Geometry

A CPP Template Class

0. Author

name csl

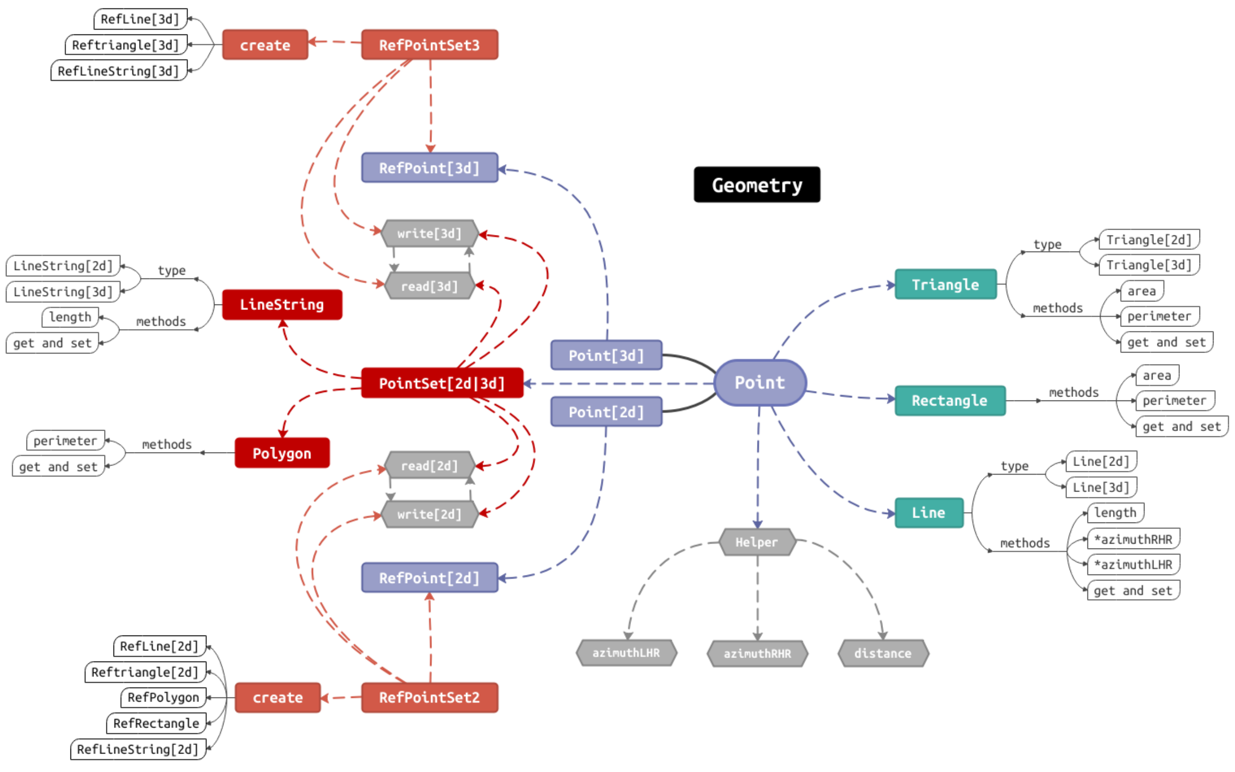
email 3079625093@qq.com

1. Overview

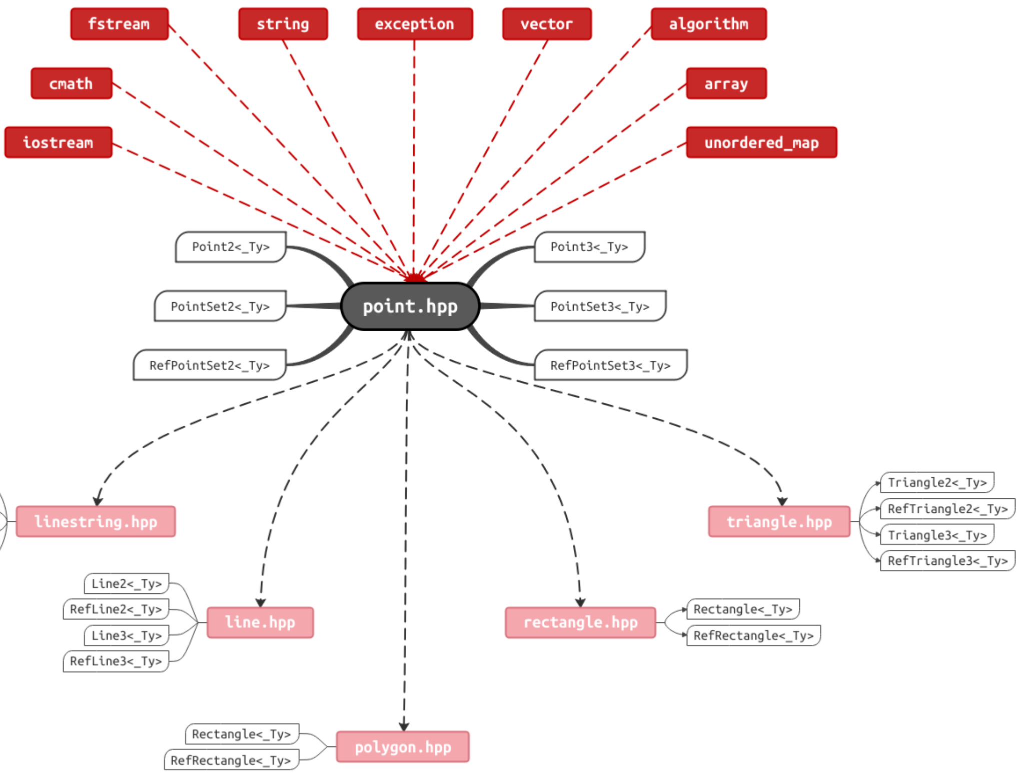
The CPP library provides two dimension point template classes: **Point2<*Ty*>** and **Point3<*Ty*>**.It also provides related operations based on two kinds of points, such as conventional "write" operation, "read" operation and distance calculation of point set, and azimuth calculation based on point2.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

```cpp
void foo_point2()
{
    Point2f p1(0.6, 0.4);
    Point2f p2(1.9, 2.7);
    Point2f p3(0.6, 0.4);
    Point2f p4(1.9, 2.7);
    PointSet2f ps({p1, p2, p3, p4});
    try
    {
        // distance between tow points
        std::cout << distance(p1, p2) << std::endl;
        // write and read point data
        // way one.
        // default write mode : std::ios::out | std::ios::binary
        ps.write("../output/point2.bin");
        ps.clear();
        // default read mode : std::ios::in | std::ios::binary
        ps.read("../output/point2.bin");
        // way two.
        // write mode : std::ios::out
        ps.write("../output/point2.txt", std::ios::out);
        ps.clear();
        // read mode : std::ios::in
        ps.read("../output/point2.txt", std::ios::in);
        // print points
        for (const auto &elem : ps)
        {
            std::cout << elem << std::endl;
        }
    }
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
    }
    return;
}
/** output
 * 2.64197
 * [0.6, 0.4]
 * [1.9, 2.7]
 * [0.6, 0.4]
 * [1.9, 2.7]
 */
```

Point3<Ty>

```cpp
void foo_point3()
{
    Point3f p1(0.6, 0.4, 1.1);
    Point3f p2(1.9, 2.7, 2.3);
    Point3f p3(0.6, 0.4, 3.5);
    Point3f p4(1.9, 2.7, 4.6);
    PointSet3f ps = {p1, p2, p3, p4};
    try
    {
        // distance between tow points
        std::cout << distance(p1, p2) << std::endl;
        // write and read point data
        // way one.
        // default write mode : std::ios::out | std::ios::binary
        ps.write("../output/point3.bin");
        ps.clear();
        // default read mode : std::ios::in | std::ios::binary
        ps.read("../output/point3.bin");

        // way two.
        // write mode : std::ios::out
        ps.write("../output/point3.txt", std::ios::out);
        ps.clear();
        // read mode : std::ios::in
        ps.read("../output/point3.txt", std::ios::in);
        // print points
        for (const auto &elem : ps)
        {
            std::cout << elem << std::endl;
        }
    }
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
    }
    return;
}
/** output
 * 2.90172
 * [0.6, 0.4, 1.1]
 * [1.9, 2.7, 2.3]
 * [0.6, 0.4, 3.5]
 * [1.9, 2.7, 4.6]
 */
```

PointSet23<Ty>

```cpp
void foo_pointset23()
{
    PointSet2f ps;
    ps.push_back(Point2f(1, 2));
    ps.push_back(Point2f(2, 3));
    ps.write("../output/pointset.csv", std::ios::out);
    ps.clear();
    ps.read("../output/pointset.csv", std::ios::in);
    for (const auto &point : ps)
        std::cout << point << std::endl;
    return;
}
/** output
 * [1, 2]
 * [2, 3]
 */
```

Point_cast<Ty>

```cpp
void foo_ponitCast_test()
{
    Point3f p(1, 2, 6);
    Point2f p2(2, 6);
    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;
    std::cout << ary2[0] << ',' << ary2[1] << std::endl;

    std::cout << Point3f(ary) << std::endl;
    std::cout << Point2f(ary2) << std::endl;

    return;
}
/** output
 * 1,2,6
 * 2,6
 * [1, 2, 6]
 * [2, 6]
 */
```

Triangle2<Ty>

```cpp
void foo_triangle2()
{
    ns_geo::Point2<double> points[3] = {
        Point2d(0, 0),
        Point2d(2, 2),
        Point2d(2, 0)};
    ns_geo::Triangle2d tri(points);
    std::cout << tri << std::endl;
    std::cout << "area : " << tri.area() << std::endl;
    std::cout << "perimeter : " << tri.perimeter() << std::endl;
    return;
}
/** output
 * {[0, 0], [2, 2], [2, 0]}
 * area : 2
 * perimeter : 6.82843
 */
```

Triangle3<Ty>

```cpp
void foo_triangle3()
{
    ns_geo::Point3<double> points[3] = {
        Point3d(0, 0, 0),
        Point3d(2, 2, 2),
        Point3d(2, 0, 0)};
    ns_geo::Triangle3d tri(points);
    std::cout << tri << std::endl;
    std::cout << "area : " << tri.area() << std::endl;
    std::cout << "perimeter : " << tri.perimeter() << std::endl;
    return;
}
/** output
 * {[0, 0, 0], [2, 2, 2], [2, 0, 0]}
 * area : 2.82843
 * perimeter : 8.29253
 */
```

Line2<Ty>

```cpp
void foo_line2()
{
    ns_geo::Line2d line(Point2d(0, 0), Point2d(2, 2));
    std::cout << line << std::endl;
    std::cout << "length : " << line.length() << std::endl;
    for (const auto &elem : line.points())
        std::cout << elem << std::endl;
    return;
}
/** output
 * {[0, 0], [2, 2]}
 * length : 2.82843
 * [0, 0]
 * [2, 2]
 */
```

Line3<Ty>

```cpp
void foo_line3()
{
    ns_geo::Line3d line(Point3d(0, 0, 0), Point3d(2, 2, 2));
    std::cout << line << std::endl;
    std::cout << "length : " << line.length() << std::endl;
    for (const auto &elem : line.points())
        std::cout << elem << std::endl;
    return;
}
/** output
 * {[0, 0, 0], [2, 2, 2]}
 * length : 3.4641
 * [0, 0, 0]
 * [2, 2, 2]
 */
```

Rectangle<Ty>

```cpp
void foo_rectangle()
{
    ns_geo::Rectangled rect(0, 4, 1, 0);
    std::cout << rect << std::endl;
    std::cout << "area : " << rect.area() << std::endl;
    std::cout << "peri : " << rect.perimeter() << std::endl;
    for (const auto &elem : rect.points())
        std::cout << elem << std::endl;
    return;
}
/** output
 * {[0, 4], [1, 0]}
 * area : 4
 * peri : 10
 * [0, 4]
 * [1, 0]
 */
```

Polygon<Ty>

```cpp
void foo_polygon()
{
    Polygond polygon({Point2d(0, 0),
                      Point2d(0, 1),
                      Point2d(1, 1),
                      Point2d(1, 0)});
    std::cout << polygon << std::endl;
    std::cout << "perimeter : " << polygon.perimeter() << std::endl;
    return;
}
/** output
 * {[0, 0], [0, 1], [1, 1], [1, 0]}
 * perimeter : 4
 */
```

LineString23<Ty>

```cpp
void foo_lineString23()
{
    LineString3d ls({Point3d(0, 0, 9),
                     Point3d(0, 1, 9),
                     Point3d(1, 1, 9),
                     Point3d(1, 0, 9)});
    std::cout << ls << std::endl;
    std::cout << ls.length() << std::endl;
    LineString2d ls2({Point2d(0, 9),
                      Point2d(1, 9),
                      Point2d(1, 9),
                      Point2d(0, 9)});
    std::cout << ls2 << std::endl;
    std::cout << ls2.length() << std::endl;
    return;
}
/** output
 * {[0, 0, 9], [0, 1, 9], [1, 1, 9], [1, 0, 9]}
 * 3
 * {[0, 9], [1, 9], [1, 9], [0, 9]}
 * 2
 */
```

RefPoint23<Ty>

```cpp
void foo_refpoint23()
{
    double ary1[3] = {1, 2, 3};
    RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
    RefPoint3d p2(1, ary1);
    std::cout << distance(p1, p2) << std::endl;
    std::cout << p1 << std::endl;

    double ary2[2] = {2, 3};
    RefPoint2d p3(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p4(1, ary2);
    std::cout << distance(p3, p4) << std::endl;
    std::cout << p3 << std::endl;
}
/** output
 * 3.74166
 * {0: [0, 0, 0]}
 * 3.60555
 * {0: [0, 0]}
 */
```

RefPointSet23<Ty>

```cpp
void foo_refpointset23()
{
    double ary2[2] = {2, 3};
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, ary2);
    RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
    RefPoint2d p4(4, ary2);
    RefPointSet2d ps2;
    ps2.insert(p2);
    ps2.insert(p4);
    ps2.insert(p3);
    ps2.insert(p1);

    for (const auto &refp : ps2)
        std::cout << refp.second << std::endl;
    std::cout << ps2.size() << std::endl;

    RefPoint3d p5(0, RefPoint3d::ary_type{0, 0, 0});
    RefPoint3d p6(1, RefPoint3d::ary_type{0, 1, 0});
    RefPoint3d p7(2, RefPoint3d::ary_type{0, 0, 1});
    RefPoint3d p8(4, RefPoint3d::ary_type{1, 0, 0});
    RefPointSet3d ps3;
    ps3.insert(p5);
    ps3.insert(p6);
    ps3.insert(p7);
    ps3.insert(p8);
    for (const auto &refp : ps3)
        std::cout << refp.second << std::endl;
    std::cout << ps3.size() << std::endl;
}
/** output
 * {0: [0, 0]}
 * {2: [0, 0]}
 * {4: [2, 3]}
 * {1: [2, 3]}
 * 4
 * {4: [1, 0, 0]}
 * {2: [0, 0, 1]}
 * {1: [0, 1, 0]}
 * {0: [0, 0, 0]}
 * 4
 */
```

RefLine23<Ty>

```
void foo_refline2()
{
    double ary2[2] = {2, 3};
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, ary2);
    RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
    RefPoint2d p4(4, ary2);
    RefPointSet2d ps;
    ps.insert(p2);
    ps.insert(p4);
    ps.insert(p3);
    ps.insert(p1);
    for (const auto &refp : ps)
        std::cout << refp.second << std::endl;
    auto refline = ps.createRefLine2(0, 1);
    std::cout << refline << std::endl;
    std::cout << refline.length() << std::endl;
}
/** output
 * {0: [0, 0]}
 * {2: [0, 0]}
 * {4: [2, 3]}
 * {1: [2, 3]}
 * {0: [0, 0], 1: [2, 3]}
 * 3.60555
 */

void foo_refline3()
{
    RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
```

```cpp
    RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
    RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
    RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
    RefPointSet3d ps;
    ps.insert(p2);
    ps.insert(p4);
    ps.insert(p3);
    ps.insert(p1);
    for (const auto &refp : ps)
        std::cout << refp.second << std::endl;
    auto refline = ps.createRefLine3(0, 1);
    std::cout << refline << std::endl;
    std::cout << refline.length() << std::endl;
    auto ary = refline.points();
}
/** output
 * {0: [0, 0, 0]}
 * {2: [0, 0, 1]}
 * {4: [1, 0, 0]}
 * {1: [0, 1, 0]}
 * {0: [0, 0, 0], 1: [0, 1, 0]}
 * 1
 */
```

RefRectangle<Ty>

```cpp
void foo_refrectangle()
{
    double ary2[2] = {2, 3};
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, ary2);
    RefPoint2d p3(2, RefPoint2d::ary_type{0, 0});
    RefPoint2d p4(4, ary2);
    RefPointSet2d ps;
    ps.insert(p2);
    ps.insert(p4);
    ps.insert(p3);
    ps.insert(p1);
    for (const auto &refp : ps)
        std::cout << refp.second << std::endl;
    auto rect = ps.createRefRectangle(0, 1);
    std::cout << rect << std::endl;
    std::cout << rect.area() << std::endl;
    std::cout << rect.perimeter() << std::endl;
}
/** output
 * {0: [0, 0]}
 * {2: [0, 0]}
 * {4: [2, 3]}
 * {1: [2, 3]}
 * {0: [0, 0], 1: [2, 3]}
 * 6
 * 10
 */
```

RefTriangle23<Ty>

```cpp
void foo_reftriangle2()
{
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
    RefPoint2d p3(2, RefPoint2d::ary_type{0, 2});
    RefPoint2d p4(4, RefPoint2d::ary_type{3, 0});
    RefPointSet2d ps;
    ps.insert(p2);
    ps.insert(p4);
    ps.insert(p3);
    ps.insert(p1);
    for (const auto &refp : ps)
        std::cout << refp.second << std::endl;
    auto tri = ps.createRefTriangle2(0, 1, 2);
    std::cout << tri << std::endl;
    std::cout << tri.perimeter() << std::endl;
    std::cout << tri.area() << std::endl;
}
/** output
 * {0: [0, 0]}
 * {2: [0, 2]}
 * {4: [3, 0]}
 * {1: [1, 0]}
 * {0: [0, 0], 1: [1, 0], 2: [0, 2]}
 * 5.23607
 * 1
 */

void foo_reftriangle3()
{
    RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
    RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
    RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
    RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
    RefPointSet3d ps;
    ps.insert(p2);
    ps.insert(p4);
    ps.insert(p3);
    ps.insert(p1);
    for (const auto &refp : ps)
        std::cout << refp.second << std::endl;
    auto tri = ps.createRefTriangle3(0, 1, 2);
    std::cout << tri << std::endl;
    std::cout << tri.area() << std::endl;
    std::cout << tri.perimeter() << std::endl;
}
/** output
 * {0: [0, 0, 0]}
 * {2: [0, 0, 1]}
 * {4: [1, 0, 0]}
 * {1: [0, 1, 0]}
 * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1]}
 * 0.5
 * 3.41421
 */
```

RefPolygon<Ty>

```cpp
void foo_refpolygon()
{
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
    RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
    RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
    RefPointSet2d rps;
    rps.insert(p2);
    rps.insert(p4);
    rps.insert(p3);
    rps.insert(p1);
    auto polygon = rps.createRefPolygon({0, 1, 2, 4});
    std::cout << polygon << std::endl;
    std::cout << "perimeter : " << polygon.perimeter() << std::endl;
}
/** output
 * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
 * perimeter : 4
 */
```

RefLinestring23<Ty>

```cpp
void foo_reflinestring2()
{
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
    RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
    RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
    RefPointSet2d rps;
    rps.insert(p2);
    rps.insert(p4);
    rps.insert(p3);
    rps.insert(p1);
    auto ls = rps.createRefLineString2({0, 1, 2, 4});
    std::cout << ls << std::endl;
    std::cout << "length : " << ls.length() << std::endl;
}
/** output
 * {0: [0, 0], 1: [1, 0], 2: [1, 1], 4: [0, 1]}
 * length : 3
 */

void foo_reflinestring3()
{
    RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
    RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
    RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
    RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
    RefPointSet3d rps;
    rps.insert(p2);
    rps.insert(p4);
    rps.insert(p3);
    rps.insert(p1);
    auto ls = rps.createRefLineString3({0, 1, 2, 4});
    std::cout << ls << std::endl;
    std::cout << "length : " << ls.length() << std::endl;
}
 /** output
  * {0: [0, 0, 0], 1: [0, 1, 0], 2: [0, 0, 1], 4: [1, 0, 0]}
  * length : 3.82843
  */
```

RefPointSet_WriteRead23<Ty>

```cpp
void foo_refpointset2_write()
{
    RefPoint2d p1(0, RefPoint2d::ary_type{0, 0});
    RefPoint2d p2(1, RefPoint2d::ary_type{1, 0});
    RefPoint2d p3(2, RefPoint2d::ary_type{1, 1});
    RefPoint2d p4(4, RefPoint2d::ary_type{0, 1});
    RefPointSet2d rps;
    rps.insert(p2);
    rps.insert(p4);
    rps.insert(p3);
    rps.insert(p1);
    rps.write("../output/refpointset2.bin");
    rps.clear();
    rps.read("../output/refpointset2.bin");
    for (const auto &[id, refp] : rps)
        std::cout << refp << std::endl;
}
/** output
 * {1: [1, 0]}
 * {4: [0, 1]}
 * {2: [1, 1]}
 * {0: [0, 0]}
 */
void foo_refpointset3_write()
{
    RefPoint3d p1(0, RefPoint3d::ary_type{0, 0, 0});
    RefPoint3d p2(1, RefPoint3d::ary_type{0, 1, 0});
    RefPoint3d p3(2, RefPoint3d::ary_type{0, 0, 1});
    RefPoint3d p4(4, RefPoint3d::ary_type{1, 0, 0});
    RefPointSet3d rps;
    rps.insert(p2);
    rps.insert(p4);
    rps.insert(p3);
    rps.insert(p1);
    rps.write("../output/refpointset3.bin");
    rps.clear();
    rps.read("../output/refpointset3.bin");
    for (const auto &[id, refp] : rps)
        std::cout << refp << std::endl;
}
/** output
 * {1: [0, 1, 0]}
 * {4: [1, 0, 0]}
 * {2: [0, 0, 1]}
 * {0: [0, 0, 0]}
 */
```

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