第六章 形体的表示及其数据结构

第一节 二维形体的表示
第二节 三维几何模型
第三节 分形
第四节 粒子系统

第六章 形体的表示及其数据结构

与空间任意形体有关的信息可以分为

1. 图形信息：点、线、面的位置，相互关系及大小等
2. 非图形信息：颜色、亮度、质量、体积。

图形信息包括

1. 几何信息：形体在空间的位置和大小
2. 拓扑信息：组成形体各部分的数目及相互间的连接关系。

形体的表示方法通常可分为两类：

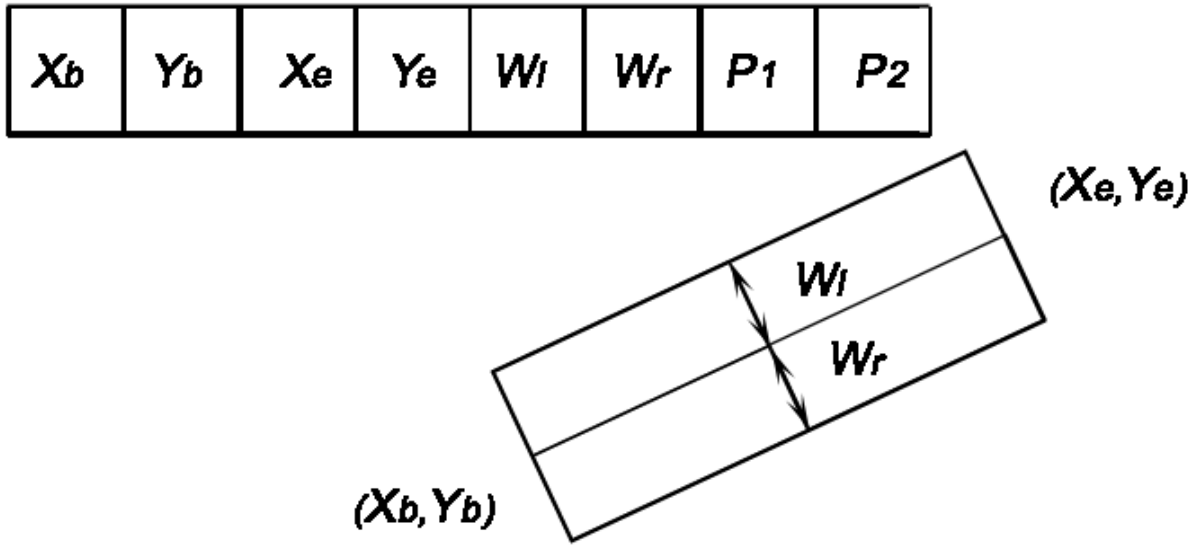
1. 边界表示：用边界将形体分为内部和外部。例如曲线曲面
2. 空间分区表示：描述形体的内部性质，将包含形体的空间区域划分为一组小的非重叠的连续实体。（四叉树和八叉树）

第一节 二维形体的表示

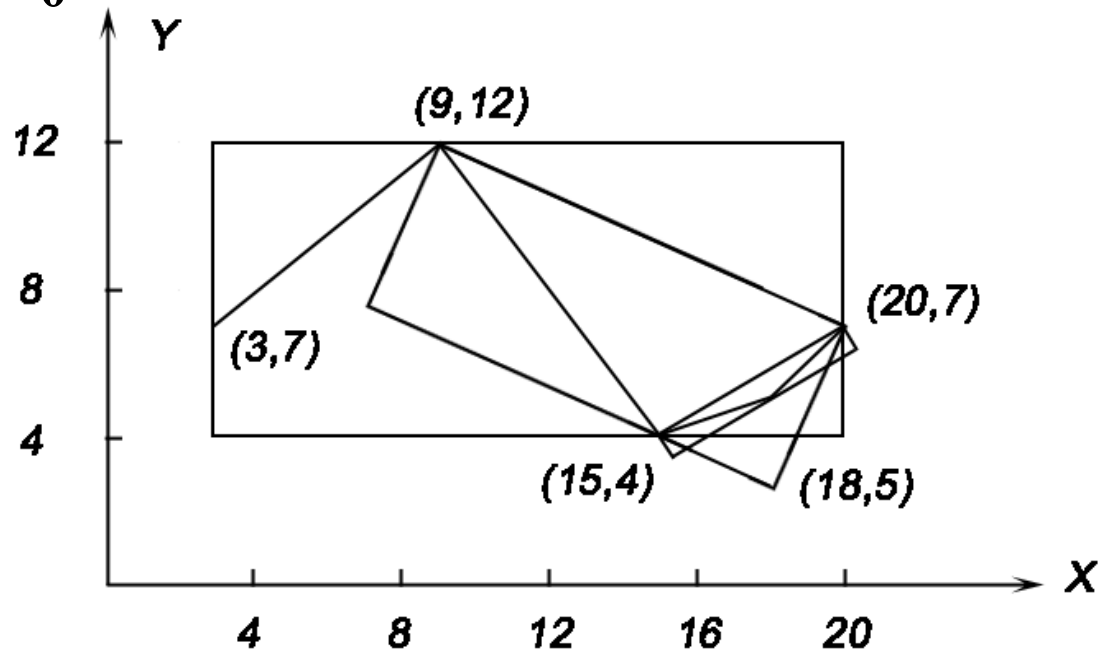
- **二维图形的边界表示**

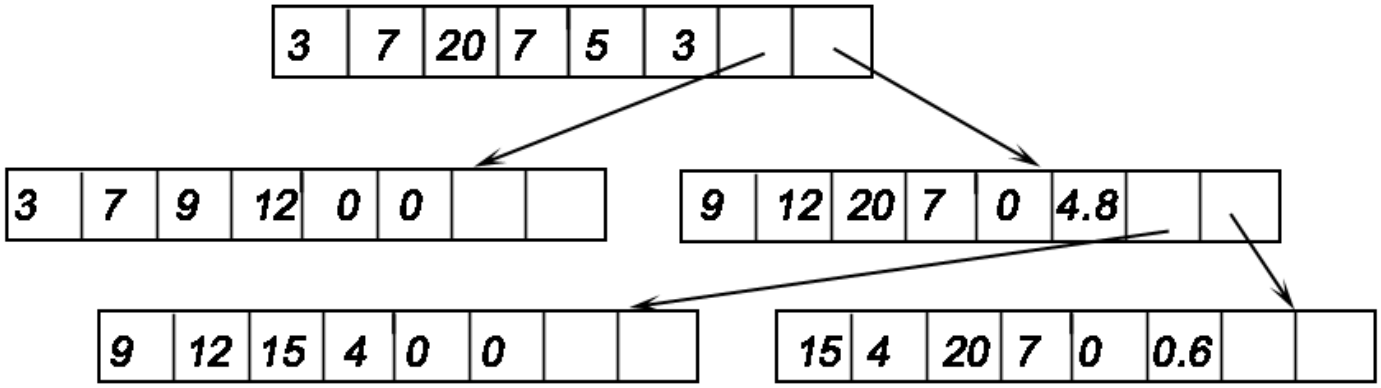
  带树法

  带树是一棵二叉树,树的每个结点对应一个矩形带段,这样每个结点可由八个字段组成,前六个字段描述矩形带段,后二个是指向两个子结点的指针, 即矩形带段的起点是(x_b,y_b),终点是(x_e,y_e)。相对从起点到终点的连线,矩形有两边与之平行,两边与之垂直,平行两边与之距离分别为w_l和w_r。



设表示曲线有5个点(3,7)(9,12),(15,4),(18,5),(20,7)，取分辨率w_0=1,则上述算法构造的带树





设要表示的曲线是由经过适当选取已确定的一组离散点$P_0,P_1,\ldots,P_n$序列给出,则生成表示曲线的分辨率为w_0的带树的算法,可简略描述如下:

```
BINARY   *Create(float *P,int i,int j，float W){
/* BINARY带树节点类型,P[i]至P[j]描述折线表示的曲线,W为分辨率*/
    Search(P,i,j,wl,wr); //确定P[i]至P[j]所有点所形成的矩形带段的宽度
    root=new(BINARY);    //获取带树节点
    CBINARY(root,wl,wr,P,i,j);  //构造根节点
    if (wl+wr<=W) return root; //返回带树根节点
    else {
          k = maxdis(P,i,j);//找出距Pi与Pj连线垂直距离最远的点Pk
          t1= Create( P,i,k, W);  //构造P[i]至P[k]点间的带树
          t2= Create( P,k,j, W);  //构造P[k]至P[j]点间的带树
          Left(root,t1);   //t1作为root左子树
          Right(root,t2); // t2作为root右子树
          return(root);//
    }
}
```

带树法解决曲线的问题：

①以不同的分辨率显示用带树表示的曲线

设给出允许的分辨率为w,表示曲线的带树的分辨率为w_0,并设$w_0 \le w$,则显示算法如下:

(1)根结点

若当前正考查结点的$W=w_l+w_r \le w$,

则显示该结点对应的矩形带段;

否则,即$W > w$

则转去分别考查该结点的左右两个子结点,

(2)对子结点做同样的处理。


```
void  Display(BINARY *root, float W){
 //BINARY带树节点类型root带树根指针, W为显示分辨率
        if(Width(root)<=W){  //wl与wr之和函数
                DisplayLine(root); //显示带树为两端点线段
                return;
        }else {
                Display(root->left,W);  //显示左子树
                Display(root->right,W); //显示右子树
                return;
        }
}
```

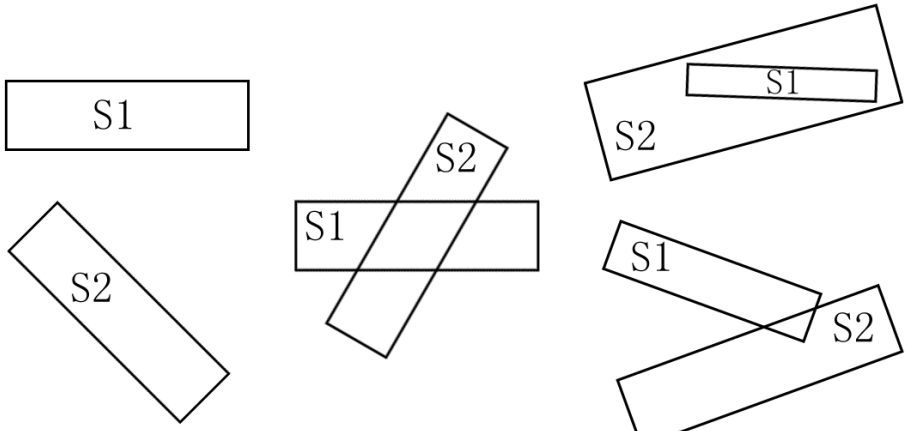
②利用带树求曲线相交

两个矩形带段S_1和S_2的位置关系有如下三种：

(1) 不相交。

(2) 良性相交,即S_1的与起点至终点连线平行的两条边都与S_2相交,S_2的与起点至终点连线平行的两条边也都与S_1相交。

(3) 可能性相交,这时不是良性相交,但也不是不相交。



(1)不相交 （2）良性相交 （3）可能性相交

设表示要求交两曲线的带树已构造得足够精确,即在树叶一层,来自不同带树的矩形带段或是不相交或是良性相交,而没有可能性相交出现。

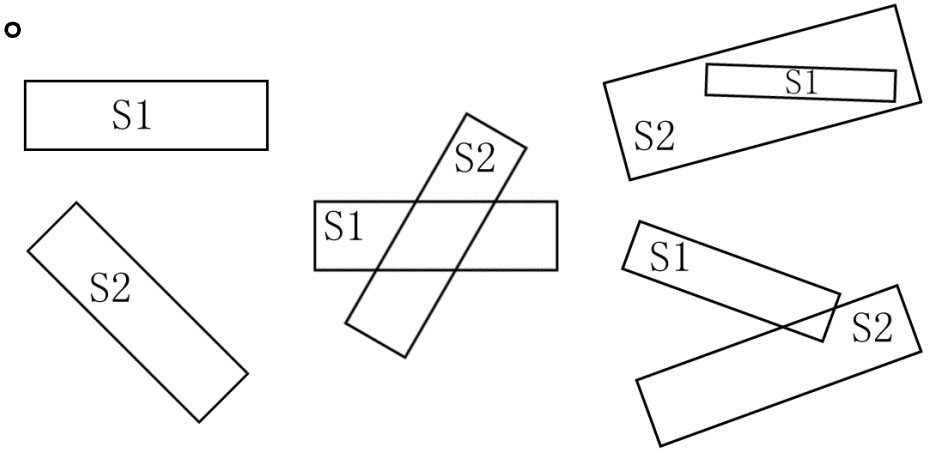
两树T_1和T_2表示的两条曲线是否相交的算法叙述如下:

算法:

1.若T_1和T_2对应的矩形带段互不相交,那么它们代表的曲线不相交;

2.若T_1和T_2对应的矩形带段良性相交,那么它们代表的曲线相交;

3.若T_1和T_2对应的矩形带段可能性相交,且T_1的面积大于或等于T_2的面积,那么分别执行T_2与T_1的左右两个儿子结点的相交性检查。



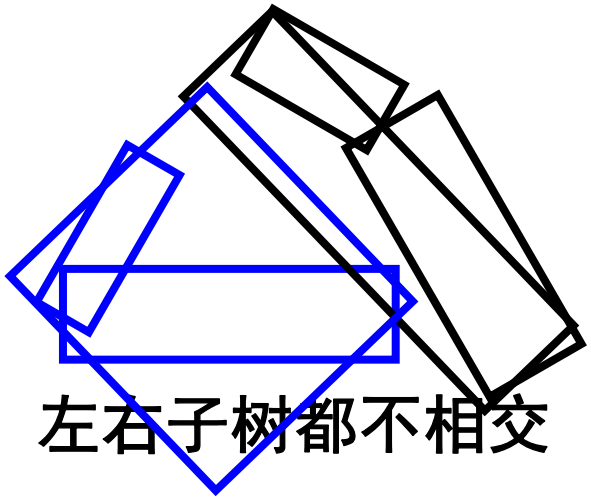
(1)不相交 (2) 良性相交 (3) 可能性相交

4.若T_1的面积小于T_2的面积,则把它们位置对换一下再如上进行两个检查。

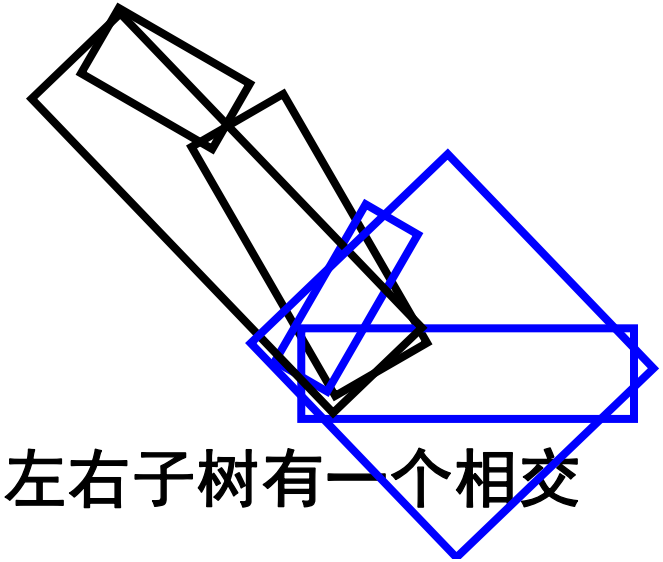
4.1.若左右两个子节点检查的结果都是不相交,则认为所表示曲线不相交;

4.2.若左右两个子节点检查中有一个是良性相交,则认为所表示曲线相交;

4.3.若不是上述两情形,即出现可能性相交,则对可能性相交的两个矩形带段中面积较大者,取其对应结点的两个子结点,如此进行可直到树叶那一层。



左右子树都不相交

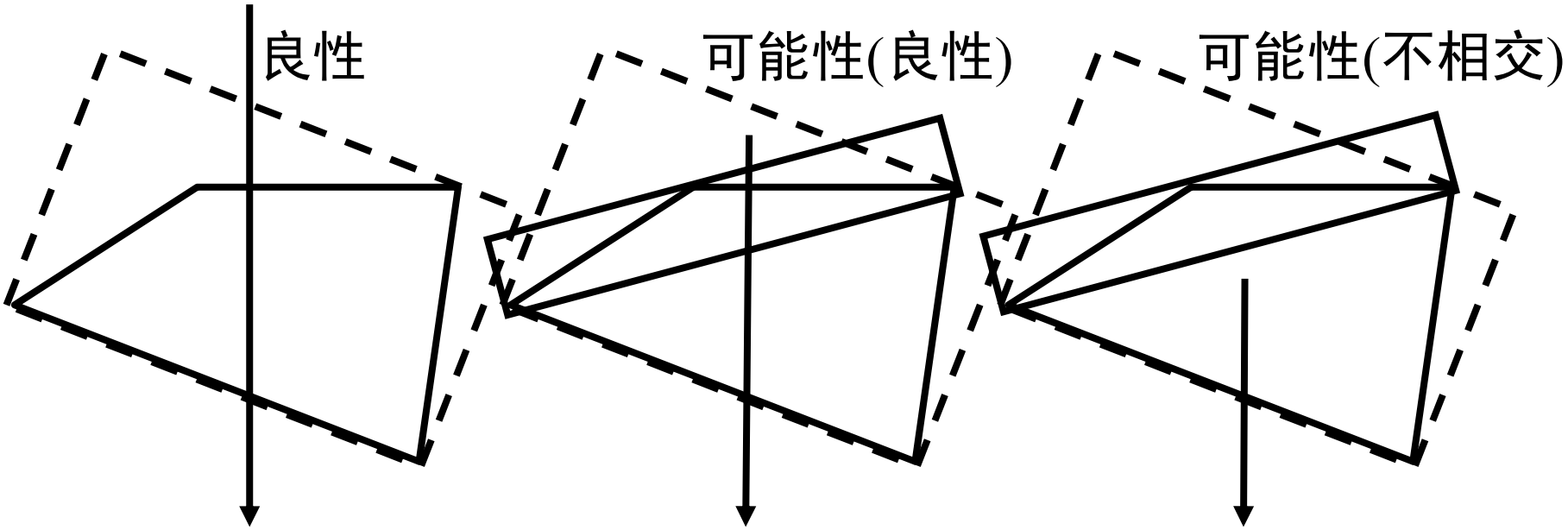


左右子树有一个相交

③ 如何确定一点是否在一封闭曲线内

从该点引射线与区域边界相交的次数为奇数，点在区域内，否则点在区域外。

判别两曲线相交：一条射线（宽度为0的带树，与表示曲线的带树求交，交点的个数即为带树良性相交次数。



带树法的优点：

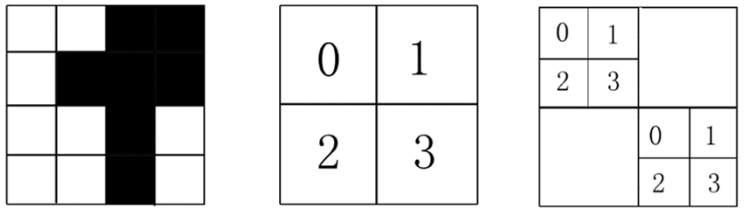
1.用带树法表示曲线对提高计算效率是有帮助的。

2.两个带树对并、交等运算是封闭的。

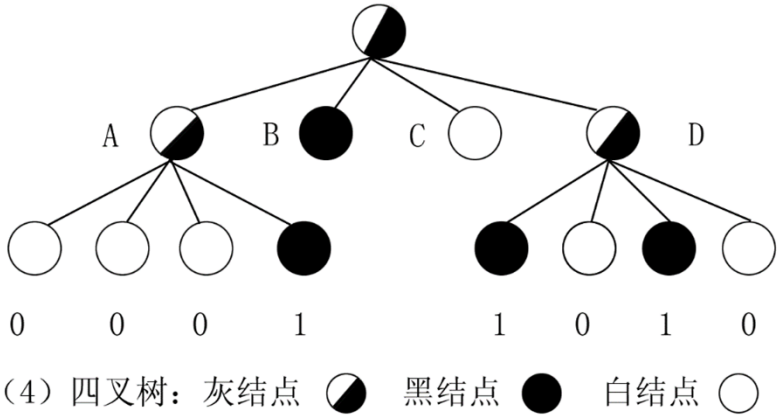
3.与用像素阵列来表示图形的方法相比，节省空间需求。

二.平面图形的四叉树表示方法（表示实区域，不是边界）

假定一个平面图形是黑白的二值图形,即组成图形像素阵列的仅有黑色像素值1,白色像素值0,设表现图形的像素阵列由$2^n \times 2^n$个像素组成。



(1)原图形 （2）第一层分等分 （3）第二层等分



（4）四叉树：灰结点 黑结点 白结点

表示该图形的四叉树结构可以如下形成:

1.图形包含于$2^n \times 2^n$的正方形中，这个正方形是四叉树的根结点。

1.1若图形整个地占据这个正方形,则图形就用该正方形表示,

1.2否则将该正方形均分为四个小正方形，每个小正方形边长为原正方形边长的一半.它们是根结点的四个子结点,可编号为0,1,2,3。

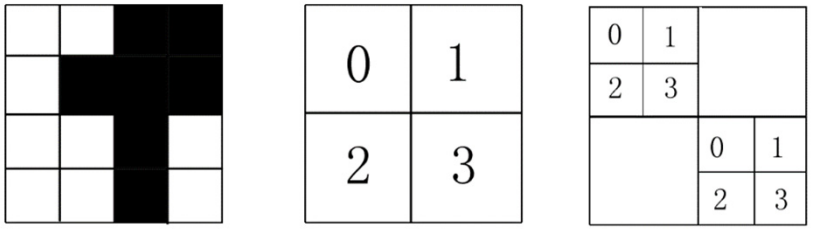
2.每个小正方形

2.1若完全被图形占据，则标记相应结点为1，称为黑结点。

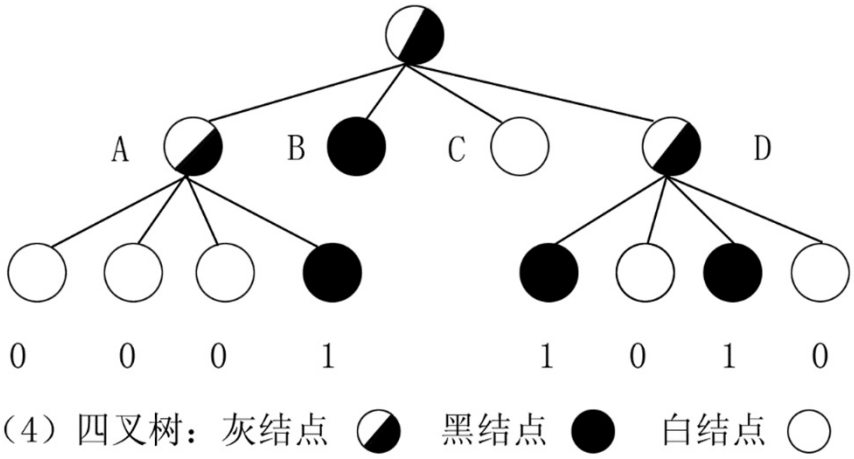
2.2若与图形完全不相交，则标记相应结点为0，称为白结点。

2.3若不是上述两情形，即与图形部分相交，称为灰结点，并将其一分为四。

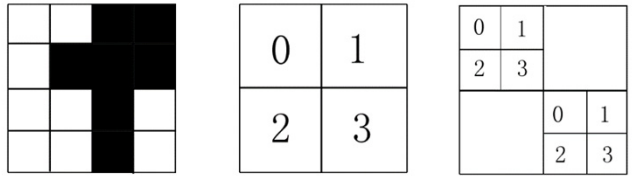
3.当再分生成小正方形边长达到一个像素单位时，再分终止，此时一般应将仍是灰结点的改为黑结点，如此形成了平面图形的四叉树表示。



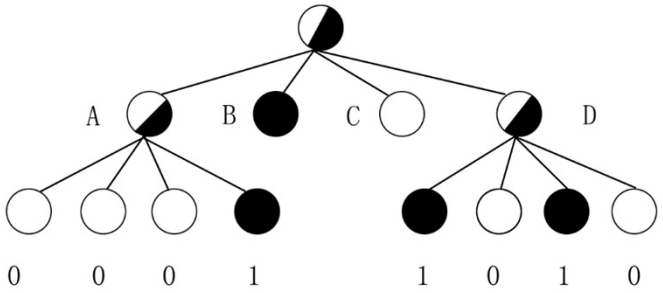
(1)原图形 （2）第一层分等分 （3）第二层等分



```
Tree CreateQuadtree (V,C,n)//V是栅格图形，C是正方形，n表示正方形边长对应的层次
{       if(intersect(V,C,n)==C) {
                new(P); P->V=1;return(P);//构造黑结点
        }else  if(intersect(V,C,n)==NULL){
                new(P); P->V=0; return NULL; //构造白结点
        }else //构造灰结点
                if(n==1) {//正方形边长为1
                        new(P) ; P->V=1;return(P);
                }else{//正方形边大于1
                        new(P); P->V=0.5;
                        C0=C.0;C1=C.1;C2=C.2;C3=C.3; //将正方形边均分为四块
                        P->F0= CreateQuadtree (V,C0,n-1);
                        P->F1= CreateQuadtree (V,C1,n-1);
                        P->F2= CreateQuadtree (V,C2,n-1);
                        P->F3= CreateQuadtree (V,C3,n-1);
                        return(P);
                }
```



原图形 （2）第一层分等分 （3）第二层等分



（4）四叉树：灰结点 黑结点 白结点

V(0/1/0.5)	F0	F1	F2	F3


```
TREE * tree_4(Box b, Graph g){
//b为正方形 g为二维形体 TREE为四叉树结点类型
        if (g包含b) {
                构造树根结点root（属性为黑）;return(root);
        }else if (b与g之交集为空){
                return null;
        }else {
                b分成b0,b1,b2,b3四个相同小正方形;
                构造树根结点root（属性为灰）;
                r1=tree_4(b0,g); r2=tree_4(b1,g);
                r3=tree_4(b2,g); r4=tree_4(b3,g);
                r1,r2,r3,r4链接为root结点的四个子结点;
                return(root);
        }
}
```

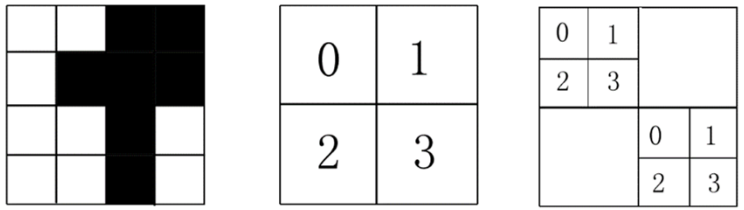
四叉树的存储方式：即规则方式、线性方式和一对四方式

1.规则四叉树

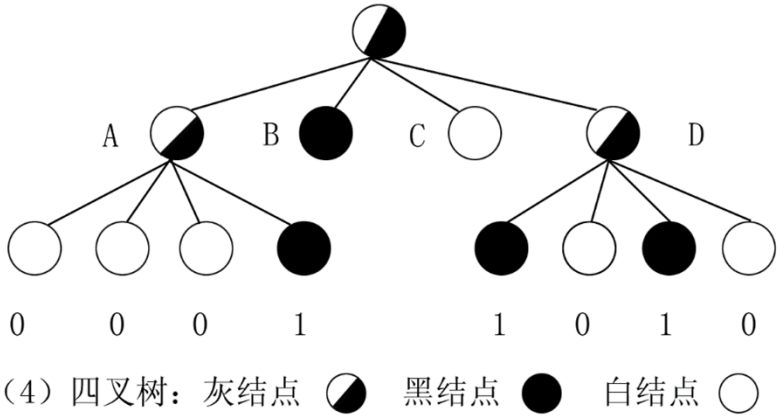
用五个字段的记录来表示树中的每个结点

一个用来描述结点的特性，即是灰、黑、白三类结点中的一种。其余四个用于存放指向四个子结点的指针。

缺点：大量的存储空间被指针占用，存储空间利用率差



(1)原图形 （2）第一层分等分 （3）第二层等分



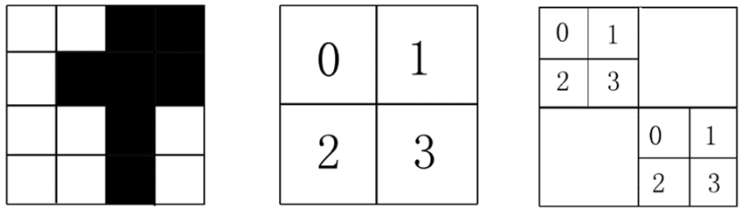
（4）四叉树：灰结点 黑结点 白结点

2.线性四叉树

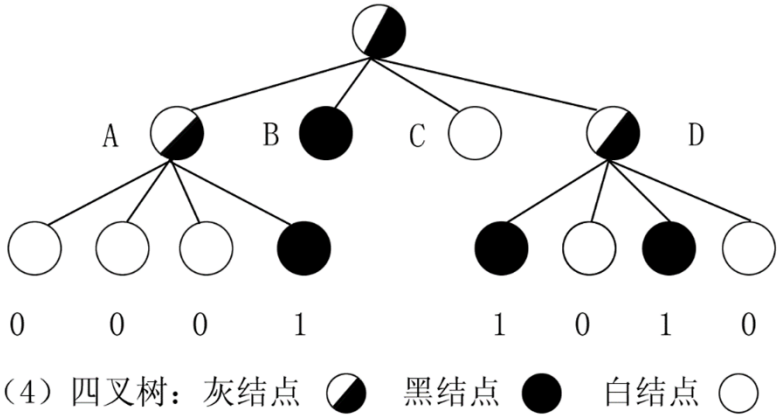
以某一预先确定的次序遍历四叉树形成一个线性表结构。

R’A’abcdBCD’efgh。其中R表示根，字母右上角加’表示是灰结点。

优点：节省存储空间空间　缺点：灵活性差



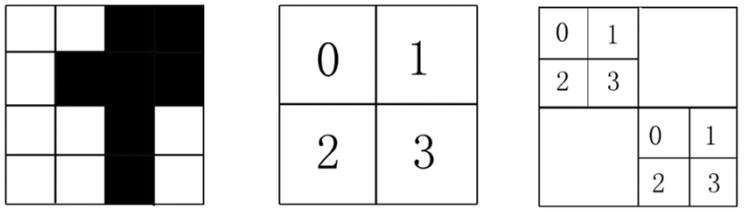
(1)原图形　（2）第一层分等分　（3）第二层等分



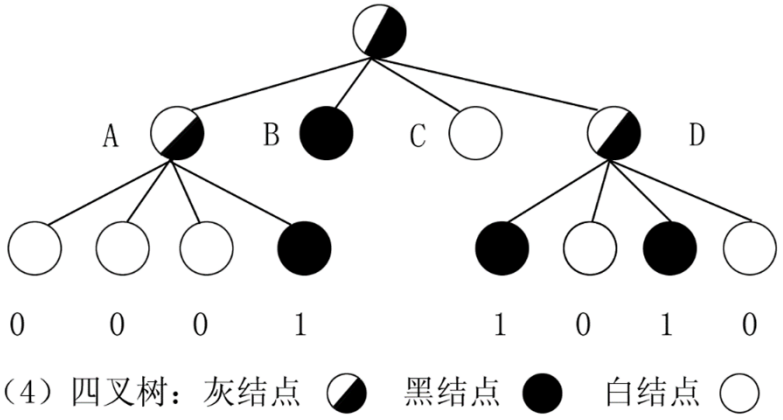
（4）四叉树：灰结点　黑结点　白结点

3.一对四式四叉树

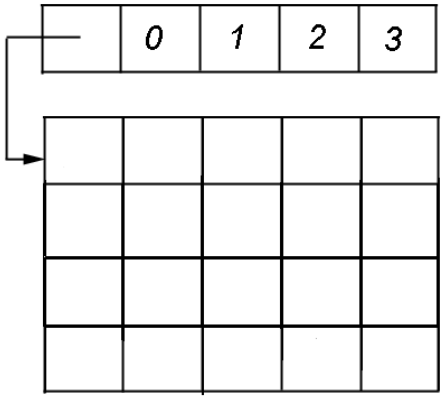
一对四式四叉树的存储结构 每个结点有五个字段，其中四个字段用来描述该结点的四个子结点的状态，另一个结点存放指向子结点记录存放处的指针。四个子结点对应的记录是依次连续存放的。

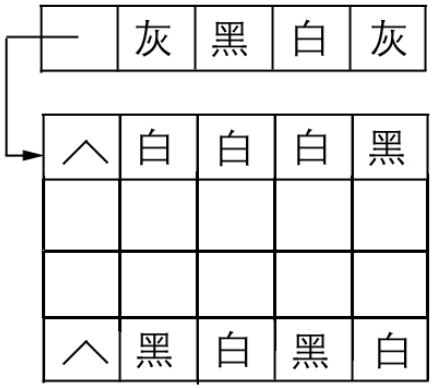


(1)原图形 (2) 第一层分等分 (3) 第二层等分



(4) 四叉树：灰结点 黑结点 白结点

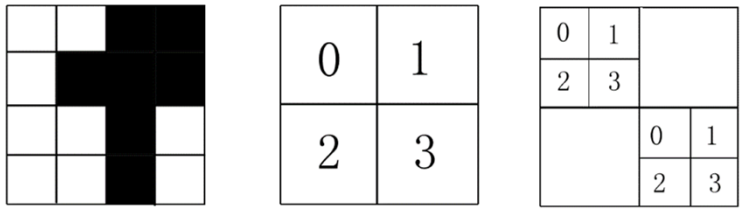




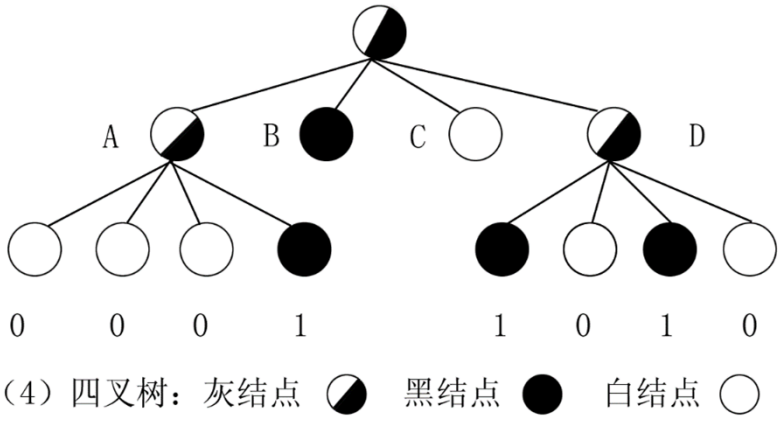
改进的一对四式四叉树(为节省存贮空间):

a.增加计算量（紧凑的一对四式四叉树）

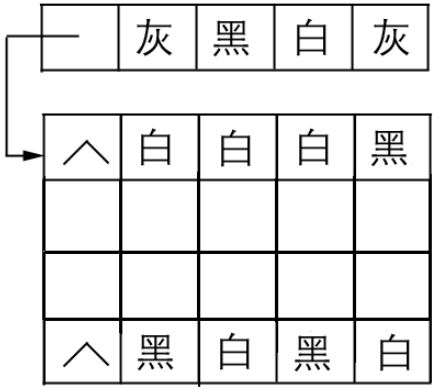
存取相应节点时，先检查其父节点，看在它之前有几个叶节点

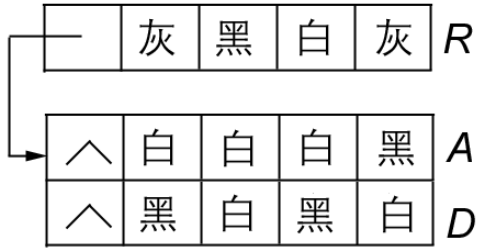


(1)原图形 （2）第一层分等分 （3）第二层等分



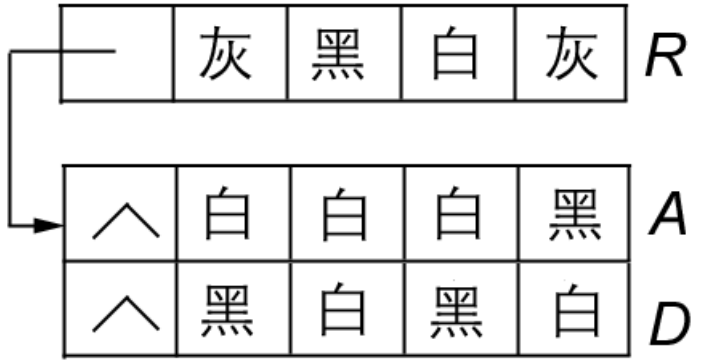
（4）四叉树：灰结点 黑结点 白结点





b.一个记录增加一个字节

该字节一分为四,每个子结点对应2位,表示它的子结点在指针指向区域中的偏移。



01|00|00|00

第二节 三维几何模型

一. 几何元素

形体的模型主要指的就是包含图形信息所形成的模型。

形体本身的构造有一定的层次性，低层部分组合构成上一层部分，而上一层部分组合又可以构成更高一层的部分，依此类推可形成多层结构。其中，每一层中的部分，称为几何元素。

(1)点

点是0维几何元素，有端点、交点、切点、孤立点等形式。

曲线、曲面的应用中会涉及到三种类型的点：

型值点 相应曲线、曲面必然经过的点。

控制点 相应曲线、曲面不一定经过的点，仅用于确定位置和形状。

插值点 在型值点之间插入的一系列点，用于提高曲线曲面的输出精度。

不同的空间中点的表示方式

(1)一维空间中用一元组{t}表示；

(2)二维空间中用二元组{x,y}或{x(t),y(t)}表示；

(3)三维空间中用三元组{x,y,z}或{x(t),y(t),z(t)}表示；

点是几何造型中的最基本的元素，曲线、曲面和其它形体都可以用有序的点集描述。

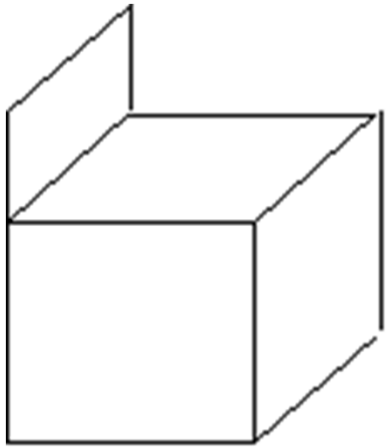
用计算机存储、管理、输出形体的实质就是对点集及其连接关系的处理。

(2)边

边是一维几何元素，是两个邻面（正则形体）或多个邻面（非正则形体）的交界。边分直线边和曲线边。直线边由起点和终点两端点确定；曲线边由一系列型值点或控制点表示，也可以用显式、隐式方程描述。

正则形体：满足欧拉公式V-E+F=2的形体（例如立方体V=8,E=12,F=6）。

非正则形体：不满足欧拉公式的形体。存在悬面、悬边的形体是非正则形体。

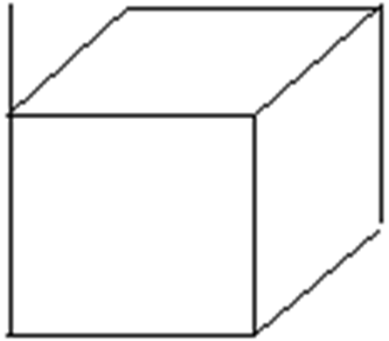


(a)有悬面

V=10

E=13

F=6

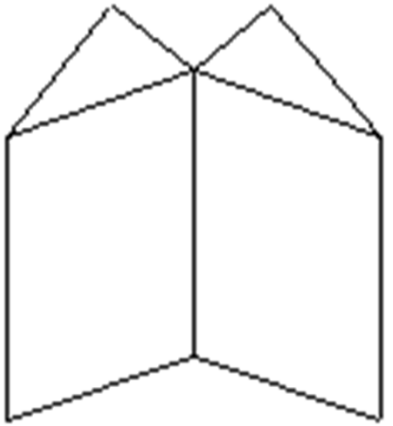


(b)有悬边

V=9

E=12

F=6



(c)一条边有两个以上的邻面（不连通）

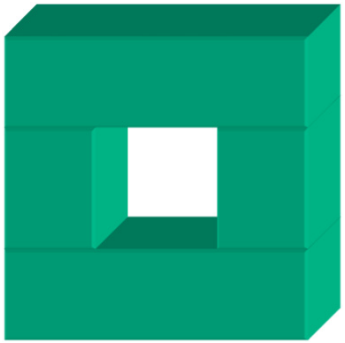
V=10

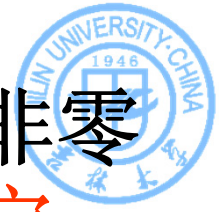
E=17

F=10

(3)环

环是有序有向边（直线段或曲线段）组成的面的封闭边界。环中的边不能自相交，相邻两条边共享一个端点。环有内外之分，确定面的最大外边界的环称之为外环，通常其边按逆时针方向排序。而把确定面中内孔或凸台边界的环称之为内环，其边相应外环排序方向相反，通常按顺时针方向排序。





(4)面

面是二维几何元素，是形体上一个有限、非零的区域，它由一个外环和若干个内环所界定。面有方向性，一般用其外法向量作为该面的正向。若一个面的外法向量向外，此面为正；否则，为反向面。

(5)体

体是三维几何元素，由封闭表面围成的空间，它是欧氏空间R^3中非空、有界的封闭子集，其边界是有限面的并集。

(6)体素

体素是可以用有限个尺寸参数定位和定型的体，常有下面三种定义形式。

一组单元实体　长方体、圆柱体、圆锥体、球体。

扫描体　由参数定义的一条（一组）截面轮廓线沿一条（一组）空间参数曲线作扫描运动而产生的形体。

用代数半空间定义的形体　在此半空间中点集可定义$\{(x,y,z)|f(x,y,z)\leq 0\}$此处的f应是不可约的多项式。

形体的层次结构

点→边→环→面→外壳→形体。

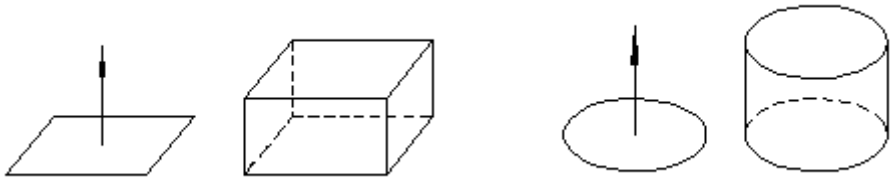
❑ 扫描体

扫描法的基本思想非常简单：“运动的物体”加上“轨迹”。
常用的扫描方式有：平移式、旋转式和广义式。

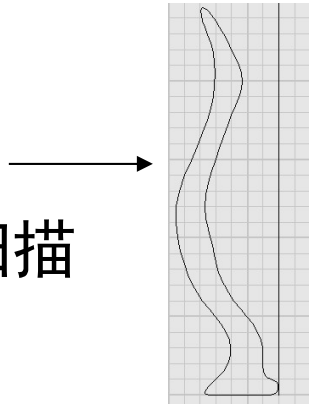
✓平移扫描：沿垂直于二维集合进行扫描；

✓旋转扫描：绕某一轴线旋转某一角度；

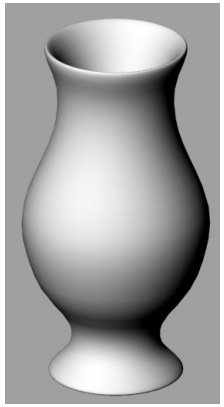
✓广义扫描：二维几何集合沿一条空间曲线的集合扫描；

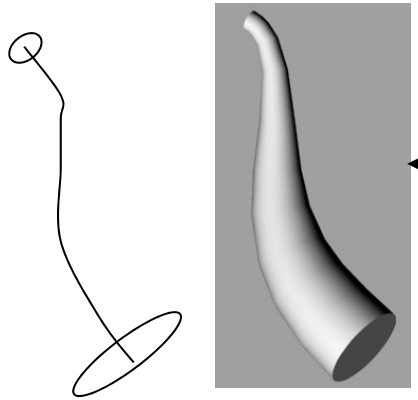


← 平移扫描



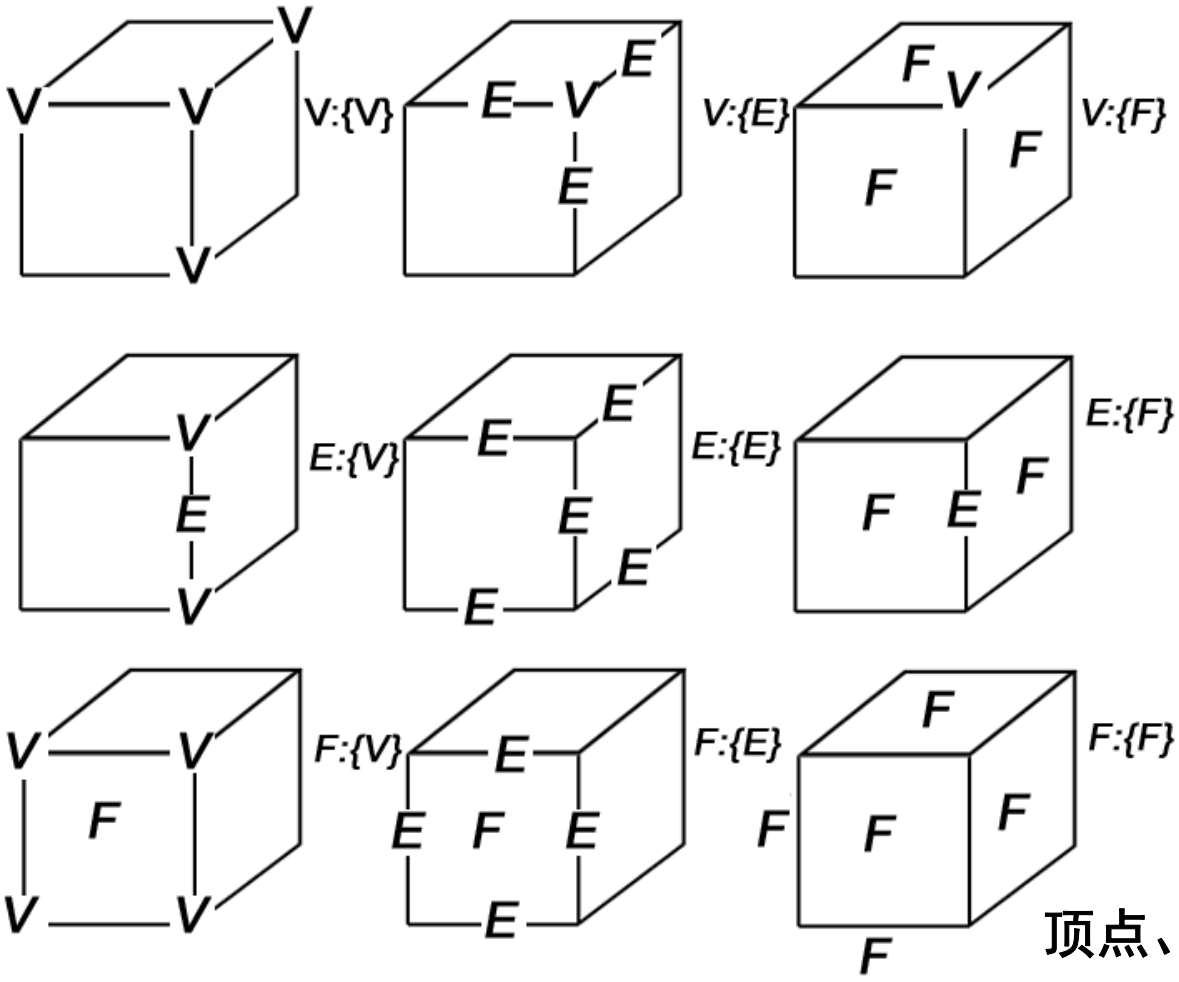
旋转扫描





广义扫描

要用实体的边界信息表示一个实体，必须同时表示出实体边界的拓扑和几何信息。物体的拓扑信息指物体上所有的顶点、棱边、表面间是怎样连接的。就多面体而言，其顶点、棱边、表面之间的连接关系可以用九种不同的形式予以描述。



数据结构中保存的拓扑关系越多，对多面体的操作越方便，但是占用的存储空间也就越大。因此要根据实际情况选择拓扑关系，以提高系统的整体效率。

顶点、棱边、表面之间的拓扑关系

二.线框、表面及实体表示（多面体的表示）

- 常用的多面体表示法是三表表示法，即采用三个表:

  顶点表：顶点坐标,用来存放多面体各顶点的坐标;

  边表：两个顶点编号表达一条边,指出哪两个顶点之间有多面体的边

  面表：边的编号表示面,指出哪些边围成了多面体的表面

- 多面体的表示

  线框模型：点表，边表

  表面模型：点表，边表，面表

  实体模型：点表，边表，面表，其它图形和非图形信息

  形体确定➡三表确定

  给出三表➡不一定表示一个真实形体(正则形体：**V-E+F=2**)

- 真实形体应满足的条件

  1.顶点表中的每个顶点至少是三边的端点;

  2.边表中的每条边是两个多边形面的公共边;

  3.每个多边形面是封闭的等等。

顶点表

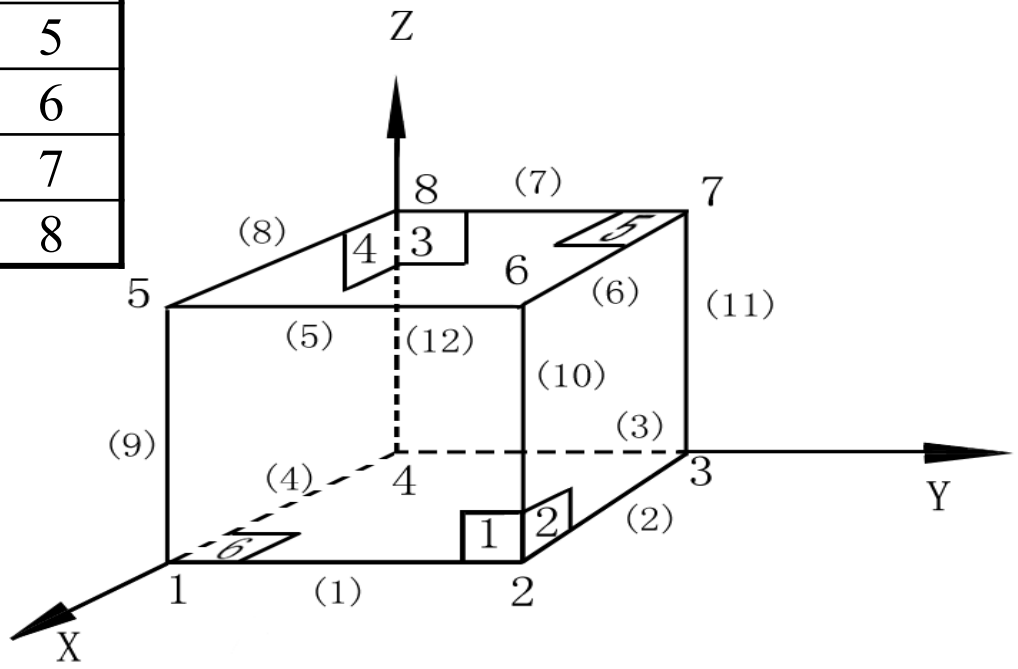
编号	X	Y	Z
1	1	0	0
2	1	1	0
3	0	1	0
4	0	0	0
5	1	0	1
6	1	1	1
7	0	1	1
8	0	0	1

边表

编号	起点	终点
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	2	6
11	3	7
12	4	8

面表

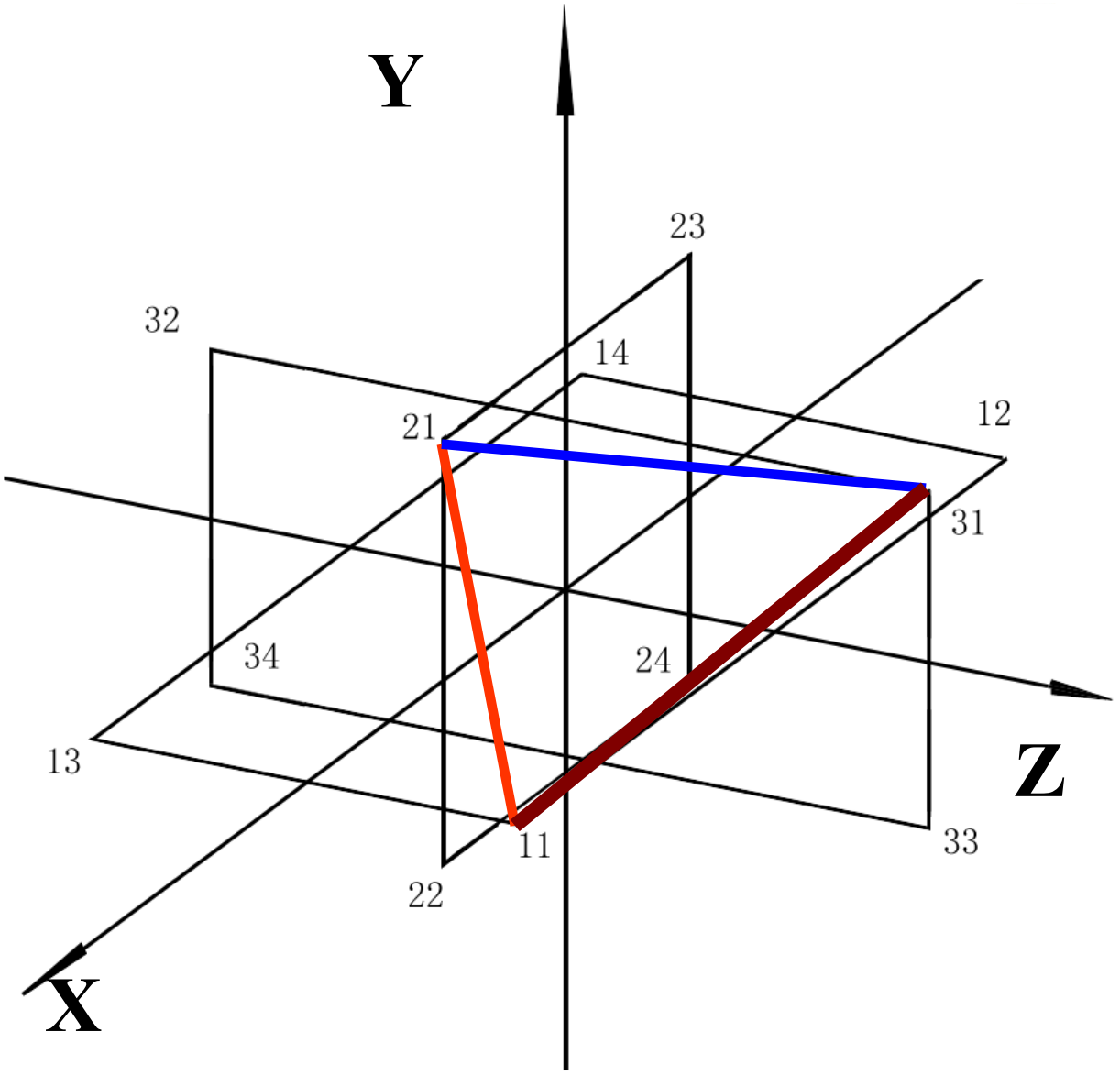
编号	边1	边2	边3	边4
1	1	10	5	9
2	2	11	6	10
3	3	12	7	11
4	4	9	8	12
5	5	6	7	8
6	3	2	1	4



空间正二十面体V_{20}, 的三表表示。

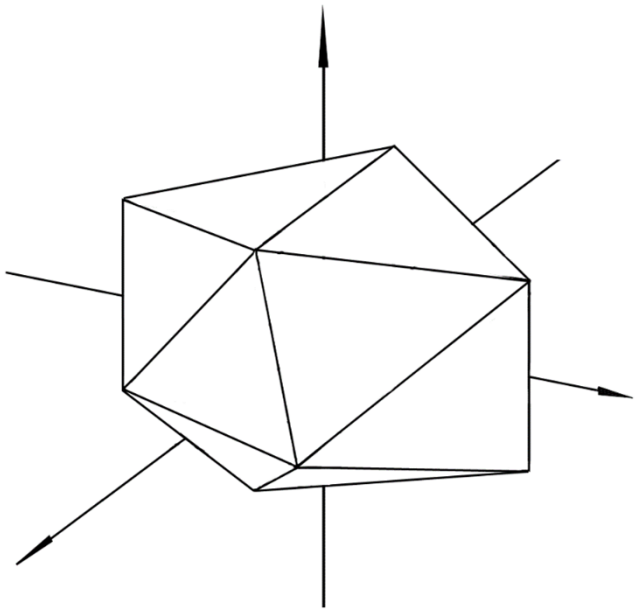
引入一个正数$\Phi>0$,它满足二次方程$\Phi^2-\Phi-1=0$,因此

$\Phi=1/2(1+\sqrt{5})\approx1.618034$。



编号	x	y	z
11	Φ	0	1
12	-Φ	0	1
13	Φ	0	-1
14	-Φ	0	-1
21	1	Φ	0
22	1	-Φ	0
23	-1	Φ	0
24	-1	-Φ	0
31	0	1	Φ
32	0	1	-Φ
33	0	-1	Φ
34	0	-1	-Φ

边编号		边编号	
1	11,13	16	21,33
2	12,14	17	22,32
3	21,23	18	22,34
4	22,24	19	23,31
5	31,33	20	23,32
6	32,34	21	24,33
7	11,21	22	24,34
8	11,22	23	31,11
9	12,23	24	31,12
10	12,24	25	32,13
11	13,21	26	32,14
12	13,22	27	33,11
13	14,23	28	33,12
14	14,24	29	34,13
15	21,31	30	34,14



面编号		面编号	
1	7，23，15	11	25，6，29
2	8，17，27	12	30，6，26
3	11，16，25	13	11，1，7
4	29，28，12	14	8，1，12
5	9，19，24	15	9，2，13
6	28，21，10	16	14，2，10
7	26，20，13	17	19，3，15
8	14，22，30	18	16，3，20
9	27，5，23	19	17，4，21
10	24，5，28	20	22，4，18

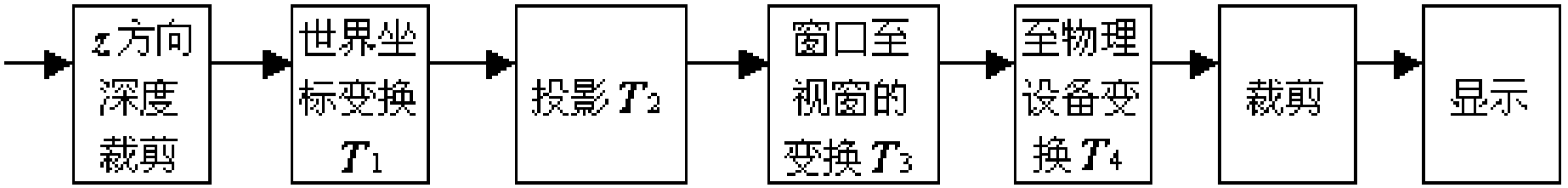


图 3.36 简单的图形处理流程

三. 三维实体表示方法

从用户角度来看，形体以特征表示和构造的实体几何表示比较适宜；从计算机对形体的存储管理和操作运算角度看，以边界表示最为实用。

1. 构造的实体几何法

构造的实体几何(CSG:Constructive Solid Geometry)法是指任意复杂的形体都可以用简单形体(体素)的组合来表示。

形体的CSG表示可看成是一棵有序的二叉树，称为CSG树。

终端结点或是体素，如长方体、圆锥等；或是刚体运动的变换参数，如平移参数T_x等；

非终端结点或是正则的集合运算，一般有交、并、差运算；或是刚体的几何变换，如平移、旋转等。

采用BNF范式可定义CSG树如下：

<CSG>::=<体素叶子>|

<CSG树><正则集合运算><CSG树>|

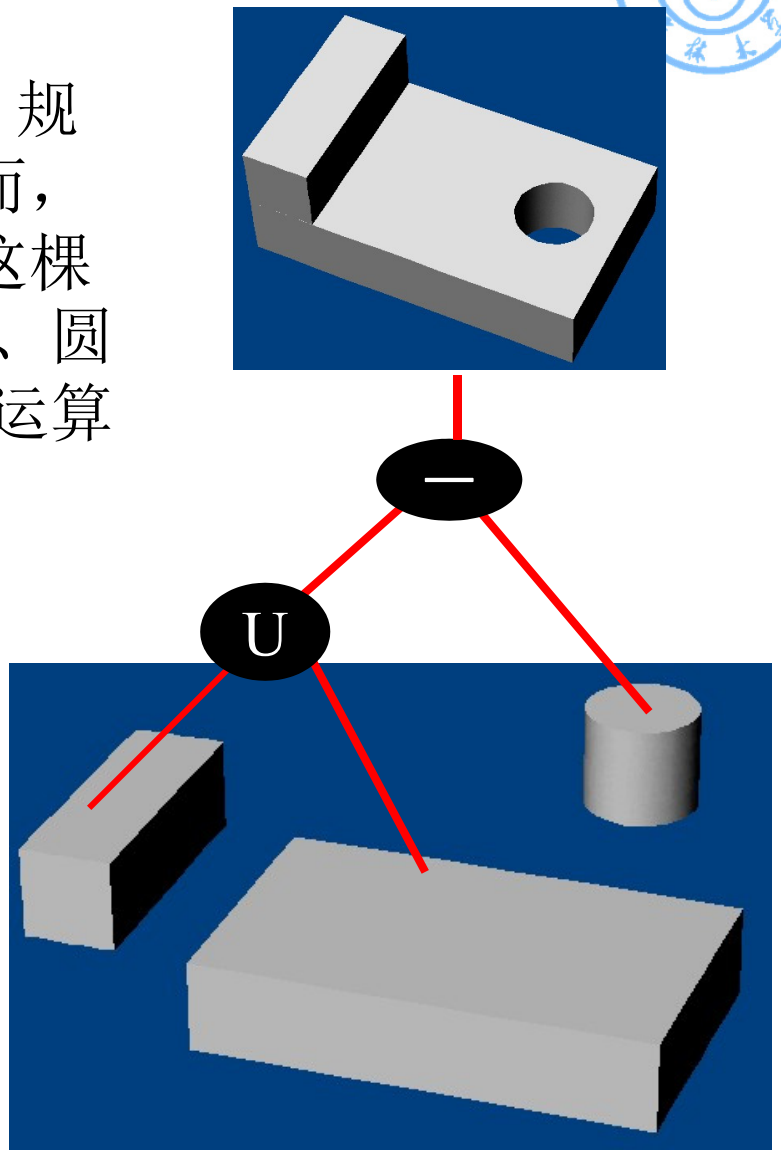
<CSG树><刚体运动><刚体运动变量>

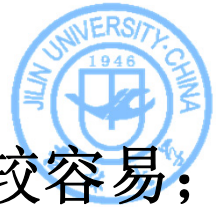
CSG树是无二义性的，但不是唯一的，其定义域取决于所用体素以及所允许的几何变换和正则集合运算算子。

❑CSG树

➢一个复杂物体可由一些比较简单、规则的物体经过布尔运算而得到。因而，这个复杂的物体可描述为一棵树。这棵树的终端结点为基本体素(如立方体、圆柱、圆锥)，而中间结点为正则集合运算结点。这棵树叫做CSG树。

CSG树





CSG表示的优点：

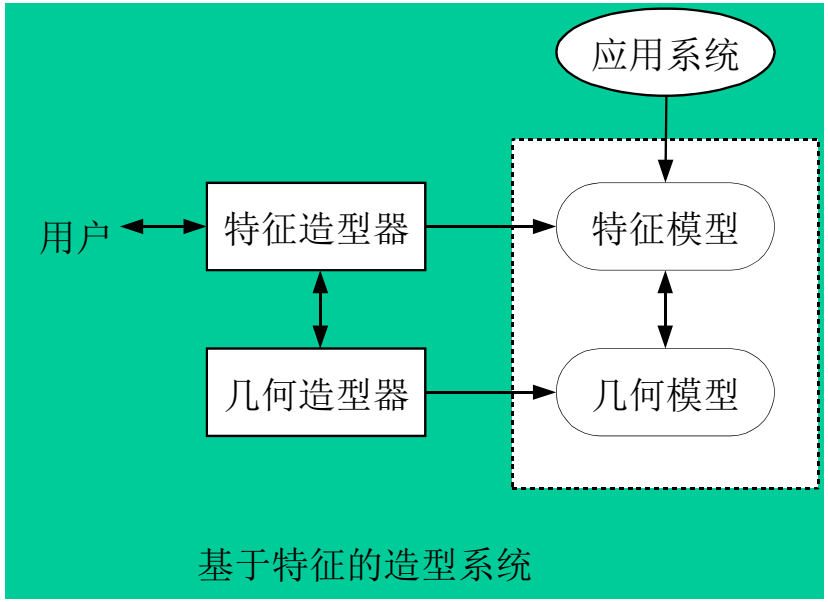
- 数据结构比较简单，数据量比较小，内部数据的管理比较容易；
- 每个CSG表示都和一个实际的有效形体所对应；
- CSG表示可方便地转换成Brep表示，从而可支持广泛的应用；
- 比较容易修改CSG表示形体的形状。

CSG表示的缺点：

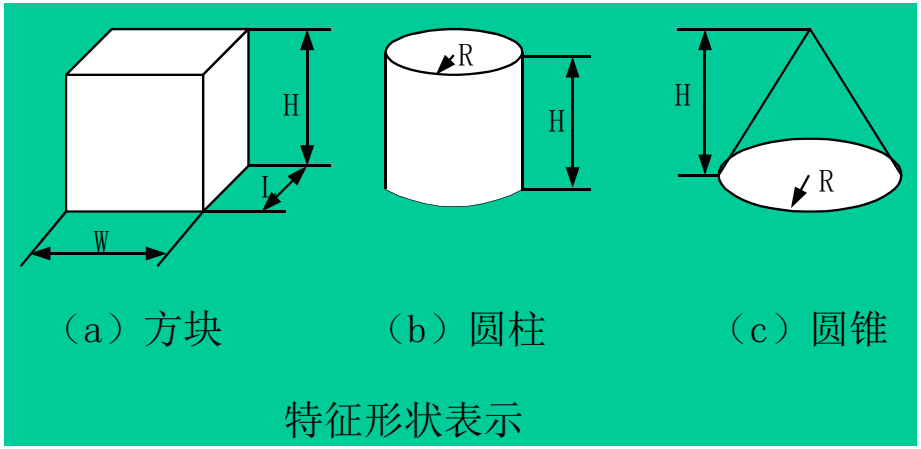
- 产生和修改形体的操作种类有限，基于集合运算对形体的局部操作不易实现；
- 由于形体的边界几何元素(点、边、面)是隐含地表示在CSG中，故显示与绘制CSG表示的形体需要较长的时间。

2 特征表示

特征表示是从应用层来定义形体，因而可以较好地表达设计者的意图，为制造和检验产品和形体提供技术依据和管理信息。从功能上看可分为形状、精度、材料和技术特征。



基于特征的造型系统



特征形状表示

2 特征表示

形状特征：体素、孔、槽、键等

精度特征：形位公差、表面粗糙度等；

材料特征：材料硬度、热处理方法等；

技术特征：形体的性能参数和特征等。

形状特征单元是一个有形的几何实体，是一组可加工表面的集合。如采用长、宽、高三尺寸表示的长方体；采用底面半径及高度表示的圆柱体均是可选用的形状特征单元。

形状特征单元的**BNF**范式可定义如下：

<形状特征单元>::=<体素>|

<形状特征单元><集合运算><形状特征单元>|

<体素><集合运算><体素>|

<体素><集合运算><形状特征单元>|

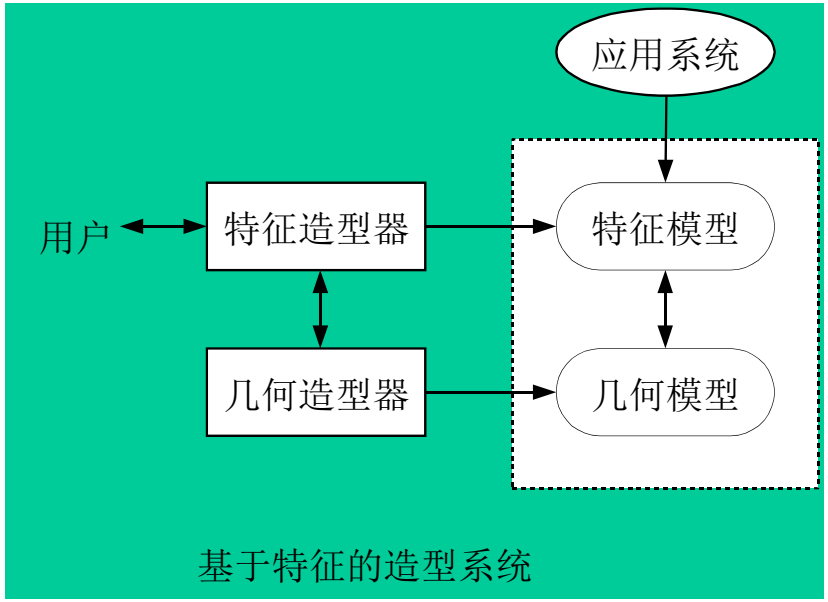
<形状特征单元><集合运算><形状特征过渡单元>;

<体素>::=长方体|圆柱体|球体|圆锥体|棱锥体|棱柱体|棱台体|圆环体|楔形体|圆角体|…;

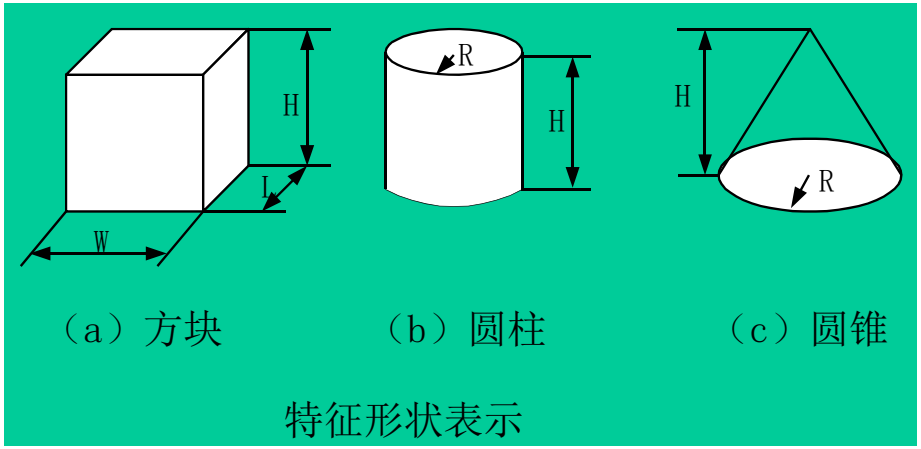
<集合运算>::=并|交|差|缩放;

<形状特征过渡单元>::=外圆角|内圆角|倒角。

特征表示



基于特征的造型系统



（a）方块　（b）圆柱　（c）圆锥

特征形状表示

3 边界表示

边界表示详细记录了构成形体的所有几何元素的几何信息及其相互连接关系—拓扑关系，便于直接存取构成形体的各个面、面的边界以及各个顶点的定义参数，有利于以面、边、点为基础的各种几何运算和操作。

形体的边界表示就是用面、环、边、点来定义形体的位置和形状。例如，一个长方体由六个面围成，对应有六个环，每个环由四条边界定义，每条边又由两个端点定义。而圆柱体则由上顶面、下底面和圆柱面所围成，对应有上顶面圆环、下底面圆环。

3 边界表示

形体的边界表示就是用面、环、边、点来定义形体的位置和形状。

按照体—面—环—边—点的层次，详细记录了构成形体的所有几何元素的几何信息及其相互连接的拓扑关系。

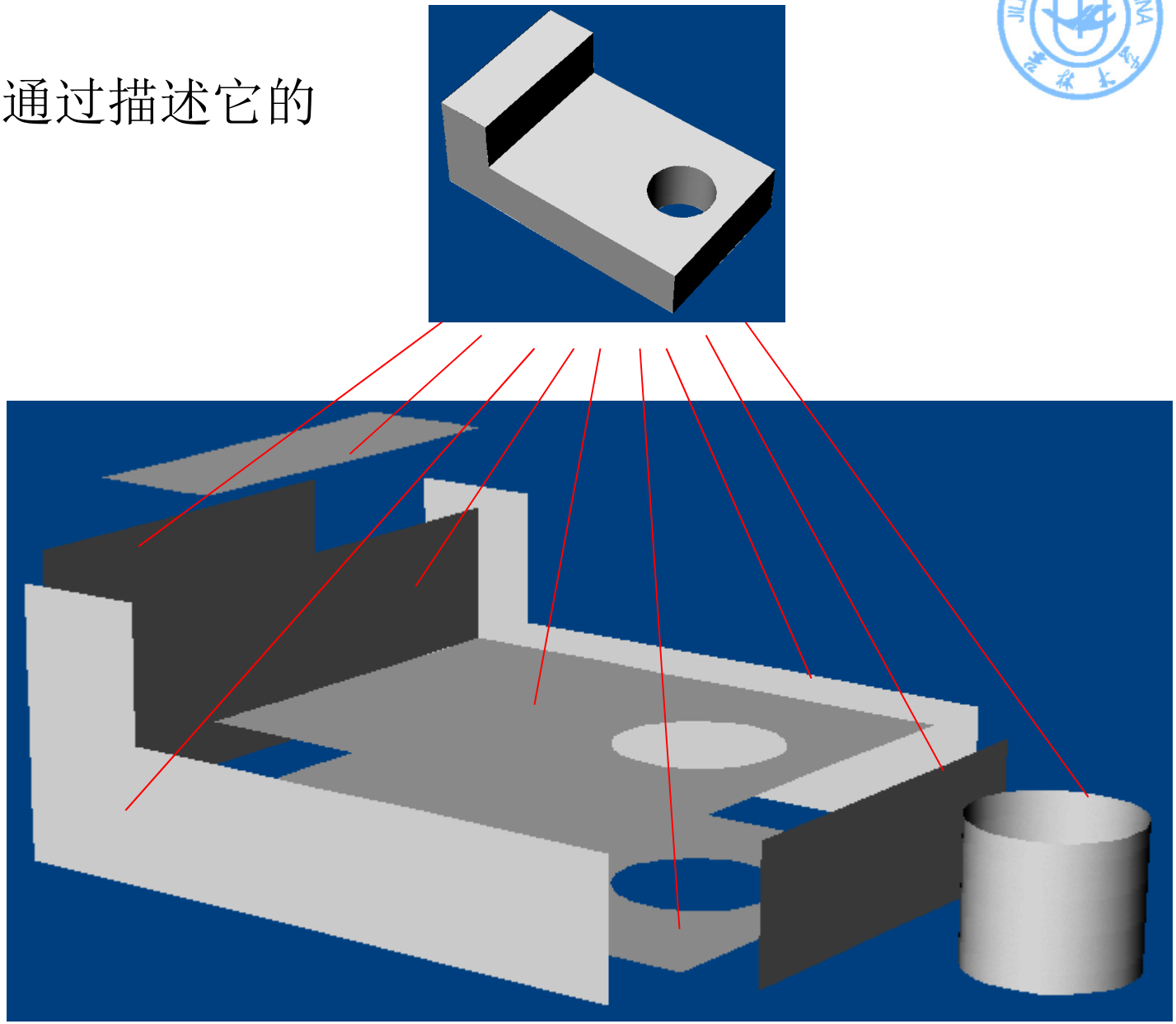
边界表示的一个重要特点是在该表示法中，描述形体的信息包括几何信息（**Geometry**）和拓扑信息（**Topology**）两个方面。

拓扑信息描述形体上的顶点、边、面的连接关系，拓扑信息形成物体边界表示的“骨架”。

形体的几何信息犹如附着在“骨架”上的肌肉。

❑ 边界表示法 （B－rep）

三维物体可以通过描述它的边界来表示。



Brep表示的优点是：

- 表示形体的点、边、面等几何元素是显式表示的，使得绘制**Brep**表示形体的速度较快，而且比较容易确定几何元素间的连接关系；
- 对形体的**Brep**表示可有多种操作和运算。

Brep表示的缺点是：

- 数据结构复杂，需要大量的存储空间，维护内部数据结构的程序比较复杂；
- 修改形体的操作比较难以实现；
- **Brep**表示并不一定对应一个有效形体，即需要有专门的程序来保证**Brep**表示形体的有效性、正则性等。

四. 八叉树

假设要表示的形体V可以放在一个充分大的正立方体C内,C的边长为2^n,形体V⊆C ,它的八叉树表示可以递归定义为:

八叉树每个结点与C的一个子立方体对应，树根就和C本身对应。

- 如果V=C,那么V八叉树仅有树根。
- 如果V≠C,则将C均分为八个子立方体,每个子立方体对应根结点的一个子结点。
- 只要某个子立方体不是完全空白或完全被V所占据,它就要被八等分,从而它对应的结点也有了八个子结点。
- 这样的递归判断及可能分割一直进行,直到结点对应的立方体或完全空白,或完全被占据,或其大小已是预先规定的体素大小.

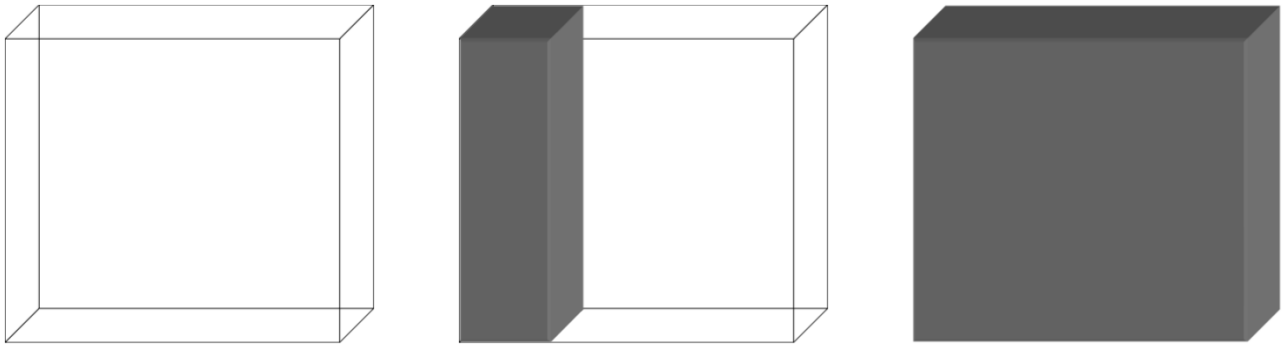
```
Tree CreateOctree (V,C,n)//V是形体，C是立方形，n表示正方形边长对应的层次
{       if(intersect(V,C,n)==C) {
                new(P); P->V=1;return(P);//构造黑结点
        }else  if(intersect(V,C,n)==NULL){
                P->V=0; return NULL; //构造白结点
        }else //构造灰结点
                if(n==1) {//正方形边长为1
                        new(P) ; P->V=1;return(P);
                }else{//正方形边大于1
                        new(P); P->V=0.5;
                        C0=C.0;C1=C.1;C2=C.2;C3=C.3; //将立方体均分为八块
                        C4=C.4;C5=C.5;C6=C.6;C7=C.7;
                        P->F0= CreateOctree (V,C0,n-1);    P->F1= CreateOctree (V,C1,n-1);
                        P->F2= CreateOctree (V,C2,n-1);    P->F3= CreateOctree (V,C3,n-1);
                        P->F4= CreateOctree (V,C4,n-1);    P->F5= CreateOctree (V,C5,n-1);
                        P->F6= CreateOctree (V,C6,n-1);    P->F7= CreateOctree (V,C7,n-1);
                        return(P);
                }
}
```

这时对它与V之交作一定的“舍入”,使体素或认为是空白,或认为是被V占据的。这里所谓的体素,就是指被分割后得到的小立方体。

对应立方体被形体V完全占据的结点为黑结点,

完全不占据的为白结点,

部分被占据的为灰结点。



存贮结构,有常规的、线性的、一对八式的八叉树等等。

优点：

1.占用存储空间少

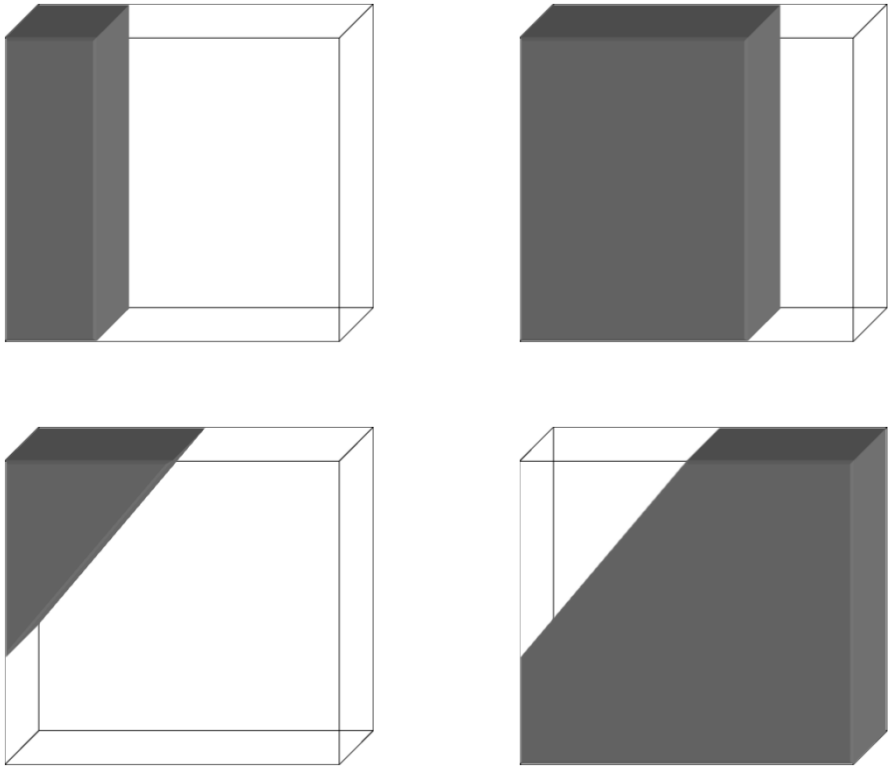
2.便于形体的交并差的集合运算

3.可以显式不同精度的实体，消隐便于实现

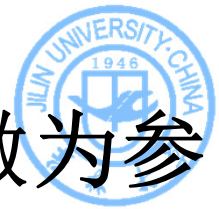
八叉树表示的三维形体的几何变换

1.比例变换

2.旋转变换　相对通过原点的一条任意方向的直线做任意角度的旋转变换。



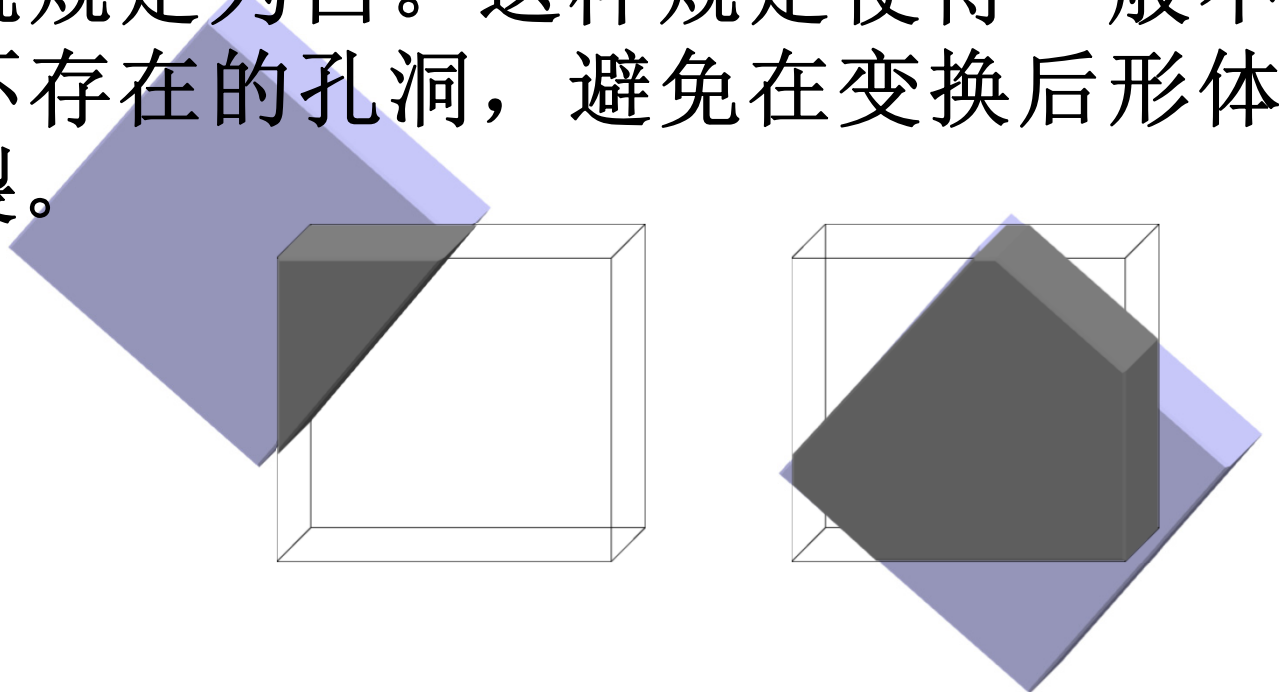
多次变换误差会积累，解决方法

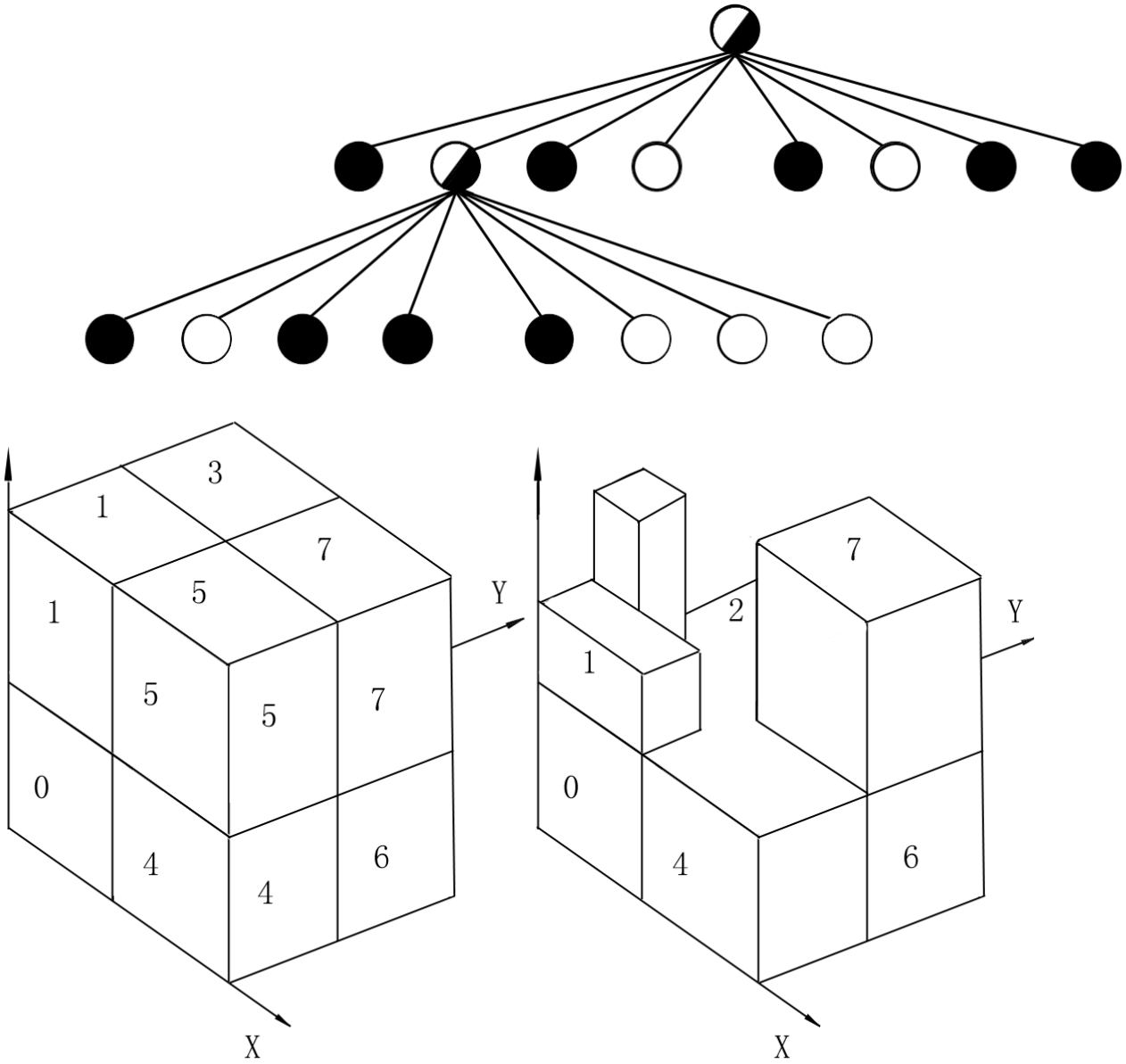


第一项措施是保持一个原始的八叉树做为参考的源树。设指定了一次变换$\mathbf{R}_1$,接着又要做变换$\mathbf{R}_2$,可以计算出复合变换$\mathbf{R}=\mathbf{R}_1\cdot\mathbf{R}_2$,然后对原始的八叉树做一次变换。这样来避免误差的积累。

多次变换误差会积累，解决方法

第二项措施是为了尽量减少“舍入”误差,可以规定一个当前正要重建的八叉树,如果它的最底层叶结点对应的体素是部分地为显示对象所占据，那么当且仅当这个体素的中心位于某个黑变换后立方体内时,这个体素才被规定为黑,否则就规定为白。这样规定使得一般不会产生原来不存在的孔洞，避免在变换后形体中间出现断裂。




3
1
7
1
5
Y
5
5
7
0
4
4
6
X
7
2
Y
1
0
4
6
X

设已采取了上述两项措施,已知形体变换前的八叉树表示T_1,已计算出要做的复合变换R,要确定变换后形体的八叉树表示T_2,可以写出如下的算法框架:

1. 遍历形体原来的八叉树T_1,对遇到的每个黑结点,做下述步2。

2. 对遇到黑结点对应的正立方体做相应变换,得它新的一般来说是斜置的新位置。若这位置已超出定义八叉树的充分大正立方体C之外,报告出错;否则执行下述的步3。

3. 从要计算求出的目标树T_2的根开始,检查2中确定的处于新位置立方体与T_2中结点对应的直立的正立方体是否相交,分以下三种情况进行处理:

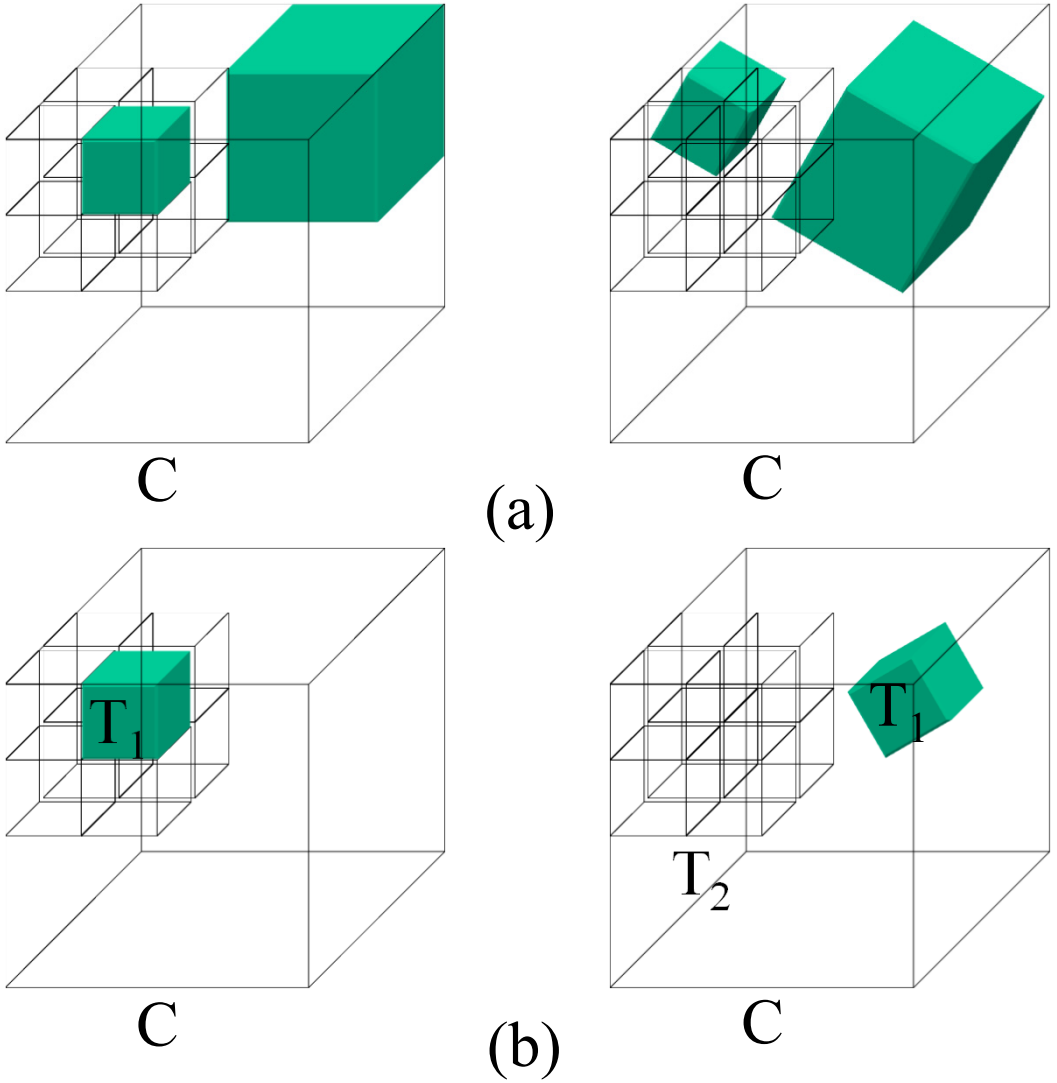
(1) 不相交,说明正考查直立正立方体未被占据,可保持为白结点,不做处理。

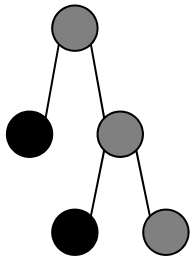
(2) 直立的正立方体整个被占据,即它在变换后"斜置"立方体内,置对应结点为黑结点。

(3) 在上述两条均不成立时,生成当前结点的八个子结点,对八个子结点对应的八个直立子立方体,依次再递归执行步3。如果最终这八个结点被标上同样特性,例如为黑结点,则应再删掉这八个子结点而把它们的共同父结点置为黑。

算法中,主要工作是检查某个直立的正立方体与一个斜置的正立方体是否相交。对所有黑结点对应正立方体处理相同,使得操作可以并行进行。

设已采取了上述两项措施,已知形体变换前的八叉树表示T_1,已计算出要做的复合变换R,要确定变换后形体的八叉树表示T_2。





算法框架：

for(遍历原形体八叉树上的每个黑结点,记为T_1){

变换黑结点T_1对应的立方体,得到新位置;

if(新位置超出定义八叉树的立方体C之外){报告出错;continue;}

T_2=C的根结点;

{ while(T_2被T_1部分占据&&T_2>预定义的最小体素)

{生成结点T_2的八个子结点;T_2= 第一个子结点;}

if(T_2和T_1不相交) T_2属性=白结点;

else if(T_2被T_1完全占据) T_2属性=黑结点;

//T_2被T_1部分占据&&T_2=预定义的最小体素

else if(T_1占据结点T_2对应的立方体的中心) T_2属性=黑结点;

else T_2属性=白结点;

while(T_2无后续兄弟结点&& T_2!=C的根结点){

T_2=T_2的父结点;

if(T_2的八个子结点被标上同样属性)

{T_2 =子结点属性;删除T_2的八个子结点;}

T_2

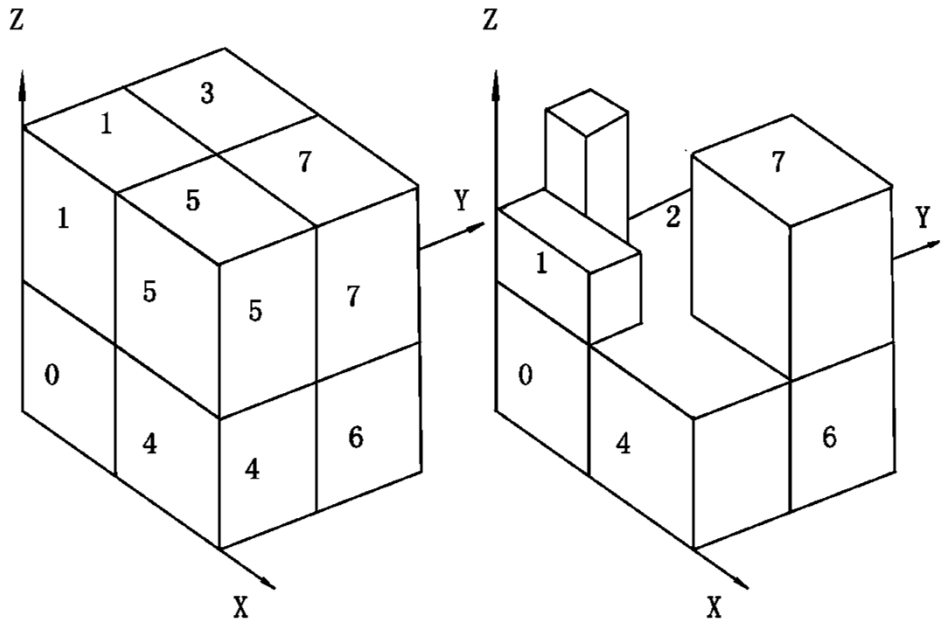
}

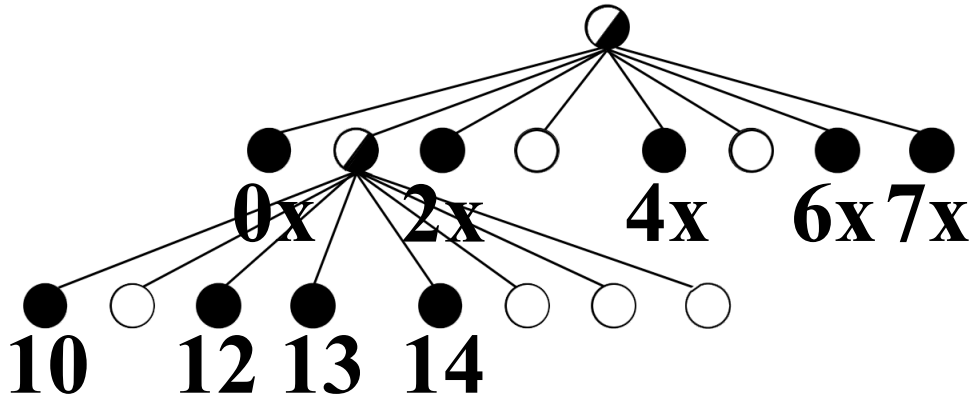
if(T_2有后续兄弟结点) T_2=下一个后续兄弟结点;

}while(T_2!=C的根结点)

线性八叉树

在对立方体做八等分时,按一致的方式, 对分出的子立方体进行编号。若划分共进行n层，则每个结点可以用n位的八进制数的数串来表示,数串从左至右,第一位对应第一次划分，第二位对应第二次划分，依此类推。据此整个八叉树就可以根据对其做深度优先遍历而依次列出的黑结点的编号序列来表示 。





上图所示三维形体,其线性八叉树表示是:

{0x,10,12,13,14,2x,4x,6x,7x}

求并运算 $C_1 \cup C_2$

两棵线性八叉树:

C_1={122,123,301,302,303,305,307}

C_2={12x,300,302,304,306}

将C_2的各结点依次插入到C_1的适当位置,使插入后编号渐增这一性质保持不变。当C_2(C_1)中结点可以包含C_1(C_2)中若干结点时,则取而代之。另外,如果插入后可以进行结点"压缩",也应该立即进行:

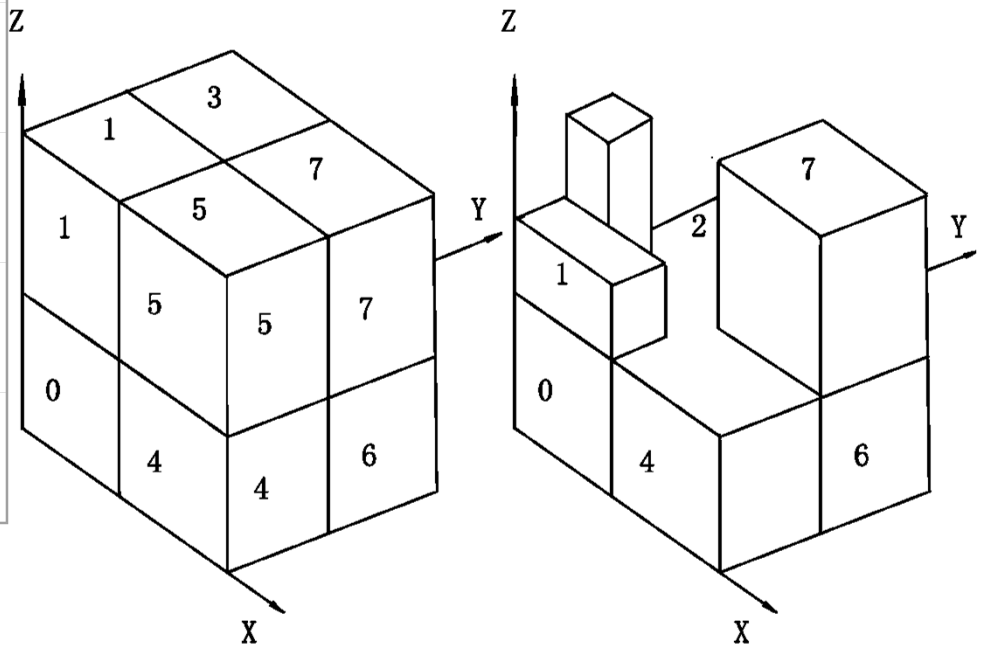
$C_1 \cup C_2$={12x,300,301,302,303,304,305,306,307}

={12x,30x}

z_1	y_1	x_1	优先级
<0	<0	<0	73561240
<0	<0	>0	37125604
<0	>0	<0	51743062
<0	>0	>0	15307426
>0	<0	<0	62470351
>0	<0	>0	26034715
>0	>0	<0	40652173
>0	>0	>0	04216537

八叉树表示形体的显示

当观察位置是$x_1>0,y_1<0,z_1>0$时,最可能被遮挡看不见的是编号2的子立方体,全部依次排出可以是26034715



$z_1>0, y_1<0, x_1>0$

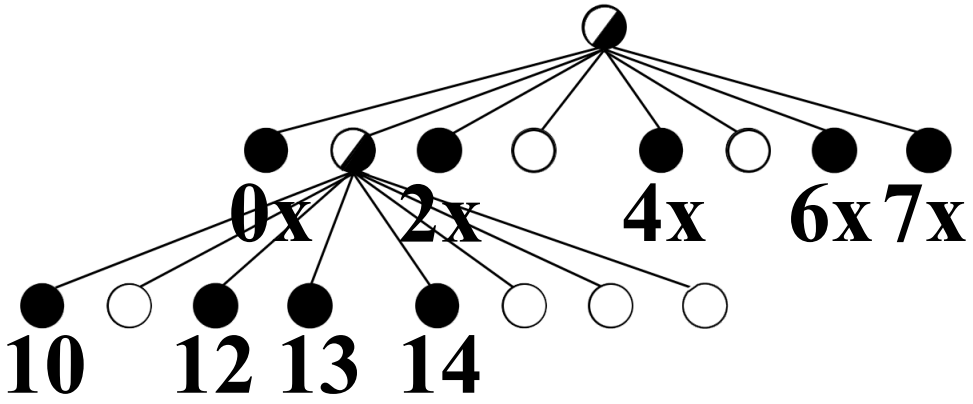
优先级26034715。

下图表示形体的线性八叉树

{0x,10,12,13,14,2x,4x,6x,7x}

按结点应显示次序排出的序列就是:

{2x,6x,0x,4x,7x,12,10,13,14}

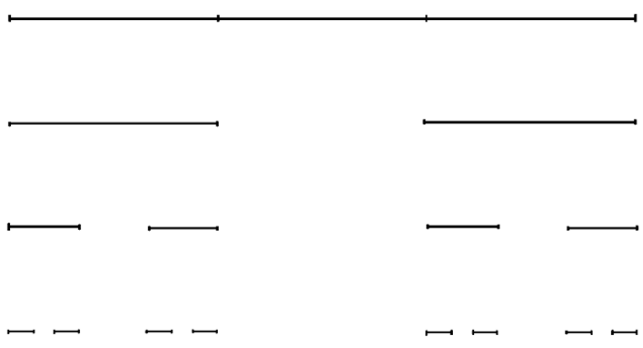


第三节 分形

一. 分形的概念

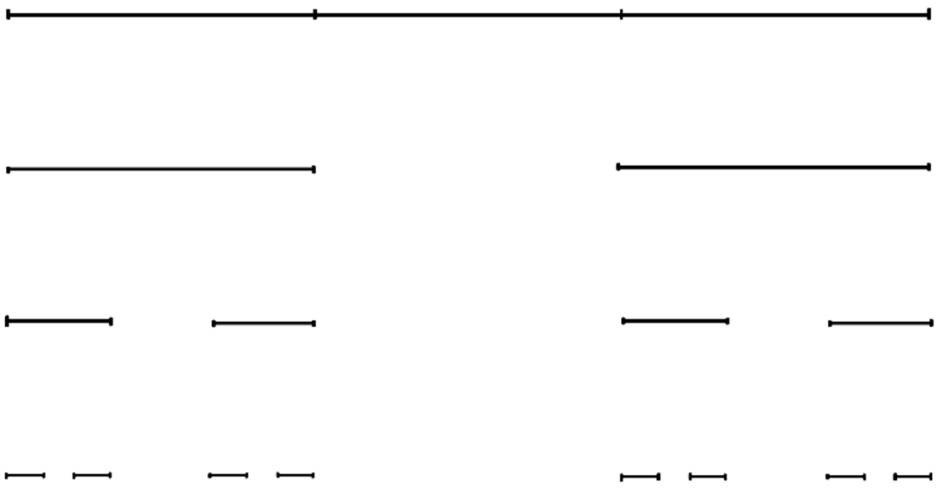
三分康托(Cantor)集

设E_0是闭区间[0,1],即E_0是满足$0 \le x \le 1$的实数x组成的点集;E_1是E_0去掉中间1/3之后的点集,即E_1是两个闭区间[0，1/3]和[1/3，2/3];E_2是分别去掉E_1中两个区间的中间1/3之后的点集,即E_2已经是四个闭区间。此过程要继续进行,E_k是2^k个长度为$1/3^k$的闭区间组成的点集。三分康托集F是属于所有的E_k的点组成的集,即 $F = \bigcup_{k=0}^{\infty} E_k$



第三节 分形

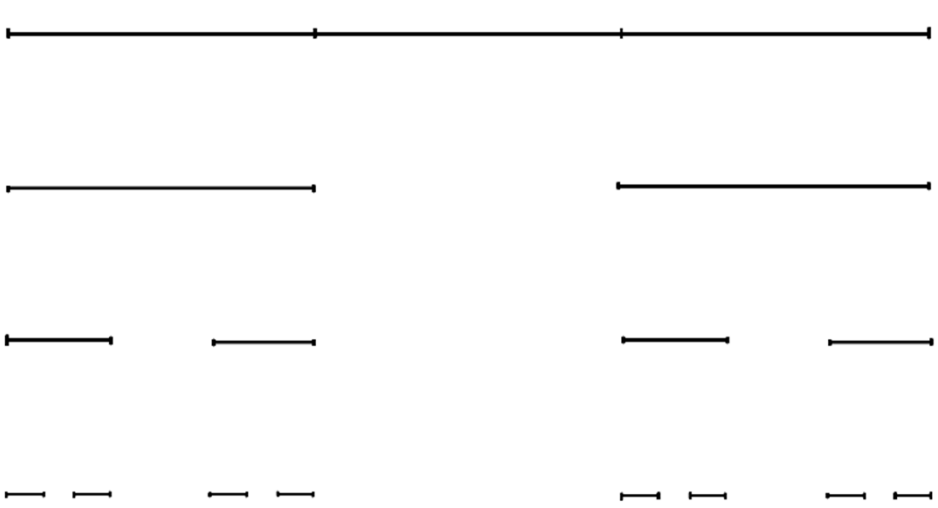
F可以看成是集序列E_k当k趋于无穷时的极限。只能画出k取定时的某个E_k。当k充分大时,E_k是对F的很好的近似的表现。



三分康托集是区间[0,1]中的可以展成以3为底的幕级数的下面形式的数组成的:

$$a_1 3^{-1}+a_2 3^{-2}+a_3 3^{-3}\ldots$$

其中a_i的取值限制为0或2,不取1。为看清这一事实,注意从E_0得到E_1时,去掉的是$a_i=1$的数,从E_1得E_2时,去掉的是$a_2=1$的数,并以此类推。



三分康托集具有的一些值得注意的特征,这些特征对许多其它的分形也是大体上适合的。

(1) F是自相似的。E_1的两个区间[0，1/3]，[2/3，1]的每一个，其内部F的部分与F整体相似，相似比为1/3。

(2) F具有“精细结构”，即它包含有任意小比例的细节。

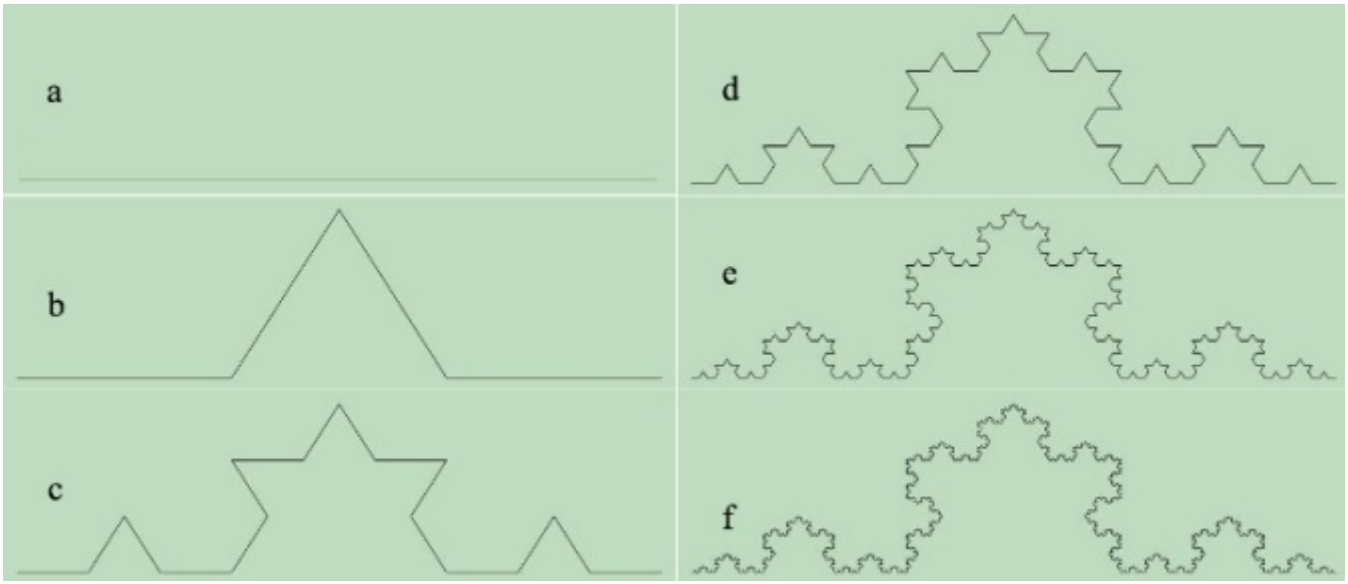
(3) F的实际定义是简单的和明确的。

(4) 传统的几何学很难描述F的性质,因为F不是满足某些简单条件的点的轨迹,也不是任何简单的方程的解的集合。

(5) F的局部几何性质也很难描述,在它的每点附近都有大量被各种不同间隔分开的其它点。

(6) 按传统几何学中的长度概念，F的长度为零。就是说,尽管从不可数集合这点上说F是一个相当大的集,但它却没有长度,或者说长度不能对F的形状或大小提供有意义的描述。

von Koch曲线



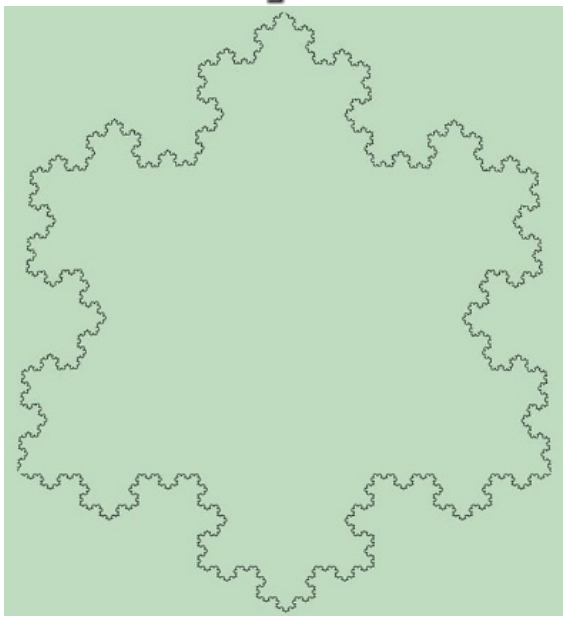
假设原正三角形的边长为1，因为每次迭代后，边长都增长为原来的4/3倍，因此n次迭代后，雪花的周长为

$$C = 3 \cdot \lim_{n\to\infty} \left(\frac{4}{3}\right)^n \to +\infty.$$

而雪花曲线所围面积为

$$S_n = \frac{\sqrt{3}}{4} \cdot \left[1 + (\frac{2}{9}) \cdot 3 + (\frac{2}{9})^2 \cdot 3 + \cdots + (\frac{2}{9})^n \cdot 3\right]$$

$$\to \frac{13\sqrt{3}}{28} < \frac{\pi}{3}, (n \to +\infty)$$



称集合F是分形,即认为它具有下面典型的性质:

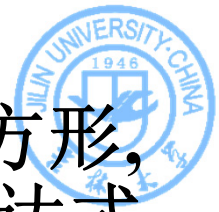
(1) F具有精细的结构，即有任意小比例的细节。

(2) F是如此的不规则以至它的整体和局部都不能用传统的几何语言来描述。

(3) F具有某种自相似的形式,可能是近似的或是统计的。

(4) 一般地,F的"分形维数"大于它的拓扑维数。

(5) 在大多数令人感兴趣的情形下,F以非常简单的方法定义,可能由迭代产生。



Hausdorff维数

考虑一个简单的几何图形,取一个边长为1的正方形,若每边扩大2倍,则正方形面积放大4倍,其数学表达式为$2^2=4$,这是2维图形。对3维图形,如考虑边长为1的立方体,令每边长放大2倍,则立方体体积扩大8倍,其数学表达式为$2^3=8$。

类似地,对一个D_f维的几何对象,若每边长扩大L倍,则这个几何对象相应地放大K倍,归纳前述结果,D_f,L,K三者间的关系式应为:

$$L^{Df}=K$$

解出D_f,有:

$$D_f=\ln K/\ln L$$

这里D_f不必是整数。这就是Hausdorff引人的维数概念,可以称为Hausdorff维数(相似性维数)。

假定有一个单位正方形,把它每边三等分得九个小的正方形,九个小正方形面积总和是原单位正方形面积,即$9\times(1/3)^2=1$。现在我们把D_f维的几何对象等分为N个小的几何图形,则每个小图形每维缩小为原来的r倍,而N个小图形的总和应有$N\cdot r^{Df}=1$。

这时解出D_F,有:

$$D_f = \frac{\ln N}{\ln(1/r)} = \frac{\ln(1/N)}{\ln(r)} =$$

容易看出式(1)和(2)本质上是相同的,即这样引入的也是Hausdorff维数

von Koch曲线,每次分为4个小图形,每个小图形缩小1/3倍,故其Hausdorff维数为D_F:

$$D_F = \frac{\ln 4}{\ln 3} = 1.2619\cdots$$

二、分形一般算法

规则分形的生成算法。对算法的输入是事先给定的一个整数k、源形E0及生成规则,算法操作步骤如下:

```
void  Fractal(Rule R,int k,Source E0,int m){
    // R分形规则；k分形迭代层数；m分形组成数；E0源形
    // i记层数；j记生成部分图形的数目；队列Q保存图形；A0记源形
    i=0; j=1; Q=φ; A0=E0;
    do {
            do {
                    //由A0和R计算它的m个分解部分;
                    calculate(A0,R, A1,A2, …, Am);
                    draw(Al,A2, …,Am); //图形绘制
                    //生成各部分图形依次加到队尾
                    insert(A1,A2,…,Am,Q);
                    delete(A0,Q);  //从队头取出一个部分图形
                    j=j+1;
            }while (j<=m^i);
            j=1;i++; //进入下一层
    }while(i<=k); //结束判断
}
```

例如：von Koch曲线

其源形E_0可以是一条线段, 记其端点坐标为P_0,P_1。

在算法步1,应令$A_0=E_0=(P_o,P_1)$,

在算法步2,需要依据P_o,P_1,计算图中P_2,P_3,P_4三点的坐标。这样m=4,分别得到四个部分图形是

$A_1=(P_0,P_2)$,$A_2=(P_2,P_3)$,$A_3=(P_3,P_4)$,$A_4=(P_4,P_1)$。

在算法步3,可画出四条线段P_0P_2,$P_2\ P_3$, $P_3\ P_4$, P_4P_1,擦去前次画线时可能画出的P_2P_4部分。

三、Von Koch算法

利用自相似变换来绘制分形

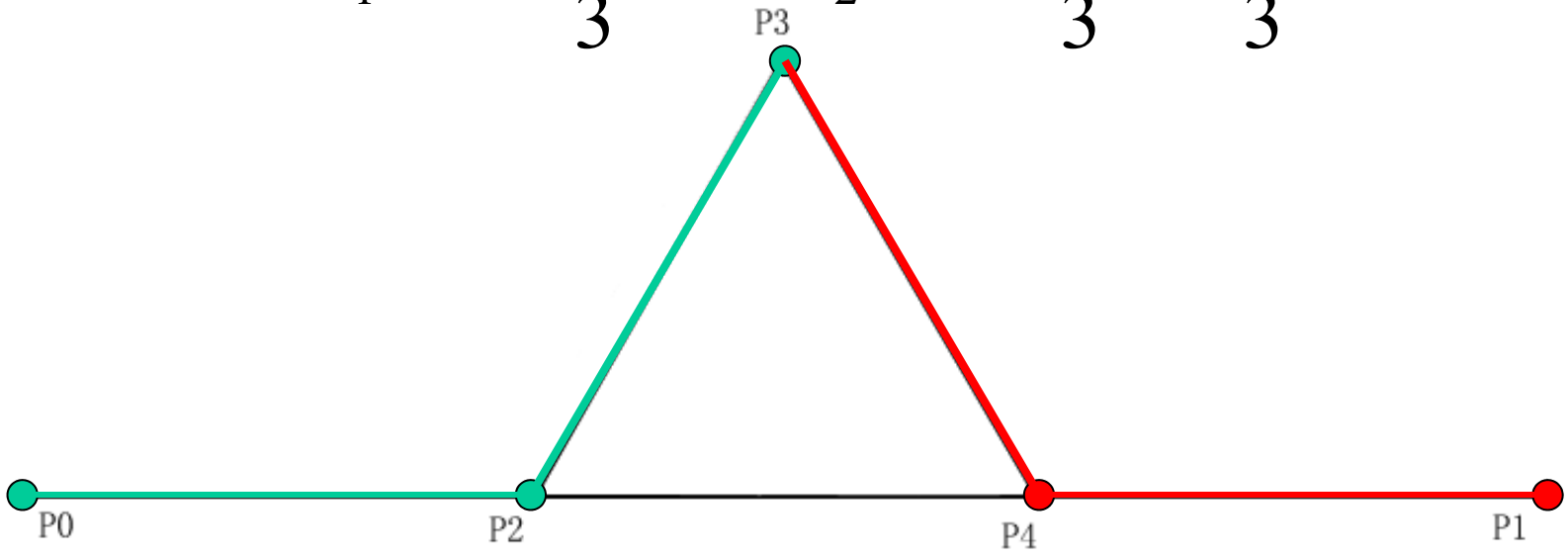
设D是欧氏空间R^n的闭子集,映射S:D→ D称为是D上的压缩，如果对所有D上的点x,y,存在一个数c,0<c<1,能使|S(x)-S(y)|≤ c|x-y|。如果其中等号成立,即若|S(x)-S(y)|=c|x-y|,则S把一个集变成了它的几何相似集,此时映射S称为是相似的。

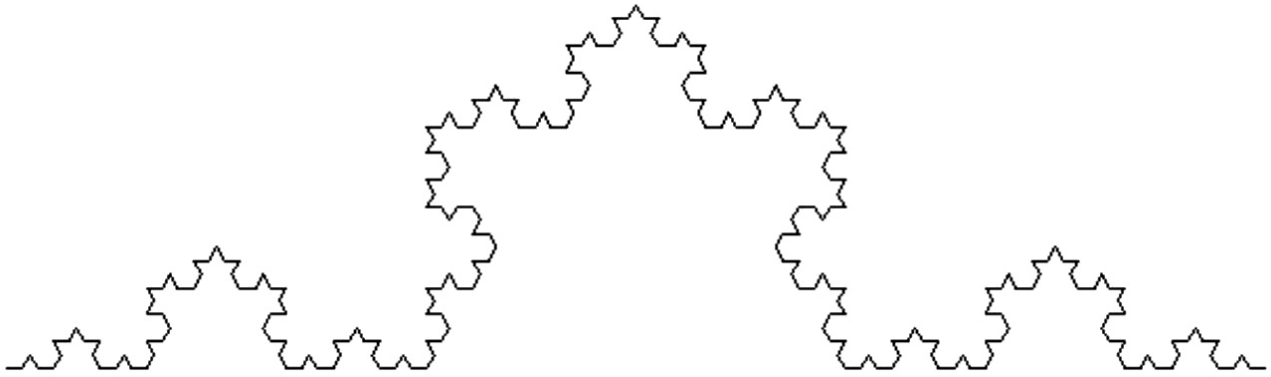
设 $S_1,\ldots,S_n$ 是压缩，称D的子集F对变换 $S_1,\ldots,S_n$ 是不变的,如果

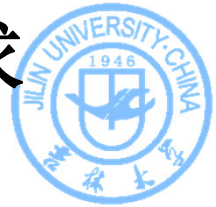
$$F = \bigcup_{i=1}^{n} S_i(F)$$

三分康托集的情形,这时令S_1,S_2是R→R的变换,分别由

$$S_1(x)=\frac{1}{3} \qquad S_2(x)=\frac{1}{3}x+\frac{2}{3}$$







P_0和P_1的坐标是(0,0)和(1,0),则可以计算求出P_2,P_3,P_4的坐标是

$$\left(\frac{1}{3},0\right),\left(\frac{1}{2},\frac{\sqrt{3}}{6}\right),\left(\frac{2}{3},0\right)$$

自相似变换S1和S2是平面变换,可一般地设变换矩阵为:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix}$$

第一个变换S_1把点P_0,P_1,P_3,依次变到P_o,P_3,P_2,这就得到:

$$\begin{bmatrix} & & \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \end{bmatrix} \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \\ \frac{1}{3} & 0 & 1 \end{bmatrix}$$

于是有

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \\ \frac{1}{3} & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} & 0 \\ \frac{\sqrt{3}}{6} & -\frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

第二个变换S_2把点P_0,P_1,P_3,依次变到P_3,P_1,P_4，这就得到：

$$\begin{bmatrix} & & \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \end{bmatrix} \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \\ 1 & 0 & 1 \\ \frac{2}{3} & 0 & 1 \end{bmatrix}$$

于是有

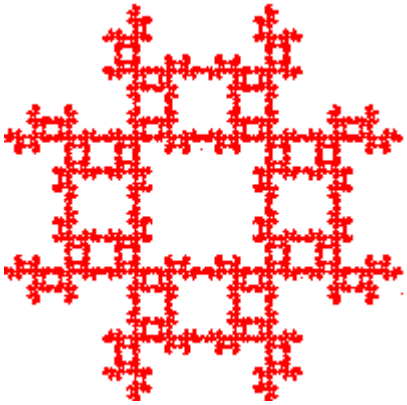
$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -\sqrt{3} & -\sqrt{3} & 2\sqrt{3} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \\ 1 & 0 & 1 \\ \frac{2}{3} & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} & 0 \\ \frac{\sqrt{3}}{6} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{6} & 1 \end{bmatrix}$$

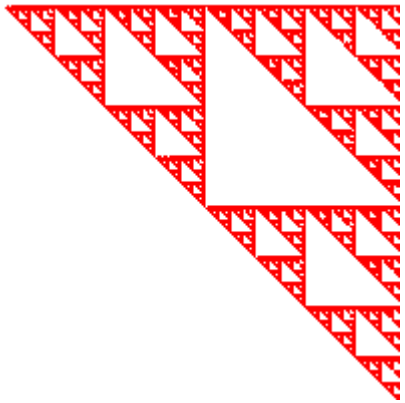
因此

$$\begin{cases} S_1\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & -\frac{1}{2} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \\ S_2\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{6} \\ -\frac{\sqrt{3}}{6} & -\frac{1}{2} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{6} \end{bmatrix} \end{cases}$$

绘制Von Koch曲线

```
void  von_Koch_display (void){
        x1=0;y1=0;// (x1,y1)为初始点
        s.push(POINT (x1,y1));
        for(int i=0;i<k;i++) {
                (x1,y1) =s.top(); //取栈顶
                x2=1/2.0*x1+sqrt(3)/6*y1;
                y2= sqrt(3)/6*x1-1/2.0*y1;
                x3=1/2.0*x1+ sqrt(3)/6*y1+1/2.0;//变换
                y3=- sqrt(3)/6*x1-1/2.0*y1+sqrt(3)/6;
                Setpixel(x2,y2, RGB(0,0,0)); //画点
                Setpixel(x3,y3, RGB(0,0,0));
                s.push(POINT (x2,y2));
                s.push(POINT (x3,y3)); //压栈
        } //结束判断
}
```





$$\begin{cases} S_1 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.7 & 0.4 \\ 0.4 & 0.7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ S_2 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.667 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.7 \end{bmatrix} \end{cases}$$

上面的变换能产生很紧松树树枝的图象。

四. Julia和Mandelbrot集

设有复数域上如下形式的二次函数:$f(z)=z^2+c$

其中c是复数值常数,做迭代操作:

$$z_{n+1}= z_n^2+c, n=0,1,2,\ldots$$

研究的问题是:

1.给定z_0,当参数c在什么范围内取值能保证$|z_n|$有界

2.当c给定,如何选取z_0,使$|z_n|$有界?

上述迭代,当c=0时,可以有以下三种情况:

1. 序列中的数按模来说越来越小,且趋于零。这时说零是$z \rightarrow z^2$的吸引子。所有与坐标原点相距小于1的点都产生趋向零的序列。

2. 序列中的数按模来说越来越大,且趋向无穷,这时"无穷"也称为过程的吸引子。与坐标原点的距离超过1的所有点都产生趋向无穷的序列;

3. 距坐标原点为1的点,序列总是产生在上面两个吸引区域之间的边界上,此时边界恰为复平面上的单位圆周。

对于上述迭代,当$c \neq 0$时,吸引子不再是零,吸引区域的边界不再是光滑的,而是具有自似形的分形结构,这种边界称为Julia集。

在复平面上，使$z \to z^2+c$的迭代过程成为有界的复参数c的集合叫做**Mandelbrot**。

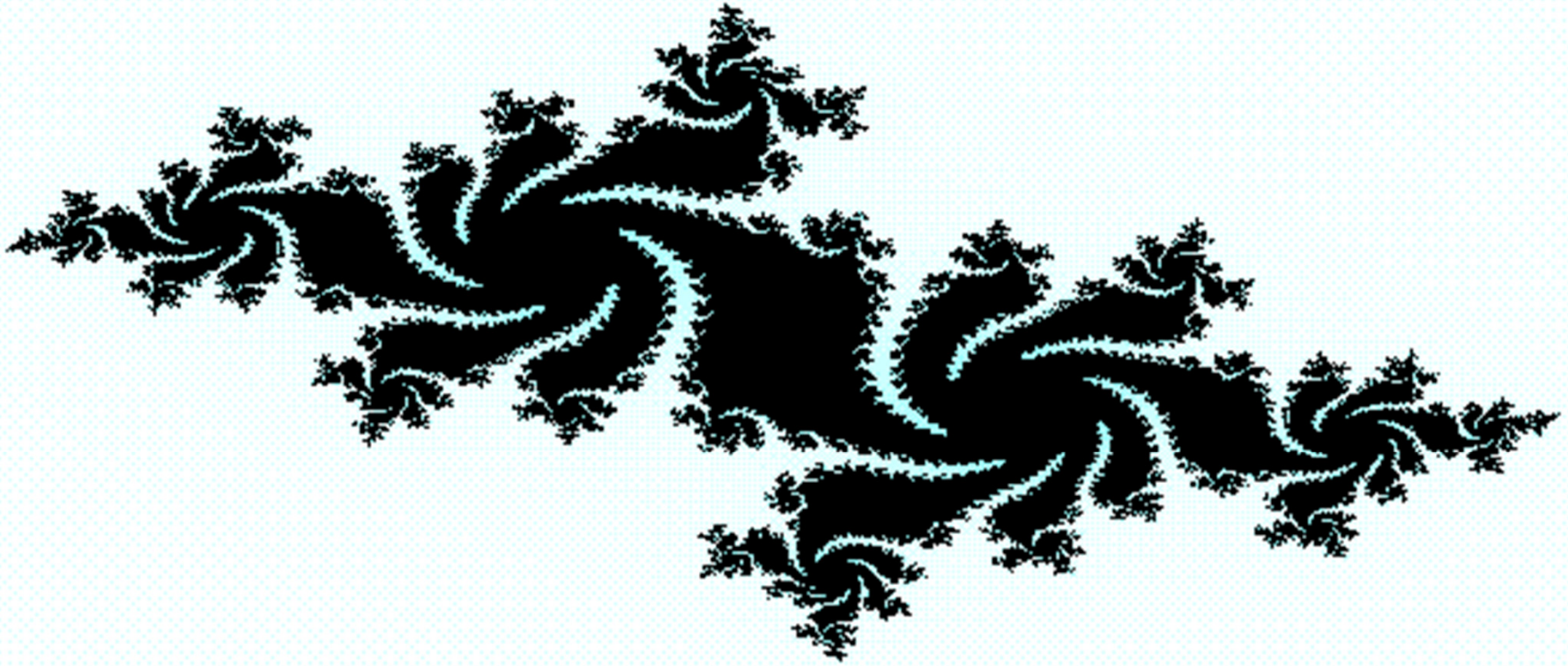
设复平面上的迭代过程是:

$$z_{k+1}=z_k^2+c$$

分离实部和虚部,记$z_k=x_k+y_k i, c=p+qi$,有

$$\begin{cases} x_{k+1} = x_k^2 - y_k^2 + p \\ y_{k+1} = 2x_k y_k + q \end{cases}$$

设计算机显示屏幕的图形分辨率是a×b点,可显示颜色是k+1种,以数字0到k表示,0表示黑色。取定p和q的值,考虑平面上每一点(x,y),探讨吸引区域的结构及其边界即Julia集。

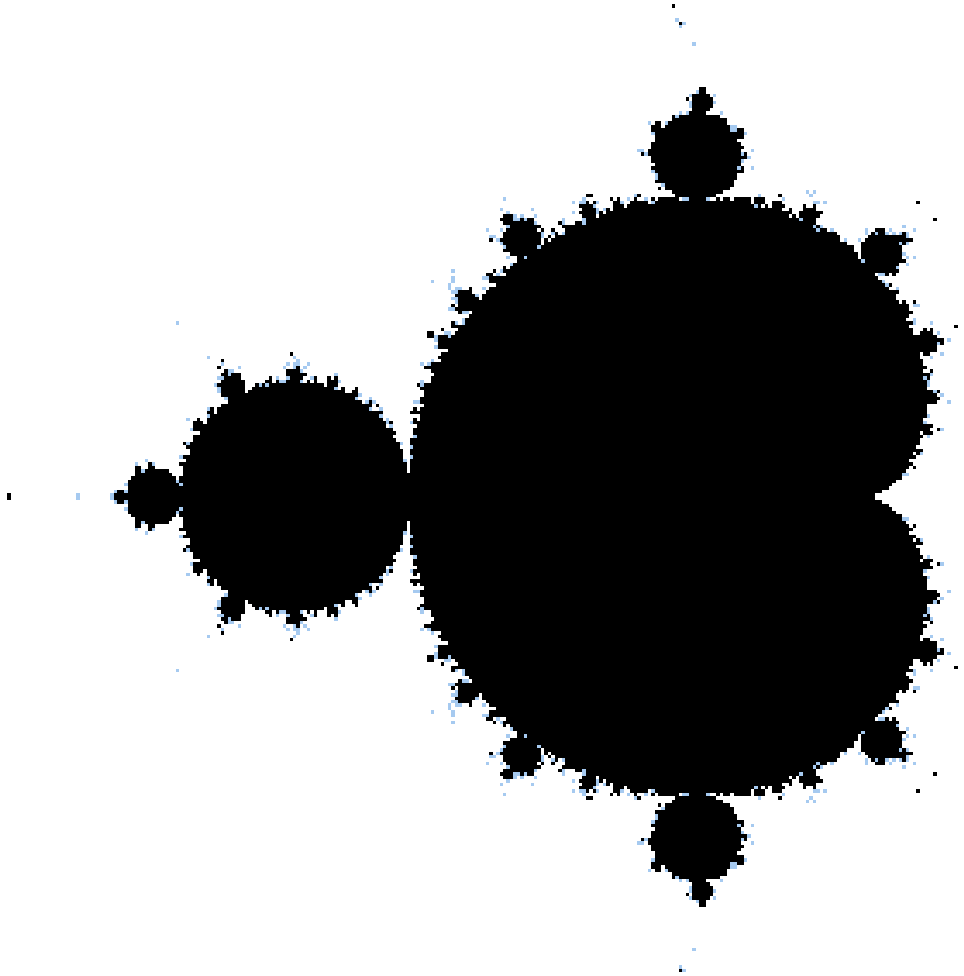


Julia集 a=600,b=400,M=100,K=100,p=-0.605,q=-0.43, -1.5≤x≤1.5, -1.5≤y≤1.5,


```
void  Julia(int a,int b,int K,int M,double p,double q){
    double xmin=-1.5,ymin=-1.5,xmax=1.5,ymax=1.5;
    int nx,ny,k;
    double tx,ty,x0,y0,xk,yk,r;
    tx=(xmax-xmin)/(a-1);ty=(ymax-ymin)/(b-1);//计算步长
    for(nx=0;nx<=a-1;nx++)
        for(ny=0;ny<=b-1;ny++){
            x0=xmin+nx*tx;
            y0=ymin+ny*ty;
            k=0;
            xk=x0*x0-y0*y0+p;//计算复数z
            yk=2*x0*y0+q;
            k++;
            r=x0*x0+y0*y0;//计算长度
            x0=xk;y0=yk;
            while (r<=M&&k<K){
                xk=x0*x0-y0*y0+p;
                yk=2*x0*y0+q;k++;
                r=x0*x0+y0*y0;
                x0=xk;y0=yk;
            }
            if(r>M) SetPixel(nx,ny,RGB(k,k,k));
            if(k==K) SetPixel(nx,ny,RGB(0,0,0));
        }
}
```

固定点(x,y),在不同的p和q值之下追踪其迭代点列,在(p,q)平面上记录结果,从而产生Mandelbrot集。设要考查的(p,q)平面范围是-2.25<p<0.75,-1.5<q<1.5,对计算机显示屏幕的假定如前。实施步骤为:


```
void  Mandelbrot(int a,int b,int K1,int M1) {
        double pmin=-2.25,qmin=-1.5,pmax=0.75,qmax=1.5;
        int M=M1,np,nq,k,K;K=K1;
        double tp,tq,p0,q0,p,q,xk,yk,r,x0,y0;
        tp=(pmax-pmin)/(a-1);tq=(qmax-qmin)/(b-1);
        for(np=0;np<=a-1;np++)
                for(nq=0;nq<=b-1;nq++){
                        p0=pmin+np*tp;
                        q0=qmin+nq*tq;
                        k=0;x0=y0=0.0;
                        xk=x0*x0-y0*y0+p0;
                        yk=2*x0*y0+q0;k++;
                        r=x0*x0+y0*y0;
                        x0=xk;y0=yk;
                        while (r<=M&&k<K) {
                                xk=x0*x0-y0*y0+p0;
                                yk=2*x0*y0+q0;k++;
                                r=x0*x0+y0*y0;
                                x0=xk;y0=yk;
                        }
                        if(r>M) SetPixel(np,nq,RGB(k,0,0));
                        if(k==K) SetPixel(np,nq,RGB(255,255,255));
                }
}
```



Mandelbrot集

a=400,b=400,M=100,K=100,x=0,y=0

pmin=-2.25,qmin=-1.5,pmax=0.75,qmax=1.5

第四节 粒子系统

- 1983年Reeves提出粒子系统(Particle System)这一重要理论,依据该理论可利用简单有效的算法来实现具有不规则形状且变化复杂的自然场景。
- 粒子系统的粒子一般都有这样一组属性,如位置、速度、颜色、大小、生命周期等。每一个粒子在系统中都要经历完整的生命周期:从产生、运动直至死亡。粒子通常在指定的区域由随机过程产生,并不断更新属性,最后死亡。由于粒子的不断运动,使得模拟的场景具有一定的动态性,因此可以采用粒子系统模拟战场爆炸和烟雾的效果。

- 采用粒子系统模拟自然场景时的步骤可描述如下:
- 1)粒子系统初始化。依据模拟绘制的场景确定所需粒子的数量、位置,并对每个粒子进行初始化属性赋值,将粒子加入粒子系统中。
- 2)粒子属性更新。依据设定的运动规律及时更新粒子的运动位置、速度、生命周期、颜色、透明度等属性。
- 3)将“死亡”粒子从系统中删除。随着粒子属性的不断更新,一些粒子已经达到了自己的生命周期,或者颜色与背景重合,为了提高整个系统的性能,需要将“死亡”粒子从系统中除去。
- 4)绘制图像。对于系统中尚存在的那些粒子,选择一定的绘制算法将其绘制成图像并通过屏幕显示出来。

- 具体步骤如下：
- 1.产生新的粒子。
- 2.赋予每一新粒子一定的属性。
- 3.删去那些已经超过生存周期的粒子。
- 4.根据粒子的动态属性对粒子进行移动和变换。
- 5.显示由有生命的粒子组成的图像。