# **Geometry**

A CPP Template Library

```
O. Author
1. Overview
2. Code Structure
3. Classes Belongs
4. Figures
5. Using example
   Point2<Ty>
    Point3<Ty>
    PointSet23<Ty>
    Point_cast<Ty>
    Triangle2<Ty>
    Triangle3<Ty>
    Line2<Ty>
    Line3<Ty>
    Rectangle<Ty>
    Polygon<Ty>
    LineString23<Ty>
    RefPoint23<Ty>
    RefPointSet23<Ty>
    RefLine23<Ty>
    RefRectangle<Ty>
    RefTriangle23<Ty>
```

Geometry

## O. Author

*Name* csl

Final

**E-Mmail** 3079625093@qq.com

RefPolygon<*Ty>*RefLinestring23<*Ty>* 

utility

RefPointSet\_WriteRead23<Ty>

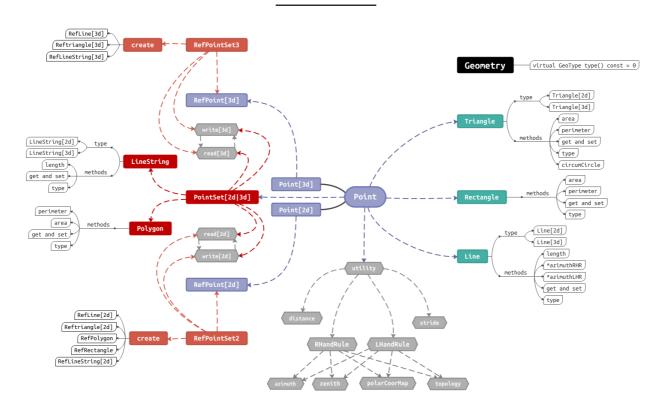
```
1
        _|_|
                     _|_|
                                     _|_|_|
               _|_|
                        _| _| _|_|_|
3
  _| _| _|_|_| _|
                  _l _l
                                     _|
                                           _|_|
              _1
4
  _l _l _l
                        _|
                            _l _l
                                      _|
                                           _|
5
         _|_|_|
   _|_|_|
                            _l
                                _|_|_|
                                                   _|_|_|
6
     _|
   _|_|
                                                   _|_|
```

## 1. Overview

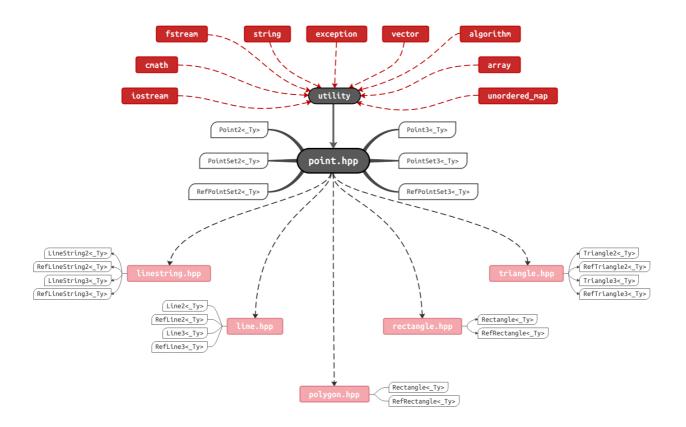
This CPP library mainly provides two dimension point template classes: Point2<Ty> and Point3<Ty> .It also provides related geometries and operations based on two kinds of points, such as conventional "write" operation, "read" operation and distance calculation of point set, and azimuth calculation based on point2. You can easily use it to assist development. And because it's a template class, you can just copy the head file to your project and use it.

There are some details of this library below. And if you find some bugs or have some bright ideas for this library, please contact me through the E-Mail above.

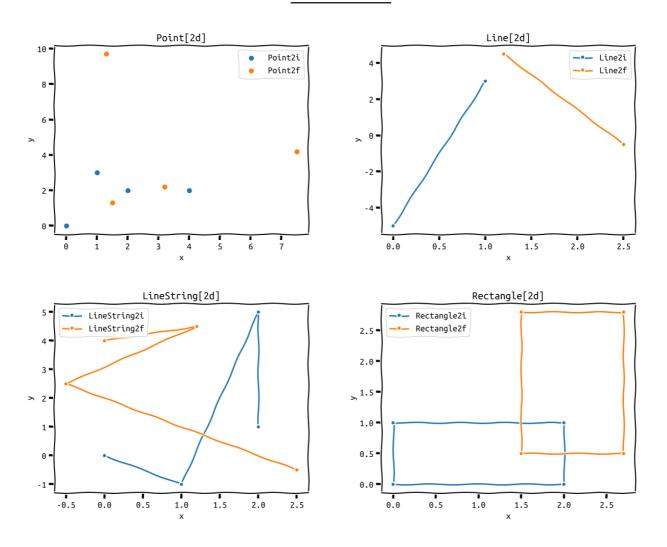
### 2. Code Structure

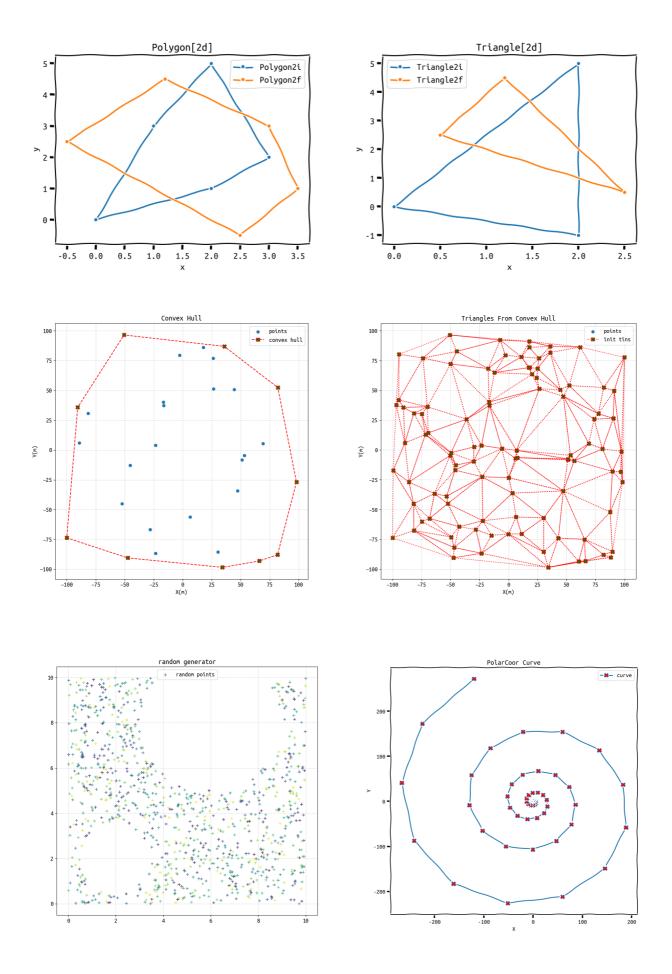


## 3. Classes Belongs



## 4. Figures





5. Using example

#### Point2<Ty>

```
void foo_point2()
 1
 2
     {
 3
         PointSet2f ps;
 4
         ps.push_back({0.6, 0.4});
 5
         ps.push_back({1.9, 2.7});
         ps.push_back({0.6, 0.4});
 7
         ps.push_back({1.9, 2.7});
         try
 9
         {
10
             // distance between tow points
             std::cout << distance(ps.front(), ps.back()) << std::endl;</pre>
11
12
             // write and read point data
13
             // way one.
14
             // default write mode : std::ios::out | std::ios::binary
15
             ps.write("../output/point2.bin");
16
             ps.clear();
17
             // default read mode : std::ios::in | std::ios::binary
             ps.read("../output/point2.bin");
18
19
             // way two.
20
             // write mode : std::ios::out
             ps.write("../output/point2.txt", std::ios::out);
21
22
             ps.clear();
23
             // read mode : std::ios::in
24
             ps.read("../output/point2.txt", std::ios::in);
25
             // print points
26
             for (const auto &elem : ps)
27
28
                 std::cout << elem << std::endl;</pre>
29
             }
30
         catch (const std::exception &e)
31
32
         {
             std::cerr << e.what() << '\n';</pre>
33
         }
         return;
35
36
    }
37
     /** output
     * 2.64197
38
39
     * [0.6, 0.4]
     * [1.9, 2.7]
40
      * [0.6, 0.4]
41
     * [1.9, 2.7]
42
     */
43
```

## Point3<Ty>

```
void foo_point3()

PointSet3f ps;
ps.push_back({0.6, 0.4, 1.1});
ps.push_back({1.9, 2.7, 2.3});
ps.push_back({0.6, 0.4, 1.1});
ps.push_back({0.6, 0.4, 1.1});
ps.push_back({1.9, 2.7, 2.3});
try
```

```
9
10
             // distance between tow points
             std::cout << distance(ps.front(), ps.back()) << std::endl;</pre>
11
12
             // write and read point data
13
             // way one.
             // default write mode : std::ios::out | std::ios::binary
14
15
             ps.write("../output/point3.bin");
             ps.clear();
16
17
             // default read mode : std::ios::in | std::ios::binary
             ps.read("../output/point3.bin");
18
19
20
             // way two.
             // write mode : std::ios::out
21
             ps.write("../output/point3.txt", std::ios::out);
22
23
             ps.clear();
24
             // read mode : std::ios::in
             ps.read("../output/point3.txt", std::ios::in);
25
26
             // print points
27
             for (const auto &elem : ps)
28
             {
                 std::cout << elem << std::endl;</pre>
29
30
31
         }
         catch (const std::exception &e)
32
33
34
             std::cerr << e.what() << '\n';</pre>
35
36
         return;
37
    /** output
38
39
     * 2.90172
40
     * [0.6, 0.4, 1.1]
41
      * [1.9, 2.7, 2.3]
     * [0.6, 0.4, 3.5]
42
43
     * [1.9, 2.7, 4.6]
44
```

### PointSet23<Ty>

```
void foo_pointset23()
 1
 2
 3
         PointSet2f ps;
         ps.push_back(Point2f(1, 2));
 4
         ps.push_back(Point2f(2, 3));
         ps.write("../output/pointset.csv", std::ios::out);
 6
 7
         ps.clear();
 8
         ps.read("../output/pointset.csv", std::ios::in);
 9
         for (const auto &point : ps)
10
             std::cout << point << std::endl;</pre>
         return;
11
12
13
    /** output
14
     * [1, 2]
15
     * [2, 3]
     */
16
```

#### Point\_cast<Ty>

```
void foo_ponitCast_test()
 1
 2
    {
 3
         Point3f p(1, 2, 6);
 4
         Point2f p2(2, 6);
 5
         auto ary = static_cast<Point3f::ary_type>(p);
         auto ary2 = static_cast<Point2f::ary_type>(p2);
         std::cout << ary[0] << ',' << ary[1] << ',' << ary[2] << std::endl;
 9
         std::cout << ary2[0] << ',' << ary2[1] << std::endl;
10
         std::cout << Point3f(ary) << std::endl;</pre>
11
12
         std::cout << Point2f(ary2) << std::endl;</pre>
13
14
         return;
15
    /** output
16
17
     * 1,2,6
     * 2,6
18
     * [1, 2, 6]
     * [2, 6]
20
     */
21
```

#### Triangle2<Ty>

```
void foo_triangle2()
 1
 2
     {
         ns_geo::Point2<double> points[3] = {
 3
 4
             Point2d(0, 0),
 5
             Point2d(2, 2),
             Point2d(2, 0)};
 7
         ns_geo::Triangle2d tri(points);
         std::cout << tri << std::endl;</pre>
         std::cout << "area : " << tri.area() << std::endl;</pre>
 9
         std::cout << "perimeter : " << tri.perimeter() << std::endl;</pre>
10
11
         return;
12
    }
13
    /** output
     * {[0, 0], [2, 2], [2, 0]}
14
15
      * area : 2
      * perimeter : 6.82843
16
17
     */
```

## Triangle3<Ty>

```
void foo_triangle3()

ns_geo::Point3<double> points[3] = {
    Point3d(0, 0, 0),
    Point3d(2, 2, 2),
    Point3d(2, 0, 0)};
```

```
7
         ns_geo::Triangle3d tri(points);
         std::cout << tri << std::endl;</pre>
 9
         std::cout << "area : " << tri.area() << std::endl;</pre>
         std::cout << "perimeter : " << tri.perimeter() << std::endl;</pre>
10
11
         return;
    }
12
13
    /** output
14
     * {[0, 0, 0], [2, 2, 2], [2, 0, 0]}
     * area : 2.82843
     * perimeter : 8.29253
16
```

#### Line2<Ty>

```
1
    void foo_line2()
 2
 3
         ns_geo::Line2d line(Point2d(0, 0), Point2d(2, 2));
         std::cout << line << std::endl;</pre>
 4
         std::cout << "length : " << line.length() << std::endl;</pre>
 5
         for (const auto &elem : line.points())
 7
             std::cout << elem << std::endl;</pre>
         return;
 9
    }
10
    /** output
     * {[0, 0], [2, 2]}
11
     * length : 2.82843
12
13
     * [0, 0]
14
      * [2, 2]
15
     */
```

## Line3<Ty>

```
1
    void foo_line3()
 2
 3
         ns_geo::Line3d line(Point3d(0, 0, 0), Point3d(2, 2, 2));
         std::cout << line << std::endl;</pre>
 5
         std::cout << "length : " << line.length() << std::endl;</pre>
 6
         for (const auto &elem : line.points())
 7
             std::cout << elem << std::endl;</pre>
 8
         return;
 9
    }
    /** output
10
     * {[0, 0, 0], [2, 2, 2]}
11
      * length : 3.4641
12
13
     * [0, 0, 0]
14
     * [2, 2, 2]
15
```

#### Rectangle<Ty>

```
void foo_rectangle()
 1
 2
     {
 3
         ns_geo::Rectangled rect(0, 4, 1, 0);
 4
         std::cout << rect << std::endl;</pre>
 5
         std::cout << "area : " << rect.area() << std::endl;</pre>
         std::cout << "peri : " << rect.perimeter() << std::endl;</pre>
 7
         for (const auto &elem : rect.points())
             std::cout << elem << std::endl;</pre>
 9
         return;
10
    1
    /** output
11
12
      * {[0, 4], [1, 0]}
     * area : 4
13
      * peri : 10
14
      * [0, 4]
15
16
     * [1, 0]
17
     */
```

#### Polygon<Ty>

```
1
     void foo_polygon()
 2
 3
         Polygond polygon({Point2d(0, 0),
                            Point2d(0, 1),
 4
                            Point2d(0.5, 2),
 6
                            Point2d(1, 1),
 7
                            Point2d(1, 0)});
 8
         std::cout << polygon << std::endl;</pre>
 9
         std::cout << "perimeter : " << polygon.perimeter() << std::endl;</pre>
10
         std::cout << "area : " << polygon.area() << std::endl;</pre>
11
         return;
12
13
    /** output
14
       * {[0, 0], [0, 1], [0.5, 2], [1, 1], [1, 0]}
15
      * perimeter : 5.23607
16
       * area : 1.5
     */
17
```

### LineString23<Ty>

```
void foo_lineString23()
 1
 2
 3
         LineString3d ls({Point3d(0, 0, 9),
 4
                           Point3d(0, 1, 9),
 5
                           Point3d(1, 1, 9),
 6
                           Point3d(1, 0, 9)});
 7
         std::cout << ls << std::endl;</pre>
 8
         std::cout << ls.length() << std::endl;</pre>
 9
         LineString2d ls2({Point2d(0, 9),
                            Point2d(1, 9),
10
```

```
11
                            Point2d(1, 9),
12
                            Point2d(0, 9)});
         std::cout << ls2 << std::endl;</pre>
13
         std::cout << ls2.length() << std::endl;</pre>
14
15
         return;
16
    }
17
     /** output
     * {[0, 0, 9], [0, 1, 9], [1, 1, 9], [1, 0, 9]}
18
      * {[0, 9], [1, 9], [1, 9], [0, 9]}
20
21
22
     */
```

#### RefPoint23<Ty>

```
void foo_refpoint23()
 1
 2
     {
 3
         double ary1[3] = \{1, 2, 3\};
 4
         RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
         RefPoint3d p2(1, ary1);
         std::cout << distance(p1, p2) << std::endl;</pre>
 6
 7
         std::cout << p1 << std::endl;</pre>
 8
 9
         double ary2[2] = \{2, 3\};
10
         RefPoint2d p3(0, RefPoint2d::ary_type{0, 0});
11
         RefPoint2d p4(1, ary2);
         std::cout << distance(p3, p4) << std::endl;</pre>
12
         std::cout << p3 << std::endl;</pre>
13
14
    /** output
15
16
     * 3.74166
     * {0: [0, 0, 0]}
17
      * 3.60555
     * {0: [0, 0]}
19
20
```

#### RefPointSet23<Ty>

```
1
     void foo_refpointset23()
 2
 3
         double ary2[2] = \{2, 3\};
 4
         RefPointSet2d rps2;
 5
         rps2.insert({0, RefPoint2d::ary_type{0, 0}});
 6
         rps2.insert({1, ary2});
 7
         rps2.insert({2, RefPoint2d::ary_type{0, 0}});
 8
         rps2.insert({4, ary2});
 9
         for (const auto &refp : rps2)
10
             std::cout << refp.second << std::endl;</pre>
11
         std::cout << rps2.size() << std::endl;</pre>
12
13
         RefPointSet3d rps3;
         rps3.insert({0, RefPoint3d::ary_type{0, 0, 0}});
14
15
         rps3.insert({1, RefPoint3d::ary_type{0, 1, 0}});
```

```
16
         rps3.insert({2, RefPoint3d::ary_type{0, 0, 1}});
         rps3.insert({3, RefPoint3d::ary_type{1, 0, 0}});
17
18
         for (const auto &refp : rps3)
19
             std::cout << refp.second << std::endl;</pre>
         std::cout << rps3.size() << std::endl;</pre>
20
21
    }
22
     /** output
     * {0: [0, 0]}
23
     * {2: [0, 0]}
24
      * {4: [2, 3]}
25
      * {1: [2, 3]}
26
     * 4
27
     * {4: [1, 0, 0]}
28
29
     * {2: [0, 0, 1]}
      * {1: [0, 1, 0]}
30
31
     * {0: [0, 0, 0]}
32
      * 4
33
```

#### RefLine23<Ty>

```
void foo_refline2()
 2
     {
         double ary2[2] = \{2, 3\};
         RefPointSet2d rps;
 4
 5
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
 6
         rps.insert({1, ary2});
 7
         rps.insert({2, RefPoint2d::ary_type{0, 0}});
 8
         rps.insert({4, ary2});
 9
         for (const auto &refp : rps)
10
             std::cout << refp.second << std::endl;</pre>
11
12
         auto refline = rps.createRefLine2(0, 1);
         std::cout << refline << std::endl;</pre>
13
         std::cout << refline.length() << std::endl;</pre>
15
     /** output
16
17
     * {0: [0, 0]}
18
      * {2: [0, 0]}
19
      * {4: [2, 3]}
      * {1: [2, 3]}
20
      * {0: [0, 0], 1: [2, 3]}
21
      * 3.60555
22
23
     */
24
25
     void foo_refline3()
26
27
         RefPointSet3d rps;
28
         rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
29
         rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
30
         rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
         rps.insert(\{3,\ RefPoint3d::ary\_type\{1,\ 0,\ 0\}\});
31
32
         for (const auto &refp : ps)
33
             std::cout << refp.second << std::endl;</pre>
34
35
         auto refline = rps.createRefLine3(0, 1);
         std::cout << refline << std::endl;</pre>
36
```

```
37
         std::cout << refline.length() << std::endl;</pre>
38
         auto ary = refline.points();
39
    /** output
40
     * {0: [0, 0, 0]}
41
     * {2: [0, 0, 1]}
42
      * {4: [1, 0, 0]}
     * {1: [0, 1, 0]}
44
     * {0: [0, 0, 0], 1: [0, 1, 0]}
     * 1
46
47
     */
```

#### RefRectangle<Ty>

```
1
     void foo_refrectangle()
 2
 3
         double ary2[2] = \{2, 3\};
 4
         RefPointSet2d rps;
 5
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
         rps.insert({1, ary2});
 7
         rps.insert({2, RefPoint2d::ary_type{0, 0}});
 8
         rps.insert({4, ary2});
         for (const auto &refp : rps)
 9
10
             std::cout << refp.second << std::endl;</pre>
11
12
         auto rect = rps.createRefRectangle(0, 1);
13
         std::cout << rect << std::endl;</pre>
14
         std::cout << rect.area() << std::endl;</pre>
15
         std::cout << rect.perimeter() << std::endl;</pre>
16
    }
17
     /** output
     * {0: [0, 0]}
18
19
      * {2: [0, 0]}
      * {4: [2, 3]}
20
      * {1: [2, 3]}
21
22
     * {0: [0, 0], 1: [2, 3]}
     * 6
23
      * 10
24
     */
25
```

### RefTriangle23<Ty>

```
void foo_reftriangle2()
 2
    {
 3
         double ary2[2] = \{2, 3\};
 4
         RefPointSet2d rps;
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
 6
         rps.insert({1, ary2});
 7
         rps.insert({2, RefPoint2d::ary_type{0, 0}});
 8
         rps.insert({4, ary2});
 9
         for (const auto &refp : rps)
10
             std::cout << refp.second << std::endl;</pre>
11
         auto tri = rps.createRefTriangle2(0, 1, 2);
```

```
12
13
         std::cout << tri << std::endl;</pre>
         std::cout << tri.perimeter() << std::endl;</pre>
14
15
         std::cout << tri.area() << std::endl;</pre>
16
    /** output
17
18
     * {0: [0, 0]}
      * {2: [0, 2]}
19
      * {4: [3, 0]}
20
21
      * {1: [1, 0]}
      * {0: [0, 0], 1: [1, 0], 2: [0, 2]}
22
      * 5.23607
23
     * 1
24
     */
25
26
27
     void foo_reftriangle3()
28
29
         RefPointSet3d rps;
30
         rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
31
         rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
         rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
32
         rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
         for (const auto &refp : rps)
34
             std::cout << refp.second << std::endl;</pre>
35
36
37
         auto tri = rps.createRefTriangle3(0, 1, 2);
38
         std::cout << tri << std::endl;</pre>
39
         std::cout << tri.area() << std::endl;</pre>
40
         std::cout << tri.perimeter() << std::endl;</pre>
41
    }
42
    /** output
43
     * {0: [0, 0, 0]}
      * {2: [0, 0, 1]}
45
     * {4: [1, 0, 0]}
46
      * {1: [0, 1, 0]}
      * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1]}
47
48
      * 0.5
     * 3.41421
49
50
      */
```

#### RefPolygon<Ty>

```
void foo_refpolygon()
 1
 2
    {
 3
         RefPointSet2d rps;
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
 4
 5
         rps.insert({1, RefPoint2d::ary_type{1, 0}});
 6
         rps.insert({2, RefPoint2d::ary_type{1, 1}});
 7
         rps.insert({4, RefPoint2d::ary_type{0, 1}});
 8
 9
         auto polygon = rps.createRefPolygon({0, 1, 2, 4});
10
         std::cout << polygon << std::endl;</pre>
11
         std::cout << "perimeter : " << polygon.perimeter() << std::endl;</pre>
12
    }
    /** output
13
     * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
14
      * perimeter : 4
15
```

#### RefLinestring23<Ty>

```
void foo_reflinestring2()
 2
     {
 3
         RefPointSet2d rps;
 4
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
         rps.insert({1, RefPoint2d::ary_type{1, 0}});
         rps.insert({2, RefPoint2d::ary_type{1, 1}});
 6
         rps.insert({4, RefPoint2d::ary_type{0, 1}});
 8
         auto ls = rps.createRefLineString2({0, 1, 2, 4});
         std::cout << ls << std::endl;</pre>
10
         std::cout << "length : " << ls.length() << std::endl;</pre>
11
12
     /** output
13
14
      * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
15
      * length: 3
16
17
     void foo_reflinestring3()
18
19
20
         RefPointSet3d rps;
         rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
21
22
         rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
         rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
23
24
         rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
25
         auto ls = rps.createRefLineString3({0, 1, 2, 4});
26
27
         std::cout << ls << std::endl;</pre>
         std::cout << "length : " << ls.length() << std::endl;</pre>
28
29
30
      /** output
       * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1], 4: [1, 0, 0]}
32
       * length : 3.82843
33
       */
```

## RefPointSet WriteRead23<Ty>

```
void foo_refpointset2_write()
1
2
3
        RefPointSet2d rps;
4
        rps.insert({0, RefPoint2d::ary_type{0, 0}});
5
         rps.insert({1, RefPoint2d::ary_type{1, 0}});
        rps.insert({2, RefPoint2d::ary_type{1, 1}});
6
         rps.insert({4, RefPoint2d::ary_type{0, 1}});
8
9
        rps.write("../output/refpointset2.bin");
10
        rps.clear();
        rps.read("../output/refpointset2.bin");
11
        for (const auto &[id, refp] : rps)
12
13
             std::cout << refp << std::endl;</pre>
```

```
14
    }
15
     /** output
     * {1: [1, 0]}
16
17
      * {4: [0, 1]}
      * {2: [1, 1]}
18
      * {0: [0, 0]}
19
20
    void foo_refpointset3_write()
21
22
    {
         RefPointSet3d rps;
23
         rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
24
25
         rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
26
         rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
         rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
27
28
29
         rps.write("../output/refpointset3.bin");
30
         rps.clear();
31
         rps.read("../output/refpointset3.bin");
32
         for (const auto &[id, refp] : rps)
33
             std::cout << refp << std::endl;</pre>
34
    }
35
    /** output
     * {1: [0, 1, 0]}
36
     * {4: [1, 0, 0]}
37
      * {2: [0, 0, 1]}
38
39
      * {0: [0, 0, 0]}
40
     */
```

#### utility

```
1
    void foo_polarCoor()
 2
 3
         Point3f p1(0.0, 0.0, 0.0);
         Point3f p2(10.0, 40.0, -2.0);
 4
         std::cout << LHandRule::polarCoorMethod(p1, distance(p1, p2),</pre>
 6
                                                   LHandRule::azimuth(p1, p2),
 7
                                                   LHandRule::zenith(p1, p2))
 8
                   << std::endl;
 9
         return;
10
     /** output
11
12
     * [10, 40, -2]
13
14
    void foo_distance()
15
16
         Point2d p1(1, 1);
17
18
         Point2d p2(2, 2);
19
         Line2d line({0, 0, 0, 1});
20
         std::cout << "p1 -> p2 : " << distance(p1, p2) << std::endl;
21
         std::cout << "p1 -> line : " << distance(p1, line) << std::endl;</pre>
22
         double ary2[2] = {2, 3};
23
         RefPointSet2d rps;
         rps.insert({0, RefPoint2d::ary_type{0, 0}});
24
25
         rps.insert({1, ary2});
26
         rps.insert({2, RefPoint2d::ary_type{0, 0}});
27
         rps.insert({4, ary2});
```

```
auto refline = rps.createRefLine2(0, 1);
std::cout << "p1 -> refline : " << distance(p1, Line2d(refline)) << std::endl;
return;
}

/** output

* p1 -> p2 : 1.41421

* p1 -> line : 1

* p1 -> refline : 0.27735

*/
```

## Final

For other implementation details, please refer to the source code.