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 Triangle

  Triangle class implementation.
- class Vec2

  Vec2 class realization.
- class Vec3

  Vec3 class realization.

Typedefs

- using Vec2D = Vec2< double >
- using Vec2F = Vec2< float >
- using Vec3D = Vec3< double >
- using Vec3F = Vec3< float >

Enumerations

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

Functions

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

  Calculates signed distance between point and plane.
- template<std::floating_point T>
  bool isIntersect (const Triangle< T > &tr1, const Triangle< T > &tr2)

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

  Intersect 2 planes and return result of intersection.
- template<std::floating_point T>
  std::variant< std::monostate, Vec3< T >, Line< T > > intersect (const Line< T > &l1, const Line< T > &l2)

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

  Line print operator.
- template<std::floating_point T>
  bool operator== (const Line< T > &lhs, const Line< T > &rhs)

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

  Plane equality operator.
- template<std::floating_point T>
  std::ostream & operator<< (std::ostream &ost, const Plane< T > &pl)

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

  Triangle print operator.
- template<std::floating_point T>
  std::istream & operator>> (std::istream &ist, Triangle< T > &tr)
- template<std::floating_point T>
  Vec2< T > operator+ (const Vec2< T > &lhs, const Vec2< T > &rhs)

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

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

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

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

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

Dot product function.

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

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

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

  Vec2 print operator.
- template<std::floating_point T>
  Vec3< T > operator+ (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Overloaded + operator.
- template<std::floating_point T>
  Vec3< T > operator- (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Overloaded - operator.
- template<Number nT, std::floating_point T>
  Vec3< T > operator* (const nT &val, const Vec3< T > &rhs)

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec3< T > operator* (const Vec3< T > &lhs, const nT &val)

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec3< T > operator/ (const Vec3< T > &lhs, const nT &val)

  Overloaded divide by value operator.
- template<std::floating_point T>
  T dot (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Dot product function.
- template<std::floating_point T>
  Vec3< T > cross (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Cross product function.
- template<std::floating_point T>
  T triple (const Vec3< T > &v1, const Vec3< T > &v2, const Vec3< T > &v3)

  Triple product function.
- template<std::floating_point T>
  bool operator== (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Vec3 equality operator.
- template<std::floating_point T>
  bool operator!= (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Vec3 inequality operator.
- template<std::floating_point T>
  std::ostream & operator<< (std::ostream &ost, const Vec3< T > &vec)

  Vec3 print operator.
- template<std::floating_point T>
  std::istream & operator>> (std::istream &ist, Vec3< T > &vec)

  Vec3 scan operator.

Variables

- template<class T >
  concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>

  Useful concept which represents floating point and integral types.

4.1.1 Detailed Description

line.hh Line class implementation

triangle.hh Triangle class implementation

Plane class implementation.

4.1.2 Typedef Documentation

4.1.2.1 Vec2D

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

Definition at line 367 of file vec2.hh.

4.1.2.2 Vec2F

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

Definition at line 368 of file vec2.hh.

4.1.2.3 Vec3D

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

Definition at line 413 of file vec3.hh.

4.1.2.4 Vec3F

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

Definition at line 414 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 18 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::helperMollerHaines(), geom::detail::isIntersectValidInvalid(), and geom::detail::isOnOneSide().

4.1.4.2 isIntersect()

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

Checks intersection of 2 triangles.

Template Parameters

T	- floating point type of coordinates

Parameters

tr1	first triangle
tr2	second triangle

Returns

true if triangles are intersect

false if triangles are not intersect

Definition at line 156 of file intersection.hh.

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

4.1.4.3 intersect() [1/2]

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

Intersect 2 planes and return result of intersection.

Common intersection case (parallel planes case is trivial):

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

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

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

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

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

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

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

$$\overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1 \Leftrightarrow \overrightarrow{n}_1 \cdot (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) = d_1 \Leftrightarrow a + b \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 = d_1$$
$$\overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2 \Leftrightarrow \overrightarrow{n}_2 \cdot (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) = d_2 \Leftrightarrow a \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 + b = d_2$$

Let's find a and b:

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

Intersection line equation:

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

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

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

Definition at line 188 of file intersection.hh.

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

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

4.1.4.4 intersect() [2/2]

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

Intersect 2 lines and return result of intersection.

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

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

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

Let's equate left sides:

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

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

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

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

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

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

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

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

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

Definition at line 215 of file intersection.hh.

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

4.1.4.5 operator==() [1/5]

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

Definition at line 116 of file boundbox.hh.

References geom::Vec3< T >::isNumEq(), geom::BoundBox< T >::maxX, geom::BoundBox< T >::maxY, geom::BoundBox< T >::minX, geom::BoundBox< T >::minY, and geom::BoundBox< T >::minZ.

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

4.1.4.6 operator<<() [1/6]

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

Definition at line 124 of file boundbox.hh.

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

4.1.4.7 operator<<() [2/6]

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

Line print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

std::ostream& modified ostream instance

Definition at line 117 of file line.hh.

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

4.1.4.8 operator==() [2/5]

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

Line equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

true if lines are equal

false if lines are not equal

Definition at line 133 of file line.hh.

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

4.1.4.9 operator==() [3/5]

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

Plane equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

true if planes are equal

false if planes are not equal

Definition at line 143 of file plane.hh.

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

4.1.4.10 operator<<() [3/6]

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

Plane print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

std::ostream& modified ostream instance

Definition at line 157 of file plane.hh.

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

4.1.4.11 operator<<() [4/6]

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

Triangle print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

std::ostream& modified ostream instance

Definition at line 106 of file triangle.hh.

4.1.4.12 operator>>() [1/2]

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

Definition at line 118 of file triangle.hh.

4.1.4.13 operator+() [1/2]

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

Overloaded + operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec2<T> sum of two vectors

Definition at line 234 of file vec2.hh.

4.1.4.14 operator-() [1/2]

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

Overloaded - operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec2<T> res of two vectors

Definition at line 250 of file vec2.hh.

4.1.4.15 operator*() [1/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec2<T> result vector

Definition at line 267 of file vec2.hh.

4.1.4.16 operator*() [2/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec2<T> result vector

Definition at line 284 of file vec2.hh.

4.1.4.17 operator/() [1/2]

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

```
            const Vec2< T > & lhs,
            const nT & val )
```

Overloaded divide by value operator.

Template Parameters

nT	type of value to divide by
T	vector template parameter

Parameters

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

Returns

Vec2<T> result vector

Definition at line 301 of file vec2.hh.

4.1.4.18 dot() [1/2]

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

Dot product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T dot production

Definition at line 317 of file vec2.hh.

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

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

4.1.4.19 operator==() [4/5]

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

Vec2 equality operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

true if vectors are equal

false otherwise

Definition at line 332 of file vec2.hh.

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

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

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

Vec2 inequality operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

true if vectors are not equal

false otherwise

Definition at line 347 of file vec2.hh.

4.1.4.21 operator<<() [5/6]

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

Vec2 print operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

std::ostream& modified stream instance

Definition at line 361 of file vec2.hh.

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

4.1.4.22 operator+() [2/2]

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

Overloaded + operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec3<T> sum of two vectors

Definition at line 236 of file vec3.hh.

4.1.4.23 operator-() [2/2]

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

Overloaded - operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vec3<T> res of two vectors

Definition at line 252 of file vec3.hh.

4.1.4.24 operator*() [3/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 269 of file vec3.hh.

4.1.4.25 operator*() [4/4]

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

Overloaded multiple by value operator.

Template Parameters

nT	type of value to multiply by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 286 of file vec3.hh.

4.1.4.26 operator/() [2/2]

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

```
            const Vec3< T > & lhs,
            const nT & val )
```

Overloaded divide by value operator.

Template Parameters

nT	type of value to divide by
T	vector template parameter

Parameters

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

Returns

Vec3<T> result vector

Definition at line 303 of file vec3.hh.

4.1.4.27 dot() [2/2]

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

Dot product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T dot production

Definition at line 319 of file vec3.hh.

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

4.1.4.28 cross()

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

Cross product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T cross production

Definition at line 333 of file vec3.hh.

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

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

4.1.4.29 triple()

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

Triple product function.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T triple production

Definition at line 348 of file vec3.hh.

References cross(), and dot().

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

4.1.4.30 operator==() [5/5]

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

Vec3 equality operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

true if vectors are equal

false otherwise

Definition at line 363 of file vec3.hh.

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

4.1.4.31 operator"!=() [2/2]

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

Vec3 inequality operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

true if vectors are not equal

false otherwise

Definition at line 378 of file vec3.hh.

4.1.4.32 operator<<() [6/6]

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

Vec3 print operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

std::ostream& modified stream instance

Definition at line 392 of file vec3.hh.

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

4.1.4.33 operator>>() [2/2]

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

Vec3 scan operator.

Template Parameters

T	vector template parameter

Parameters

in,out	*ist*	input stram
in,out	*vec*	vector to scan

Returns

std::istream& modified stream instance

Definition at line 407 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 16 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 isSameSign (It begin, It end)
- template<Number T>
  bool isSameSign (T num1, T num2)
- template<std::floating_point T>
  bool isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  Trian2< T > getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  bool isCounterClockwise (Trian2< T > &tr)
- template<std::floating_point T>
  Segment2D< T > computeInterval (const Trian2< T > &tr, const Vec2< T > &d)
- template<std::floating_point T>
  Segment3D< T > getSegment (const Triangle< T > &tr)

4.2.1 Typedef Documentation

4.2.1.1 Segment2D

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

Definition at line 15 of file detail.hh.

4.2.1.2 Trian2

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

Definition at line 18 of file detail.hh.

4.2.1.3 Segment3D

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

Definition at line 21 of file detail.hh.

4.2.2 Function Documentation

4.2.2.1 isIntersect2D()

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

Definition at line 77 of file detail.hh.

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

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

4.2.2.2 isIntersectMollerHaines()

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

Definition at line 102 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 116 of file detail.hh.

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

Referenced by isIntersectMollerHaines().

4.2.2.4 isIntersectBothInvalid()

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

Definition at line 160 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 178 of file detail.hh.

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

Referenced by geom::isIntersect().

4.2.2.6 isIntersectPointTriangle()

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

Definition at line 203 of file detail.hh.

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

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 231 of file detail.hh.

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

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 244 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 268 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 274 of file detail.hh.

Referenced by isIntersectMollerHaines(), and isIntersectSegmentSegment().

4.2.2.11 isSameSign() [1/2]

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

Definition at line 280 of file detail.hh.

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

4.2.2.12 isSameSign() [2/2]

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

Definition at line 293 of file detail.hh.

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

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 301 of file detail.hh.

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

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 314 of file detail.hh.

References 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 348 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 368 of file detail.hh.

References geom::dot().

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 384 of file detail.hh.

Referenced by isIntersectBothInvalid(), and isIntersectValidInvalid().

4.3 geom::kdtree Namespace Reference

Classes

- class Container
- class KdTree
- struct Node

Typedefs

- using Index = size_t

4.3.1 Typedef Documentation

4.3.1.1 Index

```
using geom::kdtree::Index = typedef 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)
- const T & min (Axis axis) const
- const T & max (Axis axis) const
- Axis getMaxDim () const

Public Attributes

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

5.1.1 Detailed Description

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

Definition at line 14 of file boundbox.hh.

5.1.2 Member Function Documentation

5.1.2.1 belongsTo()

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

Definition at line 37 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() [1/2]

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

Definition at line 44 of file boundbox.hh.

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

5.1.2.3 max() [1/2]

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

Definition at line 60 of file boundbox.hh.

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

5.1.2.4 min() [2/2]

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

Definition at line 76 of file boundbox.hh.

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

5.1.2.5 max() [2/2]

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

Definition at line 92 of file boundbox.hh.

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

5.1.2.6 getMaxDim()

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

Definition at line 108 of file boundbox.hh.

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

5.1.3 Member Data Documentation

5.1.3.1 minX

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

Definition at line 16 of file boundbox.hh.

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

5.1.3.2 maxX

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

Definition at line 17 of file boundbox.hh.

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

5.1.3.3 minY

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

Definition at line 19 of file boundbox.hh.

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

5.1.3.4 maxY

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

Definition at line 20 of file boundbox.hh.

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

5.1.3.5 minZ

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

Definition at line 22 of file boundbox.hh.

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

5.1.3.6 maxZ

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

Definition at line 23 of file boundbox.hh.

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

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

- include/primitives/boundbox.hh

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

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

Classes

- class ConstIterator

Public Member Functions

- Container (const KdTree< T > *tree, const Node< T > *node)
- Container (const Container &cont)=default
- Container (Container &&cont)=default
- ~Container ()=default
- Container & operator= (const Container &cont)=default
- Container & operator= (Container &&cont)=default
- 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() [1/3]

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

Definition at line 92 of file container.hh.

5.2.2.2 Container() [2/3]

```
template<std::floating_point T>
geom::kdtree::Container< T >::Container (
            const Container< T > & cont )  [default]
```

5.2.2.3 Container() [3/3]

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

5.2.2.4 ~Container()

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

5.2.3 Member Function Documentation

5.2.3.1 operator=() [1/2]

```
template<std::floating_point T>
Container& geom::kdtree::Container< T >::operator= (
            const Container< T > & cont )  [default]
```

5.2.3.2 operator=() [2/2]

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

5.2.3.3 cbegin()

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

Definition at line 96 of file container.hh.

5.2.3.4 cend()

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

Definition at line 102 of file container.hh.

5.2.3.5 begin()

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

Definition at line 108 of file container.hh.

5.2.3.6 end()

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

Definition at line 114 of file container.hh.

5.2.3.7 indexBegin()

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

Definition at line 120 of file container.hh.

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

5.2.3.8 indexEnd()

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

Definition at line 126 of file container.hh.

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

5.2.3.9 separator()

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

Definition at line 132 of file container.hh.

5.2.3.10 sepAxis()

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

Definition at line 138 of file container.hh.

5.2.3.11 boundBox()

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

Definition at line 144 of file container.hh.

5.2.3.12 triangleByIndex()

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

Definition at line 150 of file container.hh.

5.2.3.13 left()

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

Definition at line 156 of file container.hh.

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

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

5.2.3.14 right()

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

Definition at line 162 of file container.hh.

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

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

5.2.3.15 isValid()

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

Definition at line 168 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)
- ConstIterator (const ConstIterator &iter)=default
- ConstIterator (ConstIterator &&iter)=default
- ConstIterator & operator= (const ConstIterator &cont)=default
- ConstIterator & operator= (ConstIterator &&cont)=default
- ~ConstIterator ()=default
- Index getIndex ()
- ConstIterator operator++ ()
- ConstIterator operator++ (int)
- reference operator* () const
- pointer operator-> () const
- bool operator== (const ConstIterator &lhs) const
- bool operator!= (const ConstIterator &lhs) const

5.3.1 Detailed Description

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

Definition at line 51 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 54 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 55 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 56 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 57 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 58 of file container.hh.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 ConstIterator() [1/3]

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

Definition at line 178 of file container.hh.

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

5.3.3.2 ConstIterator() [2/3]

```
template<std::floating_point T>
geom::kdtree::Container< T >::ConstIterator::ConstIterator (
            const ConstIterator & iter )  [default]
```

5.3.3.3 ConstIterator() [3/3]

```
template<std::floating_point T>
geom::kdtree::Container< T >::ConstIterator::ConstIterator (
            ConstIterator && iter )  [default]
```

5.3.3.4 ~ConstIterator()

```
template<std::floating_point T>
geom::kdtree::Container< T >::ConstIterator::~ConstIterator ( )  [default]
```

5.3.4 Member Function Documentation

5.3.4.1 operator=() [1/2]

```
template<std::floating_point T>
ConstIterator& geom::kdtree::Container< T >::ConstIterator::operator= (
            const ConstIterator & cont )  [default]
```

5.3.4.2 operator=() [2/2]

```
template<std::floating_point T>
ConstIterator& geom::kdtree::Container< T >::ConstIterator::operator= (
            ConstIterator && cont )  [default]
```

5.3.4.3 getIndex()

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

Definition at line 190 of file container.hh.

5.3.4.4 operator++() [1/2]

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

Definition at line 196 of file container.hh.

5.3.4.5 operator++() [2/2]

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

Definition at line 203 of file container.hh.

5.3.4.6 operator*()

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

Definition at line 211 of file container.hh.

5.3.4.7 operator->()

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

Definition at line 218 of file container.hh.

5.3.4.8 operator==()

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

Definition at line 225 of file container.hh.

5.3.4.9 operator"!=()

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

Definition at line 231 of file container.hh.

References geom::operator==().

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

- include/kdtree/container.hh

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

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

Classes

- class ConstIterator
- struct ContainerPtr

Public Member Functions

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

Static Public Member Functions

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

5.4.1 Detailed Description

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

Definition at line 13 of file container.hh.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 KdTree() [1/4]

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

Definition at line 126 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 133 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 141 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 147 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 156 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 162 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 168 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 174 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 180 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 188 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 208 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 235 of file kdtree.hh.

5.4.3.11 empty()

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

Definition at line 242 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 248 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 254 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 260 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 266 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 275 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 281 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 287 of file kdtree.hh.

References 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 (const ConstIterator &iter)=default
- ConstIterator (ConstIterator &&iter)=default
- ConstIterator & operator= (const ConstIterator &cont)=default
- ConstIterator & operator= (ConstIterator &&cont)=default
- ~ConstIterator ()=default
- 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() [1/3]

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

Definition at line 424 of file kdtree.hh.

5.5.3.2 ConstIterator() [2/3]

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

5.5.3.3 ConstIterator() [3/3]

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

5.5.3.4 ~ConstIterator()

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

5.5.4 Member Function Documentation

5.5.4.1 operator=() [1/2]

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

5.5.4.2 operator=() [2/2]

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

5.5.4.3 operator++() [1/2]

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

Definition at line 429 of file kdtree.hh.

References geom::NONE.

5.5.4.4 operator++() [2/2]

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

Definition at line 450 of file kdtree.hh.

5.5.4.5 operator*()

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

Definition at line 458 of file kdtree.hh.

5.5.4.6 operator->()

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

Definition at line 464 of file kdtree.hh.

5.5.4.7 operator==()

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

Definition at line 470 of file kdtree.hh.

5.5.4.8 operator"!=()

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

Definition at line 476 of file kdtree.hh.

References geom::operator==().

5.5.4.9 beginFrom()

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

Definition at line 482 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 414 of file kdtree.hh.

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

5.6.3 Member Data Documentation

5.6.3.1 cont

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

Definition at line 80 of file kdtree.hh.

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

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

- include/kdtree/kdtree.hh

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

`#include <node.hh>`

Public Types

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

Public Member Functions

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

Public Attributes

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

5.7.1 Detailed Description

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

Definition at line 16 of file node.hh.

5.7.2 Member Typedef Documentation

5.7.2.1 IndexIterator

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

Definition at line 26 of file node.hh.

5.7.2.2 IndexConstIterator

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

Definition at line 27 of file node.hh.

5.7.3 Member Function Documentation

5.7.3.1 dumpRecursive()

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

Definition at line 33 of file node.hh.

5.7.4 Member Data Documentation

5.7.4.1 separator

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

Definition at line 18 of file node.hh.

5.7.4.2 sepAxis

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

Definition at line 19 of file node.hh.

5.7.4.3 boundBox

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

Definition at line 20 of file node.hh.

5.7.4.4 indicies

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

Definition at line 21 of file node.hh.

5.7.4.5 left

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

Definition at line 23 of file node.hh.

5.7.4.6 right

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

Definition at line 24 of file node.hh.

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

- include/kdtree/node.hh

5.8 geom::Line< T > Class Template Reference

Line class implementation.

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

Public Member Functions

- Line (const Vec3< T > &org, const Vec3< T > &dir)
  Construct a new Line object.
- const Vec3< T > & org () const
  Getter for origin vector.
- const Vec3< T > & dir () const
  Getter for direction vector.
- 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 139 of file line.hh.

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

5.8.3 Member Function Documentation

5.8.3.1 org()

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

Getter for origin vector.

Returns

const Vec3<T>& const reference to origin vector

Definition at line 146 of file line.hh.

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

5.8.3.2 dir()

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

Getter for direction vector.

Returns

const Vec3<T>& const reference to direction vector

Definition at line 152 of file line.hh.

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

5.8.3.3 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 159 of file line.hh.

Referenced by geom::intersect().

5.8.3.4 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 165 of file line.hh.

5.8.3.5 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 171 of file line.hh.

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

5.8.3.6 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 177 of file line.hh.

Referenced by geom::intersect().

5.8.3.7 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 183 of file line.hh.

References geom::Vec3< T >::isNumEq(), and geom::triple().

Referenced by geom::intersect().

5.8.3.8 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 190 of file line.hh.

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

- include/primitives/line.hh

5.9 geom::Plane< T > Class Template Reference

Plane class realization.

```
#include <plane.hh>
```

Public Member Functions

- T dist () const
  Getter for distance.
- const Vec3< T > & norm () const
  Getter for normal vector.
- bool belongs (const Vec3< T > &point) const
  Checks if point belongs to plane.
- bool belongs (const Line< T > &line) const
  Checks if line belongs to plane.
- bool isEqual (const Plane &rhs) const
  Checks is ∗this equals to another plane.
- bool isPar (const Plane &rhs) const
  Checks is ∗this is parallel to another plane.

Static Public Member Functions

- static Plane getBy3Points (const Vec3< T > &pt1, const Vec3< T > &pt2, const Vec3< T > &pt3)
  Get plane by 3 points.
- static Plane getParametric (const Vec3< T > &org, const Vec3< T > &dir1, const Vec3< T > &dir2)
  Get plane from parametric plane equation.
- static Plane getNormalPoint (const Vec3< T > &norm, const Vec3< T > &point)
  Get plane from normal point plane equation.
- static Plane getNormalDist (const Vec3< T > &norm, T constant)
  Get plane form normal const plane equation.

5.9.1 Detailed Description

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

Plane class realization.

Template Parameters

T	- floating point type of coordinates

Definition at line 22 of file plane.hh.

5.9.2 Member Function Documentation

5.9.2.1 dist()

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

Getter for distance.

Returns

T value of distance

Definition at line 171 of file plane.hh.

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

5.9.2.2 norm()

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

Getter for normal vector.

Returns

const Vec3<T>& const reference to normal vector

Definition at line 177 of file plane.hh.

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

5.9.2.3 belongs() [1/2]

```
template<std::floating_point T>
bool geom::Plane< T >::belongs (
            const Vec3< T > & point ) const
```

Checks if point belongs to plane.

Parameters

in	*point*	const referene to point vector

Returns

true if point belongs to plane

false if point doesn't belong to plane

Definition at line 183 of file plane.hh.

5.9.2.4 belongs() [2/2]

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

Checks if line belongs to plane.

Parameters

in	*line*	const referene to line

Returns

true if line belongs to plane

false if line doesn't belong to plane

Definition at line 189 of file plane.hh.

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

5.9.2.5 isEqual()

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

Checks is *this equals to another plane.

Parameters

in	*rhs*	const reference to another plane

Returns

true if planes are equal

false if planes are not equal

Definition at line 195 of file plane.hh.

Referenced by geom::operator==().

5.9.2.6 isPar()

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

Checks is *this is parallel to another plane.

Parameters

in	*rhs*	const reference to another plane

Returns

true if planes are parallel

false if planes are not parallel

Definition at line 201 of file plane.hh.

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

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

5.9.2.7 getBy3Points()

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

Get plane by 3 points.

Parameters

in	*pt1*	1st point
in	*pt2*	2nd point
in	*pt3*	3rd point

Returns

Plane passing through three points

Definition at line 207 of file plane.hh.

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

5.9.2.8 getParametric()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getParametric (
            const Vec3< T > & org,
            const Vec3< T > & dir1,
            const Vec3< T > & dir2 )  [static]
```

Get plane from parametric plane equation.

Parameters

in	*org*	origin vector
in	*dir1*	1st direction vector
in	*dir2*	2nd direction vector

Returns

Plane

Definition at line 213 of file plane.hh.

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

5.9.2.9 getNormalPoint()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalPoint (
            const Vec3< T > & norm,
            const Vec3< T > & point )  [static]
```

Get plane from normal point plane equation.

Parameters

in	*norm*	normal vector
in	*point*	point lying on the plane

Returns

Plane

Definition at line 220 of file plane.hh.

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

5.9.2.10 getNormalDist()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalDist (
            const Vec3< T > & norm,
            T constant )  [static]
```

Get plane form normal const plane equation.

Parameters

in	*norm*	normal vector
in	*constant*	distance

Returns

Plane

Definition at line 227 of file plane.hh.

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

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

- include/primitives/plane.hh

5.10 geom::Triangle< T > Class Template Reference

Triangle class implementation.

`#include <triangle.hh>`

Public Member Functions

- Triangle ()
  Construct a new Triangle object.
- Triangle (const Vec3< T > &p1, const Vec3< T > &p2, const Vec3< T > &p3)
  Construct a new Triangle object from 3 points.
- const Vec3< T > & operator[ ] (std::size_t idx) const
  Overloaded operator[] to get access to vertices.
- Vec3< T > & operator[ ] (std::size_t idx)
  Overloaded operator[] to get access to vertices.
- Plane< T > getPlane () const
  Get triangle's plane.
- bool isValid () const
  Check is triangle valid.
- BoundBox< T > boundBox () const
  Returns triangle's bound box.
- bool belongsTo (const BoundBox< T > &bb) const
  Checks if this Triangle belongs to BoundBox.

5.10.1 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.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Triangle() [1/2]

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

Construct a new Triangle object.

Definition at line 125 of file triangle.hh.

5.10.2.2 Triangle() [2/2]

```
template<std::floating_point T>
geom::Triangle< T >::Triangle (
            const Vec3< T > & p1,
            const Vec3< T > & p2,
            const Vec3< T > & p3 )
```

Construct a new Triangle object from 3 points.

Parameters

in	*p1*	1st point
in	*p2*	2nd point
in	*p3*	3rd point

Definition at line 129 of file triangle.hh.

5.10.3 Member Function Documentation

5.10.3.1 operator[]() [1/2]

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

Overloaded operator[] to get access to vertices.

Parameters

in	*idx*	index of vertex

Returns

const Vec3<T>& const reference to vertex

Definition at line 134 of file triangle.hh.

5.10.3.2 operator[]() [2/2]

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

Overloaded operator[] to get access to vertices.

Parameters

in	*idx*	index of vertex

Returns

Vec3<T>& reference to vertex

Definition at line 140 of file triangle.hh.

5.10.3.3 getPlane()

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

Get triangle's plane.

Returns

Plane<T>

Definition at line 146 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().

5.10.3.4 isValid()

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

Check is triangle valid.

Returns

true if triangle is valid

false if triangle is invalid

Definition at line 152 of file triangle.hh.

References geom::cross().

Referenced by geom::isIntersect().

5.10.3.5 boundBox()

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

Returns triangle's bound box.

Returns

BoundBox<T>

Definition at line 162 of file triangle.hh.

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

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

5.10.3.6 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 174 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

5.11 geom::Vec2< T > Class Template Reference

Vec2 class realization.

`#include <vec2.hh>`

Public Member Functions

- Vec2 (T coordX, T coordY)

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

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

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

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

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

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

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

  Dot product function.
- T length2 () const

  Calculate squared length of a vector function.
- T length () const

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

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

  Get normalized vector function.
- Vec2 & normalize ()

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

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

- T operator[ ] (size_t i) const

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

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

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

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

Static Public Member Functions

- static bool isNumEq (T lhs, T rhs)

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

  Set new threshold value.
- static T getThreshold ()

  Get current threshold value.
- static void setDefThreshold ()

  Set threshold to default value.

Public Attributes

- T x {}

  Vec2 coordinates.
- T y {}

5.11.1 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.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 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 46 of file vec2.hh.

5.11.2.2 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 54 of file vec2.hh.

5.11.3 Member Function Documentation

5.11.3.1 operator+=()

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

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

Parameters

in	*vec*	vector to incremented with

Returns

Vec2& reference to current instance

Definition at line 371 of file vec2.hh.

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

5.11.3.2 operator-=()

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

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

Parameters

in	*vec*	vector to decremented with

Returns

Vec2& reference to current instance

Definition at line 380 of file vec2.hh.

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

5.11.3.3 operator-()

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

Unary - operator.

Returns

Vec2 negated Vec2 instance

Definition at line 389 of file vec2.hh.

5.11.3.4 operator*=() [1/2]

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

Overloaded *= by number operator.

Template Parameters

nType	numeric type of value to multiply by

Parameters

in	*val*	value to multiply by

Returns

Vec2& reference to vector instance

5.11.3.5 operator/=() [1/2]

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

Overloaded /= by number operator.

Template Parameters

nType	numeric type of value to divide by

Parameters

in	*val*	value to divide by

Returns

Vec2& reference to vector instance

Warning

Does not check if val equals 0

5.11.3.6 dot()

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

Dot product function.

Parameters

rhs	vector to dot product with

Returns

T dot product of two vectors

Definition at line 415 of file vec2.hh.

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

Referenced by geom::dot().

5.11.3.7 length2()

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

Calculate squared length of a vector function.

Returns

T length^2

Definition at line 421 of file vec2.hh.

References geom::dot().

5.11.3.8 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 427 of file vec2.hh.

5.11.3.9 getPerp()

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

Get the perpendicular to this vector.

Returns

Vec2 perpendicular vector

Definition at line 433 of file vec2.hh.

5.11.3.10 normalized()

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

Get normalized vector function.

Returns

Vec2 normalized vector

Definition at line 439 of file vec2.hh.

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

5.11.3.11 normalize()

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

Normalize vector function.

Returns

Vec2& reference to instance

Definition at line 447 of file vec2.hh.

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

5.11.3.12 operator[]() [1/2]

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

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

Parameters

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

Returns

T& reference to coordinate value

Note

Coordinates calculated by mod 2

Definition at line 456 of file vec2.hh.

5.11.3.13 operator[]() [2/2]

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

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

Parameters

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

Returns

T coordinate value

Note

Coordinates calculated by mod 2

Definition at line 470 of file vec2.hh.

5.11.3.14 isPar()

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

Check if vector is parallel to another.

Parameters

in	*rhs*	vector to check parallelism with

Returns

true if vector is parallel

false otherwise

Definition at line 484 of file vec2.hh.

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

5.11.3.15 isPerp()

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

Check if vector is perpendicular to another.

Parameters

in	*rhs*	vector to check perpendicularity with

Returns

true if vector is perpendicular

false otherwise

Definition at line 491 of file vec2.hh.

References geom::dot().

5.11.3.16 isEqual()

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

Check if vector is equal to another.

Parameters

in	*rhs*	vector to check equality with

Returns

true if vector is equal

false otherwise

Note

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

Definition at line 497 of file vec2.hh.

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

Referenced by geom::operator==().

5.11.3.17 isNumEq()

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

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

Parameters

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

Returns

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

false otherwise

Note

Threshold defined by threshold_ static member

Definition at line 503 of file vec2.hh.

5.11.3.18 setThreshold()

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

Set new threshold value.

Parameters

in	*thres*	value to set

Definition at line 509 of file vec2.hh.

5.11.3.19 getThreshold()

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

Get current threshold value.

Definition at line 515 of file vec2.hh.

5.11.3.20 setDefThreshold()

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

Set threshold to default value.

Note

default value equals float point epsilon

Definition at line 521 of file vec2.hh.

5.11.3.21 operator*=() [2/2]

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

Definition at line 396 of file vec2.hh.

5.11.3.22 operator/=() [2/2]

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

Definition at line 406 of file vec2.hh.

5.11.4 Member Data Documentation

5.11.4.1 x

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

Vec2 coordinates.

Definition at line 38 of file vec2.hh.

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

5.11.4.2 y

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

Definition at line 38 of file vec2.hh.

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

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

- include/primitives/vec2.hh

5.12 geom::Vec3< T > Class Template Reference

Vec3 class realization.

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

Public Member Functions

- Vec3 (T coordX, T coordY, T coordZ)

  Construct a new Vec3 object from 3 coordinates.
- Vec3 (T coordX={})

  Construct a new Vec3 object with equals coordinates.
- Vec3 & operator+= (const Vec3 &vec)

  Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.
- Vec3 & operator-= (const Vec3 &vec)

  Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.
- Vec3 operator- () const

  Unary - operator.
- template<Number nType>
  Vec3 & operator*= (nType val)

  *Overloaded *= by number operator.*
- template<Number nType>
  Vec3 & operator/= (nType val)

  Overloaded /= by number operator.
- T dot (const Vec3 &rhs) const

  Dot product function.
- Vec3 cross (const Vec3 &rhs) const

  Cross product function.
- T length2 () const

  Calculate squared length of a vector function.
- T length () const

  Calculate length of a vector function.
- Vec3 normalized () const

  Get normalized vector function.
- Vec3 & normalize ()

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

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

  Overloaded operator [] (const version) To get access to coordinates.
- bool isPar (const Vec3 &rhs) const

  Check if vector is parallel to another.
- bool isPerp (const Vec3 &rhs) const

  Check if vector is perpendicular to another.
- bool isEqual (const Vec3 &rhs) const

  Check if vector is equal to another.
- template<Number nType>
  Vec3< T > & operator*= (nType val)
- template<Number nType>
  Vec3< T > & operator/= (nType val)

Static Public Member Functions

- static bool isNumEq (T lhs, T rhs)

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

  Set new threshold value.
- static T getThreshold ()

  Get current threshold value.
- static void setDefThreshold ()

  Set threshold to default value.

Public Attributes

- T x {}

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

5.12.1 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.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Vec3() [1/2]

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

Construct a new Vec3 object from 3 coordinates.

Parameters

in	*coordX*	x coordinate
in	*coordY*	y coordinate
in	*coordZ*	z coordinate

Definition at line 47 of file vec3.hh.

5.12.2.2 Vec3() [2/2]

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

Construct a new Vec3 object with equals coordinates.

Parameters

in	*coordX*	coordinate (default to {})

Definition at line 55 of file vec3.hh.

5.12.3 Member Function Documentation

5.12.3.1 operator+=()

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

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

Parameters

in	*vec*	vector to incremented with

Returns

Vec3& reference to current instance

Definition at line 417 of file vec3.hh.

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

5.12.3.2 operator-=()

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

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

Parameters

in	*vec*	vector to decremented with

Returns

Vec3& reference to current instance

Definition at line 427 of file vec3.hh.

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

5.12.3.3 operator-()

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

Unary - operator.

Returns

Vec3 negated Vec3 instance

Definition at line 437 of file vec3.hh.

5.12.3.4 operator*=() [1/2]

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

Overloaded *= by number operator.

Template Parameters

nType	numeric type of value to multiply by

Parameters

in	*val*	value to multiply by

Returns

Vec3& reference to vector instance

5.12.3.5 operator/=() [1/2]

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

Overloaded /= by number operator.

Template Parameters

nType	numeric type of value to divide by

Parameters

in	*val*	value to divide by

Returns

Vec3& reference to vector instance

Warning

Does not check if val equals 0

5.12.3.6 dot()

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

Dot product function.

Parameters

rhs	vector to dot product with

Returns

T dot product of two vectors

Definition at line 465 of file vec3.hh.

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

Referenced by geom::dot().

5.12.3.7 cross()

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

Cross product function.

Parameters

rhs	vector to cross product with

Returns

Vec3 cross product of two vectors

Definition at line 471 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().

5.12.3.8 length2()

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

Calculate squared length of a vector function.

Returns

T length^2

Definition at line 477 of file vec3.hh.

References geom::dot().

5.12.3.9 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 483 of file vec3.hh.

5.12.3.10 normalized()

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

Get normalized vector function.

Returns

Vec3 normalized vector

Definition at line 489 of file vec3.hh.

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

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

5.12.3.11 normalize()

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

Normalize vector function.

Returns

Vec3& reference to instance

Definition at line 497 of file vec3.hh.

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

5.12.3.12 operator[]() [1/2]

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

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

Parameters

i	index of coordinate (0 - x, 1 - y, 2 - z)

Returns

T& reference to coordinate value

Note

Coordinates calculated by mod 3

Definition at line 506 of file vec3.hh.

5.12.3.13 operator[]() [2/2]

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

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

Parameters

i	index of coordinate (0 - x, 1 - y, 2 - z)

Returns

T coordinate value

Note

Coordinates calculated by mod 3

Definition at line 522 of file vec3.hh.

5.12.3.14 isPar()

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

Check if vector is parallel to another.

Parameters

in	*rhs*	vector to check parallelism with

Returns

true if vector is parallel

false otherwise

Definition at line 538 of file vec3.hh.

References geom::cross().

5.12.3.15 isPerp()

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

Check if vector is perpendicular to another.

Parameters

in	*rhs*	vector to check perpendicularity with

Returns

true if vector is perpendicular

false otherwise

Definition at line 544 of file vec3.hh.

References geom::dot().

5.12.3.16 isEqual()

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

Check if vector is equal to another.

Parameters

in	*rhs*	vector to check equality with

Returns

true if vector is equal

false otherwise

Note

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

Definition at line 550 of file vec3.hh.

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

Referenced by geom::operator==().

5.12.3.17 isNumEq()

```
template<std::floating_point T>
bool geom::Vec3< T >::isNumEq (
            T lhs,
            T rhs )  [static]
```

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

Parameters

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

Returns

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

false otherwise

Note

Threshold defined by threshold_ static member

Definition at line 556 of file vec3.hh.

Referenced by geom::detail::isSameSign(), geom::Line< T >::isSkew(), and geom::operator==().

5.12.3.18 setThreshold()

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

Set new threshold value.

Parameters

in	*thres*	value to set

Definition at line 562 of file vec3.hh.

5.12.3.19 getThreshold()

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

Get current threshold value.

Definition at line 568 of file vec3.hh.

Referenced by geom::Triangle< T >::boundBox(), and geom::detail::isIntersectPointTriangle().

5.12.3.20 setDefThreshold()

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

Set threshold to default value.

Note

default value equals float point epsilon

Definition at line 574 of file vec3.hh.

5.12.3.21 operator*=() [2/2]

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

Definition at line 444 of file vec3.hh.

5.12.3.22 operator/=() [2/2]

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

Definition at line 455 of file vec3.hh.

5.12.4 Member Data Documentation

5.12.4.1 x

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

Vec3 coordinates.

Definition at line 38 of file vec3.hh.

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

5.12.4.2 y

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

Definition at line 38 of file vec3.hh.

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

5.12.4.3 z

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

Definition at line 38 of file vec3.hh.

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

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

- include/primitives/vec3.hh

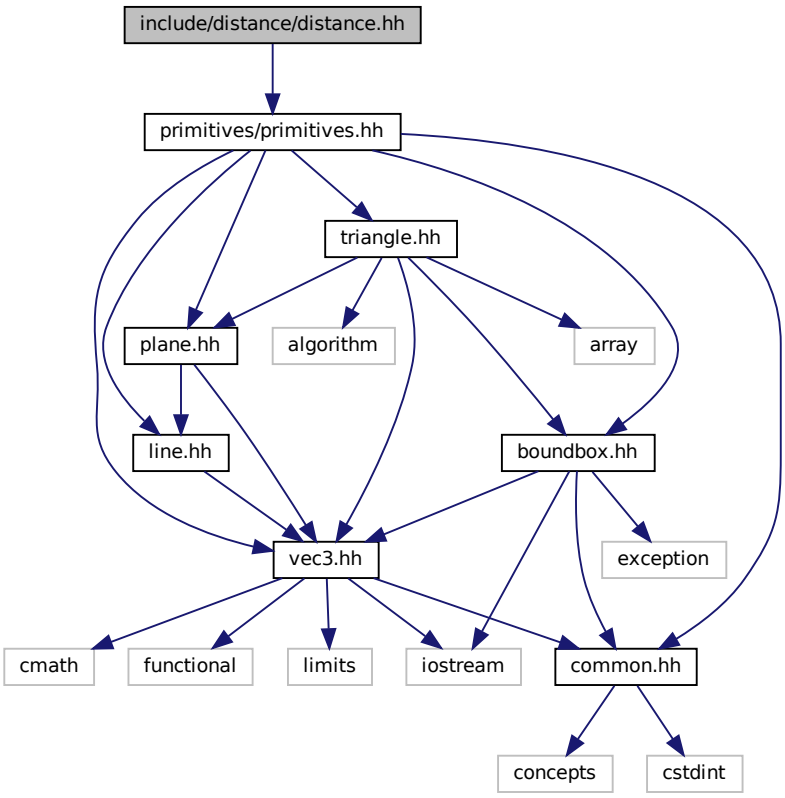
Chapter 6

File Documentation

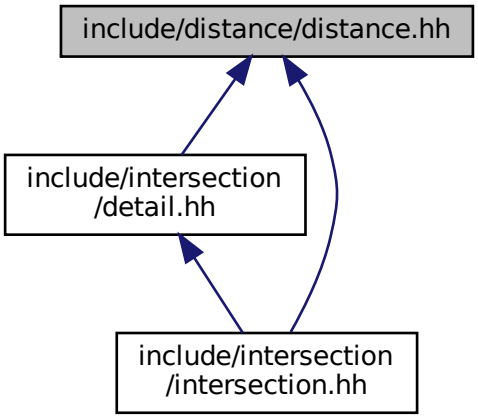
6.1 include/distance/distance.hh File Reference

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

Include dependency graph for distance.hh:



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



Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Calculates signed distance between point and plane.

6.2 distance.hh

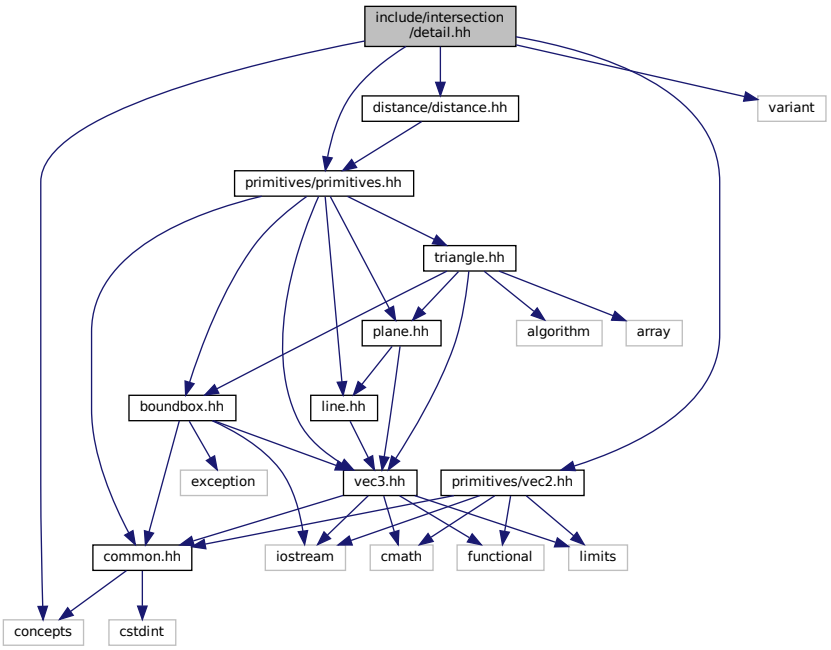
```
00001 #ifndef __INCLUDE_DISTANCE_DISTANCE_HH__
00002 #define __INCLUDE_DISTANCE_DISTANCE_HH__
00003
00004 #include "primitives/primitives.hh"
00005
00006 namespace geom
00007 {
00008
00009 /**
00010  * @brief Calculates signed distance between point and plane
00011  *
00012  * @tparam T - floating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

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

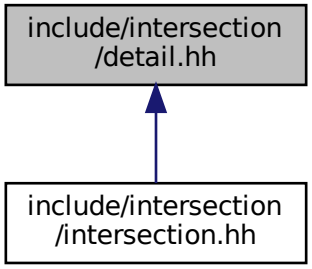
6.3 include/intersection/detail.hh File Reference

```
#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::isSameSign (It begin, It end)
- template<Number T>
  bool geom::detail::isSameSign (T num1, T num2)
- template<std::floating_point T>
  bool geom::detail::isOnOneSide (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  Trian2< T > geom::detail::getTrian2 (const Plane< T > &pl, const Triangle< T > &tr)
- template<std::floating_point T>
  bool geom::detail::isCounterClockwise (Trian2< T > &tr)
- template<std::floating_point T>
  Segment2D< T > geom::detail::computeInterval (const Trian2< T > &tr, const Vec2< T > &d)
- template<std::floating_point T>
  Segment3D< T > geom::detail::getSegment (const Triangle< T > &tr)

6.4 detail.hh

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

```
00054 bool isSameSign(It begin, It end);
00055
00056 template <Number T>
00057 bool isSameSign(T num1, T num2);
00058
00059 template <std::floating_point T>
00060 bool isOnOneSide(const Plane<T> &pl, const Triangle<T> &tr);
00061
00062 template <std::floating_point T>
00063 Trian2<T> getTrian2(const Plane<T> &pl, const Triangle<T> &tr);
00064
00065 template <std::floating_point T>
00066 bool isCounterClockwise(Trian2<T> &tr);
00067
00068 template <std::floating_point T>
00069 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d);
00070
00071 template <std::floating_point T>
00072 Segment3D<T> getSegment(const Triangle<T> &tr);
00073
00074 //=================================================================
00075
00076 template <std::floating_point T>
00077 bool isIntersect2D(const Triangle<T> &tr1, const Triangle<T> &tr2)
00078 {
00079   auto pl = tr1.getPlane();
00080
00081   auto trian1 = getTrian2(pl, tr1);
00082   auto trian2 = getTrian2(pl, tr2);
00083
00084   for (auto trian : {trian1, trian2})
00085   {
00086     for (size_t i0 = 0, i1 = 2; i0 < 3; i1 = i0, ++i0)
00087     {
00088       auto d = (trian[i0] - trian[i1]).getPerp();
00089
00090       auto s1 = computeInterval(trian1, d);
00091       auto s2 = computeInterval(trian2, d);
00092
00093       if (s2.second < s1.first || s1.second < s2.first)
00094         return false;
00095     }
00096   }
00097
00098   return true;
00099 }
00100
00101 template <std::floating_point T>
00102 bool isIntersectMollerHaines(const Triangle<T> &tr1, const Triangle<T> &tr2)
00103 {
00104   auto pl1 = tr1.getPlane();
00105   auto pl2 = tr2.getPlane();
00106
00107   auto l = std::get<Line<T>>(intersect(pl1, pl2));
00108
00109   auto params1 = helperMollerHaines(tr1, pl2, l);
00110   auto params2 = helperMollerHaines(tr2, pl1, l);
00111
00112   return isOverlap(params1, params2);
00113 }
00114
00115 template <std::floating_point T>
00116 Segment2D<T> helperMollerHaines(const Triangle<T> &tr, const Plane<T> &pl, const Line<T> &l)
00117 {
00118   /* Project the triangle vertices onto line */
00119   std::array<T, 3> vert{};
00120   for (size_t i = 0; i < 3; ++i)
00121     vert[i] = dot(l.dir(), tr[i] - l.org());
00122
00123   std::array<T, 3> sdist{};
00124   for (size_t i = 0; i < 3; ++i)
00125     sdist[i] = distance(pl, tr[i]);
00126
00127   std::array<bool, 3> isOneSide{};
00128   for (size_t i = 0; i < 3; ++i)
00129     isOneSide[i] = isSameSign(sdist[i], sdist[(i + 1) % 3]);
00130
00131   /* Looking for vertex which is alone on it's side */
00132   size_t rogue = 0;
00133   if (std::all_of(isOneSide.begin(), isOneSide.end(), [](const auto &elem) { return !elem; }))
00134   {
00135     for (size_t i = 0; i < 3; ++i)
00136       if (!Vec3<T>::isNumEq(0, sdist[i]))
00137         rogue = i;
00138   }
00139   else
00140   {
```

```
00141     for (size_t i = 0; i < 3; ++i)
00142       if (isOneSide[i])
00143         rogue = (i + 2) % 3;
00144   }
00145
00146   std::vector<T> segm{};
00147   std::array<size_t, 2> arr{(rogue + 1) % 3, (rogue + 2) % 3};
00148
00149   for (size_t i : arr)
00150     segm.push_back(vert[i] + (vert[rogue] - vert[i]) * sdist[i] / (sdist[i] - sdist[rogue]));
00151
00152   /* Sort segment's ends */
00153   if (segm[0] > segm[1])
00154     std::swap(segm[0], segm[1]);
00155
00156   return {segm[0], segm[1]};
00157 }
00158
00159 template <std::floating_point T>
00160 bool isIntersectBothInvalid(const Triangle<T> &tr1, const Triangle<T> &tr2)
00161 {
00162   auto isPoint1 = isPoint(tr1);
00163   auto isPoint2 = isPoint(tr2);
00164
00165   if (isPoint1 && isPoint2)
00166     return tr1[0] == tr2[0];
00167
00168   if (isPoint1)
00169     return isIntersectPointSegment(tr1[0], getSegment(tr2));
00170
00171   if (isPoint2)
00172     return isIntersectPointSegment(tr2[0], getSegment(tr1));
00173
00174   return isIntersectSegmentSegment(getSegment(tr1), getSegment(tr2));
00175 }
00176
00177 template <std::floating_point T>
00178 bool isIntersectValidInvalid(const Triangle<T> &valid, const Triangle<T> &invalid)
00179 {
00180   if (isPoint(invalid))
00181     return isIntersectPointTriangle(invalid[0], valid);
00182
00183   auto segm = getSegment(invalid);
00184   auto pl = valid.getPlane();
00185
00186   auto dst1 = distance(pl, segm.first);
00187   auto dst2 = distance(pl, segm.second);
00188
00189   if (dst1 * dst2 > 0)
00190     return false;
00191
00192   if (Vec3<T>::isNumEq(dst1, 0) && Vec3<T>::isNumEq(dst2, 0))
00193     return isIntersect2D(valid, invalid);
00194
00195   dst1 = std::abs(dst1);
00196   dst2 = std::abs(dst2);
00197
00198   auto pt = segm.first + (segm.second - segm.first) * dst1 / (dst1 + dst2);
00199   return isIntersectPointTriangle(pt, valid);
00200 }
00201
00202 template <std::floating_point T>
00203 bool isIntersectPointTriangle(const Vec3<T> &pt, const Triangle<T> &tr)
00204 {
00205   if (!tr.getPlane().belongs(pt))
00206     return false;
00207
00208   /* TODO: comment better */
00209   /* pt = point + u * edge1 + v * edge2 */
00210   auto point = pt - tr[0];
00211   auto edge1 = tr[1] - tr[0];
00212   auto edge2 = tr[2] - tr[0];
00213
00214   auto dotE1E1 = dot(edge1, edge1);
00215   auto dotE1E2 = dot(edge1, edge2);
00216   auto dotE1PT = dot(edge1, point);
00217
00218   auto dotE2E2 = dot(edge2, edge2);
00219   auto dotE2PT = dot(edge2, point);
00220
00221   auto denom = dotE1E1 * dotE2E2 - dotE1E2 * dotE1E2;
00222   auto u = (dotE2E2 * dotE1PT - dotE1E2 * dotE2PT) / denom;
00223   auto v = (dotE1E1 * dotE2PT - dotE1E2 * dotE1PT) / denom;
00224
00225   /* Point belongs to triangle if: (u >= 0) && (v >= 0) && (u + v <= 1) */
00226   auto eps = Vec3<T>::getThreshold();
00227   return (u > -eps) && (v > -eps) && (u + v < 1 + eps);
```

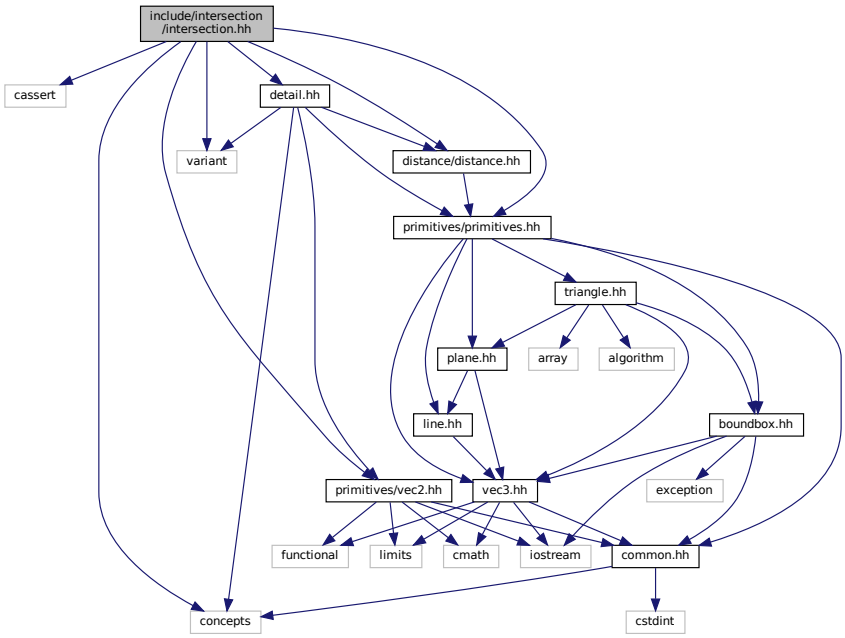
```
00228 }
00229
00230 template <std::floating_point T>
00231 bool isIntersectPointSegment(const Vec3<T> &pt, const Segment3D<T> &segm)
00232 {
00233   Line<T> l{segm.first, segm.second - segm.first};
00234   if (!l.belongs(pt))
00235     return false;
00236
00237   auto beg = dot(l.dir(), segm.first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

```
00315 {
00316   auto norm = pl.norm();
00317
00318   const Vec3<T> x{1, 0, 0};
00319   const Vec3<T> y{0, 1, 0};
00320   const Vec3<T> z{0, 0, 1};
00321
00322   std::array<Vec3<T>, 3> xyz{x, y, z};
00323   std::array<T, 3> xyzDot;
00324
00325   std::transform(xyz.begin(), xyz.end(), xyzDot.begin(),
00326                  [&norm](const auto &axis) { return std::abs(dot(axis, norm)); });
00327
00328   auto maxIt = std::max_element(xyzDot.begin(), xyzDot.end());
00329   auto maxIdx = static_cast<size_t>(maxIt - xyzDot.begin());
00330
00331   Trian2<T> res;
00332   for (size_t i = 0; i < 3; ++i)
00333     for (size_t j = 0, k = 0; j < 2; ++j, ++k)
00334     {
00335       if (k == maxIdx)
00336         ++k;
00337
00338       res[i][j] = tr[i][k];
00339     }
00340
00341   if (!isCounterClockwise(res))
00342     std::swap(res[0], res[1]);
00343
00344   return res;
00345 }
00346
00347 template <std::floating_point T>
00348 bool isCounterClockwise(Trian2<T> &tr)
00349 {
00350   /**
00351    * The triangle is counterclockwise ordered if \delta > 0
00352    * and clockwise ordered if \delta < 0.
00353    *
00354    *               +  1  1  1 +
00355    * \delta = det | x0 x1 x2 | = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0)
00356    *               + y0 y1 y2 +                      + (x0 * y1 - x1 * y0)
00357    *
00358    */
00359
00360   auto x0 = tr[0][0], x1 = tr[1][0], x2 = tr[2][0];
00361   auto y0 = tr[0][1], y1 = tr[1][1], y2 = tr[2][1];
00362
00363   auto delta = (x1 * y2 - x2 * y1) - (x0 * y2 - x2 * y0) + (x0 * y1 - x1 * y0);
00364   return (delta > 0);
00365 }
00366
00367 template <std::floating_point T>
00368 Segment2D<T> computeInterval(const Trian2<T> &tr, const Vec2<T> &d)
00369 {
00370   auto init = dot(d, tr[0]);
00371   auto min = init;
00372   auto max = init;
00373
00374   for (size_t i = 1; i < 3; ++i)
00375     if (auto val = dot(d, tr[i]); val < min)
00376       min = val;
00377     else if (val > max)
00378       max = val;
00379
00380   return {min, max};
00381 }
00382
00383 template <std::floating_point T>
00384 Segment3D<T> getSegment(const Triangle<T> &tr)
00385 {
00386   std::array<T, 3> lenArr{};
00387   for (size_t i = 0; i < 3; ++i)
00388     lenArr[i] = (tr[i] - tr[i + 1]).length2();
00389
00390   auto maxIt = std::max_element(lenArr.begin(), lenArr.end());
00391   auto maxIdx = static_cast<size_t>(maxIt - lenArr.begin());
00392
00393   return {tr[maxIdx], tr[maxIdx + 1]};
00394 }
00395
00396 } // namespace geom::detail
00397
00398 #endif // __INCLUDE_INTERSECTION_DETAIL_HH__
```

6.5 include/intersection/intersection.hh File Reference

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

Include dependency graph for intersection.hh:



Namespaces

- geom

  line.hh Line class implementation

Functions

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

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

  Intersect 2 planes and return result of intersection.
- template<std::floating_point T>
  std::variant< std::monostate, Vec3< T >, Line< T > > geom::intersect (const Line< T > &l1, const Line< T > &l2)

  Intersect 2 lines and return result of intersection.

6.6 intersection.hh

```
00001 #ifndef __INCLUDE_INTERSECTION_INTERSECTION_HH__
00002 #define __INCLUDE_INTERSECTION_INTERSECTION_HH__
00003
00004 #include <cassert>
00005 #include <concepts>
00006 #include <variant>
00007
00008 #include "distance/distance.hh"
00009 #include "primitives/primitives.hh"
00010 #include "primitives/vec2.hh"
00011
00012 #include "detail.hh"
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @brief Checks intersection of 2 triangles
00019  *
00020  * @tparam T - floating point type of coordinates
00021  * @param tr1 first triangle
00022  * @param tr2 second triangle
00023  * @return true if triangles are intersect
00024  * @return false if triangles are not intersect
00025  */
00026 template <std::floating_point T>
00027 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2);
00028
00029 /**
00030  * @brief Intersect 2 planes and return result of intersection
00031  * @details
00032  * Common intersection case (parallel planes case is trivial):
00033  *
00034  * Let \f$ \overrightarrow{P} \f$ - point in space
00035  *
00036  * \f$ pl_1 \f$ equation: \f$ \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1 \f$
00037  *
00038  * \f$ pl_2 \f$ equation: \f$ \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2 \f$
00039  *
00040  * Intersection line direction: \f$ \overrightarrow{dir} = \overrightarrow{n}_1 \times
00041  * \overrightarrow{n}_2 \f$
00042  *
00043  * Let origin of intersection line be a linear combination of \f$ \overrightarrow{n}_1 \f$
00044  * and \f$ \overrightarrow{n}_2 \f$: \f[ \overrightarrow{P} = a \cdot \overrightarrow{n}_1
00045  * + b \cdot \overrightarrow{n}_2 \f]
00046  *
00047  * \f$ \overrightarrow{P} \f$ must satisfy both \f$ pl_1 \f$ and \f$ pl_1 \f$ equations:
00048  * \f[
00049  * \overrightarrow{n}_1 \cdot \overrightarrow{P} = d_1
00050  * \Leftrightarrow
00051  * \overrightarrow{n}_1
00052  * \cdot
00053  * \left(
00054  *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00055  *  \right)
00056  *  = d_1
00057  * \Leftrightarrow
00058  * a + b \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2  = d_1
00059  * \f]
00060  * \f[
00061  * \overrightarrow{n}_2 \cdot \overrightarrow{P} = d_2
00062  * \Leftrightarrow
00063  * \overrightarrow{n}_2
00064  * \cdot
00065  * \left(
00066  *  a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2
00067  *  \right) = d_2
00068  * \Leftrightarrow
00069  * a \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 + b = d_2
00070  * \f]
00071  *
00072  * Let's find \f$a\f$ and \f$b\f$:
00073  * \f[
00074  * a = \frac{
00075  *  d_2 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_1
00076  * }{
00077  *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
00078  * }
00079  * \f]
00080  * \f[
00081  * b = \frac{
00082  * d_1 \cdot \overrightarrow{n}_1 \cdot \overrightarrow{n}_2 - d_2
00083  * }{
00084  *  \left( \overrightarrow{n}_1 \cdot \overrightarrow{n}_2\right)^2 - 1
00085  * }
```

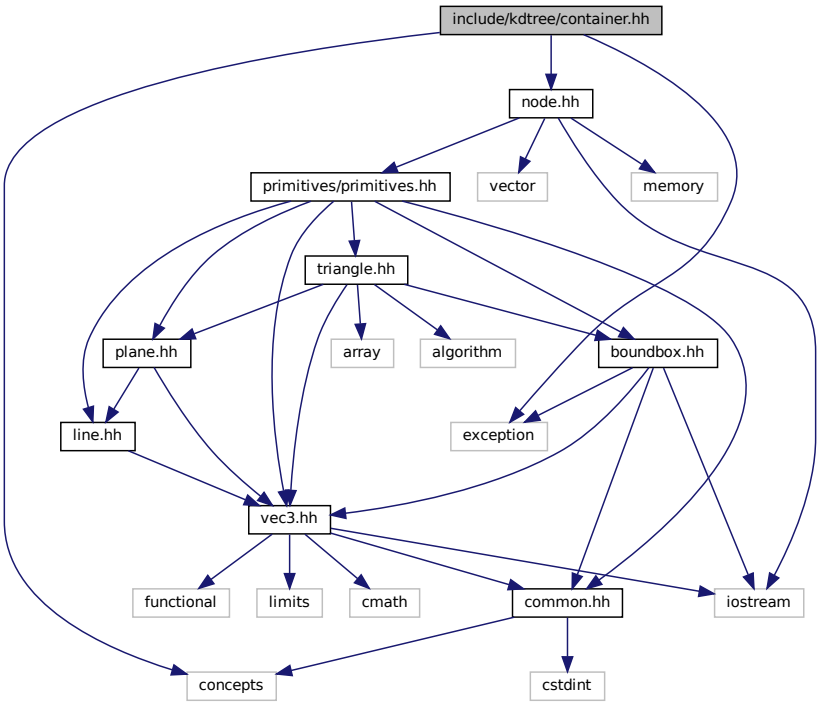
```
00086  * \f]
00087  *
00088  * Intersection line equation:
00089  * \f[
00090  * \overrightarrow{r}(t) = \overrightarrow{P} + t \cdot \overrightarrow{n}_1 \times
00091  * \overrightarrow{n}_2 = (a \cdot \overrightarrow{n}_1 + b \cdot \overrightarrow{n}_2) +
00092  * t \cdot \overrightarrow{n}_1 \times \overrightarrow{n}_2 \f]
00093  *
00094  * @tparam T - floating point type of coordinates
00095  * @param[in] pl1 first plane
00096  * @param[in] pl2 second plane
00097  * @return std::variant<std::monostate, Line<T>, Plane<T>»
00098  */
00099 template <std::floating_point T>
00100 std::variant<std::monostate, Line<T>, Plane<T>» intersect(const Plane<T> &pl1, const Plane<T> &pl2);
00101
00102 /**
00103  * @brief Intersect 2 lines and return result of intersection
00104  * @details
00105  * Common intersection case (parallel & skew lines cases are trivial):
00106  * Let \f$ \overrightarrow{P} \f$ - point in space, intersection point of two lines.
00107  *
00108  * \f$ l_1 \f$ equation: \f$ \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
00109  * \overrightarrow{P} \f$
00110  *
00111  * \f$ l_2 \f$ equation: \f$ \overrightarrow{org}_2 + \overrightarrow{dir}_2
00112  * \cdot t_2 = \overrightarrow{P} \f$
00113  *
00114  * Let's equate left sides:
00115  * \f[
00116  * \overrightarrow{org}_1 + \overrightarrow{dir}_1 \cdot t_1 =
00117  * \overrightarrow{org}_2 + \overrightarrow{dir}_2 \cdot t_2
00118  * \f]
00119  * Cross multiply both sides from right by \f$ \overrightarrow{dir}_2 \f$:
00120  * \f[
00121  * t_1 \cdot \left(\overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right) =
00122  * \left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times \overrightarrow{dir}_2
00123  * \f]
00124  * Dot multiply both sides by \f$ \frac{\overrightarrow{dir}_1 \times \overrightarrow{dir}_2}{\left|
00125  * \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2} \f$:
00126  *
00127  * \f[
00128  * t_1 = \frac{
00129  *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times
00130  * \overrightarrow{dir}_2\right) \cdot \left( \overrightarrow{dir}_1 \times \overrightarrow{dir}_2
00131  * \right)
00132  * }{
00133  * \left| \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2
00134  * }
00135  * \f]
00136  *
00137  * Thus we get intersection point parameter \f$ t_1 \f$ on \f$ l_1 \f$, let's substitute it to \f$
00138  * l_1 \f$ equation: \f[ \overrightarrow{P} = \overrightarrow{org}_1 + \frac{
00139  *  \left(\left(\overrightarrow{org}_2 - \overrightarrow{org}_1 \right) \times
00140  * \overrightarrow{dir}_2\right) \cdot \left( \overrightarrow{dir}_1 \times \overrightarrow{dir}_2
00141  * \right)
00142  * }{
00143  * \left| \overrightarrow{dir}_1 \times \overrightarrow{dir}_2 \right|^2
00144  * } \cdot \overrightarrow{dir}_1
00145  * \f]
00146  *
00147  * @tparam T - floating point type of coordinates
00148  * @param[in] l1 first line
00149  * @param[in] l2 second line
00150  * @return std::variant<std::monostate, Vec3<T>, Line<T>»
00151  */
00152 template <std::floating_point T>
00153 std::variant<std::monostate, Vec3<T>, Line<T>» intersect(const Line<T> &l1, const Line<T> &l2);
00154
00155 template <std::floating_point T>
00156 bool isIntersect(const Triangle<T> &tr1, const Triangle<T> &tr2)
00157 {
00158   auto isInv1 = !tr1.isValid();
00159   auto isInv2 = !tr2.isValid();
00160
00161   if (isInv1 && isInv2)
00162     return detail::isIntersectBothInvalid(tr1, tr2);
00163
00164   if (isInv1)
00165     return detail::isIntersectValidInvalid(tr2, tr1);
00166
00167   if (isInv2)
00168     return detail::isIntersectValidInvalid(tr1, tr2);
00169
00170   auto pl1 = tr1.getPlane();
00171   if (detail::isOnOneSide(pl1, tr2))
00172     return false;
```

```
00173
00174   auto pl2 = tr2.getPlane();
00175   if (pl1 == pl2)
00176     return detail::isIntersect2D(tr1, tr2);
00177
00178   if (pl1.isPar(pl2))
00179     return false;
00180
00181   if (detail::isOnOneSide(pl2, tr1))
00182     return false;
00183
00184   return detail::isIntersectMollerHaines(tr1, tr2);
00185 }
00186
00187 template <std::floating_point T>
00188 std::variant<std::monostate, Line<T>, Plane<T>> intersect(const Plane<T> &pl1, const Plane<T> &pl2)
00189 {
00190   const auto &n1 = pl1.norm();
00191   const auto &n2 = pl2.norm();
00192
00193   auto dir = cross(n1, n2);
00194
00195   /* if planes are parallel */
00196   if (Vec3<T>{0} == dir)
00197   {
00198     if (pl1 == pl2)
00199       return pl1;
00200
00201     return std::monostate{};
00202   }
00203
00204   auto n1n2 = dot(n1, n2);
00205   auto d1 = pl1.dist();
00206   auto d2 = pl2.dist();
00207
00208   auto a = (d2 * n1n2 - d1) / (n1n2 * n1n2 - 1);
00209   auto b = (d1 * n1n2 - d2) / (n1n2 * n1n2 - 1);
00210
00211   return Line<T>{(a * n1) + (b * n2), dir};
00212 }
00213
00214 template <std::floating_point T>
00215 std::variant<std::monostate, Vec3<T>, Line<T>> intersect(const Line<T> &l1, const Line<T> &l2)
00216 {
00217   if (l1.isPar(l2))
00218   {
00219     if (l1.isEqual(l2))
00220       return l1;
00221
00222     return std::monostate{};
00223   }
00224
00225   if (l1.isSkew(l2))
00226     return std::monostate{};
00227
00228   auto dir1xdir2 = cross(l1.dir(), l2.dir());
00229   auto org21xdir2 = cross(l2.org() - l1.org(), l2.dir());
00230
00231   auto t1_intersect = dot(org21xdir2, dir1xdir2) / dir1xdir2.length2();
00232
00233   auto point = l1.getPoint(t1_intersect);
00234
00235   return point;
00236 }
00237
00238 } // namespace geom
00239
00240 #endif // __INCLUDE_INTERSECTION_INTERSECTION_HH__
```

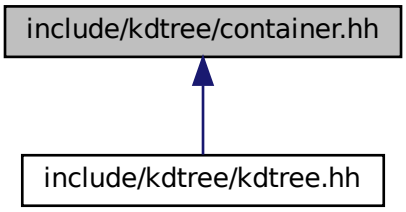
6.7 include/kdtree/container.hh File Reference

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

Include dependency graph for container.hh:



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



Classes

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

Namespaces

- geom

  line.hh Line class implementation
- geom::kdtree

6.8 container.hh

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

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

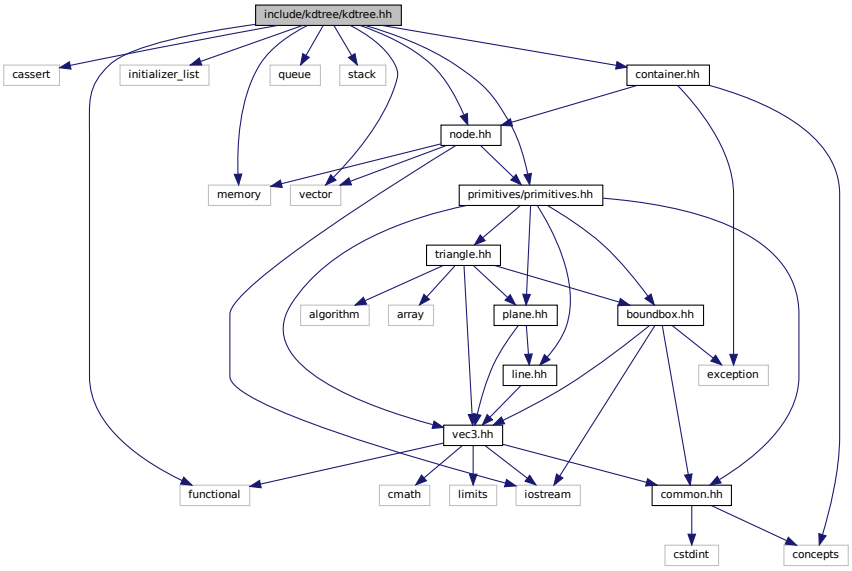
```
00173 //=================================================================================================
00174 //                                Container::ConstIterator definitions
00175 //=================================================================================================
00176
00177 template <std::floating_point T>
00178 Container<T>::ConstIterator::ConstIterator(const Container<T> *cont, bool isEnd) : cont_(cont)
00179 {
00180   if (nullptr == cont_)
00181     throw std::invalid_argument("Tried to create iterator with invalid Container pointer");
00182
00183   if (isEnd)
00184     curIdxIt_ = cont_->indexEnd();
00185   else
00186     curIdxIt_ = cont_->indexBegin();
00187 }
00188
00189 template <std::floating_point T>
00190 Index Container<T>::ConstIterator::getIndex()
00191 {
00192   return *curIdxIt_;
00193 }
00194
00195 template <std::floating_point T>
00196 typename Container<T>::ConstIterator Container<T>::ConstIterator::operator++()
00197 {
00198   ++curIdxIt_;
00199   return *this;
00200 }
00201
00202 template <std::floating_point T>
00203 typename Container<T>::ConstIterator Container<T>::ConstIterator::operator++(int)
00204 {
00205   auto tmp = *this;
00206   operator++();
00207   return tmp;
00208 }
00209
00210 template <std::floating_point T>
00211 typename Container<T>::ConstIterator::reference Container<T>::ConstIterator::operator*() const
00212 {
00213   // return cont_->triangleByIndex(*curIdxIt_);
00214   return cont_->triangleByIndex(*curIdxIt_);
00215 }
00216
00217 template <std::floating_point T>
00218 typename Container<T>::ConstIterator::pointer Container<T>::ConstIterator::operator->() const
00219 {
00220   // return cont_->triangleByIndex(*curIdxIt_);
00221   return &cont_->triangleByIndex(*curIdxIt_);
00222 }
00223
00224 template <std::floating_point T>
00225 bool Container<T>::ConstIterator::operator==(const Container<T>::ConstIterator &lhs) const
00226 {
00227   return (cont_ == lhs.cont_) && (curIdxIt_ == lhs.curIdxIt_);
00228 }
00229
00230 template <std::floating_point T>
00231 bool Container<T>::ConstIterator::operator!=(const Container<T>::ConstIterator &lhs) const
00232 {
00233   return !operator==(lhs);
00234 }
00235
00236 } // namespace geom::kdtree
00237
00238 #endif // __INCLUDE_KDTREE_CONTAINER_HH__
```

6.9 include/kdtree/kdtree.hh File Reference

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

```
#include "container.hh"
#include "node.hh"
```

Include dependency graph for kdtree.hh:



Classes

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

Namespaces

- geom

  line.hh Line class implementation
- geom::kdtree

6.10 kdtree.hh

```
00001 #ifndef __INCLUDE_KDTREE_KDTREE_HH__
00002 #define __INCLUDE_KDTREE_KDTREE_HH__
00003
00004 #include <cassert>
00005 #include <functional>
00006 #include <initializer_list>
00007 #include <memory>
00008 #include <queue>
00009 #include <stack>
00010 #include <vector>
00011
00012 #include "primitives/primitives.hh"
00013
00014 #include "container.hh"
00015 #include "node.hh"
00016
00017 namespace geom::kdtree
00018 {
00019
00020 template <std::floating_point T>
```

```
00021 class KdTree
00022 {
00023 private:
00024   std::unique_ptr<Node<T>> root_;
00025   std::vector<Triangle<T>> triangles_;
00026   std::size_t nodeCapacity_{1};
00027
00028 public:
00029   KdTree(std::initializer_list<Triangle<T>> il);
00030   KdTree(const KdTree &tree);
00031   KdTree(KdTree &&tree) = default;
00032   KdTree() = default;
00033   ~KdTree();
00034
00035   KdTree &operator=(const KdTree &tree);
00036   KdTree &operator=(KdTree &&tree) = default;
00037
00038   class ConstIterator;
00039
00040   // ConstIterators
00041   ConstIterator cbegin() const;
00042   ConstIterator cend() const;
00043
00044   ConstIterator begin() const;
00045   ConstIterator end() const;
00046
00047   ConstIterator beginFrom(const ConstIterator &iter) const;
00048
00049   // Modifiers
00050   void insert(const Triangle<T> &tr);
00051   void clear();
00052   void setNodeCapacity(std::size_t newCap);
00053
00054   // Capacity
00055   bool empty() const;
00056   std::size_t size() const;
00057   std::size_t nodeCapacity() const;
00058
00059   const Triangle<T> &triangleByIndex(Index index) const;
00060
00061   void dumpRecursive(std::ostream &ost = std::cout) const;
00062
00063   static bool isOnPosSide(Axis axis, T separator, const Triangle<T> &tr);
00064   static bool isOnNegSide(Axis axis, T separator, const Triangle<T> &tr);
00065   static bool isOnSide(Axis axis, T separator, const Triangle<T> &tr,
00066                        std::function<bool(T, T)> comparator);
00067
00068 private:
00069   void expandingInsert(const Triangle<T> &tr);
00070   void tryExpandRight(Axis axis, const BoundBox<T> &trianBB);
00071   void tryExpandLeft(Axis axis, const BoundBox<T> &trianBB);
00072
00073   void nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv = false);
00074   bool isDivisable(const Node<T> *node);
00075   void subdivide(Node<T> *node);
00076
00077 public:
00078   struct ContainerPtr final
00079   {
00080     Container<T> cont;
00081     const Container<T> *operator->() const;
00082   };
00083
00084   class ConstIterator final
00085   {
00086   public:
00087     using iterator_category = std::forward_iterator_tag;
00088     using difference_type = std::size_t;
00089     using value_type = Container<T>;
00090     using reference = Container<T>;
00091     using pointer = ContainerPtr;
00092
00093   private:
00094     const KdTree<T> *tree_;
00095     const Node<T> *node_;
00096     std::queue<const Node<T> *> fifo_;
00097
00098   public:
00099     ConstIterator(const KdTree<T> *tree, const Node<T> *node);
00100     ConstIterator(const ConstIterator &iter) = default;
00101     ConstIterator(ConstIterator &&iter) = default;
00102
00103     ConstIterator &operator=(const ConstIterator &cont) = default;
00104     ConstIterator &operator=(ConstIterator &&cont) = default;
00105
00106     ~ConstIterator() = default;
00107
```

```
00108     ConstIterator &operator++();
00109     ConstIterator operator++(int);
00110
00111     reference operator*() const;
00112     pointer operator->() const;
00113
00114     bool operator==(const ConstIterator &lhs) const;
00115     bool operator!=(const ConstIterator &lhs) const;
00116
00117     static ConstIterator beginFrom(const ConstIterator &iter);
00118   };
00119 };
00120
00121 //================================================================================================
00122 //                                    KdTree definitions
00123 //================================================================================================
00124
00125 template <std::floating_point T>
00126 KdTree<T>::KdTree(std::initializer_list<Triangle<T>> il)
00127 {
00128   for (const auto &tr : il)
00129     insert(tr);
00130 }
00131
00132 template <std::floating_point T>
00133 KdTree<T>::KdTree(const KdTree<T> &tree)
00134 {
00135   // temporary solution
00136   for (const auto &tr : tree.triangles_)
00137     insert(tr);
00138 }
00139
00140 template <std::floating_point T>
00141 KdTree<T>::~KdTree()
00142 {
00143   clear();
00144 }
00145
00146 template <std::floating_point T>
00147 KdTree<T> &KdTree<T>::operator=(const KdTree<T> &tree)
00148 {
00149   KdTree tmp{tree};
00150   operator=(std::move(tmp));
00151   return *this;
00152 }
00153
00154 // ConstIterators
00155 template <std::floating_point T>
00156 typename KdTree<T>::ConstIterator KdTree<T>::cbegin() const
00157 {
00158   return ConstIterator{this, root_.get()};
00159 }
00160
00161 template <std::floating_point T>
00162 typename KdTree<T>::ConstIterator KdTree<T>::cend() const
00163 {
00164   return ConstIterator{this, nullptr};
00165 }
00166
00167 template <std::floating_point T>
00168 typename KdTree<T>::ConstIterator KdTree<T>::begin() const
00169 {
00170   return cbegin();
00171 }
00172
00173 template <std::floating_point T>
00174 typename KdTree<T>::ConstIterator KdTree<T>::end() const
00175 {
00176   return cend();
00177 }
00178
00179 template <std::floating_point T>
00180 typename KdTree<T>::ConstIterator KdTree<T>::beginFrom(
00181   const typename KdTree<T>::ConstIterator &iter) const
00182 {
00183   return KdTree<T>::ConstIterator::beginFrom(iter);
00184 }
00185
00186 // Modifiers
00187 template <std::floating_point T>
00188 void KdTree<T>::insert(const Triangle<T> &tr)
00189 {
00190   if (nullptr == root_)
00191   {
00192     root_ = std::unique_ptr<Node<T>>{new Node<T>{T{}, Axis::NONE, tr.boundBox(), {0}}};
00193     triangles_.push_back(tr);
00194     return;
```

```
00195   }
00196
00197   if (!tr.belongsTo(root_->boundBox))
00198     expandingInsert(tr);
00199   else
00200   {
00201     auto index = triangles_.size();
00202     triangles_.push_back(tr);
00203     nonExpandingInsert(root_.get(), tr, index);
00204   }
00205 }
00206
00207 template <std::floating_point T>
00208 void KdTree<T>::clear()
00209 {
00210   if (nullptr == root_)
00211     return;
00212
00213   std::stack<std::unique_ptr<Node<T» *> stack{};
00214   stack.push(&root_);
00215
00216   while (!stack.empty())
00217   {
00218     auto *curNode = stack.top();
00219     auto *right = &curNode->get()->right;
00220     auto *left = &curNode->get()->left;
00221
00222     if ((nullptr == *right) && (nullptr == *left))
00223     {
00224       curNode->reset();
00225       stack.pop();
00226       continue;
00227     }
00228
00229     stack.push(right);
00230     stack.push(left);
00231   }
00232 }
00233
00234 template <std::floating_point T>
00235 void KdTree<T>::setNodeCapacity(std::size_t newCap)
00236 {
00237   nodeCapacity_ = newCap;
00238 }
00239
00240 // Capacity
00241 template <std::floating_point T>
00242 bool KdTree<T>::empty() const
00243 {
00244   return triangles_.empty();
00245 }
00246
00247 template <std::floating_point T>
00248 std::size_t KdTree<T>::size() const
00249 {
00250   return triangles_.size();
00251 }
00252
00253 template <std::floating_point T>
00254 std::size_t KdTree<T>::nodeCapacity() const
00255 {
00256   return nodeCapacity_;
00257 }
00258
00259 template <std::floating_point T>
00260 const Triangle<T> &KdTree<T>::triangleByIndex(Index index) const
00261 {
00262   return triangles_[index];
00263 }
00264
00265 template <std::floating_point T>
00266 void KdTree<T>::dumpRecursive(std::ostream &ost) const
00267 {
00268   ost « "digraph kdtree {" « std::endl;
00269   if (root_)
00270     root_->dumpRecursive(ost);
00271   ost « "}" « std::endl;
00272 }
00273
00274 template <std::floating_point T>
00275 bool KdTree<T>::isOnPosSide(Axis axis, T separator, const Triangle<T> &tr)
00276 {
00277   return isOnSide(axis, separator, tr, std::greater<T>{});
00278 }
00279
00280 template <std::floating_point T>
00281 bool KdTree<T>::isOnNegSide(Axis axis, T separator, const Triangle<T> &tr)
```

```
00282 {
00283   return isOnSide(axis, separator, tr, std::less<T>{});
00284 }
00285
00286 template <std::floating_point T>
00287 bool KdTree<T>::isOnSide(Axis axis, T separator, const Triangle<T> &tr,
00288                          std::function<bool(T, T)> comparator)
00289 {
00290   if (Axis::NONE == axis)
00291     return false;
00292
00293   auto axisIdx = static_cast<size_t>(axis);
00294   for (size_t i = 0; i < 3; ++i)
00295     if (!comparator(tr[i][axisIdx], separator))
00296       return false;
00297
00298   return true;
00299 }
00300
00301 template <std::floating_point T>
00302 void KdTree<T>::expandingInsert(const Triangle<T> &tr)
00303 {
00304   auto trianBB = tr.boundBox();
00305   auto index = triangles_.size();
00306   triangles_.push_back(tr);
00307
00308   for (auto axis : {Axis::X, Axis::Y, Axis::Z})
00309     tryExpandRight(axis, trianBB);
00310
00311   for (auto axis : {Axis::X, Axis::Y, Axis::Z})
00312     tryExpandLeft(axis, trianBB);
00313
00314   root_->indicies.push_back(index);
00315 }
00316
00317 template <std::floating_point T>
00318 void KdTree<T>::tryExpandRight(Axis axis, const BoundBox<T> &trianBB)
00319 {
00320   const auto &rootBB = root_->boundBox;
00321   if (trianBB.max(axis) <= rootBB.max(axis))
00322     return;
00323
00324   auto newRightBB = rootBB;
00325   newRightBB.min(axis) = rootBB.max(axis);
00326   newRightBB.max(axis) = trianBB.max(axis);
00327
00328   auto newRootBB = rootBB;
00329   newRootBB.max(axis) = newRightBB.max(axis);
00330
00331   std::unique_ptr<Node<T>> newRight{new Node<T>{T{}, Axis::NONE, newRightBB}};
00332   std::unique_ptr<Node<T>> newRoot{new Node<T>{rootBB.max(axis), axis, newRootBB}};
00333
00334   newRoot->right = std::move(newRight);
00335   newRoot->left = std::move(root_);
00336
00337   root_ = std::move(newRoot);
00338 }
00339
00340 template <std::floating_point T>
00341 void KdTree<T>::tryExpandLeft(Axis axis, const BoundBox<T> &trianBB)
00342 {
00343   const auto &rootBB = root_->boundBox;
00344   if (trianBB.min(axis) >= rootBB.min(axis))
00345     return;
00346
00347   BoundBox<T> newLeftBB = rootBB;
00348   newLeftBB.max(axis) = rootBB.min(axis);
00349   newLeftBB.min(axis) = trianBB.min(axis);
00350
00351   BoundBox<T> newRootBB = rootBB;
00352   newRootBB.min(axis) = newLeftBB.min(axis);
00353
00354   std::unique_ptr<Node<T>> newLeft{new Node<T>{T{}, Axis::NONE, newLeftBB}};
00355   std::unique_ptr<Node<T>> newRoot{new Node<T>{rootBB.min(axis), axis, newRootBB}};
00356
00357   newRoot->left = std::move(newLeft);
00358   newRoot->right = std::move(root_);
00359
00360   root_ = std::move(newRoot);
00361 }
00362
00363 template <std::floating_point T>
00364 void KdTree<T>::nonExpandingInsert(Node<T> *node, const Triangle<T> &tr, Index index, bool isSubdiv)
00365 {
00366   auto curNode = node;
00367   while (true)
00368   {
```

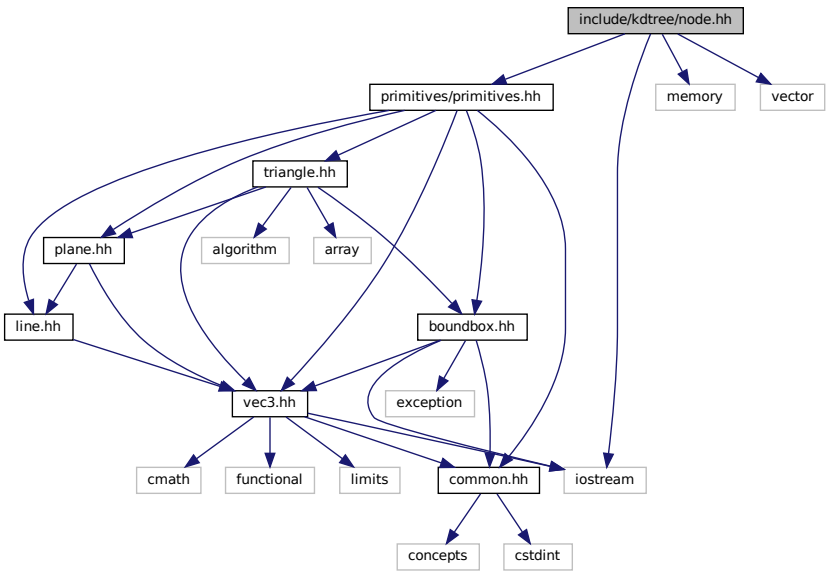
```
00369     if (isOnPosSide(curNode->sepAxis, curNode->separator, tr))
00370       curNode = curNode->right.get();
00371     else if (isOnNegSide(curNode->sepAxis, curNode->separator, tr))
00372       curNode = curNode->left.get();
00373     else
00374       break;
00375   }
00376
00377   curNode->indicies.push_back(index);
00378   if (isDivisable(curNode) && (!isSubdiv))
00379     subdivide(curNode);
00380 }
00381
00382 template <std::floating_point T>
00383 bool KdTree<T>::isDivisable(const Node<T> *node)
00384 {
00385   return (node->indicies.size() > nodeCapacity_) && (node->sepAxis == Axis::NONE);
00386 }
00387
00388 template <std::floating_point T>
00389 void KdTree<T>::subdivide(Node<T> *node)
00390 {
00391   const auto &nodeBB = node->boundBox;
00392   auto axis = node->sepAxis = nodeBB.getMaxDim();
00393   auto sep = node->separator = nodeBB.min(axis) + 0.5 * (nodeBB.max(axis) - nodeBB.min(axis));
00394
00395   auto newRightBB = nodeBB;
00396   auto newLeftBB = nodeBB;
00397
00398   newRightBB.min(axis) = newLeftBB.max(axis) = sep;
00399   node->right.reset(new Node<T>{T{}, Axis::NONE, newRightBB});
00400   node->left.reset(new Node<T>{T{}, Axis::NONE, newLeftBB});
00401
00402   auto indicies = node->indicies;
00403   node->indicies.clear();
00404
00405   for (auto index : indicies)
00406     nonExpandingInsert(node, triangles_[index], index, /* isSubdiv = */ true);
00407 }
00408
00409 //=============================================================================================
00410 //                              KdTree::ContainerPtr definitions
00411 //=============================================================================================
00412
00413 template <std::floating_point T>
00414 const Container<T> *KdTree<T>::ContainerPtr::operator->() const
00415 {
00416   return &cont;
00417 }
00418
00419 //=============================================================================================
00420 //                              KdTree::ConstIterator definitions
00421 //=============================================================================================
00422
00423 template <std::floating_point T>
00424 KdTree<T>::ConstIterator::ConstIterator(const KdTree<T> *tree, const Node<T> *node)
00425   : tree_(tree), node_(node), fifo_({node})
00426 {}
00427
00428 template <std::floating_point T>
00429 typename KdTree<T>::ConstIterator &KdTree<T>::ConstIterator::operator++()
00430 {
00431   if (0 == fifo_.size())
00432     return *this;
00433
00434   auto fifoEntry = fifo_.front();
00435   fifo_.pop();
00436
00437   if (Axis::NONE != fifoEntry->sepAxis)
00438   {
00439     if (nullptr != fifoEntry->left)
00440       fifo_.push(fifoEntry->left.get());
00441     if (nullptr != fifoEntry->right)
00442       fifo_.push(fifoEntry->right.get());
00443   }
00444
00445   node_ = (0 == fifo_.size()) ? nullptr : fifo_.front();
00446   return *this;
00447 }
00448
00449 template <std::floating_point T>
00450 typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::operator++(int)
00451 {
00452   auto tmp = *this;
00453   operator++();
00454   return tmp;
00455 }
```

```
00456
00457 template <std::floating_point T>
00458 typename KdTree<T>::ConstIterator::reference KdTree<T>::ConstIterator::operator*() const
00459 {
00460   return Container<T>{tree_, node_};
00461 }
00462
00463 template <std::floating_point T>
00464 typename KdTree<T>::ConstIterator::pointer KdTree<T>::ConstIterator::operator->() const
00465 {
00466   return ContainerPtr{{tree_, node_}};
00467 }
00468
00469 template <std::floating_point T>
00470 bool KdTree<T>::ConstIterator::operator==(const KdTree<T>::ConstIterator &lhs) const
00471 {
00472   return (tree_ == lhs.tree_) && (node_ == lhs.node_);
00473 }
00474
00475 template <std::floating_point T>
00476 bool KdTree<T>::ConstIterator::operator!=(const KdTree<T>::ConstIterator &lhs) const
00477 {
00478   return !operator==(lhs);
00479 }
00480
00481 template <std::floating_point T>
00482 typename KdTree<T>::ConstIterator KdTree<T>::ConstIterator::beginFrom(
00483   const typename KdTree<T>::ConstIterator &iter)
00484 {
00485   return ConstIterator{iter.tree_, iter.node_};
00486 }
00487
00488 } // namespace geom::kdtree
00489
00490 #endif // __INCLUDE_KDTREE_KDTREE_HH__
```

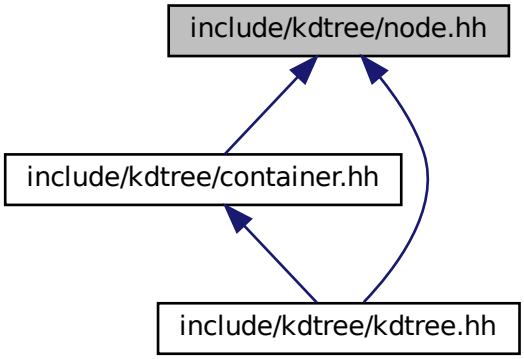
6.11 include/kdtree/node.hh File Reference

```
#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 = size_t

6.12 node.hh

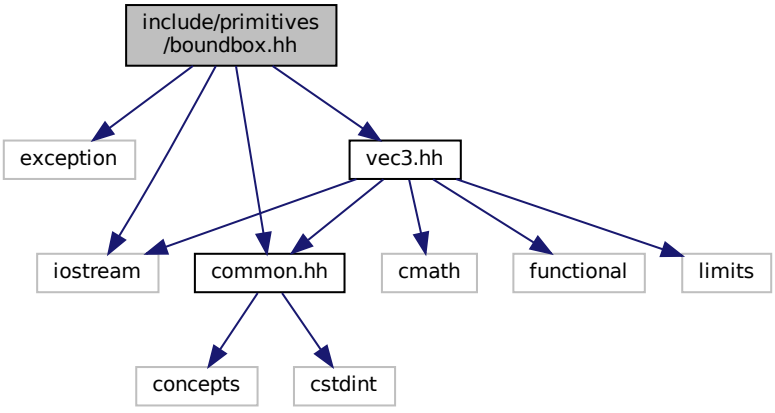
```
00001 #ifndef __INCLUDE_KDTREE_NODE_HH__
00002 #define __INCLUDE_KDTREE_NODE_HH__
00003
00004 #include <iostream>
00005 #include <memory>
00006 #include <vector>
00007
00008 #include "primitives/primitives.hh"
00009
00010 namespace geom::kdtree
00011 {
00012
00013 using Index = size_t;
00014
00015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

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

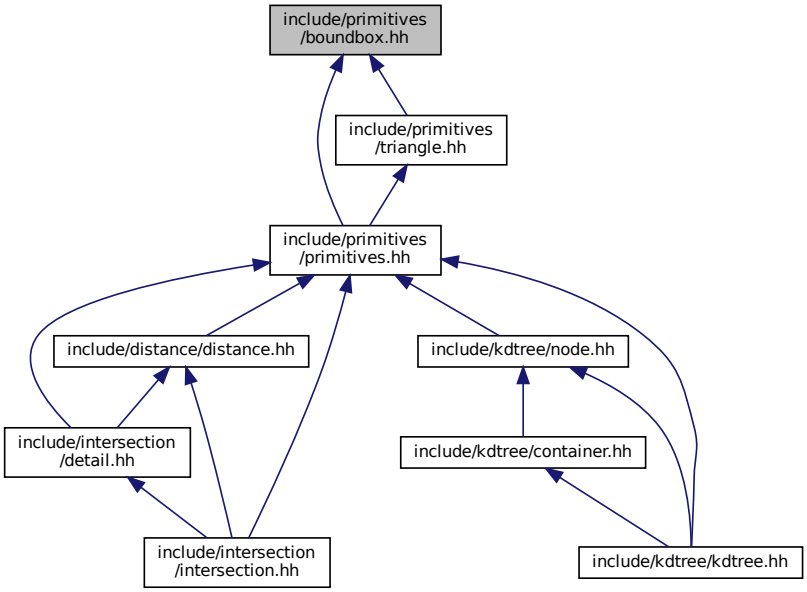
6.13 include/primitives/boundbox.hh File Reference

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

Include dependency graph for boundbox.hh:



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



Classes

- struct geom::BoundBox< T >

Namespaces

- geom

  line.hh Line class implementation

Functions

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

6.14 boundbox.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
00002 #define __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
00003
00004 #include <exception>
00005 #include <iostream>
00006
00007 #include "common.hh"
00008 #include "vec3.hh"
00009
00010 namespace geom
00011 {
00012
```

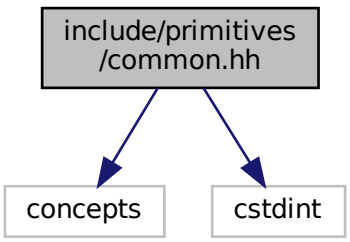
```
00013 template <std::floating_point T>
00014 struct BoundBox
00015 {
00016   T minX{};
00017   T maxX{};
00018
00019   T minY{};
00020   T maxY{};
00021
00022   T minZ{};
00023   T maxZ{};
00024
00025   bool belongsTo(const BoundBox<T> &bb);
00026
00027   T &min(Axis axis);
00028   T &max(Axis axis);
00029
00030   const T &min(Axis axis) const;
00031   const T &max(Axis axis) const;
00032
00033   Axis getMaxDim() const;
00034 };
00035
00036 template <std::floating_point T>
00037 bool BoundBox<T>::belongsTo(const BoundBox<T> &bb)
00038 {
00039   return (minX >= bb.minX) && (minY >= bb.minY) && (minZ >= bb.minZ) && (maxX <= bb.maxX) &&
00040          (maxY <= bb.maxY) && (maxZ <= bb.maxZ);
00041 }
00042
00043 template <std::floating_point T>
00044 T &BoundBox<T>::min(Axis axis)
00045 {
00046   switch (axis)
00047   {
00048   case Axis::X:
00049     return minX;
00050   case Axis::Y:
00051     return minY;
00052   case Axis::Z:
00053     return minZ;
00054   default:
00055     throw std::logic_error("BoundBox<T>::min(): Wrong input axis");
00056   }
00057 }
00058
00059 template <std::floating_point T>
00060 T &BoundBox<T>::max(Axis axis)
00061 {
00062   switch (axis)
00063   {
00064   case Axis::X:
00065     return maxX;
00066   case Axis::Y:
00067     return maxY;
00068   case Axis::Z:
00069     return maxZ;
00070   default:
00071     throw std::logic_error("BoundBox<T>::max(): Wrong input axis");
00072   }
00073 }
00074
00075 template <std::floating_point T>
00076 const T &BoundBox<T>::min(Axis axis) const
00077 {
00078   switch (axis)
00079   {
00080   case Axis::X:
00081     return minX;
00082   case Axis::Y:
00083     return minY;
00084   case Axis::Z:
00085     return minZ;
00086   default:
00087     throw std::logic_error("BoundBox<T>::min(): Wrong input axis");
00088   }
00089 }
00090
00091 template <std::floating_point T>
00092 const T &BoundBox<T>::max(Axis axis) const
00093 {
00094   switch (axis)
00095   {
00096   case Axis::X:
00097     return maxX;
00098   case Axis::Y:
00099     return maxY;
```

```
00100   case Axis::Z:
00101     return maxZ;
00102   default:
00103     throw std::logic_error("BoundBox<T>::max(): Wrong input axis");
00104   }
00105 }
00106
00107 template <std::floating_point T>
00108 Axis BoundBox<T>::getMaxDim() const
00109 {
00110   return std::max({Axis::X, Axis::Y, Axis::Z}, [this](const auto &lhs, const auto &rhs) {
00111     return (this->max(lhs) - this->min(lhs)) < (this->max(rhs) - this->min(rhs));
00112   });
00113 }
00114
00115 template <std::floating_point T>
00116 bool operator==(const BoundBox<T> &lhs, const BoundBox<T> &rhs)
00117 {
00118   return Vec3<T>::isNumEq(lhs.minX, rhs.minX) && Vec3<T>::isNumEq(lhs.maxX, rhs.maxX) &&
00119          Vec3<T>::isNumEq(lhs.minY, rhs.minY) && Vec3<T>::isNumEq(lhs.maxY, rhs.maxY) &&
00120          Vec3<T>::isNumEq(lhs.minZ, rhs.minZ) && Vec3<T>::isNumEq(lhs.maxY, rhs.maxY);
00121 }
00122
00123 template <std::floating_point T>
00124 std::ostream &operator«(std::ostream &ost, const BoundBox<T> &bb)
00125 {
00126   ost « "BB: {\\n";
00127   ost « "  x: [" « bb.minX « "; " « bb.maxX « "],\\n";
00128   ost « "  y: [" « bb.minY « "; " « bb.maxY « "],\\n";
00129   ost « "  z: [" « bb.minZ « "; " « bb.maxZ « "]\\n}";
00130   return ost;
00131 }
00132
00133 } // namespace geom
00134
00135 #endif // __INCLUDE_PRIMITIVES_BOUNDBOX_HH__
```

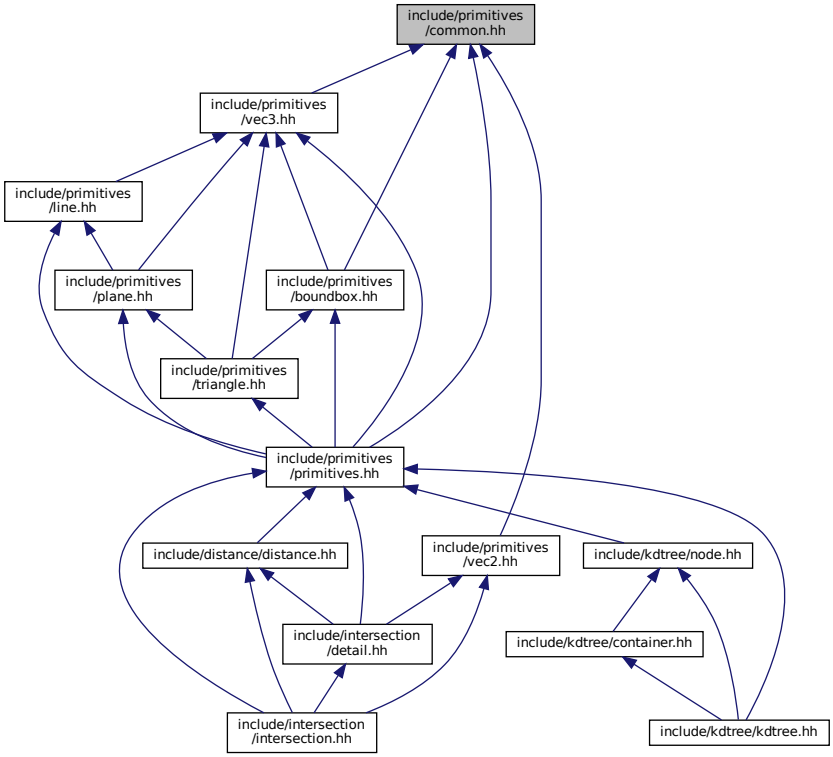
6.15 include/primitives/common.hh File Reference

```
#include <concepts>
#include <cstdint>
```

Include dependency graph for common.hh:



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



Namespaces

- geom

  line.hh Line class implementation

Enumerations

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

Variables

- template<class T >
  concept geom::Number = std::is_floating_point_v<T> || std::is_integral_v<T>

  Useful concept which represents floating point and integral types.

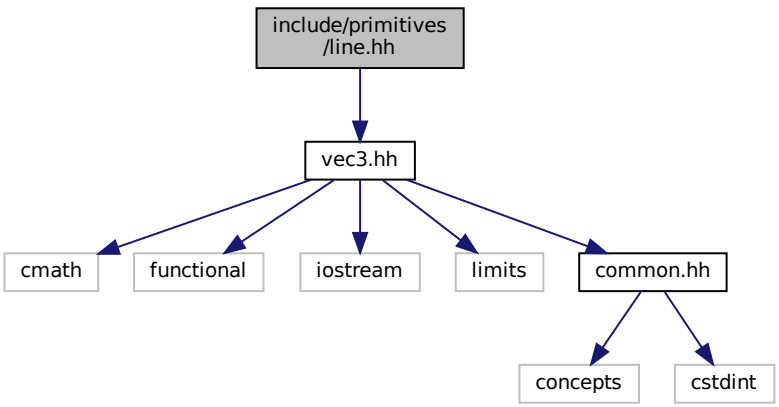
6.16 common.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_COMMON_HH__
00002 #define __INCLUDE_PRIMITIVES_COMMON_HH__
00003
00004 #include <concepts>
00005 #include <cstdint>
00006
00007 namespace geom
00008 {
00009 /**
00010  * @concept Number
00011  * @brief Useful concept which represents floating point and integral types
00012  *
00013  * @tparam T
00014  */
00015 template <class T>
00016 concept Number = std::is_floating_point_v<T> || std::is_integral_v<T>;
00017
00018 enum class Axis : std::int8_t
00019 {
00020   X = 0,
00021   Y = 1,
00022   Z = 2,
00023   NONE
00024 };
00025
00026 } // namespace geom
00027
00028 #endif // __INCLUDE_PRIMITIVES_COMMON_HH__
```

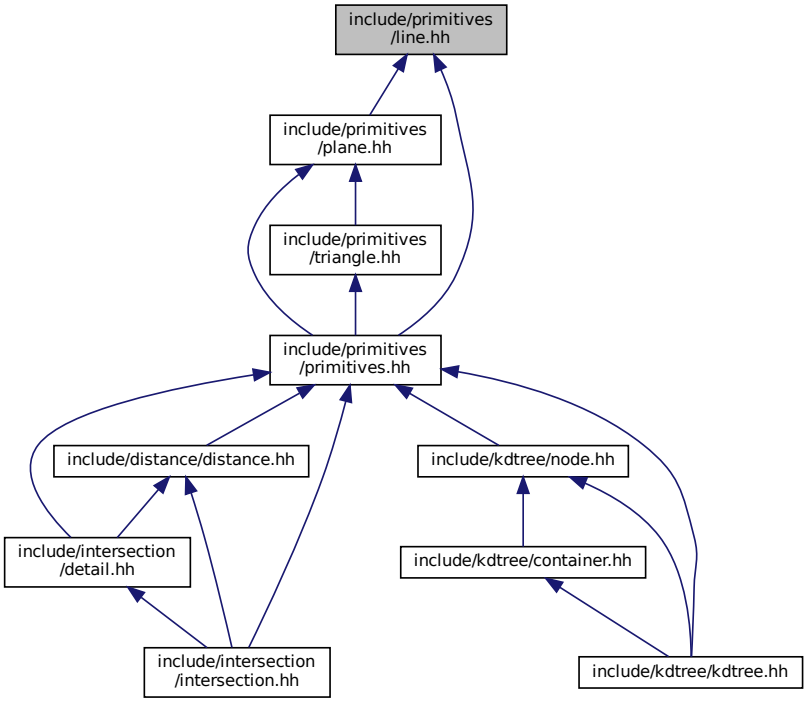
6.17 include/primitives/line.hh File Reference

`#include "vec3.hh"`

Include dependency graph for line.hh:



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



Classes

- class geom::Line< T >

  Line class implementation.

Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Line print operator.

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

  Line equality operator.

6.18 line.hh

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

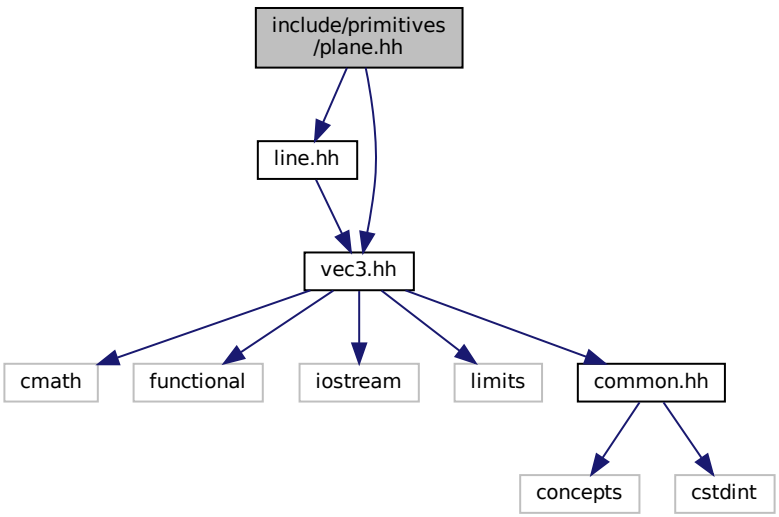
```
00086     */
00087    bool isPar(const Line &line) const;
00088
00089    /**
00090     * @brief Checks is *this is skew with another line
00091     *
00092     * @param[in] line const reference to another line
00093     * @return true if lines are skew
00094     * @return false if lines are not skew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

```
00173   return belongs(line.org_) && dir_.isPar(line.dir_);
00174 }
00175
00176 template <std::floating_point T>
00177 bool Line<T>::isPar(const Line<T> &line) const
00178 {
00179   return dir_.isPar(line.dir_);
00180 }
00181
00182 template <std::floating_point T>
00183 bool Line<T>::isSkew(const Line<T> &line) const
00184 {
00185   auto res = triple(line.org_ - org_, dir_, line.dir_);
00186   return !Vec3<T>::isNumEq(res, T{0});
00187 }
00188
00189 template <std::floating_point T>
00190 Line<T> Line<T>::getBy2Points(const Vec3<T> &p1, const Vec3<T> &p2)
00191 {
00192   return Line<T>{p1, p2 - p1};
00193 }
00194
00195 } // namespace geom
00196
00197 #endif // __INCLUDE_PRIMITIVES_LINE_HH__
```

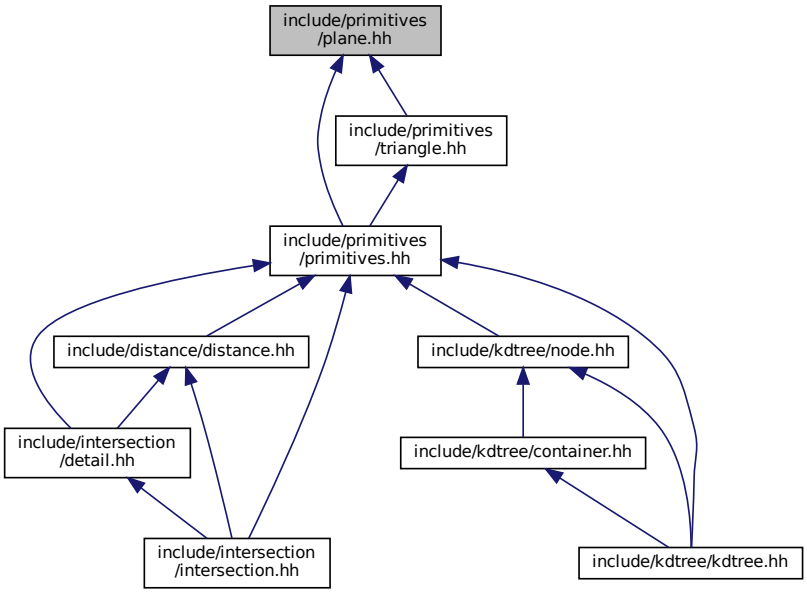
6.19 include/primitives/plane.hh File Reference

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

Include dependency graph for plane.hh:



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



Classes

- class geom::Plane< T >

  Plane class realization.

Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Plane equality operator.

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

  Plane print operator.

6.20 plane.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_PLANE_HH__
00002 #define __INCLUDE_PRIMITIVES_PLANE_HH__
00003
00004 #include "line.hh"
00005 #include "vec3.hh"
00006
00007 /**
00008  * @brief
00009  * Plane class implementation
00010  */
00011
00012 namespace geom
00013 {
00014
00015 /**
00016  * @class Plane
00017  * @brief Plane class realization
00018  *
00019  * @tparam T - floating point type of coordinates
00020  */
00021 template <std::floating_point T>
00022 class Plane final
00023 {
00024 private:
00025   /**
00026    * @brief Normal vector, length equals to 1
00027    */
00028   Vec3<T> norm_{};
00029
00030   /**
00031    * @brief Distance from zero to plane
00032    */
00033   T dist_{};
00034
00035   /**
00036    * @brief Construct a new Plane object from normal vector and distance
00037    *
00038    * @param[in] norm normal vector
00039    * @param[in] dist distance from plane to zero
00040    */
00041   Plane(const Vec3<T> &norm, T dist);
00042
00043 public:
00044   /**
00045    * @brief Getter for distance
00046    *
00047    * @return T value of distance
00048    */
00049   T dist() const;
00050
00051   /**
00052    * @brief Getter for normal vector
00053    *
00054    * @return const Vec3<T>& const reference to normal vector
00055    */
00056   const Vec3<T> &norm() const;
00057
00058   /**
00059    * @brief Checks if point belongs to plane
00060    *
00061    * @param[in] point const referene to point vector
00062    * @return true if point belongs to plane
00063    * @return false if point doesn't belong to plane
00064    */
00065   bool belongs(const Vec3<T> &point) const;
00066
00067   /**
00068    * @brief Checks if line belongs to plane
00069    *
00070    * @param[in] line const referene to line
00071    * @return true if line belongs to plane
00072    * @return false if line doesn't belong to plane
00073    */
00074   bool belongs(const Line<T> &line) const;
00075
00076   /**
00077    * @brief Checks is *this equals to another plane
00078    *
00079    * @param[in] rhs const reference to another plane
00080    * @return true if planes are equal
00081    * @return false if planes are not equal
00082    */
00083   bool isEqual(const Plane &rhs) const;
00084
00085   /**
```

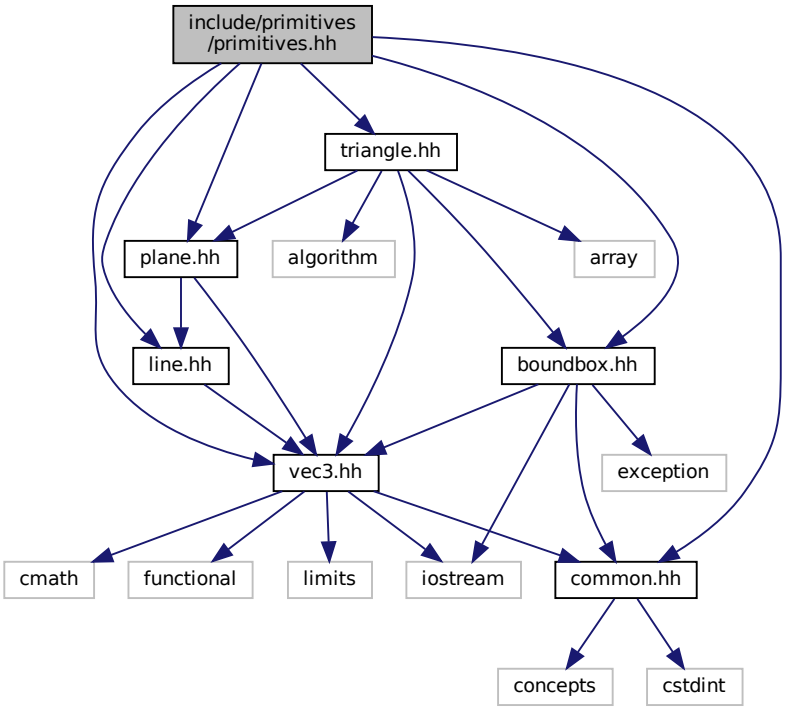
```
00086     * @brief Checks is *this is parallel to another plane
00087     *
00088     * @param[in] rhs const reference to another plane
00089     * @return true if planes are parallel
00090     * @return false if planes are not parallel
00091     */
00092    bool isPar(const Plane &rhs) const;
00093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

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

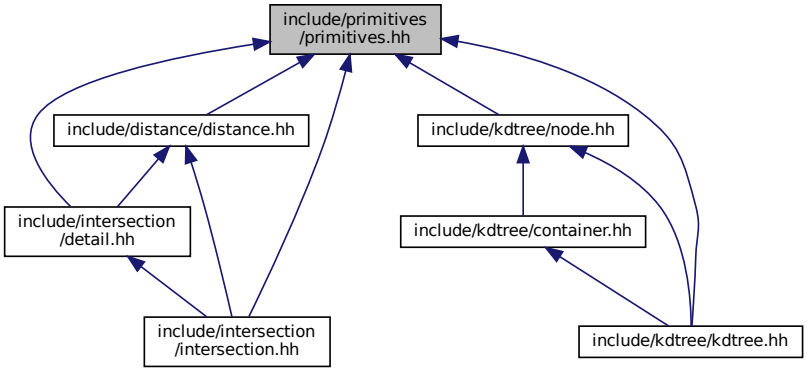
6.21 include/primitives/primitives.hh File Reference

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

Include dependency graph for primitives.hh:



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



6.22 primitives.hh

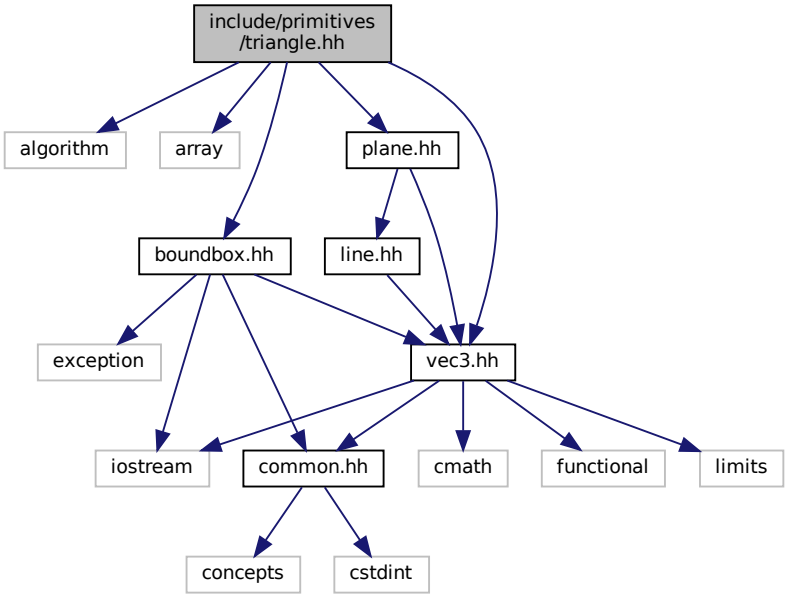
```
00001 #ifndef __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00002 #define __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
00003
00004 #include "boundbox.hh"
00005 #include "common.hh"
```

```
00006 #include "line.hh"
00007 #include "plane.hh"
00008 #include "triangle.hh"
00009 #include "vec3.hh"
00010
00011 #endif // __INCLUDE_PRIMITIVES_PRIMITIVES_HH__
```

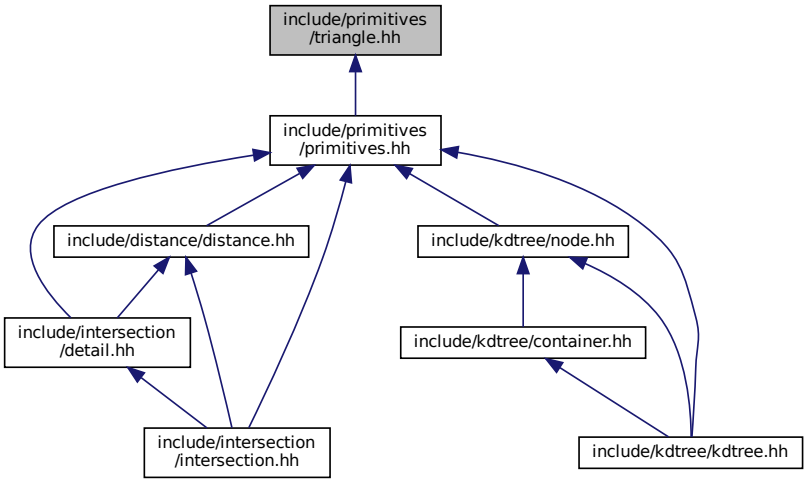
6.23 include/primitives/triangle.hh File Reference

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

Include dependency graph for triangle.hh:



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



Classes

- class geom::Triangle< T >

  Triangle class implementation.

Namespaces

- geom

  line.hh Line class implementation

Functions

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

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

6.24 triangle.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00002 #define __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00003
00004 #include <algorithm>
00005 #include <array>
00006
00007 #include "boundbox.hh"
00008 #include "plane.hh"
00009 #include "vec3.hh"
00010
00011 /**
00012  * @brief triangle.hh
00013  * Triangle class implementation
```

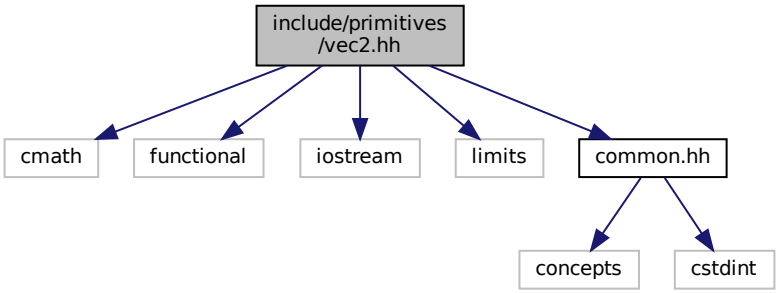
```
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Triangle
00021  * @brief Triangle class implementation
00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025 template <std::floating_point T>
00026 class Triangle final
00027 {
00028 private:
00029   /**
00030    * @brief Vertices of triangle
00031    */
00032   std::array<Vec3<T>, 3> vertices_;
00033
00034 public:
00035   /**
00036    * @brief Construct a new Triangle object
00037    */
00038   Triangle();
00039
00040   /**
00041    * @brief Construct a new Triangle object from 3 points
00042    *
00043    * @param[in] p1 1st point
00044    * @param[in] p2 2nd point
00045    * @param[in] p3 3rd point
00046    */
00047   Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3);
00048
00049   /**
00050    * @brief Overloaded operator[] to get access to vertices
00051    *
00052    * @param[in] idx index of vertex
00053    * @return const Vec3<T>& const reference to vertex
00054    */
00055   const Vec3<T> &operator[](std::size_t idx) const;
00056
00057   /**
00058    * @brief Overloaded operator[] to get access to vertices
00059    *
00060    * @param[in] idx index of vertex
00061    * @return Vec3<T>& reference to vertex
00062    */
00063   Vec3<T> &operator[](std::size_t idx);
00064
00065   /**
00066    * @brief Get triangle's plane
00067    *
00068    * @return Plane<T>
00069    */
00070   Plane<T> getPlane() const;
00071
00072   /**
00073    * @brief Check is triangle valid
00074    *
00075    * @return true if triangle is valid
00076    * @return false if triangle is invalid
00077    */
00078   bool isValid() const;
00079
00080   /**
00081    * @brief Returns triangle's bound box
00082    *
00083    * @return BoundBox<T>
00084    */
00085   BoundBox<T> boundBox() const;
00086
00087   /**
00088    * @brief Checks if this Triangle belongs to BoundBox
00089    *
00090    * @param[in] bb BoundBox
00091    * @return true if Triangle belongs to BoundBox
00092    * @return false if Triangle doesn't belong to BoundBox
00093    */
00094   bool belongsTo(const BoundBox<T> &bb) const;
00095 };
00096
00097 /**
00098  * @brief Triangle print operator
00099  *
00100  * @tparam T - floating point type of coordinates
```

```
00101  * @param[in, out] ost output stream
00102  * @param[in] tr Triangle to print
00103  * @return std::ostream& modified ostream instance
00104  */
00105 template <std::floating_point T>
00106 std::ostream &operator«(std::ostream &ost, const Triangle<T> &tr)
00107 {
00108   ost « "Triangle: {";
00109   for (size_t i = 0; i < 3; ++i)
00110     ost « tr[i] « (i == 2 ? "" : ", ");
00111
00112   ost « "}";
00113
00114   return ost;
00115 }
00116
00117 template <std::floating_point T>
00118 std::istream &operator»(std::istream &ist, Triangle<T> &tr)
00119 {
00120   ist » tr[0] » tr[1] » tr[2];
00121   return ist;
00122 }
00123
00124 template <std::floating_point T>
00125 Triangle<T>::Triangle() : vertices_()
00126 {}
00127
00128 template <std::floating_point T>
00129 Triangle<T>::Triangle(const Vec3<T> &p1, const Vec3<T> &p2, const Vec3<T> &p3)
00130   : vertices_{p1, p2, p3}
00131 {}
00132
00133 template <std::floating_point T>
00134 const Vec3<T> &Triangle<T>::operator[](std::size_t idx) const
00135 {
00136   return vertices_[idx % 3];
00137 }
00138
00139 template <std::floating_point T>
00140 Vec3<T> &Triangle<T>::operator[](std::size_t idx)
00141 {
00142   return vertices_[idx % 3];
00143 }
00144
00145 template <std::floating_point T>
00146 Plane<T> Triangle<T>::getPlane() const
00147 {
00148   return Plane<T>::getBy3Points(vertices_[0], vertices_[1], vertices_[2]);
00149 }
00150
00151 template <std::floating_point T>
00152 bool Triangle<T>::isValid() const
00153 {
00154   auto edge1 = vertices_[1] - vertices_[0];
00155   auto edge2 = vertices_[2] - vertices_[0];
00156
00157   auto cross12 = cross(edge1, edge2);
00158   return (cross12 != Vec3<T>{});
00159 }
00160
00161 template <std::floating_point T>
00162 BoundBox<T> Triangle<T>::boundBox() const
00163 {
00164   auto minMaxX = std::minmax({vertices_[0].x, vertices_[1].x, vertices_[2].x});
00165   auto minMaxY = std::minmax({vertices_[0].y, vertices_[1].y, vertices_[2].y});
00166   auto minMaxZ = std::minmax({vertices_[0].z, vertices_[1].z, vertices_[2].z});
00167
00168   return {minMaxX.first - Vec3<T>::getThreshold(), minMaxX.second + Vec3<T>::getThreshold(),
00169           minMaxY.first - Vec3<T>::getThreshold(), minMaxY.second + Vec3<T>::getThreshold(),
00170           minMaxZ.first - Vec3<T>::getThreshold(), minMaxZ.second + Vec3<T>::getThreshold()};
00171 }
00172
00173 template <std::floating_point T>
00174 bool Triangle<T>::belongsTo(const BoundBox<T> &bb) const
00175 {
00176   return boundBox().belongsTo(bb);
00177 }
00178
00179 } // namespace geom
00180
00181 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__
```

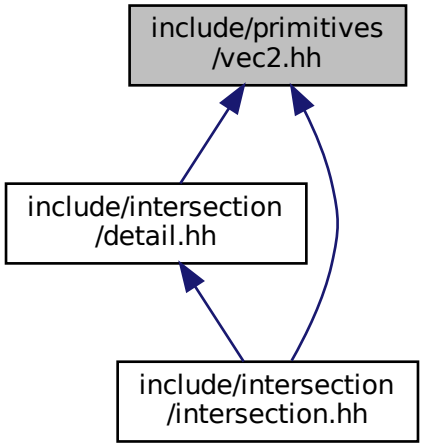
6.25 include/primitives/vec2.hh File Reference

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

Include dependency graph for vec2.hh:



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



Classes

- class geom::Vec2< T >

  Vec2 class realization.

Namespaces

- geom

  line.hh Line class implementation

Typedefs

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

Functions

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

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

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

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

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

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

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

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

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

  Vec2 print operator.

6.25.1 Detailed Description

Vec2 class implementation

Definition in file vec2.hh.

6.26 vec2.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_VEC2_HH__
00002 #define __INCLUDE_PRIMITIVES_VEC2_HH__
00003
00004 #include <cmath>
00005 #include <functional>
00006 #include <iostream>
00007 #include <limits>
00008
00009 #include "common.hh"
00010
00011 /**
00012  * @file vec2.hh
00013  * Vec2 class implementation
00014  */
00015
00016 namespace geom
00017 {
00018
00019 /**
00020  * @class Vec2
00021  * @brief Vec2 class realization
00022  *
00023  * @tparam T - floating point type of coordinates
00024  */
00025 template <std::floating_point T>
00026 struct Vec2 final
00027 {
00028 private:
00029   /**
00030    * @brief Threshold static variable for numbers comparision
00031    */
00032   static inline T threshold_ = 1e3 * std::numeric_limits<T>::epsilon();
00033
00034 public:
00035   /**
00036    * @brief Vec2 coordinates
00037    */
00038   T x{}, y{};
00039
00040   /**
00041    * @brief Construct a new Vec2 object from 3 coordinates
00042    *
00043    * @param[in] coordX x coordinate
00044    * @param[in] coordY y coordinate
00045    */
00046   Vec2(T coordX, T coordY) : x(coordX), y(coordY)
00047   {}
00048
00049   /**
00050    * @brief Construct a new Vec2 object with equals coordinates
00051    *
00052    * @param[in] coordX coordinate (default to {})
00053    */
00054   explicit Vec2(T coordX = {}) : Vec2(coordX, coordX)
00055   {}
00056
00057   /**
00058    * @brief Overloaded += operator
00059    * Increments vector coordinates by corresponding coordinates of vec
00060    * @param[in] vec vector to incremented with
00061    * @return Vec2& reference to current instance
00062    */
00063   Vec2 &operator+=(const Vec2 &vec);
00064
00065   /**
00066    * @brief Overloaded -= operator
00067    * Decrements vector coordinates by corresponding coordinates of vec
00068    * @param[in] vec vector to decremented with
00069    * @return Vec2& reference to current instance
00070    */
00071   Vec2 &operator-=(const Vec2 &vec);
00072
00073   /**
00074    * @brief Unary - operator
00075    *
00076    * @return Vec2 negated Vec2 instance
00077    */
00078   Vec2 operator-() const;
00079
00080   /**
00081    * @brief Overloaded *= by number operator
00082    *
00083    * @tparam nType numeric type of value to multiply by
00084    * @param[in] val value to multiply by
00085    * @return Vec2& reference to vector instance
```

```
00086     */
00087    template <Number nType>
00088    Vec2 &operator*=(nType val);
00089
00090    /**
00091     * @brief Overloaded /= by number operator
00092     *
00093     * @tparam nType numeric type of value to divide by
00094     * @param[in] val value to divide by
00095     * @return Vec2& reference to vector instance
00096     *
00097     * @warning Does not check if val equals 0
00098     */
00099    template <Number nType>
00100    Vec2 &operator/=(nType val);
00101
00102    /**
00103     * @brief Dot product function
00104     *
00105     * @param rhs vector to dot product with
00106     * @return T dot product of two vectors
00107     */
00108    T dot(const Vec2 &rhs) const;
00109
00110    /**
00111     * @brief Calculate squared length of a vector function
00112     *
00113     * @return T length^2
00114     */
00115    T length2() const;
00116
00117    /**
00118     * @brief Calculate length of a vector function
00119     *
00120     * @return T length
00121     */
00122    T length() const;
00123
00124    /**
00125     * @brief Get the perpendicular to this vector
00126     *
00127     * @return Vec2 perpendicular vector
00128     */
00129    Vec2 getPerp() const;
00130
00131    /**
00132     * @brief Get normalized vector function
00133     *
00134     * @return Vec2 normalized vector
00135     */
00136    Vec2 normalized() const;
00137
00138    /**
00139     * @brief Normalize vector function
00140     *
00141     * @return Vec2& reference to instance
00142     */
00143    Vec2 &normalize();
00144
00145    /**
00146     * @brief Overloaded operator [] (non-const version)
00147     * To get access to coordinates
00148     * @param i index of coordinate (0 - x, 1 - y)
00149     * @return T& reference to coordinate value
00150     *
00151     * @note Coordinates calculated by mod 2
00152     */
00153    T &operator[](size_t i);
00154
00155    /**
00156     * @brief Overloaded operator [] (const version)
00157     * To get access to coordinates
00158     * @param i index of coordinate (0 - x, 1 - y)
00159     * @return T coordinate value
00160     *
00161     * @note Coordinates calculated by mod 2
00162     */
00163    T operator[](size_t i) const;
00164
00165    /**
00166     * @brief Check if vector is parallel to another
00167     *
00168     * @param[in] rhs vector to check parallelism with
00169     * @return true if vector is parallel
00170     * @return false otherwise
00171     */
00172    bool isPar(const Vec2 &rhs) const;
```

```
00173
00174   /**
00175    * @brief Check if vector is perpendicular to another
00176    *
00177    * @param[in] rhs vector to check perpendicularity with
00178    * @return true if vector is perpendicular
00179    * @return false otherwise
00180    */
00181   bool isPerp(const Vec2 &rhs) const;
00182
00183   /**
00184    * @brief Check if vector is equal to another
00185    *
00186    * @param[in] rhs vector to check equality with
00187    * @return true if vector is equal
00188    * @return false otherwise
00189    *
00190    * @note Equality check performs using isNumEq(T lhs, T rhs) function
00191    */
00192   bool isEqual(const Vec2 &rhs) const;
00193
00194   /**
00195    * @brief Check equality (with threshold) of two floating point numbers function
00196    *
00197    * @param[in] lhs first number
00198    * @param[in] rhs second number
00199    * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
00200    * @return false otherwise
00201    *
00202    * @note Threshold defined by threshold_ static member
00203    */
00204   static bool isNumEq(T lhs, T rhs);
00205
00206   /**
00207    * @brief Set new threshold value
00208    *
00209    * @param[in] thres value to set
00210    */
00211   static void setThreshold(T thres);
00212
00213   /**
00214    * @brief Get current threshold value
00215    */
00216   static T getThreshold();
00217
00218   /**
00219    * @brief Set threshold to default value
00220    * @note default value equals float point epsilon
00221    */
00222   static void setDefThreshold();
00223 };
00224
00225 /**
00226  * @brief Overloaded + operator
00227  *
00228  * @tparam T vector template parameter
00229  * @param[in] lhs first vector
00230  * @param[in] rhs second vector
00231  * @return Vec2<T> sum of two vectors
00232  */
00233 template <std::floating_point T>
00234 Vec2<T> operator+(const Vec2<T> &lhs, const Vec2<T> &rhs)
00235 {
00236   Vec2<T> res{lhs};
00237   res += rhs;
00238   return res;
00239 }
00240
00241 /**
00242  * @brief Overloaded - operator
00243  *
00244  * @tparam T vector template parameter
00245  * @param[in] lhs first vector
00246  * @param[in] rhs second vector
00247  * @return Vec2<T> res of two vectors
00248  */
00249 template <std::floating_point T>
00250 Vec2<T> operator-(const Vec2<T> &lhs, const Vec2<T> &rhs)
00251 {
00252   Vec2<T> res{lhs};
00253   res -= rhs;
00254   return res;
00255 }
00256
00257 /**
00258  * @brief Overloaded multiple by value operator
00259  *
```

```
00260  * @tparam nT type of value to multiply by
00261  * @tparam T vector template parameter
00262  * @param[in] val value to multiply by
00263  * @param[in] rhs vector to multiply by value
00264  * @return Vec2<T> result vector
00265  */
00266 template <Number nT, std::floating_point T>
00267 Vec2<T> operator*(const nT &val, const Vec2<T> &rhs)
00268 {
00269   Vec2<T> res{rhs};
00270   res *= val;
00271   return res;
00272 }
00273
00274 /**
00275  * @brief Overloaded multiple by value operator
00276  *
00277  * @tparam nT type of value to multiply by
00278  * @tparam T vector template parameter
00279  * @param[in] val value to multiply by
00280  * @param[in] lhs vector to multiply by value
00281  * @return Vec2<T> result vector
00282  */
00283 template <Number nT, std::floating_point T>
00284 Vec2<T> operator*(const Vec2<T> &lhs, const nT &val)
00285 {
00286   Vec2<T> res{lhs};
00287   res *= val;
00288   return res;
00289 }
00290
00291 /**
00292  * @brief Overloaded divide by value operator
00293  *
00294  * @tparam nT type of value to divide by
00295  * @tparam T vector template parameter
00296  * @param[in] val value to divide by
00297  * @param[in] lhs vector to divide by value
00298  * @return Vec2<T> result vector
00299  */
00300 template <Number nT, std::floating_point T>
00301 Vec2<T> operator/(const Vec2<T> &lhs, const nT &val)
00302 {
00303   Vec2<T> res{lhs};
00304   res /= val;
00305   return res;
00306 }
00307
00308 /**
00309  * @brief Dot product function
00310  *
00311  * @tparam T vector template parameter
00312  * @param[in] lhs first vector
00313  * @param[in] rhs second vector
00314  * @return T dot production
00315  */
00316 template <std::floating_point T>
00317 T dot(const Vec2<T> &lhs, const Vec2<T> &rhs)
00318 {
00319   return lhs.dot(rhs);
00320 }
00321
00322 /**
00323  * @brief Vec2 equality operator
00324  *
00325  * @tparam T vector template parameter
00326  * @param[in] lhs first vector
00327  * @param[in] rhs second vector
00328  * @return true if vectors are equal
00329  * @return false otherwise
00330  */
00331 template <std::floating_point T>
00332 bool operator==(const Vec2<T> &lhs, const Vec2<T> &rhs)
00333 {
00334   return lhs.isEqual(rhs);
00335 }
00336
00337 /**
00338  * @brief Vec2 inequality operator
00339  *
00340  * @tparam T vector template parameter
00341  * @param[in] lhs first vector
00342  * @param[in] rhs second vector
00343  * @return true if vectors are not equal
00344  * @return false otherwise
00345  */
00346 template <std::floating_point T>
```

```
00347 bool operator!=(const Vec2<T> &lhs, const Vec2<T> &rhs)
00348 {
00349   return !(lhs == rhs);
00350 }
00351
00352 /**
00353  * @brief Vec2 print operator
00354  *
00355  * @tparam T vector template parameter
00356  * @param[in, out] ost output stream
00357  * @param[in] vec vector to print
00358  * @return std::ostream& modified stream instance
00359  */
00360 template <std::floating_point T>
00361 std::ostream &operator«(std::ostream &ost, const Vec2<T> &vec)
00362 {
00363   ost « "(" « vec.x « ", " « vec.y « ")";
00364   return ost;
00365 }
00366
00367 using Vec2D = Vec2<double>;
00368 using Vec2F = Vec2<float>;
00369
00370 template <std::floating_point T>
00371 Vec2<T> &Vec2<T>::operator+=(const Vec2 &vec)
00372 {
00373   x += vec.x;
00374   y += vec.y;
00375
00376   return *this;
00377 }
00378
00379 template <std::floating_point T>
00380 Vec2<T> &Vec2<T>::operator-=(const Vec2 &vec)
00381 {
00382   x -= vec.x;
00383   y -= vec.y;
00384
00385   return *this;
00386 }
00387
00388 template <std::floating_point T>
00389 Vec2<T> Vec2<T>::operator-() const
00390 {
00391   return Vec2{-x, -y};
00392 }
00393
00394 template <std::floating_point T>
00395 template <Number nType>
00396 Vec2<T> &Vec2<T>::operator*=(nType val)
00397 {
00398   x *= val;
00399   y *= val;
00400
00401   return *this;
00402 }
00403
00404 template <std::floating_point T>
00405 template <Number nType>
00406 Vec2<T> &Vec2<T>::operator/=(nType val)
00407 {
00408   x /= static_cast<T>(val);
00409   y /= static_cast<T>(val);
00410
00411   return *this;
00412 }
00413
00414 template <std::floating_point T>
00415 T Vec2<T>::dot(const Vec2 &rhs) const
00416 {
00417   return x * rhs.x + y * rhs.y;
00418 }
00419
00420 template <std::floating_point T>
00421 T Vec2<T>::length2() const
00422 {
00423   return dot(*this);
00424 }
00425
00426 template <std::floating_point T>
00427 T Vec2<T>::length() const
00428 {
00429   return std::sqrt(length2());
00430 }
00431
00432 template <std::floating_point T>
00433 Vec2<T> Vec2<T>::getPerp() const
```

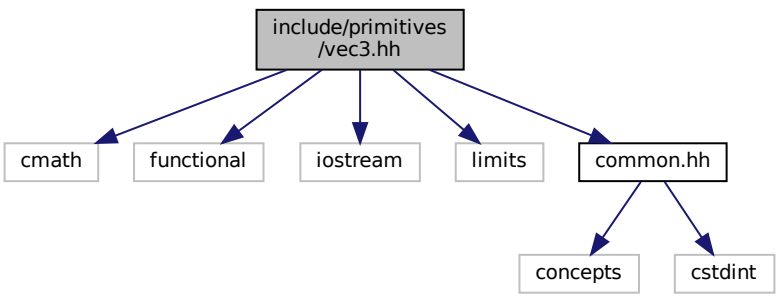
```
00434 {
00435   return {y, -x};
00436 }
00437
00438 template <std::floating_point T>
00439 Vec2<T> Vec2<T>::normalized() const
00440 {
00441   Vec2 res{*this};
00442   res.normalize();
00443   return res;
00444 }
00445
00446 template <std::floating_point T>
00447 Vec2<T> &Vec2<T>::normalize()
00448 {
00449   T len2 = length2();
00450   if (isNumEq(len2, 0) || isNumEq(len2, 1))
00451     return *this;
00452   return *this /= std::sqrt(len2);
00453 }
00454
00455 template <std::floating_point T>
00456 T &Vec2<T>::operator[](size_t i)
00457 {
00458   switch (i % 2)
00459   {
00460   case 0:
00461     return x;
00462   case 1:
00463     return y;
00464   default:
00465     throw std::logic_error{"Impossible case in operator[]\n"};
00466   }
00467 }
00468
00469 template <std::floating_point T>
00470 T Vec2<T>::operator[](size_t i) const
00471 {
00472   switch (i % 2)
00473   {
00474   case 0:
00475     return x;
00476   case 1:
00477     return y;
00478   default:
00479     throw std::logic_error{"Impossible case in operator[]\n"};
00480   }
00481 }
00482
00483 template <std::floating_point T>
00484 bool Vec2<T>::isPar(const Vec2 &rhs) const
00485 {
00486   auto det = x * rhs.y - rhs.x * y;
00487   return isNumEq(det, 0);
00488 }
00489
00490 template <std::floating_point T>
00491 bool Vec2<T>::isPerp(const Vec2 &rhs) const
00492 {
00493   return isNumEq(dot(rhs), 0);
00494 }
00495
00496 template <std::floating_point T>
00497 bool Vec2<T>::isEqual(const Vec2 &rhs) const
00498 {
00499   return isNumEq(x, rhs.x) && isNumEq(y, rhs.y);
00500 }
00501
00502 template <std::floating_point T>
00503 bool Vec2<T>::isNumEq(T lhs, T rhs)
00504 {
00505   return std::abs(rhs - lhs) < threshold_;
00506 }
00507
00508 template <std::floating_point T>
00509 void Vec2<T>::setThreshold(T thres)
00510 {
00511   threshold_ = thres;
00512 }
00513
00514 template <std::floating_point T>
00515 T Vec2<T>::getThreshold()
00516 {
00517   return threshold_;
00518 }
00519
00520 template <std::floating_point T>
```

```
00521 void Vec2<T>::setDefThreshold()
00522 {
00523   threshold_ = std::numeric_limits<T>::epsilon();
00524 }
00525
00526 } // namespace geom
00527
00528 #endif // __INCLUDE_PRIMITIVES_VEC2_HH__
```

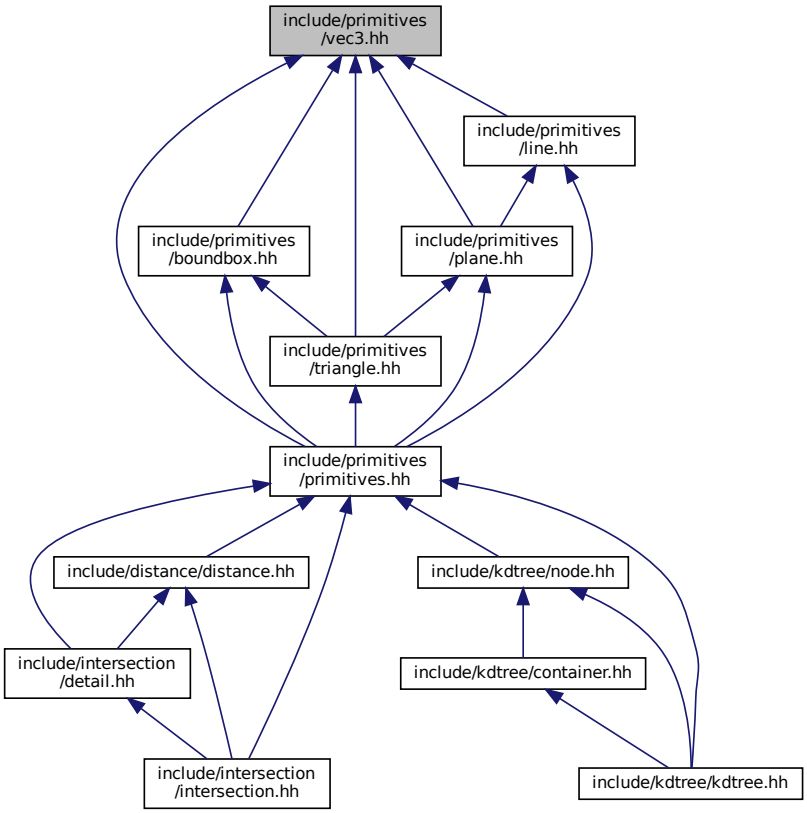
6.27 include/primitives/vec3.hh File Reference

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

Include dependency graph for vec3.hh:



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



Classes

- class geom::Vec3< T >

  Vec3 class realization.

Namespaces

- geom

  line.hh Line class implementation

Typedefs

- using geom::Vec3D = Vec3< double >
- using geom::Vec3F = Vec3< float >

Functions

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

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

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

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec3< T > geom::operator* (const Vec3< T > &lhs, const nT &val)

  Overloaded multiple by value operator.
- template<Number nT, std::floating_point T>
  Vec3< T > geom::operator/ (const Vec3< T > &lhs, const nT &val)

  Overloaded divide by value operator.
- template<std::floating_point T>
  T geom::dot (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Dot product function.
- template<std::floating_point T>
  Vec3< T > geom::cross (const Vec3< T > &lhs, const Vec3< T > &rhs)

  Cross product function.
- template<std::floating_point T>
  T geom::triple (const Vec3< T > &v1, const Vec3< T > &v2, const Vec3< T > &v3)

  Triple product function.
- template<std::floating_point T>
  bool geom::operator== (const Vec3< T > &lhs, const Vec3< T > &rhs)

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

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

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

  Vec3 scan operator.

6.27.1 Detailed Description

Vec3 class implementation

Definition in file vec3.hh.

6.28 vec3.hh

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

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

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

```
00260  * @brief Overloaded multiple by value operator
00261  *
00262  * @tparam nT type of value to multiply by
00263  * @tparam T vector template parameter
00264  * @param[in] val value to multiply by
00265  * @param[in] rhs vector to multiply by value
00266  * @return Vec3<T> result vector
00267  */
00268 template <Number nT, std::floating_point T>
00269 Vec3<T> operator*(const nT &val, const Vec3<T> &rhs)
00270 {
00271   Vec3<T> res{rhs};
00272   res *= val;
00273   return res;
00274 }
00275
00276 /**
00277  * @brief Overloaded multiple by value operator
00278  *
00279  * @tparam nT type of value to multiply by
00280  * @tparam T vector template parameter
00281  * @param[in] val value to multiply by
00282  * @param[in] lhs vector to multiply by value
00283  * @return Vec3<T> result vector
00284  */
00285 template <Number nT, std::floating_point T>
00286 Vec3<T> operator*(const Vec3<T> &lhs, const nT &val)
00287 {
00288   Vec3<T> res{lhs};
00289   res *= val;
00290   return res;
00291 }
00292
00293 /**
00294  * @brief Overloaded divide by value operator
00295  *
00296  * @tparam nT type of value to divide by
00297  * @tparam T vector template parameter
00298  * @param[in] val value to divide by
00299  * @param[in] lhs vector to divide by value
00300  * @return Vec3<T> result vector
00301  */
00302 template <Number nT, std::floating_point T>
00303 Vec3<T> operator/(const Vec3<T> &lhs, const nT &val)
00304 {
00305   Vec3<T> res{lhs};
00306   res /= val;
00307   return res;
00308 }
00309
00310 /**
00311  * @brief Dot product function
00312  *
00313  * @tparam T vector template parameter
00314  * @param[in] lhs first vector
00315  * @param[in] rhs second vector
00316  * @return T dot production
00317  */
00318 template <std::floating_point T>
00319 T dot(const Vec3<T> &lhs, const Vec3<T> &rhs)
00320 {
00321   return lhs.dot(rhs);
00322 }
00323
00324 /**
00325  * @brief Cross product function
00326  *
00327  * @tparam T vector template parameter
00328  * @param[in] lhs first vector
00329  * @param[in] rhs second vector
00330  * @return T cross production
00331  */
00332 template <std::floating_point T>
00333 Vec3<T> cross(const Vec3<T> &lhs, const Vec3<T> &rhs)
00334 {
00335   return lhs.cross(rhs);
00336 }
00337
00338 /**
00339  * @brief Triple product function
00340  *
00341  * @tparam T vector template parameter
00342  * @param[in] v1 first vector
00343  * @param[in] v2 second vector
00344  * @param[in] v3 third vector
00345  * @return T triple production
00346  */
```

```
00347 template <std::floating_point T>
00348 T triple(const Vec3<T> &v1, const Vec3<T> &v2, const Vec3<T> &v3)
00349 {
00350   return dot(v1, cross(v2, v3));
00351 }
00352
00353 /**
00354  * @brief Vec3 equality operator
00355  *
00356  * @tparam T vector template parameter
00357  * @param[in] lhs first vector
00358  * @param[in] rhs second vector
00359  * @return true if vectors are equal
00360  * @return false otherwise
00361  */
00362 template <std::floating_point T>
00363 bool operator==(const Vec3<T> &lhs, const Vec3<T> &rhs)
00364 {
00365   return lhs.isEqual(rhs);
00366 }
00367
00368 /**
00369  * @brief Vec3 inequality operator
00370  *
00371  * @tparam T vector template parameter
00372  * @param[in] lhs first vector
00373  * @param[in] rhs second vector
00374  * @return true if vectors are not equal
00375  * @return false otherwise
00376  */
00377 template <std::floating_point T>
00378 bool operator!=(const Vec3<T> &lhs, const Vec3<T> &rhs)
00379 {
00380   return !(lhs == rhs);
00381 }
00382
00383 /**
00384  * @brief Vec3 print operator
00385  *
00386  * @tparam T vector template parameter
00387  * @param[in, out] ost output stream
00388  * @param[in] vec vector to print
00389  * @return std::ostream& modified stream instance
00390  */
00391 template <std::floating_point T>
00392 std::ostream &operator«(std::ostream &ost, const Vec3<T> &vec)
00393 {
00394   ost « "(" « vec.x « ", " « vec.y « ", " « vec.z « ")";
00395   return ost;
00396 }
00397
00398 /**
00399  * @brief Vec3 scan operator
00400  *
00401  * @tparam T vector template parameter
00402  * @param[in, out] ist input stram
00403  * @param[in, out] vec vector to scan
00404  * @return std::istream& modified stream instance
00405  */
00406 template <std::floating_point T>
00407 std::istream &operator»(std::istream &ist, Vec3<T> &vec)
00408 {
00409   ist » vec.x » vec.y » vec.z;
00410   return ist;
00411 }
00412
00413 using Vec3D = Vec3<double>;
00414 using Vec3F = Vec3<float>;
00415
00416 template <std::floating_point T>
00417 Vec3<T> &Vec3<T>::operator+=(const Vec3 &vec)
00418 {
00419   x += vec.x;
00420   y += vec.y;
00421   z += vec.z;
00422
00423   return *this;
00424 }
00425
00426 template <std::floating_point T>
00427 Vec3<T> &Vec3<T>::operator-=(const Vec3 &vec)
00428 {
00429   x -= vec.x;
00430   y -= vec.y;
00431   z -= vec.z;
00432
00433   return *this;
```

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

```
00521 template <std::floating_point T>
00522 T Vec3<T>::operator[](size_t i) const
00523 {
00524   switch (i % 3)
00525   {
00526   case 0:
00527     return x;
00528   case 1:
00529     return y;
00530   case 2:
00531     return z;
00532   default:
00533     throw std::logic_error{"Impossible case in operator[]\n"};
00534   }
00535 }
00536
00537 template <std::floating_point T>
00538 bool Vec3<T>::isPar(const Vec3 &rhs) const
00539 {
00540   return cross(rhs).isEqual(Vec3<T>{0});
00541 }
00542
00543 template <std::floating_point T>
00544 bool Vec3<T>::isPerp(const Vec3 &rhs) const
00545 {
00546   return isNumEq(dot(rhs), 0);
00547 }
00548
00549 template <std::floating_point T>
00550 bool Vec3<T>::isEqual(const Vec3 &rhs) const
00551 {
00552   return isNumEq(x, rhs.x) && isNumEq(y, rhs.y) && isNumEq(z, rhs.z);
00553 }
00554
00555 template <std::floating_point T>
00556 bool Vec3<T>::isNumEq(T lhs, T rhs)
00557 {
00558   return std::abs(rhs - lhs) < threshold_;
00559 }
00560
00561 template <std::floating_point T>
00562 void Vec3<T>::setThreshold(T thres)
00563 {
00564   threshold_ = thres;
00565 }
00566
00567 template <std::floating_point T>
00568 T Vec3<T>::getThreshold()
00569 {
00570   return threshold_;
00571 }
00572
00573 template <std::floating_point T>
00574 void Vec3<T>::setDefThreshold()
00575 {
00576   threshold_ = std::numeric_limits<T>::epsilon();
00577 }
00578
00579 } // namespace geom
00580
00581 #endif // __INCLUDE_PRIMITIVES_VEC3_HH__
```