3 Member Function Documentation

5.13.3.1 operator+=()

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

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

Parameters

in	*vec*	vector to incremented with

Returns

Vec3& reference to current instance

Definition at line 388 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

5.13.3.2 operator-=()

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

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

Parameters

in	*vec*	vector to decremented with

Returns

Vec3& reference to current instance

Definition at line 398 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

5.13.3.3 operator-()

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

Unary - operator.

Returns

Vec3 negated Vec3 instance

Definition at line 408 of file vec3.hh.

5.13.3.4 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

5.13.3.5 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

5.13.3.6 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 438 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

Referenced by geom::dot().

5.13.3.7 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 444 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().

5.13.3.8 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 450 of file vec3.hh.

References geom::dot().

5.13.3.9 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 456 of file vec3.hh.

5.13.3.10 normalized()

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

Get normalized vector function.

Returns

Vec3 normalized vector

Definition at line 462 of file vec3.hh.

References geom::Vec3< T >::normalize().

Referenced by geom::Plane< T >::getNormalDist(), and geom::Plane< T >::getNormalPoint().

5.13.3.11 normalize()

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

Normalize vector function.

Returns

Vec3& reference to instance

Definition at line 470 of file vec3.hh.

Referenced by geom::Vec3< T >::normalized().

5.13.3.12 operator[]() [1/3]

```
template<std::floating_point T>
T & geom::Vec3< T >::operator[] (
            std::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 479 of file vec3.hh.

5.13.3.13 operator[]() [2/3]

```
template<std::floating_point T>
T geom::Vec3< T >::operator[] (
            std::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 495 of file vec3.hh.

5.13.3.14 operator[]() [3/3]

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

Overloaded operator [] (rvalue `this` 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 511 of file vec3.hh.

5.13.3.15 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 527 of file vec3.hh.

References geom::cross().

5.13.3.16 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 533 of file vec3.hh.

References geom::dot(), and geom::ThresComp< T >::isZero().

5.13.3.17 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

Definition at line 539 of file vec3.hh.

References geom::Vec3< T >::x, geom::Vec3< T >::y, and geom::Vec3< T >::z.

Referenced by geom::operator==().

5.13.3.18 operator*=() [2/2]

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

Definition at line 415 of file vec3.hh.

5.13.3.19 operator/=() [2/2]

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

Definition at line 427 of file vec3.hh.

5.13.4 Member Data Documentation

5.13.4.1 x

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

Vec3 coordinates.

Definition at line 31 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>>().

5.13.4.2 y

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

Definition at line 31 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>>().

5.13.4.3 z

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

Definition at line 31 of file vec3.hh.

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

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

- include/primitives/vec3.hh

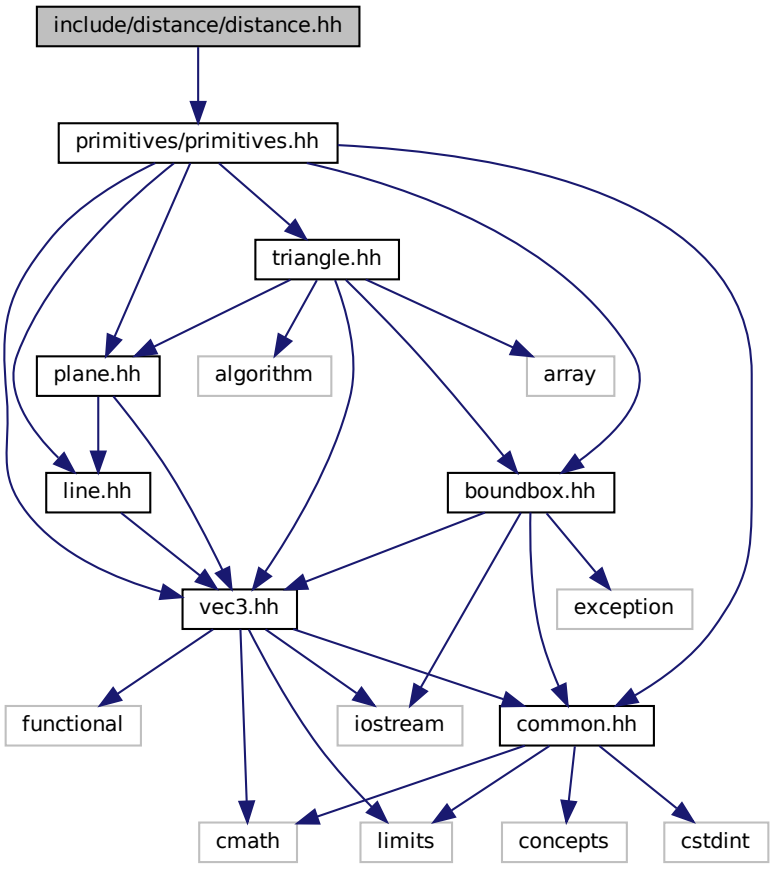
Chapter 6

File Documentation

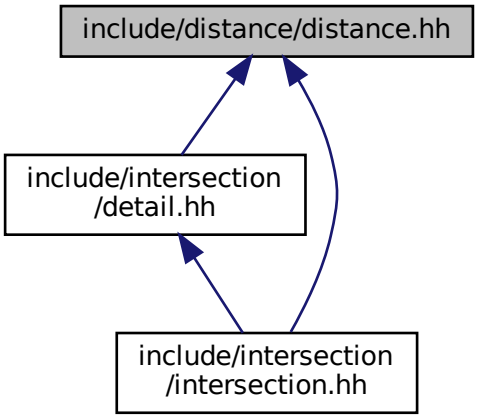
6.1 include/distance/distance.hh File Reference

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

Include dependency graph for distance.hh:



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



Namespaces

- geom

  line.hh Line class implementation

Functions

- template<std::floating_point T>
  T geom::distance (const Plane< T > &pl, const Vec3< T > &pt)

  Calculates signed distance between point and plane.

6.2 distance.hh

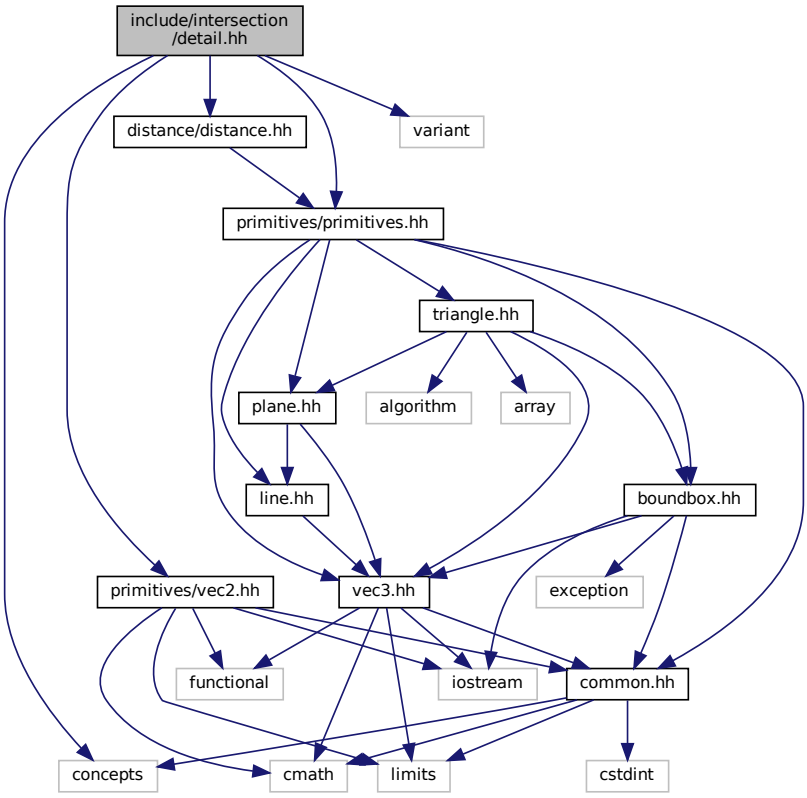
```
00001 #ifndef __INCLUDE_DISTANCE_DISTANCE_HH__
00002 #define __INCLUDE_DISTANCE_DISTANCE_HH__
00003
00004 #include "primitives/primitives.hh"
00005
00006 namespace geom
00007 {
00008
00009 /**
00010  * @brief Calculates signed distance between point and plane
00011  *
00012  * @tparam T - floating point type of coordinates
00013  * @param pl plane
00014  * @param pt point
00015  * @return T signed distance between point and plane
00016  */
00017 template <std::floating_point T>
00018 T distance(const Plane<T> &pl, const Vec3<T> &pt);
00019
00020 } // namespace geom
00021
00022 namespace geom
00023 {
00024
00025 template <std::floating_point T>
```

```
00026 T distance(const Plane<T> &pl, const Vec3<T> &pt)
00027 {
00028   return dot(pt, pl.norm()) - pl.dist();
00029 }
00030
00031 } // namespace geom
00032
00033 #endif // __INCLUDE_DISTANCE_DISTANCE_HH__
```

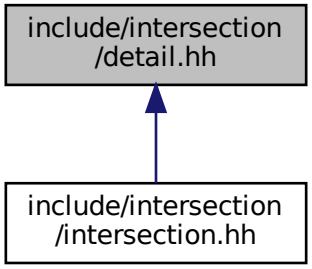
6.3 include/intersection/detail.hh File Reference

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

Include dependency graph for detail.hh:



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



Namespaces

- geom
  line.hh Line class implementation
- geom::detail

Typedefs

- template<typename T >
  using geom::detail::Segment2D = std::pair< T, T >
- template<std::floating_point T>
  using geom::detail::Trian2 = std::array< Vec2< T >, 3 >
- template<std::floating_point T>
  using geom::detail::Segment3D = std::pair< Vec3< T >, Vec3< T > >

Functions

- template<std::floating_point T>
  bool geom::detail::isIntersect2D (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool geom::detail::isIntersectMollerHaines (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  Segment2D< T > geom::detail::helperMollerHaines (const Triangle< T > &tr, const Plane< T > &pl, const Line< T > &l)
- template<std::floating_point T>
  bool geom::detail::isIntersectBothInvalid (const Triangle< T > &tr1, const Triangle< T > &tr2)
- template<std::floating_point T>
  bool geom::detail::isIntersectValidInvalid (const Triangle< T > &valid, const Triangle< T > &invalid)
- template<std::floating_point T>
  bool geom::detail::isIntersectPointTriangle (const Vec3< T > &pt, const Triangle< T > &tr)
- template<std::floating_point T>
  bool geom::detail::isIntersectPointSegment (const Vec3< T > &pt, const Segment3D< T > &segm)
- template<std::floating_point T>
  bool geom::detail::isIntersectSegmentSegment (const Segment3D< T > &segm1, const Segment3D< T > &segm2)

- template<std::floating_point T>
  bool geom::detail::isPoint (const Triangle< T > &tr)
- template<std::floating_point T>
  bool geom::detail::isOverlap (Segment2D< T > &segm1, Segment2D< T > &segm2)
- template<std::forward_iterator It>
  bool geom::detail::isAllPosNeg (It begin, It end)
- template<std::floating_point T>
  bool geom::detail::isAllPosNeg (T num1, T num2)
- template<std::floating_point T>
  bool geom::detail::isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  Trian2< T > geom::detail::getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  bool geom::detail::isCounterClockwise (Trian2< T > &tr)
- template<std::floating_point T>
  Segment2D< T > geom::detail::computeInterval (const Trian2< T > &tr, const Vec2< T > &d)
- template<std::floating_point T>
  Segment3D< T > geom::detail::getSegment (const Triangle< T > &tr)

6.4 detail.hh

```
00001 #ifndef __INCLUDE_INTERSECTION_DETAIL_HH__
00002 #define __INCLUDE_INTERSECTION_DETAIL_HH__
00003
00004 #include <concepts>
00005 #include <variant>
00006
00007 #include "distance/distance.hh"
00008 #include "primitives/primitives.hh"
00009 #include "primitives/vec2.hh"
00010
00011 namespace geom::detail
00012 {
00013
00014 template <typename T>
00015 using Segment2D = std::pair<T, T>;
00016
00017 template <std::floating_point T>
00018 using Trian2 = std::array<Vec2<T>, 3>;
00019
00020 template <std::floating_point T>
00021 using Segment3D = std::pair<Vec3<T>, Vec3<T>>;
00022
00023 template <std::floating_point T>
00024 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2);
00025
00026 template <std::floating_point T>
00027 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2);
00028
00029 template <std::floating_point T>
00030 Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l);
00031
00032 template <std::floating_point T>
00033 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2);
00034
00035 template <std::floating_point T>
00036 bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid);
00037
00038 template <std::floating_point T>
00039 bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr);
00040
00041 template <std::floating_point T>
00042 bool isIntersectPointSegment(const Vec3<T> &pt, const Segment3D<T> &segm);
00043
00044 template <std::floating_point T>
00045 bool isIntersectSegmentSegment(const Segment3D<T> &segm1, const Segment3D<T> &segm2);
00046
00047 template <std::floating_point T>
00048 bool isPoint(const Triangle<T> &tr);
00049
00050 template <std::floating_point T>
00051 bool isOverlap(Segment2D<T> &segm1, Segment2D<T> &segm2);
00052
00053 template <std::forward_iterator It>
```

```
00054 bool isAllPosNeg(It begin, It end);
00055
00056 template <std::floating_point T>
00057 bool isAllPosNeg(T num1, T num2);
00058
00059 template <std::floating_point T>
00060 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr);
00061
00062 template <std::floating_point T>
00063 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr);
00064
00065 template <std::floating_point T>
00066 bool isCounterClockwise(Trian2<T> &tr);
00067
00068 template <std::floating_point T>
00069 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d);
00070
00071 template <std::floating_point T>
00072 Segment3D<T> getSegment(const Triangle<T> &tr);
00073
00074 //=========================================================================
00075
00076 template <std::floating_point T>
00077 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
00078 {
00079   auto pl = tr1.getPlane();
00080
00081   auto trian1 = getTrian2(pl, tr1);
00082   auto trian2 = getTrian2(pl, tr2);
00083
00084   for (auto trian : {trian1, trian2})
00085     for (std::size_t i0 = 0, i1 = 2; i0 < 3; i1 = i0, ++i0)
00086     {
00087       auto d = (trian[i0] - trian[i1]).getPerp();
00088
00089       auto s1 = computeInterval(trian1, d);
00090       auto s2 = computeInterval(trian2, d);
00091
00092       if (s2.second < s1.first || s1.second < s2.first)
00093         return false;
00094     }
00095
00096   return true;
00097 }
00098
00099 template <std::floating_point T>
00100 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
00101 {
00102   auto pl1 = tr1.getPlane();
00103   auto pl2 = tr2.getPlane();
00104
00105   auto l = std::get<Line<T>>(intersect(pl1, pl2));
00106
00107   auto params1 = helperMollerHaines(tr1, pl2, l);
00108   auto params2 = helperMollerHaines(tr2, pl1, l);
00109
00110   return isOverlap(params1, params2);
00111 }
00112
00113 template <std::floating_point T>
00114 Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
00115 {
00116   /* Project the triangle vertices onto line */
00117   std::array<T, 3> vert{};
00118   std::transform(tr.begin(), tr.end(), vert.begin(),
00119                  [dir = l.dir(), org = l.org()](auto &&v) { return dot(dir, v - org); });
00120
00121   std::array<T, 3> sdist{};
00122   std::transform(tr.begin(), tr.end(), sdist.begin(), std::bind_front(distance<T>, pl));
00123
00124   std::array<bool, 3> isOneSide{};
00125   for (std::size_t i = 0; i < 3; ++i)
00126     isOneSide[i] = isAllPosNeg(sdist[i], sdist[(i + 1) % 3]);
00127
00128   /* Looking for vertex which is alone on it's side */
00129   std::size_t rogue = 0;
00130   if (std::all_of(isOneSide.begin(), isOneSide.end(), [](const auto &elem) { return !elem; }))
00131   {
00132     auto rogueIt = std::find_if_not(sdist.rbegin(), sdist.rend(), ThresComp<T>::isZero);
00133     if (rogueIt != sdist.rend())
00134       rogue = std::distance(rogueIt, sdist.rend()) - 1;
00135   }
00136   else
00137   {
00138     for (std::size_t i = 0; i < 3; ++i)
00139       if (isOneSide[i])
00140         rogue = (i + 2) % 3;
```

```
00141   }
00142
00143   std::array<T, 2> segm{};
00144   std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};
00145   std::transform(arr.begin(), arr.end(), segm.begin(), [&vert, &sdist, rogue](auto i) {
00146     return vert[i] + (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]);
00147   });
00148
00149   std::sort(segm.begin(), segm.end());
00150   return {segm[0], segm[1]};
00151 }
00152
00153 template <std::floating_point T>
00154 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
00155 {
00156   auto isPoint1 = isPoint(tr1);
00157   auto isPoint2 = isPoint(tr2);
00158
00159   if (isPoint1 && isPoint2)
00160     return tr1[0] == tr2[0];
00161
00162   if (isPoint1)
00163     return isIntersectPointSegment(tr1[0], getSegment(tr2));
00164
00165   if (isPoint2)
00166     return isIntersectPointSegment(tr2[0], getSegment(tr1));
00167
00168   return isIntersectSegmentSegment(getSegment(tr1), getSegment(tr2));
00169 }
00170
00171 template <std::floating_point T>
00172 bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid)
00173 {
00174   if (isPoint(invalid))
00175     return isIntersectPointTriangle(invalid[0], valid);
00176
00177   auto segm = getSegment(invalid);
00178   auto pl = valid.getPlane();
00179
00180   auto dst1 = distance(pl, segm.first);
00181   auto dst2 = distance(pl, segm.second);
00182
00183   if (dst1 * dst2 > 0)
00184     return false;
00185
00186   if (ThresComp<T>::isZero(dst1) && ThresComp<T>::isZero(dst2))
00187     return isIntersect2D(valid, invalid);
00188
00189   dst1 = std::abs(dst1);
00190   dst2 = std::abs(dst2);
00191
00192   auto pt = segm.first + (segm.second - segm.first) * dst1 / (dst1 + dst2);
00193   return isIntersectPointTriangle(pt, valid);
00194 }
00195
00196 template <std::floating_point T>
00197 bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr)
00198 {
00199   if (!tr.getPlane().belongs(pt))
00200     return false;
00201
00202   /* TODO: comment better */
00203   /* pt = point + u * edge1 + v * edge2 */
00204   auto point = pt - tr[0];
00205   auto edge1 = tr[1] - tr[0];
00206   auto edge2 = tr[2] - tr[0];
00207
00208   auto dotE1E1 = dot(edge1, edge1);
00209   auto dotE1E2 = dot(edge1, edge2);
00210   auto dotE1PT = dot(edge1, point);
00211
00212   auto dotE2E2 = dot(edge2, edge2);
00213   auto dotE2PT = dot(edge2, point);
00214
00215   auto denom = dotE1E1 * dotE2E2 - dotE1E2 * dotE1E2;
00216   auto u = (dotE2E2 * dotE1PT - dotE1E2 * dotE2PT) / denom;
00217   auto v = (dotE1E1 * dotE2PT - dotE1E2 * dotE1PT) / denom;
00218
00219   /* Point belongs to triangle if: (u >= 0) && (v >= 0) && (u + v <= 1) */
00220   auto eps = ThresComp<T>::getThreshold();
00221   return (u > -eps) && (v > -eps) && (u + v < 1 + eps);
00222 }
00223
00224 template <std::floating_point T>
00225 bool isIntersectPointSegment(const Vec3<T> &pt, const Segment3D<T> &segm)
00226 {
00227   Line<T> l{segm.first, segm.second - segm.first};
```

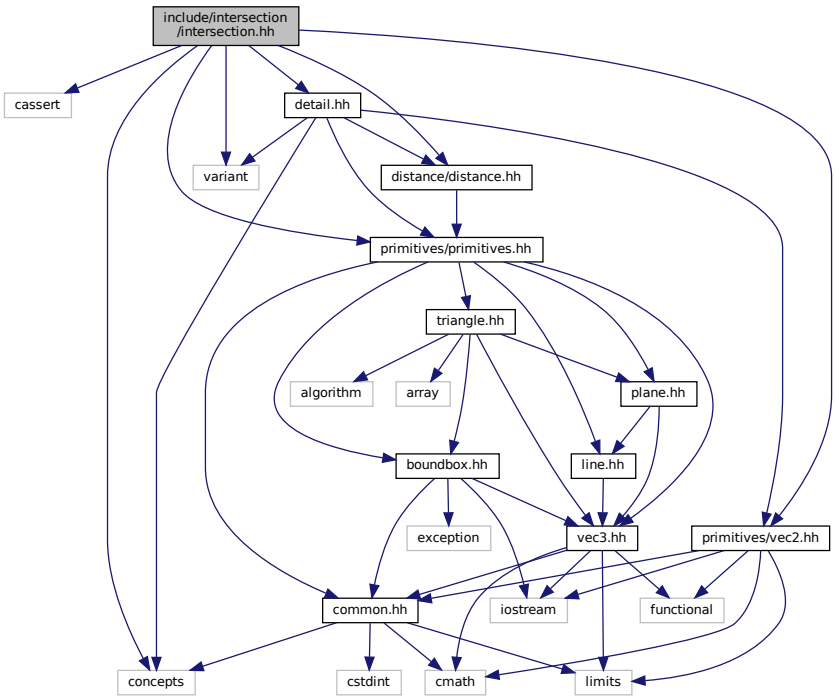
```
00228   if (!l.belongs(pt))
00229     return false;
00230
00231   auto beg = dot(l.dir(), segm.first - pt);
00232   auto end = dot(l.dir(), segm.second - pt);
00233
00234   return !isAllPosNeg(beg, end);
00235 }
00236
00237 template <std::floating_point T>
00238 bool isIntersectSegmentSegment(const Segment3D<T> &segm1, const Segment3D<T> &segm2)
00239 {
00240   Line<T> l1{segm1.first, segm1.second - segm1.first};
00241   Line<T> l2{segm2.first, segm2.second - segm2.first};
00242   auto intersectionResult = intersect(l1, l2);
00243
00244   if (std::holds_alternative<Line<T>>(intersectionResult))
00245   {
00246     const auto &dir = l1.dir();
00247     Segment2D<T> s1{dot(dir, segm1.first), dot(dir, segm1.second)};
00248     Segment2D<T> s2{dot(dir, segm2.first), dot(dir, segm2.second)};
00249     return isOverlap(s1, s2);
00250   }
00251
00252   if (std::holds_alternative<Vec3<T>>(intersectionResult))
00253   {
00254     auto pt = std::get<Vec3<T»(intersectionResult);
00255     return isIntersectPointSegment(pt, segm1) && isIntersectPointSegment(pt, segm2);
00256   }
00257
00258   return false;
00259 }
00260
00261 template <std::floating_point T>
00262 bool isPoint(const Triangle<T> &tr)
00263 {
00264   return (tr[0] == tr[1]) && (tr[0] == tr[2]);
00265 }
00266
00267 template <std::floating_point T>
00268 bool isOverlap(Segment2D<T> &segm1, Segment2D<T> &segm2)
00269 {
00270   return (segm2.first <= segm1.second) && (segm2.second >= segm1.first);
00271 }
00272
00273 template <std::forward_iterator It>
00274 bool isAllPosNeg(It begin, It end)
00275 {
00276   if (begin == end)
00277     return true;
00278
00279   bool fst = (*begin > 0);
00280   return std::none_of(std::next(begin), end, [fst](auto &&elt) {
00281     return (elt > 0) != fst || ThresComp<std::remove_reference_t<decltype(elt)»::isZero(elt);
00282   });
00283 }
00284
00285 template <std::floating_point T>
00286 bool isAllPosNeg(T num1, T num2)
00287 {
00288   auto thres = ThresComp<T>::getThreshold();
00289   return (num1 > thres && num2 > thres) || (num1 < -thres && num2 < -thres);
00290 }
00291
00292 template <std::floating_point T>
00293 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
00294 {
00295   std::array<T, 3> sdist{};
00296   std::transform(tr.begin(), tr.end(), sdist.begin(), std::bind_front(distance<T>, pl));
00297   return detail::isAllPosNeg(sdist.begin(), sdist.end());
00298 }
00299
00300 template <std::floating_point T>
00301 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr)
00302 {
00303   auto norm = pl.norm();
00304
00305   const Vec3<T> x{1, 0, 0};
00306   const Vec3<T> y{0, 1, 0};
00307   const Vec3<T> z{0, 0, 1};
00308
00309   std::array<Vec3<T>, 3> xyz{x, y, z};
00310   std::array<T, 3> xyzDot;
00311
00312   std::transform(xyz.begin(), xyz.end(), xyzDot.begin(),
00313                  [&norm](const auto &axis) { return std::abs(dot(axis, norm)); });
00314
```

```
00315   auto maxIt = std::max_element(xyzDot.begin(), xyzDot.end());
00316   auto maxIdx = static_cast<std::size_t>(std::distance(xyzDot.begin(), maxIt));
00317
00318   Trian2<T> res;
00319   for (std::size_t i = 0; i < 3; ++i)
00320     for (std::size_t j = 0, k = 0; j < 2; ++j, ++k)
00321     {
00322       if (k == maxIdx)
00323         ++k;
00324
00325       res[i][j] = tr[i][k];
00326     }
00327
00328   if (!isCounterClockwise(res))
00329     std::swap(res[0], res[1]);
00330
00331   return res;
00332 }
00333
00334 template <std::floating_point T>
00335 bool isCounterClockwise(Trian2<T> &tr)
00336 {
00337   /**
00338    * The triangle is counterclockwise ordered if \delta > 0
00339    * and clockwise ordered if \delta < 0.
00340    *
00341    *              +  1  1  1 +
00342    * \delta = det | x0 x1 x2 | = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0)
00343    *              + y0 y1 y2 +                       + (x0 * y1 - x1 * y0)
00344    *
00345    */
00346
00347   auto x0 = tr[0][0], x1 = tr[1][0], x2 = tr[2][0];
00348   auto y0 = tr[0][1], y1 = tr[1][1], y2 = tr[2][1];
00349
00350   auto delta = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0) + (x0 * y1 - x1 * y0);
00351   return (delta > 0);
00352 }
00353
00354 template <std::floating_point T>
00355 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
00356 {
00357   std::array<T, 3> dotArr{};
00358   std::transform(tr.begin(), tr.end(), dotArr.begin(), [&d](auto &&v) { return dot(d, v); });
00359   auto mmIt = std::minmax_element(dotArr.begin(), dotArr.end());
00360   return {*mmIt.first, *mmIt.second};
00361 }
00362
00363 template <std::floating_point T>
00364 Segment3D<T> getSegment(const Triangle<T> &tr)
00365 {
00366   std::array<T, 3> lenArr{};
00367   for (std::size_t i = 0; i < 3; ++i)
00368     lenArr[i] = (tr[i] - tr[i + 1]).length2();
00369
00370   auto maxIt = std::max_element(lenArr.begin(), lenArr.end());
00371   auto maxIdx = static_cast<std::size_t>(std::distance(lenArr.begin(), maxIt));
00372
00373   return {tr[maxIdx], tr[maxIdx + 1]};
00374 }
00375
00376 } // namespace geom::detail
00377
00378 #endif // __INCLUDE_INTERSECTION_DETAIL_HH__
```

6.5 include/intersection/intersection.hh File Reference

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

Include dependency graph for intersection.hh:



Namespaces

- geom

  line.hh Line class implementation

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>
  std::variant< std::monostate, Vec3< T >, Line< T > > geom::intersect (const Line< T > &l1, const Line< T > &l2)

  Intersect 2 lines and return result of intersection.

6.6 intersection.hh

```
00001 #ifndef __INCLUDE_INTERSECTION_INTERSECTION_HH__
00002 #define __INCLUDE_INTERSECTION_INTERSECTION_HH__
00003
00004 #include <cassert>
00005 #include <concepts>
```

```
00006 #include <variant>
00007
00008 #include "distance/distance.hh"
00009 #include "primitives/primitives.hh"
00010 #include "primitives/vec2.hh"
00011
00012 #include "detail.hh"
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @brief Checks intersection of 2 triangles
00019  *
00020  * @tparam T - floating point type of coordinates
00021  * @param tr1 first triangle
00022  * @param tr2 second triangle
00023  * @return true if triangles are intersect
00024  * @return false if triangles are not intersect
00025  */
00026 template <std::floating_point T>
00027 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2);
00028
00029 /**
00030  * @brief Intersect 2 planes and return result of intersection
00031  * @details
00032  * Common intersection case (parallel planes case is trivial):
00033  *
00034  * Let \f$ \overrightarrow{P} \f$ - point in space
00035  *
00036  * \f$ pl_1 \f$ equation: \f$ \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1 \f$
00037  *
00038  * \f$ pl_2 \f$ equation: \f$ \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2 \f$
00039  *
00040  * Intersection line direction: \f$ \overrightarrow{dir} = \overrightarrow{n}_1 \times
00041  * \overrightarrow{n}_2 \f$
00042  *
00043  * Let origin of intersection line be a linear combination of \f$ \overrightarrow{n}_1 \f$
00044  * and \f$ \overrightarrow{n}_2 \f$: \f[ \overrightarrow{P} = a \cdot \overrightarrow{n}_1
00045  * + b \cdot \overrightarrow{n}_2 \f]
00046  *
00047  * \f$ \overrightarrow{P} \f$ must satisfy both \f$ pl_1 \f$ and \f$ pl_1 \f$ equations:
00048  * \f[
00049  * \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1
00050  * \Leftrightarrow
00051  * \overrightarrow{n}_1
00052  * \cdot
00053  * \left(
00054  *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00055  *  \right)
00056  *  = d_1
00057  * \Leftrightarrow
00058  * a + b \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2  = d_1
00059  * \f]
00060  * \f[
00061  * \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2
00062  * \Leftrightarrow
00063  * \overrightarrow{n}_2
00064  * \cdot
00065  * \left(
00066  *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00067  *  \right) = d_2
00068  * \Leftrightarrow
00069  * a \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 + b = d_2
00070  * \f]
00071  *
00072  * Let's find \f$a\f$ and \f$b\f$:
00073  * \f[
00074  * a = \frac{
00075  *  d_2 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_1
00076  * }{
00077  *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
00078  * }
00079  * \f]
00080  * \f[
00081  * b = \frac{
00082  * d_1 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_2
00083  * }{
00084  *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
00085  * }
00086  * \f]
00087  *
00088  * Intersection line equation:
00089  * \f[
00090  * \overrightarrow{r}(t) = \overrightarrow{P} + t \cdot \overrightarrow{n}_1 \times
00091  * \overrightarrow{n}_2 = (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) +
00092  * t \cdot \overrightarrow{n}_1 \times \overrightarrow{n}_2 \f]
```

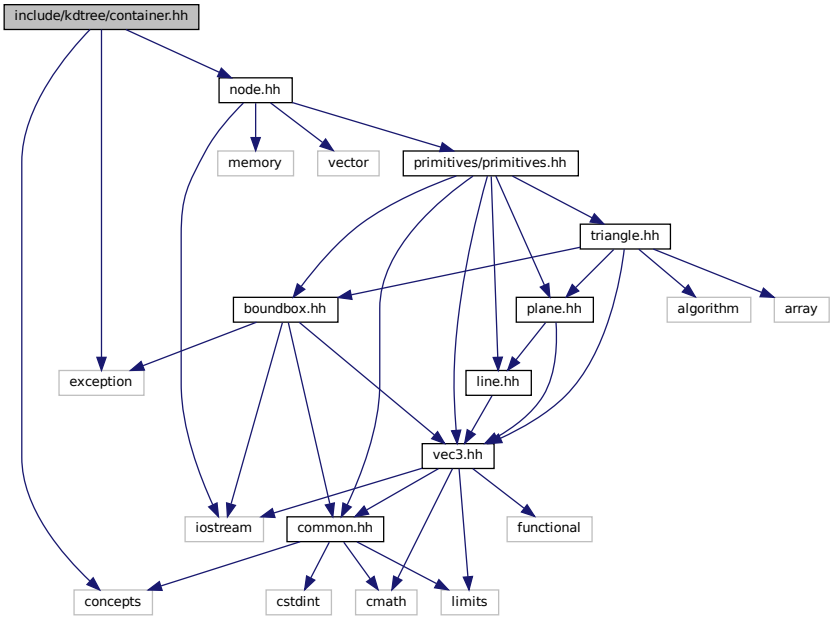
```
00093  *
00094  * @tparam T - floating point type of coordinates
00095  * @param[in] pl1 first plane
00096  * @param[in] pl2 second plane
00097  * @return std::variant<std::monostate, Line<T>, Plane<T»
00098  */
00099 template <std::floating_point T>
00100 std::variant<std::monostate, Line<T>, Plane<T» intersect(const Plane<T> &pl1, const Plane<T> &pl2);
00101
00102 /**
00103  * @brief Intersect 2 lines and return result of intersection
00104  * @details
00105  * Common intersection case (parallel & skew lines cases are trivial):
00106  * Let \f$ \overrightarrow{P} \f$ - point in space, intersection point of two lines.
00107  *
00108  * \f$ l_1 \f$ equation: \f$ \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
00109  * \overrightarrow{P} \f$
00110  *
00111  * \f$ l_2 \f$ equation: \f$ \overrightarrow{org}_2 + \overrightarrow{dir}_2
00112  * \cdot t_2 = \overrightarrow{P} \f$
00113  *
00114  * Let's equate left sides:
00115  * \f[
00116  * \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
00117  * \overrightarrow{org}_2 + \overrightarrow{dir}_2 \cdot t_2
00118  * \f]
00119  * Cross multiply both sides from right by \f$ \overrightarrow{dir}_2 \f$:
00120  * \f[
00121  * t_1 \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right) =
00122  * \left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times \overrightarrow{dir}_2
00123  * \f]
00124  * Dot multiply both sides by \f$ \frac{\overrightarrow{dir}_1 \times \overrightarrow{dir}_2}{\left|
00125  * \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2} \f$:
00126  *
00127  * \f[
00128  * t_1 = \frac{
00129  *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times
00130  * \overrightarrow{dir}_2\right) \cdot \left( \overrightarrow{dir}_1 \times \overrightarrow{dir}_2
00131  * \right)
00132  * }{
00133  * \left| \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2
00134  * }
00135  * \f]
00136  *
00137  * Thus we get intersection point parameter \f$ t_1 \f$ on \f$ l_1 \f$, let's substitute it to \f$
00138  * l_1 \f$ equation: \f[ \overrightarrow{P} = \overrightarrow{org}_1 + \frac{
00139  *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times
00140  * \overrightarrow{dir}_2\right) \cdot \left( \overrightarrow{dir}_1 \times \overrightarrow{dir}_2
00141  * \right)
00142  * }{
00143  * \left| \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2
00144  * } \cdot \overrightarrow{dir}_1
00145  * \f]
00146  *
00147  * @tparam T - floating point type of coordinates
00148  * @param[in] l1 first line
00149  * @param[in] l2 second line
00150  * @return std::variant<std::monostate, Vec3<T>, Line<T»
00151  */
00152 template <std::floating_point T>
00153 std::variant<std::monostate, Vec3<T>, Line<T» intersect(const Line<T> &l1, const Line<T> &l2);
00154
00155 template <std::floating_point T>
00156 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
00157 {
00158   auto isInv1 = !tr1.isValid();
00159   auto isInv2 = !tr2.isValid();
00160
00161   if (isInv1 && isInv2)
00162     return detail::isIntersectBothInvalid(tr1, tr2);
00163
00164   if (isInv1)
00165     return detail::isIntersectValidInvalid(tr2, tr1);
00166
00167   if (isInv2)
00168     return detail::isIntersectValidInvalid(tr1, tr2);
00169
00170   auto pl1 = tr1.getPlane();
00171   if (detail::isOnOneSide(pl1, tr2))
00172     return false;
00173
00174   auto pl2 = tr2.getPlane();
00175   if (pl1 == pl2)
00176     return detail::isIntersect2D(tr1, tr2);
00177
00178   if (pl1.isPar(pl2))
00179     return false;
```

```
00180
00181   if (detail::isOnOneSide(pl2, tr1))
00182     return false;
00183
00184   return detail::isIntersectMollerHaines(tr1, tr2);
00185 }
00186
00187 template <std::floating_point T>
00188 std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &pl1, const Plane<T> &pl2)
00189 {
00190   const auto &n1 = pl1.norm();
00191   const auto &n2 = pl2.norm();
00192
00193   auto dir = cross(n1, n2);
00194
00195   /* if planes are parallel */
00196   if (Vec3<T>{0} == dir)
00197   {
00198     if (pl1 == pl2)
00199       return pl1;
00200
00201     return std::monostate{};
00202   }
00203
00204   auto n1n2 = dot(n1, n2);
00205   auto d1 = pl1.dist();
00206   auto d2 = pl2.dist();
00207
00208   auto a = (d2 * n1n2 - d1) / (n1n2 * n1n2 - 1);
00209   auto b = (d1 * n1n2 - d2) / (n1n2 * n1n2 - 1);
00210
00211   return Line<T>{(a * n1) + (b * n2), dir};
00212 }
00213
00214 template <std::floating_point T>
00215 std::variant<std::monostate, Vec3<T>, Line<T>> intersect(const Line<T> &l1, const Line<T> &l2)
00216 {
00217   if (l1.isPar(l2))
00218   {
00219     if (l1.isEqual(l2))
00220       return l1;
00221
00222     return std::monostate{};
00223   }
00224
00225   if (l1.isSkew(l2))
00226     return std::monostate{};
00227
00228   auto dir1xdir2 = cross(l1.dir(), l2.dir());
00229   auto org21xdir2 = cross(l2.org() - l1.org(), l2.dir());
00230
00231   auto t1_intersect = dot(org21xdir2, dir1xdir2) / dir1xdir2.length2();
00232
00233   auto point = l1.getPoint(t1_intersect);
00234
00235   return point;
00236 }
00237
00238 } // namespace geom
00239
00240 #endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__
```

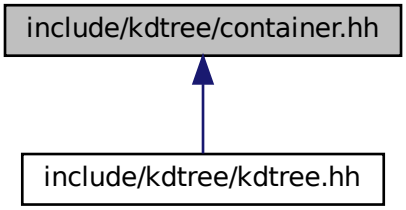
6.7 include/kdtree/container.hh File Reference

```
#include <concepts>
#include <exception>
#include "node.hh"
```

Include dependency graph for container.hh:



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



Classes

- class geom::kdtree::KdTree< T >
- class geom::kdtree::Container< T >
- class geom::kdtree::Container< T >::ConstIterator

Namespaces

- geom

  line.hh Line class implementation
- geom::kdtree

6.8 container.hh

```
00001 #ifndef __INCLUDE_KDTREE_CONTAINER_HH__
00002 #define __INCLUDE_KDTREE_CONTAINER_HH__
00003
00004 #include <concepts>
00005 #include <exception>
00006
00007 #include "node.hh"
00008
00009 namespace geom::kdtree
00010 {
00011
00012 template <std::floating_point T>
00013 class KdTree;
00014
00015 template <std::floating_point T>
00016 class Container final
00017 {
00018 private:
00019   const KdTree<T> *tree_;
00020   const Node<T> *node_;
00021
00022 public:
00023   Container(const KdTree<T> *tree, const Node<T> *node);
00024
00025   class ConstIterator;
00026   ConstIterator cbegin() const &;
00027   ConstIterator cend() const &;
00028
00029   ConstIterator begin() const &;
00030   ConstIterator end() const &;
00031
00032   typename Node<T>::IndexConstIterator indexBegin() const &;
00033   typename Node<T>::IndexConstIterator indexEnd() const &;
00034
00035   T separator() const;
00036   Axis sepAxis() const;
00037   BoundBox<T> boundBox() const;
00038   const Triangle<T> &triangleByIndex(Index index) const &;
00039
00040   Container left() const;
00041   Container right() const;
00042
00043   bool isValid() const;
00044
00045   class ConstIterator final
00046   {
00047   public:
00048     using iterator_category = std::forward_iterator_tag;
00049     using difference_type = std::size_t;
00050     using value_type = Triangle<T>;
00051     using reference = const Triangle<T> &;
00052     using pointer = const Triangle<T> *;
00053
00054   private:
00055     const Container *cont_;
00056     std::vector<Index>::const_iterator curIdxIt_{};
00057
00058   public:
00059     ConstIterator(const Container *cont, bool isEnd = false);
00060
00061     Index getIndex();
00062
00063     ConstIterator &operator++();
00064     ConstIterator operator++(int);
00065
00066     reference operator*() const;
00067     pointer operator->() const;
00068
00069     bool operator==(const ConstIterator &lhs) const;
00070     bool operator!=(const ConstIterator &lhs) const;
00071   };
00072 };
00073
00074 //=================================================================================================
00075 //                                    Container definitions
00076 //=================================================================================================
00077
00078 template <std::floating_point T>
00079 Container<T>::Container(const KdTree<T> *tree, const Node<T> *node) : tree_(tree), node_(node)
00080 {}
00081
00082 template <std::floating_point T>
00083 typename Container<T>::ConstIterator Container<T>::cbegin() const &
00084 {
00085   return ConstIterator{this};
```

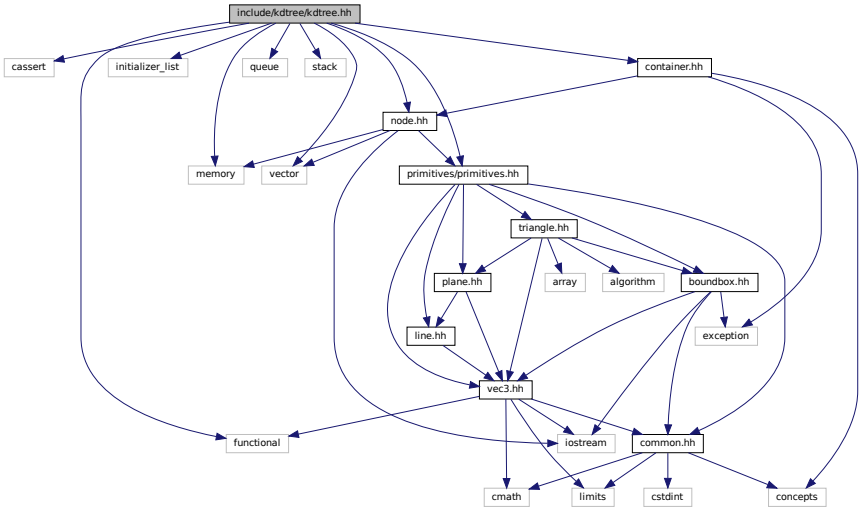
```
00086 }
00087
00088 template <std::floating_point T>
00089 typename Container<T>::ConstIterator Container<T>::cend() const &
00090 {
00091   return ConstIterator{this, /* isEnd = */ true};
00092 }
00093
00094 template <std::floating_point T>
00095 typename Container<T>::ConstIterator Container<T>::begin() const &
00096 {
00097   return cbegin();
00098 }
00099
00100 template <std::floating_point T>
00101 typename Container<T>::ConstIterator Container<T>::end() const &
00102 {
00103   return cend();
00104 }
00105
00106 template <std::floating_point T>
00107 typename Node<T>::IndexConstIterator Container<T>::indexBegin() const &
00108 {
00109   return node_->indicies.begin();
00110 }
00111
00112 template <std::floating_point T>
00113 typename Node<T>::IndexConstIterator Container<T>::indexEnd() const &
00114 {
00115   return node_->indicies.end();
00116 }
00117
00118 template <std::floating_point T>
00119 T Container<T>::separator() const
00120 {
00121   return node_->separator;
00122 }
00123
00124 template <std::floating_point T>
00125 Axis Container<T>::sepAxis() const
00126 {
00127   return node_->sepAxis;
00128 }
00129
00130 template <std::floating_point T>
00131 BoundBox<T> Container<T>::boundBox() const
00132 {
00133   return node_->boundBox;
00134 }
00135
00136 template <std::floating_point T>
00137 const Triangle<T> &Container<T>::triangleByIndex(Index index) const &
00138 {
00139   return tree_->triangleByIndex(index);
00140 }
00141
00142 template <std::floating_point T>
00143 Container<T> Container<T>::left() const
00144 {
00145   return Container<T>{tree_, node_->left.get()};
00146 }
00147
00148 template <std::floating_point T>
00149 Container<T> Container<T>::right() const
00150 {
00151   return Container<T>{tree_, node_->right.get()};
00152 }
00153
00154 template <std::floating_point T>
00155 bool Container<T>::isValid() const
00156 {
00157   return (tree_ != nullptr) && (node_ != nullptr);
00158 }
00159
00160 //=================================================================================================
00161 //                                Container::ConstIterator definitions
00162 //=================================================================================================
00163
00164 template <std::floating_point T>
00165 Container<T>::ConstIterator::ConstIterator(const Container<T> *cont, bool isEnd) : cont_(cont)
00166 {
00167   if (nullptr == cont_)
00168     throw std::invalid_argument("Tried to create iterator with invalid Container pointer");
00169
00170   if (isEnd)
00171     curIdxIt_ = cont_->indexEnd();
00172   else
```

```
00173     curIdxIt_ = cont_->indexBegin();
00174 }
00175
00176 template <std::floating_point T>
00177 Index Container<T>::ConstIterator::getIndex()
00178 {
00179   return *curIdxIt_;
00180 }
00181
00182 template <std::floating_point T>
00183 typename Container<T>::ConstIterator &Container<T>::ConstIterator::operator++()
00184 {
00185   ++curIdxIt_;
00186   return *this;
00187 }
00188
00189 template <std::floating_point T>
00190 typename Container<T>::ConstIterator Container<T>::ConstIterator::operator++(int)
00191 {
00192   auto tmp = *this;
00193   operator++();
00194   return tmp;
00195 }
00196
00197 template <std::floating_point T>
00198 typename Container<T>::ConstIterator::reference Container<T>::ConstIterator::operator*() const
00199 {
00200   return cont_->triangleByIndex(*curIdxIt_);
00201 }
00202
00203 template <std::floating_point T>
00204 typename Container<T>::ConstIterator::pointer Container<T>::ConstIterator::operator->() const
00205 {
00206   return &cont_->triangleByIndex(*curIdxIt_);
00207 }
00208
00209 template <std::floating_point T>
00210 bool Container<T>::ConstIterator::operator==(const Container<T>::ConstIterator &lhs) const
00211 {
00212   return (cont_ == lhs.cont_) && (curIdxIt_ == lhs.curIdxIt_);
00213 }
00214
00215 template <std::floating_point T>
00216 bool Container<T>::ConstIterator::operator!=(const Container<T>::ConstIterator &lhs) const
00217 {
00218   return !operator==(lhs);
00219 }
00220
00221 } // namespace geom::kdtree
00222
00223 #endif // __INCLUDE_KDTREE_CONTAINER_HH__
```

6.9 include/kdtree/kdtree.hh File Reference

```
#include <cassert>
#include <functional>
#include <initializer_list>
#include <memory>
#include <queue>
#include <stack>
#include <vector>
#include "primitives/primitives.hh"
#include "container.hh"
#include "node.hh"
```

Include dependency graph for kdtree.hh:



Classes

- class geom::kdtree::KdTree< T >
- struct geom::kdtree::KdTree< T >::ContainerPtr
- class geom::kdtree::KdTree< T >::ConstIterator

Namespaces

- geom

  line.hh Line class implementation
- geom::kdtree

6.10 kdtree.hh

```
00001 #ifndef __INCLUDE_KDTREE_KDTREE_HH__
00002 #define __INCLUDE_KDTREE_KDTREE_HH__
00003
00004 #include <cassert>
00005 #include <functional>
00006 #include <initializer_list>
00007 #include <memory>
00008 #include <queue>
00009 #include <stack>
00010 #include <vector>
00011
00012 #include "primitives/primitives.hh"
00013
00014 #include "container.hh"
00015 #include "node.hh"
00016
00017 namespace geom::kdtree
00018 {
00019
00020 template <std::floating_point T>
00021 class KdTree
00022 {
00023 private:
00024   std::unique_ptr<Node<T>> root_{};
00025   std::vector<Triangle<T>> triangles_{};
00026   std::size_t nodeCapacity_{1};
```

```
00027
00028 public:
00029   KdTree(std::initializer_list<Triangle<T>> il);
00030   KdTree(const KdTree &tree);
00031   KdTree(KdTree &&tree) = default;
00032   KdTree() = default;
00033   ~KdTree();
00034
00035   KdTree &operator=(const KdTree &tree);
00036   KdTree &operator=(KdTree &&tree) = default;
00037
00038   class ConstIterator;
00039
00040   // ConstIterators
00041   ConstIterator cbegin() const &;
00042   ConstIterator cend() const &;
00043
00044   ConstIterator begin() const &;
00045   ConstIterator end() const &;
00046
00047   ConstIterator beginFrom(const ConstIterator &iter) const &;
00048
00049   // Modifiers
00050   void insert(const Triangle<T> &tr);
00051   void clear();
00052   void setNodeCapacity(std::size_t newCap);
00053
00054   // Capacity
00055   bool empty() const;
00056   std::size_t size() const;
00057   std::size_t nodeCapacity() const;
00058
00059   const Triangle<T> &triangleByIndex(Index index) const &;
00060
00061   void dumpRecursive(std::ostream &ost = std::cout) const;
00062
00063   static bool isOnPosSide(Axis axis, T separator, const Triangle<T> &tr);
00064   static bool isOnNegSide(Axis axis, T separator, const Triangle<T> &tr);
00065   static bool isOnSide(Axis axis, T separator, const Triangle<T> &tr,
00066                        std::function<bool(T, T)> comparator);
00067
00068 private:
00069   void expandingInsert(const Triangle<T> &tr);
00070   void tryExpandRight(Axis axis, const BoundBox<T> &trianBB);
00071   void tryExpandLeft(Axis axis, const BoundBox<T> &trianBB);
00072
00073   void nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv = false);
00074   bool isDivisable(const Node<T> *node);
00075   void subdivide(Node<T> *node);
00076
00077 public:
00078   struct ContainerPtr final
00079   {
00080     Container<T> cont;
00081     const Container<T> *operator->() const;
00082   };
00083
00084   class ConstIterator final
00085   {
00086   public:
00087     using iterator_category = std::forward_iterator_tag;
00088     using difference_type = std::size_t;
00089     using value_type = Container<T>;
00090     using reference = Container<T>;
00091     using pointer = ContainerPtr;
00092
00093   private:
00094     const KdTree<T> *tree_;
00095     const Node<T> *node_;
00096     std::queue<const Node<T> *> fifo_;
00097
00098   public:
00099     ConstIterator(const KdTree<T> *tree, const Node<T> *node);
00100
00101     ConstIterator &operator++();
00102     ConstIterator operator++(int);
00103
00104     reference operator*() const;
00105     pointer operator->() const;
00106
00107     bool operator==(const ConstIterator &lhs) const;
00108     bool operator!=(const ConstIterator &lhs) const;
00109
00110     static ConstIterator beginFrom(const ConstIterator &iter);
00111   };
00112 };
00113
```

```
00114 //================================================================================================
00115 //                                    KdTree definitions
00116 //================================================================================================
00117
00118 template <std::floating_point T>
00119 KdTree<T>::KdTree(std::initializer_list<Triangle<T>> il)
00120 {
00121   for (const auto &tr : il)
00122     insert(tr);
00123 }
00124
00125 template <std::floating_point T>
00126 KdTree<T>::KdTree(const KdTree<T> &tree)
00127 {
00128   // temporary solution
00129   for (const auto &tr : tree.triangles_)
00130     insert(tr);
00131 }
00132
00133 template <std::floating_point T>
00134 KdTree<T>::~KdTree()
00135 {
00136   clear();
00137 }
00138
00139 template <std::floating_point T>
00140 KdTree<T> &KdTree<T>::operator=(const KdTree<T> &tree)
00141 {
00142   KdTree tmp{tree};
00143   operator=(std::move(tmp));
00144   return *this;
00145 }
00146
00147 // ConstIterators
00148 template <std::floating_point T>
00149 typename KdTree<T>::ConstIterator KdTree<T>::cbegin() const &
00150 {
00151   return ConstIterator{this, root_.get()};
00152 }
00153
00154 template <std::floating_point T>
00155 typename KdTree<T>::ConstIterator KdTree<T>::cend() const &
00156 {
00157   return ConstIterator{this, nullptr};
00158 }
00159
00160 template <std::floating_point T>
00161 typename KdTree<T>::ConstIterator KdTree<T>::begin() const &
00162 {
00163   return cbegin();
00164 }
00165
00166 template <std::floating_point T>
00167 typename KdTree<T>::ConstIterator KdTree<T>::end() const &
00168 {
00169   return cend();
00170 }
00171
00172 template <std::floating_point T>
00173 typename KdTree<T>::ConstIterator KdTree<T>::beginFrom(
00174   const typename KdTree<T>::ConstIterator &iter) const &
00175 {
00176   return KdTree<T>::ConstIterator::beginFrom(iter);
00177 }
00178
00179 // Modifiers
00180 template <std::floating_point T>
00181 void KdTree<T>::insert(const Triangle<T> &tr)
00182 {
00183   if (nullptr == root_)
00184   {
00185     root_ = std::unique_ptr<Node<T>>{new Node<T>{T{}, Axis::NONE, tr.boundBox(), {0}}};
00186     triangles_.push_back(tr);
00187     return;
00188   }
00189
00190   if (!tr.belongsTo(root_->boundBox))
00191     expandingInsert(tr);
00192   else
00193   {
00194     auto index = triangles_.size();
00195     triangles_.push_back(tr);
00196     nonExpandingInsert(root_.get(), tr, index);
00197   }
00198 }
00199
00200 template <std::floating_point T>
```

```
00201 void KdTree<T>::clear()
00202 {
00203   if (nullptr == root_)
00204     return;
00205
00206   std::stack<std::unique_ptr<Node<T» *> stack{};
00207   stack.push(&root_);
00208
00209   while (!stack.empty())
00210   {
00211     auto *curNode = stack.top();
00212     auto *right = &curNode->get()->right;
00213     auto *left = &curNode->get()->left;
00214
00215     if ((nullptr == *right) && (nullptr == *left))
00216     {
00217       curNode->reset();
00218       stack.pop();
00219       continue;
00220     }
00221
00222     stack.push(right);
00223     stack.push(left);
00224   }
00225 }
00226
00227 template <std::floating_point T>
00228 void KdTree<T>::setNodeCapacity(std::size_t newCap)
00229 {
00230   nodeCapacity_ = newCap;
00231 }
00232
00233 // Capacity
00234 template <std::floating_point T>
00235 bool KdTree<T>::empty() const
00236 {
00237   return triangles_.empty();
00238 }
00239
00240 template <std::floating_point T>
00241 std::size_t KdTree<T>::size() const
00242 {
00243   return triangles_.size();
00244 }
00245
00246 template <std::floating_point T>
00247 std::size_t KdTree<T>::nodeCapacity() const
00248 {
00249   return nodeCapacity_;
00250 }
00251
00252 template <std::floating_point T>
00253 const Triangle<T> &KdTree<T>::triangleByIndex(Index index) const &
00254 {
00255   return triangles_[index];
00256 }
00257
00258 template <std::floating_point T>
00259 void KdTree<T>::dumpRecursive(std::ostream &ost) const
00260 {
00261   ost « "digraph kdtree {" « std::endl;
00262   if (root_)
00263     root_->dumpRecursive(ost);
00264   ost « "}" « std::endl;
00265 }
00266
00267 template <std::floating_point T>
00268 bool KdTree<T>::isOnPosSide(Axis axis, T separator, const Triangle<T> &tr)
00269 {
00270   return isOnSide(axis, separator, tr, std::greater<T>{});
00271 }
00272
00273 template <std::floating_point T>
00274 bool KdTree<T>::isOnNegSide(Axis axis, T separator, const Triangle<T> &tr)
00275 {
00276   return isOnSide(axis, separator, tr, std::less<T>{});
00277 }
00278
00279 template <std::floating_point T>
00280 bool KdTree<T>::isOnSide(Axis axis, T separator, const Triangle<T> &tr,
00281                          std::function<bool(T, T)> comparator)
00282 {
00283   if (Axis::NONE == axis)
00284     return false;
00285
00286   auto axisIdx = static_cast<size_t>(axis);
00287   return std::all_of(tr.begin(), tr.end(),
```

```
00288                     [&](auto &&v) { return comparator(v[axisIdx], separator); });
00289 }
00290
00291 template <std::floating_point T>
00292 void KdTree<T>::expandingInsert(const Triangle<T> &tr)
00293 {
00294   auto trianBB = tr.boundBox();
00295   auto index = triangles_.size();
00296   triangles_.push_back(tr);
00297
00298   for (auto axis : {Axis::X, Axis::Y, Axis::Z})
00299     tryExpandRight(axis, trianBB);
00300
00301   for (auto axis : {Axis::X, Axis::Y, Axis::Z})
00302     tryExpandLeft(axis, trianBB);
00303
00304   root_->indicies.push_back(index);
00305 }
00306
00307 template <std::floating_point T>
00308 void KdTree<T>::tryExpandRight(Axis axis, const BoundBox<T> &trianBB)
00309 {
00310   const auto &rootBB = root_->boundBox;
00311   if (trianBB.max(axis) <= rootBB.max(axis))
00312     return;
00313
00314   auto newRightBB = rootBB;
00315   newRightBB.min(axis) = rootBB.max(axis);
00316   newRightBB.max(axis) = trianBB.max(axis);
00317
00318   auto newRootBB = rootBB;
00319   newRootBB.max(axis) = newRightBB.max(axis);
00320
00321   std::unique_ptr<Node<T>> newRight{new Node<T>{T{}, Axis::NONE, newRightBB}};
00322   std::unique_ptr<Node<T>> newRoot{new Node<T>{rootBB.max(axis), axis, newRootBB}};
00323
00324   newRoot->right = std::move(newRight);
00325   newRoot->left = std::move(root_);
00326
00327   root_ = std::move(newRoot);
00328 }
00329
00330 template <std::floating_point T>
00331 void KdTree<T>::tryExpandLeft(Axis axis, const BoundBox<T> &trianBB)
00332 {
00333   const auto &rootBB = root_->boundBox;
00334   if (trianBB.min(axis) >= rootBB.min(axis))
00335     return;
00336
00337   BoundBox<T> newLeftBB = rootBB;
00338   newLeftBB.max(axis) = rootBB.min(axis);
00339   newLeftBB.min(axis) = trianBB.min(axis);
00340
00341   BoundBox<T> newRootBB = rootBB;
00342   newRootBB.min(axis) = newLeftBB.min(axis);
00343
00344   std::unique_ptr<Node<T>> newLeft{new Node<T>{T{}, Axis::NONE, newLeftBB}};
00345   std::unique_ptr<Node<T>> newRoot{new Node<T>{rootBB.min(axis), axis, newRootBB}};
00346
00347   newRoot->left = std::move(newLeft);
00348   newRoot->right = std::move(root_);
00349
00350   root_ = std::move(newRoot);
00351 }
00352
00353 template <std::floating_point T>
00354 void KdTree<T>::nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv)
00355 {
00356   auto curNode = node;
00357   while (true)
00358   {
00359     if (isOnPosSide(curNode->sepAxis, curNode->separator, tr))
00360       curNode = curNode->right.get();
00361     else if (isOnNegSide(curNode->sepAxis, curNode->separator, tr))
00362       curNode = curNode->left.get();
00363     else
00364       break;
00365   }
00366
00367   curNode->indicies.push_back(index);
00368   if (isDivisable(curNode) && (!isSubdiv))
00369     subdivide(curNode);
00370 }
00371
00372 template <std::floating_point T>
00373 bool KdTree<T>::isDivisable(const Node<T> *node)
00374 {
```

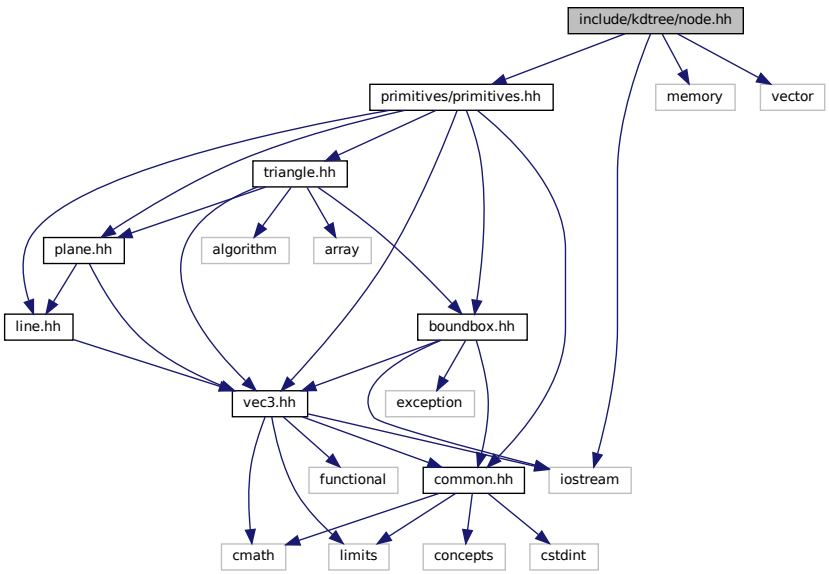
```
00375   return (node->indicies.size() > nodeCapacity_) && (node->sepAxis == Axis::NONE);
00376 }
00377
00378 template <std::floating_point T>
00379 void KdTree<T>::subdivide(Node<T> *node)
00380 {
00381   const auto &nodeBB = node->boundBox;
00382   auto axis = node->sepAxis = nodeBB.getMaxDim();
00383   auto sep = node->separator = nodeBB.min(axis) + (nodeBB.max(axis) - nodeBB.min(axis)) / 2;
00384
00385   auto newRightBB = nodeBB;
00386   auto newLeftBB = nodeBB;
00387
00388   newRightBB.min(axis) = newLeftBB.max(axis) = sep;
00389   node->right.reset(new Node<T>{T{}, Axis::NONE, newRightBB});
00390   node->left.reset(new Node<T>{T{}, Axis::NONE, newLeftBB});
00391
00392   auto indicies = node->indicies;
00393   node->indicies.clear();
00394
00395   for (auto index : indicies)
00396     nonExpandingInsert(node, triangles_[index], index, /* isSubdiv = */ true);
00397 }
00398
00399 //================================================================================================
00400 //                              KdTree::ContainerPtr definitions
00401 //================================================================================================
00402
00403 template <std::floating_point T>
00404 const Container<T> *KdTree<T>::ContainerPtr::operator->() const
00405 {
00406   return &cont;
00407 }
00408
00409 //================================================================================================
00410 //                              KdTree::ConstIterator definitions
00411 //================================================================================================
00412
00413 template <std::floating_point T>
00414 KdTree<T>::ConstIterator::ConstIterator(const KdTree<T> *tree, const Node<T> *node)
00415   : tree_(tree), node_(node), fifo_({node})
00416 {}
00417
00418 template <std::floating_point T>
00419 typename KdTree<T>::ConstIterator &KdTree<T>::ConstIterator::operator++()
00420 {
00421   if (0 == fifo_.size())
00422     return *this;
00423
00424   auto fifoEntry = fifo_.front();
00425   fifo_.pop();
00426
00427   if (Axis::NONE != fifoEntry->sepAxis)
00428   {
00429     if (nullptr != fifoEntry->left)
00430       fifo_.push(fifoEntry->left.get());
00431     if (nullptr != fifoEntry->right)
00432       fifo_.push(fifoEntry->right.get());
00433   }
00434
00435   node_ = (0 == fifo_.size()) ? nullptr : fifo_.front();
00436   return *this;
00437 }
00438
00439 template <std::floating_point T>
00440 typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::operator++(int)
00441 {
00442   auto tmp = *this;
00443   operator++();
00444   return tmp;
00445 }
00446
00447 template <std::floating_point T>
00448 typename KdTree<T>::ConstIterator::reference KdTree<T>::ConstIterator::operator*() const
00449 {
00450   return Container<T>{tree_, node_};
00451 }
00452
00453 template <std::floating_point T>
00454 typename KdTree<T>::ConstIterator::pointer KdTree<T>::ConstIterator::operator->() const
00455 {
00456   return ContainerPtr{{tree_, node_}};
00457 }
00458
00459 template <std::floating_point T>
00460 bool KdTree<T>::ConstIterator::operator==(const KdTree<T>::ConstIterator &lhs) const
00461 {
```

```
00462   return (tree_ == lhs.tree_) && (node_ == lhs.node_);
00463 }
00464
00465 template <std::floating_point T>
00466 bool KdTree<T>::ConstIterator::operator!=(const KdTree<T>::ConstIterator &lhs) const
00467 {
00468   return !operator==(lhs);
00469 }
00470
00471 template <std::floating_point T>
00472 typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::beginFrom(
00473   const typename KdTree<T>::ConstIterator &iter)
00474 {
00475   return ConstIterator{iter.tree_, iter.node_};
00476 }
00477
00478 } // namespace geom::kdtree
00479
00480 #endif // __INCLUDE_KDTREE_KDTREE_HH__
```

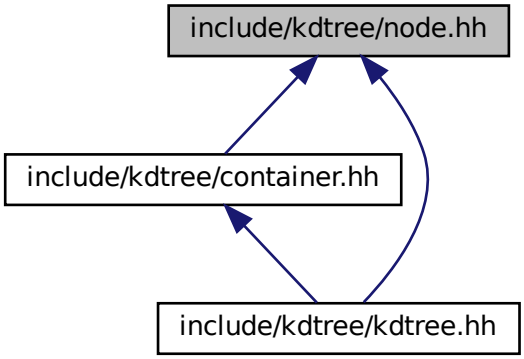
6.11 include/kdtree/node.hh File Reference

```
#include <iostream>
#include <memory>
#include <vector>
#include "primitives/primitives.hh"
```

Include dependency graph for node.hh:



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



Classes

- struct geom::kdtree::Node< T >

Namespaces

- geom

  line.hh Line class implementation
- geom::kdtree

Typedefs

- using geom::kdtree::Index = std::size_t

6.12 node.hh

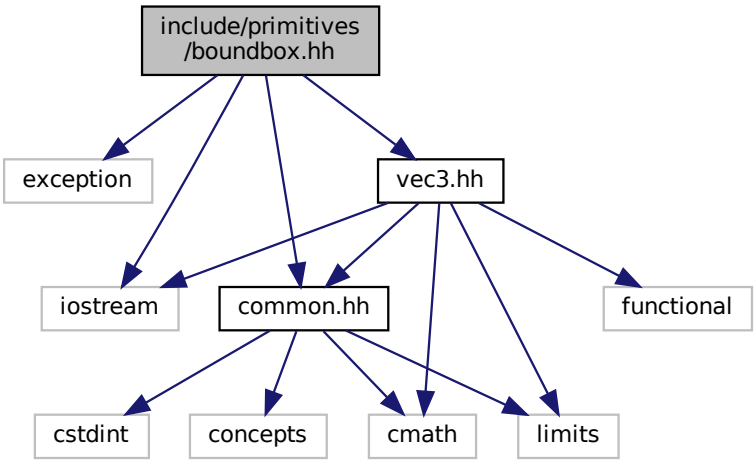
```
00001 #ifndef __INCLUDE_KDTREE_NODE_HH__
00002 #define __INCLUDE_KDTREE_NODE_HH__
00003
00004 #include <iostream>
00005 #include <memory>
00006 #include <vector>
00007
00008 #include "primitives/primitives.hh"
00009
00010 namespace geom::kdtree
00011 {
00012
00013 using Index = std::size_t;
00014
00015 template <std::floating_point T>
00016 struct Node final
00017 {
00018   T separator{};            // separator's coordinate on separation axis
00019   Axis sepAxis{Axis::NONE}; // separation axis
00020   BoundBox<T> boundBox{};
00021   std::vector<Index> indicies{};
00022
```

```
00023   std::unique_ptr<Node> left{nullptr};
00024   std::unique_ptr<Node> right{nullptr};
00025
00026   using IndexIterator = std::vector<Index>::iterator;
00027   using IndexConstIterator = std::vector<Index>::const_iterator;
00028
00029   void dumpRecursive(std::ostream &ost) const;
00030 };
00031
00032 template <std::floating_point T>
00033 void Node<T>::dumpRecursive(std::ostream &ost) const
00034 {
00035   ost « reinterpret_cast<std::uintptr_t>(this)
00036       « " [shape=box,label=\"axis: " « static_cast<int>(sepAxis) « ",\\n"
00037       « boundBox « ",\\nvec: {";
00038
00039   for (auto elem : indicies)
00040     ost « elem « " ";
00041
00042   ost « "}\"];" « std::endl;
00043
00044   if (left)
00045   {
00046     left->dumpRecursive(ost);
00047     ost « reinterpret_cast<std::uintptr_t>(this) « " -> "
00048         « reinterpret_cast<std::uintptr_t>(left.get()) « " [label=\"L\"];" « std::endl;
00049   }
00050   if (right)
00051   {
00052     right->dumpRecursive(ost);
00053     ost « reinterpret_cast<std::uintptr_t>(this) « " -> "
00054         « reinterpret_cast<std::uintptr_t>(right.get()) « " [label=\"R\"];" « std::endl;
00055   }
00056 }
00057
00058 } // namespace geom::kdtree
00059
00060 #endif // __INCLUDE_KDTREE_NODE_HH__
```

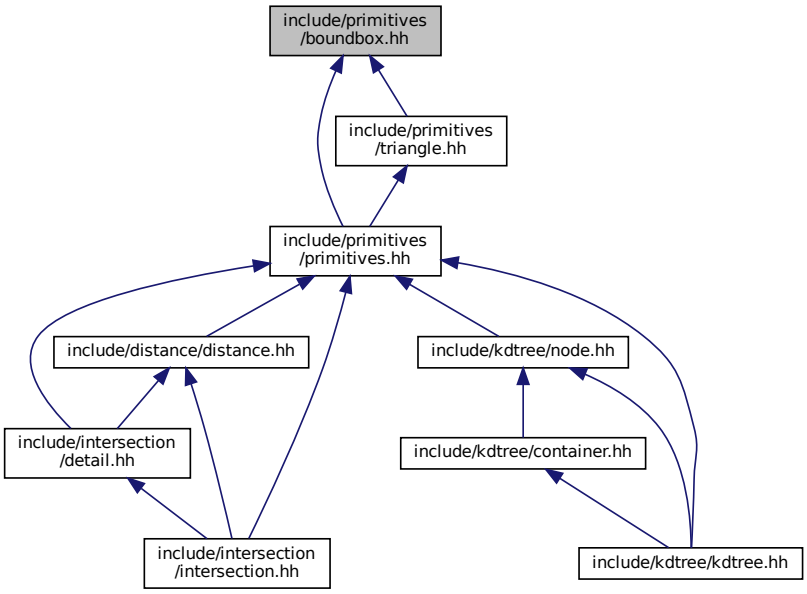
6.13 include/primitives/boundbox.hh File Reference

```
#include <exception>
#include <iostream>
#include "common.hh"
#include "vec3.hh"
```

Include dependency graph for boundbox.hh:



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



Classes

- struct geom::BoundBox< T >

Namespaces

- geom

  line.hh Line class implementation

Macros

- #define BBFILL(minmax)

Functions

- template<std::floating_point T>
  bool geom::operator== (const BoundBox< T > &lhs, const BoundBox< T > &rhs)
- template<std::floating_point T>
  std::ostream & geom::operator<< (std::ostream &ost, const BoundBox< T > &bb)

6.13.1 Macro Definition Documentation

6.13.1.1 BBFILL

```
#define BBFILL(
               minmax )
```

Value:

```
  do                                                                                                  \
  {                                                                                                   \
    switch (axis)                                                                                     \
    {                                                                                                 \
    case Axis::X:                                                                                     \
      return minmax##X;                                                                               \
    case Axis::Y:                                                                                     \
      return minmax##Y;                                                                               \
    case Axis::Z:                                                                                     \
      return minmax##Z;                                                                               \
    case Axis::NONE:                                                                                  \
    default:                                                                                          \
      throw std::logic_error("BoundBox<T>::" #minmax " (): Wrong input axis");                        \
    }                                                                                                 \
  } while (false)
```

Definition at line 46 of file boundbox.hh.

6.14 boundbox.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
00002 #define __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
00003
00004 #include <exception>
00005 #include <iostream>
00006
00007 #include "common.hh"
00008 #include "vec3.hh"
00009
00010 namespace geom
00011 {
00012
00013 template <std::floating_point T>
00014 struct BoundBox
00015 {
00016   T minX{};
00017   T maxX{};
00018
00019   T minY{};
00020   T maxY{};
00021
00022   T minZ{};
00023   T maxZ{};
00024
00025   bool belongsTo(const BoundBox<T> &bb);
00026
00027   T &min(Axis axis) &;
00028   T &max(Axis axis) &;
00029
00030   T min(Axis axis) &&;
00031   T max(Axis axis) &&;
00032
00033   T min(Axis axis) const &;
00034   T max(Axis axis) const &;
00035
00036   Axis getMaxDim() const;
00037 };
00038
00039 template <std::floating_point T>
00040 bool BoundBox<T>::belongsTo(const BoundBox<T> &bb)
00041 {
00042   return (minX >= bb.minX) && (minY >= bb.minY) && (minZ >= bb.minZ) && (maxX <= bb.maxX) &&
00043          (maxY <= bb.maxY) && (maxZ <= bb.maxZ);
00044 }
00045
00046 #define BBFILL(minmax)                                                                        \
00047   do                                                                                          \
00048   {                                                                                           \
00049     switch (axis)                                                                             \
00050     {                                                                                         \
00051     case Axis::X:                                                                             \
00052       return minmax##X;                                                                       \
00053     case Axis::Y:                                                                             \
```

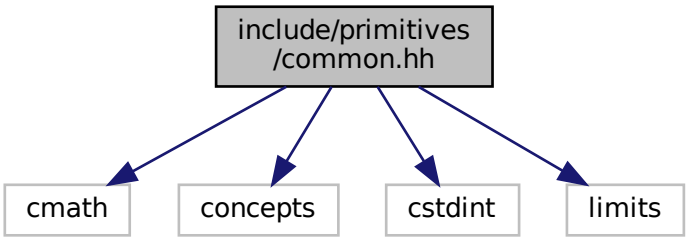
```
00054       return minmax##Y;                                                                   \
00055     case Axis::Z:                                                                         \
00056       return minmax##Z;                                                                   \
00057     case Axis::NONE:                                                                      \
00058     default:                                                                              \
00059       throw std::logic_error("BoundBox<T>::" #minmax " (): Wrong input axis");             \
00060     }                                                                                     \
00061   } while (false)
00062
00063 template <std::floating_point T>
00064 T &BoundBox<T>::min(Axis axis) &
00065 {
00066   BBFILL(min);
00067 }
00068
00069 template <std::floating_point T>
00070 T &BoundBox<T>::max(Axis axis) &
00071 {
00072   BBFILL(max);
00073 }
00074
00075 template <std::floating_point T>
00076 T BoundBox<T>::min(Axis axis) &&
00077 {
00078   BBFILL(min);
00079 }
00080
00081 template <std::floating_point T>
00082 T BoundBox<T>::max(Axis axis) &&
00083 {
00084   BBFILL(max);
00085 }
00086
00087 template <std::floating_point T>
00088 T BoundBox<T>::min(Axis axis) const &
00089 {
00090   BBFILL(min);
00091 }
00092
00093 template <std::floating_point T>
00094 T BoundBox<T>::max(Axis axis) const &
00095 {
00096   BBFILL(max);
00097 }
00098
00099 #undef BBFILL
00100
00101 template <std::floating_point T>
00102 Axis BoundBox<T>::getMaxDim() const
00103 {
00104   return std::max({Axis::X, Axis::Y, Axis::Z}, [this](const auto &lhs, const auto &rhs) {
00105     return (this->max(lhs) - this->min(lhs)) < (this->max(rhs) - this->min(rhs));
00106   });
00107 }
00108
00109 template <std::floating_point T>
00110 bool operator==(const BoundBox<T> &lhs, const BoundBox<T> &rhs)
00111 {
00112   return ThresComp<T>::isEqual(lhs.minX, rhs.minX) && ThresComp<T>::isEqual(lhs.maxX, rhs.maxX) &&
00113          ThresComp<T>::isEqual(lhs.minY, rhs.minY) && ThresComp<T>::isEqual(lhs.maxY, rhs.maxY) &&
00114          ThresComp<T>::isEqual(lhs.minZ, rhs.minZ) && ThresComp<T>::isEqual(lhs.maxY, rhs.maxY);
00115 }
00116
00117 template <std::floating_point T>
00118 std::ostream &operator«(std::ostream &ost, const BoundBox<T> &bb)
00119 {
00120   ost « "BB: {\\n";
00121   ost « "  x: [" « bb.minX « "; " « bb.maxX « "],\\n";
00122   ost « "  y: [" « bb.minY « "; " « bb.maxY « "],\\n";
00123   ost « "  z: [" « bb.minZ « "; " « bb.maxZ « "]\\n}";
00124   return ost;
00125 }
00126
00127 } // namespace geom
00128
00129 #endif // __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
```

6.15 include/primitives/common.hh File Reference

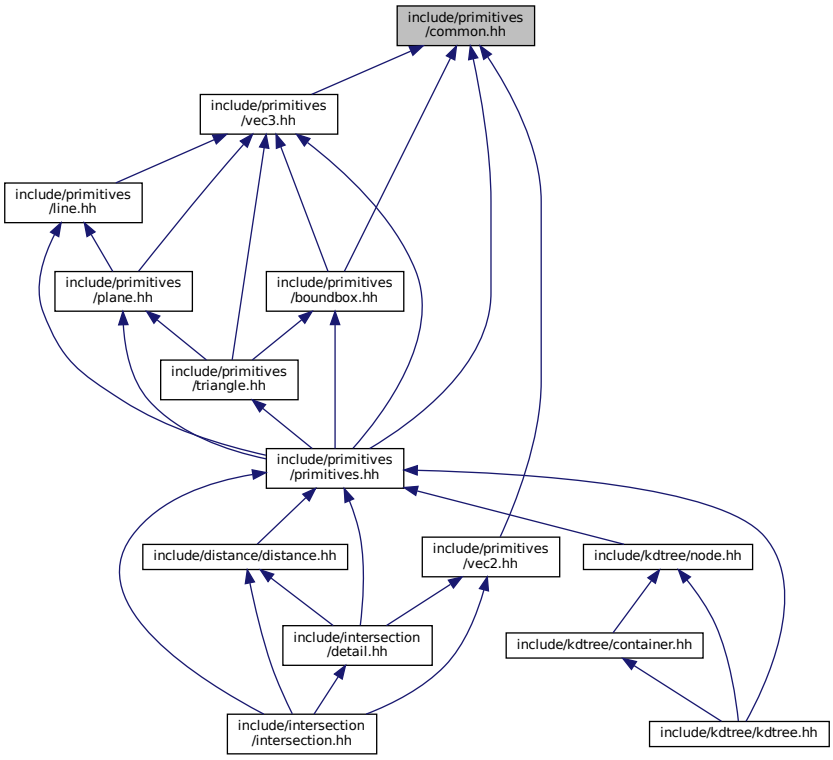
```
#include <cmath>
#include <concepts>
```

```
#include <cstdint>
#include <limits>
```

Include dependency graph for common.hh:



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



Classes

- class geom::ThresComp< T >

Namespaces

- geom

  line.hh Line class implementation

Enumerations

- enum geom::Axis : std::int8_t { geom::Axis::X = 0, geom::Axis::Y = 1, geom::Axis::Z = 2, geom::Axis::NONE }

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.

6.16 common.hh

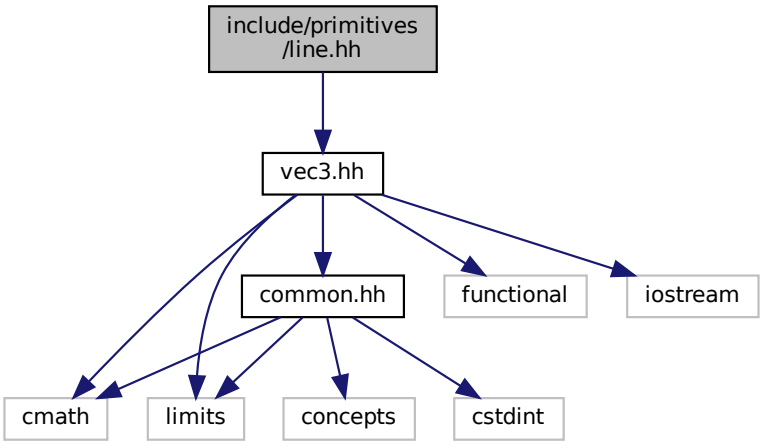
```
00001 #ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
00002 #define __INCLUDE_PRIMITIVES_COMMON_HH__
00003
00004 #include <cmath>
00005 #include <concepts>
00006 #include <cstdint>
00007 #include <limits>
00008 namespace geom
00009 {
00010 /**
00011  * @concept Number
00012  * @brief Useful concept which represents floating point and integral types
00013  *
00014  * @tparam T
00015  */
00016 template <class T>
00017 concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>;
00018
00019 enum class Axis : std::int8_t
00020 {
00021   X = 0,
00022   Y = 1,
00023   Z = 2,
00024   NONE
00025 };
00026
00027 template <Number T>
00028 class ThresComp final
00029 {
00030 private:
00031   static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00032
00033 public:
00034   ThresComp() = delete;
00035
00036   static void setThreshold(T thres) requires std::is_floating_point_v<T>
00037   {
00038     threshold_ = thres;
00039   }
00040
00041   static T getThreshold() requires std::is_floating_point_v<T>
00042   {
00043     return threshold_;
00044   }
00045
00046   static void scaleThreshold(T factor) requires std::is_floating_point_v<T>
00047   {
00048     threshold_ *= factor;
00049   }
00050
00051   static void resetThreshold() requires std::is_floating_point_v<T>
00052   {
00053     threshold_ = std::numeric_limits<T>::epsilon();
00054   }
00055
```

```
00056   static bool isEqual(T lhs, T rhs)
00057   {
00058     if constexpr (std::is_floating_point_v<T>)
00059       return std::abs(rhs - lhs) < threshold_;
00060     else
00061       return lhs == rhs;
00062   }
00063
00064   static bool isZero(T num)
00065   {
00066     return isEqual(num, T{});
00067   }
00068 };
00069
00070 } // namespace geom
00071
00072 #endif // __INCLUDE_PRIMITIVES_COMMON_HH__
```

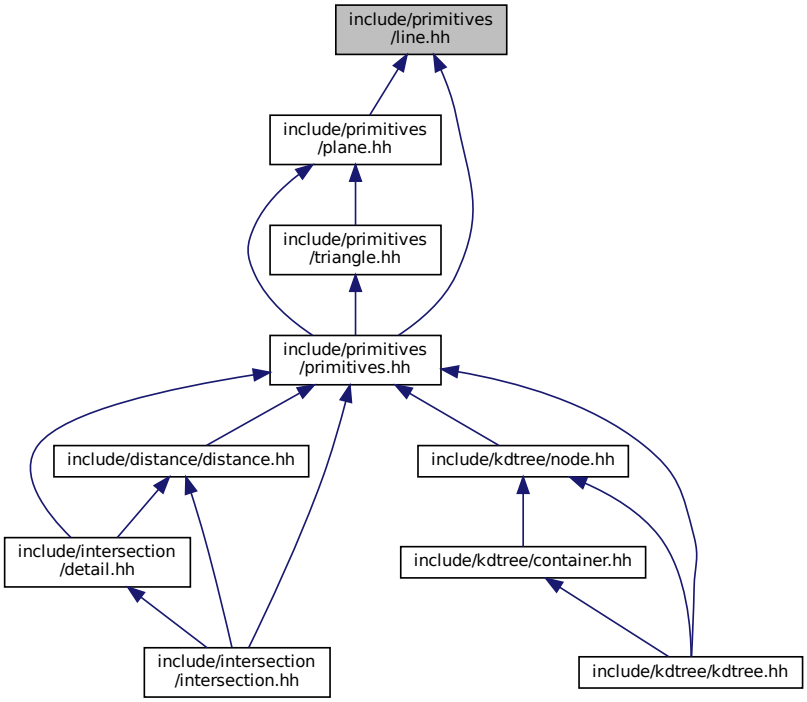
6.17 include/primitives/line.hh File Reference

`#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.

6.18 line.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_LINE_HH__
00002 #define __INCLUDE_PRIMITIVES_LINE_HH__
00003
00004 #include "vec3.hh"
00005
00006 /**
00007  * @brief line.hh
00008  * Line class implementation
00009  */
00010
00011 namespace geom
00012 {
00013
00014 /**
00015  * @class Line
00016  * @brief Line class implementation
00017  *
00018  * @tparam T - floating point type of coordinates
00019  */
00020 template <std::floating_point T>
00021 class Line final
00022 {
00023 private:
00024   /**
00025    * @brief Origin and direction vectors
00026    */
00027   Vec3<T> org_{}, dir_{};
00028
00029 public:
00030   /**
00031    * @brief Construct a new Line object
00032    *
00033    * @param[in] org origin vector
00034    * @param[in] dir direction vector
00035    */
00036   Line(const Vec3<T> &org, const Vec3<T> &dir);
00037
00038   /**
00039    * @brief Getter for origin vector
00040    *
00041    * @return const Vec3<T>& const reference to origin vector
00042    */
00043   const Vec3<T> &org() const &;
00044
00045   /**
00046    * @brief Getter for direction vector
00047    *
00048    * @return const Vec3<T>& const reference to direction vector
00049    */
00050   const Vec3<T> &dir() const &;
00051
00052   /**
00053    * @brief Getter for origin vector
00054    *
00055    * @return Vec3<T>&& reference to origin vector
00056    */
00057   Vec3<T> &&org() &&;
00058
00059   /**
00060    * @brief Getter for direction vector
00061    *
00062    * @return Vec3<T>&& reference to direction vector
00063    */
00064   Vec3<T> &&dir() &&;
00065
00066   /**
00067    * @brief Get point on line by parameter t
00068    *
00069    * @tparam nType numeric type
00070    * @param[in] t point paramater from line's equation
00071    * @return Vec3<T> Point related to parameter
00072    */
00073   template <Number nType>
00074   Vec3<T> getPoint(nType t) const;
00075
00076   /**
00077    * @brief Checks is point belongs to line
00078    *
00079    * @param[in] point const reference to point vector
00080    * @return true if point belongs to line
00081    * @return false if point doesn't belong to line
00082    */
00083   bool belongs(const Vec3<T> &point) const;
00084
00085   /**
```

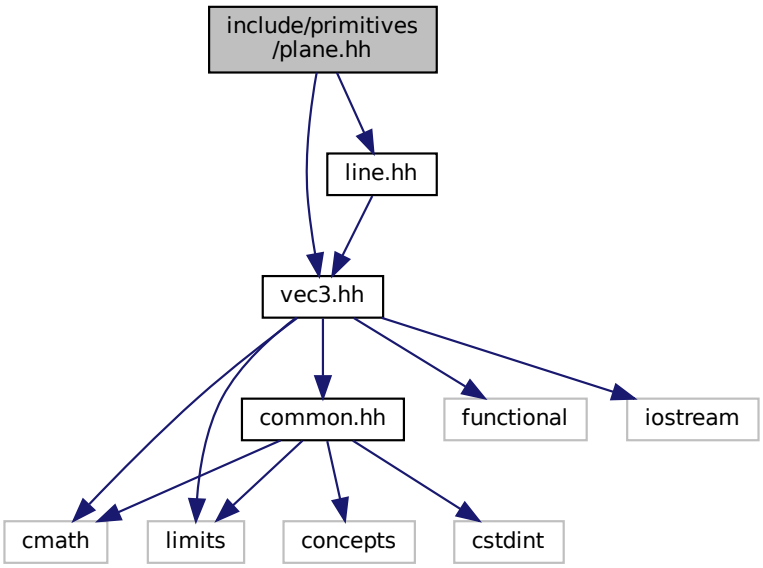
```
00086    * @brief Checks is *this equals to another line
00087    *
00088    * @param[in] line const reference to another line
00089    * @return true if lines are equal
00090    * @return false if lines are not equal
00091    */
00092   bool isEqual(const Line &line) const;
00093
00094   /**
00095    * @brief Checks is *this parallel to another line
00096    * @note Assumes equal lines as parallel
00097    * @param[in] line const reference to another line
00098    * @return true if lines are parallel
00099    * @return false if lines are not parallel
00100    */
00101   bool isPar(const Line &line) const;
00102
00103   /**
00104    * @brief Checks is *this is skew with another line
00105    *
00106    * @param[in] line const reference to another line
00107    * @return true if lines are skew
00108    * @return false if lines are not skew
00109    */
00110   bool isSkew(const Line<T> &line) const;
00111
00112   /**
00113    * @brief Get line by 2 points
00114    *
00115    * @param[in] p1 1st point
00116    * @param[in] p2 2nd point
00117    * @return Line passing through two points
00118    */
00119   static Line getBy2Points(const Vec3<T> &p1, const Vec3<T> &p2);
00120 };
00121
00122 /**
00123  * @brief Line print operator
00124  *
00125  * @tparam T - floating point type of coordinates
00126  * @param[in, out] ost output stream
00127  * @param[in] line Line to print
00128  * @return std::ostream& modified ostream instance
00129  */
00130 template <std::floating_point T>
00131 std::ostream &operator«(std::ostream &ost, const Line<T> &line)
00132 {
00133   ost « line.org() « " + " « line.dir() « " * t";
00134   return ost;
00135 }
00136
00137 /**
00138  * @brief Line equality operator
00139  *
00140  * @tparam T - floating point type of coordinates
00141  * @param[in] lhs 1st line
00142  * @param[in] rhs 2nd line
00143  * @return true if lines are equal
00144  * @return false if lines are not equal
00145  */
00146 template <std::floating_point T>
00147 bool operator==(const Line<T> &lhs, const Line<T> &rhs)
00148 {
00149   return lhs.isEqual(rhs);
00150 }
00151
00152 template <std::floating_point T>
00153 Line<T>::Line(const Vec3<T> &org, const Vec3<T> &dir) : org_{org}, dir_{dir}
00154 {
00155   if (dir_ == Vec3<T>{0})
00156     throw std::logic_error{"Direction vector equals zero."};
00157 }
00158
00159 template <std::floating_point T>
00160 const Vec3<T> &Line<T>::org() const &
00161 {
00162   return org_;
00163 }
00164
00165 template <std::floating_point T>
00166 const Vec3<T> &Line<T>::dir() const &
00167 {
00168   return dir_;
00169 }
00170
00171 template <std::floating_point T>
00172 Vec3<T> &&Line<T>::org() &&
```

```
00173 {
00174   return std::move(org_);
00175 }
00176
00177 template <std::floating_point T>
00178 Vec3<T> &&Line<T>::dir() &&
00179 {
00180   return std::move(dir_);
00181 }
00182
00183 template <std::floating_point T>
00184 template <Number nType>
00185 Vec3<T> Line<T>::getPoint(nType t) const
00186 {
00187   return org_ + dir_ * t;
00188 }
00189
00190 template <std::floating_point T>
00191 bool Line<T>::belongs(const Vec3<T> &point) const
00192 {
00193   return dir_.cross(point - org_) == Vec3<T>{0};
00194 }
00195
00196 template <std::floating_point T>
00197 bool Line<T>::isEqual(const Line<T> &line) const
00198 {
00199   return belongs(line.org_) && dir_.isPar(line.dir_);
00200 }
00201
00202 template <std::floating_point T>
00203 bool Line<T>::isPar(const Line<T> &line) const
00204 {
00205   return dir_.isPar(line.dir_);
00206 }
00207
00208 template <std::floating_point T>
00209 bool Line<T>::isSkew(const Line<T> &line) const
00210 {
00211   auto res = triple(line.org_ - org_, dir_, line.dir_);
00212   return !ThresComp<T>::isZero(res);
00213 }
00214
00215 template <std::floating_point T>
00216 Line<T> Line<T>::getBy2Points(const Vec3<T> &p1, const Vec3<T> &p2)
00217 {
00218   return Line<T>{p1, p2 - p1};
00219 }
00220
00221 } // namespace geom
00222
00223 #endif // __INCLUDE_PRIMITIVES_LINE_HH__
```

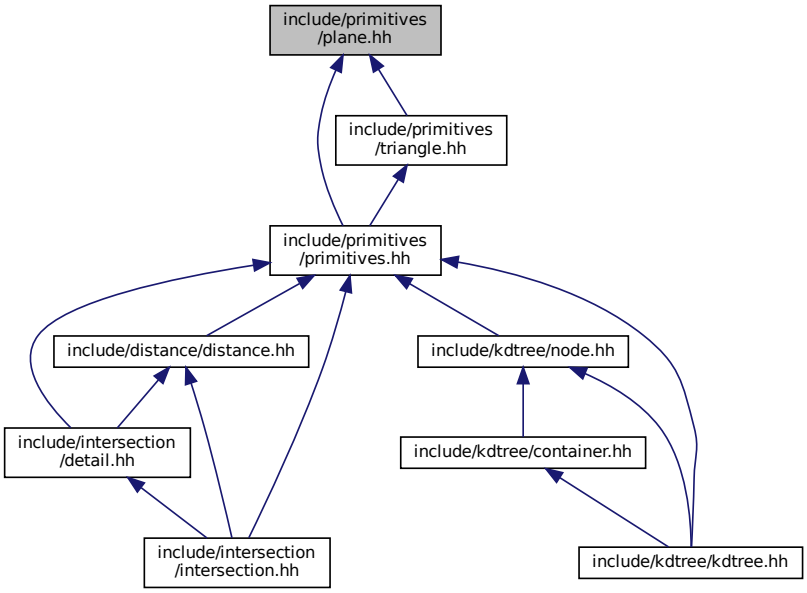
6.19 include/primitives/plane.hh File Reference

```
#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.

6.20 plane.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_PLANE_HH__
00002 #define __INCLUDE_PRIMITIVES_PLANE_HH__
00003
00004 #include "line.hh"
00005 #include "vec3.hh"
00006
00007 /**
00008  * @brief
00009  * Plane class implementation
00010  */
00011
00012 namespace geom
00013 {
00014
00015 /**
00016  * @class Plane
00017  * @brief Plane class realization
00018  *
00019  * @tparam T - floating point type of coordinates
00020  */
00021 template <std::floating_point T>
00022 class Plane final
00023 {
00024 private:
00025   /**
00026    * @brief Normal vector, length equals to 1
00027    */
00028   Vec3<T> norm_{};
00029
00030   /**
00031    * @brief Distance from zero to plane
00032    */
00033   T dist_{};
00034
00035   /**
00036    * @brief Construct a new Plane object from normal vector and distance
00037    *
00038    * @param[in] norm normal vector
00039    * @param[in] dist distance from plane to zero
00040    */
00041   Plane(const Vec3<T> &norm, T dist);
00042
00043 public:
00044   /**
00045    * @brief Getter for distance
00046    *
00047    * @return T value of distance
00048    */
00049   T dist() const;
00050
00051   /**
00052    * @brief Getter for normal vector
00053    *
00054    * @return const Vec3<T>& const reference to normal vector
00055    */
00056   const Vec3<T> &norm() const &;
00057
00058   /**
```

```
00059     * @brief Getter for normal vector
00060     *
00061     * @return Vec3<T>&& reference to normal vector
00062     */
00063    Vec3<T> &&norm() &&;
00064
00065    /**
00066     * @brief Checks if point belongs to plane
00067     *
00068     * @param[in] point const referene to point vector
00069     * @return true if point belongs to plane
00070     * @return false if point doesn't belong to plane
00071     */
00072    bool belongs(const Vec3<T> &point) const;
00073
00074    /**
00075     * @brief Checks if line belongs to plane
00076     *
00077     * @param[in] line const referene to line
00078     * @return true if line belongs to plane
00079     * @return false if line doesn't belong to plane
00080     */
00081    bool belongs(const Line<T> &line) const;
00082
00083    /**
00084     * @brief Checks is *this equals to another plane
00085     *
00086     * @param[in] rhs const reference to another plane
00087     * @return true if planes are equal
00088     * @return false if planes are not equal
00089     */
00090    bool isEqual(const Plane &rhs) const;
00091
00092    /**
00093     * @brief Checks is *this is parallel to another plane
00094     *
00095     * @param[in] rhs const reference to another plane
00096     * @return true if planes are parallel
00097     * @return false if planes are not parallel
00098     */
00099    bool isPar(const Plane &rhs) const;
00100
00101    /**
00102     * @brief Get plane by 3 points
00103     *
00104     * @param[in] pt1 1st point
00105     * @param[in] pt2 2nd point
00106     * @param[in] pt3 3rd point
00107     * @return Plane passing through three points
00108     */
00109    static Plane getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3);
00110
00111    /**
00112     * @brief Get plane from parametric plane equation
00113     *
00114     * @param[in] org origin vector
00115     * @param[in] dir1 1st direction vector
00116     * @param[in] dir2 2nd direction vector
00117     * @return Plane
00118     */
00119    static Plane getParametric(const Vec3<T> &org, const Vec3<T> &dir1, const Vec3<T> &dir2);
00120
00121    /**
00122     * @brief Get plane from normal point plane equation
00123     *
00124     * @param[in] norm normal vector
00125     * @param[in] point point lying on the plane
00126     * @return Plane
00127     */
00128    static Plane getNormalPoint(const Vec3<T> &norm, const Vec3<T> &point);
00129
00130    /**
00131     * @brief Get plane form normal const plane equation
00132     *
00133     * @param[in] norm normal vector
00134     * @param[in] constant distance
00135     * @return Plane
00136     */
00137    static Plane getNormalDist(const Vec3<T> &norm, T constant);
00138 };
00139
00140 /**
00141  * @brief Plane equality operator
00142  *
00143  * @tparam T - floating point type of coordinates
00144  * @param[in] lhs 1st plane
00145  * @param[in] rhs 2nd plane
```

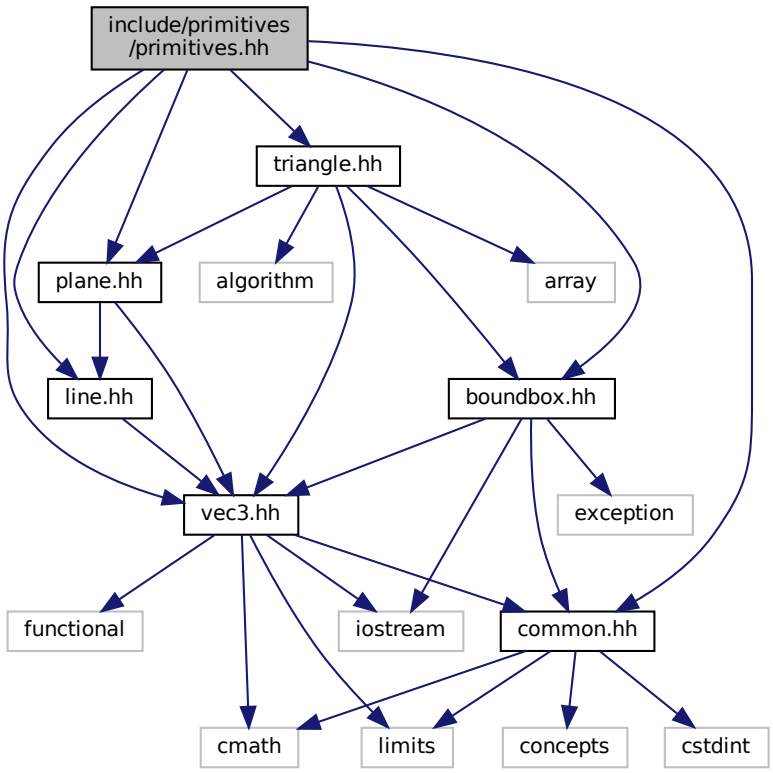
```
00146  * @return true if planes are equal
00147  * @return false if planes are not equal
00148  */
00149 template <std::floating_point T>
00150 bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
00151 {
00152   return lhs.isEqual(rhs);
00153 }
00154
00155 /**
00156  * @brief Plane print operator
00157  *
00158  * @tparam T - floating point type of coordinates
00159  * @param[in, out] ost output stream
00160  * @param[in] pl plane to print
00161  * @return std::ostream& modified ostream instance
00162  */
00163 template <std::floating_point T>
00164 std::ostream &operator«(std::ostream &ost, const Plane<T> &pl)
00165 {
00166   ost « pl.norm() « " * X = " « pl.dist();
00167   return ost;
00168 }
00169
00170 template <std::floating_point T>
00171 Plane<T>::Plane(const Vec3<T> &norm, T dist) : norm_(norm), dist_(dist)
00172 {
00173   if (norm == Vec3<T>{0})
00174     throw std::logic_error{"normal vector equals to zero"};
00175 }
00176
00177 template <std::floating_point T>
00178 T Plane<T>::dist() const
00179 {
00180   return dist_;
00181 }
00182
00183 template <std::floating_point T>
00184 const Vec3<T> &Plane<T>::norm() const &
00185 {
00186   return norm_;
00187 }
00188
00189 template <std::floating_point T>
00190 Vec3<T> &&Plane<T>::norm() &&
00191 {
00192   return std::move(norm_);
00193 }
00194
00195 template <std::floating_point T>
00196 bool Plane<T>::belongs(const Vec3<T> &pt) const
00197 {
00198   return ThresComp<T>::isEqual(norm_.dot(pt), dist_);
00199 }
00200
00201 template <std::floating_point T>
00202 bool Plane<T>::belongs(const Line<T> &line) const
00203 {
00204   return norm_.isPerp(line.dir()) && belongs(line.org());
00205 }
00206
00207 template <std::floating_point T>
00208 bool Plane<T>::isEqual(const Plane &rhs) const
00209 {
00210   return (norm_ * dist_ == rhs.norm_ * rhs.dist_) && (norm_.isPar(rhs.norm_));
00211 }
00212
00213 template <std::floating_point T>
00214 bool Plane<T>::isPar(const Plane &rhs) const
00215 {
00216   return norm_.isPar(rhs.norm_);
00217 }
00218
00219 template <std::floating_point T>
00220 Plane<T> Plane<T>::getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3)
00221 {
00222   return getParametric(pt1, pt2 - pt1, pt3 - pt1);
00223 }
00224
00225 template <std::floating_point T>
00226 Plane<T> Plane<T>::getParametric(const Vec3<T> &org, const Vec3<T> &dir1, const Vec3<T> &dir2)
00227 {
00228   auto norm = dir1.cross(dir2);
00229   return getNormalPoint(norm, org);
00230 }
00231
00232 template <std::floating_point T>
```

```
00233 Plane<T> Plane<T>::getNormalPoint(const Vec3<T> &norm, const Vec3<T> &pt)
00234 {
00235   auto normalized = norm.normalized();
00236   return Plane{normalized, normalized.dot(pt)};
00237 }
00238
00239 template <std::floating_point T>
00240 Plane<T> Plane<T>::getNormalDist(const Vec3<T> &norm, T dist)
00241 {
00242   auto normalized = norm.normalized();
00243   return Plane{normalized, dist};
00244 }
00245
00246 } // namespace geom
00247
00248 #endif // __INCLUDE_PRIMITIVES_PLANE_HH__
```

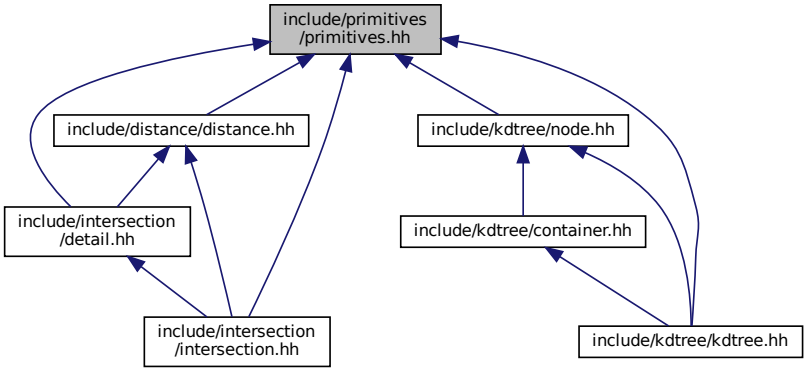
6.21 include/primitives/primitives.hh File Reference

```
#include "boundbox.hh"
#include "common.hh"
#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:



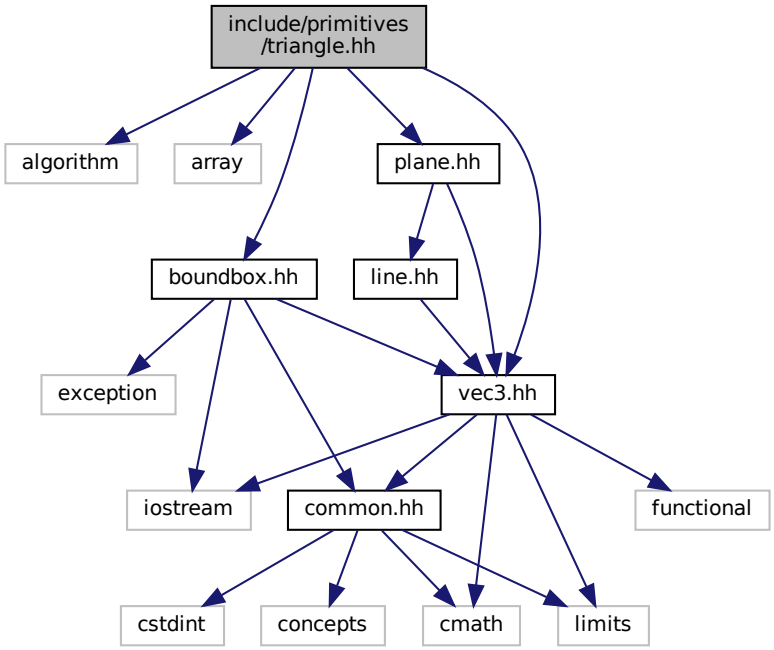
6.22 primitives.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00002 #define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00003
00004 #include "boundbox.hh"
00005 #include "common.hh"
00006 #include "line.hh"
00007 #include "plane.hh"
00008 #include "triangle.hh"
00009 #include "vec3.hh"
00010
00011 #endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

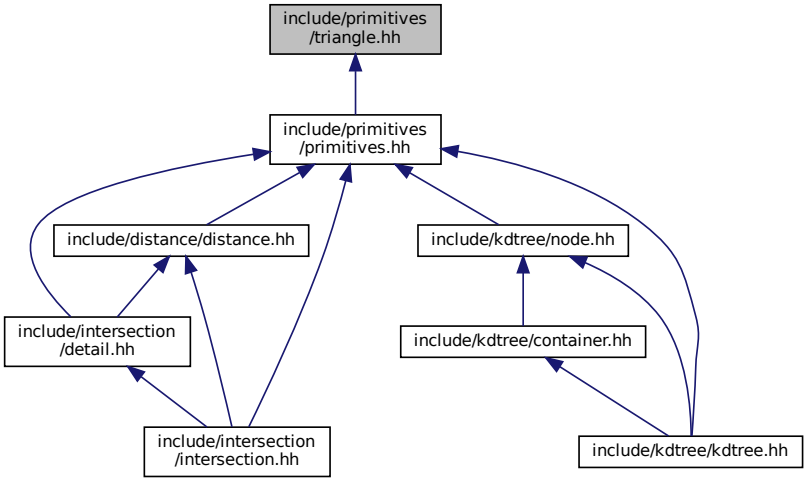
6.23 include/primitives/triangle.hh File Reference

```
#include <algorithm>
#include <array>
#include "boundbox.hh"
#include "plane.hh"
#include "vec3.hh"
```

Include dependency graph for triangle.hh:



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



Classes

- class geom::Triangle< T >

  Triangle class implementation.

Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Triangle print operator.
- template<std::floating_point T>
  std::istream & geom::operator>> (std::istream &ist, Triangle< T > &tr)

6.24 triangle.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00002 #define __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00003
00004 #include <algorithm>
00005 #include <array>
00006
00007 #include "boundbox.hh"
00008 #include "plane.hh"
00009 #include "vec3.hh"
00010
00011 /**
00012  * @brief triangle.hh
00013  * Triangle class implementation
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Triangle
00021  * @brief Triangle class implementation
00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025 template <std::floating_point T>
00026 class Triangle final
00027 {
00028 private:
00029   /**
00030    * @brief Vertices of triangle
00031    */
00032   std::array<Vec3<T>, 3> vertices_;
00033
00034 public:
00035   using Iterator = typename std::array<Vec3<T>, 3>::iterator;
00036   using ConstIterator = typename std::array<Vec3<T>, 3>::const_iterator;
00037
00038   /**
00039    * @brief Construct a new Triangle object
00040    */
00041   Triangle();
00042
00043   /**
00044    * @brief Construct a new Triangle object from 3 points
00045    *
00046    * @param[in] p1 1st point
00047    * @param[in] p2 2nd point
00048    * @param[in] p3 3rd point
00049    */
00050   Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3);
00051
00052   /**
00053    * @brief Overloaded operator[] to get access to vertices
00054    *
00055    * @param[in] idx index of vertex
00056    * @return const Vec3<T>& const reference to vertex
00057    */
00058   const Vec3<T> &operator[](std::size_t idx) const &;
00059
00060   /**
```

```
00061    * @brief Overloaded operator[] to get access to vertices
00062    *
00063    * @param[in] idx index of vertex
00064    * @return Vec3<T>&& reference to vertex
00065    */
00066   Vec3<T> &&operator[](std::size_t idx) &&;
00067
00068   /**
00069    * @brief Overloaded operator[] to get access to vertices
00070    *
00071    * @param[in] idx index of vertex
00072    * @return Vec3<T>& reference to vertex
00073    */
00074   Vec3<T> &operator[](std::size_t idx) &;
00075
00076   /**
00077    * @brief Get begin iterator
00078    * @return Iterator
00079    */
00080   Iterator begin() &;
00081
00082   /**
00083    * @brief Get end iterator
00084    * @return Iterator
00085    */
00086   Iterator end() &;
00087
00088   /**
00089    * @brief Get begin const iterator
00090    * @return ConstIterator
00091    */
00092   ConstIterator begin() const &;
00093
00094   /**
00095    * @brief Get end const iterator
00096    * @return ConstIterator
00097    */
00098   ConstIterator end() const &;
00099
00100   /**
00101    * @brief Get triangle's plane
00102    *
00103    * @return Plane<T>
00104    */
00105   Plane<T> getPlane() const;
00106
00107   /**
00108    * @brief Check is triangle valid
00109    *
00110    * @return true if triangle is valid
00111    * @return false if triangle is invalid
00112    */
00113   bool isValid() const;
00114
00115   /**
00116    * @brief Returns triangle's bound box
00117    *
00118    * @return BoundBox<T>
00119    */
00120   BoundBox<T> boundBox() const;
00121
00122   /**
00123    * @brief Checks if this Triangle belongs to BoundBox
00124    *
00125    * @param[in] bb BoundBox
00126    * @return true if Triangle belongs to BoundBox
00127    * @return false if Triangle doesn't belong to BoundBox
00128    */
00129   bool belongsTo(const BoundBox<T> &bb) const;
00130 };
00131
00132 /**
00133  * @brief Triangle print operator
00134  *
00135  * @tparam T - floating point type of coordinates
00136  * @param[in, out] ost output stream
00137  * @param[in] tr Triangle to print
00138  * @return std::ostream& modified ostream instance
00139  */
00140 template <std::floating_point T>
00141 std::ostream &operator«(std::ostream &ost, const Triangle<T> &tr)
00142 {
00143   ost « "Triangle: {";
00144   for (std::size_t i = 0; i < 3; ++i)
00145     ost « tr[i] « (i == 2 ? "" : ", ");
00146
00147   ost « "}";
```

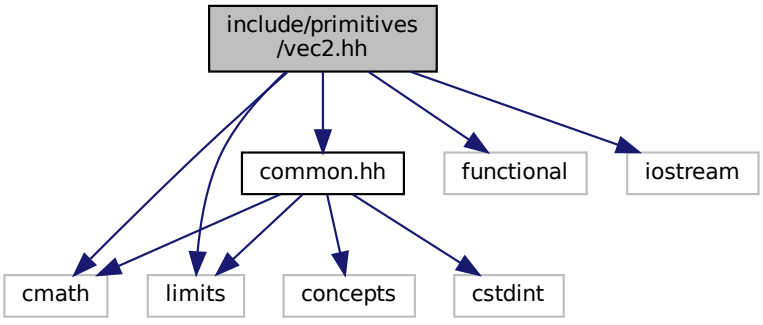
```
00148
00149   return ost;
00150 }
00151
00152 template <std::floating_point T>
00153 std::istream &operator»(std::istream &ist, Triangle<T> &tr)
00154 {
00155   ist » tr[0] » tr[1] » tr[2];
00156   return ist;
00157 }
00158
00159 template <std::floating_point T>
00160 Triangle<T>::Triangle() : vertices_()
00161 {}
00162
00163 template <std::floating_point T>
00164 Triangle<T>::Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3)
00165   : vertices_{p1, p2, p3}
00166 {}
00167
00168 template <std::floating_point T>
00169 const Vec3<T> &Triangle<T>::operator[](std::size_t idx) const &
00170 {
00171   return vertices_[idx % 3];
00172 }
00173
00174 template <std::floating_point T>
00175 Vec3<T> &&Triangle<T>::operator[](std::size_t idx) &&
00176 {
00177   return std::move(vertices_[idx % 3]);
00178 }
00179
00180 template <std::floating_point T>
00181 Vec3<T> &Triangle<T>::operator[](std::size_t idx) &
00182 {
00183   return vertices_[idx % 3];
00184 }
00185
00186 template <std::floating_point T>
00187 typename Triangle<T>::Iterator Triangle<T>::begin() &
00188 {
00189   return vertices_.begin();
00190 }
00191
00192 template <std::floating_point T>
00193 typename Triangle<T>::Iterator Triangle<T>::end() &
00194 {
00195   return vertices_.end();
00196 }
00197
00198 template <std::floating_point T>
00199 typename Triangle<T>::ConstIterator Triangle<T>::begin() const &
00200 {
00201   return vertices_.begin();
00202 }
00203
00204 template <std::floating_point T>
00205 typename Triangle<T>::ConstIterator Triangle<T>::end() const &
00206 {
00207   return vertices_.end();
00208 }
00209
00210 template <std::floating_point T>
00211 Plane<T> Triangle<T>::getPlane() const
00212 {
00213   return Plane<T>::getBy3Points(vertices_[0], vertices_[1], vertices_[2]);
00214 }
00215
00216 template <std::floating_point T>
00217 bool Triangle<T>::isValid() const
00218 {
00219   auto edge1 = vertices_[1] - vertices_[0];
00220   auto edge2 = vertices_[2] - vertices_[0];
00221
00222   auto cross12 = cross(edge1, edge2);
00223   return (cross12 != Vec3<T>{});
00224 }
00225
00226 template <std::floating_point T>
00227 BoundBox<T> Triangle<T>::boundBox() const
00228 {
00229   auto minMaxX = std::minmax({vertices_[0].x, vertices_[1].x, vertices_[2].x});
00230   auto minMaxY = std::minmax({vertices_[0].y, vertices_[1].y, vertices_[2].y});
00231   auto minMaxZ = std::minmax({vertices_[0].z, vertices_[1].z, vertices_[2].z});
00232
00233   return {
00234     minMaxX.first - ThresComp<T>::getThreshold(), minMaxX.second + ThresComp<T>::getThreshold(),
```

```
00235     minMaxY.first - ThresComp<T>::getThreshold(), minMaxY.second + ThresComp<T>::getThreshold(),
00236     minMaxZ.first - ThresComp<T>::getThreshold(), minMaxZ.second + ThresComp<T>::getThreshold()};
00237 }
00238
00239 template <std::floating_point T>
00240 bool Triangle<T>::belongsTo(const BoundBox<T> &bb) const
00241 {
00242   return boundBox().belongsTo(bb);
00243 }
00244
00245 } // namespace geom
00246
00247 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__
```

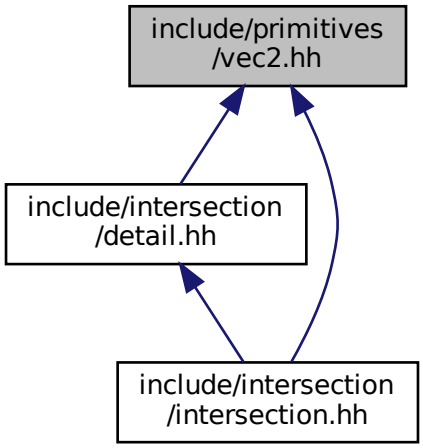
6.25 include/primitives/vec2.hh File Reference

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

Include dependency graph for vec2.hh:



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



Classes

- class geom::Vec2< T >
  Vec2 class realization.

Namespaces

- geom
  line.hh Line class implementation

Typedefs

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

Functions

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

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

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

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

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

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

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

  Vec2 print operator.

6.25.1 Detailed Description

Vec2 class implementation

Definition in file vec2.hh.

6.26 vec2.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_VEC2_HH__
00002 #define __INCLUDE_PRIMITIVES_VEC2_HH__
00003
00004 #include <cmath>
00005 #include <functional>
00006 #include <iostream>
00007 #include <limits>
00008
00009 #include "common.hh"
00010
00011 /**
00012  * @file vec2.hh
00013  * Vec2 class implementation
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Vec2
00021  * @brief Vec2 class realization
00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025 template <std::floating_point T>
00026 struct Vec2 final
00027 {
00028   /**
00029    * @brief Vec2 coordinates
00030    */
00031   T x{}, y{};
00032
00033   /**
00034    * @brief Construct a new Vec2 object from 3 coordinates
00035    *
00036    * @param[in] coordX x coordinate
00037    * @param[in] coordY y coordinate
00038    */
00039   Vec2(T coordX, T coordY) : x(coordX), y(coordY)
```

```
00040   {}
00041
00042   /**
00043    * @brief Construct a new Vec2 object with equals coordinates
00044    *
00045    * @param[in] coordX coordinate (default to {})
00046    */
00047   explicit Vec2(T coordX = {}) : Vec2(coordX, coordX)
00048   {}
00049
00050   /**
00051    * @brief Overloaded += operator
00052    * Increments vector coordinates by corresponding coordinates of vec
00053    * @param[in] vec vector to incremented with
00054    * @return Vec2& reference to current instance
00055    */
00056   Vec2 &operator+=(const Vec2 &vec);
00057
00058   /**
00059    * @brief Overloaded -= operator
00060    * Decrements vector coordinates by corresponding coordinates of vec
00061    * @param[in] vec vector to decremented with
00062    * @return Vec2& reference to current instance
00063    */
00064   Vec2 &operator-=(const Vec2 &vec);
00065
00066   /**
00067    * @brief Unary - operator
00068    *
00069    * @return Vec2 negated Vec2 instance
00070    */
00071   Vec2 operator-() const;
00072
00073   /**
00074    * @brief Overloaded *= by number operator
00075    *
00076    * @tparam nType numeric type of value to multiply by
00077    * @param[in] val value to multiply by
00078    * @return Vec2& reference to vector instance
00079    */
00080   template <Number nType>
00081   Vec2 &operator*=(nType val);
00082
00083   /**
00084    * @brief Overloaded /= by number operator
00085    *
00086    * @tparam nType numeric type of value to divide by
00087    * @param[in] val value to divide by
00088    * @return Vec2& reference to vector instance
00089    *
00090    * @warning Does not check if val equals 0
00091    */
00092   template <Number nType>
00093   Vec2 &operator/=(nType val);
00094
00095   /**
00096    * @brief Dot product function
00097    *
00098    * @param rhs vector to dot product with
00099    * @return T dot product of two vectors
00100    */
00101   T dot(const Vec2 &rhs) const;
00102
00103   /**
00104    * @brief Calculate squared length of a vector function
00105    *
00106    * @return T length^2
00107    */
00108   T length2() const;
00109
00110   /**
00111    * @brief Calculate length of a vector function
00112    *
00113    * @return T length
00114    */
00115   T length() const;
00116
00117   /**
00118    * @brief Get the perpendicular to this vector
00119    *
00120    * @return Vec2 perpendicular vector
00121    */
00122   Vec2 getPerp() const;
00123
00124   /**
00125    * @brief Get normalized vector function
00126    *
```

```
00127     * @return Vec2 normalized vector
00128     */
00129    Vec2 normalized() const;
00130
00131    /**
00132     * @brief Normalize vector function
00133     *
00134     * @return Vec2& reference to instance
00135     */
00136    Vec2 &normalize() &;
00137
00138    /**
00139     * @brief Overloaded operator [] (non-const version)
00140     * To get access to coordinates
00141     * @param i index of coordinate (0 - x, 1 - y)
00142     * @return T& reference to coordinate value
00143     *
00144     * @note Coordinates calculated by mod 2
00145     */
00146    T &operator[](std::size_t i) &;
00147
00148    /**
00149     * @brief Overloaded operator [] (const version)
00150     * To get access to coordinates
00151     * @param i index of coordinate (0 - x, 1 - y)
00152     * @return T coordinate value
00153     *
00154     * @note Coordinates calculated by mod 2
00155     */
00156    T operator[](std::size_t i) const &;
00157
00158    /**
00159     * @brief Overloaded operator [] (const version)
00160     * To get access to coordinates
00161     * @param i index of coordinate (0 - x, 1 - y)
00162     * @return T coordinate value
00163     *
00164     * @note Coordinates calculated by mod 2
00165     */
00166    T &&operator[](std::size_t i) &&;
00167
00168    /**
00169     * @brief Check if vector is parallel to another
00170     *
00171     * @param[in] rhs vector to check parallelism with
00172     * @return true if vector is parallel
00173     * @return false otherwise
00174     */
00175    bool isPar(const Vec2 &rhs) const;
00176
00177    /**
00178     * @brief Check if vector is perpendicular to another
00179     *
00180     * @param[in] rhs vector to check perpendicularity with
00181     * @return true if vector is perpendicular
00182     * @return false otherwise
00183     */
00184    bool isPerp(const Vec2 &rhs) const;
00185
00186    /**
00187     * @brief Check if vector is equal to another
00188     *
00189     * @param[in] rhs vector to check equality with
00190     * @return true if vector is equal
00191     * @return false otherwise
00192     */
00193    bool isEqual(const Vec2 &rhs) const;
00194 };
00195
00196 /**
00197  * @brief Overloaded + operator
00198  *
00199  * @tparam T vector template parameter
00200  * @param[in] lhs first vector
00201  * @param[in] rhs second vector
00202  * @return Vec2<T> sum of two vectors
00203  */
00204 template <std::floating_point T>
00205 Vec2<T> operator+(const Vec2<T> &lhs, const Vec2<T> &rhs)
00206 {
00207    Vec2<T> res{lhs};
00208    res += rhs;
00209    return res;
00210 }
00211
00212 /**
00213  * @brief Overloaded - operator
```

```
00214  *
00215  * @tparam T vector template parameter
00216  * @param[in] lhs first vector
00217  * @param[in] rhs second vector
00218  * @return Vec2<T> res of two vectors
00219  */
00220 template <std::floating_point T>
00221 Vec2<T> operator-(const Vec2<T> &lhs, const Vec2<T> &rhs)
00222 {
00223   Vec2<T> res{lhs};
00224   res -= rhs;
00225   return res;
00226 }
00227
00228 /**
00229  * @brief Overloaded multiple by value operator
00230  *
00231  * @tparam nT type of value to multiply by
00232  * @tparam T vector template parameter
00233  * @param[in] val value to multiply by
00234  * @param[in] rhs vector to multiply by value
00235  * @return Vec2<T> result vector
00236  */
00237 template <Number nT, std::floating_point T>
00238 Vec2<T> operator*(const nT &val, const Vec2<T> &rhs)
00239 {
00240   Vec2<T> res{rhs};
00241   res *= val;
00242   return res;
00243 }
00244
00245 /**
00246  * @brief Overloaded multiple by value operator
00247  *
00248  * @tparam nT type of value to multiply by
00249  * @tparam T vector template parameter
00250  * @param[in] val value to multiply by
00251  * @param[in] lhs vector to multiply by value
00252  * @return Vec2<T> result vector
00253  */
00254 template <Number nT, std::floating_point T>
00255 Vec2<T> operator*(const Vec2<T> &lhs, const nT &val)
00256 {
00257   Vec2<T> res{lhs};
00258   res *= val;
00259   return res;
00260 }
00261
00262 /**
00263  * @brief Overloaded divide by value operator
00264  *
00265  * @tparam nT type of value to divide by
00266  * @tparam T vector template parameter
00267  * @param[in] val value to divide by
00268  * @param[in] lhs vector to divide by value
00269  * @return Vec2<T> result vector
00270  */
00271 template <Number nT, std::floating_point T>
00272 Vec2<T> operator/(const Vec2<T> &lhs, const nT &val)
00273 {
00274   Vec2<T> res{lhs};
00275   res /= val;
00276   return res;
00277 }
00278
00279 /**
00280  * @brief Dot product function
00281  *
00282  * @tparam T vector template parameter
00283  * @param[in] lhs first vector
00284  * @param[in] rhs second vector
00285  * @return T dot production
00286  */
00287 template <std::floating_point T>
00288 T dot(const Vec2<T> &lhs, const Vec2<T> &rhs)
00289 {
00290   return lhs.dot(rhs);
00291 }
00292
00293 /**
00294  * @brief Vec2 equality operator
00295  *
00296  * @tparam T vector template parameter
00297  * @param[in] lhs first vector
00298  * @param[in] rhs second vector
00299  * @return true if vectors are equal
00300  * @return false otherwise
```

```
00301  */
00302 template <std::floating_point T>
00303 bool operator==(const Vec2<T> &lhs, const Vec2<T> &rhs)
00304 {
00305   return lhs.isEqual(rhs);
00306 }
00307
00308 /**
00309  * @brief Vec2 inequality operator
00310  *
00311  * @tparam T vector template parameter
00312  * @param[in] lhs first vector
00313  * @param[in] rhs second vector
00314  * @return true if vectors are not equal
00315  * @return false otherwise
00316  */
00317 template <std::floating_point T>
00318 bool operator!=(const Vec2<T> &lhs, const Vec2<T> &rhs)
00319 {
00320   return !(lhs == rhs);
00321 }
00322
00323 /**
00324  * @brief Vec2 print operator
00325  *
00326  * @tparam T vector template parameter
00327  * @param[in, out] ost output stream
00328  * @param[in] vec vector to print
00329  * @return std::ostream& modified stream instance
00330  */
00331 template <std::floating_point T>
00332 std::ostream &operator«(std::ostream &ost, const Vec2<T> &vec)
00333 {
00334   ost « "(" « vec.x « ", " « vec.y « ")";
00335   return ost;
00336 }
00337
00338 using Vec2D = Vec2<double>;
00339 using Vec2F = Vec2<float>;
00340
00341 template <std::floating_point T>
00342 Vec2<T> &Vec2<T>::operator+=(const Vec2 &vec)
00343 {
00344   x += vec.x;
00345   y += vec.y;
00346
00347   return *this;
00348 }
00349
00350 template <std::floating_point T>
00351 Vec2<T> &Vec2<T>::operator-=(const Vec2 &vec)
00352 {
00353   x -= vec.x;
00354   y -= vec.y;
00355
00356   return *this;
00357 }
00358
00359 template <std::floating_point T>
00360 Vec2<T> Vec2<T>::operator-() const
00361 {
00362   return Vec2{-x, -y};
00363 }
00364
00365 template <std::floating_point T>
00366 template <Number nType>
00367 Vec2<T> &Vec2<T>::operator*=(nType val)
00368 {
00369   x *= val;
00370   y *= val;
00371
00372   return *this;
00373 }
00374
00375 template <std::floating_point T>
00376 template <Number nType>
00377 Vec2<T> &Vec2<T>::operator/=(nType val)
00378 {
00379   x /= static_cast<T>(val);
00380   y /= static_cast<T>(val);
00381
00382   return *this;
00383 }
00384
00385 template <std::floating_point T>
00386 T Vec2<T>::dot(const Vec2 &rhs) const
00387 {
```

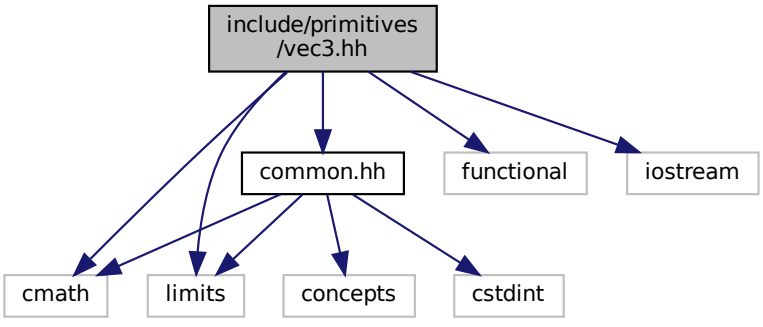
```
00388   return x * rhs.x + y * rhs.y;
00389 }
00390
00391 template <std::floating_point T>
00392 T Vec2<T>::length2() const
00393 {
00394   return dot(*this);
00395 }
00396
00397 template <std::floating_point T>
00398 T Vec2<T>::length() const
00399 {
00400   return std::sqrt(length2());
00401 }
00402
00403 template <std::floating_point T>
00404 Vec2<T> Vec2<T>::getPerp() const
00405 {
00406   return {y, -x};
00407 }
00408
00409 template <std::floating_point T>
00410 Vec2<T> Vec2<T>::normalized() const
00411 {
00412   Vec2 res{*this};
00413   res.normalize();
00414   return res;
00415 }
00416
00417 template <std::floating_point T>
00418 Vec2<T> &Vec2<T>::normalize() &
00419 {
00420   T len2 = length2();
00421   if (ThresComp<T>::isZero(len2) || ThresComp<T>::isEqual(len2, T{1}))
00422     return *this;
00423   return *this /= std::sqrt(len2);
00424 }
00425
00426 template <std::floating_point T>
00427 T &Vec2<T>::operator[](std::size_t i) &
00428 {
00429   switch (i % 2)
00430   {
00431   case 0:
00432     return x;
00433   case 1:
00434     return y;
00435   default:
00436     throw std::logic_error{"Impossible case in operator[]\n"};
00437   }
00438 }
00439
00440 template <std::floating_point T>
00441 T Vec2<T>::operator[](std::size_t i) const &
00442 {
00443   switch (i % 2)
00444   {
00445   case 0:
00446     return x;
00447   case 1:
00448     return y;
00449   default:
00450     throw std::logic_error{"Impossible case in operator[]\n"};
00451   }
00452 }
00453
00454 template <std::floating_point T>
00455 T &&Vec2<T>::operator[](std::size_t i) &&
00456 {
00457   switch (i % 2)
00458   {
00459   case 0:
00460     return std::move(x);
00461   case 1:
00462     return std::move(y);
00463   default:
00464     throw std::logic_error{"Impossible case in operator[]\n"};
00465   }
00466 }
00467
00468 template <std::floating_point T>
00469 bool Vec2<T>::isPar(const Vec2 &rhs) const
00470 {
00471   auto det = x * rhs.y - rhs.x * y;
00472   return ThresComp<T>::isZero(det);
00473 }
00474
```

```
00475 template <std::floating_point T>
00476 bool Vec2<T>::isPerp(const Vec2 &rhs) const
00477 {
00478   return ThresComp<T>::isZero(dot(rhs));
00479 }
00480
00481 template <std::floating_point T>
00482 bool Vec2<T>::isEqual(const Vec2 &rhs) const
00483 {
00484   return ThresComp<T>::isEqual(x, rhs.x) && ThresComp<T>::isEqual(y, rhs.y);
00485 }
00486
00487 } // namespace geom
00488
00489 #endif // __INCLUDE_PRIMITIVES_VEC2_HH__
```

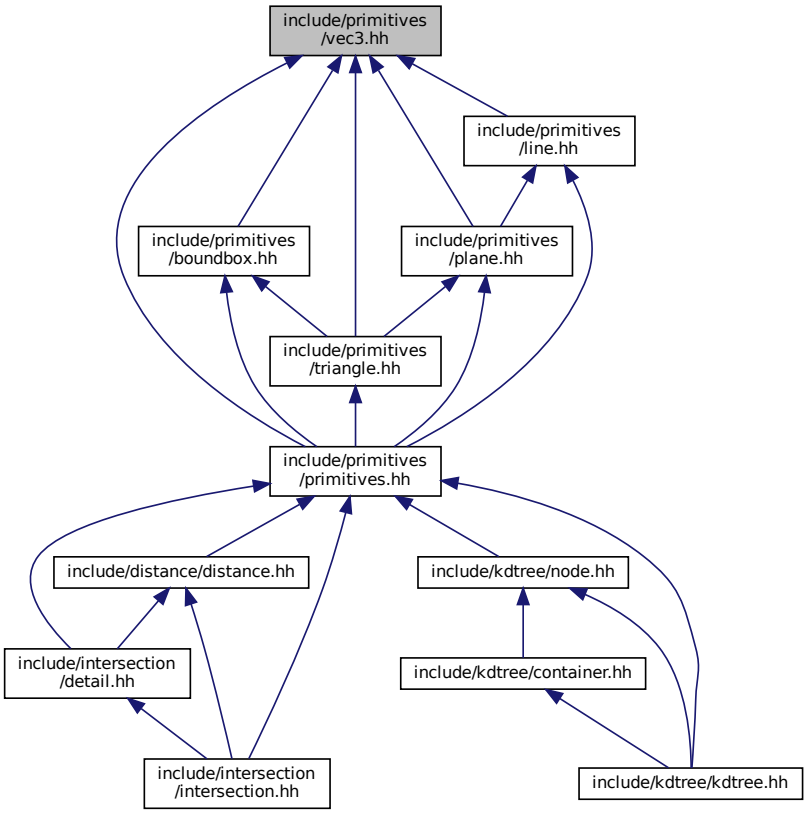
6.27 include/primitives/vec3.hh File Reference

```
#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>
  T geom::triple (const Vec3< T > &v1, const Vec3< T > &v2, const Vec3< T > &v3)

  Triple 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.

6.27.1 Detailed Description

Vec3 class implementation

Definition in file vec3.hh.

6.28 vec3.hh

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

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

```
00173     * @param[in] rhs vector to check parallelism with
00174     * @return true if vector is parallel
00175     * @return false otherwise
00176     */
00177    bool isPar(const Vec3 &rhs) const;
00178
00179    /**
00180     * @brief Check if vector is perpendicular to another
00181     *
00182     * @param[in] rhs vector to check perpendicularity with
00183     * @return true if vector is perpendicular
00184     * @return false otherwise
00185     */
00186    bool isPerp(const Vec3 &rhs) const;
00187
00188    /**
00189     * @brief Check if vector is equal to another
00190     *
00191     * @param[in] rhs vector to check equality with
00192     * @return true if vector is equal
00193     * @return false otherwise
00194     */
00195    bool isEqual(const Vec3 &rhs) const;
00196 };
00197
00198 /**
00199  * @brief Overloaded + operator
00200  *
00201  * @tparam T vector template parameter
00202  * @param[in] lhs first vector
00203  * @param[in] rhs second vector
00204  * @return Vec3<T> sum of two vectors
00205  */
00206 template <std::floating_point T>
00207 Vec3<T> operator+(const Vec3<T> &lhs, const Vec3<T> &rhs)
00208 {
00209    Vec3<T> res{lhs};
00210    res += rhs;
00211    return res;
00212 }
00213
00214 /**
00215  * @brief Overloaded - operator
00216  *
00217  * @tparam T vector template parameter
00218  * @param[in] lhs first vector
00219  * @param[in] rhs second vector
00220  * @return Vec3<T> res of two vectors
00221  */
00222 template <std::floating_point T>
00223 Vec3<T> operator-(const Vec3<T> &lhs, const Vec3<T> &rhs)
00224 {
00225    Vec3<T> res{lhs};
00226    res -= rhs;
00227    return res;
00228 }
00229
00230 /**
00231  * @brief Overloaded multiple by value operator
00232  *
00233  * @tparam nT type of value to multiply by
00234  * @tparam T vector template parameter
00235  * @param[in] val value to multiply by
00236  * @param[in] rhs vector to multiply by value
00237  * @return Vec3<T> result vector
00238  */
00239 template <Number nT, std::floating_point T>
00240 Vec3<T> operator*(const nT &val, const Vec3<T> &rhs)
00241 {
00242    Vec3<T> res{rhs};
00243    res *= val;
00244    return res;
00245 }
00246
00247 /**
00248  * @brief Overloaded multiple by value operator
00249  *
00250  * @tparam nT type of value to multiply by
00251  * @tparam T vector template parameter
00252  * @param[in] val value to multiply by
00253  * @param[in] lhs vector to multiply by value
00254  * @return Vec3<T> result vector
00255  */
00256 template <Number nT, std::floating_point T>
00257 Vec3<T> operator*(const Vec3<T> &lhs, const nT &val)
00258 {
00259    Vec3<T> res{lhs};
```

```
00260   res *= val;
00261   return res;
00262 }
00263
00264 /**
00265  * @brief Overloaded divide by value operator
00266  *
00267  * @tparam nT type of value to divide by
00268  * @tparam T vector template parameter
00269  * @param[in] val value to divide by
00270  * @param[in] lhs vector to divide by value
00271  * @return Vec3<T> result vector
00272  */
00273 template <Number nT, std::floating_point T>
00274 Vec3<T> operator/(const Vec3<T> &lhs, const nT &val)
00275 {
00276   Vec3<T> res{lhs};
00277   res /= val;
00278   return res;
00279 }
00280
00281 /**
00282  * @brief Dot product function
00283  *
00284  * @tparam T vector template parameter
00285  * @param[in] lhs first vector
00286  * @param[in] rhs second vector
00287  * @return T dot production
00288  */
00289 template <std::floating_point T>
00290 T dot(const Vec3<T> &lhs, const Vec3<T> &rhs)
00291 {
00292   return lhs.dot(rhs);
00293 }
00294
00295 /**
00296  * @brief Cross product function
00297  *
00298  * @tparam T vector template parameter
00299  * @param[in] lhs first vector
00300  * @param[in] rhs second vector
00301  * @return T cross production
00302  */
00303 template <std::floating_point T>
00304 Vec3<T> cross(const Vec3<T> &lhs, const Vec3<T> &rhs)
00305 {
00306   return lhs.cross(rhs);
00307 }
00308
00309 /**
00310  * @brief Triple product function
00311  *
00312  * @tparam T vector template parameter
00313  * @param[in] v1 first vector
00314  * @param[in] v2 second vector
00315  * @param[in] v3 third vector
00316  * @return T triple production
00317  */
00318 template <std::floating_point T>
00319 T triple(const Vec3<T> &v1, const Vec3<T> &v2, const Vec3<T> &v3)
00320 {
00321   return dot(v1, cross(v2, v3));
00322 }
00323
00324 /**
00325  * @brief Vec3 equality operator
00326  *
00327  * @tparam T vector template parameter
00328  * @param[in] lhs first vector
00329  * @param[in] rhs second vector
00330  * @return true if vectors are equal
00331  * @return false otherwise
00332  */
00333 template <std::floating_point T>
00334 bool operator==(const Vec3<T> &lhs, const Vec3<T> &rhs)
00335 {
00336   return lhs.isEqual(rhs);
00337 }
00338
00339 /**
00340  * @brief Vec3 inequality operator
00341  *
00342  * @tparam T vector template parameter
00343  * @param[in] lhs first vector
00344  * @param[in] rhs second vector
00345  * @return true if vectors are not equal
00346  * @return false otherwise
```

```
00347  */
00348 template <std::floating_point T>
00349 bool operator!=(const Vec3<T> &lhs, const Vec3<T> &rhs)
00350 {
00351   return !(lhs == rhs);
00352 }
00353
00354 /**
00355  * @brief Vec3 print operator
00356  *
00357  * @tparam T vector template parameter
00358  * @param[in, out] ost output stream
00359  * @param[in] vec vector to print
00360  * @return std::ostream& modified stream instance
00361  */
00362 template <std::floating_point T>
00363 std::ostream &operator«(std::ostream &ost, const Vec3<T> &vec)
00364 {
00365   ost « "(" « vec.x « ", " « vec.y « ", " « vec.z « ")";
00366   return ost;
00367 }
00368
00369 /**
00370  * @brief Vec3 scan operator
00371  *
00372  * @tparam T vector template parameter
00373  * @param[in, out] ist input stram
00374  * @param[in, out] vec vector to scan
00375  * @return std::istream& modified stream instance
00376  */
00377 template <std::floating_point T>
00378 std::istream &operator»(std::istream &ist, Vec3<T> &vec)
00379 {
00380   ist » vec.x » vec.y » vec.z;
00381   return ist;
00382 }
00383
00384 using Vec3D = Vec3<double>;
00385 using Vec3F = Vec3<float>;
00386
00387 template <std::floating_point T>
00388 Vec3<T> &Vec3<T>::operator+=(const Vec3 &vec)
00389 {
00390   x += vec.x;
00391   y += vec.y;
00392   z += vec.z;
00393
00394   return *this;
00395 }
00396
00397 template <std::floating_point T>
00398 Vec3<T> &Vec3<T>::operator-=(const Vec3 &vec)
00399 {
00400   x -= vec.x;
00401   y -= vec.y;
00402   z -= vec.z;
00403
00404   return *this;
00405 }
00406
00407 template <std::floating_point T>
00408 Vec3<T> Vec3<T>::operator-() const
00409 {
00410   return Vec3{-x, -y, -z};
00411 }
00412
00413 template <std::floating_point T>
00414 template <Number nType>
00415 Vec3<T> &Vec3<T>::operator*=(nType val)
00416 {
00417   auto fval = static_cast<T>(val);
00418   x *= fval;
00419   y *= fval;
00420   z *= fval;
00421
00422   return *this;
00423 }
00424
00425 template <std::floating_point T>
00426 template <Number nType>
00427 Vec3<T> &Vec3<T>::operator/=(nType val)
00428 {
00429   auto fval = static_cast<T>(val);
00430   x /= fval;
00431   y /= fval;
00432   z /= fval;
00433
```

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

```
00521   default:
00522     throw std::logic_error{"Impossible case in operator[]\n"};
00523   }
00524 }
00525
00526 template <std::floating_point T>
00527 bool Vec3<T>::isPar(const Vec3 &rhs) const
00528 {
00529   return cross(rhs).isEqual(Vec3<T>{0});
00530 }
00531
00532 template <std::floating_point T>
00533 bool Vec3<T>::isPerp(const Vec3 &rhs) const
00534 {
00535   return ThresComp<T>::isZero(dot(rhs));
00536 }
00537
00538 template <std::floating_point T>
00539 bool Vec3<T>::isEqual(const Vec3 &rhs) const
00540 {
00541   return ThresComp<T>::isEqual(x, rhs.x) && ThresComp<T>::isEqual(y, rhs.y) &&
00542          ThresComp<T>::isEqual(z, rhs.z);
00543 }
00544
00545 } // namespace geom
00546
00547 #endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```