

# Geometry

A CPP Template Library

## 0. Author

**Name** csl

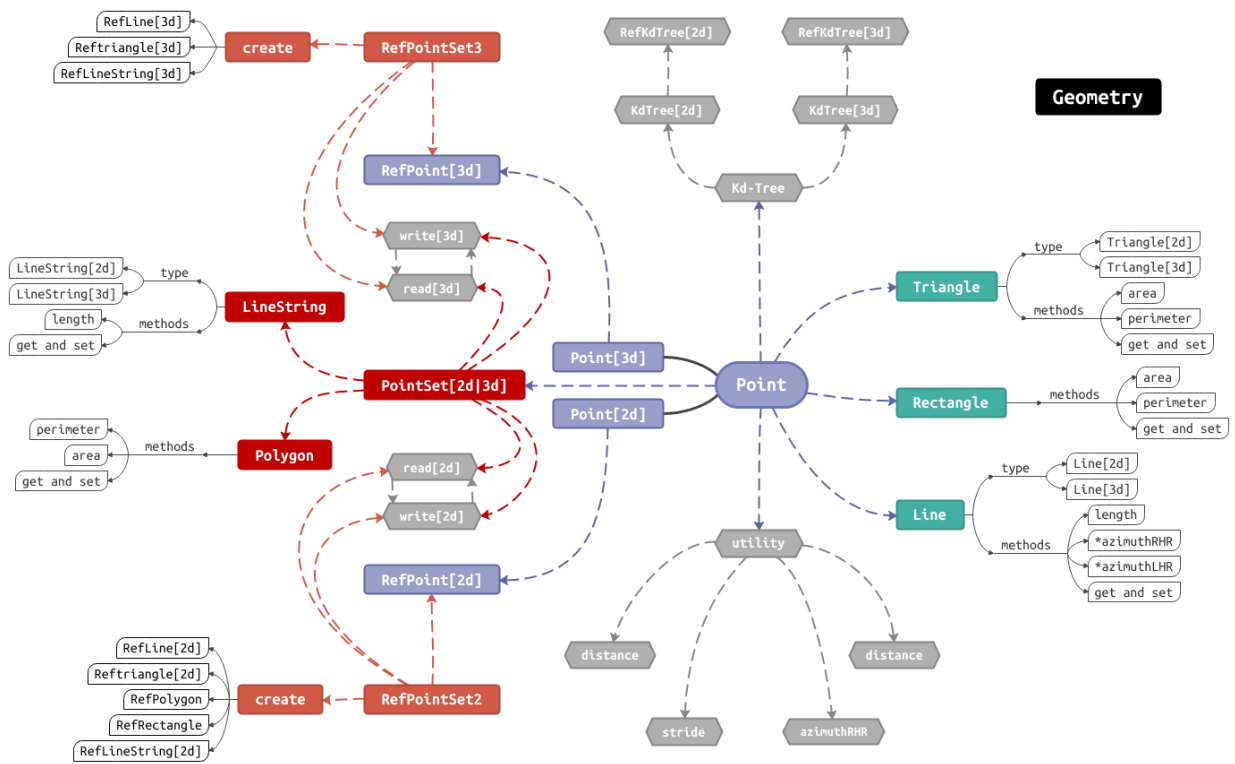
**E-Mail** [3079625093@qq.com](mailto:3079625093@qq.com)

## 1. Overview

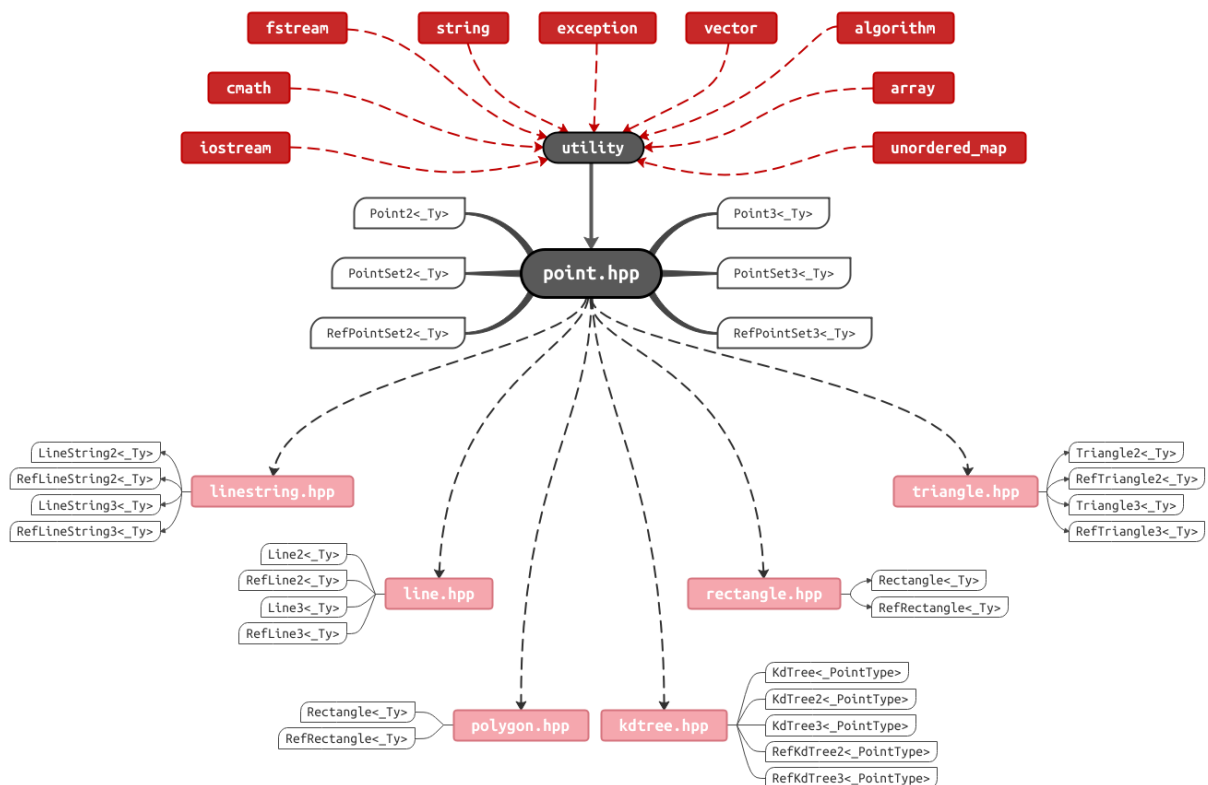
This CPP library mainly provides two dimension point template classes: `Point2<Ty>` and `Point3<Ty>`. It also provides related geometries and operations based on two kinds of points, such as conventional "write" operation, "read" operation and distance calculation of point set, and azimuth calculation 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.

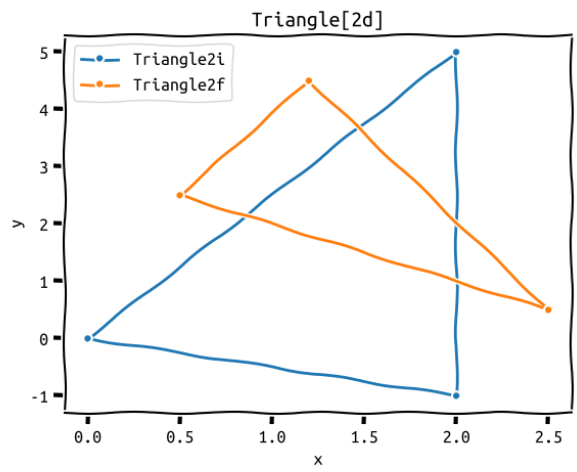
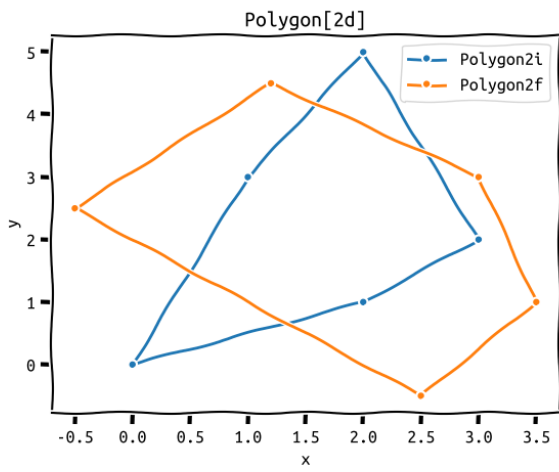
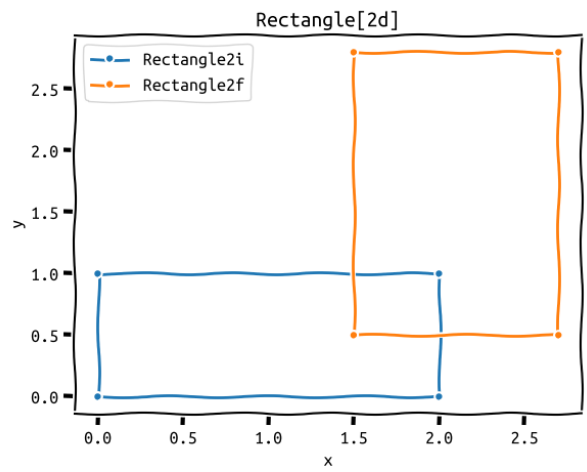
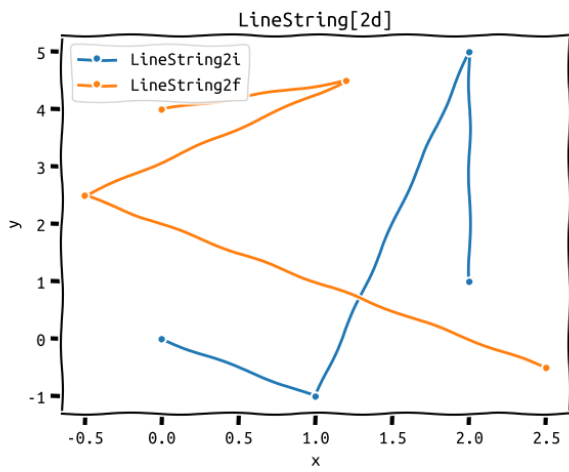
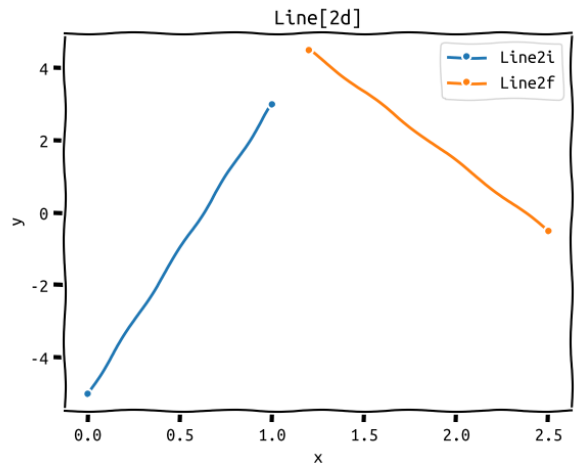
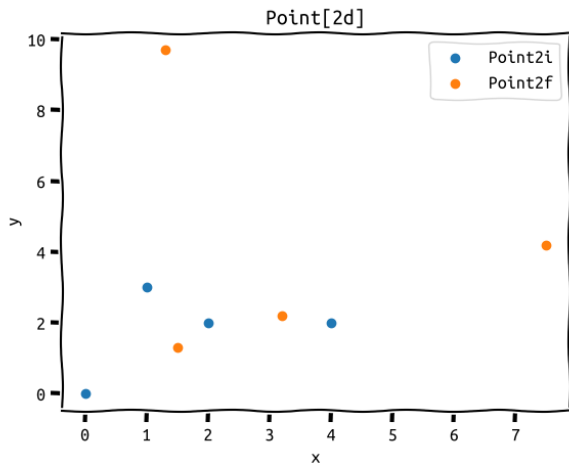
## 2. Code Structure

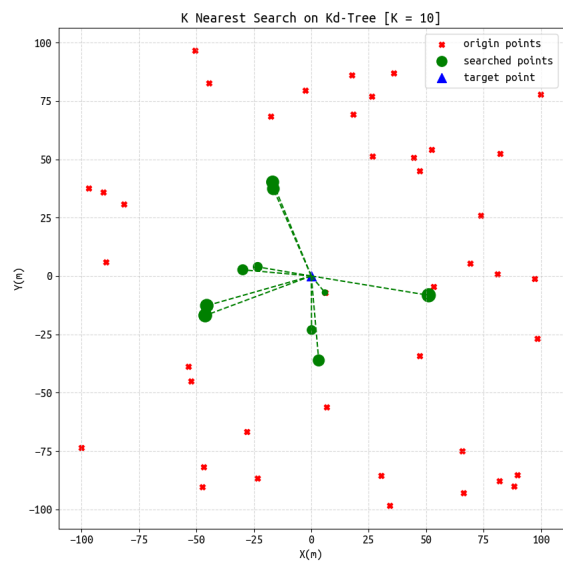
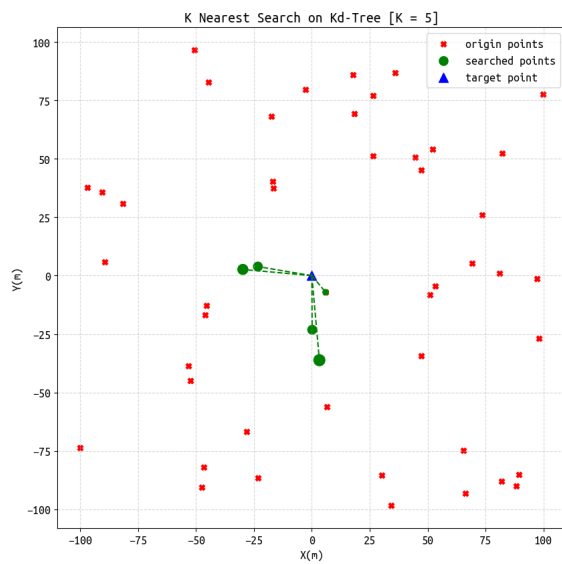
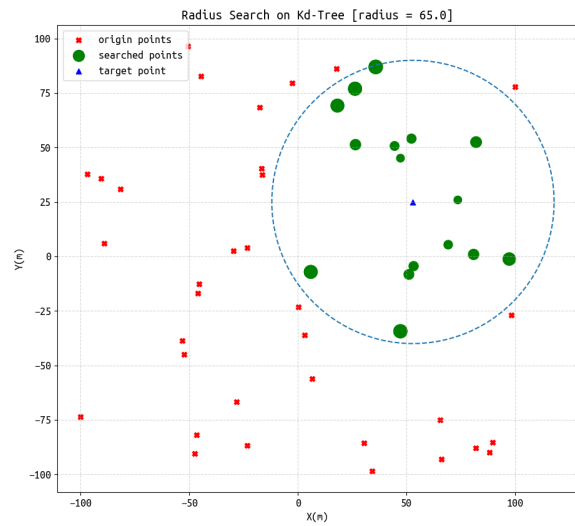
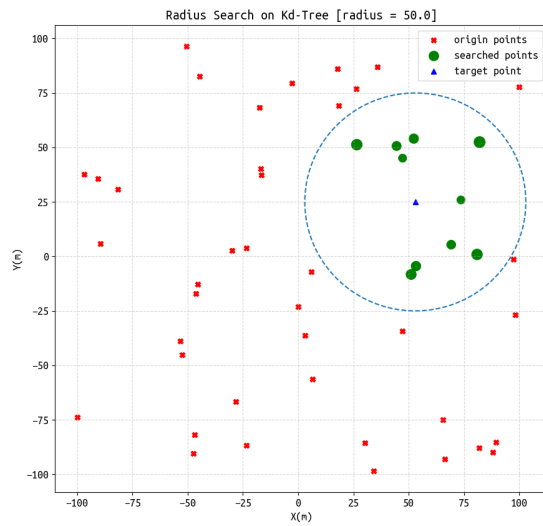


### 3. Classes Belongs



### 4. Figures





## 5. Using example

**Point2<Ty>**

```
1 void foo_point2()
2 {
3     PointSet2f ps;
4     ps.push_back({0.6, 0.4});
5     ps.push_back({1.9, 2.7});
6     ps.push_back({0.6, 0.4});
7     ps.push_back({1.9, 2.7});
8     try
9     {
10        // distance between tow points
11        std::cout << distance(ps.front(), ps.back()) << 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     PointSet3f ps;
4     ps.push_back({0.6, 0.4, 1.1});
5     ps.push_back({1.9, 2.7, 2.3});
6     ps.push_back({0.6, 0.4, 1.1});
7     ps.push_back({1.9, 2.7, 2.3});
8     try
9     {
10         // distance between tow points
11         std::cout << distance(ps.front(), ps.back()) << 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_pointCast_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(0.5, 2),
6                       Point2d(1, 1),
7                       Point2d(1, 0)});
8     std::cout << polygon << std::endl;
9     std::cout << "perimeter : " << polygon.perimeter() << std::endl;
10    std::cout << "area : " << polygon.area() << std::endl;
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>***

```

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     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;
11    std::cout << rps2.size() << std::endl;
12
13    RefPointSet3d rps3;
14    rps3.insert({0, RefPoint3d::ary_type{0, 0, 0}});
15    rps3.insert({1, RefPoint3d::ary_type{0, 1, 0}});
16    rps3.insert({2, RefPoint3d::ary_type{0, 0, 1}});
17    rps3.insert({3, RefPoint3d::ary_type{1, 0, 0}});
18    for (const auto &refp : rps3)
19        std::cout << refp.second << std::endl;
20    std::cout << rps3.size() << std::endl;
21 }
22 /** output
23  * {0: [0, 0]}
24  * {2: [0, 0]}
25  * {4: [2, 3]}
26  * {1: [2, 3]}
27  * 4
28  * {4: [1, 0, 0]}
29  * {2: [0, 0, 1]}
30  * {1: [0, 1, 0]}
31  * {0: [0, 0, 0]}
```

```
32 | * 4
33 | */
```

### RefLine23<Ty>

```
1 void foo_refline2()
2 {
3     double ary2[2] = {2, 3};
4     RefPointSet2d rps;
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;
11
12     auto reffline = rps.createRefLine2(0, 1);
13     std::cout << reffline << std::endl;
14     std::cout << reffline.length() << std::endl;
15 }
16 /** output
17  * {0: [0, 0]}
18  * {2: [0, 0]}
19  * {4: [2, 3]}
20  * {1: [2, 3]}
21  * {0: [0, 0], 1: [2, 3]}
22  * 3.60555
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}});
31     rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
32     for (const auto &refp : ps)
33         std::cout << refp.second << std::endl;
34
35     auto reffline = rps.createRefLine3(0, 1);
36     std::cout << reffline << std::endl;
37     std::cout << reffline.length() << std::endl;
38     auto ary = reffline.points();
39 }
40 /** output
41  * {0: [0, 0, 0]}
42  * {2: [0, 0, 1]}
43  * {4: [1, 0, 0]}
44  * {1: [0, 1, 0]}
45  * {0: [0, 0, 0], 1: [0, 1, 0]}
46  * 1
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}});
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;
11
12     auto rect = rps.createRefRectangle(0, 1);
13     std::cout << rect << std::endl;
14     std::cout << rect.area() << std::endl;
15     std::cout << rect.perimeter() << std::endl;
16 }
17 /** output
18  * {0: [0, 0]}
19  * {2: [0, 0]}
20  * {4: [2, 3]}
21  * {1: [2, 3]}
22  * {0: [0, 0], 1: [2, 3]}
23  * 6
24  * 10
25  */
```

## RefTriangle23<Ty>

```
1 void foo_reftriangle2()
2 {
3     double ary2[2] = {2, 3};
4     RefPointSet2d rps;
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;
11     auto tri = rps.createRefTriangle2(0, 1, 2);
12
13     std::cout << tri << std::endl;
14     std::cout << tri.perimeter() << std::endl;
15     std::cout << tri.area() << std::endl;
16 }
17 /** output
18  * {0: [0, 0]}
19  * {2: [0, 2]}
20  * {4: [3, 0]}
21  * {1: [1, 0]}
22  * {0: [0, 0], 1: [1, 0], 2: [0, 2]}
23  * 5.23607
24  * 1
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}});
32     rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
33     rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
34     for (const auto &refp : rps)
35         std::cout << refp.second << std::endl;
36
37     auto tri = rps.createRefTriangle3(0, 1, 2);
38     std::cout << tri << std::endl;
39     std::cout << tri.area() << std::endl;
40     std::cout << tri.perimeter() << std::endl;
41 }
42 /** output
43  * {0: [0, 0, 0]}
44  * {2: [0, 0, 1]}
45  * {4: [1, 0, 0]}
46  * {1: [0, 1, 0]}
47  * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1]}
48  * 0.5
49  * 3.41421
50  */

```

### ***RefPolygon<Ty>***

```

1 void foo_refpolygon()
2 {
3     RefPointSet2d rps;
4     rps.insert({0, RefPoint2d::ary_type{0, 0}});
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;
11    std::cout << "perimeter : " << polygon.perimeter() << std::endl;
12 }
13 /** output
14  * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
15  * perimeter : 4
16  */

```

### ***RefLineString23<Ty>***

```

1 void foo_reflinestring2()
2 {
3     RefPointSet2d rps;
4     rps.insert({0, RefPoint2d::ary_type{0, 0}});
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 ls = rps.createRefLineString2({0, 1, 2, 4});
10    std::cout << ls << std::endl;
11    std::cout << "length : " << ls.length() << std::endl;
12}
13/** output
14 * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
15 * length : 3
16 */
17
18void foo_reflinestring3()
19{
20    RefPointSet3d rps;
21    rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
22    rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
23    rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
24    rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
25
26    auto ls = rps.createRefLineString3({0, 1, 2, 4});
27    std::cout << ls << std::endl;
28    std::cout << "length : " << ls.length() << std::endl;
29}
30/** output
31 * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1], 4: [1, 0, 0]}
32 * length : 3.82843
33 */

```

### ***RefPointSet\_WriteRead23<Ty>***

```

1 void foo_refpointset2_write()
2 {
3     RefPointSet2d rps;
4     rps.insert({0, RefPoint2d::ary_type{0, 0}});
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     rps.write("../output/refpointset2.bin");
10    rps.clear();
11    rps.read("../output/refpointset2.bin");
12    for (const auto &[id, refp] : rps)
13        std::cout << refp << std::endl;
14}
15/** output
16 * {1: [1, 0]}
17 * {4: [0, 1]}
18 * {2: [1, 1]}
19 * {0: [0, 0]}
20 */
21void foo_refpointset3_write()
22{
23    RefPointSet3d rps;
24    rps.insert({0, RefPoint3d::ary_type{0, 0, 0}});
25    rps.insert({1, RefPoint3d::ary_type{0, 1, 0}});
26    rps.insert({2, RefPoint3d::ary_type{0, 0, 1}});
27    rps.insert({3, RefPoint3d::ary_type{1, 0, 0}});
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;
34 }
35 /** output
36  * {1: [0, 1, 0]}
37  * {4: [1, 0, 0]}
38  * {2: [0, 0, 1]}
39  * {0: [0, 0, 0]}
40  */

```

## distance\_

```

1 void foo_distance()
2 {
3     Point2d p1(1, 1);
4     Point2d p2(2, 2);
5     Line2d line({0, 0, 0, 1});
6     std::cout << "p1 -> p2 : " << distance(p1, p2) << std::endl;
7     std::cout << "p1 -> line : " << distance(p1, line) << std::endl;
8     double ary2[2] = {2, 3};
9     RefPointSet2d rps;
10    rps.insert({0, RefPoint2d::ary_type{0, 0}});
11    rps.insert({1, ary2});
12    rps.insert({2, RefPoint2d::ary_type{0, 0}});
13    rps.insert({4, ary2});
14    auto refline = rps.createRefLine2(0, 1);
15    std::cout << "p1 -> refline : " << distance(p1, Line2d(refline)) << std::endl;
16    return;
17 }
18 /** output
19  * p1 -> p2 : 1.41421
20  * p1 -> line : 1
21  * p1 -> refline : 0.27735
22  */

```

## kdtree

```

1 void foo_kdtree()
2 {
3     PointSet3f ps({{3, 1, 4},
4                    {2, 3, 7},
5                    {2, 0, 3},
6                    {2, 4, 5},
7                    {1, 4, 4},
8                    {0, 5, 7}});
9     KdTree3<Point3f> kdtree(ps);
10    kdtree.printKdTree();
11    return;
12 }
13 /** output

```

```

14  * [2, 4, 5]:[Y] [3, 1, 4]:[Z] [2, 0, 3]:[X] [2, 3, 7]:[X] [0, 5, 7]:[Z] [1, 4, 4]:[X]
15  */
16  void foo_refkdtree()
17  {
18      RefPointSet3f ps;
19      ps.insert({0, 3, 1, 4});
20      ps.insert({1, 2, 3, 7});
21      ps.insert({2, 2, 0, 3});
22      ps.insert({3, 2, 4, 5});
23      ps.insert({4, 1, 4, 4});
24      ps.insert({5, 0, 5, 7});
25      RefKdTree3f kdtree(ps);
26      kdtree.printKdTree();
27      return;
28  }
29  /** output
30  * {4: [1, 4, 4]}:[Y] {0: [3, 1, 4]}:[Z] {2: [2, 0, 3]}:[X] {1: [2, 3, 7]}:[X] {5: [0, 5, 7]}:[Z] {3:
31  * [2, 4, 5]}:[X]
32  */
33  void foo_kdtreeRadiusSearch()
34  {
35      PointSet2f ps({{2, 3},
36                    {5, 4},
37                    {9, 6},
38                    {4, 7},
39                    {8, 1},
40                    {7, 2}});
41      KdTree2f Kdtree(ps);
42      Kdtree.printKdTree();
43      std::vector<float> dis;
44      std::vector<Point2f> sps;
45      Kdtree.radiusSearch({6, 5}, 4, sps, dis);
46      for (int i = 0; i != dis.size(); ++i)
47          std::cout << sps.at(i) << ' ' << dis.at(i) << std::endl;
48      return;
49  }
50  /** output
51  * [7, 2]:[X] [5, 4]:[Y] [2, 3]:[X] [4, 7]:[X] [9, 6]:[Y] [8, 1]:[X]
52  * [4, 7] 2.82843
53  * [5, 4] 1.41421
54  * [7, 2] 3.16228
55  * [9, 6] 3.16228
56  */
57  void foo_kdtreeNearestKSearch()
58  {
59      std::default_random_engine e;
60      std::uniform_real_distribution<float> u(-100.0f, 100.0f);
61      PointSet2f ps;
62      std::fstream file1("../pyDrawer/kdtree/nearest1.csv", std::ios::out);
63      std::fstream file2("../pyDrawer/kdtree/nearest2.csv", std::ios::out);
64      for (int i = 0; i != 50; ++i)
65      {
66          ps.push_back({u(e), u(e)});
67          file1 << ps.back().x() << ',' << ps.back().y() << std::endl;
68      }
69      KdTree2f kdtree(ps);
70      std::vector<float> dis;
71      std::vector<Point2f> sps;
72      kdtree.nearestKSearch({0, 0}, 6, sps, dis);
73      for (int i = 0; i != dis.size(); ++i)
74          file2 << sps.at(i).x() << ',' << sps.at(i).y() << ',' << dis.at(i) << std::endl;

```



```
74 |     return;  
75 | }
```

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