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

Classes

- class Line
  Line class implementation.
- class Plane
  Plane class realization.
- class Triangle
  Triangle class implementation.
- class Vector
  Vector class realization.

Typedefs

- using VectorD = Vector< double >
- using VectorF = Vector< float >

Functions

- 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>
  Vector< T > operator+ (const Vector< T > &lhs, const Vector< T > &rhs)

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

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

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

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

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

  Dot product operator.
- template<std::floating_point T>
  Vector< T > operator% (const Vector< T > &lhs, const Vector< T > &rhs)

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

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

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

  Vector print 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 VectorD

```
using geom::VectorD = typedef Vector<double>
```

Definition at line 393 of file vector.hh.

4.1.2.2 VectorF

```
using geom::VectorF = typedef Vector<float>
```

Definition at line 394 of file vector.hh.

4.1.3 Function Documentation

4.1.3.1 operator<<() [1/4]

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

Line print operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

std::ostream& modified ostream instance

Definition at line 89 of file line.hh.

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

4.1.3.2 operator==() [1/3]

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

Line equality operator.

Template Parameters

T	- floating point type of coordinates

Parameters

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

Returns

true if lines are equal

false if lines are not equal

Definition at line 105 of file line.hh.

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

4.1.3.3 operator==() [2/3]

```
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 138 of file plane.hh.

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

4.1.3.4 operator<<() [2/4]

```
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 152 of file plane.hh.

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

4.1.3.5 operator<<() [3/4]

```
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 60 of file triangle.hh.

4.1.3.6 operator+()

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

Overloaded + operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vector<T> sum of two vectors

Definition at line 246 of file vector.hh.

4.1.3.7 operator-()

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

Overloaded - operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

Vector<T> res of two vectors

Definition at line 262 of file vector.hh.

4.1.3.8 operator*() [1/2]

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

Vector<T> result vector

Definition at line 279 of file vector.hh.

4.1.3.9 operator*() [2/2]

```
template<Number nT, std::floating_point T>
Vector<T> geom::operator* (
```

```
        const Vector< 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

Vector<T> result vector

Definition at line 296 of file vector.hh.

4.1.3.10 operator/()

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

Vector<T> result vector

Definition at line 313 of file vector.hh.

4.1.3.11 operator&()

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

```
            const Vector< T > & lhs,
            const Vector< T > & rhs )
```

Dot product operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T dot production

Definition at line 329 of file vector.hh.

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

4.1.3.12 operator%()

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

Cross product operator.

Template Parameters

T	vector template parameter

Parameters

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

Returns

T cross production

Definition at line 343 of file vector.hh.

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

4.1.3.13 operator==() [3/3]

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

Vector 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 358 of file vector.hh.

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

4.1.3.14 operator"!=()

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

Vector 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 373 of file vector.hh.

4.1.3.15 operator<<() [4/4]

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

Vector 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 387 of file vector.hh.

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

4.1.4 Variable Documentation

4.1.4.1 Number

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

Useful concept which represents floating point and integral types.

@concept Number

Template Parameters

T	

Definition at line 25 of file vector.hh.

Chapter 5

Class Documentation

5.1 geom::Line< T > Class Template Reference

Line class implementation.

`#include <line.hh>`

Public Member Functions

- Line (const Vector< T > &org, const Vector< T > &dir)
  Construct a new Line object.
- const Vector< T > & org () const
  Getter for origin vector.
- const Vector< T > & dir () const
  Getter for direction vector.
- bool belongs (const Vector< T > &point) const
  Checks is point belongs to line.
- bool isEqual (const Line &line) const
  Checks is ∗this equals to another line.

Static Public Member Functions

- static Line getBy2Points (const Vector< T > &p1, const Vector< T > &p2)
  Get line by 2 points.

5.1.1 Detailed Description

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

Line class implementation.

Template Parameters

T	- floating point type of coordinates

Definition at line 21 of file line.hh.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Line()

```
template<std::floating_point T>
geom::Line< T >::Line (
            const Vector< T > & org,
            const Vector< T > & dir )
```

Construct a new Line object.

Parameters

in	*org*	origin vector
in	*dir*	direction vector

Definition at line 111 of file line.hh.

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

5.1.3 Member Function Documentation

5.1.3.1 org()

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

Getter for origin vector.

Returns

const Vector<T>& const reference to origin vector

Definition at line 118 of file line.hh.

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

5.1.3.2 dir()

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

Getter for direction vector.

Returns

const Vector<T>& const reference to direction vector

Definition at line 124 of file line.hh.

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

5.1.3.3 belongs()

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

Checks is point belongs to line.

Parameters

in	*point*	const reference to point vector

Returns

true if point belongs to line

false if point doesn't belong to line

Definition at line 130 of file line.hh.

5.1.3.4 isEqual()

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

Checks is *this equals to another line.

Parameters

in	*line*	const reference to another line

Returns

true if lines are equal

false if lines are not equal

Definition at line 136 of file line.hh.

Referenced by geom::operator==().

5.1.3.5 getBy2Points()

```
template<std::floating_point T>
Line< T > geom::Line< T >::getBy2Points (
            const Vector< T > & p1,
            const Vector< T > & p2 )  [static]
```

Get line by 2 points.

Parameters

in	*p1*	1st point
in	*p2*	2nd point

Returns

Line passing through two points

Definition at line 142 of file line.hh.

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

- include/primitives/line.hh

5.2 geom::Plane< T > Class Template Reference

Plane class realization.

`#include <plane.hh>`

Public Member Functions

- T dist () const
  Getter for distance.
- const Vector< T > & norm () const
  Getter for normal vector.
- bool belongs (const Vector< 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.

Static Public Member Functions

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

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

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

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

  Get plane form normal const plane equation.

5.2.1 Detailed Description

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

Plane class realization.

Template Parameters

T	- floating point type of coordinates

Definition at line 24 of file plane.hh.

5.2.2 Member Function Documentation

5.2.2.1 dist()

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

Getter for distance.

Returns

T value of distance

Definition at line 166 of file plane.hh.

Referenced by geom::operator<<().

5.2.2.2 norm()

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

Getter for normal vector.

Returns

const Vector<T>& const reference to normal vector

Definition at line 172 of file plane.hh.

Referenced by geom::operator<<().

5.2.2.3 belongs() [1/2]

```
template<std::floating_point T>
bool geom::Plane< T >::belongs (
            const Vector< 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 178 of file plane.hh.

5.2.2.4 belongs() [2/2]

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

Checks if line belongs to plane.

Parameters

in	*line*	const referene to line

Returns

true if line belongs to plane

false if line doesn't belong to plane

Definition at line 184 of file plane.hh.

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

5.2.2.5 isEqual()

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

Checks is ∗this equals to another plane.

Parameters

in	*rhs*	const reference to another plane

Returns

true if planes are equal

false if planes are not equal

Definition at line 190 of file plane.hh.

Referenced by geom::operator==().

5.2.2.6 getBy3Points()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getBy3Points (
            const Vector< T > & pt1,
            const Vector< T > & pt2,
            const Vector< 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 196 of file plane.hh.

5.2.2.7 getParametric()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getParametric (
            const Vector< T > & org,
            const Vector< T > & dir1,
            const Vector< 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 203 of file plane.hh.

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

5.2.2.8 getNormalPoint()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalPoint (
            const Vector< T > & norm,
            const Vector< 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 211 of file plane.hh.

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

5.2.2.9 getNormalDist()

```
template<std::floating_point T>
Plane< T > geom::Plane< T >::getNormalDist (
            const Vector< 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 218 of file plane.hh.

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

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

- include/primitives/plane.hh

5.3 geom::Triangle< T > Class Template Reference

Triangle class implementation.

`#include <triangle.hh>`

Public Member Functions

- Triangle (const Vector< T > &p1, const Vector< T > &p2, const Vector< T > &p3)
  Construct a new Triangle object from 3 points.
- const Vector< T > & operator[ ] (std::size_t idx) const
  Overloaded operator[] to get access to vertices.

5.3.1 Detailed Description

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

Triangle class implementation.

Template Parameters

T	- floating point type of coordinates

Definition at line 24 of file triangle.hh.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Triangle()

```
template<std::floating_point T>
geom::Triangle< T >::Triangle (
            const Vector< T > & p1,
            const Vector< T > & p2,
            const Vector< 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 72 of file triangle.hh.

5.3.3 Member Function Documentation

5.3.3.1 operator[]()

```
template<std::floating_point T>
const Vector< 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 Vector<T>& const reference to vertex

Definition at line 78 of file triangle.hh.

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

- include/primitives/triangle.hh

5.4 geom::Vector< T > Class Template Reference

Vector class realization.

`#include <vector.hh>`

Public Member Functions

- Vector (T coordX, T coordY, T coordZ)
  Construct a new Vector object from 3 coordinates.
- Vector (T coordX={})
  Construct a new Vector object with equals coordinates.
- Vector & operator+= (const Vector &vec)
  Overloaded += operator Increments vector coordinates by corresponding coordinates of vec.
- Vector & operator-= (const Vector &vec)
  Overloaded -= operator Decrements vector coordinates by corresponding coordinates of vec.
- Vector operator- () const
  Unary - operator.
- template<Number nType>
  Vector & operator*= (nType val)
  *Overloaded *= by number operator.*
- template<Number nType>
  Vector & operator/= (nType val)
  Overloaded /= by number operator.
- T dot (const Vector &rhs) const
  Dot product function.
- Vector cross (const Vector &rhs) const
  Cross product function.
- T length2 () const
  Calculate squared length of a vector function.
- T length () const
  Calculate length of a vector function.
- Vector normalized () const
  Get normalized vector function.
- Vector & 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 Vector &rhs) const

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

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

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

  Get current threshold value.
- static void setDefThreshold ()

  Set threshold to default value.

Public Attributes

- T x {}

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

5.4.1 Detailed Description

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

Vector class realization.

Template Parameters

T	- floating point type of coordinates

Definition at line 34 of file vector.hh.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Vector() [1/2]

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

Construct a new Vector object from 3 coordinates.

Parameters

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

Definition at line 55 of file vector.hh.

5.4.2.2 Vector() [2/2]

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

Construct a new Vector object with equals coordinates.

Parameters

in	*coordX*	coordinate (default to {})

Definition at line 64 of file vector.hh.

5.4.3 Member Function Documentation

5.4.3.1 operator+=()

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

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

Parameters

in	*vec*	vector to incremented with

Returns

Vector& reference to current instance

Definition at line 397 of file vector.hh.

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

5.4.3.2 operator-=()

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

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

Parameters

in	*vec*	vector to decremented with

Returns

Vector& reference to current instance

Definition at line 407 of file vector.hh.

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

5.4.3.3 operator-()

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

Unary - operator.

Returns

Vector negated Vector instance

Definition at line 417 of file vector.hh.

5.4.3.4 operator*=() [1/2]

```
template<std::floating_point T>
template<Number nType>
Vector& geom::Vector< 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

Vector& reference to vector instance

5.4.3.5 operator/=() [1/2]

```
template<std::floating_point T>
template<Number nType>
Vector& geom::Vector< 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

Vector& reference to vector instance

Warning

Does not check if val equals 0

5.4.3.6 dot()

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

Dot product function.

Parameters

rhs	vector to dot product with

Returns

T dot product of two vectors

Definition at line 445 of file vector.hh.

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

Referenced by geom::operator&().

5.4.3.7 cross()

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

Cross product function.

Parameters

rhs	vector to cross product with

Returns

Vector cross product of two vectors

Definition at line 451 of file vector.hh.

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

Referenced by geom::Plane< T >::getParametric(), and geom::operator%().

5.4.3.8 length2()

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

Calculate squared length of a vector function.

Returns

T length^2

Definition at line 457 of file vector.hh.

5.4.3.9 length()

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

Calculate length of a vector function.

Returns

T length

Definition at line 463 of file vector.hh.

5.4.3.10 normalized()

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

Get normalized vector function.

Returns

Vector normalized vector

Definition at line 469 of file vector.hh.

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

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

5.4.3.11 normalize()

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

Normalize vector function.

Returns

Vector& reference to instance

Definition at line 477 of file vector.hh.

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

5.4.3.12 operator[]() [1/2]

```
template<std::floating_point T>
T & geom::Vector< 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 486 of file vector.hh.

5.4.3.13 operator[]() [2/2]

```
template<std::floating_point T>
T geom::Vector< 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 502 of file vector.hh.

5.4.3.14 isPar()

```
template<std::floating_point T>
bool geom::Vector< T >::isPar (
            const Vector< 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 518 of file vector.hh.

5.4.3.15 isPerp()

```
template<std::floating_point T>
bool geom::Vector< T >::isPerp (
            const Vector< 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 524 of file vector.hh.

5.4.3.16 isEqual()

```
template<std::floating_point T>
bool geom::Vector< T >::isEqual (
            const Vector< 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 530 of file vector.hh.

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

Referenced by geom::operator==().

5.4.3.17 isNumEq()

```
template<std::floating_point T>
bool geom::Vector< 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 536 of file vector.hh.

5.4.3.18 setThreshold()

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

Set new threshold value.

Parameters

in	*thres*	value to set

Definition at line 542 of file vector.hh.

5.4.3.19 getThreshold()

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

Get current threshold value.

Definition at line 548 of file vector.hh.

5.4.3.20 setDefThreshold()

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

Set threshold to default value.

Note

default value equals float point epsilon

Definition at line 554 of file vector.hh.

5.4.3.21 operator*=() [2/2]

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

Definition at line 424 of file vector.hh.

5.4.3.22 operator/=() [2/2]

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

Definition at line 435 of file vector.hh.

5.4.4 Member Data Documentation

5.4.4.1 x

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

Vector coordinates.

Definition at line 46 of file vector.hh.

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

5.4.4.2 y

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

Definition at line 46 of file vector.hh.

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

5.4.4.3 z

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

Definition at line 46 of file vector.hh.

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

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

- include/primitives/vector.hh

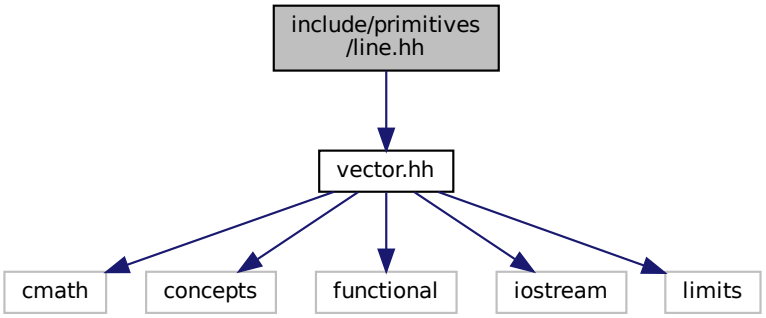
Chapter 6

File Documentation

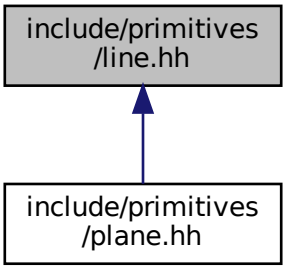
6.1 include/primitives/line.hh File Reference

```
#include "vector.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.

6.2 line.hh

```
00001 #ifndef __INCLUDE_PRIMITIVES_LINE_HH__
00002 #define __INCLUDE_PRIMITIVES_LINE_HH__
00003
00004 #include "vector.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   Vector<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 Vector<T> &org, const Vector<T> &dir);
00037
00038   /**
00039    * @brief Getter for origin vector
00040    *
00041    * @return const Vector<T>& const reference to origin vector
00042    */
00043   const Vector<T> &org() const;
00044
00045   /**
00046    * @brief Getter for direction vector
00047    *
00048    * @return const Vector<T>& const reference to direction vector
```

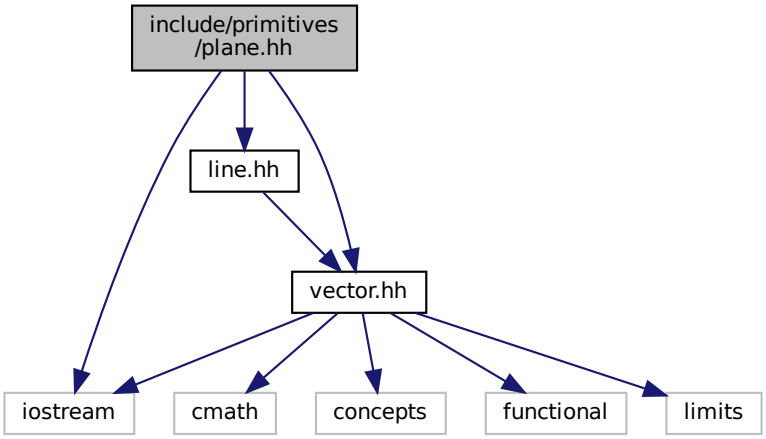
```
00049    */
00050   const Vector<T> &dir() const;
00051
00052   /**
00053    * @brief Checks is point belongs to line
00054    *
00055    * @param[in] point const reference to point vector
00056    * @return true if point belongs to line
00057    * @return false if point doesn't belong to line
00058    */
00059   bool belongs(const Vector<T> &point) const;
00060
00061   /**
00062    * @brief Checks is *this equals to another line
00063    *
00064    * @param[in] line const reference to another line
00065    * @return true if lines are equal
00066    * @return false if lines are not equal
00067    */
00068   bool isEqual(const Line &line) const;
00069
00070   /**
00071    * @brief Get line by 2 points
00072    *
00073    * @param[in] p1 1st point
00074    * @param[in] p2 2nd point
00075    * @return Line passing through two points
00076    */
00077   static Line getBy2Points(const Vector<T> &p1, const Vector<T> &p2);
00078 };
00079
00080 /**
00081  * @brief Line print operator
00082  *
00083  * @tparam T - floating point type of coordinates
00084  * @param[in, out] ost output stream
00085  * @param[in] line Line to print
00086  * @return std::ostream& modified ostream instance
00087  */
00088 template <std::floating_point T>
00089 std::ostream &operator«(std::ostream &ost, const Line<T> &line)
00090 {
00091   ost « line.org() « " + " « line.dir() « " * t";
00092   return ost;
00093 }
00094
00095 /**
00096  * @brief Line equality operator
00097  *
00098  * @tparam T - floating point type of coordinates
00099  * @param[in] lhs 1st line
00100  * @param[in] rhs 2nd line
00101  * @return true if lines are equal
00102  * @return false if lines are not equal
00103  */
00104 template <std::floating_point T>
00105 bool operator==(const Line<T> &lhs, const Line<T> &rhs)
00106 {
00107   return lhs.isEqual(rhs);
00108 }
00109
00110 template <std::floating_point T>
00111 Line<T>::Line(const Vector<T> &org, const Vector<T> &dir) : org_{org}, dir_{dir}
00112 {
00113   if (dir_ == Vector<T>{0})
00114     throw std::logic_error{"Direction vector equals zero."};
00115 }
00116
00117 template <std::floating_point T>
00118 const Vector<T> &Line<T>::org() const
00119 {
00120   return org_;
00121 }
00122
00123 template <std::floating_point T>
00124 const Vector<T> &Line<T>::dir() const
00125 {
00126   return dir_;
00127 }
00128
00129 template <std::floating_point T>
00130 bool Line<T>::belongs(const Vector<T> &point) const
00131 {
00132   return dir_.cross(point - org_) == Vector<T>{0};
00133 }
00134
00135 template <std::floating_point T>
```

```
00136 bool Line<T>::isEqual(const Line<T> &line) const
00137 {
00138   return belongs(line.org_) && dir_.isPar(line.dir_);
00139 }
00140
00141 template <std::floating_point T>
00142 Line<T> Line<T>::getBy2Points(const Vector<T> &p1, const Vector<T> &p2)
00143 {
00144   return Line<T>{p1, p2 - p1};
00145 }
00146
00147 } // namespace geom
00148
00149 #endif // __INCLUDE_PRIMITIVES_LINE_HH__
```

6.3 include/primitives/plane.hh File Reference

```
#include <iostream>
#include "line.hh"
#include "vector.hh"
```

Include dependency graph for plane.hh:



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.

6.4 plane.hh

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

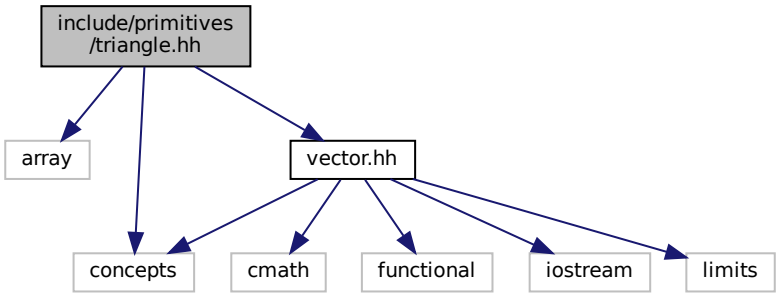
```
00069   /**
00070    * @brief Checks if line belongs to plane
00071    *
00072    * @param[in] line const referene to line
00073    * @return true if line belongs to plane
00074    * @return false if line doesn't belong to plane
00075    */
00076   bool belongs(const Line<T> &line) const;
00077
00078   /**
00079    * @brief Checks is *this equals to another plane
00080    *
00081    * @param[in] rhs const reference to another plane
00082    * @return true if planes are equal
00083    * @return false if planes are not equal
00084    */
00085   bool isEqual(const Plane &rhs) const;
00086
00087   /**
00088    * @brief Get plane by 3 points
00089    *
00090    * @param[in] pt1 1st point
00091    * @param[in] pt2 2nd point
00092    * @param[in] pt3 3rd point
00093    * @return Plane passing through three points
00094    */
00095   static Plane getBy3Points(const Vector<T> &pt1, const Vector<T> &pt2,
00096                             const Vector<T> &pt3);
00097
00098   /**
00099    * @brief Get plane from parametric plane equation
00100    *
00101    * @param[in] org origin vector
00102    * @param[in] dir1 1st direction vector
00103    * @param[in] dir2 2nd direction vector
00104    * @return Plane
00105    */
00106   static Plane getParametric(const Vector<T> &org, const Vector<T> &dir1,
00107                              const Vector<T> &dir2);
00108
00109   /**
00110    * @brief Get plane from normal point plane equation
00111    *
00112    * @param[in] norm normal vector
00113    * @param[in] point point lying on the plane
00114    * @return Plane
00115    */
00116   static Plane getNormalPoint(const Vector<T> &norm, const Vector<T> &point);
00117
00118   /**
00119    * @brief Get plane form normal const plane equation
00120    *
00121    * @param[in] norm normal vector
00122    * @param[in] constant distance
00123    * @return Plane
00124    */
00125   static Plane getNormalDist(const Vector<T> &norm, T constant);
00126 };
00127
00128 /**
00129  * @brief Plane equality operator
00130  *
00131  * @tparam T - floating point type of coordinates
00132  * @param[in] lhs 1st plane
00133  * @param[in] rhs 2nd plane
00134  * @return true if planes are equal
00135  * @return false if planes are not equal
00136  */
00137 template <std::floating_point T>
00138 bool operator==(const Plane<T> &lhs, const Plane<T> &rhs)
00139 {
00140   return lhs.isEqual(rhs);
00141 }
00142
00143 /**
00144  * @brief Plane print operator
00145  *
00146  * @tparam T - floating point type of coordinates
00147  * @param[in, out] ost output stream
00148  * @param[in] pl plane to print
00149  * @return std::ostream& modified ostream instance
00150  */
00151 template <std::floating_point T>
00152 std::ostream &operator«(std::ostream &ost, const Plane<T> &pl)
00153 {
00154   ost « pl.norm() « " * X = " « pl.dist();
00155   return ost;
```

```
00156 }
00157
00158 template <std::floating_point T>
00159 Plane<T>::Plane(const Vector<T> &norm, T dist) : norm_(norm), dist_(dist)
00160 {
00161   if (norm == Vector<T>{0})
00162     throw std::logic_error{"normal vector equals to zero"};
00163 }
00164
00165 template <std::floating_point T>
00166 T Plane<T>::dist() const
00167 {
00168   return dist_;
00169 }
00170
00171 template <std::floating_point T>
00172 const Vector<T> &Plane<T>::norm() const
00173 {
00174   return norm_;
00175 }
00176
00177 template <std::floating_point T>
00178 bool Plane<T>::belongs(const Vector<T> &pt) const
00179 {
00180   return Vector<T>::isNumEq(norm_.dot(pt), dist_);
00181 }
00182
00183 template <std::floating_point T>
00184 bool Plane<T>::belongs(const Line<T> &line) const
00185 {
00186   return norm_.isPerp(line.dir()) && belongs(line.org());
00187 }
00188
00189 template <std::floating_point T>
00190 bool Plane<T>::isEqual(const Plane &rhs) const
00191 {
00192   return (norm_ * dist_ == rhs.norm_ * rhs.dist_) && (norm_.isPar(rhs.norm_));
00193 }
00194
00195 template <std::floating_point T>
00196 Plane<T> Plane<T>::getBy3Points(const Vector<T> &pt1, const Vector<T> &pt2,
00197                                 const Vector<T> &pt3)
00198 {
00199   return getParametric(pt1, pt2 - pt1, pt3 - pt1);
00200 }
00201
00202 template <std::floating_point T>
00203 Plane<T> Plane<T>::getParametric(const Vector<T> &org, const Vector<T> &dir1,
00204                                  const Vector<T> &dir2)
00205 {
00206   auto norm = dir1.cross(dir2);
00207   return getNormalPoint(norm, org);
00208 }
00209
00210 template <std::floating_point T>
00211 Plane<T> Plane<T>::getNormalPoint(const Vector<T> &norm, const Vector<T> &pt)
00212 {
00213   auto normalized = norm.normalized();
00214   return Plane{normalized, normalized.dot(pt)};
00215 }
00216
00217 template <std::floating_point T>
00218 Plane<T> Plane<T>::getNormalDist(const Vector<T> &norm, T dist)
00219 {
00220   auto normalized = norm.normalized();
00221   return Plane{normalized, dist};
00222 }
00223
00224 } // namespace geom
00225
00226 #endif // __INCLUDE_PRIMITIVES_PLANE_HH__
```

6.5 include/primitives/triangle.hh File Reference

```
#include <array>
#include <concepts>
#include "vector.hh"
```

Include dependency graph for triangle.hh:



Classes

- class geom::Triangle< T >

  Triangle class implementation.

Namespaces

- geom

  line.hh Line class implementation

Functions

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

  Triangle print operator.

6.6 triangle.hh

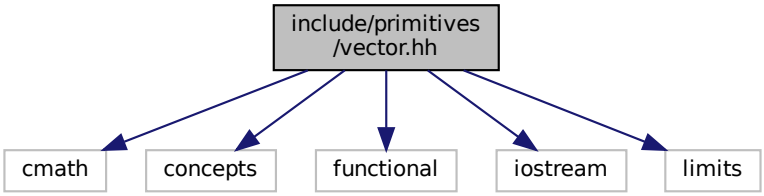
```
00001 #ifndef __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00002 #define __INCLUDE_PRIMITIVES_TRIANGLE_HH__
00003
00004 #include <array>
00005 #include <concepts>
00006
00007 #include "vector.hh"
00008
00009 /**
00010  * @brief triangle.hh
00011  * Triangle class implementation
00012  */
00013
00014 namespace geom
00015 {
00016
00017 /**
00018  * @class Triangle
00019  * @brief Triangle class implementation
00020  *
00021  * @tparam T - floating point type of coordinates
00022  */
00023 template <std::floating_point T>
00024 class Triangle final
```

```
00025 {
00026 private:
00027   /**
00028    * @brief Vertices of triangle
00029    */
00030   std::array<Vector<T>, 3> vertices_;
00031
00032 public:
00033   /**
00034    * @brief Construct a new Triangle object from 3 points
00035    *
00036    * @param[in] p1 1st point
00037    * @param[in] p2 2nd point
00038    * @param[in] p3 3rd point
00039    */
00040   Triangle(const Vector<T> &p1, const Vector<T> &p2, const Vector<T> &p3);
00041
00042   /**
00043    * @brief Overloaded operator[] to get access to vertices
00044    *
00045    * @param[in] idx index of vertex
00046    * @return const Vector<T>& const reference to vertex
00047    */
00048   const Vector<T> &operator[](std::size_t idx) const;
00049 };
00050
00051 /**
00052  * @brief Triangle print operator
00053  *
00054  * @tparam T - floating point type of coordinates
00055  * @param[in, out] ost output stream
00056  * @param[in] tr Triangle to print
00057  * @return std::ostream& modified ostream instance
00058  */
00059 template <std::floating_point T>
00060 std::ostream &operator«(std::ostream &ost, const Triangle<T> &tr)
00061 {
00062   ost « "Triangle: {";
00063   for (size_t i : {0, 1, 2})
00064     ost « tr[i] « (i == 2 ? "" : ", ");
00065
00066   ost « "}";
00067
00068   return ost;
00069 }
00070
00071 template <std::floating_point T>
00072 Triangle<T>::Triangle(const Vector<T> &p1, const Vector<T> &p2, const Vector<T> &p3)
00073   : vertices_{p1, p2, p3}
00074 {
00075 }
00076
00077 template <std::floating_point T>
00078 const Vector<T> &Triangle<T>::operator[](std::size_t idx) const
00079 {
00080   return vertices_[idx % 3];
00081 }
00082
00083 } // namespace geom
00084
00085 #endif // __INCLUDE_PRIMITIVES_TRIANGLE_HH__
```

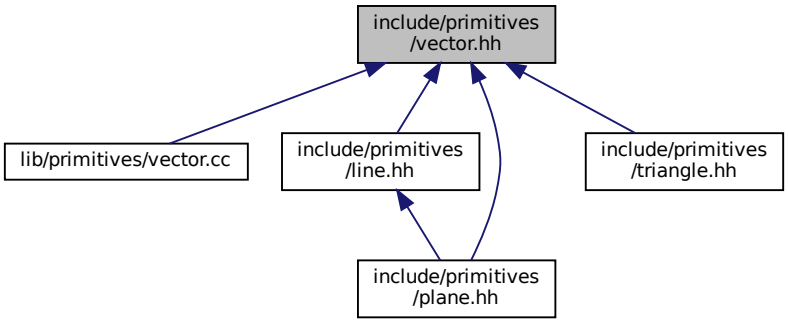
6.7 include/primitives/vector.hh File Reference

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

Include dependency graph for vector.hh:



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



Classes

- class geom::Vector< T >

  Vector class realization.

Namespaces

- geom

  line.hh Line class implementation

Typedefs

- using geom::VectorD = Vector< double >
- using geom::VectorF = Vector< float >

Functions

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

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

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

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

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

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

  Dot product operator.
- template<std::floating_point T>
  Vector< T > geom::operator% (const Vector< T > &lhs, const Vector< T > &rhs)

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

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

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

  Vector print operator.

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.

6.7.1 Detailed Description

Vector class implementation

Definition in file vector.hh.

6.8 vector.hh

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

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

```
00173     * @note Coordinates calculated by mod 3
00174     */
00175    T operator[](size_t i) const;
00176
00177    /**
00178     * @brief Check if vector is parallel to another
00179     *
00180     * @param[in] rhs vector to check parallelism with
00181     * @return true if vector is parallel
00182     * @return false otherwise
00183     */
00184    bool isPar(const Vector &rhs) const;
00185
00186    /**
00187     * @brief Check if vector is perpendicular to another
00188     *
00189     * @param[in] rhs vector to check perpendicularity with
00190     * @return true if vector is perpendicular
00191     * @return false otherwise
00192     */
00193    bool isPerp(const Vector &rhs) const;
00194
00195    /**
00196     * @brief Check if vector is equal to another
00197     *
00198     * @param[in] rhs vector to check equality with
00199     * @return true if vector is equal
00200     * @return false otherwise
00201     *
00202     * @note Equality check performs using isNumEq(T lhs, T rhs) function
00203     */
00204    bool isEqual(const Vector &rhs) const;
00205
00206    /**
00207     * @brief Check equality (with threshold) of two floating point numbers function
00208     *
00209     * @param[in] lhs first number
00210     * @param[in] rhs second number
00211     * @return true if numbers equals with threshold (|lhs - rhs| < threshold)
00212     * @return false otherwise
00213     *
00214     * @note Threshold defined by threshold_ static member
00215     */
00216    static bool isNumEq(T lhs, T rhs);
00217
00218    /**
00219     * @brief Set new threshold value
00220     *
00221     * @param[in] thres value to set
00222     */
00223    static void setThreshold(T thres);
00224
00225    /**
00226     * @brief Get current threshold value
00227     */
00228    static void getThreshold();
00229
00230    /**
00231     * @brief Set threshold to default value
00232     * @note default value equals float point epsilon
00233     */
00234    static void setDefThreshold();
00235 };
00236
00237 /**
00238  * @brief Overloaded + operator
00239  *
00240  * @tparam T vector template parameter
00241  * @param[in] lhs first vector
00242  * @param[in] rhs second vector
00243  * @return Vector<T> sum of two vectors
00244  */
00245 template <std::floating_point T>
00246 Vector<T> operator+(const Vector<T> &lhs, const Vector<T> &rhs)
00247 {
00248    Vector<T> res{lhs};
00249    res += rhs;
00250    return res;
00251 }
00252
00253 /**
00254  * @brief Overloaded - operator
00255  *
00256  * @tparam T vector template parameter
00257  * @param[in] lhs first vector
00258  * @param[in] rhs second vector
00259  * @return Vector<T> res of two vectors
```

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

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

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

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

6.9 lib/primitives/line.cc File Reference

6.10 line.cc

6.11 lib/primitives/plane.cc File Reference

6.12 plane.cc

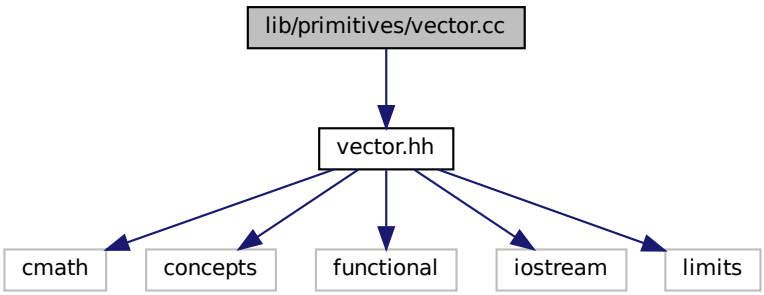
6.13 lib/primitives/triangle.cc File Reference

6.14 triangle.cc

6.15 lib/primitives/vector.cc File Reference

```
#include "vector.hh"
```

Include dependency graph for vector.cc:



Namespaces

- geom

  line.hh Line class implementation

6.16 vector.cc

```
00001 #include "vector.hh"
00002
00003 namespace geom
00004 {
00005
00006 // template <std::floating_point T>
00007 // T Vector<T>::threshold_ = std::numeric_limits<T>::epsilon();
00008
00009 } // namespace geom
```