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>
  std::ostream & operator<< (std::ostream &ost, const BoundBox< T > &bb)
- template<Number T>
  bool isEqualThreshold (T num1, T num2)
- template<Number T>
  bool isZeroThreshold (T num)
- template<std::floating_point T>
  std::ostream & operator<< (std::ostream &ost, const Line< T > &line)

  Line print 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>
  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>
  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 328 of file vec2.hh.

4.1.2.2 Vec2F

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

Definition at line 329 of file vec2.hh.

4.1.2.3 Vec3D

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

Definition at line 374 of file vec3.hh.

4.1.2.4 Vec3F

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

Definition at line 375 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::isIntersectValidInvalid(), and geom::detail::roguePos().

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<<() [1/6]

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

Definition at line 127 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.

4.1.4.6 isEqualThreshold()

```
template<Number T>
bool geom::isEqualThreshold (
            T num1,
            T num2 )
```

Definition at line 71 of file common.hh.

Referenced by geom::Plane< T >::belongs(), geom::Vec2< T >::isEqual(), geom::Vec3< T >::isEqual(), geom::Vec2< T >::normalize(), geom::Vec3< T >::normalize(), and geom::BoundBox< T >::operator==().

4.1.4.7 isZeroThreshold()

```
template<Number T>
bool geom::isZeroThreshold (
             T num )
```

Definition at line 77 of file common.hh.

References geom::ThresComp< T >::isZero().

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

4.1.4.8 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 151 of file line.hh.

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

4.1.4.9 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 169 of file plane.hh.

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

4.1.4.10 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.

4.1.4.11 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.

4.1.4.12 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 225 of file vec2.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> res of two vectors

Definition at line 241 of file vec2.hh.

4.1.4.14 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 258 of file vec2.hh.

4.1.4.15 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 275 of file vec2.hh.

4.1.4.16 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 292 of file vec2.hh.

4.1.4.17 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 308 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().

4.1.4.18 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 322 of file vec2.hh.

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

4.1.4.19 operator+() [2/2]

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

Overloaded + operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec3<T> sum of two vectors

Definition at line 227 of file vec3.hh.

4.1.4.20 operator-() [2/2]

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

Overloaded - operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec3<T> res of two vectors

Definition at line 243 of file vec3.hh.

4.1.4.21 operator*() [3/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 260 of file vec3.hh.

4.1.4.22 operator*() [4/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 277 of file vec3.hh.

4.1.4.23 operator/() [2/2]

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

Overloaded divide by value operator.

Template Parameters

nT	type of value to divide by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 294 of file vec3.hh.

4.1.4.24 dot() [2/2]

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

Dot product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T dot production

Definition at line 310 of file vec3.hh.

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

4.1.4.25 cross()

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

Cross product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T cross production

Definition at line 324 of file vec3.hh.

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

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

4.1.4.26 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 339 of file vec3.hh.

References cross(), and dot().

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

4.1.4.27 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 353 of file vec3.hh.

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

4.1.4.28 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 368 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)
- template<std::bidirectional_iterator It>
  std::size_t roguePos (It begin, It end)

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 80 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 103 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 117 of file detail.hh.

References geom::Triangle< T >::begin(), geom::Line< T >::dir(), geom::Triangle< T >::end(), geom::Line< T >::org(), and roguePos().

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 140 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 158 of file detail.hh.

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

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 183 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 211 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 224 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 248 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 254 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 260 of file detail.hh.

References geom::isZeroThreshold().

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

4.2.2.12 isAllPosNeg() [2/2]

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

Definition at line 271 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 278 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 286 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 320 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 340 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 349 of file detail.hh.

References geom::distance().

Referenced by isIntersectBothInvalid(), and isIntersectValidInvalid().

4.2.2.18 roguePos()

```
template<std::bidirectional_iterator It>
std::size_t geom::detail::roguePos (
             It begin,
             It end )
```

Definition at line 362 of file detail.hh.

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

Referenced by helperMollerHaines().

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
- bool operator== (const BoundBox &rhs) const
- bool operator!= (const BoundBox &rhs) 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 43 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 67 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 73 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 79 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 85 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 91 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 97 of file boundbox.hh.

References BBFILL.

5.1.2.8 getMaxDim()

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

Definition at line 105 of file boundbox.hh.

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

5.1.2.9 operator==()

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

Definition at line 113 of file boundbox.hh.

References geom::isEqualThreshold(), geom::BoundBox< T >::maxX, geom::BoundBox< T >::maxY, geom::BoundBox< T >::min geom::BoundBox< T >::minY, and geom::BoundBox< T >::minZ.

5.1.2.10 operator"!=()

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

Definition at line 121 of file boundbox.hh.

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::BoundBox< T >::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::BoundBox< T >::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::BoundBox< T >::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::BoundBox< T >::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::BoundBox< T >::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 78 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 82 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 88 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 94 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 100 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 106 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 112 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 118 of file container.hh.

5.2.3.8 sepAxis()

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

Definition at line 124 of file container.hh.

5.2.3.9 boundBox()

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

Definition at line 130 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 136 of file container.hh.

5.2.3.11 left()

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

Definition at line 142 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 148 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 154 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 =default

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 164 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 176 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 182 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 189 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 197 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 203 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  [default]
```

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.

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.
- bool operator== (const Line &rhs) const

  Line equality operator.
- bool operator!= (const Line &rhs) const

  Line inequality operator.
- 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 158 of file line.hh.

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

5.8.3 Member Function Documentation

5.8.3.1 operator==()

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

Line equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in	*rhs*	2nd line

Returns

true if lines are equal

false if lines are not equal

Definition at line 165 of file line.hh.

5.8.3.2 operator"!=()

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

Line inequality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in	*rhs*	2nd line

Returns

true if lines are not equal

false if lines are equal

Definition at line 171 of file line.hh.

5.8.3.3 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 177 of file line.hh.

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

5.8.3.4 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 183 of file line.hh.

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

5.8.3.5 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

5.8.3.6 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

5.8.3.7 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 202 of file line.hh.

Referenced by geom::intersect().

5.8.3.8 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 208 of file line.hh.

5.8.3.9 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 214 of file line.hh.

Referenced by geom::intersect().

5.8.3.10 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 220 of file line.hh.

Referenced by geom::intersect().

5.8.3.11 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 226 of file line.hh.

References geom::isZeroThreshold(), and geom::triple().

Referenced by geom::intersect().

5.8.3.12 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 233 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

- bool operator== (const Plane &rhs) const

  Plane equality operator.
- bool operator!= (const Plane &rhs) const

  Plane inequality operator.
- 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 operator==()

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

Plane equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in	*rhs*	2nd plane

Returns

true if planes are equal

false if planes are not equal

Definition at line 183 of file plane.hh.

5.9.2.2 operator"!=()

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

Plane inequality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

in	*rhs*	2nd plane

Returns

true if planes are not equal

false if planes are equal

Definition at line 189 of file plane.hh.

5.9.2.3 dist()

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

Getter for distance.

Returns

T value of distance

Definition at line 195 of file plane.hh.

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

5.9.2.4 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 201 of file plane.hh.

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

5.9.2.5 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

5.9.2.6 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 213 of file plane.hh.

References geom::isEqualThreshold().

5.9.2.7 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 219 of file plane.hh.

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

5.9.2.8 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 225 of file plane.hh.

5.9.2.9 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 231 of file plane.hh.

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

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

5.9.2.10 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 237 of file plane.hh.

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

5.9.2.11 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 243 of file plane.hh.

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

5.9.2.12 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 250 of file plane.hh.

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

5.9.2.13 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 257 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::isZeroThreshold().

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.
- bool operator== (const Vec2 &rhs) const

  Vec2 equality operator.
- bool operator!= (const Vec2 &rhs) const

  Vec2 equality operator.
- 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>
bool geom::Vec2< T >::operator== (
            const Vec2< T > & rhs ) const
```

Vec2 equality operator.

Template Parameters

T	vector template parameter

Parameters

in	*rhs*	second vector

Returns

true if vectors are equal

false otherwise

Definition at line 332 of file vec2.hh.

5.12.3.2 operator"!=()

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

Vec2 equality operator.

Template Parameters

T	vector template parameter

Parameters

in	*rhs*	second vector

Returns

true if vectors are not equal

false otherwise

Definition at line 338 of file vec2.hh.

5.12.3.3 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 344 of file vec2.hh.

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

5.12.3.4 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 353 of file vec2.hh.

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

5.12.3.5 operator-()

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

Unary - operator.

Returns

Vec2 negated Vec2 instance

Definition at line 362 of file vec2.hh.

5.12.3.6 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

5.12.3.7 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

5.12.3.8 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 388 of file vec2.hh.

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

Referenced by geom::dot().

5.12.3.9 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 394 of file vec2.hh.

References geom::dot().

5.12.3.10 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 400 of file vec2.hh.

5.12.3.11 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 406 of file vec2.hh.

5.12.3.12 normalized()

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

Get normalized vector function.

Returns

Vec2 normalized vector

Definition at line 412 of file vec2.hh.

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

5.12.3.13 normalize()

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

Normalize vector function.

Returns

Vec2& reference to instance

Definition at line 420 of file vec2.hh.

References geom::isEqualThreshold(), and geom::isZeroThreshold().

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

5.12.3.14 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 429 of file vec2.hh.

5.12.3.15 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 443 of file vec2.hh.

5.12.3.16 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 457 of file vec2.hh.

5.12.3.17 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 471 of file vec2.hh.

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

5.12.3.18 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 478 of file vec2.hh.

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

5.12.3.19 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 484 of file vec2.hh.

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

5.12.3.20 operator*=() [2/2]

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

Definition at line 369 of file vec2.hh.

5.12.3.21 operator/=() [2/2]

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

Definition at line 379 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.
- bool operator== (const Vec3 &rhs) const

  Vec3 equality operator.
- bool operator!= (const Vec3 &rhs) const

  Vec3 inequality operator.
- 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>
bool geom::Vec3< T >::operator== (
             const Vec3< T > & rhs ) const
```

Vec3 equality operator.

Template Parameters

T	vector template parameter

Parameters

in	*rhs*	second vector

Returns

true if vectors are equal

false otherwise

Definition at line 378 of file vec3.hh.

5.13.3.2 operator"!=()

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

Vec3 inequality operator.

Template Parameters

T	vector template parameter

Parameters

in	*rhs*	second vector

Returns

true if vectors are not equal

false otherwise

Definition at line 384 of file vec3.hh.

5.13.3.3 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 390 of file vec3.hh.

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

5.13.3.4 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 400 of file vec3.hh.

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

5.13.3.5 operator-()

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

Unary - operator.

Returns

Vec3 negated Vec3 instance

Definition at line 410 of file vec3.hh.

5.13.3.6 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

5.13.3.7 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

5.13.3.8 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 440 of file vec3.hh.

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

Referenced by geom::dot().

5.13.3.9 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 446 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().

5.13.3.10 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 452 of file vec3.hh.

References geom::dot().

5.13.3.11 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 458 of file vec3.hh.

5.13.3.12 normalized()

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

Get normalized vector function.

Returns

Vec3 normalized vector

Definition at line 464 of file vec3.hh.

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

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

5.13.3.13 normalize()

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

Normalize vector function.

Returns

Vec3& reference to instance

Definition at line 472 of file vec3.hh.

References geom::isEqualThreshold(), and geom::isZeroThreshold().

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

5.13.3.14 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 481 of file vec3.hh.

5.13.3.15 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 497 of file vec3.hh.

5.13.3.16 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 513 of file vec3.hh.

5.13.3.17 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 529 of file vec3.hh.

References geom::cross().

5.13.3.18 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 535 of file vec3.hh.

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

5.13.3.19 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 541 of file vec3.hh.

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

5.13.3.20 operator*=() [2/2]

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

Definition at line 417 of file vec3.hh.

5.13.3.21 operator/=() [2/2]

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

Definition at line 429 of file vec3.hh.

5.13.4 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

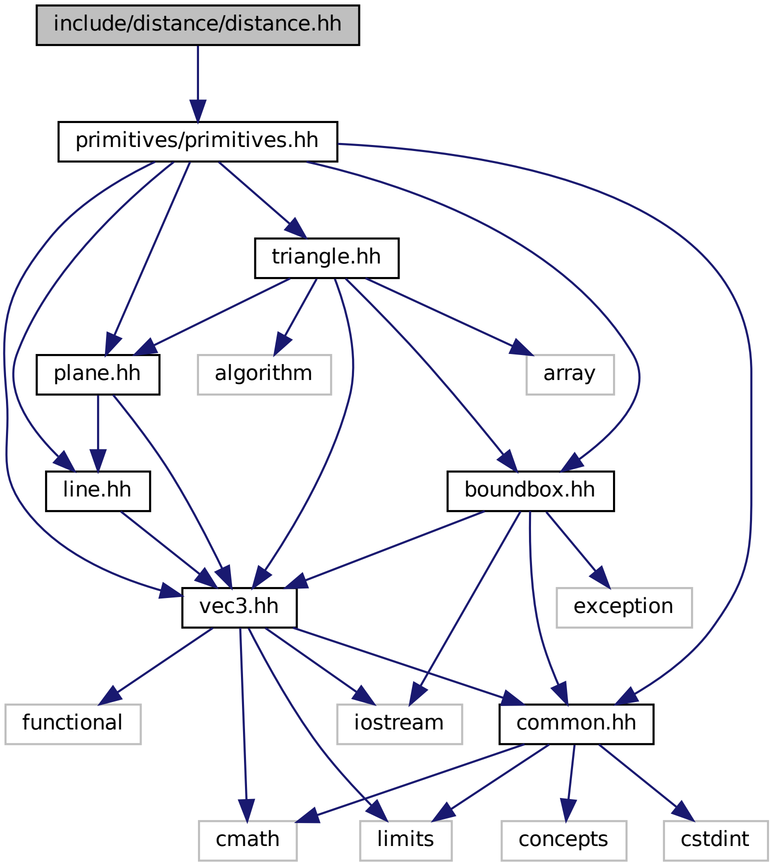
Chapter 6

File Documentation

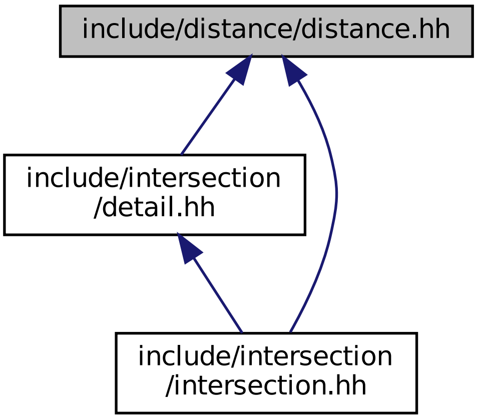
6.1 include/distance/distance.hh File Reference

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

Include dependency graph for distance.hh:



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


include/distance/distance.hh
include/intersection /detail.hh
include/intersection /intersection.hh

Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Calculates signed distance between point and plane.

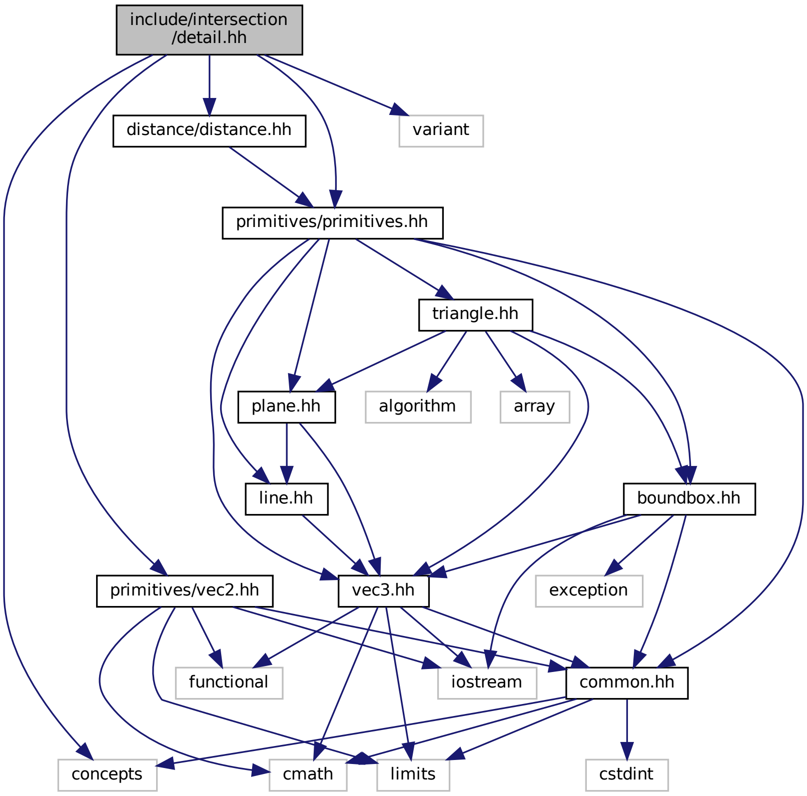
6.2 distance.hh

```
#ifndef __INCLUDE_DISTANCE_DISTANCE_HH__
#define __INCLUDE_DISTANCE_DISTANCE_HH__

#include "primitives/primitives.hh"

namespace geom
{

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

} // namespace geom

namespace geom
{

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

} // namespace geom

#endif // __INCLUDE_DISTANCE_DISTANCE_HH__
```

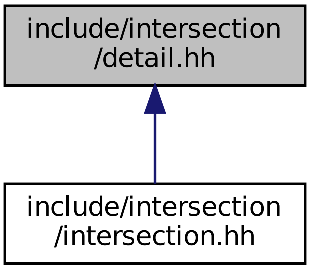
6.3 include/intersection/detail.hh File Reference

```
#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)
- template<std::bidirectional_iterator It>
  std::size_t geom::detail::roguePos (It begin, It end)

6.4 detail.hh

```
#ifndef __INCLUDE_INTERSECTION_DETAIL_HH__
#define __INCLUDE_INTERSECTION_DETAIL_HH__

#include <concepts>
#include <variant>

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

namespace geom::detail
{

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>>;

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);

template <std::bidirectional_iterator It>
std::size_t roguePos(It begin, It end);

//=====================================================================

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

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

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

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

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

  return true;
}

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

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

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

  return isOverlap(params1, params2);
}

template <std::floating_point T>
Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
{
  /* Project the triangle vertices onto line */
  std::array<T, 3> vert{};
  std::transform(tr.begin(), tr.end(), vert.begin(),
                 [dir = l.dir(), org = l.org()](auto &&v) { return dot(dir, v - org); });

  std::array<T, 3> sdist{};
  std::transform(tr.begin(), tr.end(), sdist.begin(), std::bind_front(distance<T>, pl));

  /* Looking for vertex which is alone on it's side */
  std::size_t rogue = roguePos(sdist.begin(), sdist.end());

  std::array<T, 2> segm{};
  std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};
  std::transform(arr.begin(), arr.end(), segm.begin(), [&vert, &sdist, rogue](auto i) {
    return vert[i] + (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]);
  });

  return std::minmax(segm[0], segm[1]);
}
```

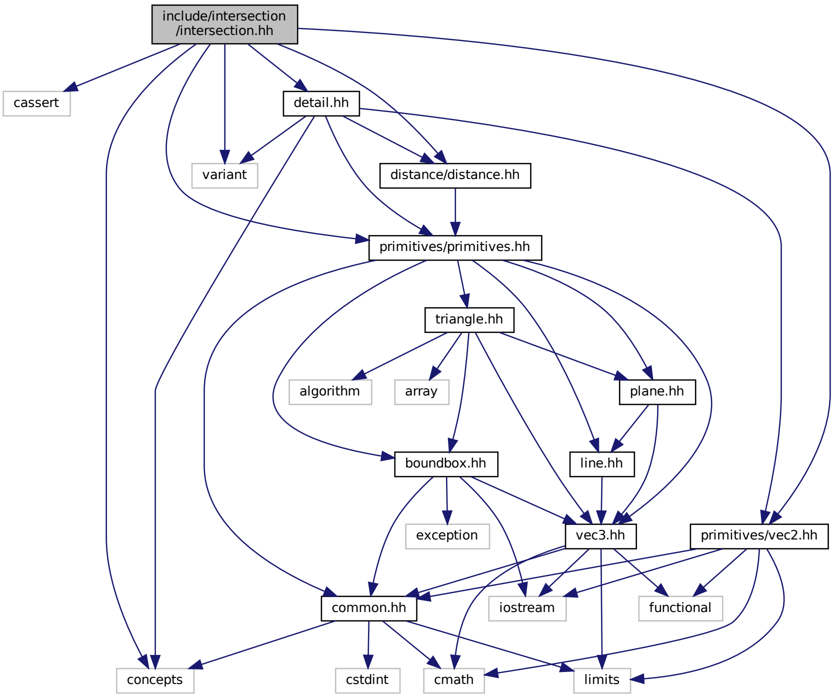
```

template <std::floating_point T>
bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  auto isPoint1 = isPoint(tr1);
  auto isPoint2 = isPoint(tr2);

  if (isPoint1 && isPoint2)
    return tr1[0] == tr2[0];

  if (isPoint1)
    return isIntersectPointSegment(tr1[0], getSegment(tr2));

  if (isPoint2)
    return isIntersectPointSegment(tr2[0], getSegment(tr1));

  return isIntersectSegmentSegment(getSegment(tr1), getSegment(tr2));
}

template <std::floating_point T>
bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid)
{
  if (isPoint(invalid))
    return isIntersectPointTriangle(invalid[0], valid);

  auto segm = getSegment(invalid);
  auto pl = valid.getPlane();

  auto dst1 = distance(pl, segm.first);
  auto dst2 = distance(pl, segm.second);

  if (dst1 * dst2 > 0)
    return false;

  if (isZeroThreshold(dst1) && isZeroThreshold(dst2))
    return isIntersect2D(valid, invalid);

  dst1 = std::abs(dst1);
  dst2 = std::abs(dst2);

  auto pt = segm.first + (segm.second - segm.first) * dst1 / (dst1 + dst2);
  return isIntersectPointTriangle(pt, valid);
}

template <std::floating_point T>
bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr)
{
  if (!tr.getPlane().belongs(pt))
    return false;

  /* TODO: comment better */
  /* pt = point + u * edge1 + v * edge2 */
  auto point = pt - tr[0];
  auto edge1 = tr[1] - tr[0];
  auto edge2 = tr[2] - tr[0];

  auto dotE1E1 = dot(edge1, edge1);
  auto dotE1E2 = dot(edge1, edge2);
  auto dotE1PT = dot(edge1, point);

  auto dotE2E2 = dot(edge2, edge2);
  auto dotE2PT = dot(edge2, point);

  auto denom = dotE1E1 * dotE2E2 - dotE1E2 * dotE1E2;
  auto u = (dotE2E2 * dotE1PT - dotE1E2 * dotE2PT) / denom;
  auto v = (dotE1E1 * dotE2PT - dotE1E2 * dotE1PT) / denom;

  /* Point belongs to triangle if: (u >= 0) && (v >= 0) && (u + v <= 1) */
  auto eps = ThresComp<T>::getThreshold();
  return (u > -eps) && (v > -eps) && (u + v < 1 + eps);
}

template <std::floating_point T>
bool isIntersectPointSegment(const Vec3<T> &pt, const Segment3D<T> &segm)
{
  Line<T> l{segm.first, segm.second - segm.first};
  if (!l.belongs(pt))
    return false;

  auto beg = dot(l.dir(), segm.first - pt);
  auto end = dot(l.dir(), segm.second - pt);

  return !isAllPosNeg(beg, end);
}

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

```
{
  Line<T> l1{segm1.first, segm1.second - segm1.first};
  Line<T> l2{segm2.first, segm2.second - segm2.first};
  auto intersectionResult = intersect(l1, l2);

  if (std::holds_alternative<Line<T>>(intersectionResult))
  {
    const auto &dir = l1.dir();
    Segment2D<T> s1{dot(dir, segm1.first), dot(dir, segm1.second)};
    Segment2D<T> s2{dot(dir, segm2.first), dot(dir, segm2.second)};
    return isOverlap(s1, s2);
  }

  if (std::holds_alternative<Vec3<T>>(intersectionResult))
  {
    auto pt = std::get<Vec3<T»(intersectionResult);
    return isIntersectPointSegment(pt, segm1) && isIntersectPointSegment(pt, segm2);
  }

  return false;
}

template <std::floating_point T>
bool isPoint(const Triangle<T> &tr)
{
  return (tr[0] == tr[1]) && (tr[0] == tr[2]);
}

template <std::floating_point T>
bool isOverlap(Segment2D<T> &segm1, Segment2D<T> &segm2)
{
  return (segm2.first <= segm1.second) && (segm2.second >= segm1.first);
}

template <std::forward_iterator It>
bool isAllPosNeg(It begin, It end)
{
  if (begin == end)
    return true;

  bool fst = (*begin > 0);
  return std::none_of(std::next(begin), end,
                      [fst](auto &&elt) { return (elt > 0) != fst || isZeroThreshold(elt); });
}

template <std::floating_point T>
bool isAllPosNeg(T num1, T num2)
{
  auto thres = ThresComp<T>::getThreshold();
  return (num1 > thres && num2 > thres) || (num1 < -thres && num2 < -thres);
}

template <std::floating_point T>
bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr)
{
  std::array<T, 3> sdist{};
  std::transform(tr.begin(), tr.end(), sdist.begin(), std::bind_front(distance<T>, pl));
  return detail::isAllPosNeg(sdist.begin(), sdist.end());
}

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

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

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

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

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

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

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

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

  return res;
}

template <std::floating_point T>
bool isCounterClockwise(Trian2<T> &tr)
{
  /**
   * The triangle is counterclockwise ordered if \delta > 0
   * and clockwise ordered if \delta < 0.
   *
   *              +  1  1  1 +
   * \delta = det | x0 x1 x2 | = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0)
   *              + y0 y1 y2 +                       + (x0 * y1 - x1 * y0)
   *
   */

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

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

template <std::floating_point T>
Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
{
  std::array<T, 3> dotArr{};
  std::transform(tr.begin(), tr.end(), dotArr.begin(), [&d](auto &&v) { return dot(d, v); });
  auto mmIt = std::minmax_element(dotArr.begin(), dotArr.end());
  return {*mmIt.first, *mmIt.second};
}

template <std::floating_point T>
Segment3D<T> getSegment(const Triangle<T> &tr)
{
  std::array<T, 3> lenArr{};
  for (std::size_t i = 0; i < 3; ++i)
    lenArr[i] = (tr[i] - tr[i + 1]).length2();

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

  return {tr[maxIdx], tr[maxIdx + 1]};
}

template <std::bidirectional_iterator It>
std::size_t roguePos(It beg, It end)
{
  using T = typename std::iterator_traits<It>::value_type;

  auto isDiffSides = [thres = ThresComp<T>::getThreshold()](auto lhs, auto rhs) {
    return (lhs > thres && rhs < -thres) || (lhs < -thres && rhs > thres);
  };

  for (std::size_t i = 0; i < 3; ++i)
    if (isDiffSides(*(beg + i), *(beg + (i + 1) % 3)))
      return i;

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

  if (std::none_of(isOneSide.begin(), isOneSide.end(), std::identity{}))
  {
    auto rbeg = std::reverse_iterator(end);
    auto rend = std::reverse_iterator(beg);
    auto rogueIt = std::find_if_not(rbeg, rend, isZeroThreshold<T>);
    return (rogueIt == rend) ? 0 : std::distance(rogueIt, rend) - 1;
  }

  for (std::size_t i = 0; i < 3; ++i)
    if (isOneSide[i])
      return (i + 2) % 3;

  return 0;
}

} // namespace geom::detail

#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.

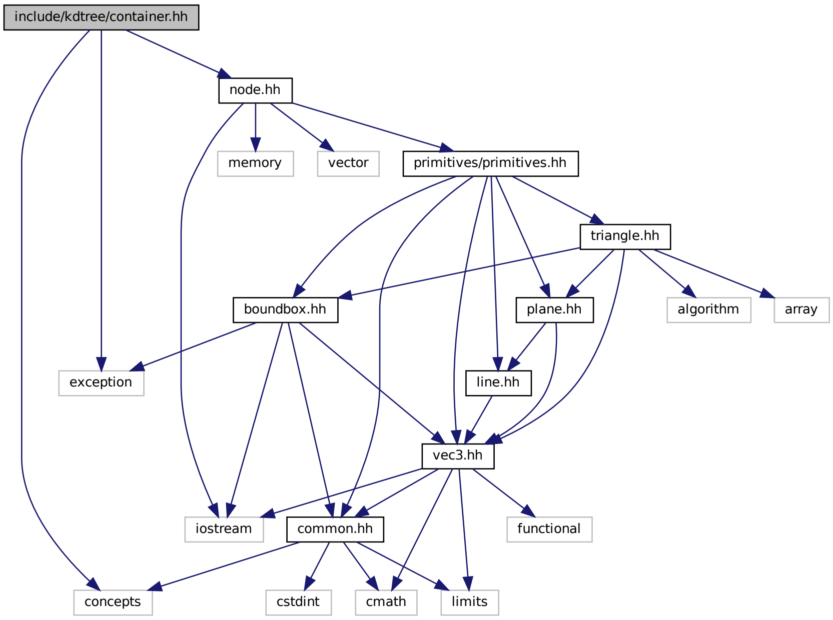
6.6 intersection.hh

```
#ifndef __INCLUDE_INTERSECTION_INTERSECTION_HH__
#define __INCLUDE_INTERSECTION_INTERSECTION_HH__

#include <cassert>
#include <concepts>
#include <variant>

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

#include "detail.hh"

namespace geom
{

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

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

/**
 * @brief Intersect 2 lines and return result of intersection
 * @details
 * Common intersection case (parallel & skew lines cases are trivial):
 * Let \f$ \overrightarrow{P} \f$ - point in space, intersection point of two lines.
 *
 * \f$ l_1 \f$ equation: \f$ \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
 * \overrightarrow{P} \f$
 *
 * \f$ l_2 \f$ equation: \f$ \overrightarrow{org}_2 + \overrightarrow{dir}_2
 * \cdot t_2 = \overrightarrow{P} \f$
 *
 * Let’s equate left sides:
 * \f[
 * \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
 * \overrightarrow{org}_2 + \overrightarrow{dir}_2 \cdot t_2
 * \f]
 * Cross multiply both sides from right by \f$ \overrightarrow{dir}_2 \f$:
 * \f[
 * t_1 \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right) =
 * \left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times \overrightarrow{dir}_2
 * \f]
 * Dot multiply both sides by \f$ \frac{\overrightarrow{dir}_1 \times \overrightarrow{dir}_2}{\left|
 * \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2} \f$:
 *
 * \f[
 * t_1 = \frac{
 *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times
 * \overrightarrow{dir}_2\right) \cdot \left( \overrightarrow{dir}_1 \times \overrightarrow{dir}_2
 * \right)
 * }{
 * \left| \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2
 * }
 * \f]
 *
 * Thus we get intersection point parameter \f$ t_1 \f$ on \f$ l_1 \f$, let’s substitute it to \f$
 * l_1 \f$ equation: \f[ \overrightarrow{P} = \overrightarrow{org}_1 + \frac{
 *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \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
 * \f]
 *
 * @tparam T - floating point type of coordinates
 * @param[in] l1 first line
 * @param[in] l2 second line
 * @return std::variant<std::monostate, Vec3<T>, Line<T»
 */
template <std::floating_point T>
std::variant<std::monostate, Vec3<T>, Line<T» intersect(const Line<T> &l1, const Line<T> &l2);

template <std::floating_point T>
bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
{
  auto isInv1 = !tr1.isValid();
  auto isInv2 = !tr2.isValid();

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

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

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

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

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

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

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

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

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

  auto dir = cross(n1, n2);

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

    return std::monostate{};
  }

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

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

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

template <std::floating_point T>
std::variant<std::monostate, Vec3<T>, Line<T>> intersect(const Line<T> &l1, const Line<T> &l2)
{
  if (l1.isPar(l2))
  {
    if (l1.isEqual(l2))
      return l1;

    return std::monostate{};
  }

  if (l1.isSkew(l2))
    return std::monostate{};

  auto dir1xdir2 = cross(l1.dir(), l2.dir());
  auto org21xdir2 = cross(l2.org() - l1.org(), l2.dir());

  auto t1_intersect = dot(org21xdir2, dir1xdir2) / dir1xdir2.length2();

  auto point = l1.getPoint(t1_intersect);

  return point;
}

} // namespace geom

#endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__
```

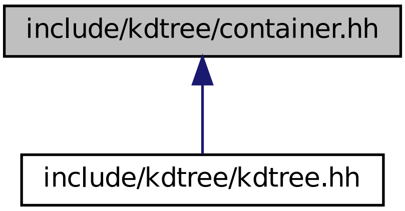
6.7 include/kdtree/container.hh File Reference

```
#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

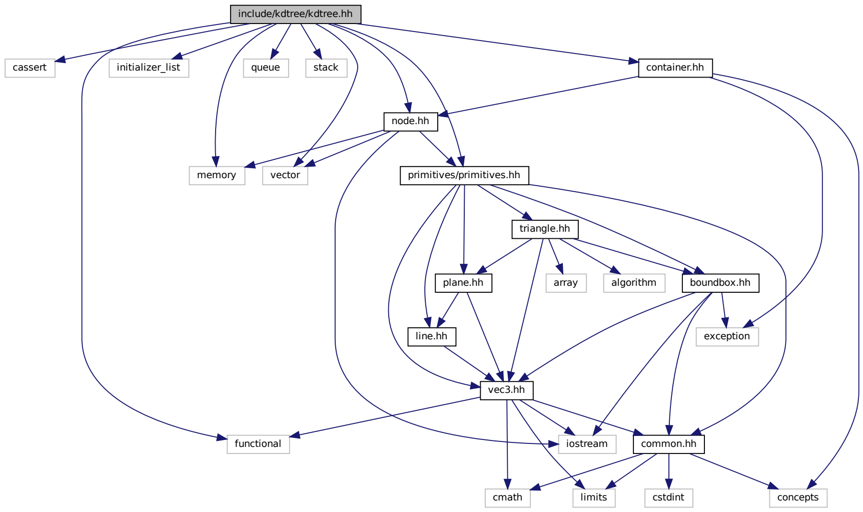
```cpp
#ifndef __INCLUDE_KDTREE_CONTAINER_HH__
#define __INCLUDE_KDTREE_CONTAINER_HH__

#include <concepts>
#include <exception>

#include "node.hh"

namespace geom::kdtree
{

template <std::floating_point T>
class KdTree;

template <std::floating_point T>
class Container final
{
private:
  const KdTree<T> *tree_;
  const Node<T> *node_;

public:
  Container(const KdTree<T> *tree, const Node<T> *node);

  class ConstIterator;
  ConstIterator cbegin() const &;
  ConstIterator cend() const &;

  ConstIterator begin() const &;
  ConstIterator end() const &;

  typename Node<T>::IndexConstIterator indexBegin() const &;
  typename 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;

  class ConstIterator final
  {
  public:
    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> *;

  private:
    const Container *cont_;
    std::vector<Index>::const_iterator curIdxIt_{};

  public:
    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 = default;
  };
};

//============================================================================================
//                                    Container definitions
//============================================================================================

template <std::floating_point T>
Container<T>::Container(const KdTree<T> *tree, const Node<T> *node) : tree_(tree), node_(node)
{}

template <std::floating_point T>
typename Container<T>::ConstIterator Container<T>::cbegin() const &
{
  return ConstIterator{this};
}
```

```

template <std::floating_point T>
typename Container<T>::ConstIterator Container<T>::cend() const &
{
  return ConstIterator{this, /* isEnd = */ true};
}

template <std::floating_point T>
typename Container<T>::ConstIterator Container<T>::begin() const &
{
  return cbegin();
}

template <std::floating_point T>
typename Container<T>::ConstIterator Container<T>::end() const &
{
  return cend();
}

template <std::floating_point T>
typename Node<T>::IndexConstIterator Container<T>::indexBegin() const &
{
  return node_->indicies.begin();
}

template <std::floating_point T>
typename Node<T>::IndexConstIterator Container<T>::indexEnd() const &
{
  return node_->indicies.end();
}

template <std::floating_point T>
T Container<T>::separator() const
{
  return node_->separator;
}

template <std::floating_point T>
Axis Container<T>::sepAxis() const
{
  return node_->sepAxis;
}

template <std::floating_point T>
BoundBox<T> Container<T>::boundBox() const
{
  return node_->boundBox;
}

template <std::floating_point T>
const Triangle<T> &Container<T>::triangleByIndex(Index index) const &
{
  return tree_->triangleByIndex(index);
}

template <std::floating_point T>
Container<T> Container<T>::left() const
{
  return Container<T>{tree_, node_->left.get()};
}

template <std::floating_point T>
Container<T> Container<T>::right() const
{
  return Container<T>{tree_, node_->right.get()};
}

template <std::floating_point T>
bool Container<T>::isValid() const
{
  return (tree_ != nullptr) && (node_ != nullptr);
}

//============================================================================================
//                                Container::ConstIterator definitions
//============================================================================================

template <std::floating_point T>
Container<T>::ConstIterator::ConstIterator(const Container<T> *cont, bool isEnd) : cont_(cont)
{
  if (nullptr == cont_)
    throw std::invalid_argument("Tried to create iterator with invalid Container pointer");

  if (isEnd)
    curIdxIt_ = cont_->indexEnd();
  else
    curIdxIt_ = cont_->indexBegin();
}

template <std::floating_point T>
Index Container<T>::ConstIterator::getIndex()
{
  return *curIdxIt_;
}

template <std::floating_point T>
typename Container<T>::ConstIterator &Container<T>::ConstIterator::operator++()
{
  ++curIdxIt_;
  return *this;
}

template <std::floating_point T>
typename Container<T>::ConstIterator Container<T>::ConstIterator::operator++(int)
{
  auto tmp = *this;
  operator++();
  return tmp;
}

template <std::floating_point T>
typename Container<T>::ConstIterator::reference Container<T>::ConstIterator::operator*() const
{
  return cont_->triangleByIndex(*curIdxIt_);
}

template <std::floating_point T>
typename Container<T>::ConstIterator::pointer Container<T>::ConstIterator::operator->() const
{
  return &cont_->triangleByIndex(*curIdxIt_);
}

} // namespace geom::kdtree

#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

```
#ifndef __INCLUDE_KDTREE_KDTREE_HH__
#define __INCLUDE_KDTREE_KDTREE_HH__

#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"

namespace geom::kdtree
{

template <std::floating_point T>
class KdTree final
{
private:
  std::unique_ptr<Node<T» root_{};
  std::vector<Triangle<T» triangles_{};
  std::size_t nodeCapacity_{1};

public:
  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;

  class ConstIterator;

  // ConstIterators
  ConstIterator cbegin() const &;
  ConstIterator cend() const &;

  ConstIterator begin() const &;
  ConstIterator end() const &;

  ConstIterator beginFrom(const ConstIterator &iter) const &;

  // Modifiers
  void insert(const Triangle<T> &tr);
  void clear();
  void setNodeCapacity(std::size_t newCap);

  // Capacity
  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 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);

private:
  void expandingInsert(const Triangle<T> &tr);
  void tryExpandRight(Axis axis, const BoundBox<T> &trianBB);
  void tryExpandLeft(Axis axis, const BoundBox<T> &trianBB);

  void nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv = false);
  bool isDivisable(const Node<T> *node);
  void subdivide(Node<T> *node);

public:
  struct ContainerPtr final
  {
    Container<T> cont;
    const Container<T> *operator->() const;
  };

  class ConstIterator final
  {
  public:
    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;

  private:
    const KdTree<T> *tree_;
    const Node<T> *node_;
    std::queue<const Node<T> *> fifo_;

  public:
    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 ConstIterator beginFrom(const ConstIterator &iter);
  };
};

```

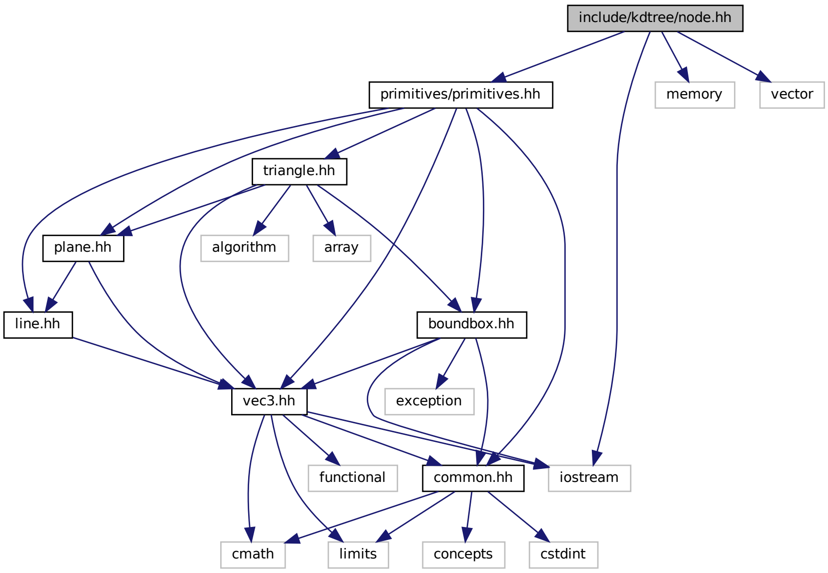
```
//============================================================================================
//                                    KdTree definitions
//============================================================================================

template <std::floating_point T>
KdTree<T>::KdTree(std::initializer_list<Triangle<T>> il)
{
  for (const auto &tr : il)
    insert(tr);
}

template <std::floating_point T>
KdTree<T>::KdTree(const KdTree<T> &tree)
{
  // temporary solution
  for (const auto &tr : tree.triangles_)
    insert(tr);
}

template <std::floating_point T>
KdTree<T>::~KdTree()
{
  clear();
}

template <std::floating_point T>
KdTree<T> &KdTree<T>::operator=(const KdTree<T> &tree)
{
  KdTree tmp{tree};
  operator=(std::move(tmp));
  return *this;
}

// ConstIterators
template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::cbegin() const &
{
  return ConstIterator{this, root_.get()};
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::cend() const &
{
  return ConstIterator{this, nullptr};
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::begin() const &
{
  return cbegin();
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::end() const &
{
  return cend();
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::beginFrom(
  const typename KdTree<T>::ConstIterator &iter) const &
{
  return KdTree<T>::ConstIterator::beginFrom(iter);
}

// Modifiers
template <std::floating_point T>
void KdTree<T>::insert(const Triangle<T> &tr)
{
  if (nullptr == root_)
  {
    root_ = std::unique_ptr<Node<T»{new Node<T>{T{}, Axis::NONE, tr.boundBox(), {0}}};
    triangles_.push_back(tr);
    return;
  }

  if (!tr.belongsTo(root_->boundBox))
    expandingInsert(tr);
  else
  {
    auto index = triangles_.size();
    triangles_.push_back(tr);
    nonExpandingInsert(root_.get(), tr, index);
  }
}

template <std::floating_point T>
```

```
void KdTree<T>::clear()
{
  if (nullptr == root_)
    return;

  std::stack<std::unique_ptr<Node<T» *> stack{};
  stack.push(&root_);

  while (!stack.empty())
  {
    auto *curNode = stack.top();
    auto *right = &curNode->get()->right;
    auto *left = &curNode->get()->left;

    if ((nullptr == *right) && (nullptr == *left))
    {
      curNode->reset();
      stack.pop();
      continue;
    }

    stack.push(right);
    stack.push(left);
  }
}

template <std::floating_point T>
void KdTree<T>::setNodeCapacity(std::size_t newCap)
{
  nodeCapacity_ = newCap;
}

// Capacity
template <std::floating_point T>
bool KdTree<T>::empty() const
{
  return triangles_.empty();
}

template <std::floating_point T>
std::size_t KdTree<T>::size() const
{
  return triangles_.size();
}

template <std::floating_point T>
std::size_t KdTree<T>::nodeCapacity() const
{
  return nodeCapacity_;
}

template <std::floating_point T>
const Triangle<T> &KdTree<T>::triangleByIndex(Index index) const &
{
  return triangles_[index];
}

template <std::floating_point T>
void KdTree<T>::dumpRecursive(std::ostream &ost) const
{
  ost « "digraph kdtree {" « std::endl;
  if (root_)
    root_->dumpRecursive(ost);
  ost « "}" « std::endl;
}

template <std::floating_point T>
bool KdTree<T>::isOnPosSide(Axis axis, T separator, const Triangle<T> &tr)
{
  return isOnSide(axis, separator, tr, std::greater<T>{});
}

template <std::floating_point T>
bool KdTree<T>::isOnNegSide(Axis axis, T separator, const Triangle<T> &tr)
{
  return isOnSide(axis, separator, tr, std::less<T>{});
}

template <std::floating_point T>
bool KdTree<T>::isOnSide(Axis axis, T separator, const Triangle<T> &tr,
                         std::function<bool(T, T)> comparator)
{
  if (Axis::NONE == axis)
    return false;

  auto axisIdx = static_cast<size_t>(axis);
  return std::all_of(tr.begin(), tr.end(),
                     [&](auto &&v) { return comparator(v[axisIdx], separator); });
}

template <std::floating_point T>
void KdTree<T>::expandingInsert(const Triangle<T> &tr)
{
  auto trianBB = tr.boundBox();
  auto index = triangles_.size();
  triangles_.push_back(tr);

  for (auto axis : {Axis::X, Axis::Y, Axis::Z})
    tryExpandRight(axis, trianBB);

  for (auto axis : {Axis::X, Axis::Y, Axis::Z})
    tryExpandLeft(axis, trianBB);

  root_->indicies.push_back(index);
}

template <std::floating_point T>
void KdTree<T>::tryExpandRight(Axis axis, const BoundBox<T> &trianBB)
{
  const auto &rootBB = root_->boundBox;
  if (trianBB.max(axis) <= rootBB.max(axis))
    return;

  auto newRightBB = rootBB;
  newRightBB.min(axis) = rootBB.max(axis);
  newRightBB.max(axis) = trianBB.max(axis);

  auto newRootBB = rootBB;
  newRootBB.max(axis) = newRightBB.max(axis);

  std::unique_ptr<Node<T>» newRight{new Node<T>{T{}, Axis::NONE, newRightBB}};
  std::unique_ptr<Node<T>» newRoot{new Node<T>{rootBB.max(axis), axis, newRootBB}};

  newRoot->right = std::move(newRight);
  newRoot->left = std::move(root_);

  root_ = std::move(newRoot);
}

template <std::floating_point T>
void KdTree<T>::tryExpandLeft(Axis axis, const BoundBox<T> &trianBB)
{
  const auto &rootBB = root_->boundBox;
  if (trianBB.min(axis) >= rootBB.min(axis))
    return;

  BoundBox<T> newLeftBB = rootBB;
  newLeftBB.max(axis) = rootBB.min(axis);
  newLeftBB.min(axis) = trianBB.min(axis);

  BoundBox<T> newRootBB = rootBB;
  newRootBB.min(axis) = newLeftBB.min(axis);

  std::unique_ptr<Node<T>» newLeft{new Node<T>{T{}, Axis::NONE, newLeftBB}};
  std::unique_ptr<Node<T>» newRoot{new Node<T>{rootBB.min(axis), axis, newRootBB}};

  newRoot->left = std::move(newLeft);
  newRoot->right = std::move(root_);

  root_ = std::move(newRoot);
}

template <std::floating_point T>
void KdTree<T>::nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv)
{
  auto curNode = node;
  while (true)
  {
    if (isOnPosSide(curNode->sepAxis, curNode->separator, tr))
      curNode = curNode->right.get();
    else if (isOnNegSide(curNode->sepAxis, curNode->separator, tr))
      curNode = curNode->left.get();
    else
      break;
  }

  curNode->indicies.push_back(index);
  if (isDivisable(curNode) && (!isSubdiv))
    subdivide(curNode);
}

template <std::floating_point T>
bool KdTree<T>::isDivisable(const Node<T> *node)
{
  return (node->indicies.size() > nodeCapacity_) && (node->sepAxis == Axis::NONE);
}

template <std::floating_point T>
void KdTree<T>::subdivide(Node<T> *node)
{
  const auto &nodeBB = node->boundBox;
  auto axis = node->sepAxis = nodeBB.getMaxDim();
  auto sep = node->separator = nodeBB.min(axis) + (nodeBB.max(axis) - nodeBB.min(axis)) / 2;

  auto newRightBB = nodeBB;
  auto newLeftBB = nodeBB;

  newRightBB.min(axis) = newLeftBB.max(axis) = sep;
  node->right.reset(new Node<T>{T{}, Axis::NONE, newRightBB});
  node->left.reset(new Node<T>{T{}, Axis::NONE, newLeftBB});

  auto indicies = node->indicies;
  node->indicies.clear();

  for (auto index : indicies)
    nonExpandingInsert(node, triangles_[index], index, /* isSubdiv = */ true);
}

//=============================================================================================
//                              KdTree::ContainerPtr definitions
//=============================================================================================

template <std::floating_point T>
const Container<T> *KdTree<T>::ContainerPtr::operator->() const
{
  return &cont;
}

//=============================================================================================
//                              KdTree::ConstIterator definitions
//=============================================================================================

template <std::floating_point T>
KdTree<T>::ConstIterator::ConstIterator(const KdTree<T> *tree, const Node<T> *node)
  : tree_(tree), node_(node), fifo_({node})
{}

template <std::floating_point T>
typename KdTree<T>::ConstIterator &KdTree<T>::ConstIterator::operator++()
{
  if (0 == fifo_.size())
    return *this;

  auto fifoEntry = fifo_.front();
  fifo_.pop();

  if (Axis::NONE != fifoEntry->sepAxis)
  {
    if (nullptr != fifoEntry->left)
      fifo_.push(fifoEntry->left.get());
    if (nullptr != fifoEntry->right)
      fifo_.push(fifoEntry->right.get());
  }

  node_ = (0 == fifo_.size()) ? nullptr : fifo_.front();
  return *this;
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::operator++(int)
{
  auto tmp = *this;
  operator++();
  return tmp;
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator::reference KdTree<T>::ConstIterator::operator*() const
{
  return Container<T>{tree_, node_};
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator::pointer KdTree<T>::ConstIterator::operator->() const
{
  return ContainerPtr{{tree_, node_}};
}

template <std::floating_point T>
bool KdTree<T>::ConstIterator::operator==(const KdTree<T>::ConstIterator &lhs) const
{
  return (tree_ == lhs.tree_) && (node_ == lhs.node_);
}

template <std::floating_point T>
bool KdTree<T>::ConstIterator::operator!=(const KdTree<T>::ConstIterator &lhs) const
{
  return !operator==(lhs);
}

template <std::floating_point T>
typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::beginFrom(
  const typename KdTree<T>::ConstIterator &iter)
{
  return ConstIterator{iter.tree_, iter.node_};
}

} // namespace geom::kdtree

#endif // __INCLUDE_KDTREE_KDTREE_HH__
```

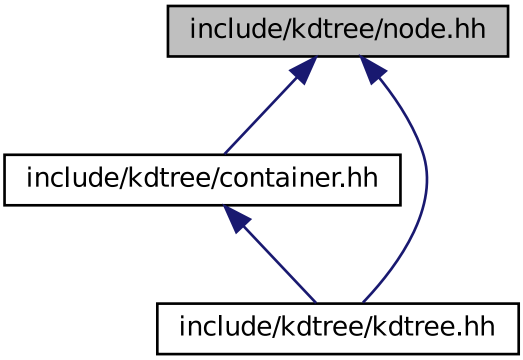
6.11 include/kdtree/node.hh File Reference

```
#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

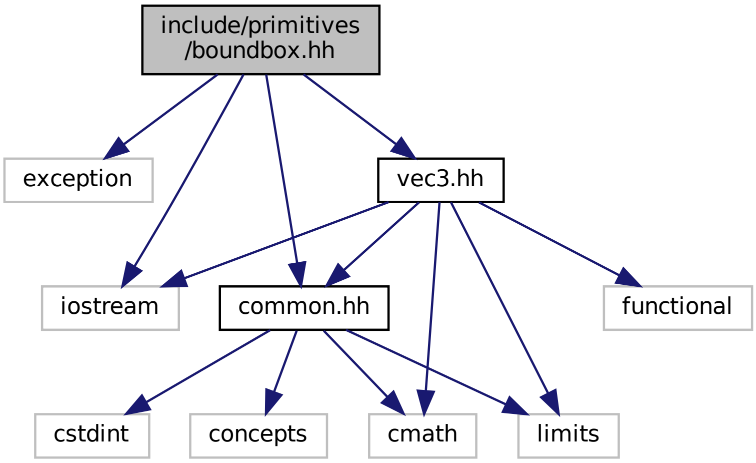
6.12 node.hh

```
#ifndef __INCLUDE_KDTREE_NODE_HH__
#define __INCLUDE_KDTREE_NODE_HH__

#include <iostream>
#include <memory>
#include <vector>

#include "primitives/primitives.hh"

namespace geom::kdtree
{

using Index = std::size_t;

template <std::floating_point T>
struct Node final
{
  T separator{};            // separator's coordinate on separation axis
  Axis sepAxis{Axis::NONE}; // separation axis
  BoundBox<T> boundBox{};
  std::vector<Index> indicies{};

  std::unique_ptr<Node> left{nullptr};
  std::unique_ptr<Node> right{nullptr};

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

  void dumpRecursive(std::ostream &ost) const;
};

template <std::floating_point T>
void Node<T>::dumpRecursive(std::ostream &ost) const
{
  ost « reinterpret_cast<std::uintptr_t>(this)
      « " [shape=box,label=\"axis: " « static_cast<int>(sepAxis) « ",\\n"
      « boundBox « ",\\nvec: {";

  for (auto elem : indicies)
    ost « elem « " ";

  ost « "}\"];" « std::endl;

  if (left)
  {
    left->dumpRecursive(ost);
    ost « reinterpret_cast<std::uintptr_t>(this) « " -> "
        « reinterpret_cast<std::uintptr_t>(left.get()) « " [label=\"L\"];" « std::endl;
  }
  if (right)
  {
    right->dumpRecursive(ost);
    ost « reinterpret_cast<std::uintptr_t>(this) « " -> "
        « reinterpret_cast<std::uintptr_t>(right.get()) « " [label=\"R\"];" « std::endl;
  }
}

} // namespace geom::kdtree

#endif // __INCLUDE_KDTREE_NODE_HH__
```

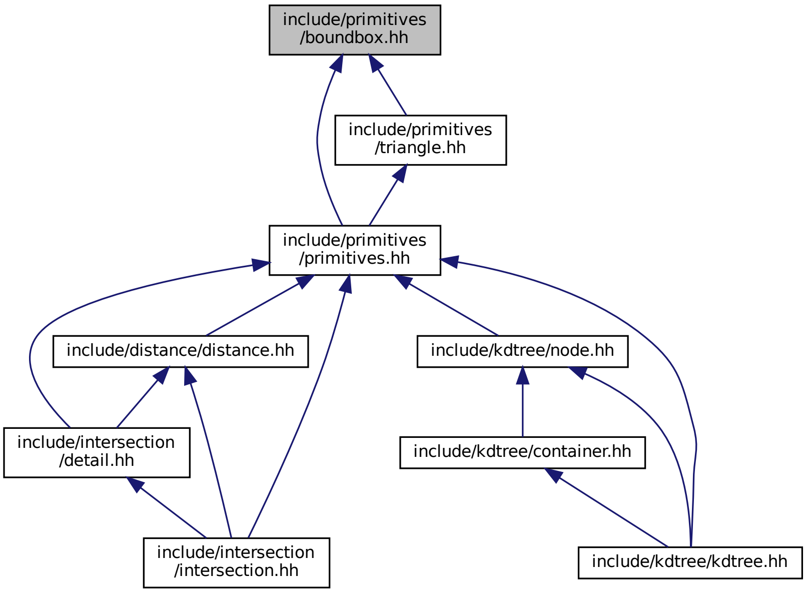
6.13 include/primitives/boundbox.hh File Reference

```
#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>
  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 49 of file boundbox.hh.

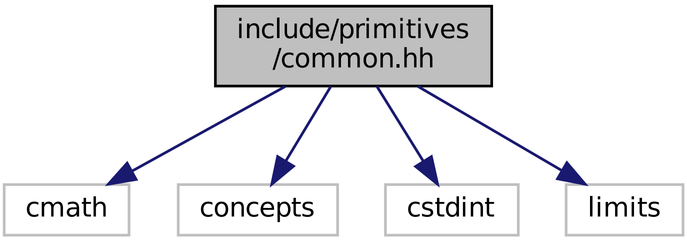
6.14 boundbox.hh

```
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 final
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   bool operator==(const BoundBox &rhs) const;
00039   bool operator!=(const BoundBox &rhs) const;
00040 };
00041
00042 template <std::floating_point T>
00043 bool BoundBox<T>::belongsTo(const BoundBox<T> &bb)
00044 {
00045   return (minX >= bb.minX) && (minY >= bb.minY) && (minZ >= bb.minZ) && (maxX <= bb.maxX) &&
00046          (maxY <= bb.maxY) && (maxZ <= bb.maxZ);
00047 }
00048
00049 #define BBFILL(minmax)                                                                              \
00050   do                                                                                                \
00051   {                                                                                                 \
00052     switch (axis)                                                                                   \
00053     {                                                                                               \
    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)

template <std::floating_point T>
T &BoundBox<T>::min(Axis axis) &
{
  BBFILL(min);
}

template <std::floating_point T>
T &BoundBox<T>::max(Axis axis) &
{
  BBFILL(max);
}

template <std::floating_point T>
T BoundBox<T>::min(Axis axis) &&
{
  BBFILL(min);
}

template <std::floating_point T>
T BoundBox<T>::max(Axis axis) &&
{
  BBFILL(max);
}

template <std::floating_point T>
T BoundBox<T>::min(Axis axis) const &
{
  BBFILL(min);
}

template <std::floating_point T>
T BoundBox<T>::max(Axis axis) const &
{
  BBFILL(max);
}

#undef BBFILL

template <std::floating_point T>
Axis BoundBox<T>::getMaxDim() const
{
  return std::max({Axis::X, Axis::Y, Axis::Z}, [this](auto lhs, auto rhs) {
    return (this->max(lhs) - this->min(lhs)) < (this->max(rhs) - this->min(rhs));
  });
}

template <std::floating_point T>
bool BoundBox<T>::operator==(const BoundBox &rhs) const
{
  return isEqualThreshold(minX, rhs.minX) && isEqualThreshold(maxX, rhs.maxX) &&
         isEqualThreshold(minY, rhs.minY) && isEqualThreshold(maxY, rhs.maxY) &&
         isEqualThreshold(minZ, rhs.minZ) && isEqualThreshold(maxY, rhs.maxY);
}

template <std::floating_point T>
bool BoundBox<T>::operator!=(const BoundBox &rhs) const
{
  return !operator==(rhs);
}

template <std::floating_point T>
std::ostream &operator«(std::ostream &ost, const BoundBox<T> &bb)
{
  ost « "BB: {\\n";
  ost « "  x: [" « bb.minX « "; " « bb.maxX « "],\\n";
  ost « "  y: [" « bb.minY « "; " « bb.maxY « "],\\n";
  ost « "  z: [" « bb.minZ « "; " « bb.maxZ « "]\\n}";
  return ost;
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
```

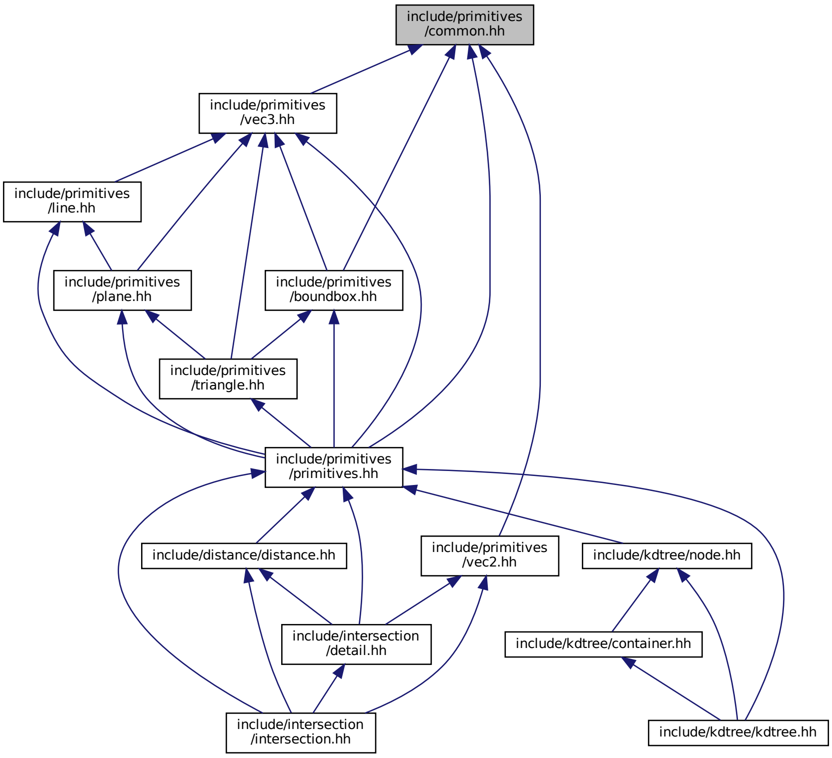
6.15 include/primitives/common.hh File Reference

```
#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 }

Functions

- template<Number T>
  bool geom::isEqualThreshold (T num1, T num2)
- template<Number T>
  bool geom::isZeroThreshold (T num)

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.

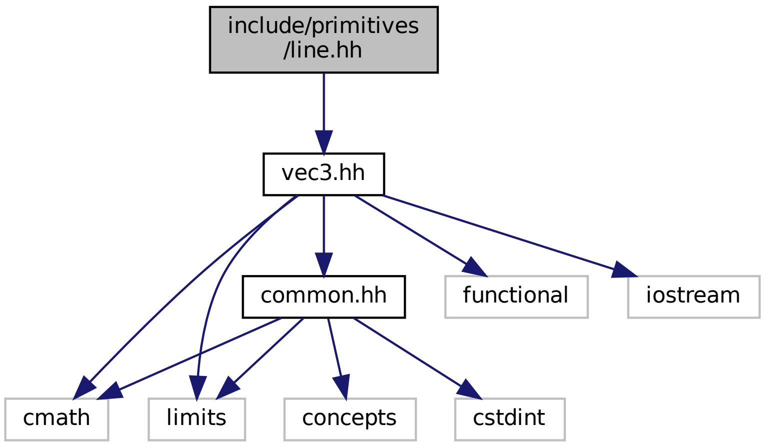
6.16 common.hh

```
#ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
#define __INCLUDE_PRIMITIVES_COMMON_HH__

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

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

template <Number T>
class ThresComp final
{
private:
  static inline T threshold_ = 1e2 * std::numeric_limits<T>::epsilon();

public:
  ThresComp() = delete;

  static void setThreshold(T thres) requires std::is_floating_point_v<T>
  {
    threshold_ = thres;
  }

  static T getThreshold() requires std::is_floating_point_v<T>
  {
    return threshold_;
  }

  static void scaleThreshold(T factor) requires std::is_floating_point_v<T>
  {
    threshold_ *= factor;
  }

  static void resetThreshold() requires std::is_floating_point_v<T>
  {
    threshold_ = std::numeric_limits<T>::epsilon();
  }

  static bool isEqual(T lhs, T rhs)
  {
    if constexpr (std::is_floating_point_v<T>)
      return std::abs(rhs - lhs) < threshold_;
    else
      return lhs == rhs;
  }

  static bool isZero(T num)
  {
    return isEqual(num, T{});
  }
};

template <Number T>
bool isEqualThreshold(T num1, T num2)
{
  return ThresComp<T>::isEqual(num1, num2);
}

template <Number T>
bool isZeroThreshold(T num)
{
  return ThresComp<T>::isZero(num);
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_COMMON_HH__
```

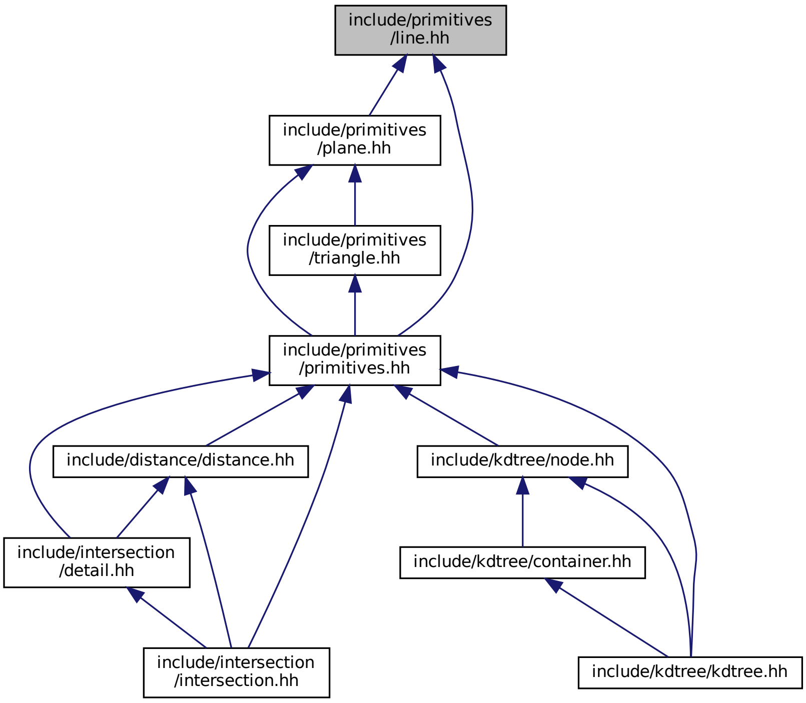
6.17 include/primitives/line.hh File Reference

`#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.

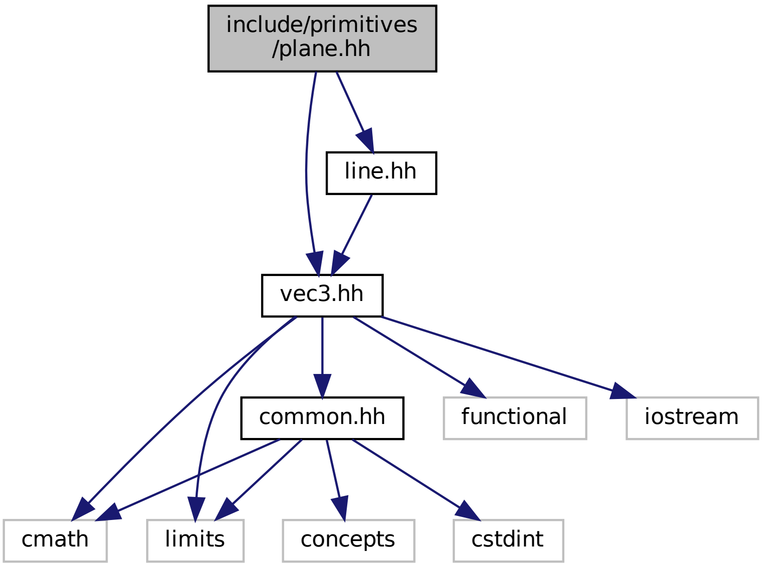
6.18 line.hh

```
#ifndef __INCLUDE_PRIMITIVES_LINE_HH__
#define __INCLUDE_PRIMITIVES_LINE_HH__

#include "vec3.hh"

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

namespace geom
{

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

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

  /**
   * @brief Line equality operator
   *
   * @tparam T - floating point type of coordinates
   * @param[in] rhs 2nd line
   * @return true if lines are equal
   * @return false if lines are not equal
   */
  bool operator==(const Line &rhs) const;

  /**
   * @brief Line inequality operator
   *
   * @tparam T - floating point type of coordinates
   * @param[in] rhs 2nd line
   * @return true if lines are not equal
   * @return false if lines are equal
   */
  bool operator!=(const Line &rhs) const;

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

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

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

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

  /**
   * @brief Get point on line by parameter t
   *
   * @tparam nType numeric type
   * @param[in] t point paramater from line’s equation
   * @return Vec3<T> Point related to parameter
   */
  template <Number nType>
  Vec3<T> getPoint(nType t) const;

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

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

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

  /**
   * @brief Checks is *this is skew with another line
   *
   * @param[in] line const reference to another line
   * @return true if lines are skew
   * @return false if lines are not skew
   */
  bool isSkew(const Line<T> &line) const;

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

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

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

template <std::floating_point T>
bool Line<T>::operator==(const Line &rhs) const
{
  return isEqual(rhs);
}

template <std::floating_point T>
bool Line<T>::operator!=(const Line &rhs) const
{
  return !operator==(rhs);
}

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

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

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

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

template <std::floating_point T>
template <Number nType>
Vec3<T> Line<T>::getPoint(nType t) const
{
  return org_ + dir_ * t;
}

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

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

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

template <std::floating_point T>
bool Line<T>::isSkew(const Line<T> &line) const
{
  auto res = triple(line.org_ - org_, dir_, line.dir_);
  return !isZeroThreshold(res);
}

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

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_LINE_HH__
```

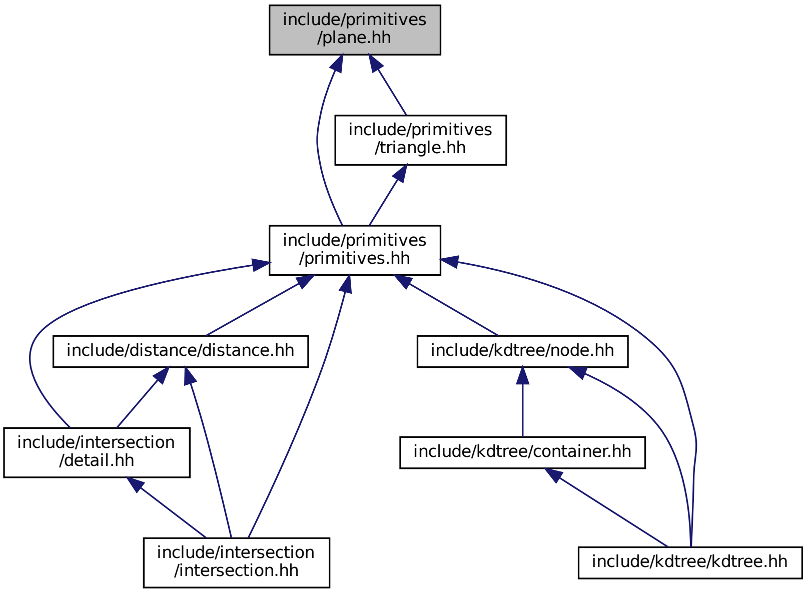
6.19 include/primitives/plane.hh File Reference

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

  Plane print operator.

6.20 plane.hh

```
#ifndef __INCLUDE_PRIMITIVES_PLANE_HH__
#define __INCLUDE_PRIMITIVES_PLANE_HH__

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

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

namespace geom
{

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

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

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

public:
  /**
   * @brief Plane equality operator
   *
   * @tparam T - floating point type of coordinates
   * @param[in] rhs 2nd plane
   * @return true if planes are equal
   * @return false if planes are not equal
   */
  bool operator==(const Plane &rhs) const;

  /**
   * @brief Plane inequality operator
   *
   * @tparam T - floating point type of coordinates
   * @param[in] rhs 2nd plane
   * @return true if planes are not equal
   * @return false if planes are equal
   */
  bool operator!=(const Plane &rhs) const;

```

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

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

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

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

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

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

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

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

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

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

/**
```

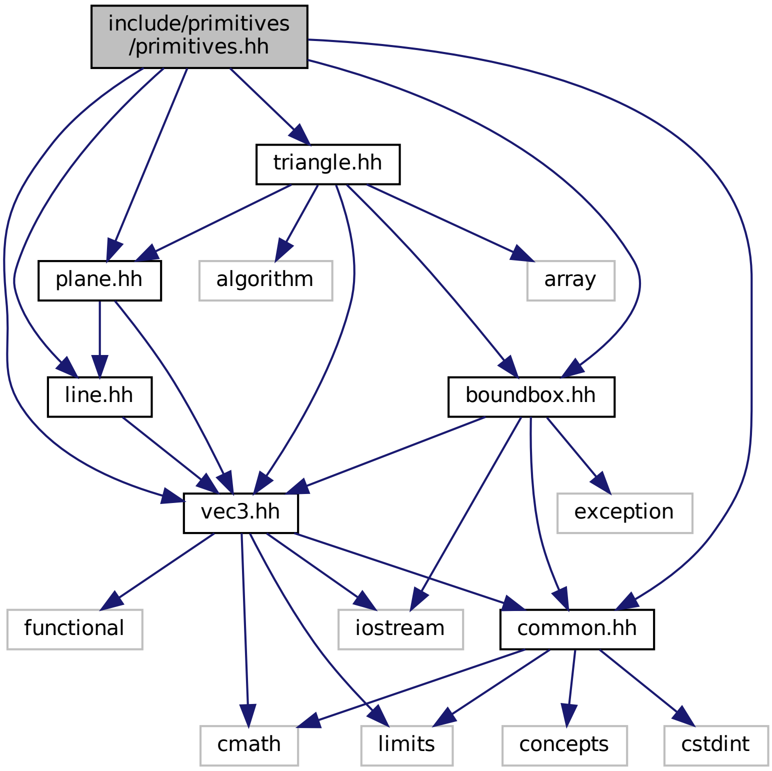
```
   * @brief Get plane form normal const plane equation
   *
   * @param[in] norm normal vector
   * @param[in] constant distance
   * @return Plane
   */
  static Plane getNormalDist(const Vec3<T> &norm, T constant);
};

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

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

template <std::floating_point T>
bool Plane<T>::operator==(const Plane &rhs) const
{
  return isEqual(rhs);
}

template <std::floating_point T>
bool Plane<T>::operator!=(const Plane &rhs) const
{
  return !operator==(rhs);
}

template <std::floating_point T>
T Plane<T>::dist() const
{
  return dist_;
}

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

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

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

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

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

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

template <std::floating_point T>
Plane<T> Plane<T>::getBy3Points(const Vec3<T> &pt1, const Vec3<T> &pt2, const Vec3<T> &pt3)
```

```
{
  return getParametric(pt1, pt2 - pt1, pt3 - pt1);
}

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

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

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

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_PLANE_HH__
```

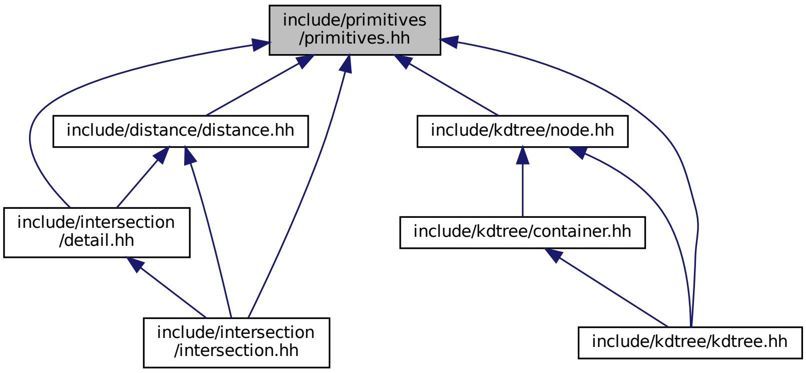
6.21 include/primitives/primitives.hh File Reference

```
#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

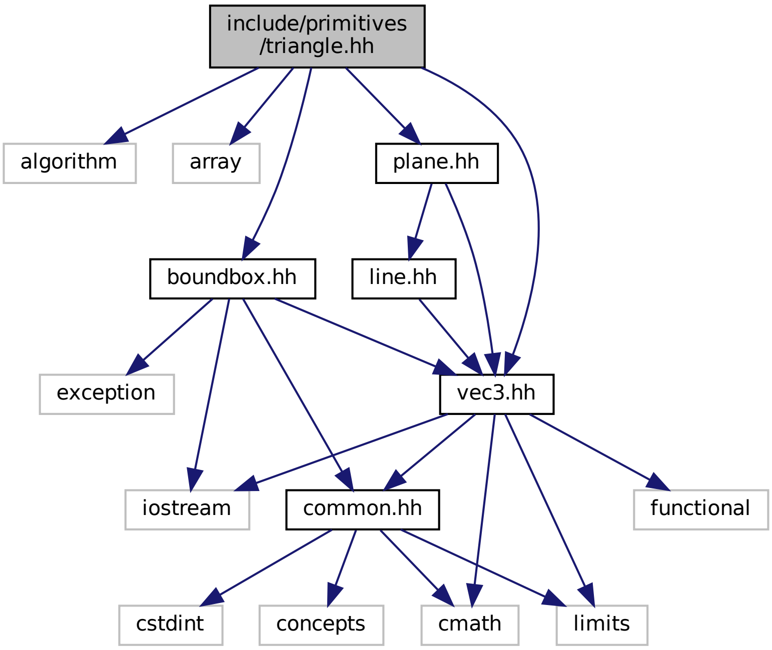
```
00001 #ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00002 #define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

```
#include "boundbox.hh"
#include "common.hh"
#include "line.hh"
#include "plane.hh"
#include "triangle.hh"
#include "vec3.hh"

#endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

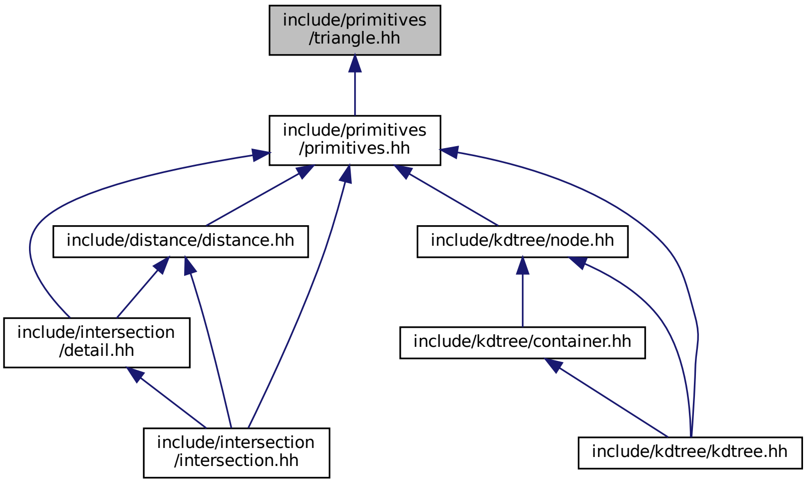
6.23 include/primitives/triangle.hh File Reference

```
#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:


include/primitives /triangle.hh
include/primitives /primitives.hh
include/distance/distance.hh
include/kdtree/node.hh
include/intersection /detail.hh
include/kdtree/container.hh
include/intersection /intersection.hh
include/kdtree/kdtree.hh

Classes

- class geom::Triangle< T >

    Triangle class implementation.

Namespaces

- geom

    line.hh Line class implementation

Functions

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

    Triangle print operator.

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

6.24 triangle.hh

```
#ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
#define __INCLUDE_PRIMITIVES_TRIANGLE_HH__

#include <algorithm>
#include <array>

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

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

namespace geom
{

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

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

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

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

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

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

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

  /**
   * @brief Get begin iterator
   * @return Iterator
   */
  Iterator begin() &;

  /**
   * @brief Get end iterator
   * @return Iterator
   */
  Iterator end() &;

  /**
   * @brief Get begin const iterator
   * @return ConstIterator
   */
  ConstIterator begin() const &;

  /**
   * @brief Get end const iterator
   * @return ConstIterator
   */
  ConstIterator end() const &;

  /**
```

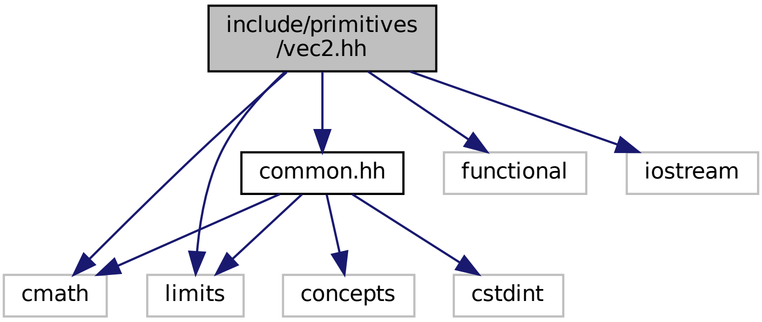
```
   * @brief Get triangle’s plane
   *
   * @return Plane<T>
   */
  Plane<T> getPlane() const;

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

  /**
   * @brief Returns triangle’s bound box
   *
   * @return BoundBox<T>
   */
  BoundBox<T> boundBox() const;

  /**
   * @brief Checks if this Triangle belongs to BoundBox
   *
   * @param[in] bb BoundBox
   * @return true if Triangle belongs to BoundBox
   * @return false if Triangle doesn’t belong to BoundBox
   */
  bool belongsTo(const BoundBox<T> &bb) const;
};

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

  ost « "}";

  return ost;
}

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

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

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

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

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

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

template <std::floating_point T>
typename Triangle<T>::Iterator Triangle<T>::begin() &
```

```
{
  return vertices_.begin();
}

template <std::floating_point T>
typename Triangle<T>::Iterator Triangle<T>::end() &
{
  return vertices_.end();
}

template <std::floating_point T>
typename Triangle<T>::ConstIterator Triangle<T>::begin() const &
{
  return vertices_.begin();
}

template <std::floating_point T>
typename Triangle<T>::ConstIterator Triangle<T>::end() const &
{
  return vertices_.end();
}

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

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

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

template <std::floating_point T>
BoundBox<T> Triangle<T>::boundBox() const
{
  auto minMaxX = std::minmax({vertices_[0].x, vertices_[1].x, vertices_[2].x});
  auto minMaxY = std::minmax({vertices_[0].y, vertices_[1].y, vertices_[2].y});
  auto minMaxZ = std::minmax({vertices_[0].z, vertices_[1].z, vertices_[2].z});

  return {
    minMaxX.first - ThresComp<T>::getThreshold(), minMaxX.second + ThresComp<T>::getThreshold(),
    minMaxY.first - ThresComp<T>::getThreshold(), minMaxY.second + ThresComp<T>::getThreshold(),
    minMaxZ.first - ThresComp<T>::getThreshold(), minMaxZ.second + ThresComp<T>::getThreshold()};
}

template <std::floating_point T>
bool Triangle<T>::belongsTo(const BoundBox<T> &bb) const
{
  return boundBox().belongsTo(bb);
}

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__
```

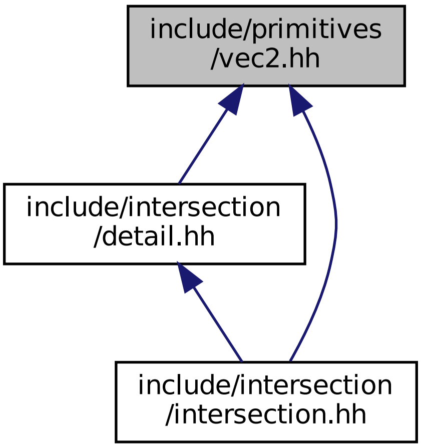
6.25 include/primitives/vec2.hh File Reference

```
#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>
  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

```
#ifndef __INCLUDE_PRIMITIVES_VEC2_HH__
#define __INCLUDE_PRIMITIVES_VEC2_HH__

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

#include "common.hh"

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

namespace geom
{

/**
 * @class Vec2
 * @brief Vec2 class realization
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
struct Vec2 final
{
  /**
   * @brief Vec2 coordinates
   */
  T x{}, y{};

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

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

  /**
   * @brief Vec2 equality operator
   *
   * @tparam T vector template parameter
   * @param[in] rhs second vector
   * @return true if vectors are equal
   * @return false otherwise
   */
  bool operator==(const Vec2 &rhs) const;

  /**
   * @brief Vec2 equality operator
   *
   * @tparam T vector template parameter
   * @param[in] rhs second vector
   * @return true if vectors are not equal
   * @return false otherwise
   */
  bool operator!=(const Vec2 &rhs) const;

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

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

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

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

  /**
   * @brief Overloaded /= by number operator
   *
   * @tparam nType numeric type of value to divide by
   * @param[in] val value to divide by
   * @return Vec2& reference to vector instance
   *
```

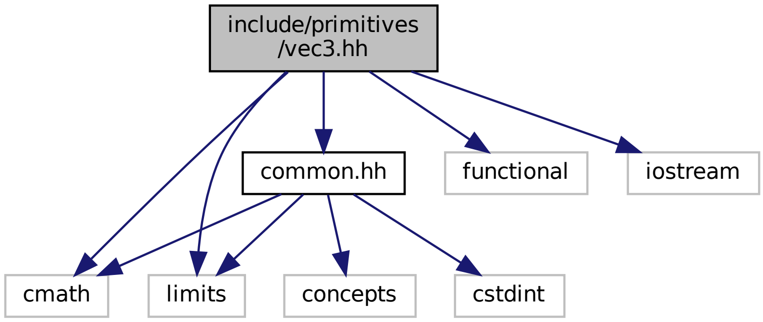
```
     * @warning Does not check if val equals 0
     */
    template <Number nType>
    Vec2 &operator/=(nType val);

    /**
     * @brief Dot product function
     *
     * @param rhs vector to dot product with
     * @return T dot product of two vectors
     */
    T dot(const Vec2 &rhs) const;

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

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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

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

template <std::floating_point T>
bool Vec2<T>::operator==(const Vec2 &rhs) const
{
  return isEqual(rhs);
}

template <std::floating_point T>
bool Vec2<T>::operator!=(const Vec2 &rhs) const
{
  return !operator==(rhs);
}

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

  return *this;
}

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

  return *this;
}

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

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

  return *this;
}

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

  return *this;
}

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

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

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

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

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

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

template <std::floating_point T>
T &Vec2<T>::operator[](std::size_t i) &
{
  switch (i % 2)
  {
  case 0:
    return x;
  case 1:
    return y;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

template <std::floating_point T>
T Vec2<T>::operator[](std::size_t i) const &
{
  switch (i % 2)
  {
  case 0:
    return x;
  case 1:
    return y;
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

template <std::floating_point T>
T &&Vec2<T>::operator[](std::size_t i) &&
{
  switch (i % 2)
  {
  case 0:
    return std::move(x);
  case 1:
    return std::move(y);
  default:
    throw std::logic_error{"Impossible case in operator[]\n"};
  }
}

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

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

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

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_VEC2_HH__
```

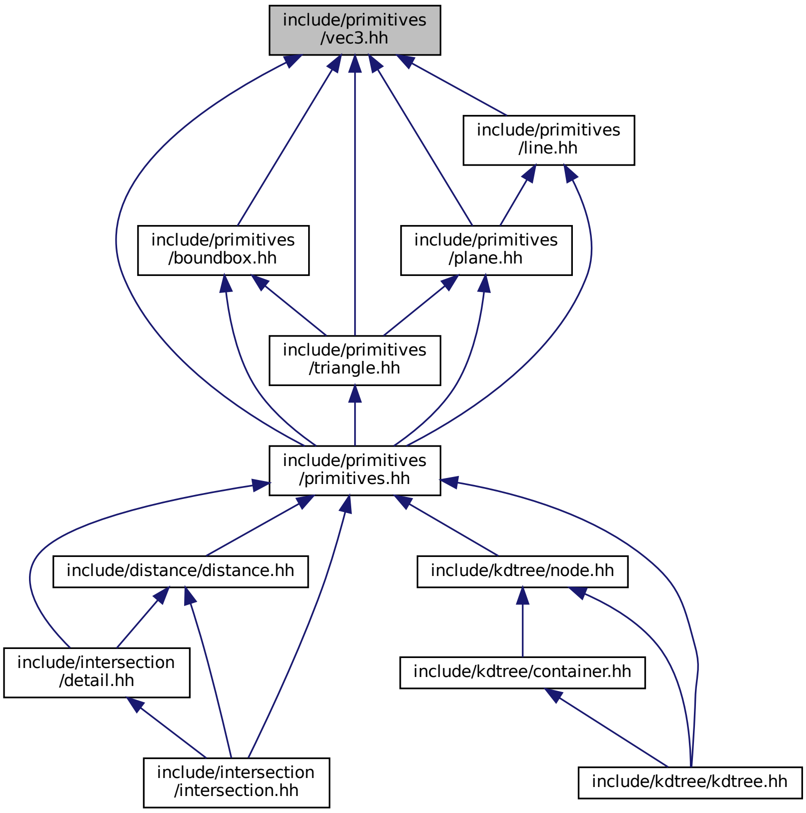
6.27 include/primitives/vec3.hh File Reference

```
#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:


include/primitives /vec3.hh
include/primitives /line.hh
include/primitives /boundbox.hh
include/primitives /plane.hh
include/primitives /triangle.hh
include/primitives /primitives.hh
include/distance/distance.hh
include/kdtree/node.hh
include/intersection /detail.hh
include/kdtree/container.hh
include/intersection /intersection.hh
include/kdtree/kdtree.hh

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>
  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

```
#ifndef __INCLUDE_PRIMITIVES_VEC3_HH__
#define __INCLUDE_PRIMITIVES_VEC3_HH__

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

#include "common.hh"

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

namespace geom
```

```
{

/**
 * @class Vec3
 * @brief Vec3 class realization
 *
 * @tparam T - floating point type of coordinates
 */
template <std::floating_point T>
struct Vec3 final
{
  /**
   * @brief Vec3 coordinates
   */
  T x{}, y{}, z{};

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

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

  /**
   * @brief Vec3 equality operator
   *
   * @tparam T vector template parameter
   * @param[in] rhs second vector
   * @return true if vectors are equal
   * @return false otherwise
   */
  bool operator==(const Vec3 &rhs) const;

  /**
   * @brief Vec3 inequality operator
   *
   * @tparam T vector template parameter
   * @param[in] rhs second vector
   * @return true if vectors are not equal
   * @return false otherwise
   */
  bool operator!=(const Vec3 &rhs) const;

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

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

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

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

```

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

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

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

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

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

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

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

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

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

/**
 * @brief Overloaded operator [] (rvalue 'this' version)
 * To get access to coordinates
 * @param i index of coordinate (0 - x, 1 - y, 2 - z)
 * @return T coordinate value
 *
 * @note Coordinates calculated by mod 3
 */
T &&operator[](std::size_t i) &&;

/**
```

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

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

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

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

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

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

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

```
{
  Vec3<T> res{lhs};
  res *= val;
  return res;
}

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

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

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

/**
 * @brief Triple product function
 *
 * @tparam T vector template parameter
 * @param[in] v1 first vector
 * @param[in] v2 second vector
 * @param[in] v3 third vector
 * @return T triple production
 */
template <std::floating_point T>
T triple(const Vec3<T> &v1, const Vec3<T> &v2, const Vec3<T> &v3)
{
  return dot(v1, cross(v2, v3));
}

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

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

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

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

template <std::floating_point T>
bool Vec3<T>::operator!=(const Vec3 &rhs) const
{
  return !operator==(rhs);
}

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

  return *this;
}

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

  return *this;
}

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

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

  return *this;
}

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

  return *this;
}

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

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

template <std::floating_point T>
```

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

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

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

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

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

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

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

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

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

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

} // namespace geom

#endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```