第八章 消隐

因为计算机图形处理的过程中，不会自动消去隐藏部分，相反会将所有的线和面都显示出来，所以如果想真实显示三维物体，必须在视点确定之后，将对象表面上不可见的点、线、面消去。执行这种功能的算法，称为消隐算法。根据消隐对象的不同可分为：线消隐 (Hidden-line)和面消隐。

8．1 线消隐

线消隐处理对象为线框模型，是以场景中的物体为处理单元，将一个物体与其余的 k－1 个物体逐一比较，仅显示它可见的表面以达到消隐的目的。此类算法通常用于消除隐藏线。**1.** 凸多面体的隐藏线消隐

凸多面体是由若干个平面围成的物体。假设这些平面方程为

$$a_i x+b_i y+c_i z+d_i=0, \quad i=1, 2, \ldots, n \tag{8.1}$$

物体内一点 $P_0(x_0, y_0, z_0)$满足 $a_i x_0+b_i y_0+c_i z_0+d_i<0$，平面法向量$(a_i, b_i, c_i)$指向物体外部的。此凸多面体在以视点为顶点的视图四棱锥内，视点与第 i 个面上一点连线的方向为(l_i, m_i, n_i)。那么第 i 个面为自隐藏面的判断方法是：

$$(a_i, b_i, c_i)\times(l_i, m_i, n_i)>0$$

对于任意凸多面体，可先求出所有隐藏面，然后检查每条边，若相交于某条边的两个面均为自隐藏面，根据任意两个自隐藏面的交线，为自隐藏线，可知该边为自隐藏边（自隐藏线应该用虚线输出）。

2. 凹多面体的隐藏线消隐

凹多面体的隐藏线消除比较复杂。• 假设凹多面体用它的表面多边形的集合表示 ，消除隐藏线的问题可归结为：一条空间线段 P_1P_2 和一个多边形 a，判断线段是否被多边形遮挡。如果被遮挡，求出隐藏部分。以视点为投影中心，把线段与多边形顶点投影到屏幕上，将各对应投影点连线的方程联立求解，即可求得线段与多边形投影的交点。

如果线段与多边形的任何边都不相交，则有两种可能，线段投影与多边形投影分离或线段投影在多边形投影之中，前一种情况，线段完全可见。后一种情况，线段完全隐藏或完全可见。然后通过线段中点向视点引，若此射线与多边形相交，相应线段被多边形隐藏；否则，线段完全可见。

若线段与多边形有交点，那么多边形的边把线段投影的参数区间[0，1]分割成若干子区间，每个子区间对应一条子线段(如图 8-1 所示)，每条子线段上的所有点具有相同的隐藏性，如图所示。为进一步判断各子线段的隐藏性，首先要判断该子线段是否落在该多边形投影内。对于子线段与多边形的隐藏关系的判定方法与上述整条线段与多边形无交点时的判定方法相同。图

8-1 线段投影被分为若干子线段 把上述线段与所有需要比较的多边形依次进行隐藏性判断，记下各条边隐藏子线段的位置，最后对所有这些区域进行求并集运算，即可确定总的隐藏子线段的位置，余下的则是可见子线段。

8.2 面消隐

面消隐(Hidden-surface)处理对象为填色图模型，是以窗口内的每个像素为处理单元，确定在每一个像素处，场景中的物体哪一个距离观察点最近（可见的），从而用它的颜色来显示该像素。此类算法通常用于消除隐藏面。

8.2.1 区域排序算法基本思想：在图像空间中，将待显示的所有多边形按深度值从小到大排序，用前面可见多边形去切割后面的多边形，最终使得每个多边形要么是完全可见，要么是完全不可见。用区域排序算法消隐，需要用到一个多边形裁剪算法。当对两个形体相应表面的多边形进行裁剪时，我们称用来裁剪的多边形为裁剪多边形，另一个多边形为被裁剪多边形。算法要求多边形的边都是有向的，不妨设多边形的外环总是顺时针方向的，并且沿着边的走向，左侧始终是多边形的外部，右侧是多边形的内部。若两多边形相交，新的多边形可以用“遇到交点后向右拐”的规则来生成。于是被裁剪多边形被分为两个乃至多个多边形；我们把其中落在裁剪多边形外的多边形叫

作外部多边形；把落在裁剪多边形之内的多边形叫作内部多边形。 算法的步骤：

（1） 进行初步深度排序，如可按各多边形 z 向坐标最小值(或最大值、平均值)排序。

（2）选择当前深度最小(离视点最近)的多边形为裁剪多边形。

（3）用裁剪多边形对那些深度值更大的多边形进行裁剪。

（4）比较裁剪多边形与各个内部多边形的深度，检查裁剪多边形是否是离视点最近的多边形。如果裁剪多边形深度大于某个内部多边形的深度，则恢复被裁剪的各个多边形的原形，选择新的裁剪多边形，回到步骤(3)再

做，否则做步骤⑸。

（5）选择下一个深度最小的多边形作为裁剪多边形，从步骤⑶开始做，直到所有多边形都处理过为止。在得到的多边形中，所有内部多边形是不可见的，其余多边形均为可见多边形。**8.2.2　深度缓存(Z-buffer)算法**

深度缓存(Z-buffer)是一种在图像空间下的消隐算法，包括：帧缓冲器 –保存各像素颜色值(CB)；z 缓冲器 --保存各像素处物体深度值(ZB)。其中 z 缓冲器中的单元与帧缓冲器中的单元一一对应。

深度缓存(Z-buffer)思路：先将 z 缓冲器中个单元的初始值置为-1(规范视见体的最小 n 值)。当要改变某个像素的颜色值时，首先检查当前多边形的

深度值是否大于该像素原来的深度值（保存在该像素所对应的 Z 缓冲器的单元中），如果大于，说明当前多边形更靠近观察点，用它的颜色替换像素原来的颜色；否则说明在当前像素处，当前多边形被前面所绘制的多边形遮挡了，是不可见的，像素的颜色值不改变。Z-buffer 算法的步骤如下：

⑴ 初始化 ZB 和 CB，使得 ZB(i, j)=Z_{max}，CB(i, j)=背景色。其中，i=1, 2, …, m，j=1, 2, …, n。

⑵ 对多边形 a，计算它在点(i, j)处的深度值 $z_{i,j}$。

⑶ 若 z_{ij}<ZB(i, j)，则 ZB(i, j)=z_{ij}，CB(i, j)=多边形 a 的颜色。

⑷ 对每个多边形重复⑵、⑶两步。最后，在 CB 中存放的就是消隐后的图

形。

8.2.3　　扫描线算法在多边形填充算法中，活性边表的使用取得了节省运行空间的效果。用同样的思想改造 Z-buffer 算法：将整个绘图区域分割成若干个小区域，然后一个区域一个区域地显示，这样 Z 缓冲器的单元数只要等于一个区域内像素的个数就可以了。如果将小区域取成屏幕上的扫描线，就得到扫描线 Z 缓冲器算法。扫描线算法描述 **for (各条扫描线)**

{将帧缓冲器 **I(x)**置为背景色；

将 **Z** 缓冲器的 **Z(x)**置为最大值；

for (每个多边形**)**

```
{ 求出多边形在投影平面上的投影与当前扫描线的相交区间
for （该区间内的每个像素）
    if ( 多边形在此处的 Z 值小于 Z(x) )
        { 置帧缓冲器 I(x)值为当前多边形颜色;
          置 Z 缓冲器 Z(x)值为多边形在此处的 Z 值; }
    }
}
```

用计算机编制绘图程序时，除了利用上述消隐方法外，常常考虑图形的

几何特点（如下面要讲的图形的几何构造等）使图形生成更直观、方便或快捷。

8.3 图形几何构造

从图形几何构造的角度讲，一条边由两个点组成，一个面由几条或若干条边围成，场景中的任意物体都是由点、边、面按一定的拓扑结构组成的。称点、边、面为基本元素，简称基元，其中，点是最小的基元。对图形设计而言，通过几何点并按一定的拓扑结构可以完成场景中任意物体的几何描述，同理，用户可操作的物体对象还有线（边）、三角形小面、四边形小面、n 边形、圆、弧、异型面、立方体、球、圆柱和圆环面等线元、面元和体元，点元、线元、面元和体元同称为图形系统的基元。

基元是建立图形系统和应用图形系统的基石。当我们需要通过程序来显示一个特定的场景，点、边和三角形一般为基本的图元,任何一个多边形，无论为单连通还是多连通区域都

可以通过系化规则转化成多个三角形来处理。对于二维和三维及图形软件系统，最小的几何图元是几何定点，其次为直线（边）、面和其它面元与体元。对于一个三维实体，无论为凸的、凹的有洞的还是无洞的，计算机所显示的是它的表面。而表面是通过小面来逼近的，这一过程称为表面离散化。在物体表面急过渡区域常采用较多的小面来逼近，在较为平坦的区域，可以用较少的小面来逼近。

定义基元是基于图形描述和图形应用的方便性的而言。基元往往在世界坐标中进行描述并在光栅系统（屏幕坐标或窗口坐标系中）完成显示任务的。可以给基元赋属性，比如，几何定点的属性由几何坐标、顶点颜色、定点法线等，直线的属性有直线的几何拓扑、线宽、线型等，面的属性有面的几何拓扑、填充模式等。在构造几何数据时，往往需要采用链表、队列、树等结构。下面介绍常用的几种基元的几何构造方法：

（1）直线段

一条直线段由两个端点定义，为方便可采用 Windows 编程中经常使用的 MoveTo()和 LineTo()函数绘制直线。首先提供一对顶点，来定义第一条直线

段，以后每增加一条直线段，如第 i 条线段由第 i-1 和第 i 个顶点来定义，其颜色由第 i-1 个顶点的颜色决定。这样构成了一条折线。绘制一条闭合的折线，可将最后的顶点和第一个顶点进行连线，其颜色由最后一个顶点的颜色所决定。

（2）小面

一个物体，无论其表面形状多么复杂，计算机通常将它视为由若干小面组成的。将一个曲面分割成若干小面，首先要对曲面进行网格的划分，称为有限单元化。其次要获得网格的各种顶点坐标，对顶点进行几何描述形成单个边，并由边构成面，通过面循环构成整个物体的拓扑结构。接着要赋予顶

点信息（如颜色、纹理坐标、法向量等等），这三个重要的步骤通常称为前处理，最后作为输入信息传递给图形处理系统。用什么样的小面逼近曲面是十分重要的，比如，将一球用标示地理位置的经纬线来划分，可得到许多四边形小面，而两端要用三角形来逼近，事实上四边形可划分为两个三角形，这样解决了三角形的着色问题也就解决了所有面的着色问题。小面通常为三角形和四边形的小面。

对于四边形也可在一个四边形上延伸出一个四边形片，如图 8-2 所示。四边形的顶点处理的顺序定义如表 8-1。

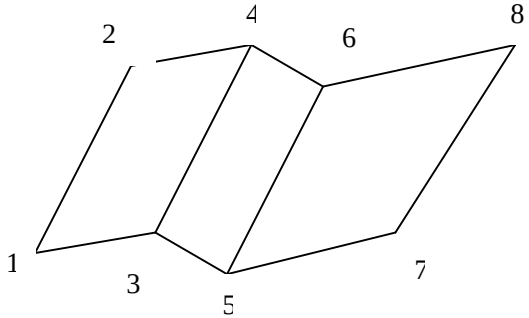

图 8-2 四边形片示意图

表 8-1 四边形片中顶点处理顺序

四边形序号	顶点处理顺序
1	顶点 1，顶点 2，顶点 3，顶点 4
2	顶点 3，顶点 3，顶点 4，顶点 5
3	顶点 5，顶点 6，顶点 7，顶点 8

由于曲面都是由三角形小面和四边形小面来逼近，而三角形小面是自定义的基本面元，因而需要对四边形小面进行细化。一个四边形可细化为两个三角形，如图 8-3 所示。

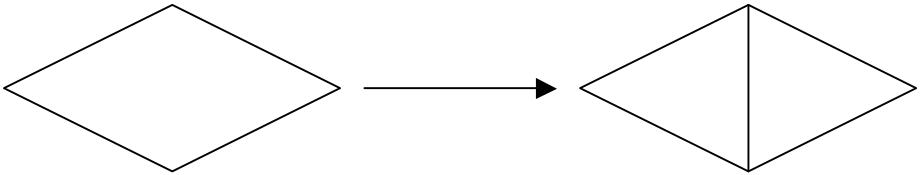

图 8-3　一个四边形细化为两个三角

对于任意一个平面区域或曲面，甚至有离散数据点组成的区域，都应该进行三角剖分，三角剖分法事实上属于几何拓扑分析方法。

三角形由三个顶点来定义，称为三角形基元。多个三角形一般由三角形片或三角形扇来定义。

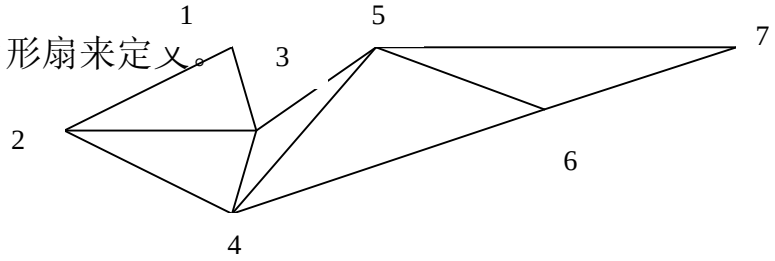

图 8-4 三角形片示意图

如图 8-4 定义了一个三角形片，从图中可知，每增加一个顶点便可增加一个三角形。三角形的顶点处理的顺序可以定义为表 8-2。

表 8-2 三角形的顶点处理的顺序

三角形序号	顶点处理顺序
1	顶点 1，顶点 2，顶点 3，
2	顶点 3，顶点 2，顶点 4，
3	顶点 3，顶点 4，顶点 5，
4	顶点 5，顶点 4，顶点 6，
5	顶点 5，顶点 6，顶点 7，

图 8-5 表示了一个三角形扇，尽管每增加一个顶点仍增加一个三角形，但每个三角形共用一个顶点，显然这种方法表示的终极部分会很方便，表 8-2 表示三角扇形中三角形的顶点处理顺序。

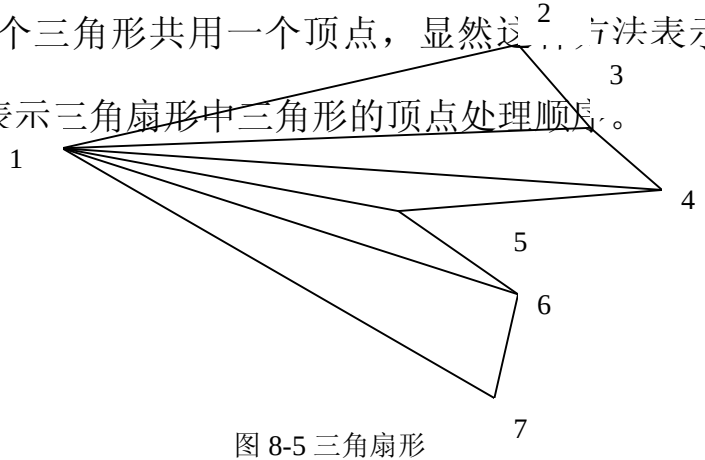

图 8-5 三角扇形

表 8-2 三角扇形中三角形的顶点处理顺序

三角形序号	顶点处理顺序
1	顶点 1，顶点 2，顶点 3，
2	顶点 1，顶点 3，顶点 4，
3	顶点 1，顶点 4，顶点 5，
4	顶点 1，顶点 5，顶点 6，

5	顶点 1，顶点 6，顶点 7，

（3）边的可见性

一个面元（比如三角形）往往为一个凸区域，要绘制图 8-6 左边所示的凹多边形会很费事，可通过定义边的可见性来达到这一目的。将凹区域分成两个凸四边形，使其中增加的连线不可见，从而达到绘制多边形的目的。

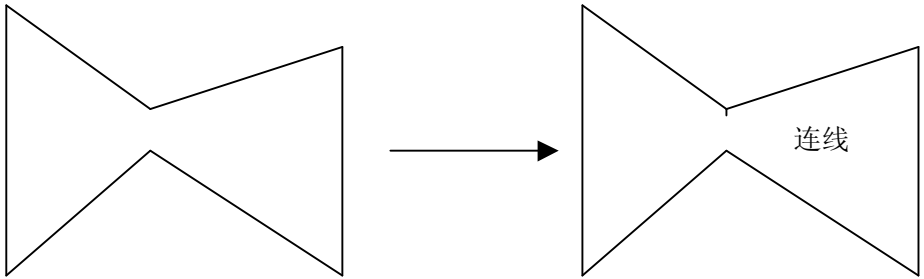

图 8-6 凹区域分成两个凸四边形

下面“消隐”程序设计中，利用远近法绘制圆环。把圆环看成由许许多多四边形基元构成，首先计算各处四边形的顶点坐标，计算各中心点的 Z 坐标并进行排序，根据 Z 值大小沿 Z 轴方向由远及近依次绘制出各四边形，形成整个图形，从而实现隐线处理。

8.4 消隐技术编程案例

一、程序设计功能说明

利用上述基本原理编制的“消隐”应用程序，包括消隐绘制凸多面体和利用远近法绘制的圆环。如图 8-4-1 所示。

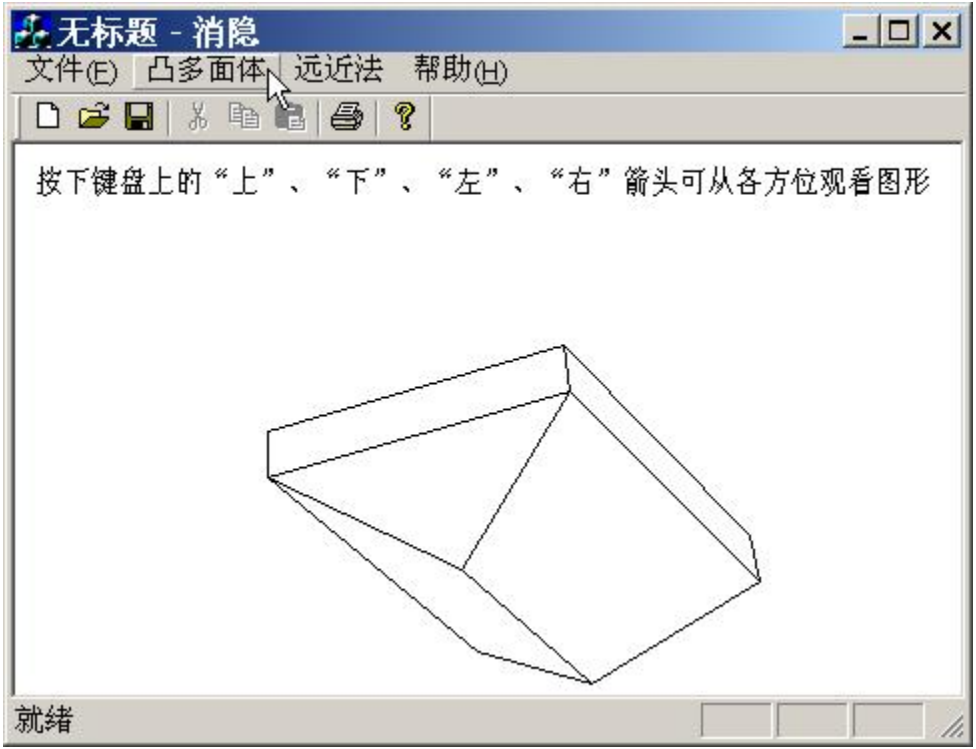
无标题 - 消隐
文件(F) 凸多面体 远近法 帮助(H)
按下键盘上的“上”、“下”、“左”、“右”箭头可从各方位观看图形
就绪

图 8-4-1

二、程序设计步骤

1. 创建单文档应用程序框架

2. 编辑菜单资源，添加消息处理函数

设计如图 1-4 所示的菜单项。在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 8.1 表中的定义编辑菜单资源，建立消息映射函数，完成有关的函数声明。

表 8.1 菜单项的消息处理函数

菜单标题	菜单项 ID	消息	消息处理函数

凸多面体	ID_TUMIANTI	CONMMAN	OnTumianti
	ID_	CONMMAN	On

3. 添加程序结构代码，在 CMyView.h、CMyView.cpp 文件中相应位置添加如下代码：

```
// 消隐 View.h : interface of the CMyView class
//
/////////////////////////////////////////////////////////////////////////////
```

程序代码见纸书

}

基类代码说明：

为绘制凸多边形与环面，与前面许多绘制方法构建基类不同，本程序通过构建了如下函数：Project(float X, float Y, float Z)、WLineTo(float X, float Y, float Z,CDC* pDC)、WMoveTo(float X, float Y, float Z,CDC* PDC)、ReadVertics()、ReadFaces()、VisionVector(int St1)、NormalVector(int St1, int St2, int St3)、ScaleProduct(float v1, float v2, float v3, float n1, float n2, float n3)、DrawFace(CDC* pDC)、 DrawObject()、VisionPoint()来实现。

8.10 练 习 题

1．消隐的意义

2．试简述单个凸多面体消隐的基本方法。

3．编程绘制经过消隐的正方体。