

Geometry

A CPP Template Class

0. Author

name csl

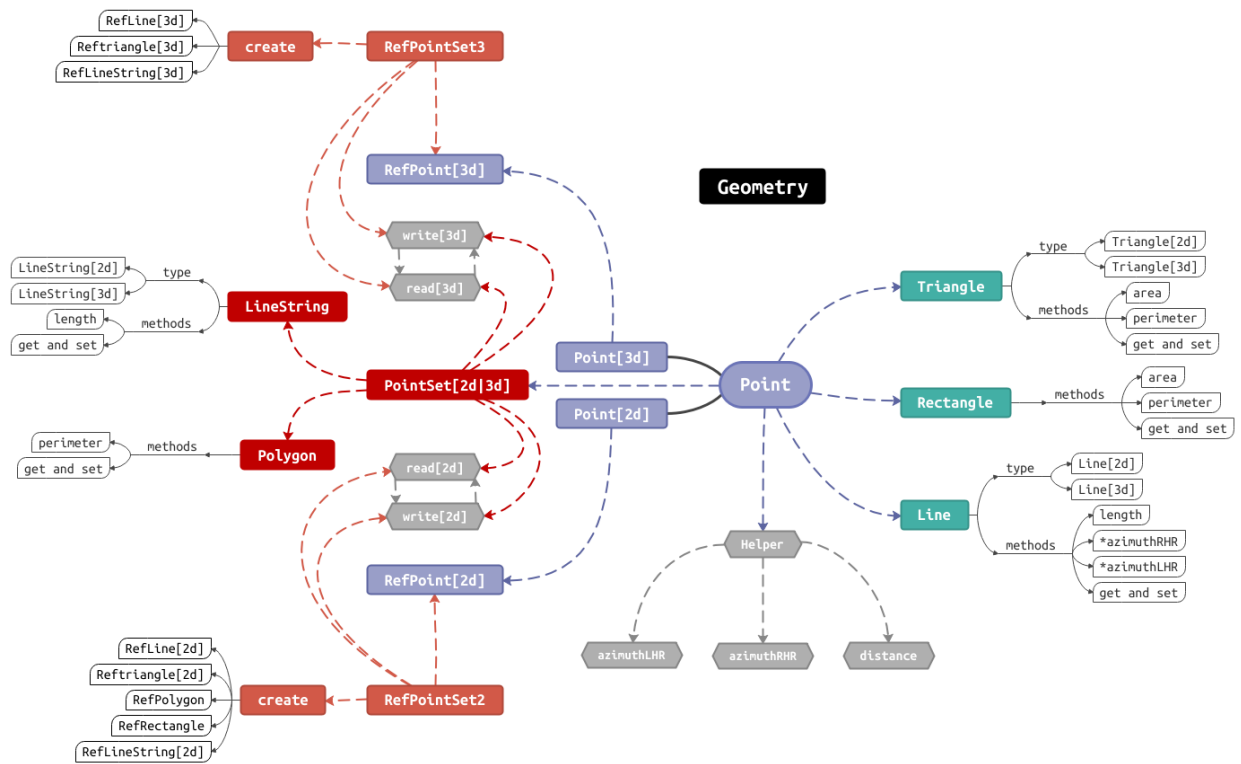
email 3079625093@qq.com

1. Overview

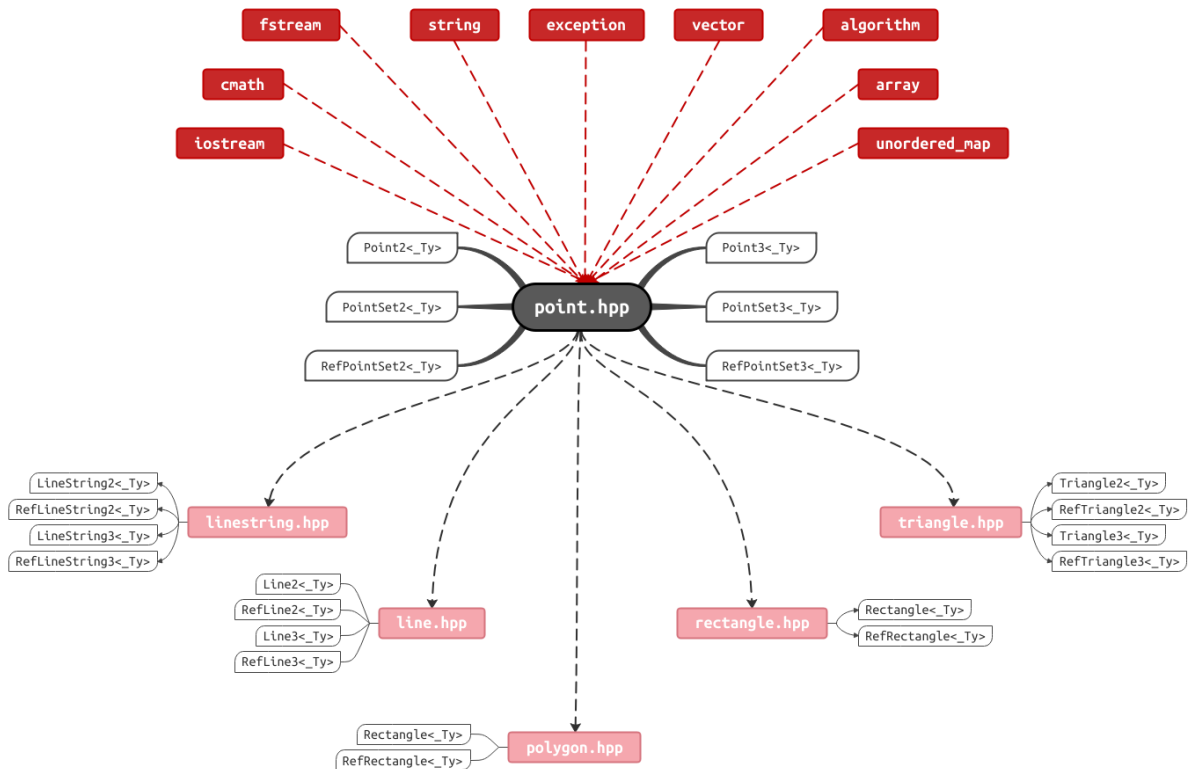
The CPP library provides two dimension point template classes: `Point2<Ty>` and `Point3<Ty>`. It also provides related operations based on two kinds of points, such as conventional "write" operation, "read" operation and distance calculation of point set, and azimuth calculation based on point2. You can easily use it to assist development. Here are some details of this class.

And because it's a template class, you can just copy the head file to your project and use it.

2. Code Structure



3. Classes Belongs



4. Using example

Point2<Ty>

```
1 void foo_point2()
2 {
3     Point2f p1(0.6, 0.4);
4     Point2f p2(1.9, 2.7);
5     Point2f p3(0.6, 0.4);
6     Point2f p4(1.9, 2.7);
7     PointSet2f ps({p1, p2, p3, p4});
8     try
9     {
10         // distance between tow points
11         std::cout << distance(p1, p2) << std::endl;
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
18         ps.read("../output/point2.bin");
19         // way two.
20         // write mode : std::ios::out
21         ps.write("../output/point2.txt", std::ios::out);
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;
29         }
30     }
31     catch (const std::exception &e)
32     {
33         std::cerr << e.what() << '\n';
34     }
35     return;
36 }
37 /** output
38  * 2.64197
39  * [0.6, 0.4]
40  * [1.9, 2.7]
41  * [0.6, 0.4]
42  * [1.9, 2.7]
43  */
```

Point3<Ty>

```
1 void foo_point3()
2 {
3     Point3f p1(0.6, 0.4, 1.1);
4     Point3f p2(1.9, 2.7, 2.3);
5     Point3f p3(0.6, 0.4, 3.5);
6     Point3f p4(1.9, 2.7, 4.6);
7     PointSet3f ps = {p1, p2, p3, p4};
8     try
```

```

9      {
10         // distance between tow points
11         std::cout << distance(p1, p2) << std::endl;
12         // write and read point data
13         // way one.
14         // default write mode : std::ios::out | std::ios::binary
15         ps.write("../output/point3.bin");
16         ps.clear();
17         // default read mode : std::ios::in | std::ios::binary
18         ps.read("../output/point3.bin");
19
20         // way two.
21         // write mode : std::ios::out
22         ps.write("../output/point3.txt", std::ios::out);
23         ps.clear();
24         // read mode : std::ios::in
25         ps.read("../output/point3.txt", std::ios::in);
26         // print points
27         for (const auto &elem : ps)
28         {
29             std::cout << elem << std::endl;
30         }
31     }
32     catch (const std::exception &e)
33     {
34         std::cerr << e.what() << '\n';
35     }
36     return;
37 }
38 /** output
39  * 2.90172
40  * [0.6, 0.4, 1.1]
41  * [1.9, 2.7, 2.3]
42  * [0.6, 0.4, 3.5]
43  * [1.9, 2.7, 4.6]
44  */

```

PointSet23<Ty>

```

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

```

Point_cast<Ty>

```
1 void foo_ponitCast_test()
2 {
3     Point3f p(1, 2, 6);
4     Point2f p2(2, 6);
5     auto ary = static_cast<Point3f::ary_type>(p);
6     auto ary2 = static_cast<Point2f::ary_type>(p2);
7
8     std::cout << ary[0] << ',' << ary[1] << ',' << ary[2] << std::endl;
9     std::cout << ary2[0] << ',' << ary2[1] << std::endl;
10
11     std::cout << Point3f(ary) << std::endl;
12     std::cout << Point2f(ary2) << std::endl;
13
14     return;
15 }
16 /** output
17  * 1,2,6
18  * 2,6
19  * [1, 2, 6]
20  * [2, 6]
21  */
```

Triangle2<Ty>

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

Triangle3<Ty>

```
1 void foo_triangle3()
2 {
3     ns_geo::Point3<double> points[3] = {
4         Point3d(0, 0, 0),
5         Point3d(2, 2, 2),
6         Point3d(2, 0, 0)};
```

```

7     ns_geo::Triangle3d tri(points);
8     std::cout << tri << std::endl;
9     std::cout << "area : " << tri.area() << std::endl;
10    std::cout << "perimeter : " << tri.perimeter() << std::endl;
11    return;
12 }
13 /** output
14  * {[0, 0, 0], [2, 2, 2], [2, 0, 0]}
15  * area : 2.82843
16  * perimeter : 8.29253
17  */

```

Line2<Ty>

```

1 void foo_line2()
2 {
3     ns_geo::Line2d line(Point2d(0, 0), Point2d(2, 2));
4     std::cout << line << std::endl;
5     std::cout << "length : " << line.length() << std::endl;
6     for (const auto &elem : line.points())
7         std::cout << elem << std::endl;
8     return;
9 }
10 /** output
11  * {[0, 0], [2, 2]}
12  * length : 2.82843
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));
4     std::cout << line << std::endl;
5     std::cout << "length : " << line.length() << std::endl;
6     for (const auto &elem : line.points())
7         std::cout << elem << std::endl;
8     return;
9 }
10 /** output
11  * {[0, 0, 0], [2, 2, 2]}
12  * length : 3.4641
13  * [0, 0, 0]
14  * [2, 2, 2]
15  */

```

Rectangle<Ty>

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

Polygon<Ty>

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

LineString23<Ty>

```
1 void foo_lineString23()
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;
8     std::cout << ls.length() << std::endl;
9     LineString2d ls2({Point2d(0, 9),
10                      Point2d(1, 9),
11                      Point2d(1, 9),
12                      Point2d(0, 9)});
13     std::cout << ls2 << std::endl;
```

```

14     std::cout << ls2.length() << std::endl;
15     return;
16 }
17 /** output
18  * {[0, 0, 9], [0, 1, 9], [1, 1, 9], [1, 0, 9]}
19  * 3
20  * {[0, 9], [1, 9], [1, 9], [0, 9]}
21  * 2
22  */

```

RefPoint23<Ty>

```

1 void foo_refpoint23()
2 {
3     double ary1[3] = {1, 2, 3};
4     RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
5     RefPoint3d p2(1, ary1);
6     std::cout << distance(p1, p2) << std::endl;
7     std::cout << p1 << std::endl;
8
9     double ary2[2] = {2, 3};
10    RefPoint2d p3(0, RefPoint2d::ary_type{0, 0});
11    RefPoint2d p4(1, ary2);
12    std::cout << distance(p3, p4) << std::endl;
13    std::cout << p3 << std::endl;
14 }
15 /** output
16  * 3.74166
17  * {0: [0, 0, 0]}
18  * 3.60555
19  * {0: [0, 0]}
20  */

```

RefPointSet23<Ty>

```

1 void foo_refpointset23()
2 {
3     double ary2[2] = {2, 3};
4     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
5     RefPoint2d p2(1, ary2);
6     RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
7     RefPoint2d p4(4, ary2);
8     RefPointSet2d ps2;
9     ps2.insert(p2);
10    ps2.insert(p4);
11    ps2.insert(p3);
12    ps2.insert(p1);
13
14    for (const auto &refp : ps2)
15        std::cout << refp.second << std::endl;
16    std::cout << ps2.size() << std::endl;
17
18    RefPoint3d p5(0, RefPoint3d::ary_type{0, 0, 0});

```



```

19     RefPoint3d p6(1, RefPoint3d::ary_type{0, 1, 0});
20     RefPoint3d p7(2, RefPoint3d::ary_type{0, 0, 1});
21     RefPoint3d p8(4, RefPoint3d::ary_type{1, 0, 0});
22     RefPointSet3d ps3;
23     ps3.insert(p5);
24     ps3.insert(p6);
25     ps3.insert(p7);
26     ps3.insert(p8);
27     for (const auto &refp : ps3)
28         std::cout << refp.second << std::endl;
29     std::cout << ps3.size() << std::endl;
30 }
31 /** output
32  * {0: [0, 0]}
33  * {2: [0, 0]}
34  * {4: [2, 3]}
35  * {1: [2, 3]}
36  * 4
37  * {4: [1, 0, 0]}
38  * {2: [0, 0, 1]}
39  * {1: [0, 1, 0]}
40  * {0: [0, 0, 0]}
41  * 4
42  */

```

RefLine23<Ty>

```

1 void foo_refline2()
2 {
3     double ary2[2] = {2, 3};
4     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
5     RefPoint2d p2(1, ary2);
6     RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
7     RefPoint2d p4(4, ary2);
8     RefPointSet2d ps;
9     ps.insert(p2);
10    ps.insert(p4);
11    ps.insert(p3);
12    ps.insert(p1);
13    for (const auto &refp : ps)
14        std::cout << refp.second << std::endl;
15    auto reffline = ps.createRefLine2(0, 1);
16    std::cout << reffline << std::endl;
17    std::cout << reffline.length() << std::endl;
18 }
19 /** output
20  * {0: [0, 0]}
21  * {2: [0, 0]}
22  * {4: [2, 3]}
23  * {1: [2, 3]}
24  * {0: [0, 0], 1: [2, 3]}
25  * 3.60555
26  */
27
28 void foo_refline3()
29 {
30     RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});

```

```

31     RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
32     RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
33     RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
34     RefPointSet3d ps;
35     ps.insert(p2);
36     ps.insert(p4);
37     ps.insert(p3);
38     ps.insert(p1);
39     for (const auto &refp : ps)
40         std::cout << refp.second << std::endl;
41     auto refline = ps.createRefLine3(0, 1);
42     std::cout << refline << std::endl;
43     std::cout << refline.length() << std::endl;
44     auto ary = refline.points();
45 }
46 /** output
47  * {0: [0, 0, 0]}
48  * {2: [0, 0, 1]}
49  * {4: [1, 0, 0]}
50  * {1: [0, 1, 0]}
51  * {0: [0, 0, 0], 1: [0, 1, 0]}
52  * 1
53  */

```

RefRectangle<Ty>

```

1 void foo_refrectangle()
2 {
3     double ary2[2] = {2, 3};
4     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
5     RefPoint2d p2(1, ary2);
6     RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
7     RefPoint2d p4(4, ary2);
8     RefPointSet2d ps;
9     ps.insert(p2);
10    ps.insert(p4);
11    ps.insert(p3);
12    ps.insert(p1);
13    for (const auto &refp : ps)
14        std::cout << refp.second << std::endl;
15    auto rect = ps.createRefRectangle(0, 1);
16    std::cout << rect << std::endl;
17    std::cout << rect.area() << std::endl;
18    std::cout << rect.perimeter() << std::endl;
19 }
20 /** output
21  * {0: [0, 0]}
22  * {2: [0, 0]}
23  * {4: [2, 3]}
24  * {1: [2, 3]}
25  * {0: [0, 0], 1: [2, 3]}
26  * 6
27  * 10
28  */

```

RefTriangle23<Ty>

```
1 void foo_reftriangle2()
2 {
3     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
4     RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
5     RefPoint2d p3(2, RefPoint2d::ary_type{0, 2});
6     RefPoint2d p4(4, RefPoint2d::ary_type{3, 0});
7     RefPointSet2d ps;
8     ps.insert(p2);
9     ps.insert(p4);
10    ps.insert(p3);
11    ps.insert(p1);
12    for (const auto &refp : ps)
13        std::cout << refp.second << std::endl;
14    auto tri = ps.createRefTriangle2(0, 1, 2);
15    std::cout << tri << std::endl;
16    std::cout << tri.perimeter() << std::endl;
17    std::cout << tri.area() << std::endl;
18 }
19 /** output
20  * {0: [0, 0]}
21  * {2: [0, 2]}
22  * {4: [3, 0]}
23  * {1: [1, 0]}
24  * {0: [0, 0], 1: [1, 0], 2: [0, 2]}
25  * 5.23607
26  * 1
27  */
28
29 void foo_reftriangle3()
30 {
31     RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
32     RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
33     RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
34     RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
35     RefPointSet3d ps;
36     ps.insert(p2);
37     ps.insert(p4);
38     ps.insert(p3);
39     ps.insert(p1);
40     for (const auto &refp : ps)
41         std::cout << refp.second << std::endl;
42     auto tri = ps.createRefTriangle3(0, 1, 2);
43     std::cout << tri << std::endl;
44     std::cout << tri.area() << std::endl;
45     std::cout << tri.perimeter() << std::endl;
46 }
47 /** output
48  * {0: [0, 0, 0]}
49  * {2: [0, 0, 1]}
50  * {4: [1, 0, 0]}
51  * {1: [0, 1, 0]}
52  * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1]}
53  * 0.5
54  * 3.41421
55  */
```

RefPolygon<Ty>

```
1 void foo_refpolygon()
2 {
3     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
4     RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
5     RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
6     RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
7     RefPointSet2d rps;
8     rps.insert(p2);
9     rps.insert(p4);
10    rps.insert(p3);
11    rps.insert(p1);
12    auto polygon = rps.createRefPolygon({0, 1, 2, 4});
13    std::cout << polygon << std::endl;
14    std::cout << "perimeter : " << polygon.perimeter() << std::endl;
15 }
16 /** output
17  * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
18  * perimeter : 4
19  */
```

RefLineString23<Ty>

```
1 void foo_reflinestring2()
2 {
3     RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
4     RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
5     RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
6     RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
7     RefPointSet2d rps;
8     rps.insert(p2);
9     rps.insert(p4);
10    rps.insert(p3);
11    rps.insert(p1);
12    auto ls = rps.createRefLineString2({0, 1, 2, 4});
13    std::cout << ls << std::endl;
14    std::cout << "length : " << ls.length() << std::endl;
15 }
16 /** output
17  * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
18  * length : 3
19  */
20
21 void foo_reflinestring3()
22 {
23     RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
24     RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
25     RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
26     RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
27     RefPointSet3d rps;
28     rps.insert(p2);
29     rps.insert(p4);
30     rps.insert(p3);
31     rps.insert(p1);
32     auto ls = rps.createRefLineString3({0, 1, 2, 4});
```

```

33     std::cout << ls << std::endl;
34     std::cout << "length : " << ls.length() << std::endl;
35 }
36 /** output
37  * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1], 4: [1, 0, 0]}
38  * length : 3.82843
39  */

```

RefPointSet_WriteRead23<Ty>

```

1  void foo_refpointset2_write()
2  {
3      RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
4      RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
5      RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
6      RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
7      RefPointSet2d rps;
8      rps.insert(p2);
9      rps.insert(p4);
10     rps.insert(p3);
11     rps.insert(p1);
12     rps.write("../output/refpointset2.bin");
13     rps.clear();
14     rps.read("../output/refpointset2.bin");
15     for (const auto &[id, refp] : rps)
16         std::cout << refp << std::endl;
17 }
18 /** output
19  * {1: [1, 0]}
20  * {4: [0, 1]}
21  * {2: [1, 1]}
22  * {0: [0, 0]}
23  */
24 void foo_refpointset3_write()
25 {
26     RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
27     RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
28     RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
29     RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
30     RefPointSet3d rps;
31     rps.insert(p2);
32     rps.insert(p4);
33     rps.insert(p3);
34     rps.insert(p1);
35     rps.write("../output/refpointset3.bin");
36     rps.clear();
37     rps.read("../output/refpointset3.bin");
38     for (const auto &[id, refp] : rps)
39         std::cout << refp << std::endl;
40 }
41 /** output
42  * {1: [0, 1, 0]}
43  * {4: [1, 0, 0]}
44  * {2: [0, 0, 1]}
45  * {0: [0, 0, 0]}
46  */

```

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