C++程序实现:

基于层次结构的
3维凸多面体求交算法

蔡洪旭
2004310470
hxcai00@mails.tsinghua.edu.cn

王琛
2004310420
wc00@mails.tsinghua.edu.cn

张靖
2004310461
mitjj00@mails.tsinghua.edu.cn

2004.11—-2005.1

第一章　引言

计算两个凸多面体（在二维中是凸多边形）交集的结构是计算几何中非常基本的问题之一，在例如solid modeling等方面有很多的应用，因此被进行了广泛的研究。对于二维的情况，也就是多边形，无论是对于该问题还是相对应的判定问题（就是判断交集是否非空），在理论和实现上都有很好的结果。已知的结果是，无论对于判定问题还是原始的问题，都存在 $O(n)$ 时间的算法。而且这些算法的描述和实现都相当的直观。

但是，正如很多计算几何中的问题那样，将结论从平面推广到三维情况问题的难度就增加了很多。 Dobkin 和 Kirkpatrick [3]通过将二维的 Kirkpatrick 层次结构的思想推广到三维，证明了对于两个组合复杂性分别为 n_p 和 n_q 的两个凸多面体 P 和 Q，通过线性时间和空间的预处理，可以在 $O(\log n_p \log n_q)$ 内判定是否交集非空。该算法提出了多面体的 Dobkin-Kirkpatrick 层次结构的概念。对于这个问题的原始版本，也就是计算交集的组合结构的问题，首先是 Muller 和 Preparata 给出了一个 $O(n\log n)$ 的算法。那么是否存在线性时间的算法呢？直到好几年以后 Chazelle 给出了这样一个算法（参看文献[1]）。他巧妙地运用 Dobkin-Kirkpatrick 层次结构以及多面体的对偶形式得到了理论上的最优时间复杂度的算法。但是，由于其算法的设计很大的依赖于对偶变换，使得证明过程非常复杂，同时算法时间的常数系数相当大。Martin[6]对于其算法作了一定的改进，将原空间和对偶空间的层次结构转变成全部在原空间中的内外层次结构，使得复杂性的分析简化了很多，同时也改进了时间的常数。

我们小组在程序实现中的算法主要依据 Chazelle[1] 和 Martin[6] 的论文。由于算法本身实现的难度，在实际编程过程中我们在降低了对时间复杂性的要求的条件下，重点在于多面体的 Dobkin-Kirkpatrick 层次结构以及多面体交集演示，具体使用的算法和数据结构与原文有一些区别。这些内容具体参看第三章。

本文第二章主要简略介绍 Dopkin-Kirkpatrick 层次结构和原文中的算法，以及一些被证明的结论。第三章就我们所具体使用的算法和数据结构加以详细说明。第四章对程序的整体设计和界面显示，尤其是3D显示进行介绍，并对程序的性能进行一些数据点测试。最后是我们对大实验的总结和体会。

第二章　Dobkin-Kirkpatrick层次与求交算法

由于定义和算法的复杂程度，本文不可能详细解释和证明完全的算法和结论，细节请参看参考文献[6]。本章中的定义、记号都尽量和原文一致，以方便比较、阅读。另外，为方便起见，本章中将简记 Dobkin-Kirkpatrick 层次为DK层次。对于原文中的英文单词，仅作直译或直接用英文，如有不妥，请老师指正。

2.1 直观的例子

对于求解类似于凸多面体交问题，一个符合直觉的想法就是对原来的多面体进行改造，使得其组合复杂性降低，同时通过优化的数据结构保持新老多面体之间的关系，以期通过先对较为“简单”的多面体求交，再将该结果传递给所关注的多面体上。由于凸多面体的组合复杂性即其顶点个数，我们怎样将它的顶点数降下来呢？DK层次结构就起到了这样的效果。下面先看一个例子：

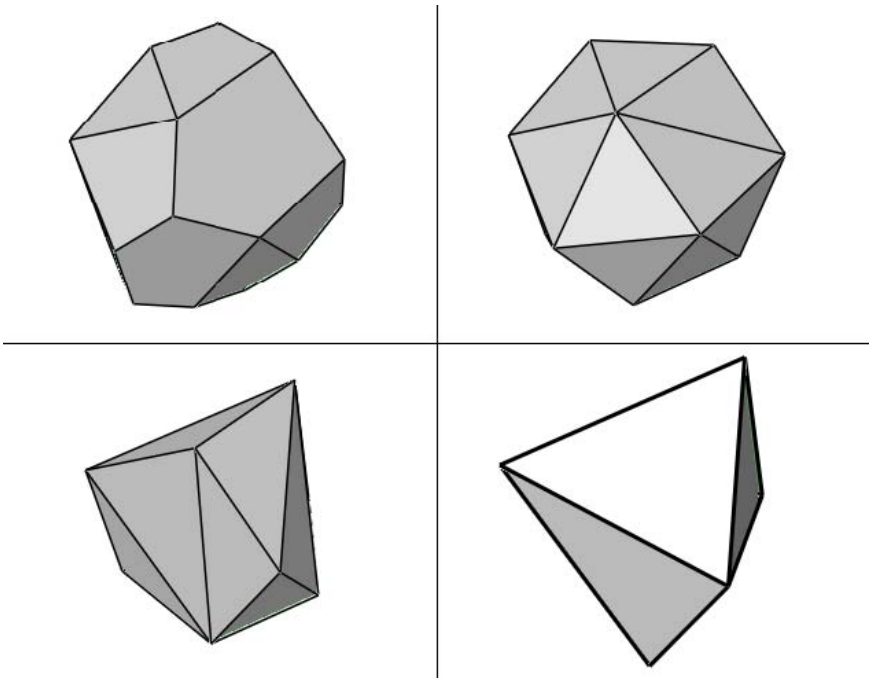

图 2.1　Dobkin-Kirkpatrick 层次（部分）示例

图中，依次从开始的凸多面体上去掉一些点，并对由于减点而产生的“空洞”处进行处理，使得得到的多面体仍是凸的。然后继续作下去，知道最终得到一个三维的单纯形－－正四面体为止。这样一来，我们就得到了关于凸多面体的一个层次结构，我们记录各层之间的关系，然后通过对较低复杂性的多面体进行操作获得原来多面体的性质。

当然，这样的直观认识并不能代替具体的算法和证明。下面为了说明算法的内容，我们先介绍一下能够用来说明DK层次的抽象结构“胞复形”。

2.2 胞复形（cell complex）

定义 2.1: 设 $S = P_1, P_2, \ldots, P_\alpha$ 是一个 d 维空间中的多面体序列，并且满足 $P_{i+1} \subseteq P_i$，$\forall 1 \leqslant i < \alpha$。则对于任意的 i，我们称 $P_i - P_{i+1}$ 中的每一个连通部分为一个*真胞（proper cell）*，并将所有的真胞和 P_α 以及 $\overline{P_1}$ 统称为胞。S 诱导出的所有胞构成一个*胞复形（cell complex）*，记为 $\mathcal{C}(S)$。（注意胞不一定是凸集）

对于 $\mathcal{C}(S)$ 中一个真胞 $c \in \mathcal{C}(S)$ 我们定义它的*层数（level）* $\mathcal{L}(c) = i$，当且仅当 $c \subseteq P_i - P_{i+1}$，同时定义 $\mathcal{L}(\overline{P_1}) = 0, \mathcal{L}(P_\alpha) = \alpha$。对于 d 维空间中的任一个点 x，将 x 所在的那个胞（注意到胞的定义可以看出所在的胞是唯一的）的层数也定义为 x 的层数，即为 $\mathcal{L}(x)$。

对于一个处于第 i 层的胞 c，我们称它是*简单的（simple）*，如果它的边界面的个数小于某个事先确定的常数 ρ。如果一个胞复形 $\mathcal{C}(S)$ 的所有真胞胞都是简单的，则称 $\mathcal{C}(S)$ 为*简单的（simple）*。如果 $\overline{P_1}$（P_α）也是简单的，则称 $\mathcal{C}(S)$ 为*o-简单的，o-simple*（*i-简单的，i-simple*）。如果所有的胞都是简单的，称为*t-简单的（t-simple）*。

对于一个胞复形，我们需要一个数据结构对它进行储存。由于多面体的边和点形成的图同构于一个平面图，因此我们可以用一系列的课程中也多次用到的结构－－双向链接边表（DCEL）来表示 S 序列中多面体。为了表现胞复形的层次关系，我们还要求对于每个胞，我们可以在常数时间得到包围它的所有 P_i 和 P_{i+1} 的面的列表。同时，对于 P_i 中的每一个支撑面 H，必须能够常数时间内给出它所在的唯一的一个胞。另外，也是非常重要的一点是，对于 P_i 的每个面 f，如果它包含了 P_{i+1} 的某个面，则它需要包含指针指向这个关联的面；如果 P_{i+1} 中没有与之关联的面，则它也需要能够通过指针指向它所在的第i层的那个胞。建立在DCEL的基础上，这些要求都是很容易实现的。

跟我们的目的相关的胞复形上的操作主要是 Point Location 和 Ray Tracing，这也是我们后面需要的基本（primitive）操作。因此首先给出它们的定义。

定义 2.2: （Point Location 和 Ray Tracing）在给定的胞复形 $\mathcal{C}(S)$ 中，

Point Location 操作：给定空间的点 x 的坐标，确认其所在的唯一的胞 c；

Ray Tracing 操作：给定某个胞 c 中的任一点 x 和一个向量 $\overrightarrow{v}$，计算以 x 为起点，以 $\overrightarrow{v}$ 为方向的射线 r 从 c 中射出后将进入的下一个胞复形 c'。

对于它们在胞复形上的复杂性，我们不加证明的引用下面的定理来说明：

定理 2.1: 给定 $C = C(P1, P_2, \ldots, P_\alpha)$ 是一个任意的简单胞复形。给定区域 $P_1 \cap \overline{P_\alpha}$ 中或者 P_α 的一条边上的点 x，若给出它所在的真胞 c 或者 P_α 上的边，则由 x 发出的射线 r 所进入的下一个胞 c' 可以在 $O(|\mathcal{L}(c') - \mathcal{L}(x)|)$ 时间内得到。

这个定理是本节最主要的结果，在后面的求交算法中将被作为基本操作直接应用。由该定理，我们可以把区域 $C_N = P_1 - \overline{(P_\alpha)} \bigcup_{e \in E(P_\alpha)} e$ 定义为 C 的*可导航区域*（*Navigable Region*）。在该区域内的 Ray Tracing 操作可以在定理中所提的时间内完成。

对于 i（o，t）-简单的胞复形，可以得到类似定理2.1中的结论，只是可导航区域将扩大至 P_i 内（分别为 $\overline{P_\alpha}$ 内,和全空间）。

对一个点 p 进行 Point Location，可以利用 Ray tracing 的结论，从某一个已知道其所在胞的点 q 出发，以 $\overrightarrow{qp}$ 方向作 *Raytracing*，从而可以在 $O(L)$ 时间内得到 p 点的定位，其中 L 为胞复形的最大层数。

下面我们将注意力回到凸多面体，考虑其内外的 DK 层次结构。

2.3 多面体内外层次表达

定义 2.3: 对于 $\mathcal{R}^d$ 中的一个凸多面体 P，我们把一系列多面体 $P_1, P_2, \ldots, P_k$ 称为 P 的一个*内层次表达*（*Inner Hierarchical Representation*），当且仅当

1. $P_1 = P$，
2. P_k 为d维的单纯形（在三维就是一个正四面体），
3. $V(P_{i+1}) \subseteq V(P_i)$，其中 $V(P_i)$ 表示 P_i 的顶点集，
4. $V(P_i) - V(P_{i+1})$ 中的任两个顶点在 P_i 上不相邻，
5. $V(P_i) - V(P_{i+1})$ 中所有的顶点在 P_i 中的度不超过一预先给定常数 κ。

类似的我们把多面体的外层次表达的定义列出：

定义 2.4: 对于 $\mathcal{R}^d$ 中的一个凸多面体 P，我们把一系列多面体 $P_1, P_2, \ldots, P_k$ 称为 P 的一个*外层次表达*（*Outer Hierarchical Representation*），当且仅当

1. $P^1 = P$，
2. P^k 为d维的单纯形，
3. $D(P^{i+1}) \subseteq D(P^i)$，其中 $D(P^i)$ 表示 P^i 的支撑平面的集合，

4. $F(P^i)-F(P^{i+1})$ 中的任两个极大面在 P^i 不相邻，其中 $F(P^i)$ 表示 P^i 的极大面的集合。

5. $F(P^i)-F(P^{i+1})$ 中所有的极大面在 P^i 中的相邻的面的个数不超过一预先给定常数 κ。

容易看到，一个多面体的外层次相当于其对偶多面体的内层次的对偶，因此，对于内层次结构的所有结论，我们可以很容易的推广到外层次中。另外在前一节所得到的胞复形的结论，可以看出均可用在层次结构上。

2.4 层次结构的深度与复杂性

那么一个很直接的问题是，一个多面体的层次结构构造的时间和空间复杂性是怎么样的呢？

让我们首先来看看内层次结构如何构造。根据我们对于层次表达的要求，可以明显地看出，我们可以通过每次从多面体 P_i 的顶点集合中去掉一个独立集 I（Independent Set），并且该独立集中每个的点满足度数不超过 κ 即可得到满足条件的 P_{i+1}。这样，P_i 和 P_{i+1} 的顶点个数之差恰为独立集的势 $|I|$。

Kirkpatrick[5]非常直接的证明了任意一个多面体，我们可以在 $O(n)$ 时间内找到一个不小于 $\frac{n}{24}$ 个顶点的独立集，并且每个点的度不超过11，其中 n 为多面体的顶点个数。而Edelsbrunner[4]改进了这个结果，将独立集的势改进至 $n/7$，如果允许 $\kappa \leqslant 12$ 的话。Martin[6]将该结论加以推广，得到一个跟多面体的面与顶点个数的比例相关的结果：在令 $\kappa \leqslant 12$ 时，可以找到超过 $\frac{v}{2\beta+1}$ 大小的独立集，其中 v 为顶点的个数，$\beta = \frac{f}{v}$。这个结果对于多面体层次结构的构造没有太大的影响，但对于线性时间的求交算法是个很有意义的结果。

从理论的意义上，上面这些结果很明显的表明，多面体的层次结构可以在 $O(n)$ 的时间和空间内构造出来。因为我们对于原始的多面体，我们通过上面提到的结论，可以使它每一层减少常数系数的点数，从而经过 $O(\log n)$ 次可以将该层多面体的复杂性降到 $O(1)$。同时，每一次我们需要的时间和空间都线性正比于该层多面体的点数 n_i，若把顶点个数下降的比例计为 $c(<1)$，则可得到总的时间（或空间）复杂度为：

$$O\left(n+cn+c^2n+\cdots\right)=O\left(n\times\frac{1}{1-c}\right)。$$

为线性的。

最后给出下面的定理，它可以由上面的结论推出。后面的线性时间的求交算法将用到该定理。

定理 2.2: 对于已经给定 DCEL 结构的3维凸多面体 P，我们可以建立一系列嵌套的多面体 $S = P^k, P^{k+1}, \ldots, P^2, P, P_2, \ldots, P_k$，使得 $C(S)$ 是一个简单多面体胞复形，有 $\rho < 72$，并且由此得 $|P^k| + |P_k| < 6|P|(\frac{2}{5})(\frac{6}{7})^{k-1}$。

2.5 多面体的求交算法（一）

在这一节和下一节中，我们将统一把两个待求交集的多面体分别记为 P、Q，并以 $P_k, P_{k-1}, \ldots P_2, P, P^2, \ldots P^k$ 和 $Q_k, Q_{k-1}, \ldots Q_2, Q, Q^2, \ldots Q^k$ 分别表示 P 和 Q 的从内层次的第 k 层开始至外层次的第 k 层的层次结构（k 不一定是内外层次的最大深度，即可以取包含 P 的部分的层次结构）。

给定两个凸多面体 P，Q，求二者的交集等价于求两个凸多面体的边界 $\partial P \cap \partial Q$（对于两者是包含关系的特例，可以先调用 Dobkin 和 Kirkpatrick [3]的算法在次线性时间内得到）。因此给定多面体的层次结构，我们首先可以设计如下的算法。

在 P 上任意找一点 v_0，对它在 P 的所有内外层次中进行 Point location，从而找到他所在的胞。然后从 v_0 出发，以它发出的一条边 (v_0, v_1) 的方向进行 Ray tracing，如果这条边在到达另一个端点 v_1 之前与 Q 的面相交（这可以从 Ray tracing 的过程中返回的信息得到确认），则记录该交点出现的位置、该交点所在的 Q 的面，并继续往前进行 Ray tracing。直到到达 v_1，再以 v_1 为起点继续进行 Ray tracing，同时实现对 P 的边的遍历。

然后再以 P 的层次结构为基础，对 Q 上的点和边进行类似的遍历。这样我们就得到所有的边与面的交点，通过运用指针和 DCEL 的数据结构，我们可以很容易的将两者边界面的交集的 *DCEL* 结构构造出来，由于后面介绍算法实现时将对这部分具体介绍，这里不再赘述。更详细的内容可以参考源代码。

这个算法从描述到实现都比较清晰，也是我们最后所选择实现的算法。我们先分析一下它的复杂性。预处理需要计算两个多面体的层次表达，我们已知需要线性的时间和空间。进行边的遍历时，所消耗的时间主要是进行 Ray Tracing 的时间，对于每条边来说，最多穿越另一个多面体的所有层，则最多需要 $O(\log n)$ 的时间复杂度，而每个多面体有 $O(n)$ 条边，故共需要时间 $O(n \log n)$。而计算空间主要用于录所有的边与面的交点，由于每条边最多交对方多面体两次，故所以边需要的的空间为 $O(n)$。而构建最后的多面体可以在 $O(n)$ 的时间、空间中得到，因此最终该算法的时间复杂度为 $O(n \log n)$ 和 $O(n)$。

2.6 多面体的求交算法（二）

上面算法虽然结构清晰，但没能达到线性的时间。因此， Chazelle [1]提出了

对DK层次进行截断的方法，并利用精细的递归，得到了线性的方法。这里我们描述的线性时间算法仍然是基于文献[6] 的改进算法。

首先，跟前一节一样，我们仍然根据在前一节的观察，认为给定两个凸多面体 P 、 Q ，求二者的交集等价于求两个凸多面体的边界 $\partial P \cap \partial Q$ 的交。但为了对他们的每条边进行遍历，我们不能对完整的层次结构进行 Ray Tracing 。因此，我们将其截断为 $C_p = C(P_k, P_{k-1}, \ldots P_2, P, P^2, \ldots P^k)$ 和 $C_q = C(Q_k, Q_{k-1}, \ldots Q_2, Q, Q^2, \ldots Q^k)$ ，其中 k 的选取将用来保证算法的线性时间。但是同时，我们必须保证所要进行遍历的边都在“导航区域”之内，从而才能实现 Ray tracing ，因此我们不直接对两者进行遍历，而是对一个特殊设计的非凸的结构， $(P^k \cap Q) \cup (P \cap Q^k)$ ，的边界 Σ 在导航区域内进行 Ray Tracing 。注意到 $(P^k \cap Q) \cap (P \cup Q^k) = P \cap Q$ ，所以 $P \cap Q$ 上的点都会落在 Σ 上。因此通过遍历之后我们即可以得到 $P \cap Q$ 的所有结构信息，而很容易相信，依靠 *DCEL* 结构和指针的应用，我们可以完成这最后一部的构造工作。

那么现在为止还有三个问题没有得到解决：

1. 是否能保证 Σ 上的边全部在导航区域之内？
2. 如何能遍历 Σ 的所有边？
3. 主要和常见的退化情况是什么？

对第一个问题，由于 $P \cap Q^k$ 和 $P^k \cap Q$ 均是 P^k 和 Q^k 的子集，所以我们在导航区域内进行 Ray Tracing 时唯一有可能离开导航区域的是进入到 P_k 或 Q_k 内部。作者证明了对于任何一条我们所遍历的 Σ 的边，如果它不在 $C(P)$ （或 $C(Q)$ ）的导航区域内，则它要么落在 P_k （或 Q_k ）的某个极大面上，要么落在 $P_k \cap Q^k$ （或者 $Q_k \cap P^k$ ）的一条边上。因此，我们需要递归的计算出 $P_k \cap Q^k$ 和 $Q_k \cap P^k$ 的结构，在此基础上进行导航。（请注意这一部分递归调用的形式。这将对最后程序时间产生最主要的影响。）

对于问题2，我们虽然不能事先求出 Σ ，但是如果事先给定一个 Σ 边上的点，我们就可以从该点出发进行了。我们先构造出 P 和 Q 的层次结构，然后调用 [2] 的算法从而在 $O(\log|P| + \log|Q|)$ 时间内得到 $P \cap Q$ 内的一个点。然后从该点出发进行查找从而得到 Σ 上的点。在对边遍历进行导航时，每条边所需要的时间为当前层次的深度 $O(k)$ 。

在操作时主要对结果有影响的是由于 $(P^k \cap Q) \cup (P \cap Q^k)$ 不是凸集，即可能出现 Σ 上的边互相不连通。例如出现“穿刺”的情况。在这种情况下，其实只要通过在对边进行 *RayTracing* 时，除了对于该边，还要对该边所邻的那个面进行 $O(k)$ 时间的检测，从而对该退化情况作出处理。

这样，我们对于该算法给出了一个直观的描述，至于算法中的细节，不再详细说明。下面分析一下其复杂度。

从上面的介绍看出，为计算 $P \cap Q$，除了我们要递归计算 $P_k \cap Q^k$ 和 $Q_k \cap P^k$ 外，其他的步骤都能在 $O(|P|+|Q|)$ 内完成。因此整个复杂性的递归表示为：

$$T(|P|,|Q|) = T(|P^k|,|Q_k|) + T(|P_k|,|Q^k|) + O(|P|+|Q|)。$$

为了总结得到 $T(|P|,|Q|) = O(|P|,|Q|)$，我们需要选择合适的 k，使

$$|P^k| + |Q_k| + |P_k| + |Q^k| < \delta(|P|+|Q|),$$

对某个常数 $\delta < 1$ 成立。因此只需要满足 $|P^k| + |P_k| < \delta|P|$ 和 $|Q^k| + |Q_k| < \delta|Q|$。回忆定理 2.2，我们得到

$$|P^k| + |P_k| < 6|P| \left(\frac{2}{5}\right)\left(\frac{6}{7}\right)^{k-1},$$

和

$$|Q^k| + |Q_k| < 6|Q| \left(\frac{2}{5}\right)\left(\frac{6}{7}\right)^{k-1}。$$

因此我们需要 $(12/5)(6/7)^{k-1} \leqslant 1$，或者说是 $(6/7)^{k+1} \leqslant 5/12$。取 $k = 7$ 即可满足该条件。从而我们证明了该递归算法的时间复杂度为 $O(|P|+|Q|) = O(n)$。

□

第三章　算法的实现与数据结构

我们以参考文献[6]的算法为基础，基于Dobkin-Kirkpatrick的多面体层次结构，实现了三维空间中两凸多面体的求交运算，理论时间复杂度为O(nlogn)，空间复杂度为O(n)。后面将附实际对比测试数据。

算法的实现主要分为两大部分：多面体层次结构(Polyhedron Hierarchy)的构造与两多面体的求交，有关数据结构的描述主要在第一节。

3.1 多面体层次结构的构造

如前文所述，多面体层次结构(Polyhedron Hierarchy)是胞复形(Cell Complex)的一个特例。在我们的程序中的层次结构便是基于Cell_complex这样一个类。它利用了CGAL计算几何算法库中的多面体（Polyhedron）结构。该类记录了一个完整的多面体层次结构，此外还记录了相邻层次多面体及每个Cell与多面体之间的相互关系，Cell本身并未显式存储。Cell_complex类的完整声明在文件Cell_complex.h中。

Cell_complex类主要包含多面体和Cell两部分信息。

数组P[Poly_min],...,P[Poly_max]存储了从内到外每一层多面体，它的每个分量均为Polyhedron类型，主要内部结构为多面体的DCEL表示，此外它的每个面还包含额外信息，将在后面叙述。其中Poly_min为最内层多面体，Poly_max为最外层多面体，而P[Poly_origin]表示原始的多面体所在的层。

C[0],...,C[Cell_number-1],C_inner,C_outer均为Cell类型，存储了所有Cell的信息。每个Cell实际上包含的是一组指向其各表面（它们均为某层多面体的表面）的指针Facet_list，以及该表面的类型信息type（外表面还是内表面）。C_inner是最内层Cell（即最内层多面体），C_outer是最外层Cell（即最外层多面体的外部空间）。Cell类的声明在cell.h文件中。

点p_c为对偶中心点，在用对偶方法计算外层层次结构时用到。

为了在Ray Tracing时更好的处理每个Cell与多面体之间的关系，在多面体的每个面上存储了额外信息（声明在poly_item.h中），包括：

int id; //编号，用来辨识该表面

int level; //该面所在的多面体的层数

Cell* Cell_out; //指向和该表面外切的Cell（其存在性和唯一性在参考文献[6]的Lemma 3.2中得到证明）

std::list<Cell*> Cell_in; //指向和该表面内切的Cell(由于不一定唯一，故用链表存储)

void* Facet_in; // 指向下一曾多面体中与该表面内切的表面（如果不存在，则为空指针）

层次结构的构造主要通过调用Cell_complex类的构造函数Cell_complex (Polyhedron Poly)来实现。内层多面体主要通过删除不相邻顶点，并维护多面体的凸性来构造。而外层多面体的构造方法是先求对偶多面体的内层多面体，再对偶回来。在整个构造过程中，一直维护着Cell及多面体表面的各项指针，保证它们的正确性。

在构造层次结构的过程中，一个最基本的操作即是删去一个顶点并维护剩余多面体的凸性，在构造外层多面体时还需注意不能允许出现无界情况（主要包括can_remove、remove_vertex_inner和remove_vertex_outer方法）。它也在时间复杂度中占有很重的比例。在编写这部分程序时，我们最初采用了CGAL库Polyhedron类的一些固有方法，效果不是十分理想。为了提高效率，后来我们索性在这部分代码中完全抛开Polyhedron类，自己建立了一个类似DCEL的myDCEL类。事实证明效率提高了很多。

层次结构构造的理论时间复杂度为线性，请参见后面的实际测试数据。

有关层次结构的构造更详细的说明请参看程序源代码的注释。

3.2 两多面体的求交算法

在建立好多面体的层次结构后，主要的任务便是利用它来求交了。如上一章所述，求交主要需要进行的操作有三个，分别是程序中的Ray_tracing方法、Point_location方法和Traverse方法，最后的求交是由Compute_intersection方法来完成的。

Ray_tracing方法的主要作用是，给定一个有向线段segment及其初始点所在Cell，找出从初始点出发下一个要进入的Cell。Point_location则是给定一个点，找到它所在的Cell（通过从中心点p_c到当前点做Ray_Tracing即可完成此项操作）。Traverse是对多面体的每条边在另一个多面体的层次结构中进行一个遍历，由此找到所有边与另一个多面体表面的交。基于前面所述的Cell_Complex结构，易知这三个方法都是较容易实现的。

比较难处理的是退化情况。当进行Traverse的边恰好在某个Cell的表面或边上时，系统造成的舍入误差便有可能对Traverse的结果造成无法预料的重大影响。本程序目前尚无对付此类退化情况的有效办法。暂时的解决方案是对于此类情况部分采用Brute-Force算法。

有关求交算法更详细的说明请参看程序源代码的注释。

第四章　系统介绍和测试

4.1 系统构成

本次实验我们小组选择采用 C++ 作为开发语言。采用 Microsoft Visual C++.net 2003 作为开发工具，整个框架是 MFC 的单文档结构 SDI 。 3D 的显示多用 OpenGL ，并为了进行快速开发，我们使用了由 Kitware® 公司开发的封装 OpenGL 的程序包 vtk 。在求交算法中则利用了 CGAL 算法库建立数据结构和调用它的一些基本的计算几何算法。算法部分代码编译生成intersection.dll文件，供主界面程序PolyhedronIntersection.exe调用。

有关 vtk 以及 CGAL 的安装和使用说明文档，请分别参看 http://public.kitware.com/VTK/ 和 http://www.cgal.org 。

系统主要由前端控制、显示模块和后台的算法模块组成，具体的结构图如图 4.1 所示。

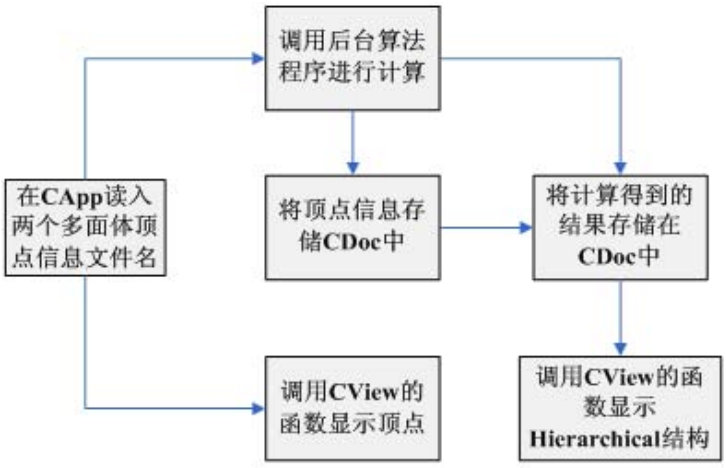

图 4.1　系统结构示意图

前端模块的主要功能是输入输出和控制：

1. 输入方面：曾考虑过从屏幕输入和文件输入两种方式。从屏幕输入在2D的问题中是一种很好的输入方式，简单直接。但是在3D的问题中，直接从屏幕输入的时候并不能点一次就完成一个点的输入，一般至少需要点两次或者用输入框辅助输入第三个坐标的系数而且并不能直观。所以再三考虑决定采用文件输入的方式。规定了输入文件的格式，由用户选择输入的文件名。

2. 输出方面：最重要的就是Hierarchical结构如何显示。因为Hierarchical的

结构在3D显示的时候是嵌套的，如果直接用面来显示，内层的会被外层的屏蔽。可以考虑的改进方式是采用不同的透明度。曾经实验过，但是当层次很多的时候，显示会比较乱。最后选择了用多面体的顶点和边来显示Hierarchical结构。处于不同的层次的点和线用不同颜色来区分。

3. 控制方面：主要是调用后端的算法并将返回的结果从到前台显示。

前台与后台算法的通讯通过文件读写来实现，这样可以减少两部分工作的耦合程度，便于分工实现编码工作，而且，也是最主要的原因，是方便调试。

4.2 输入输出介绍

输入文件格式可以选择为文本（.txt）与OFF文件。

文本输入：内容为任意分布的点集的三维坐标列表。在这种情况下，程序先对所有的点求凸包得到其凸包的DCEL结构，然后作为输入求交集。（由于相对于求交集来说，求三维凸包的运行时间可以忽略。）

OFF文件：是通用的专门用来显示多面体的文件格式，除了包含点的信息外，还要包含所有面、边信息，较为复杂。具体请参看 http://www.geom.uiuc.edu/software/geomview/ 上的相关介绍。

3维界面显示主要是在一个基于 OpenGL 的开放源码的库 vtk 基础上做的，使用的主要数据类型是 vtkPolyData：

点的表示是 vtkPoint 类型，主要属性是 3D 坐标系中的坐标。

边的表示是 vtkCellArray 类型，主要属性是颜色。

面的表示是 vtkCellArray 类型，主要属性是颜色和透明度。

用上面这个三个部分初始化 vtkPolyData 类型的 points， line， polys。显示的流程参见图 4.2。

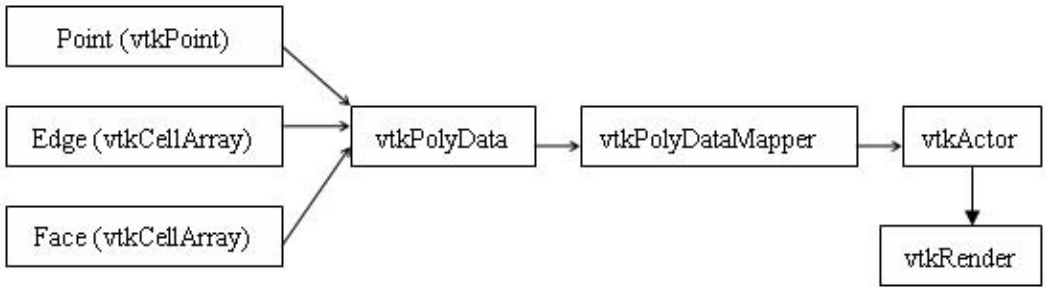

图 4.2　3D 显示流程示意图

而具体显示的层次结构是由后台全部算完之后再统一显示。将层次信息存储在链表中，按照用户的控制显示相应的层次。

4.3 用户操作流程

双击PolyhedronIntersection.exe文件，打开程序主界面。

用户输入接口：菜单输入文件一，输入文件二。

用户控制类型：

显示类型：连续显示vs. 单步显示(设置单步时间长度)。

多面体类型：显示线vs. 显示面vs. 显示线和面。

具体的操作流程如下：

1. 输入文件： Import Points → import polyone ， import polytwo （见图 4.3 ）。在文件对话框中选择需要的 .OFF 文件或是 .TXT 文件；

2. 配置显示： Show → Configuration ，会弹出配置的对话框（见图 4.3 ），打勾表示显示面图，不打勾表示显示线框图，图 4.3 中现在默认是线框图；

3. 运行计算：点击界面上的 Start 键，会运行计算层次结构和交。当点数较多的时候可能需要等待片刻，计算完毕后会弹出对话框显示计算需要的时间；

4. 选择显示的方式： Continuous 表示连续显示， Discrete 表示单步显示。如果在显示过程中需要改变显示方式，也可以使用在这两个按钮之间转换。Time Interval 可以设置在连续显示的时候的时间间隔（见图 4.3 ）。如果选择单步显示，通过按 Next 按钮显示每一步的结果。全部显示完后通过 Clear 清除界面。

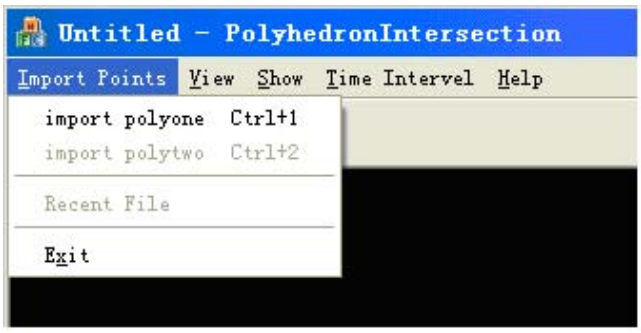

图 4.3　输入文件菜单

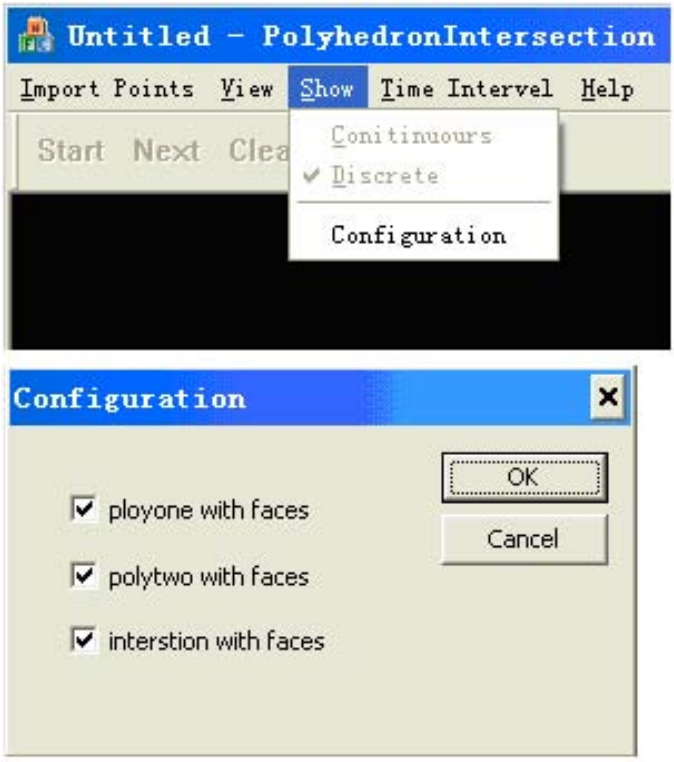

图 4.4　配置显示模式对话框

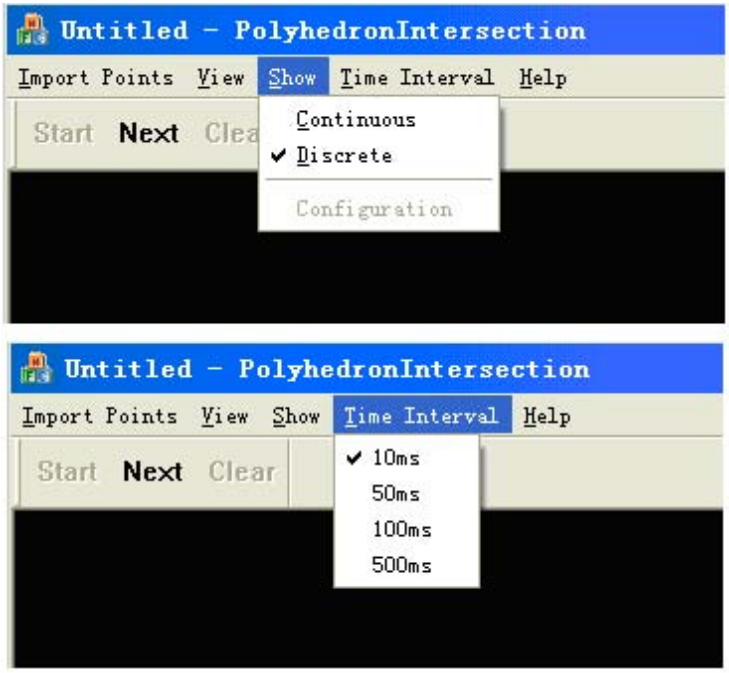

图 4.5　层次结构显示方式对话框

4.4 测试数据

4.4.1 随机点性能测试

为了对程序的算法性能进行实际测试，这里我们采用以下一些随机生成的点作为输入，得到的实验结果如表 4.4.1 所示。实验运行环境为Pentium 4 2.8GHz CPU, 512MB DDR333内存。

序号	顶点数1	顶点数2	时间(层次结构1)/s	时间(层次结构2)/s	时间(求交)/s
1	100	100	< 1	< 1	< 1
2	300	300	< 1	< 1	2
3	500	500	< 1	< 1	4
6	1000	1000	< 1	1	10
7	2000	2000	1	1	20
8	3000	3000	3	2	57
9	5000	5000	6	6	92
10	10000	10000	19	19	238
11	20000	20000	61	63	394

表 4.1　一些随机点测试数据的运行时间

这里所取的随机点全部是分布在球面上的，之所以这样选择是因为所求的是凸多面体的交，对于一般的分布，在求完凸包之后将仅剩下很少的顶点，因此不能达到测试程序时间性能的真实效果。

从表中可以看出，求交所需的时间还是基本接近线形的。但Hierarchy的构造虽然点数少的时候所需时间很短，但随着点数的增加，时间增长还是比较快的。我们猜测随着点数的进一步增长，时间的增长会放缓，但尚未验证这一猜测。

另外，在测试过程中我们注意到，受到硬件和软件的限制，当多面体顶点数达到2000时，相交多面体的显示即已困难，当顶点数达到5000时，原始多面体亦无法正常显示出来。

实验总结与体会

这次的大实验给我们小组所有人都留下了深刻的印象，通过这次大实验，包括邓老师所讲的课，让我们受益良多。

这次我们从刚开选题，就有相当的挑战性。该题目本身作为计算几何最基本的问题之一并不难理解，但原作者（Chazelle和Martin）论文所用的算法，确实相当难以理解。我们对算法的理解分析大概花了我们1个多月的时间。最终我们认为原文中的递归算法，除了其编程难度之外，对于该算法本身的实际性能并不容乐观，尽管是线性时间的算法，但其所用的时间和空间在相对还比较小的输入下就已经无法接受了。所以最后我们选择了实现同样是建立在层次结构基础上的直接遍历的方法。

在算法实现过程中，我们也遇到很多的问题，刚开始主要是调试和退化情况分析的问题。我们选择了一道3D的题目，就注定了是一块硬骨头。由于数据结构十分庞大，并且充满指针，并且着手编写算法模块是在显示模块之前开始的，查错工作难以进行。在bug调试时，必须具备丰富的空间思维能力及极好的记忆力，经常需要在电脑前进行长时间的单步跟踪操作，有的时候我们甚至要在纸上一笔一划画出三维模型来帮助分析。在缺乏直观的检测下，程序的编写可以说步步维艰。直到后来借助偶然找到的CGAL的3维模型显示程序，才开始能够进行一些调试。而调试所用的实验数据，由于该显示程序只能读取OFF文件，使得我们还需要手工将纯粹的点坐标文件变成标准的多面体OFF格式。

这次大作业过程中的另一个体会就是：现成的东西往往不如自己的。我们使用了CGAL和VTK两个C++库，它们都极为庞大，特点是功能全，封装好，然而缺点也在此。学习这些库本身便是非常耗时间的（我们都花了数周的时间来熟悉这些库），而很多东西用起来却不十分顺手，并且由于这些库都并非完美，便会遇到很多不明所以的bug，最后到要修改其源代码的程度（计算凸包的convex_hull_3.h文件我们便有所修改）。某些模块实现的时候，完全采用自己编写的数据结构常常编写效率更高，且运行更快。当然，我们并没有对放弃对这些库的使用的重要原因便是它们的兼容性和通用性更好。

在这中间由于去香港开会，耽误了宝贵的两周时间。在两个模块分别完成之后，在一起进行调试中也出现了很多的问题。在刚匆匆完成半成品时，就被进行了大实验的检查，果然是漏洞百出。非常感谢邓老师在从算法优化到系统设计再到显示效果的各个方面给了我们很多的建议，也让我们理清了思路，也更清楚了解到大实验的意义所在，从而才有了后来较大的改进。由于紧接着考试周，所以我们组成员都抽出有限的时间进行算法改进和程序调试，同时完成最后实验报告，大家都付出了很大的努力，对于培养我们团队合作精神也是很有好处的。

下面谈一谈在大实验中对于计算几何算法的体会。作为理论组的学生，对于我们可能是第一次要将这样的非常理论化的算法变成代码加以实现，感觉非常特别。习惯于 big-O 的我们真正体会到理论与应用上的距离，无论我们以后是否将一直从事理论的工作，但这次这样的体会对于今后的我们都是很有帮助的。

再次非常感谢邓老师无论在课内还是课外一直所给于我们的帮助，同时您积极、严谨的作风也给我们为人治学树立了榜样。

参考文献

[1] Bernard Chazelle. An Optimal Algorithm for Intersecting Three-dimensional Convex Polyhedra. *SIAM Journal on Computing*, Vol 21:195–210, 1992.

[2] D. Dobkin and D. Kirkpatrick. Fast Detection of Polyhedral Intersection. *Theoretical Computer Science*, 27, 1983.

[3] D. Dobkin and D. Kirkpatrick. Dertermine the Separation of Preprocessed Polyhedra－a Unified Approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of Lecture Notes in Comput. Sci., pages 400–413, 1990.

[4] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg, Germany, 1987.

[5] David Kirkpatrick. Optimal Search in Planar Subdivisions . *SIAM Journal on Computing*, Vol 12(1), 1983.

[6] Andrew K Martin. A Simple Primal Algorithm for Intersecting 3-polyhedra in Linear Time. Technical Report 91-16, July 1991.