目录

Chapter 1

3D Boolean Algebra

I'm coming to design a c++ program for Boolean Algebra on Yin sets in $\mathbb{R}^3$

1.1 Translating from Mathematical Concepts to Classes

1.1.1 Point to Class Point

Properties:

1 Coordinates in $\mathbb{R}^3$
$\longrightarrow$ double coord[3]

2 Identity
$\longrightarrow$ int id

3 Contained by some segments
$\longrightarrow$ vector<int> insegment

4 Contained by a Yin set
$\longrightarrow$ int inYinset

Operators:

1 Return coordinates
$\longrightarrow$ double operator[](const int)

2 Computing vector between two points
$\longrightarrow$ Direction operator-(const Point)

3 Get a point obtained by displacement in the direction of vector
$\longrightarrow$ Point operator+(const Direction)

4 Determining the order relation and equivalence relation of every points
$\longrightarrow$ bool operator==(const Point)
bool operator<(const Point)
bool operaotr>(const Point)

1.1.2 Vector to Class Direction

Property:

- Represent a Vector
$\longrightarrow$ double coord[3]

Operator:

1 Plus and Minus between vectors
$\longrightarrow$ Direction operator+-(const Direction)

2 Quantitative Product of vector
$\longrightarrow$ Direction operator*/(const double)

3 Dot Product of vector
$\longrightarrow$ double dot(const Direction)

4 Cross Product of vector
$\longrightarrow$ Direction cross(const Direction)

5 Modulus operation
$\longrightarrow$ double norm()

6 Unitization operation
$\longrightarrow$ Direction unit()

1.1.3 Straight Line to Class Line

Property:

- A point in this line
$\longrightarrow$ Point fixpoint

- Direction of the line
$\longrightarrow$ Direction direct

Operator:

- Determining whether two straight lines intersect
  $\longrightarrow$ bool ifintersectionLine(const Line)
- Calculating the intersection of two lines
  $\longrightarrow$ Point intersectionLine(const Line)
- Determining whether contain a point
  $\longrightarrow$ bool ifcontainPoint(const Point)

1.1.4 Plane to Class Flat

Property:

- A point is contained by the plane
  $\longrightarrow$ Point fixpoint
- A normal vector of the plane
  $\longrightarrow$ Direction normaldirect

Operator:

- Used to calculate intersection between Planes
  $\longrightarrow$ bool ifintersectionFlat(const Flat)
  Line intersectionFlat(const Flat)
- Calculate intersection of a straight line and the plane
  $\longrightarrow$ bool ifintersectionLine(const Line)
  Point intersectionLine(const Line)

1.2 UML Class Diagram

1.3 Algorithm Implementation

1.3.1 Class Point

Point::operator[](const int)

契约

input const int

output X,Y,Z-coordinate respectively

precondition 0, 1, 2

postcondition double

算法实现

```
return directly
```

证明

Point::operator-+()

契约

input Two Points (a Point and a Direction)

output A Direction (Point)

precondition non

postcondition Match the relationship between two points and the vector between them.

算法实现

```
for i : 0-2
lhs.coord[i] -(+) rhs.coord[i]
```

证明

Point::operator==><(const Point)

契约

input Two Points and double Tol::t

output Bool value

precondition Two Point p1, p2.

postcondition Satisfy the dictionary order relation of points with the tolerance's value equal Tol::t.

算法实现

```
operator <():
if(coord[2] < q.coord[2] - Tol::t)
        return true;
else if((coord[2] < q.coord[2] + Tol::t)
&& (coord[1] < q.coord[1] - Tol::t))
        return true;
else if((coord[2] < q.coord[2] + Tol::t) &&
(coord[1] < q.coord[1] + Tol::t) &&
(coord[0] < q.coord[0] - Tol::t))
        return true;
else
        return false;
```

证明

1.3.2 Class Direction

Accomplishing vector's +-,Quantitative Product, Dot Product and Cross Product.

1.3.3 Class Line

Line::ifcontainPoint(const Point)

契约

input A Line l and A Point p , Tol::t

output Bool value

precondition non

postcondition set the smallest distance d from p to l ,return d < Tol::t.

算法实现

```
Direction d1 = l.drect.unit();
Direction d2 = p - l.fixpoint;
double d = d1.cross(d2).norm();
return d < Tol::t
```

证明 |d1| = 1 and θ is the angle between d1 and d2
d1.cross(d2) = |d1| * |d2| * sinθ
= |d2| * sinθ
= d

Line::(if)intersectionLine(const Line)

契约

input Two Lines l1, l2, Tol:t

output Bool or the intersection

precondition Take intersectionLine() if and only if ifintersectionLine() return true.

postcondition return false when l1 parallel with l2 or return the intersection of l1 and l2.

算法实现

```
Direction d1 = l1.drect, d2 = l2.drect;
Direction d3 = d1.cross(d2).unit();
ifintersectionLine() :
return
fabs(d3.dot(l1.fixpoint - l2.fixpoint))
< Tol::t;

Direction d4 = d1.cross(d3);
Flat f(l1.fixpoint, d4);
intersectionLine() :
return f.intersectionLine(l2);
```

证明 d3 $\perp$ d1 and d3 $\perp$ d2. And |d3.unit()| = 1

So d3 dot product with l1.fixpoint - l2.fixpoint value is the smallest distance between l1 and l2.

The smallest distance vector d0 must be perpendicular to d1 and d2.

So d0 has same direct with d3. and d0 must intersect with l1,get d0 in the plane f.

d0 must intersect with l2, then d0 contain the point p get from f intersect l2.

Choose p as intersection of l1 and l2. return.

1.3.4 Class Flat

Flat::(if)intersectionLine()

契约

input A Flat f and A Line l, Tol::t

output Bool or intersection.

precondition Take intersectionLine() if and only if ifintersectionLine() return true.

postcondition While parallel return false , or return true and the intersection point.

算法实现

```
Direction d1 = f.normaldirect,
          d2 = l.direct;
ifintersectionLine() :
return fabs(d1.dot(d2)) > Tol::t;

calculate intersection is
"using Cramer's Rule to
solve ternary equations."
```

证明 点乘判断直线方向和平面法向量之间夹角是否接近 90 度. 直线和平面相交是三元方程组, 若有解有唯一解.

Flat::(if)intersectionFlat(const Flat)

契约

input Two Flat f1, f2. Tol::t

output False or true and a Line.

precondition Take intersectionFlat() if and only if ifintersectionFlat() return true.

postcondition While parallel return false , or return true and the intersection Line.

算法实现

```
Direction d1 = f1.normaldirect,
                    d2 = f2.normaldirect;
Direction d3 = d1.cross(d2).unit();
ifintersectionFlat() :
return d1.cross(d2).norm() < Tol::t

"assumpting d3[0] > d3[1] and
d3[0] > d3[2].
add a equation x = 1,
using Cramer's Rule to
solve ternary equations."
```