第0章 预备知识

本书叙述的内容不属于欧几里得的几何证明公理化范畴，而是属于欧几里得的几何构造，即由算法和复杂性分析所组成。欧几里得的几何构造满足算法的所有要求：无二义性、有穷性、确定性、能行性、输入、输出、正确性等。在欧几里得的几何构造中，限定了可允许使用的工具（直尺和圆规）及原始运算（圆规的一个脚置于一个给定点或一条直线上；作一个圆；直尺的边通过一个给定点；作一条直线）。但欧几里得原始运算并不能胜任所有的几何计算（如角的三等分），这一点直到19世纪，阿贝尔、伽罗华等数学家才给出了证明。

在一个几何构造过程中，执行原始运算的总次数称为该过程的复杂性度量，这个概念对应于算法的时间复杂度。同样地，还有对应于算法的空间复杂度的概念。这是欧几里得几何构造过程复杂性的定量测度。

称为计算几何的学科大致有下述几种：Forrest等人依据样条函数处理曲线和曲面（实际上更接近于数值分析）；Minsky和Papert写的一本名为《感知机》的书（副标题为“计算几何”），该书陈述用简单回路构成的网络实现模式识别的可能性（应属于人工神经网络）；计算机图形学是研究用计算机进行图形信息处理（包括表示、输入、输出、存储、显示、检索与变换等）和图形运算（如图的并、交运算）的一门学科，而不

是算法分析；几何定理的机器证明，主要研究定理证明的探索方法及证明过程的推断，而不是几何本身。本书讨论的内容与上述计算几何学科所研究的内容不同，是属于 Shamos 的文章(1975a)中命名的“计算几何”。为了不与上述“计算几何”的命名混淆，并突出 Shamos 的计算几何是研究几何问题的算法及复杂性的，故本书中将 Shamos 的计算几何称为 S 计算几何，而书名仍称为计算几何。

欧几里得货郎担问题、最小生成树问题、隐藏线(面)问题和线性规划问题等许多问题是 S 计算几何研究的基本问题。在 19 世纪的文献中，已经出现了对这些问题的算法研究，但对几何问题进行几何算法的系统研究还是近 20 多年的事情。

本书将通过对几何问题的研究，在以下各章节中给出 S 计算几何的观点、研究方法与几种重要的几何结构。S 计算几何的一个基本观点是，经典的几何对象的表征常常不适合于有效算法的设计。因此有必要建立一些概念及相关的性质，以适应于有效算法的设计。

本章将介绍有关算法与数据结构的一些知识、几何知识及计算模型。这些内容是以后章节所需要的。

0.1 算法与数据结构

0.1.1 算法

众所周知，算法是求解一个问题类的无二义性的有穷过程。这里的过程是指求解问题执行的一步一步动作的集合，每一步动作只需要有限的存储单元和有限的操作时间。另外，如果详细说明一台典型的计算机以及与这种计算机通信的语言，那么凡用这种语言编写的、可以在给定的计算机上执行的过程便称为算法。随机存取机器(RAM)、图灵机等可以作为典型的计算机，拟 ALGOL 语言作为描述算法而非执行的语言。应该指出，算法不等于程序，因此描述算法的方式将是多种形式的，如在拟 ALGOL 语言的描述中可以使用数学记号和自然语言。为了把算法转换成上机程序，还需要进行编程工作。

算法的复杂性包括算法的时间复杂性和算法的空间复杂性。为了说明复杂性的概念，先介绍问题规模的概念。用一个与问题相关的整数量来衡量问题的大小，该整数量表示输入数据量的尺度，称为问题的规模。比如，行列式的规模可以用其阶数 n 来表示，图问题的规模可以用其边数或顶点数来表示，等等。

利用某算法处理一个问题规模为 n 的输入所需要的时间，称为该算法的时间复杂性。它显然是 n 的函数，记为 $T(n)$。

类似地，可以定义算法的空间复杂性 $S(n)$。

下面主要讨论算法的时间复杂性。由于一般不需要知道精确的时间耗费，只

要知道时间耗费的增长率大体在什么范围内即可，因此我们引入算法复杂性阶的概念。

如果对某一正常数 c，一个算法在时间 cn^2 内能处理规模为 n 的输入，则称此算法的时间复杂性是 $O(n^2)$，读作“n^2 阶”。一般定义如下：

如果存在正常数 c 和 n_0，使得当 $n \geqslant n_0$ 时有 $T(n) \leqslant cf(n)$，则称 $T(n)$ 是 $O(f(n))$，记作 $T(n)=O(f(n))$。此时，$f(n)$ 是 $T(n)$ 的增长率的一个上界。

如果 $T(n)=O(f(n))$ 和 $f(n)=O(T(n))$ 同时成立，则称 $T(n)$ 是 $\theta(f(n))$，记作 $T(n)=\theta(f(n))$。此时，$T(n)$ 和 $f(n)$ 的增长率是同阶的。

如果存在正常数 c，使得对无穷多个 n 关系式 $T(n) \geqslant cf(n)$ 成立，则称 $T(n)$ 是 $\Omega(f(n))$，记作 $T(n)=\Omega(f(n))$。此时，$f(n)$ 是 $T(n)$ 的增长率的一个下界。

如果 $T(n)=O(f(n))$ 和 $T(n)=\Omega(f(n))$ 同时成立，则有 $T(n)=\theta(f(n))$。

根据上述定义，两个函数如果同阶，那么它们可以相差一个常数因子，还可以相差比阶低的项，即函数中的低阶项并不影响它的阶数。这时，若要进一步分析 $T(n)$，就应考虑常数因子和低阶项对 $T(n)$ 的影响。

一般情况下，都是取一个简单形式的函数作为阶数的规范表示，如 n^2，n^3，n^6 等。实用中也是这样处理的。

一个算法的时间复杂性如果是 $O(2^n)$，则称此算法需要指数时间。而时间复杂性如果是 $O(n^k)$（k 为有理数），则称此算法需要多项式时间。当 n 很大时，指数时间和多项式时间存在很大的差别。以多项式时间为限界的算法称为有效算法。有效算法的一个本质特征是可以按常规的方法在较短的时间内用一个确定的计算装置进行机械的计算。该计算装置的典型模型是图灵机和 RAM(random access machine)机。

算法的时间复杂性分为最坏情况的时间复杂性和平均情况的时间复杂性。对于给定规模为 n 的各种输入，某算法执行每条指令所需要的时间之和的最大值，称为该算法的最坏情况的时间复杂性；对于给定规模为 n 的各种输入，执行每条指令所需要的时间之和的平均值，称为平均情况的时间复杂性（或期望复杂性或平均特性）。由于规模为 n 的“输入”出现的概率不同，所以有时要考虑加权平均特性。

为了求平均特性，必须对输入量的分布作某种假设。然而切合实际情况的假设，在数学上往往不易处理。因此，确定平均特性比确定最坏情况的时间复杂性要难。

在最坏情况的时间复杂性和平均特性的定义中，都提到“指令”。由于不同的机器执行某一指令（如加法指令）所需要的时间可能不同，因此同一算法在不同机器上执行时所需要的时间也可能不同。为了消除这一差距，人们引入一个理想的计算模型，用它代替具体的机器，以建立时空复杂性概念。这个理想的计算模型称

为随机存取机器(RAM),它只有一个累加器,而且不允许自行修改指令。

RAM 机器由一条只读输入带、一条只写输出带、一个程序存储器、一个存储器和指令计数器组成,如图 0-1 所示。

输入带被划分为一系列方格,每个方格可以放一个整数。每当从输入带读出一个符号时,带头向右移动一格。输出带也被划分为一系列方格。执行一条写指令时,在输出带当前处于带头下的方格里打印一个整数,且带头右移一格,带头不能改写已印出的整数。

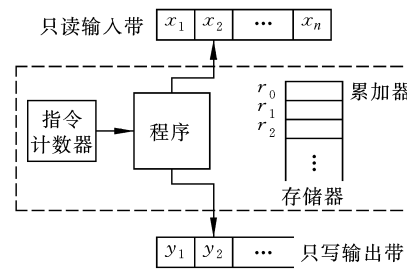


图 0-1 RAM 机器

存储器由一系列寄存器 $r_0, r_1, r_2, \cdots$ 组成,每个寄存器能放下一个任意大小的整数。寄存器 r_i 的个数不受限定。

RAM 的程序不存放在存储器里,于是可假设程序不能修改自身。程序是带(也可不带)标号的指令序列,指令与实际的计算机一样,有算术指令、输入输出指令、间接寻址和转移指令等。用这些指令编写的程序称为 RAM 程序。RAM 的所有计算都在第一个寄存器 r_0(称为累加器)中进行。r_0 中也可放一个任意大小的整数。

RAM 程序的最坏情况时间复杂性和平均特性的定义与上面相同。

下面介绍最佳算法的概念。

在解决某一问题 P 的一类算法 $\mathscr{A}$ 中,需要操作次数最少的算法称为求解问题 P 算法类 $\mathscr{A}$ 的最佳算法。这里是用指定基本操作来确定算法类的。而算法 A 的基本操作是指 A 的关键操作,并且 A 的操作总数与基本操作次数成正比(当 n 增加时)。

形式上,设算法类 $\mathscr{A}$ 求解问题 P。对任一 $A\in\mathscr{A}$,其时间复杂性为 $T_A(n)$,定义

$$C_P(n) = \min_{A\in\mathscr{A}}\{T_A(n)\}$$

称为问题 P 关于 $\mathscr{A}$ 的计算复杂性。问题 P 的计算复杂性是 P 所固有的。由于 A 是 $\mathscr{A}$ 中任一算法,所以 $C_P(n)$难以估计。为此可以对问题 P 先求出在算法类 $\mathscr{A}$ 下 $C_P(n)$的下界(称为下界问题),然后努力寻找达到这一计算复杂度的算法。

若能构造函数 $g(n)$,并可以证明,对任一 $A\in\mathscr{A}$,有 $T_A(n)\geqslant g(n)$成立,则 $g(n)$是$C_P(n)$的一个下界。

设已找到求解 P 的算法 A,$A\in\mathscr{A}$,并分析 A 的复杂性 $T_A(n)$,有

$$C_P(n) \leqslant T_A(n) = f(n)$$

则 $f(n)$是 $C_P(n)$的一个上界。

设 $f(n)$,$g(n)$分别为 $C_P(n)$的上界和下界,如果 $f(n)=\theta(g(n))$,而且 $f(n)=$

$T_A(n)$，则 A 是 $\mathscr{A}$ 中求解问题 P 的最佳算法(复杂性的阶不可能降低，但系数可能减少)。如果 $f(n)\neq\theta(g(n))$，则存在优于 A 的算法或者存在一个更低的下界。

设 A 是求解问题 P 的算法，那么可以用执行 A 时的“循环次数”作为算法时间复杂性的度量标准；在定义“最坏情况时间复杂性和平均特性”时，可考虑将“执行每条指令所需要的时间之和”作为度量标准；另外，“基本操作的次数”也可作为度量标准。前者使用方便，但估计粗糙；中者与机器的类型有关，需要引入一个抽象的计算模型，它对建立“最坏情况时间复杂性和平均特性”的概念有益；后者与机器、所采用的语言、实现方式等均无关，是较理想的度量标准。值得注意的是，我们只在同一种度量标准下比较某类算法中不同算法的优劣，而且在比较阶的同时，要特别注意以何种运算作为时间单位。如果时间单位不同，那么同一算法可能有不同的阶。

本书的主要目的是阐述求解几何问题的算法，以及对算法最坏情况复杂性的估计。

算法设计中的常用方法也适用于几何算法设计，如分治法、贪心法、递归法、随机化方法和动态规划方法等。除了这些方法之外，依据几何问题的特征，将采用一些特殊的技巧，如累接、修剪和搜索、几何变换、轨迹和扫描等。下面简要介绍平面扫描方法，其他方法将在后续章节中结合具体问题予以介绍。

给定平面上 n 条线段的集合 S，报告线段集合中的所有交点。考虑垂直直线 l，它将平面分割成两个半平面，即左半平面 L 和右半平面 R。显然，问题的解是 L 中的解和 R 中的解的并。当直线 l 穿过线段集合 S 时必与某些线段相交。另外，容易观察到，当 l 非常接近某两条线段的交点时，该两条线段和 l 的两个交点在 l 上是相邻的。因此，只要对 S 生成所有的垂直切割，就可以发现所有的交点。这就相当于把直线 l 从左向右扫过平面，故称为平面扫描方法。当然，不可能连续地生成所有垂直切割的无限集合，但可以通过把平面划分为若干个垂直长条来实现，这些垂直长条由 S 中线段端点或交点确定。因此，只需从垂直长条的左边界跳到右边界，修改长条的次序，并测试 l 上新的相邻线段是否相交。l 从左向右扫过平面时，在称为“事件点”的特殊点处暂停，以测试 S 中某些线段对是否相交。为此，需要两个基本结构：(1)事件点进度表。S 中线段端点的 x 坐标的排序，它确定了扫描线 l 的暂停位置。执行平面扫描算法过程中，发现交点时便要修改事件点进度表。(2)扫描线状态，它是扫描线 l 和 S 中线段的交点的一种描述。在每个事件点处修改扫描线状态。

0.1.2 数据结构

数据结构是组织信息的方式，它和算法一起使问题可能得到有效的解。下面简要回顾有关的数据结构及其功能。

几何算法设计中遇到的对象是集合和序列。设 U 是某种几何对象的全集，u 是 U 中的任意元素，而 S 是 U 的子集。集合操作中的基本运算有：(1)MEMBER(u,S)：$u\in S$? (2)INSERT(u,S)：把 u 加入 S；(3)DELETE(u,S)：从 S 中删去 u。设$\{S_1,S_2,\cdots,S_k\}$是一组两两不相交的集合，其上的运算有：(1)FIND(u)：若 $u\in S_i$，则报告 i；(2)UNION (S_i,S_j；S_k)：产生 S_i 和 S_j 的并，且记为 S_k。当 U 是全序时，有运算：(1)MIN(S)：报告 S 的最小元素；(2)SPLIT(u,S)：把 S 划分为 S_1 和 S_2，且 $S_1=\{v\mid v\in S$ 并且 $v\leqslant u\}$，$S_2=S-S_1$；(3)CONCATENATE(S_1，S_2)：设对于任意 $a\in S_1$ 和 $b\in S_2$，有 $a\leqslant b$，产生有序集 $S=S_1\cup S_2$。

对于有序集，依据支持的运算分类数据结构如下：字典，支持运算 MEMBER，INSERT，DELETE；优先队列，支持 MIN，INSERT，DELETE；可并队列，支持 INSERT，DELETE，SPLIT，CONCATENATE。通常以均衡二叉树来实现上述数据结构，此时，完成 INSERT 等运算的时间和存储在数据结构中元素个数的对数成比例；存储和集合大小成比例。另外，上述数据结构还可以表示为一个线性的元素组(称为表)，根据插入和删除的位置不同而分为队和堆栈，以 U_1 表示队或堆栈。记号“$\Rightarrow U_1$”表示插入 U_1，而“$U_1\Rightarrow$”表示从 U_1 中删去。

通过对无序集元素采用赋予“名字”，且用字母次序的办法可以把无序集转变成有序集，这种情况的一种数据结构是可归并堆栈，它支持 INSERT，DELETE，FIND，UNION 等运算。仍以均衡树来实现该数据结构，并且耗费时间 $O(\log n)$完成 INSERT 等运算，其中 n 是存储在可归并堆栈中集合的大小。如无特别说明，本书中出现的 $\log n$ 均是以 2 为底的对数函数。

几何算法设计中广泛使用上述数据结构，此外，依据几何问题的特征，我们还采用线段树和双重连接边表等特殊的数据结构。

1. 线段树

线段树是一棵二叉树，记为 $T(a,b)$，并用 v 表示它的根，其中参数 a，b 是两个整数，它们表示一个区间的始点和终点，记为 $B[v]=a$，$E[v]=b$。线段树 $T(a,b)$可以递归地构造如下：它包含一个根 v，根具有参数 a 和 b，且若 $b-a>1$，则它有左子树 $T(a,\lfloor(B[v]+E[v])/2\rfloor)$和右子树 $T(\lfloor(B[v]+E[v])/2\rfloor,b)$，其中“$\lfloor\ \rfloor$”表示低限。左、右子树的根分别为 LSON[$v$]和 RSON[$v$]。结点 v 所代表的区间由 $B[v]$和 $E[v]$确定，并满足关系式$[B[v],E[v]]\subseteq[a,b]$。图 0-2 表示线段树 $T(2,14)$。线段树的叶结点所代表的区间，称为基本区间，而其他结点表示$[B[v],E[v]]=[a,b]$中的某个子区间。容易将 $T(a,b)$建成均衡树(所有叶结点位于相邻的两极上)且具有深度$\lceil\log(b-a)\rceil$，其中“$\lceil\ \rceil$”表示高限。

线段树支持插入和删除运算。给定线段树 $T(a,b)$和任意区间$[c,d]$，下述过程将$[c,d]$插入线段树 $T(a,b)$。

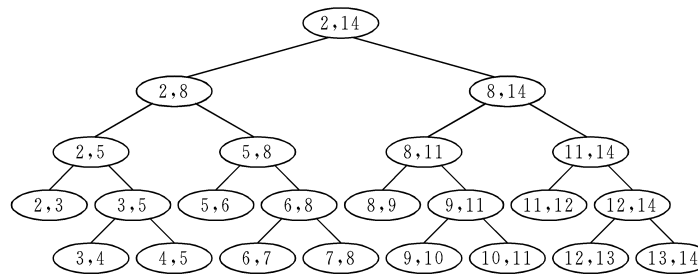


图 0-2 线段树 $T(2,14)$

procedure INSERT(c,d; v)
begin
  if $c\leqslant B[v] \wedge E[v]\leqslant d$ **then** 把$[c,d]$分配给 v
  else if $c<\lfloor (B[v]+E[v])/2 \rfloor$ **then** INSERT(c,d; LSON$[v]$);
    if $\lfloor (B[v]+E[v])/2 \rfloor<d$ **then** INSERT(c,d; RSON$[v]$)
end

INSERT(c,d; root(T))运算在 T 中沿下述路径进行：从根到结点(称为树权)v^* 的初始路径，称为 P_{IN}；从 v^* 引出两条路径 P_L 和 P_R。或者把整个$[c,d]$分配给 v^*(此时，P_L 和 P_R 均为空)；或者整个分配给 P_L 结点的所有右子结点(这些右子结点不在 P_L 上)，以及 P_R 结点的所有左子结点(这些左子结点不在 P_R 上)，确定$[c,d]$的分配结点。如图 0-3 所示，其中双圆圈结点是分配结点。

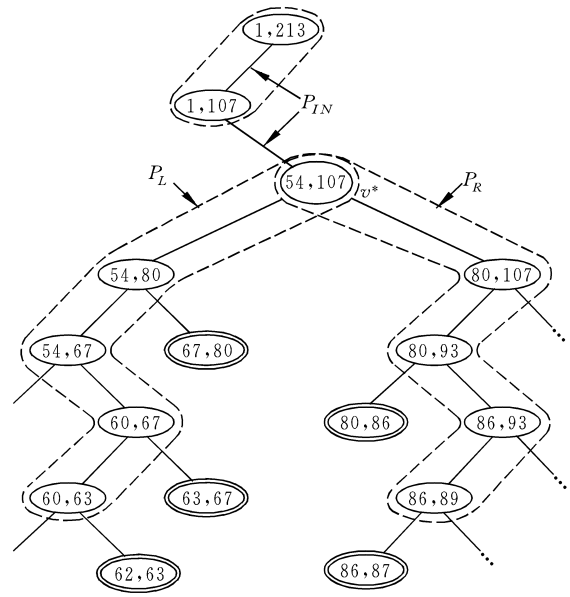


图 0-3 把区间[62,87]插入 $T(1,213)$

通常只需要知道分配给结点 v 的区间集合的基数，用 $c[v]$表示该基数，然后若将$[c,d]$再分配给 v，则有

$$c[v] \leftarrow c[v]+1$$

DELETE(c,d; v)运算表示如下：

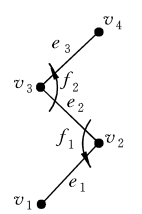
procedure DELETE(c,d; v)
begin
  if $c \leqslant B[v] \wedge E[v] \leqslant d$ **then** $c[v] \leftarrow c[v]-1$
  else if $c < \lfloor (B[v]+E[v])/2 \rfloor$ **then** DELETE(c,d; LSON$[v]$);
    if $\lfloor (B[v]+E[v])/2 \rfloor < d$ **then** DELETE(c,d; RSON$[v]$)
end

线段树是一种常用的数据结构。若要知道包含给定点 x 的区间个数，则对线段树 T 进行一次二叉查找(即从根到一个叶的一条路径的遍历)即可解决问题。

2. 双重连接边表

双重连接边表是另一种常用的数据结构，它适合于表示嵌入平面的平面图(连通)。给定平面图 $G=(V,E)$，G 的平面嵌入是将 V 中的顶点映射到平面中的一点，且 E 中的边映射到边的端点的两个像之间的一条简单曲线，所有简单曲线除端点外互不相交。可以取简单曲线为直线段。

设平面图 $G=(V,E)$的顶点集 $V=\{v_1,v_2,\cdots,v_n\}$，边集 $E=\{e_1,e_2,\cdots,e_m\}$。G 的双重连接边表如图 0-4 所示。每条边 e_i 给 4 个信息段 V_1，V_2，F_1 和 F_2 及两个指示字段 P_1 和 P_2，所以可以用相同名字的 6 个数组实现对应的数据结构，每个数组包含 m 个单元。各信息段的意义如下：V_1 与 V_2 分别为边 e_i 的起点和终点；F_1，F_2 分别表示位于有向边 e_i 的左侧面和右侧面的名字；指示字 P_1(P_2)指向一条边及端点，该边是边$\overrightarrow{V_1V_2}$绕 V_1(V_2)按逆时针方向旋转后遇到的第一条边。



	V_1	V_2	F_1	F_2	P_1	P_2
e_1	v_1	v_2	f_1			
e_2	v_2	v_3	f_1	f_2	e_1	e_3
e_3	v_3	v_4		f_2		

图 0-4 双重连接边表的说明

从双重连接边表可以得到关联于给定顶点的边或者围绕一个给定面的边。假设 G 有 n 个顶点和 f 个面，另外，数组 $HV[1..n]$和 $HF[1..f]$分别是顶点和面表的首标，用$O(n)$时间扫描数组 V_1 和 F_1 能填满这些数组。下述过程产生关联于 v_j

的边序列。

```
procedure VERTEX(j)
begin
  a←HV[j];
  a0←a;
  A[1]←a;
  i←2;
  if V1[a]=j then a←P1[a] else a←P2[a];
  while a≠a0 do
    A[i]←a;
    if V1[a]=j then a←P1[a] else a←P2[a];
    i←i+1
end
```

VERTEX(j)的执行时间和关联于 v_j 的边数成比例。把上述过程中的 HV 和 V_1 分别换成 HF 和 F_1，即可得到围绕 f_j 的边序列。

边表是描述平面图 G 的另一种形式，对于每个顶点 $V_j \in V$，边表包含与 V_j 关联的边，且这些边按逆时针方向排列。显然耗费 $O(V)$时间可以把 G 的边表转换成双重连接边表。

0.2 相关的几何知识

0.2.1 基本定义

本书考虑的对象是欧几里得空间的点集，包含通过两个给定点的直线、直线上两定点确定的直线段、3 个给定点确定的平面及由有序点列确定的多边形，等等。另外，假设有一个坐标系，使得每个点表示为相应维数的笛卡儿坐标的向量。我们还约定点集是有限可列举的。本节将简要回顾相关的几何知识，其细节请参见有关资料。

用 E^d（或 R^d）表示 d 维欧几里得空间，即具有度量 $\left(\sum_{i=1}^{d} x_i^2\right)^{1/2}$ 的实数 $x_i(i=\overline{1,d})$ 的 d 元组$(x_1,x_2,\cdots,x_d)$ 的空间。下面给出本书所涉及的基本对象的定义。

(1) **点** 用 p 表示一个点，E^d 中的点 p 定义为一个 d 元组$(x_1,x_2,\cdots,x_d)$。点 p 也可解释为有 d 个分量的向量，此向量的起点为坐标原点，终点为点 p。

(2) **线，线性簇** 在 E^d 中给定两个不同的点 p_1 和 p_2，线性组合

$$\alpha p_1+(1-\alpha)p_2 \quad (\alpha \text{ 是实数，即 } \alpha \in \mathbf{R}) \tag{0-1}$$

是 E^d 中的一条线。如果给定 E^d 中 k 个线性独立的点 $p_1, p_2, \cdots, p_k (k \leqslant d)$，则线性组合

$$\alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_{k-1} p_{k-1} + (1 - \alpha_1 - \cdots - \alpha_{k-1}) p_k \quad (\alpha_i \in \mathbf{R})$$

是 E^d 中的$(k-1)$维的线性簇。

(3) **线段** 在 E^d 中给定两个不同的点 p_1 和 p_2，若在式(0-1)中加入条件 $0 \leqslant \alpha \leqslant 1$，则得到 p_1 和 p_2 的凸组合，它描述了连接两点 p_1 和 p_2 的直线段，并记为 $\overline{p_1 p_2}$(无序对)。

(4) **凸集** 设 D 是 E^d 中的域，且 p_1 和 p_2 是 D 中的任意两点，如果线段 $\overline{p_1\ p_2}$ 完全包含在 D 中，则域 D 是凸的。

可以证明，两个凸域的交是一个凸域。

(5) **凸壳** E^d 中点的集合 S 的凸壳是 E^d 中包含 S 的最小凸域的边界。

(6) **多边形** 多边形定义为线段的有限集合，该集合中每条线段的端点恰好为两条边所共有，而且没有边的子集具有这个性质。线段为多边形的边，其端点是多边形的顶点，而且顶点数和边数相等。

若多边形的不相邻边对不相交，则称该多边形为简单多边形。简单多边形把平面划分为两个不相交的区域：内部区域(有界的)和外部区域(无界的)。一般情况下，多边形理解为边界和内部区域的并。

若简单多边形 P 的内部区域是凸集，则称 P 是凸的。如果 P 内存在点 q，使得对于 P 的所有点 p，线段 $\overline{qp}$ 完全位于 P 内，则此简单多边形是星形多边形。具有上述性质的 q 的轨迹称为 P 的核。

(7) **平面图** 若图 $G=(V,E)$ 能不交叉地嵌入平面，则 G 是平面图。平面图的直线平面嵌入确定平面的一个划分，称为平面剖分。设平面图的顶点数、边数和区域数分别为 v，e 和 f，则由欧拉公式有

$$v - e + f = 2 \tag{0-2}$$

如果每个顶点的度数$\geqslant 3$，则 v，e 和 f 是两两成比例的。

(8) **三角剖分** 若平面剖分的所有有界区域是三角形，则此平面剖分称为三角剖分。有限点集 S 的三角剖分是 S 上具有最大边数的平面图。或者说，由不相交的直线段来连接 S 的点得到 S 的三角剖分，以至于每个三角形区域都在点集 S 凸壳的内部。

(9) **多面体** E^3 中，多面体定义为平面多边形的一个有限集，并且多边形的每条边和相邻的另一个多边形共有，而且没有多边形子集具有相同的性质。多边形的顶点和边是多面体的顶点和边；多边形是多面体的小面。

多面体中，如果没有不相邻的小面对共点，则此多面体称为简单多面体。简单多面体将空间划分为不相交的两部分(有界的内部域和无界的外部域)。一般说来，多面体是指边界和内部域的并。