CDT 算法及其实现和研究

王德军 995568　　孙剑 995570　　鲁爱东 994991

[二维部分原理]:

执笔：王德军

首先，介绍几个定义和概念

1，平面图：v 是有限点集（R2 空间）A 是由 v 确定的一系列的边集合

于是，G=(V,A)满足：

。任意 ab 为 A 中元素，ab 上没有 V 中的点。

。任意两条边都不相交。

2，三角化：一个三角化是指一个平面图，且使其边数最大。

3，Delaunay Trianglulation

一个三角化，如果，其所有的边都满足空圆性质（对于所有 V 中的点）

而空圆性质是指：存在关于 a,b 的一个圆（a,b 在圆上），且使其他所有的点都在这个圆之外。

4，visiblity(可视)

对于图 G=(V,A)中的两个点 a,b,他们之间是可视的是指：ab 线段不会与边集合中的任何一条边线段相交。

5，CDT,(Constrained Delaunay Triangulation)

对与图 G(V,A)是一个平面图，而且，A 非空，于是称 B=(S,AUA')为一个 CDT (关于 G), 是指 A'中的任意一条边 ab 满足以下条件，1，a 与 b 是可视的，2，对于与 a,b 均可视的点集，ab 这条线段是满足空圆性质的。

我们得到的算法将是递增的，也就是说对于某个图，我们将是依次增加一些点，或一些边，来得到 CDT.

当然，对于原始的边集合 A,（是用来限定三角化的）被称为不动边，当我们在逐次地递增点或者边时，这些不动边始终保持不变。

对于动态的三角剖分当中，很明显，我们这个算法的优势就体现了出来，那就是，它非常容易实现，而且运行很有效。

下面我们将讲述这个算法的理论框架，以及具体的实现（加点，加边）以及其中可能碰到的问题及其解决。

首先我们介绍关于点插入的算法：

命题一：

令 A 为一些给定的点组成的点集所形成的 Delaunay Triangulation.那么，当我们在这个点集所构成的区域的内部，插入一个点 p,且 p 不属于原来的哪个点集，这将形成一个新的 Delaunay Triangulation,SU{p},需要做出修正的仅仅是修正原来哪个 Delaunay Triangulation 的三角形（且 p 位于其外接圆的内部）；

这个命题本身其实非常自然，因为对于那些，其外接圆内含有新增点的三角形，是必须要进行修正的，否则，便不满足空圆性质，那么，我们又可以想象，对于原来那些，本来其外接圆就不包含新增点的，其对与着个新增点以及以前的点都满足空圆性质，因而，很自然

地，我们只需对与那些其外接圆内部含有新增点的三角形做出改动。

上面这个命题是针对 DT,那么，很自然地，我们也可以根据 CDT 的性质，得到关于 CDT 的命题二：

令 A 为一些给定的点组成的点集所形成的 CDT,那么，当我们在这个点集所构成的区域的内部，插入一个点 p,且 p 不属于原来的哪个点集，这将形成一个新的 CDT,SU{p},需要做出修正的仅仅是原来那个 CDT 的三角形，该三角形满足下列两个条件：

1,p 在它的外接圆内部

2，这个三角形的三个顶点关于 p 都是可视的。

事实上，这个命题也是很自然的，根据 CDT 的定义本身。可以看出，CDT 的命题二与命题一有着很相似之处。当然，有一点也是很重要的，必须注意的是，对于一开始就限定了整个 CDT 的那些 edges ,我们在操作过程中，是不能动的，它们是不动边。在我们的程序中，我们对于每条边都设定了一个 fix 变量，用以标记其是否为 fixed 的。

下面，在每个插入点都在原来那个 CDT 的区域内部的前提之下，我们做出了下面这个伪代码程序。

```
Procedure AddPointCDT(T:CDT,p:Vertex)
stack:=EmptyStack;
Find the triangle t that contains p
Divide t in three triangles,t1,t2 and t3 Depending on p
Push(stack,t1)
Push(stack,t2)
Push(stack, t3)
While notempty(stack) do
    T:=Pop(stack)
    Topo:=OpposedTriangle(t,p)
    If the edge shared by t and topo is not fixed and P 在 Circumcircle(topo)
      Then Swap the edge shared by t and topo
      Push (stack,t)
      Push(stack,topo)
    Endif
Endwhile
EndProc
```

其中， 把 t 分成三个部分，是指当 t 在该三角形内部时，将 t 分别和三角形的三个顶点相连，从而，得到了三个新的三角形，当然，此时我们的旧三角形就被删掉，事实上，我们在处理这个问题的时候是这样处理的，对于每个三角形，都设立有一个变量用来表示这个三角形是否合法，于是，此时，我们对于这个旧三角形就称其为不合法。

而对于 opposed triangle,先给定该三角形 t 的一个顶点 p ,得到了该三角形的一个领接三角形，且这个三角形与原三角形有一个公共的边，有一个顶点与原来的三角形顶点不一致。在我们的程序当中，我们是在每个 triangle 中有三个变量来记录其领接的三角形的序号（我们的三角形的存储是给每个三角形进行编号来进行存储的。这一点对于我们求 opposed triangle 来讲，是取得了方便的。（而不用去进行遍历来得到 opposed triangle,浪费时间）

其中关于判断一个点是否在另外一个三角形的外接圆内部是很容易的，只要先求得那个三角形的圆心，再判断另一个点与这个圆心的距离是否小于外接圆的半径即可。

对于 swap 函数，这个操作则是指将两个领接三角形的公共边先去掉，同时将两个三角形的相异点连接，得到新的边，及其两个新的三角形。

对于这个算法，进行它的复杂度问题，首先，在所有的三角形族中找到一个三角形，使得其内部含有插入点，显然，这个过程最都需要 n 步(n 为所有的三角形的数目)。而对于迭代过程的数目，由于，我们只动那些满足条件的三角形，所以，其最坏的情况也将只是 n,因此，这个算法的复杂度将是 O(n)。

以上，我们提出了一个算法，可以看出，这个算法应该能够解决我们的加点问题，下面我们将通过理论证明来证明它确实是能够解决我们的关于 CDT 的点插入问题的算法。

引理一：

在一个 CDT 中，令 t1=T(a,b,c),且 t2=T(q,b,a) 是两个相邻的三角形而其公共边 ab 不是一不动边，于是，我们有一个结论就是：任意位于 CC(t1)中，且是 T(a,b,c)关于 c 的相对点也在 CC(t2)的内部。

证明如下：

由于 t1 和 t2 都是 CDT 当中的三角形，那么它们必然都满足空圆性质，特别地，由于 ab 并非不动边，那么 q 必然在 T(a,b,c)之外，因为，若不然，在 T(a,b,c)之内的话，则对于，T(a,b,c)就不满足空圆性质，那么就矛盾了。于是，我们知道，ab 把 CC(t1)切成两个部分，一个在 CC(t2)内部，一个在 CC(t2)外部。于是，任意一个在 CC(t1)内，且与 c,关于 ab 相对，必然会在 CC(t2) 内，因为，若不然，那就在不在 CC(t2)内的 CC(t1)内，则将导致 t1 不满足空圆性质。

下面我们给出命题三：

对于我们前面给出的基于 swap 的算法，当我们应用于基于一个图 G={v,A}的 CDT 的插入点问题，（且 p 不在 V 内），这样我们将得到一个新的 CDTG={vU{p},A}.

证明如下：

我们已经由命题二就知道，要求得新的 CDT,我们只需考虑那些包含有 p 的三角形，而且其各个顶点都与 p 是可视的。这些三角形形成了一个区域，且其是被一个星形的多边形 Qp 所包围的，于是，CDT of G’通过连接所有的顶点与 p 来得到。令 t 是 Qp 的一个三角形我们将证明这个算法将通过做一些 swap 操作，来最后结束，从而得到新的 CDT,且保持 Qp 的外部不变。

其实，我们的证明的关键在于是，我们的算法中只用到了当，p 在三角形内部，而我们的命题二所指出的是要动的是那些 p 在其外接圆的三角形，显然，这样的三角形要比那些在三角形本身内部的三角形要多，关键是怎么处理对于点在三角形的外部而且，在外接圆内部的情况下，我们是怎么通过我们的算法来做到的。

当 p 本身就在三角形内部，那么，在我们算法的一开始，就将把这个三角形分成三个部分，于是，t 的顶点都将与 p 相连。

否则，当 p 在 t 外面，于是，我们知道在 t 中，存在唯一的一个顶点，使得该点为 p 的相对点，记为 c,其他两个顶点分别记为 a,b.

对于 t,由于它是 CDT 中的一个三角形，所以其外接圆 CC(a,b,c)内部将不会有任何 V 中的点。不过，不失一般性，将有一系列的边会分隔 c 和 p,当然这些的边都是在原来那个 CDT 当中的，而且，这些边的端点都不会在 CC(a,b,c)的内部。当然，这些边都不可能是不动边，

因为，我们要对于我们要处理的情况来说，c,和 p 是可视的，而一旦这些边有不动边，则，c 和 p 将是不可视的。我们将对于这些 edge 做数量上的归纳。

事实，对于这些 edge 来说，其数量至少为一，因为,ab 就是一条。这样的话，我们的基本情况就是只有一条边，那就是 ab.在这样的情况里，我们可以知道 p 会在 t 的某一个相邻三角形当中，同时将其分为三部分。于是将会因为这个原因而形成了 T(a,b,p)，推入栈中.于是，我们的算法又将会在某个时间里将这个三角形 pop 出来，进行考虑是否需要 swap 边 ab.

下面我们考虑存在有 n 个分隔边的情况，也就是说，T(a,b,c)是原来那个 CDT 中的一个三角形，而且它的三个顶点都是与 p 可视的，而且，p 在 CC(a,b,c)内部，且，T(a,b,c)与 p 被 n 个分隔边所分隔。不失一般性，假定除了 ab 外的下一个分隔边是 aq(当然，bq 的情形可以类似处理)，由引理一，我们可以得到 p 会在 CC(a,b,q)内部。这是因为 t 和 T(q,b,a)是 CDT 中的两个相邻三角形而且，p 是在 CC(a,b,c)中.于是，我们还可以得到这样的结论，那就是，q 将与 p 是可视的，由于 T(a,b,q)是 CDT 中的一个三角形，则 CC(a,b,q)则不会含有任何其他的 V 中的点。

根据以上的归纳假设，由于 T(a,b,q)是与 p 被 n-1 条边分隔的，而且，其所有的顶点都与 p 是可视的。而且，它的外接圆将包含 p,这个算法将在某个时候通过 swap 得到 T(p,b,q)和 T(a,b,p).于是 t 将被推入栈中，我们的算法同样会将考虑 swap 边 ab 来得到 pc.

以上就证明了我们所需证明的关键问题，从而使我们知道我们的算法是能够得到 CDT 的一个算法。

关于加边的算法

一般地，对于在一个已有的 CDT 当中，我们加入一条边一般是遵循以下的步骤

1，得到那些能与 ab 相交的边，并且把它们去掉，当然，此时留下了一个未被三角化的区域。

2，把 ab 这条边加入到结果当中，在我们的程序当中，表现为加入到边数组中。

3，重新三角化 ab 的逆时针部和顺时针部（而且是在第一步中未被三角化的部分）

逆时针部是指，任意一点在逆时针部中，若其与 a 的连线所得到的向量可以通过 b 与 a 所连得到的向量进行逆时针运动所能与之重合（角度范围在（0，pi/2））.

顺时针部是指，任意一点在顺时针部中，若其与 a 的连线所得到的响亮可以通过 b 与 a 所连得到的向量进行顺时针运动所能与之重合（角度范围在（0,pi/2））.

第一步，我们必须找到 CDT 当中的三角形，其中一个顶点为 a,且使其被 ab 切割。要得到一个含有某一个点的三角形，在一个 CDT 当中我们已经可以很容易就得到了，因为我们在我们定义的数据结构当中，已经对每一个点都定义了其领接的三角形的序号族。然后的操作便是进行寻找被切割的边，并去掉之，最有效的方式就是通过一个三角形，根据响应的

边进行寻找其相邻边，一直寻找下去，知道发现，得到的三角形有一个顶点已经是 b 了才结束。

当然其中有一个很关键的函数就是在已经找到了 a 的相邻三角形族的时候，如何找到其中一个三角形使之是有一条边与 ab 是相交的，因为一旦找到了这个三角形，那么我们下面的工作就可以顺理成章了，事实上，我是这样做的，一开始，先根据点找到其相邻的三角形族，同时，根据其序号，按逆时针方向进行遍历，我们可能碰到三种情况，一种就是找到某个三角形其有一个边与 ab 相交，而且交点在 ab 内部，第二种就是，发现该三角形的以 a 为一个端点的边与 ab 重合，且这两条边整个重合，第三种就是，这两条边仍然重合，且这两条边不重合。那么如果是第一种情况，则，我们还按正常的步骤走，如果是第二种情况，我们可以很清楚地看到我们加的这条边是重边，可以当作无意义的边来处理，如果是第三种情况的话，如果，三角形的那条与 ab 重合的边的另一个端点在 ab 之内，设为 c,则我们对于 c 又其邻接三角形，使得其也有边被 cb 切割，如果，c 在 ab 之外，则这种情况是不可能的。因为否则将说明，c 本来就是 CDT 中的，则我们找到的三角形事实上就不可能是我们要找的了。

在我们找到了这个三角形之后，然后我们进行的就是找相邻的三角形，于是，在同时，我们可以进行分类操作，集合{a,s1,…,sn,b}表示 ab 的逆时针集,事实上，他们定义了一个伪多边形，称为 Pu.同时，集合{b,i1,…..in,a}表示 ab 的顺时针集，事实上，他们定义了一个伪多边形，称为 Pl.于是，这两个多边形是需要重新三角化的。

事实上，我们必须注意到，对于这两个多边形，它们并不是传统意义上的多边形，因为，它可能含有一些重复点，这些重复点对于多边形的轮廓是没有贡献的，当然有一点必须明确的是，这些重复点的存在不会说复杂化我们的算法，它只是在当我们在分析新得到的三角形和已经有的三角形之间的关系的时候必须小心，因为那些重复点之间的边在经过我们的算法的时候可能是根本就没有改变。

在关于这两个未三角化的区域的重新三角化的过程，我们用的是递归的方法。这两个部分是分别做的，因为我们得到的新的三角化是不可能存在一条边，一个在顺时针部，一个逆时针部。

首先，我们知道，对于 Pu,肯定会含有一个三角形，且 ab 是它的一条边，又由于，三角形必须是满足空圆性质，所以 T(a,b,c)内部必须没有任何 pu 的顶点。所以，我在算法中采用的是对于 Pu 中的任何一个顶点，计算他们相对与 a,b 的张角，当对于某个点，我们找到了最大的那个值，则取之做为 c,再继续由 ca,cb 来进行递归。

这个递归算法如下：

１：找到 c=sl 使得 T(a,b,c)的外接圆内不含有任何 Pu 中的点。

２:形成了新的三角形 T(a,b,c),把 Pu 分成了两个子区域,Pe={a,s1,…,sl}和 Pd={sl,…sn,b};

3:递归应用算法于 Pe 和 Pd,对于 ac 和 cb.

命题四：

令 B 是图 G={V,A}的 CDT,且令 ab 不在 A 中，a 和 b 都是 V 中的。然后，通过移去被

ab 切割的边，并且加入 ab,并且重新三角化伪多边形 Pu,Pl,则将得到新的 CDT(G={V,AU{ab}};

证明： ab 必然是新的 CDT 的一条边。又由于这些都是 triangulation,于是那些被 ab 切割的边必然都不能存在，因而必须删掉。于是，剩下的边都满足空圆性质才能成为新的 CDT 的边。所以，通过先删掉与 ab 相交的边，然后，针对相对的 Pu,Pl 的伪多边形，来进行重新的三角化，可以得到新的 CDT.

下面的就是如何加边的伪代码：

```
Procedure AddEdgeCDT(T:CDT,ab:Edge)

Pu:=EmptyList

PL:=EmptyLIst

V:=a

While b not in t do

Tseg:=OpposdTriangle(t,v)

Vseg:=OpposedVertex(tseg,t)

If Vseg above the edge ab then

AddList(Pu,Vseg)

V:=Vertex shared by t and tseg above ab

Else

AddList(Pl,vseg)

V:=Vertex shared byt and tseg below ab

Endif

Remove t from t

T:=tseg

Endwhile

TriangulatePseudopolygonDelaunay(Pu,ab,T)

TriangulatePseudopolygonDelauneay(PL,ab,T)

Reconstitute the triangl adjacencies of T
```

Add edge ab to T

Mark the edge ab from T as fixed

Endproc

其中的 opposedtriangle 函数与前面关于点的插入问题中的函数是一样的。Opposedvertex 的自变量值是两个三角形 t,topo,然后根据它们来得到 topo 中的一个顶点 ，而且这个顶点不为 t 的任何一个顶点。

下面是关于重新三角化 PU,Pl 这两个伪三角形的算法我们采用的递归方式，每次寻找相应点都是采用最大角法。

Procedure TriangulatePseudopolygonDelaunay(P:VertexList,ab:Edge,t:CDT)

If P has more than one element then

C:=First Vertex of P

For each vertex v 属于 P do

If v 在 circumcircle(a,b,c)内部，then

C:=v

Endif

Endfor

Divide P into Pe and Pd,giving

P=Pe+{c}+Pd

TriangulatePseudopolygonDelaunay(Pe,ac,t)

TriangulatePseudopolygonDelaunay(Pd,cb,t)

Endif

If P is not empty then

Add triangle with vertices ,a,b,c into T

Endif

Endproc

下面将是我们关于边插入的复杂度分析

令 n 是我们的初始 CDT 的三角形数目为 n,且被 ab 所切割的三角形数为 e.于是，我们一

开始的寻找以 a 为端点并且被 ab 所切割的三角形最坏将需要 O(n)次。因为由我们的数据结构决定，每个点都有它的相邻三角形的序号记录。而这样的寻找次数就是 O(n).剩下的对于 ab 的顺时针部和逆时针部的未被三角化的部分的重新三角化的计算。首先，得到这样一个伪多边形将需要 O(e)次。然后进行迭代用来重新三角化这两个伪多边形将需要 O(e*e)次，这是因为我们在我们的迭代过程每次的迭代用以确定左右边界的总点数目都会要递减一次，而我们最坏的情况是需要迭代 e 次，所以通过计算，应该是不大与 e+(e-1)+(e-2)+....+1,所以应该不会超过 e*e 次，因此，最坏的情况将是 O(e*e)次。所以，addEdge CDT 算法的最坏次数将是 O(n*n).

综上所述，我们知道我们的 CDT 算法是一种递增算法，但是，我们知道我们的算法都要求每次增加点都必须保证这个点就在原来的三角形区域里面，因而，我们的初始 CDT 所构成的区域就必须包含所有的点。我们采用的算法是首先根据我们要做三角化的点来先设计一个 super_triangle,所有的点都在这个 super_triangle 里面，这可以很容易就做到。

（这么做的好处在于非常清楚，可以很方便得就用于我们的算法，但是，我们必须在最后又重新做一个程序来确定从我们所得到的三角形当中，哪些三角形是被显示的，哪些是不被显示的。粗看起来似乎有点麻烦，但是，我们知道很可能在我们的那些点的内部有些是洞，而这些洞也是需要处理三角形是否被显示的问题的，所以说不完全是额外的负担，但是我在想，这一点是不是可以改进，那就是进一步的动态化，即根本就不存在最外面的 super triangle 这么一说，每一次加点，如果它还在原来的区域当中，自然用原来的算法，如果不在的话，那就通过这个点来得到新的三角形，其另外一条边是本来就存在的，它应该是离这个点最近的。再通过 swap 算法来往下做，这个想法是希望进一步的动态化，当然，在这一次的实验当中，我们还是采用的 super triangle 的方法。）

下面就是关于我们最后得到了这么一些三角形如何去处理和显示它们的方法，我们可以想象，很多三角形是在我们原来所希望的区域外面的，而不是希望显示，同时，内部可能还有一些洞，也不是我们所希望显示出来的。我们将采用遍历方法，首先从一个外部的三角形，具体可取为离 super triangle 比较近的三角形开始，求领接三角形，一直遍历所有的三角形，然后我们根据 fixed edge 来确定是否显示的问题。

处于加边的考虑，我们应该先加入内部洞的点和 fix 边，然后再加外部的点和边，这样可以使加边所引起的修正数目尽可能的少。

[二维算法实现]:

执笔：孙剑.

数据结构：

1. 点的数据结构：
   用一个动态数组存储所有的点，每个点包含以下信息：
   a：点坐标 x，y；
   b：用一个动态数组存储包含有该点的三角形列。
2. 线的数据结构：
   用一个动态数组存储所有的边界线，每一条线包含以下信息：

a：起始点和终止点在点数组中的编号；

b：边所属多边形的编号

3．三角形的数据结构：

用一个动态数组存储所有生成的三角形，每一个三角形包含以下信息：

a：用数组存储三个顶点在点数组中的编号， 按逆时针排序；

b：用数组存储与该三角形边相邻三角形在三角形数组中的编号，按逆时针排序， 若没有，则设置为-1；

c：用数组存储该三角形的三条边是否为固定边。

具体算法：

具体的算法主要有两个，一是加点的算法，二是加边的算法。

下面以加点算法为例说明，加边的算法与之类似，主要的问题是在加边和加点过程中要维护上述数据结构。

```
AddPointCDT(CDT T, int PointIndex)
{//T 代表已有的三角形网格和拓扑信息，PointIndex 代表要插入点在点数组的编号
    //判断点在 T 中的位置：
    If (PointIndex 在 T 外部或插入点与已有点重合)
        Return;
    Else if (PointIndex 在 T 中一个三角形内部)
        InitAddPointCDTIn(CDT  T, int Index);
    Else  // PointIndex 在 T 中的一条边上
        InitAddPointCDTSide(CDT T, int Index);
  //  判断新生成的三角形是否满足 Delaunay Trianglulation  的要求
    for(PointIndex 的三角形列中的每一个三角形序号  TIndex)
        Push(Stack, TIndex);
    While( not empty(Stack) )
    {
        TIndex=Pop(Stack);
        TIndexopo=不与 TIndex 共享点 PointIndex 的三角形
        If (TIndexopo!=-1)
        {
            If(TIndex 与 TIndexopo 的共享边不为 Fix 边)
            {
                If(  点在三角形 TIndexopo 的外接圆)
                {
                    //TnewoneIndex, TnewtwoIndex 为两新生成的三角形
                    Swap(TIndex, TIndexopo, &TnewoneIndex, &TnewtwoIndex)
                    Push(Stack, TnewoneIndex)
                    Push(Stack, TnewtwoIndex)
                }
            }
        }
    }
}
```

下面对三个子程序进行说明

```
InitAddPointCDTIn(CDT    T, int TIndex, int PointIndex)
{//T 代表已有的三角形网格和拓扑信息，TIndex 代表包含插入点的三角形的序
//号, PointIndex 代表要插入点在点数组的编号
//如图 2-1,由三角形 TIndex 生成三个三角形
      Generate (T1, PointIndex, p1, p2);
      Generate (T2, PointIndex, p2, p3);
      Generate (T3, PointIndex, p3, p1);
//根据 Tindex 的边相邻单元信息和边 Fix 信息, 设置新生成单元的边相邻单元信息和边
//Fix 信息
      SetBorderFix(T1, TIndex);
      SetBorderFix(T2, TIndex);
      SetBorderFix(T3, TIndex);
//无效三角形 TIndex, 并修改 TIndex 顶点的三角形列信息, 从中删除三角形 Tindex
      SetInvalid(TIndex):
      DeleteTriangleFromPoint(TIndex);
}
```

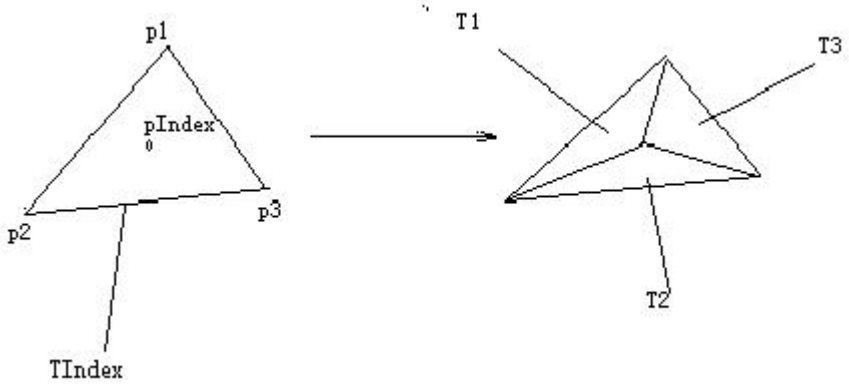

图 2-1

```
InitAddPointCDTSide(CDT    T, int TIndex, int PointIndex)
{//T 代表已有的三角形网格和拓扑信息，TIndex 代表其某边包含插入点的三角形的序
//号, PointIndex 代表要插入点在点数组的编号
//得到与 TIndex 相邻的三角形 TIndexopo, 其共享边包含插入点
      TIndexopo=GetOppositeT(TIndex, PointIndex);
//如图 2-2,由三角形 Tindex 和其对三角形生成四个三角形
      Generate (T1, PointIndex, p4, p1);
      Generate (T2, PointIndex, p1, p2);
      Generate (T3, PointIndex, p2, p3);
      Generate (T4, PointIndex, p3, p4);
//根据 TIndex 和 TIndexopo 的边相邻单元信息和边 Fix 信息, 设置新生成单元的边相邻
单元信息和边//Fix 信息
      SetBorderFix(T1, TIndex);
      SetBorderFix(T2, TIndex);
      SetBorderFix(T3, TIndexopo);
      SetBorderFix(T4, TIndexopo);
```

//无效三角形 TIndex 和 TIndexopo, 并修改 Tindex 顶点的三角形列信息, 从中删除三角形 TIndex

SetInvalid(TIndex):

DeleteTriangleFromPoint(TIndex);

SetInvalid(TIndexopo):

DeleteTriangleFromPoint(TIndexopo);

}

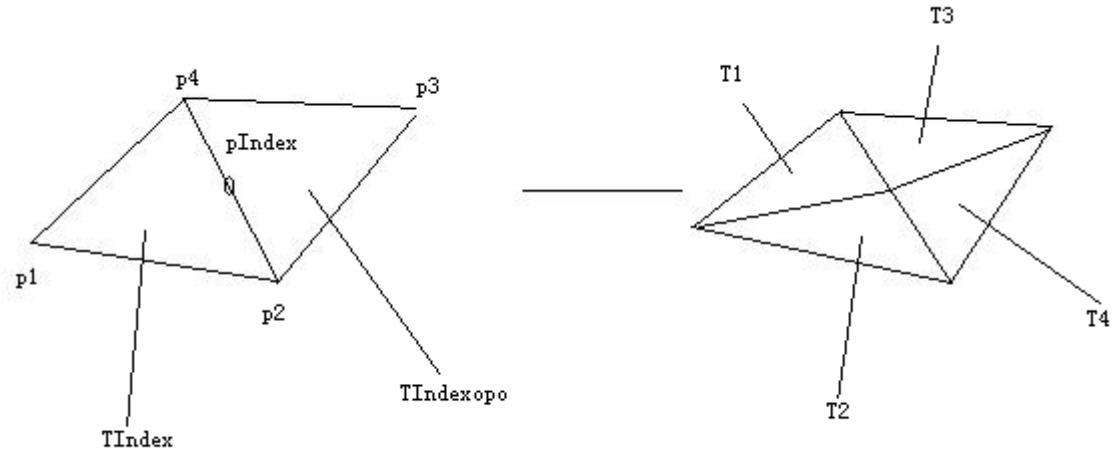

图 2-2

Swap(int TIndex, int TIndexopo, &TnewoneIndex, &TnewtwoIndex)

{// TIndex 和 Tindexopo 代表要 Swap 的三角形，TnewoneIndex， TnewtwoIndex 为新

//生成的三角形的序号

//如图 2-2,由三角形 TIndex 和 TIndexopo，

Generate (TnewoneIndex, p4, p1, p3);

Generate (TnewtwoIndex,p2, p3, p1);

//根据 TIndex 和 TIndexopo 的边相邻单元信息和边 Fix 信息, 设置新生成单元的边相邻单元信息和边//Fix 信息

SetBorderFix(TnewoneIndex, TIndex);

SetBorderFix(TnewoneIndex, TIndex);

SetBorderFix(TnewtwoIndex, TIndexopo);

SetBorderFix(TnewtwoIndex, TIndexopo);

//无效三角形 TIndex 和 TIndexopo, 并修改 TIndex 顶点的三角形列信息, 从中删除三角形 TIndex

SetInvalid(TIndex):

DeleteTriangleFromPoint(TIndex);

SetInvalid(TIndexopo):

DeleteTriangleFromPoint(TIndexopo);

}

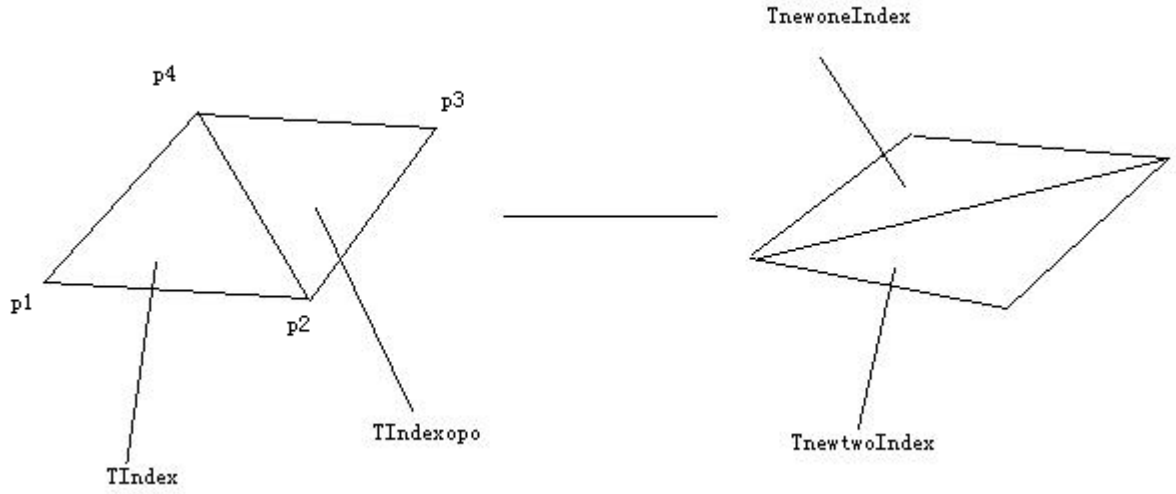

图 2-3

[三维部分]:

执笔：鲁爱东.

这里[2][3]采用了一种三角划分剪裁曲面的新方法. 基本的思想就是完全在参数空间内进行三角划分. 首先, 经过剪裁的曲面被映射到参数空间并近似成为二维的多边形区域, 然后在这个区域中进行严格的 Delaunay 三角划分. 划分出来的三角形将进一步进行分解, 直到所有三角形的边长都小于从允许确定的误差和曲面定义得到的最大边长值. 最后, 所有的三角形再被映射回三维空间. 这个方法比较有效而且较快, 它还可以避免曲面之间产生断层. 这个算法比较适用于生成固体的三角分割模型.

1.算法简述

将剪裁的曲面映射到二维的参数空间, 线性化区域的边界, 从而得到二维的多边形区域. 这个区域包括一个最外层多边形和若干个内层的多边形. 最外层多边形对应于剪裁区域的边界, 内层多边形对应于曲面上的洞. 把整个曲面剪裁成若干曲面片, 每一片都映射得到一个二维的多边形区域.

基本的思想就是把三角划分都放在参数空间进行, 第一步是把曲面剪裁成若干个子面片, 每个子面片的参数空间区域互不相交. 由这些子面片得到一组参数空间的多边形区域. 相邻子面片的信息要用数据结构存储起来. 注意相邻子面片的共同边界都要在各自的空间中进行划分, 所以可能造成同一边界不同化法. 第二步就是在参数空间中三角划分各个子多边形区域, 直到每个三角形的边长都小于计算得到的最大边长. 第三步是把参数空间中三角划分的结果映射回三维空间.

2.创建参数空间中的多边形

定理: $f(t):t\in[a,b]$是C^2曲线, $l(t)$是线段, 并且$l(a)=f(a),l(b)=f(b)$. 则$\sup\|f(t)-l(t)\|\le(b-a)^2\sup\|f''(t)\|/8, a\le t\le b$, 给定允许误差$\varepsilon$, 可得线段划分的数目 n : $n=|b-a|\left(\sup\|f''(t)\|/8\varepsilon\right)^{1/2}$.

注意这一过程是在三维空间中进行的,而不是参数空间. 所以, 由上式得到的误差就是三角划分的误差, 并且同一条曲线的划分是相同的.

划分的结果应该满足下列条件:

- 每条边由两个面共享.
- 每个顶点周围是一圈简单的边和面.(指不相交。)
- 面与面之间只在通常的边与点上相交。

3.计算三角形最大边长 （此项孙剑执笔）

假设曲面的参数表达式为：

$$S(u,v)=\sum_{i,j}a_{ij}*u^i v^j$$

根据[2]最大边长表达式为：

$$\psi=3*\sqrt{(\frac{\varepsilon}{2*(M1+M2+M3)})}$$

其中

$$M1=\sup_{(u,v)}\|\frac{\partial^2 S(u,v)}{\partial u^2}\|$$

$$M2=\sup_{(u,v)}\|\frac{\partial^2 S(u,v)}{\partial u\partial v}\|$$

$$M3=\sup_{(u,v)}\|\frac{\partial^2 S(u,v)}{\partial v^2}\|$$

为了得到*M*1，*M*2，*M*3，先把曲面的多项式形式转换成以车比雪夫基形式：

$$S(u,v)=\sum_{i,j}t_{ij}C_i(u)C_j(v)$$

其中 Ci(u)为 i 次车比雪夫基

根据[3]可以得到 T=$\{t_{ij}\}$和 C=$\{a_{ij}\}$之间的转换矩阵

这样 *M*1，*M*2，*M*3 就可以由以下表达式得到：

$$M1(x) = \frac{4}{3}\sum_{j=0}^{m}\sum_{i=2}^{n}|t_{ij}(x)|\,i^2(i^2-1);$$

$$M2(x) = 4\sum_{j=1}^{m}\sum_{i=1}^{n}|t_{ij}(x)|\,i^2j^2;$$

$$M3(x) = \frac{4}{3}\sum_{j=2}^{m}\sum_{i=0}^{n}|t_{ij}(x)|\,j^2(j^2-1);$$

4.严格的 DELAUNAY 三角划分

一旦参数空间的子多边形区域确定了，就可以分别进行三角划分。三角划分的目的就是划分成边长不超过最大允许边长的三角形. 为了减少生成的三角形的数目，显然要使生成的三角形的边长小于最大允许边长,但是尽可能的接近最大允许边长.(而且使三角形最接近等边三角形.)

应用了严格的 DELAUNAY 三角划分的过程后, 产生了一系列三角形，遍历所有的有效的三角形, 判断每个三角形的边是否满足最大允许三角形边长的条件。这里有三种情况:

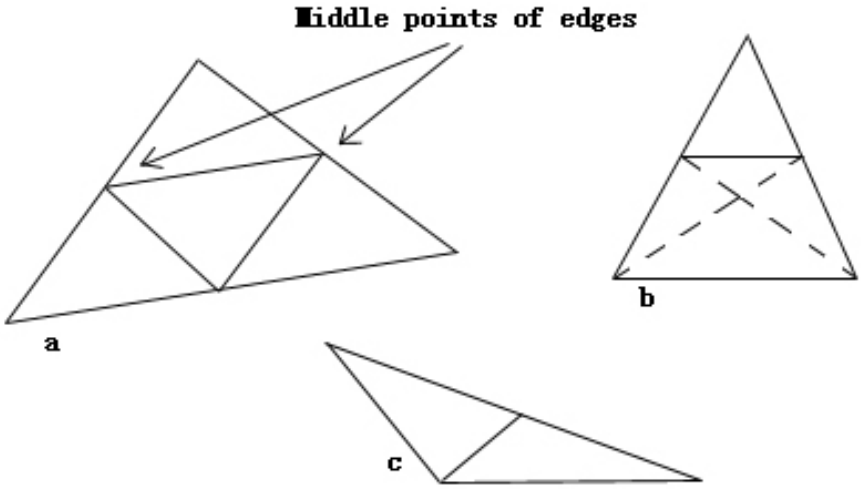

图: 三角形的分解; (a)三角形的三条边都大于最大允许三角形边界,(b)两条边大于最大允许三角形边界,(c)一条边大于最大允许三角形边界。

如图所示, 当三角形的三条边都大于最大允许三角形边界时, 取三边中点, 将原来的三角形分解成为新的四个三角形; 当三角形的两条边大于最大允许三角形边界时, 取这两边的中点，再连接图中两条虚线中的任一条，将原来的三角形分解成为新的三个三角形; 当三角形的一条边大于最大允许三角形边界时, 取这条边的中点，连接这条边对应的顶点，将原来的三角形分解成为新的两个三角形;

遍历所有有效的三角形, 分解不满足最大允许三角形边界的三角形, 直到所有的三角形都满足条件.

5.实验情况以及出现的问题

对三维的这部分, 开始时对面片模型的考虑不够, 没有意识到原始数据输入中将存在的问题. 原因是二维中的数据全部通过在平面上直接画出, 有相应的过程分析处理这些输入数据, 再由保存的输入数据结构构造算法所需的辅助数据结构. 而三维中原始数据的输入是通过分别处理曲面的每一条边界曲线, 计算每条边界曲线的分割数目, 然后得到一些曲线上的分割点, 再把这些三维上的点映射到二维空间上, 按照一定顺序连接起来. 这样就有可能出现在二维输入中不会出现的情况. 比如说, 点的类成员包括坐标和序号, 程序中是通过点的序号来识别不同点的. 当加入一个序号不同坐标相同的点的时候, 就会造成一定的冗余数据, 当加入多个序号不同坐标相同的点的时候, 可能会被误判断为出现不合法情况. 三维的原始曲面的数据的输入因三维的原始曲面的类型而异. 而且即使是同一种曲面, 数据的输入也很可能会不同. 当原始曲面是一个比较简单的情况时, 还可以控制原始输入的顺序性; 但是当原始曲面是一种比较复杂的情况时, 就很难控制原始输入的顺序性和保证数据的不重复输入. 我们从最简单的八分之一椭球面开始尝试, 遇到了上面所述的种种问题。

在二维部分程序的最初设计时, 我们以为所建立的严格的 DELAUNAY 三角划分算法可以直接被三维部分的程序调用，而三维部分程序的最复杂的情况将出现在数据的输入和二三维空间的映射上，但是当三维部分调用二维的算法时, 才发现在二维程序中设定的一些判断情况不适用于三维的程序。比如说, 二维中设定 FIX 为真的边不可以分割, 这实际上就是严格的 DELAUNAY 三角划分的目的所在, 但是在三维中会出现问题.

还有一个比较失败的地方是做三维部分程序的时候，没有一开始就先从二维部分的基础上进行, 而是重新构造界面, 由于本来打算用 OPENGL 显示输出结果, 但是又不熟悉 OPENGL 和 MFC 的联合使用, 用了很多时间实践界面, 最后才发现三维部分和二维部分存在一些差异, 没有时间再重新修改二维部分的程序了. 如果一开始就从二维部分的基础上, 只把结果映射到一个视面上显示出来, 就有可能完成三维部分的示例. 因为如果原始曲面是八分之一椭球面，而且选用标准坐标系的话, 它的映射就比较简单, 只需要修改部分二维部分的判断情况, 和一个判断满足最大允许三角形边长条件并分割不符合的三角形的函数即可, 而这个判断的过程只能用遍历来实现, 不存在算法复杂度的改善问题。

我做的工作是编写二维部分和三维部分所有的界面, 构造存储原始数据的数据结构及其相应成员函数, 包括一些算法需要的特别的类的成员函数。二维部分中构造最外层三角形(Super_Tri()), 整理分割好的三角形序列(Tri_Trim(); Tri_Recursion()), 和一些三角划分中判断情况的函数, 以及最后的显示。

[参考文献]:

[1] “AN IMPROVED INCREMENTAL ALGORITHM FOR CONSTRUNCTING RESTRICTED DELAUNAY TRIANGULATIONS “, Marc Vigo Anglada, Spain, *Computer & Graphics*, vol.21.No.2.pp 215-223,1997.

[2] “Triangulation of trimmed surfaces in parametric space”, X Sheng and B E Hirsch, *COMPUTER-AIDED DESIGN*, vol.24, n.8, pp.437-444, august, 1992.

[3] “Directional adaptive surface triangulation”, Marc Vigo Anglada, Nuria Pla Garcia, Pere Brunet Crosa, COMPUTER AIDED GEOMETRID DESIGN, 16, pp.107-126, 1999

[4] “Degree reduction of Bezier curvers”, M A Watkins and A Jworsey, Computer Aider Design

Vol.2.No.7.Sep.pp 398-405,1988