

Chapter 1

类成员简介

1.1 Yinset

1.1.1 成员数据

`vector<GluingClosedSurface> vecGCS`

这里存放了构成殷集边界的有向黏合紧曲面

`vector<HasseNode> Hasse`

存放了有向黏合紧曲面之间的包含关系

1.1.2 成员函数

`Yinset meet(const Yinset&) const`

实现了两个殷集的求交

`Yinset join(const Yinset&) const`

实现了两个殷集的求并

`Yinset complement() const`

实现了殷集求补集

`buildHasse()`

计算黏合紧曲面的包含关系，输出 Hasse 图

1.2 GluingClosedSurface

表示一个黏合紧曲面

1.2.1 成员数据

`vector<Triangle> vecTriangle`

存放了黏合紧曲面的三角剖分

`bool orientation`

存放了黏合紧曲面的方向

1.3 SurfacePatch

表示切割后的曲面片

1.3.1 成员数据

`vector<Triangle> vecTriangle`

存放了曲面片的三角剖分

`vector<pair<Segment> boundary`

存放了曲面片的边界

1.3.2 成员函数

`reverse()`

将曲面片反向，也就是将所有三角形的顶点顺序取反

1.4 PrePaste

实现了将曲面沿闭合交线切割成曲面片的过程

1.4.1 成员数据

`vector<GluingClosedSurface> vecGCS`

存放了不需要进行切割的黏合紧曲面

`vector<SurfacePatch> vecSP`

存放了切割以后得到的曲面片

1.4.2 成员函数

`operator()(const vector<Triangle>&)`

将一个股集中所有三角形放在一起作为输入，将这些三角形黏合起来，直到遇到边界，这个过程等效于将黏合紧曲面沿交线进行切割。

1.5 Paste

实现了将曲面片沿边界黏合成黏合紧曲面的过程

1.5.1 成员函数

`vector<GluingClosedSurface> operator()(const vector<SurfacePatch>&)`

将输入的曲面片沿边界黏合成黏合紧曲面并输出

1.6 Locate

1.6.1 成员函数

`bool operator()(const Point&, const GluingClosedSurface&)`

判断一个点是否在一个黏合紧曲面的有界补集内部

1.7 TriangleIntersect

1.7.1 成员数据

`vector<pair<vector<Segment>, vector<vector<Triangle>::iterator>>> : resultA, resultB`

存放了两个股集的所有三角形之间相交的信息和重合的信息

1.7.2 成员函数

`operator()(const Triangle&, const Triangle&)`

实现两个三角形的求交

`vector<Triangle> collapse()` 将所有三角形根据相交信息进行三角剖分

1.8 Triangulate

1.8.1 成员函数

`bool operator()(const Triangle&, const vector<Segment>&)` 将输入的三角形根据交线进行三角剖分

1.9 Triangle

实现了三角剖分所需的三角形

1.9.1 成员数据

`vector<Point> vecPoint`

存放了三角形三个顶点，顶点顺序与定向有关

`pairt<int,int> InFace` 记录在哪一个曲面中

1.9.2 成员函数

`Triangle<2> project(int n)`

将三维空间三角形投影到某个坐标平面

`intersect(const Line&)`

实现空间中三角形与一条直线求交

`intersectCoplane(const Line<2>&)` 实现平面中三角形和直线求交

`Triangle reverse()` 将三角形顶点顺序反向

1.10 Plane

表示三角形所在平面

1.10.1 成员数据

Real para[Dim+1]

存放了平面方程的四个参数

1.10.2 成员函数

Real angle(const Plane&)

求两个平面的夹角

Line intersect(const Plane&

实现两个平面求交，输出交的直线

1.11 Line

表示一条直线

1.11.1 成员数据

Point fixPoint

存放了直线上一点坐标

Vec direct

存放了直线的方向向量

1.11.2 成员函数

Line<2> project(int n)

将空间中直线投影到某个坐标平面

1.12 Edge

表示一条线段

1.12.1 成员数据

Point endPoint[2]

表示线段的两个端点

1.12.2 成员函数

Edge<2> project(int n)

将空间中线段投影到某个坐标平面

1.13 Segment

表示一条交线

1.13.1 成员数据

Point endPoint[2]

表示交线的两个端点

vector<Triangle>

存放了交线对应的两个三角形

1.14 Point

表示空间中一个点

1.14.1 成员数据

Real coord[Dim]

表示点的坐标

1.15 Vec

表示一个向量

1.15.1 成员数据

Real p[Dim]

表示向量的各个分量

1.15.2 成员函数

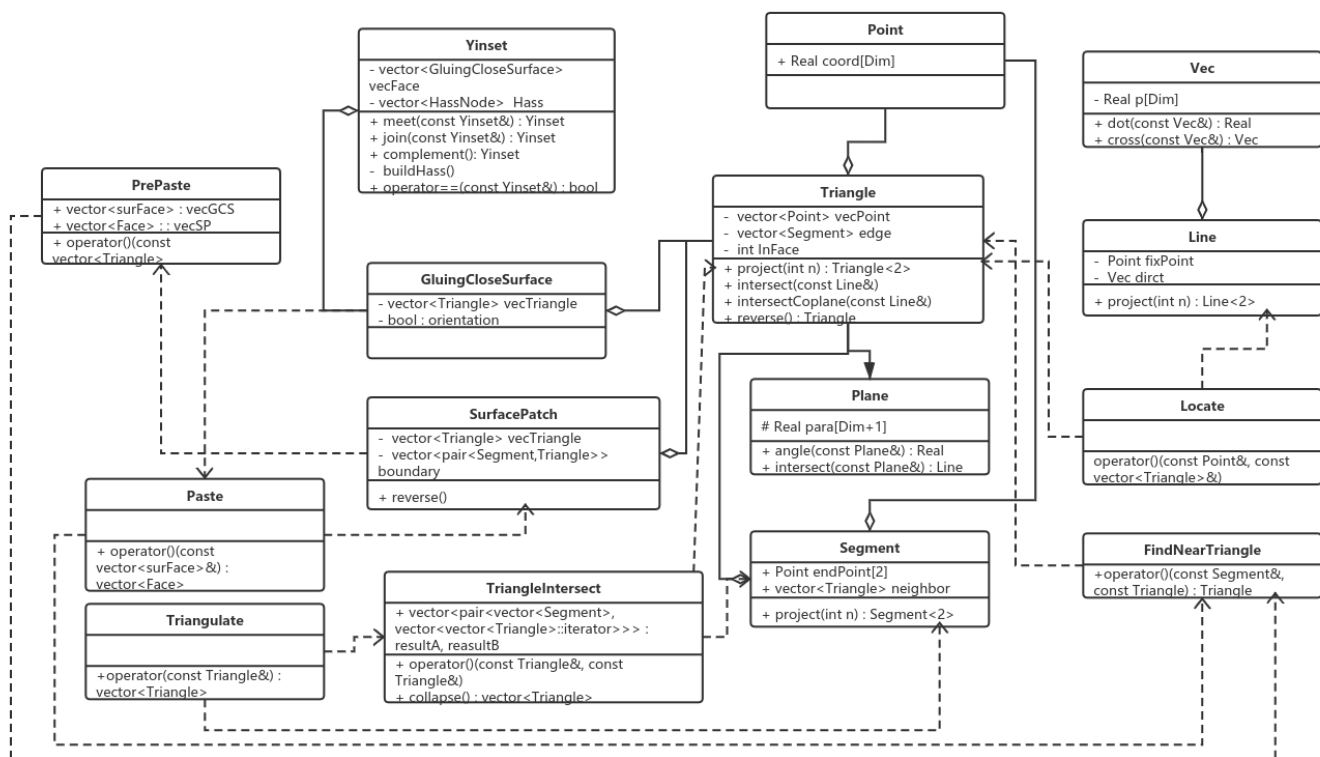
Real dot(const Vec&)

实现向量点乘

Real cross(const Vec&)

实现向量叉乘

图 1.1: UML 类图



Chapter 2

算法伪代码

2.1 算法证明

2.1.1 算法 4: 寻找恰当的粘合三角形

在确定恰当粘合三角形中, 为了满足 DEFINITION 2.11 中的好配对. 选取两个相邻的广义扇形配对, 在程序中即为两个相邻三角形.

这里定义相邻, 两个三角形 a, b 公共一条边 e 且 a, b 在 e 上的方向相反. a 沿着边 e , 以 a 的法向量方向反方向旋转, 与 a 共面的满足之上条件的所有三角形中 b 是第一个, 则称 a, b 相邻, 易见 b 到 a 的旋转也满足条件, 所以以相邻配对是唯一的. 又结合 Yinset 边界是闭曲面沿 1 维 CW 复形粘合结果, 因此相邻配对是存在的.

2-7 行得到两个共边三角形的法向量 $normtri, normneightri$. 不妨设为 A, B ,

$$A * B = \|A\| \cdot \|B\| \cdot \cos(\alpha) \quad (2.1)$$

$$A \times B = \|A\| \cdot \|B\| \cdot \sin(\alpha) \quad (2.2)$$

$$\alpha = atan2(\sin(\alpha), \cos(\alpha)) = atan2(A * B, A \times B) \quad (2.3)$$

因为 $atan2$ 不能得到大于 π 的角度, 通过 8-12 行得到 A, B 在 $(0, 2\pi)$ 范围的夹角.

此时有两种情况如下图.

容易得到第一种情况平面之间的夹角 $\beta = \pi - \alpha$, 另一种情况 $\beta = 3\pi - \alpha$. 5-21 行输出与三角形 tri 以 seg 为公共边的所有三角形计算所需旋转的角度 $angle$ 并输出最小 $angle$ 的三角形, 即相邻的三角形.

图 2.1: 当法线之间角度小于 π 时和大于 π 的两种情况



2.1.2 算法 10: 判断点与 Yinset 之间的位置关系

已知 Yinset 的边界为有限个闭曲面的粘合. 所以 Yinset 边界将全空间分为有限个连通区域. 其中部分为 Yinset 内部, 部分为外部. 即只需判断点 p 在 Yinset 边界上, 或被某个连通区域 G 包含, 连通区域 G 在 Yinset 内部或者外部.

判断连通区域在 Yinset 内部或外部可以通过 Yinset 的边界包含的 G 的边界. 计算过程中边界是空间三角形的集合.

通过点 p 发出一条射线 r , 与表示 Yinset 的边界的所有三角形求交, 选取与 p 点最接近的交点 q (算法 3-4 行) 若点 p, q 重合显然 p 在 Yinset 边界上 (算法 5-8 行). 当 p, q 不重合时, 由最接近性质, 线段 pq (除 q 点) 与 Yinset 边界无交点, 又因为 Yinset 边界将全空间划分为有限个连通区域, 线段 pq (除 q 点) 内部是连通的, 所以 pq 属于同一个连通区域, 而 p 属于连通区域 G , 所以 pq 属于 G , 因此 q 在 G 的边界上.

当 tri 是 G 的边界时, 通过 $tri(G$ 的边界) 的法向量与 pq 的位置可以得到连通区域 G 是 Yinset 的内部或外部. (程序 27 行), 即 q 与 Yinset 的位置关系. (仅有当 pq 与 tri 平行时无法判断, 当包含 q 点的所有三角形与 pq 平行同时也包含 q , 此时 pq 内部必有与这些三角形的重合点, 与 q 为最近交点矛盾)

10-11 行, 包含 q 点的三角形只有 tri 时, 因为 tri 包含 G 中在 pq 上的某些点序列的极限点, 所以 tri 只能是 G 的边界.

在程序 13-34 行中处理 pq 交三角形数量大于 1 时. 对每个包含点 q 的三角形 ta , 取三角形内的线段 dq , 满足角度 $\angle pqd_{ta}$ 最小, 计算所有包含点 q 的三角形, 得到使得 $\angle pqd_{tri}$ 最小的所有三角形集合 $tris(\angle pqd_{tri} < \pi$, 或者 tri 的三条边不能构成面积非 0 的三角形). 计算 $tris$ 中的三角形 tri 满足 $\triangle pqd_{tri}, tri$ 沿着 qd_{tri} 之间夹角最小. 下证 tri 为连通区域 G 的边界

取 tri 和 pq 内部上充分接近 q 点的两点 a, b , 使得 $\angle bqa$ 充分接近 $\angle pqd_{tri}$. 若线段 ab 内部与 Yinset 边界无交点, 折线 pba 连通且只与 Yinset 边界交于 a, p . pba 属于 G 且包含极限逼近 tri 内部点 a 的点序列, (所有表示 Yinset 边界的三角形只有 tri 包含点 j) 所以 tri 是 G 的边界. 若 ab 内部与 Yinset 边界中的三角形 $tri1$ 相交与点 c , 显然有 $\angle pqa > \angle pqc$. 因为 a, b 都充分接近点 q , 因此 $tri1$ 与 q 充分接近即 $tri1$ 包含 q . 因此 q 必在 tri 与 $tri1$ 的边上, 若 d_{tri} 在 tri 的内部, 不妨令 a 在线段 qd_{tri} 上, 此时 $\angle pqd_{tri} = \angle aqb > \angle aqc \geq \angle pqd_{tri1}$, 与 $\angle pqd_{tri}$ 矛盾. 所以 d_{tri} 在 tri 的边上. 令 a 充分接近线段 qd_{tri} 可得 $tri1$ 必包含 qd_{tri} 中的一段, 取 $tri1$ 在 qd_{tri} 上一点 e , 由定义有 $\angle pqd_{tri} = \angle pqe \geq \angle pqd_{tri1}$ 因 $\angle pqd_{tri}$ 是选出的最小的 $\angle pqd_{tri} \leq \angle qpd_{tri1}$, 结合可得 $\angle tri1 = \angle tri$, 由对称性, 不妨令 $d_{tri1} = e$. $\angle aqb$ 投影到垂直于 qd_{tri} 的平面上, 由 tri 和 $tri1$ 都包含了 qd_{tri} 的一部分有内部点 a, c 必不与 q 的投影重合, 而 $\angle pq_{tri} < \pi$ 知 p 与 q 不重合. 综上 $\angle pqa, \angle pqc$ 的投影到 $\triangle pqd_{tri}, \triangle pqd_{tri1}$, $tri, tri1$ 的公共边的垂直平面上, 角度分别等于三角形沿着 qd_{tri} 的夹角, 而投影不改变角度之间的大小关系, 所以 $\angle pqa > \angle pqc$ 即为 $\triangle pqd_{tri1}, tri1$ 沿着 qd_{tri1} 之间的夹角更小, 与 tri 的取法矛盾. 综上所述, tri 是 G 的边界.

算法 1 Yinset 求交算法

Input : *Yinset yinsetA, yinsetB.*

Precondition : *yinsetA, yinsetB* 的边界曲面已经完成三角划分

Output : *Yinset yinsetC*

Postcondition : *yinsetC* 是 *yinsetA, yinsetB* 的交集, 且边界曲面完成三角划分.

```
1: function MEET(yinsetA, yinsetB)
2:   Let TriangleA, TriangleB be vector of Triangle,
3:   FaceA, FaceB be vector of SurfacePatch. closeFaceC be vector of GluingCloseSurface.
4:   TriangleA, TriangleB, IntersectInfo  $\leftarrow$  Collapse(yinsetA, yinsetB).
5:   TriangleIntersect(TriangleA, TriangleB, IntersectInfo).
6:   Let TriangulA, TriangulB store Triangulation result.
7:   EdgeInfo  $\leftarrow$  Triangulate(IntersectInfo, TriangleA, TriangleB, TriangulA, TriangulB).
8:   vecTriA, vecTriB  $\leftarrow$ 
   RemoveOverlap(IntersectInfo, TriangleA, TriangleB, TriangulA, TriangulB, EdgeInfo).
9:   FaceA  $\leftarrow$  PrePast(vecTriA, EdgeInfo.A), FaceB  $\leftarrow$  PrePast(vecTriB, EdgeInfo.B).
10:  Locate(FaceA, TriangleB), Locate(FaceB, TriangleA).
11:  closeFaceC  $\leftarrow$  Past(FaceA + FaceB).
12:  yinsetC  $\leftarrow$  Yinset(closeFaceC).
13:  yinsetC.buildHass().
14:  return yinsetC.
15: end function
16:
17: function COLLAPSE(A, B)
18:   TriangleA  $\leftarrow$  0, TriangleB  $\leftarrow$  0 .
19:   for Face fa  $\in$  A.vecFace, fb  $\in$  B.vecFace do
20:     for Triangle tria  $\in$  fa, trib  $\in$  fb do
21:       TriangleA[end]  $\leftarrow$  tria.
22:       IntersectInfo.resultA[tria]  $\leftarrow$  tria'sedges.
23:       TriangleB[end]  $\leftarrow$  trib.
24:       IntersectInfo.resultB[trib]  $\leftarrow$  trib'sedges.
25:     end for
26:   end for
27:   return TriangleA, TriangleB.
28: end function
```

算法 2 三维三角形求交算法

Input : $vector < Triangle > TriangleA, TriangleB$.

Precondition : None

Output : $vector < pair < vector < Segment >, vector < vector < Triangle > :: iterator >>> resultA, resultB$

Postcondition : $resultA$ 对应于 $TriangleA$ 中相同下标的 $tria$, $it.first$ 是 $tria$ 与 $TriangleB$ 中所有 $Triangle$ 的交线段的集合, 当 $Triangle$ 之间重合时, 输出边与另一个 $Triangle$ 的交. $it.second$ 是 $TriangleB$ 中与 $tria$ 有重合部分的 $Triangle$ 的集合.

```
1: function TRIANGLEINTERSECT( $TriangleA, TriangleB$ )
2:   Let  $resultA, resultB$  be empty vector.
3:   for  $tria \in TriangleA$  do
4:     for  $trib \in TriangleB$  do
5:       SingleCalculate( $tria, trib, resultA, resultB$ ).
6:     end for
7:   end for
8:   return  $resultA, resultB$ .
9: end function

10:
11: function SINGLECALCULATE( $tria, trib, resultA, resultB$ )
12:   Let  $interLine$  be empty vector of  $Line$ .
13:   Get the  $Planepla, plb$  of the  $Triangle tria, trib$ .
14:   if  $pla, plb$  is parallel. then
15:     Insert the lines on which the  $Triangle tria, trib$  edges lie into  $interLine$ .
16:      $resultA[tria].second[end] \leftarrow trib$ .
17:      $resultB[trib].second[end] \leftarrow tria$ .
18:   else
19:     Insert the line which is intersection of  $Plane pla, plb$  into  $interLine$ .
20:   end if
21:   for  $Line l \in interLine$  do
22:      $sega \leftarrow \text{intersectCoplane}(tria, l)$ .
23:      $segb \leftarrow \text{intersectCoplane}(trib, l)$ .
24:     Get the coincidence  $Segment seg$  of  $Segment sega, segb$ .
25:     Set  $seg$ 's neighbour is  $tria, trib$ .
26:     if  $seg$  longer than 0. then
27:        $resultA[tria].first[end] \leftarrow seg$ .
28:        $resultB[trib].first[end] \leftarrow seg$ .
29:     end if
30:   end for
31:   return
32: end function
```

算法 3 共面三角形与直线求交

Input : *Triangle tri, Line l.*

Precondition : *tri* 与 *l* 共面.

Output : *Segment seg.*

Postcondition : *seg* 是 *tri, l* 的交集. 当交集为空时输出默认构造长度为 0 的 *seg*.

```
1: function INTERSECTCOPLANE(tri, l)
2:   According to normal vector of tri, project tri and l to a appropriate plane (xy-,yz- or xz-plane).
3:   Calculate coincident part Segment  $< 2 >$  proseg of projection of tri and l.
4:   Get the primary image Segment  $< 3 >$  seg of proseg in 3D space.
5:   return seg.
6: end function
```

算法 4 寻找恰当的粘合三角形

Input : *Triangle tri, Segment edge.*

Precondition : *edge* 是 *tri* 的边, *edge.neighbor* 指向所有以 *edge* 为边的 *Triangle*, *Triangle* 集合中不存在共面的两个 *Triangle*.

Output : *Triangle neartri.*

Postcondition : 按 *tri* 法向量方向绕 *edge* 反方向旋转, *edge.neighbor* 中最早与 *tri* 重合的 *Triangle*.

```
1: function FINDNEARTRIANGLE(edge, tri)
2:   Assign norm vector of tri to Vec normtri.
3:   Get Direction Vecobserve of edge in tri.
4:   bestangle  $\leftarrow 2 * PI$ , neartri  $\leftarrow 0$ .
5:   for neightri  $\in$  edge.neighbor do
6:     if neightri's direct on edge opposite to tri's then
7:       Assign norm vector of neightri to normneightri.
8:       angle  $\leftarrow atan2(cross(normtri, normneightri).norm(), dot(normtri, normneightri))$ .
9:       if dot(cross(normtri, normneightri), observe)  $< 0$  then
10:        angle  $\leftarrow 2 * PI - angle$ .
11:       end if
12:       if angle  $< PI$  then
13:        angle  $\leftarrow PI - angle$ .
14:       else
15:        angle  $\leftarrow 3 * PI - angle$ .
16:       end if
17:       if angle  $< bestangle$  then
18:        bestangle  $\leftarrow angle$ , neartri  $\leftarrow neightri$ .
19:       end if
20:     end if
21:   end for
22:   return neightri.
23: end function
```

算法 5 三角划分算法

Input : *IntersectInfo*, *vector* \langle *Triangle* \rangle *TriangleA*, *TriangleB*.

Precondition : *IntersectInfo* 中的 *resultA.first* 是 *TriangleA* 相同下标的 *tria* 与 *TriangleB* 中所有 *Triangle* 的交线段和共面 *Triangle* 的集合. *resultA.second* 是与 *tria* 有重合部分的 *trib* \in *TriangleB* *resultB* 同样.

Output : *vector* \langle *vector* \langle *Triangle* \rangle *TriangulA*, *TriangulB*, *vector* \langle *vector* \langle *array* \langle *Segment* \rangle \rangle *EdgeInfo.A*, *EdgeInfo.B*.

Postcondition : *TriangulA* 存储 *TriangleA* 相同下标代表的 *Triangle* 的三角划分的 *Triagnle* 集合. 保证重合部分的三角划分一致.

```
1: function TRIANGULATE(IntersectInfo, TriangleA, TriangleB)
2:   intersectInfoA, intersectInfoB  $\leftarrow$  IntersectInfo.resultA, IntersectInfo.resultB.
3:   for tria  $\in$  TriangleA do
4:     if intersectInfoA.first, intersectInfoB.second is not empty. then
5:       EdgeInfo.A[tria]
          $\leftarrow$  SingleTriangulate(tria, intersectInfoA[tria], TriangulA[tria], IntersectInfoB).
6:     end if
7:   end for
8:   for trib  $\in$  TriangleB do
9:     if intersectInfoB.first, intersectInfoB.second is not empty. then
10:      EdgeInfo.B[trib]  $\leftarrow$  SingleTriangulate(trib, intersectInfoB[trib], TriangulB[trib]).
11:    end if
12:  end for
13:  return TriangulA, TriangulB, EdgeInfo.
14: end function
15:
16: function SINGLETRIANGULATE(tria, IntersectInfoA[tria], TriangulA[tria], IntersectInfoB)
17:   Let newEdge be vector of Segment after Triangulation.
18:   Project all Segment to a appropriate plane.
19:   newEdge  $\leftarrow$  AddSegment(IntersectInfoA[tria])
20:   if newEdge is not trivial And IntersectInfoA[tria]'s Overlap Triangle is not empty then
21:     for trib  $\in$  IntersectInfoA[tria].second do
22:       for seg  $\in$  newEdge do
23:         if trib contain seg then
24:           Add trib into seg's neighbor.
25:           IntersectInfoB.first[end]  $\leftarrow$  seg.
26:         end if
27:       end for
28:     end for
29:   end if
30:   return GeneratePolygon(newEdge, TirangulA[tria]).
31: end function
```

算法 6 AddSegment(*segs*)

Input : *segs*, vector of *Segment* in 2D space.

Precondition : None

Output : *newEdge*, vector of *Segment*.

Postcondition : *newEdge* 将平面划分为一个无界区域和一些三角形.

- 1: **function** ADDSEGMENTS(*segs*)
 - 2: 三角划分算法.
 - 3: **end function**
-

算法 7 消除重合的三角面片

Input : *IntersectInfo*, vector $\langle \text{Triangle} \rangle$ *TriangleA*, *TriangleB*, vector $\langle \text{vector} \langle \text{Triangle} \rangle \rangle$ *TriangulaA*, *TriangulB*.

Precondition : *intersectInfo.resultA.second* 包含 *TriangleA* 中相同下标的 *tria* 重合的所有 *TriangleB* 中的 *Triangle*. *TriangulaA* 一一对应于 *TriangleA* 求交后的三角划分结果. 重合曲面部分的三角划分完全一致, 即重合部分的小三角形顶点位置一致

Output : vector $\langle \text{Triangle} \rangle$ *vecTriA*, *vecTriB*.

Postcondition : *vecTriA*, *vecTriB* 包含三角划分后的结果. 重合部分若法向量一致只保留 *TriangleA* 中的三角划分, 若相反重合部分的三角划分都不保留.

- 1: **function** REMOVEOVERLAP(*IntersectInfo*, *TriangleA*, *TriangleB*, *TriangulaA*, *TriangulB*)
 - 2: Let *OverlapA* represent *IntersectInfo.resultA*.
 - 3: **for** *tria* \in *TriangleA* **do**
 - 4: **if** (*thenOverlap.second* is not empty)
 - 5: **for** *trib* \in *OverlapA.second* **do**
 - 6: **for** *smalltria* \in *TriangulaA* **do**
 - 7: Find *smalltrib* in *TriangulB* success *smalltrib* == *smalltria*.
 - 8: **if** *tria*, *trib* have contrary normal direc **then**
 - 9: *TriangulaA*[*tria*].erase(*smalltria*).
 - 10: *IntersectInfo.resultA*[*tr*]
 - 11: **end if**
 - 12: *TriangulB*[*trib*].erase(*smalltrib*).
 - 13: **end for**
 - 14: **end for**
 - 15: **end if**
 - 16: **end for**
 - 17: Insert all *Triangle* in *TriangulaA*, *TriangulB* to *vecTriA*, *vecTrib* respectively.
 - 18: Correct *EdgeInfo*'s order respect *vecTriA*, *vecTriB*.
 - 19: **return** *vecTriA*, *vecTriB*.
 - 20: **end function**
-

算法 8 预粘合算法

Input : $vector < Triangle > TriangleA, vector < array < Segment > > EdgeInfo.$

Precondition : $\forall tria \in TriangleA, tria$ 的边 $EdgeInfo[tria]$ 中的 $seg.neighbor$ 存储有所有以 seg 为边的 $Triangle$. 并且 $tria.yin$ 记录求交算法前属于的 $yinset$.

Output : $FaceA$ 是 $GluingsCloseSurface$ 和 $SurfacePatch$ 的集合.

Postcondition : $FaceA$ 中的 $GluingsCloseSurface$ 等于 $yinsetA$ 的 $vecFace$ 中与 $yinsetB$ 中的面无交集的子集集合. 另一部分 $SurfacePatch$ 是由另一部分 $yinsetA.vecFace$ 子集沿与 $yinsetB$ 表面的交进行切割后的曲面片, 即曲面片的边界 $boundary$ 是 $yinsetA, yinsetB$ 的表面的交线. (曲面重合情况在 **RemoveOverlap** 中移除).

```
1: function PREPAST( $TriangleA$ )
2:   Let  $FaceA.gCSface$  be empty vector of  $GluingsCloseSurface$ ,  $FaceB.SfacePatch$  be empty vector of  $SurfacePatch$ .
3:   Empty set  $F, T$  of  $Triangle$ . Insert every  $Triangle$  in  $TriangleA$  into  $T$ ,
4:   Empty set  $vecF$  of  $Triangle$ , Empty vector  $boundary$  of  $Segment$ .
5:    $F.insert(T.begin()), T.erase(T.begin())$ .
6:   while  $T$  is not empty do
7:      $tria \leftarrow F.begin(), F.erase(tria)$ .
8:      $vecF.insert(tria)$ .
9:     for  $e \in EdgeInfo[tria]$  do
10:       $neartri \leftarrow \mathbf{FindNearTriangle}(e, tria)$ .
11:      if  $neartri.inFace$  equal  $tria.inFace$  then
12:        if  $vecF$  is not contain  $neartri$  then
13:           $F.insert(neartri)$ .
14:           $T.erase(neartri)$ .
15:        end if
16:      else
17:         $boundary.insert(pair(e, tria))$ .
18:      end if
19:    end for
20:    if  $F$  is empty then
21:      if  $boundary$  is empty then
22:         $FaceA.gCSface[end] \leftarrow GluingsCloseSurface(vecF)$ .
23:      else
24:         $FaceA.SfacePatch[end] \leftarrow getsSurfacePatch(vecF, boundary)$ .
25:        Empty  $boundary$ .
26:      end if
27:      Empty  $vecF$ .
28:      if  $T$  is empty then
29:         $Break$ .
30:      end if
31:       $F.insert(T.begin()), T.erase(T.begin())$ .
32:    end if
33:  end while
34: end function
```

算法 9 选取求交后 Yinset 表面上的曲面

Input : *GluingCloseSurface* or *SurfacePatch FaceA*, *vector < Triangle > TriangleB*.

Precondition : *TriangleB* 包含 *YinsetyinsetB* 的表面上所有 *TriangleB*.

Output : *FaceA*

Postcondition : 保留 *FaceA* 中在 *Yinset yinsetB* 内的曲面片.

```
1: function LOCATE(FaceA, TriangleB)
2:   Name vector < Point > vpa, array < char > vb.
3:   for fa  $\in$  FaceA do
4:     Select Triangle tria  $\in$  fa.
5:     Get center of gravity pa of tria.
6:     vpa[end]  $\leftarrow$  pa.
7:   end for
8:   vb  $\leftarrow$  LocatePoint(vp, TriangleB)
9:   for b  $\in$  vb do
10:    if b is 'o' then
11:      FaceA.erase(fa).
12:    end if
13:  end for
14: end function
```

算法 10 判断点与 Yinset 之间的位置关系

Input : $vector < Point > vpa, vector < Triangle > TriangleB$.

Precondition : $TriangleB$ 包含 $Yinset$ $YinsetB$ 的表面上所有 $TriangleB$.

Output : $string vb$

Postcondition : 当 vb 使用 'i', 'o', 'b' 分别标记 vpa 中的点在 $YinsetB$ 的内部, 外部和边界上.

```
1: function DETACHALONGBOUNDARY( $vecF$ )
2:   for  $p \in vpa$  do
3:     Get arbitrary  $Linell$  cross  $p$ .
4:     Intersect  $l$  with every  $Triangle$  in  $TriangleB$ , Select the closest intersection  $q$ .
5:     if  $q == p$  then
6:        $vb[p] \leftarrow 'b'$ .
7:       Break.
8:     end if
9:      $vqp \leftarrow (p - q) / \|p - q\|, normtri \leftarrow 0$ .
10:    if  $vecTri$  contains only one  $Triangle tri$  then
11:       $normtri \leftarrow$  normal vector of  $tri$ .
12:    else
13:       $smallangle = 1$ 
14:      for  $tri$  In  $vecTri$  do
15:        Project segment  $pq$  into the Plane that contains  $tri$  get  $pjpg$ .
16:        if  $pjpg \cap tri - q$  is not empty then
17:          Let  $Point d$  be arbitrary point in  $pjpg - q$ .
18:           $angle \leftarrow \angle pqd$ .
19:        else
20:          Let  $Point a, b, c$  be vertex of  $tri$  except coincident with  $q$ .
21:           $(angle, d) \leftarrow \min((\angle pqa, a), (\angle pqb, b), (\angle pqc, c))$ .
22:        end if
23:        if  $angle == smallangle$  then
24:          Triangle vector  $tris$  add  $tri$  and  $segs$  add  $qd$ .
25:        end if
26:         $(smallangle, tris) \leftarrow angle < smallangle ? (angle, tri), : (smallangle, tris)$ .
27:      end for
28:      if  $tris$  contains only one  $Triangle tri$  then
29:         $normtri \leftarrow$  normal vector of  $tri$ .
30:      else
31:        Get  $tri$  in  $tris$  such that angle between  $tri$  and  $pqd$  is smallest.
32:         $normtri \leftarrow$  normal vector of  $tri$ .
33:      end if
34:    end if
35:     $vb[p] \leftarrow dot(vqp, normtri) ? 'o' : 'i'$ .
36:  end for
37:  return  $vb$ .
38: end function
```

算法 11 曲面片之间的粘合算法

Input : *GluingCloseSurface* 和 *SurfacePatch* 的集合 *vecF*.

Precondition : *vecF* 中的曲面片, 交且仅交于 *boundary* 上.

Output : *GluingCloseSurface closeFaceC*.

Postcondition : *closeFaceC* 中的所有面都是有向曲面, 自相交区域不包含闭合曲线, 且两两之间的交集是 1 维 CW 复形.

```
1: function PAST(vecF)
2:   Set set  $\langle int \rangle$  connectFace, allF, vector  $\langle int \rangle$  pastFace.
3:   Empty set  $\langle Segment \rangle$  pastBoundary.
4:   Insert all index of Face in vecF into allF.
5:   connectFace.insert(allF.begin()), allF.erase(allF.begin()).
6:   while allF is not empty Or connectF is not empty do
7:     f  $\leftarrow$  vecF[connectF.begin()], connectF.erase(f).
8:     pastFace.insert(f).
9:     for seg  $\in$  f.neighbor do
10:      Get Triangle tri in f and contain seg.
11:      neartri  $\leftarrow$  FindNearTriangle(seg, tri).
12:      if neartri.inFace not in pastFace then
13:        connectFace.insert(neartri.inFace).
14:        allF.erase(neartri.inFace)
15:      end if
16:    end for
17:    if connectF is empty then
18:      closeFaceC[end]  $\leftarrow$  SurfacePatch(vecF[pastFace]).
19:      Empty pastFace
20:      if allF is empty then
21:        Break.
22:      end if
23:      connectFace.insert(allF.begin()), allF.erase(allF.begin()).
24:    end if
25:  end while
26:  return closeFaceC.
27: end function
```
