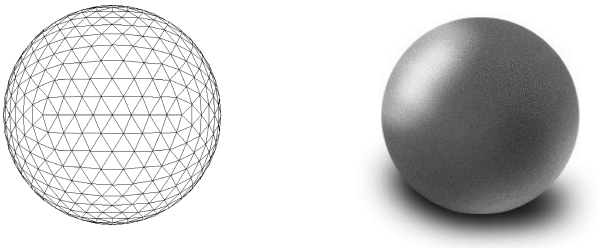
第 3 章　填充多边形

光栅扫描显示器的特点是它具有表示实区域(solid area)的能力。根据顶点的描述生成实区域的过程称为实区域光栅化,相应的算法称为区域填充算法。常用的区域填充算法分为两大类:光栅化算法与种子填充算法。

3.1　多边形的光栅化

计算机中早期表示物体的方法是线框模型。线框模型用定义物体轮廓线的直线或曲线绘制。线框模型并不存在面的信息,每一段轮廓线都是单独构造出来的。图 3-1(a)所示为球体线框模型。为了提升真实感效果,从 20 世纪 70 年代开始,计算机中物体的表示方法开始向表面模型转换。与线框模型相比,表面模型显得更加生动、直观,真实感更强。对线框模型添加材质属性后,根据场景中视点、光源的位置及朝向,先计算出多边形网格顶点的颜色,然后使用光滑着色模式填充每个多边形内部,便得到表面模型。球体表面模型如图 3-1(b)所示。



(a) 线框模型　　(b) 表面模型

图 3-1　球体的计算机表示法

3.1.1　多边形的定义

多边形是由折线段组成的平面封闭图形。它由有序顶点的点集 $P_i(i=0,\cdots,n-1)$及有向边的线集 $E_i(i=0,\cdots,n-1)$定义,n 为多边形的顶点数或边数,且 $E_i=P_iP_{i+1}(i=0,\cdots,n-1)$。这里 $P_n=P_0$,保证了多边形的闭合。多边形可以分为凸、凹多边形以及环,如图 3-2 所示。多边形具有顶点、边、面和法线等基本几何属性。

1. 凸多边形

含有凸点的多边形称为凸多边形。凸点对应的内角小于 180°。多边形上任意两顶点间的连线都在多边形之内。

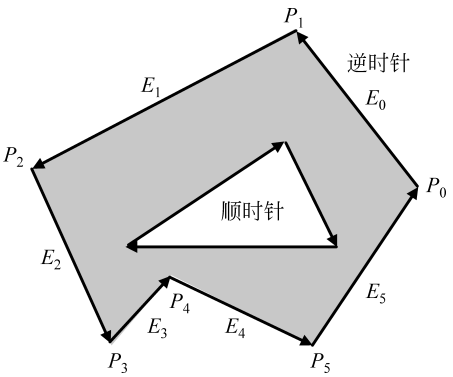


图 3-2　多边形的定义

2. 凹多边形

至少有一个凹点的多边形称为凹多边形。凹点对应的内角大于 180°。多边形上任意两顶点间的连线有不在多边形内部的部分。

3. 环

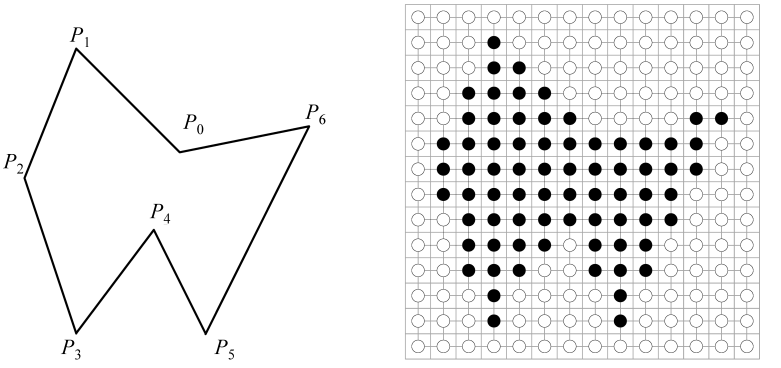
多边形内部含有另外的多边形称为环。如果规定每条有向边的左侧为其内部区域，则当观察者沿着边界行走时，内部区域总在其左侧。这就是说，多边形外轮廓线的环形方向为逆时针，内轮廓线的环形方向为顺时针。

3.1.2 多边形的表示

在计算机图形学中，多边形有两种表示方法：点元表示法与面元表示法。

1. 点元表示法

点元表示法是一种用多边形的顶点序列来描述多边形的方法，其特点是直观、占内存少、易于进行几何变换，但由于没有明确指出哪些像素位于多边形之内，所以不能直接进行填充。点元表示法是多边形线框模型描述的形式，如图 3-3(a)所示。



(a) 点元表示法　　(b) 面元表示法

图 3-3　多边形的表示法

2. 面元表示法

面元表示法是一种用位于多边形内部的像素点集来描述多边形的方法。这种表示方法虽然失去了顶点、边界等许多重要的几何信息，但便于直接读取像素来填充多边形。面元表示法是多边形表面模型描述的形式，如图 3-3(b)所示。

3. 多边形的光栅化

将多边形的描述从点元表示法变换到面元表示法的过程，称为多边形的光栅化。即从多边形的顶点信息出发，计算位于多边形轮廓线内部的各个像素点的信息，并按照扫描线顺序，将像素点颜色写入多边形中。

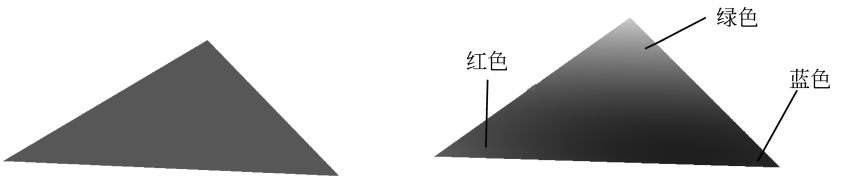
3.1.3 多边形着色模式

多边形可以使用平面着色模式(flat shading mode)或光滑着色模式(smooth shading mode)进行填充。无论采用哪种着色模式，都意味着要根据多边形的顶点颜色计算多边形内部各个像素点的颜色。回顾一下，直线的平面着色模式是使用一个顶点的颜色绘制直线，

例如使用起点颜色作为直线的颜色，可以绘制出单一颜色的直线。直线的光滑着色模式是使用两个顶点颜色来绘制直线，例如使用起点颜色和终点颜色的线性插值作为直线的颜色，可以绘制出颜色渐变的直线。

1. 平面着色模式

多边形的平面着色模式是指使用多边形任意一个顶点的颜色填充多边形，多边形具有单一颜色。图 3-4(a)为三角形的平面着色。三角形 3 个顶点的颜色分别为红色、绿色、蓝色。三角形的填充色取自第 1 个顶点颜色。



(a) 平面着色　　(b) 光滑着色

图 3-4　三角形着色模式

2. 光滑着色模式

多边形的光滑着色模式假定多边形顶点的颜色不同，多边形任意一点的颜色由各顶点的颜色进行双线性插值(bilinear interpolation)得到。基于顶点颜色的光滑着色模式也称为 Gouraud 光滑着色模式。Gouraud 是一名法国计算机科学家，以提出 Gouraud 着色模式而闻名。图 3-4(b)为三角形的 Gouraud 着色。三角形填充色为 3 个顶点颜色的双线性插值结果。

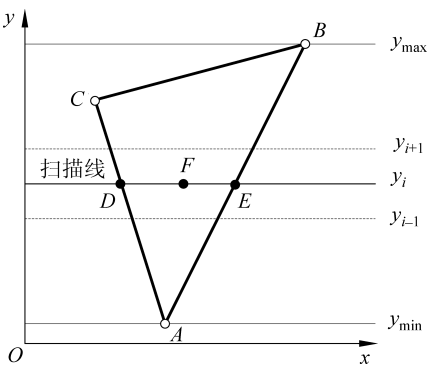


图 3-5　光滑着色模式

以图 3-5 所示三角形为例，说明 Gouraud 光滑着色算法原理。假定三角形 ABC 的 3 个顶点坐标为 $A(x_A,y_A)$，$B(x_B,y_B)$，$C(x_C,y_C)$。A 点的颜色为 c_A，B 点的颜色为 c_B，C 点的颜色为 c_C。在自定义坐标系中，y 轴向上为正。当前扫描线为 y_i，扫描线最小值为 y_{min}，最大值为 y_{max}，扫描线从 y_{min} 向 y_{max} 移动，执行 $y_{i+1}=y_i+1$ 操作。

当前扫描线上，D 点的颜色可以通过 A 点颜色与 C 点颜色的线性插值得到

$$c_D=(1-t)c_A+tc_C,\quad t\in[0,1] \tag{3-1}$$

当前扫描线上，E 点的颜色可以通过 A 点颜色与 B 点颜色的线性插值得到

$$c_E=(1-t)c_A+tc_B,\quad t\in[0,1] \tag{3-2}$$

当前扫描线上，DE 跨度内任意一点 F 的颜色通过 D 点颜色与 E 点颜色插值得到

$$c_F=(1-t)c_D+tc_E,\quad t\in[0,1] \tag{3-3}$$

随着扫描线从 y_{min} 向 y_{max} 移动，F 点会遍历三角形内部，其颜色为三角形的 3 个顶点颜色的双线性插值。

3. 马赫带

图 3-6 所示图形是一组亮度递增变化的平面着色矩形块。由于矩形块的亮度发生轻微

的跳变，边界处的亮度对比度增强，使得矩形轮廓表现得非常明显。1868 年奥地利物理学家 Mach 发现了这种明暗对比的视觉效应，称为马赫带效应。在亮度变化的一侧感知到正向尖峰效果，看到一条更亮的线；在另一侧感知到负向尖峰效果，看到一条更暗的线。马赫带效应不是一种物理现象，而是一种心理现象，是由人类视觉系统造成的。马赫带效应夸大了平面着色的渲染效果，使得人眼感知到的亮度变化比实际的亮度变化要大，如图 3-7 所示。一个具有复杂光滑表面的物体是由一系列多边形（主要是三角形和四边形）网格表示的。如果采用平面着色模式填充多边形，就会出现马赫带效应，边界特别明显。物体看上去就像是一片一片拼接起来的，显得很不真实。改善的方法是用光滑着色模式代替平面着色模式来填充多边形。

图 3-6　马赫带

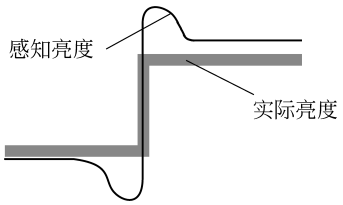


图 3-7　边界位置的实际亮度与感知亮度

3.2　边界像素处理规则

首先自定义二维坐标系，x 轴向右，y 轴向上，扫描线自下向上移动。由于 CDC 类的 Rectangle( ) 成员函数使用画笔绘制矩形的边界，使用画刷填充矩形内部。CDC 类的 FillSolidRect( ) 成员函数不使用画笔绘制边界，仅使用当前画刷填充整个矩形，包括左边界和下边界，但不包括右边界和上边界。本节以矩形的光栅化为例，说明多边形边界像素的处理规则。首先讨论以平面着色模式填充矩形，然后讨论以光滑着色模式填充矩形。

3.2.1　平面着色模式填充矩形

矩形由左下角点 $P_0(x_{\min}, y_{\min})$ 与右上角点 $P_1(x_{\max}, y_{\max})$ 唯一定义，如图 3-8 所示。在每条扫描线上，将矩形跨度内的所有像素都置成相同的颜色。填充单一的矩形时，从数学上讲可以填充矩形所覆盖的内部像素及全部边界像素。但当多个矩形连接存在共享边界

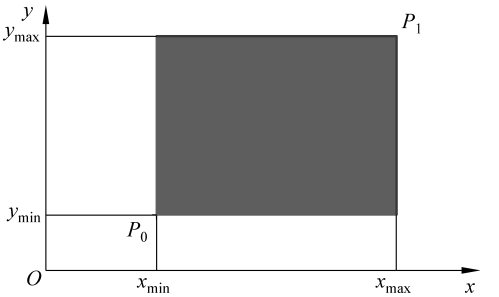


图 3-8　矩形填充效果图

时，就不能填充矩形的全部边界像素。所有的光栅化算法对运算符＜、＞和＝的使用都十分敏感。

3.2.2 处理边界像素

图 3-9 所示矩形 $P_0P_1P_2P_3$ 被等分为 4 个小矩形。假定左下方的小矩形 $P_0P_5P_8P_4$ 填充为绿色，右下方的小矩形 $P_5P_1P_6P_8$ 填充为黄色，右上方的小矩形 $P_8P_6P_2P_7$ 填充为绿色，左上方的小矩形 $P_4P_8P_7P_3$ 填充为黄色。4 个小矩形的公共边为 P_5P_8、P_8P_7、P_4P_8 和 P_8P_6。考虑到公共边 P_5P_8 既是小矩形 $P_0P_5P_8P_4$ 的右边界，又是小矩形 $P_5P_1P_6P_8$ 的左边界；考虑到公共边 P_4P_8 既是小矩形 $P_0P_5P_8P_4$ 的上边界，又是小矩形 $P_4P_8P_7P_3$ 的下边界，那么 P_5P_8 和 P_4P_8 作为相邻小矩形的共享边界，应该着色为哪个小矩形的颜色？同理，P_8P_7 和 P_8P_6 也作为相邻小矩形的共享边界，应该着色为哪个小矩形的颜色？如果对公共边不做处理，则可能将公共边先设置为一种颜色，然后又设置为另一种颜色。一条边界两次不同的着色会导致混乱的视觉效果。图 3-9 的正确处理结果如图 3-10 所示，每个小矩形的右边界像素和上边界像素都不填充，等待与其相连接的后续小矩形进行填充。最终，边界 P_5P_8 和 P_4P_8 填充为黄色，P_8P_7 和 P_8P_6 填充为绿色，而边界 P_3P_7、P_7P_2、P_1P_6 和 P_6P_2 并未进行填充。

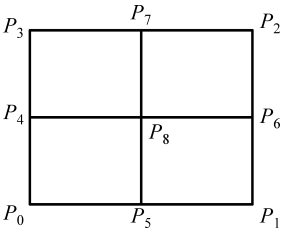


图 3-9 边界像素的问题

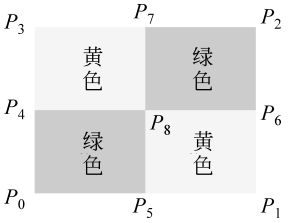
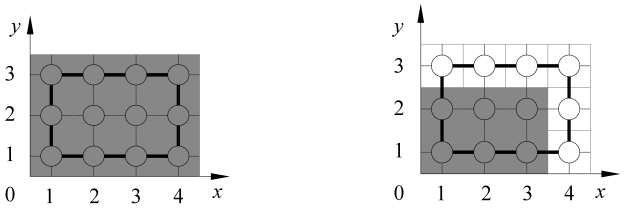


图 3-10 共享边界像素的处理

在实际填充过程中，也需要考虑到像素面积大小的影响：填充左下角为(1,1)、右上角为(4,3)的矩形时，若将边界上的所有像素全部着色，就得到图 3-11(a)所示的效果。矩形光栅化后的像素覆盖面积为 4×3 个单位，而实际矩形的面积只有 3×2 个单位，如图 3-11(b)所示。如果不填充矩形的上边界和右边界，则可以保证其面积为 3×2 个单位。



(a) 面积为4×3　　(b) 面积为3×2

图 3-11 根据像素计算矩形面积

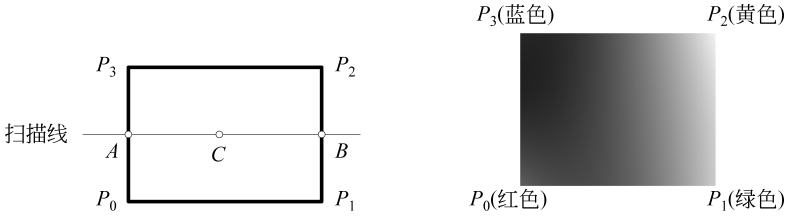
边界像素处理规则为，由一条边界确定的包含图元的半平面，如果位于该边界的左方或下方，那么这条边界上的像素就不属于该图元。可以将其简单表述为“左闭右开，下闭上

开”，即绘制矩形左边界和下边界上的像素，不绘制矩形右边界和上边界上的像素。共享的水平边将“属于”有共享边的两个矩形中靠上的那个；共享的垂直边将“属于”有共享边的两个矩形中靠右的那个。本规则适用于任意形状的多边形，而不是只局限于矩形。本规则会导致多边形遗失最上一行像素和最右一列像素，图形出现瑕疵。为了避免共享边界上的像素发生两次重绘，没有比这更好的解决方法。

3.2.3 光滑着色模式填充矩形

光滑着色模式不再以单一颜色去填充矩形，矩形内部填充颜色是 4 个顶点的颜色的双线性插值。这样越靠近顶点，顶点颜色就越突出。Gouraud 着色模式为矩形填充了光滑的渐变颜色。

图 3-12(a)中，假定 P_0 点的颜色为红色、P_1 点的颜色为绿色、P_2 点的颜色为黄色、P_3 点的颜色为蓝色。沿着 x 方向和 y 方向对顶点颜色进行双线性插值得到内点颜色，就可以绘制出颜色渐变的图像。下面以一条扫描线为例进行讲解，首先由 P_0 点的颜色与 P_3 点的颜色进行线性插值计算出扫描线上 A 点的颜色。由 P_1 点的颜色和 P_2 点的颜色进行线性插值计算出在同一条水平扫描线上 B 点的颜色。在该扫描线上，由 A 点的颜色和 B 点的颜色可以进行线性插值计算出 C 点的颜色。当扫描线从 P_0P_1 边界向上移动到 P_3P_2 边界时，在每条扫描线上，C 点从 A 点移动到 B 点，那么 C 点将遍历矩形覆盖的所有像素。根据“左闭右开”，矩形中每个跨度是在左边封闭而右边开放的区间内，不绘制每条扫描线的最右像素点。根据“下闭上开”规则，不绘制最上一条扫描线。矩形的光滑着色效果如图 3-12(b)所示。



(a) 设计图　　(b) 效果图

图 3-12　矩形的光滑着色

3.3 边标志算法

3.3.1 基本思想

由 Agkland 和 Weste 于 1981 年提出的边标志算法，不仅可以处理凸多边形，而且可以处理凹多边形。边标志算法分两步实现：第 1 步勾勒轮廓线。对多边形的每条边进行光栅化，亦即对多边形边界所经过的像素打上标志，在每条扫描线上建立各跨度的边界像素点对。第 2 步填充多边形。沿着扫描线由小往大的顺序，按照从左到右的顺序，填充像素点对所构成的跨度之间的全部像素。

3.3.2 光栅化边

扫描线是 y 方向连续的，根据扫描线的连续性，所有边均可使用第 2 个八分圆域的 DDA 算法进行光栅化。扫描线由边的低端（$y=y_{min}$）向高端（$y=y_{max}$）运动，交点的 y 坐标每次加 1，交点的 x 坐标加 $1/k$。k 是边的斜率。

边的低端坐标为(x_0,y_0)，边与下一条扫描线的交点为(x_{i+1},y_{i+1})。其中，$x_{i+1}=x_i+1/k=x_i+\Delta x/\Delta y=x_i+m$，$y_{i+1}=y_i+1$。交点相关性如图 3-13 所示。这说明，随着扫描线的移动，扫描线与有效边交点的 x 坐标，从起点开始可以按增量 $m=1/k$ 计算出来。

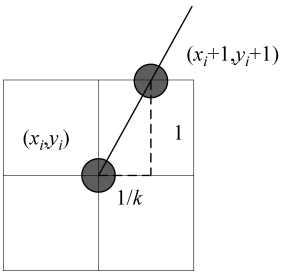


图 3-13　计算扫描线与边的交点

3.3.3 判断点与边的位置关系

使用向量叉积运算，可以判断点与边的位置关系。图 3-14(a)中，使用向量叉积，可以判断点 P_1 与边 P_0P_2 的位置关系。写为从 P_0 点发出的向量

$$\overrightarrow{P_0P_2}=\{x_2-x_0,y_2-y_0,0\},\overrightarrow{P_0P_1}=\{x_1-x_0,y_1-y_0,0\}$$

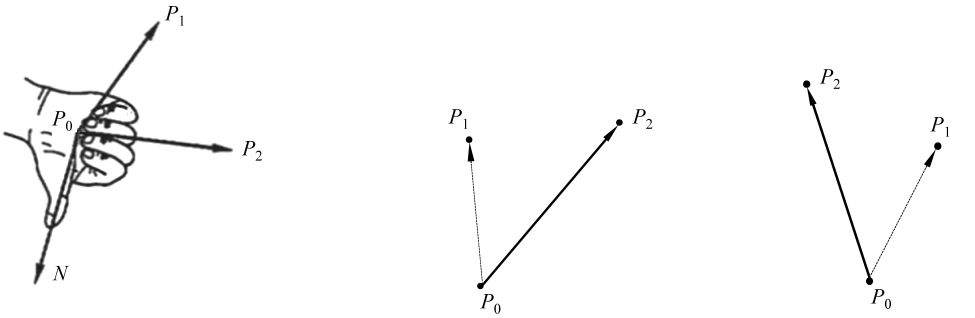
计算两向量的叉积 $\boldsymbol{N}=\overrightarrow{P_0P_2}\times\overrightarrow{P_0P_1}$，有

$$\boldsymbol{N}=\{0,0,(x_2-x_0)(y_1-y_0)-(y_2-y_0)(x_1-x_0)\}$$

假设 Δz 代表三角形法向量的 z 分量，有

$$\Delta z=(x_2-x_0)(y_1-y_0)-(y_2-y_0)(x_1-x_0) \tag{3-4}$$

如果 $\Delta z>0$，则 P_1 点位于边 P_0P_2 的左侧，如图 3-14(b)所示；否则，P_1 点位于边 P_0P_2 的右侧，如图 3-14(c)所示。



(a) 右手法则　　(b) 点位于边的左侧　　(c) 点位于边的右侧

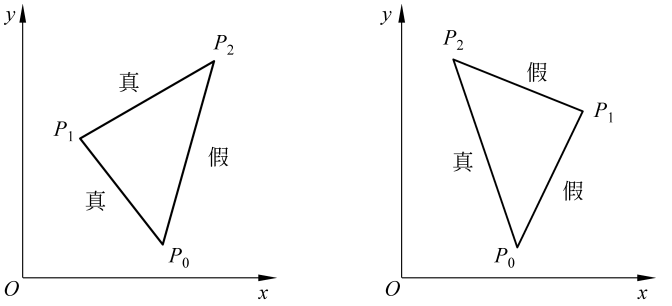
图 3-14　判断点与边的位置关系

说明：由于用到了向量的叉积，这里使用的是三维向量来表示平面二维向量，只是$\overrightarrow{P_0P_2}$和$\overrightarrow{P_0P_1}$向量的 z 分量为 0，而垂直于平面的法向量 $\boldsymbol{N}$ 的 x 分量和 y 分量为 0。

3.3.4 平面着色模式填充三角形

无论多么复杂的物体，最终都可以使用三角形小面逼近。解决了填充三角形的问题，就解决了物体的表面着色问题。三角形是一个凸多边形，扫描线与三角形相交只有一对交点，形成一个相交区间，称为跨度。本节以三角形的光栅化为例讲解边标志算法。

在三角形 $P_0P_1P_2$ 中，对顶点进行排序，使 P_0 点为 y 坐标最小的点，P_2 点为 y 坐标最大的点，P_1 点的 y 坐标位于二者之间。P_0P_2 称为三角形的主边。若 P_1 点位于主边左侧，称为左三角形；若 P_1 点位于主边右侧，称为右三角形。为了区分跨度的起点与终点，约定位于三角形跨度左侧的边的特征为真，位于跨度右侧的边的特征为假，如图 3-15 所示。三角形覆盖的扫描线的最小值为 $y_{\min}=P_0y$，最大值为 $y_{\max}=P_2y$。三角形所覆盖的扫描线条数 $n=y_{\max}-y_{\min}+1$。使用 DDA 算法，将三条边离散到标志数组 SpanLeft[n]和 SpanRight[n]中。SpanLeft 数组存放边特征为真的离散点标志，SpanRight 数组存放边特征为假的离散点标志。在图 3-15(a)中，SpanLeft 数组存放的是 P_0P_1 边与 P_1P_2 边的标志点，SpanRight 数组存放的是 P_0P_2 边的标志点；在图 3-15(b)中，SpanLeft 数组存放的是 P_0P_2 边的标志点，SpanRight 数组存放的是 P_0P_1 边与 P_1P_2 边的标志点。当扫描线从 $y_{\min}$ 向 $y_{\max}$ 移动时，基于标志数组内标志点的颜色，使用颜色线性插值算法计算跨度内每个像素点的颜色。填充时，根据“左闭右开”的规则，不填充每条扫描线上的最右一个像素；根据下闭上开的规则，不填充最后一条扫描线 $y_{\max}$。



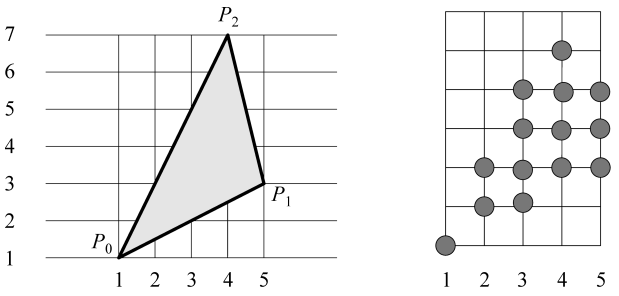
(a) 左三角形　　(b) 右三角形

图 3-15　三角形的分类

算法 9：平面着色的三角形填充算法

例 3-1　使用边标志算法填充图 3-16(a)所示的三角形，写出 SpanLeft 数组与 SpanRight 数组存储的标志。

从图中可知，三角形顶点为 $P_0(1,1)$、$P_1(5,3)$ 和 $P_2(4,7)$。根据三角形主边顶点坐标 P_0 和 P_2，可以计算出扫描线个数 $n=y_2-y_0+1=7$。定义左边标志数组为 SpanLeft[7]和右边标志数组为 SpanRight[7]。由于三角形为右三角形，所以 SpanLeft[7]数组存放 P_0P_2 边的标志，SpanRight[7]数组存放 P_0P_1 边和 P_1P_2 边的标志。这样，由于数组的下标索引从 0 开始，所以 SpanLeft[0]和 SpanRight[0]数组对存放三角形所覆盖的第一条扫描线上



(a) 三角形　　(b) 内存像素示意图

图 3-16　三角形离散为像素点

跨度两端的边标志；SpanLeft[6]和 SpanRight[6]数组对存放三角形所覆盖的最后一条扫描线上跨度两端的边标志。

第 1 条扫描线：SpanLeft[0]=(1,1),SpanRight[0]=(1,1)；

第 2 条扫描线：SpanLeft[1]=(2,2),SpanRight[1]=(3,2)；

第 3 条扫描线：SpanLeft[2]=(2,3),SpanRight[2]=(5,3)；

第 4 条扫描线：SpanLeft[3]=(3,4),SpanRight[3]=(5,4)；

第 5 条扫描线：SpanLeft[4]=(3,5),SpanRight[4]=(5,5)；

第 6 条扫描线：SpanLeft[5]=(4,6),SpanRight[5]=(4,6)。

图 3-16(b)中，边标志用黑色实心小圆表示，跨度内部的像素点用空心小圆表示。

3.3.5　光滑着色模式填充三角形

在平面着色模式填充三角形的基础上，设定三个顶点的颜色后，可以对三角形进行光滑着色。光滑着色是真实感图形的生成基础，可以使画面明暗自如、色彩丰富。假定，三角形顶点 P_0 的颜色为红色、P_1 的颜色为绿色、P_2 的颜色为蓝色，沿着边和扫描线方向对颜色进行双线性插值，绘制的三角形光滑着色效果如图 3-17 所示。

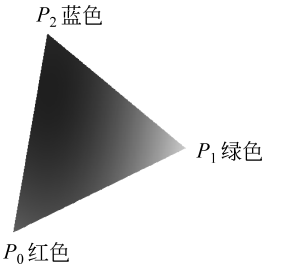
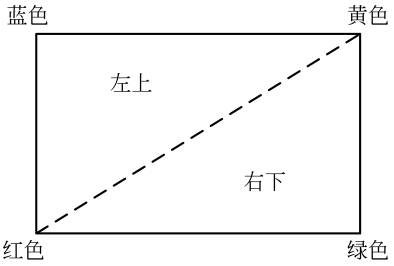


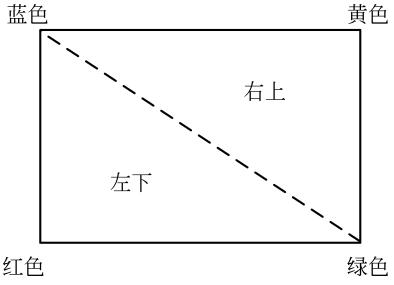
图 3-17　三角形光滑着色效果图

假定四边形的 4 个顶点颜色分别为红、绿、黄、蓝。四边形细分为左上和右下两个三角形，或左下和右上两个三角形，如图 3-18 所示。将其与图 3-12(b)进行对比后可以看出，虽然通过填充两个三角形可以完成填充四边形的任务，但二者的效果有一定差异，两个三角形填充的四边形有明显的分界线。尽管如此，OpenGL 或者 DirextX 中仍将四边形细分为两个三角形后，才分别进行填充。



(a) 左上与右下三角形划分法

(b) 效果图1



(c) 左下与右上三角形划分法

(d) 效果图2

图 3-18　四边形细分为左上和右下两个三角形，或左下和右上两个三角形

算法 10：光滑着色的三角形填充算法

3.4　有效边表算法

3.3 节介绍的边标志算法用于填充三角形。如果需要填充复杂的多边形，则可以使用 x 扫描线算法。该算法可以一次性完成复杂多边形的填充，而不用将多边形细分为三角形。

3.4.1　x 扫描线法

多边形分为凸多边形与凹多边形。扫描线与凸多边形相交只有一个跨度。扫描线与凹多边形边界可能会出现多个跨度。x 扫描线算法填充多边形的基本思想是按扫描线顺序，计算扫描线与多边形的相交区间，再用指定的颜色显示这些区间的像素。x 扫描线算法的核心是须按 x 递增顺序排列交点的 x 坐标序列。由此可以得到 x 扫描线算法步骤如下。

(1) 确定多边形覆盖的扫描线条数，得到多边形顶点的最小 y 值 y_{min} 和最大 y 值 y_{max}。

(2) 从 $y=y_{min}$ 到 $y=y_{max}$，每次用一条扫描线进行填充。对每条扫描线的填充过程可分为以下 4 个步骤。

① 求交：计算扫描线与多边形各边的交点。

② 排序：把所有交点按 x 坐标递增顺序进行排序。

③ 配对：将相邻交点配对，每对交点代表扫描线与多边形相交的一个跨度。

④ 着色：把这些跨度内的像素置为填充色。

x 扫描线算法在处理每条扫描线时，需要与多边形的所有边求交，处理效率很低。这是

因为一条扫描线往往只与多边形的少数几条边相交，甚至与整个多边形都不相交。若在处理每条扫描线时，把所有边都拿出来与扫描线求交，则其中绝大多数运算都是徒劳的。因此将 x 扫描线算法加以改进，形成有效边表算法，也称为 y 连贯性算法。

有效边表算法的基本思想是按照扫描线从小到大的移动顺序，计算当前扫描线与多边形有效边的交点，然后把这些交点按 x 值递增的顺序进行排序、配对，以确定填充区间。有效边表算法通过维护边表(edge table，ET)与有效边表(active edge table，AET)，避开了扫描线与多边形所有边求交的复杂运算，已成为最常用的多边形光栅化算法之一。有效边表算法可以填充凸多边形、凹多边形和环。

3.4.2 示例多边形

以图 3-19 所示的凹多边形为示例多边形，讲解有效边表算法。示例多边形的点元表示法为 $P_0(7,8)$、$P_1(3,12)$、$P_2(1,7)$、$P_3(3,1)$、$P_4(6,5)$、$P_5(8,1)$、$P_6(12,9)$。多边形覆盖的扫描线最小值为 $y_{min}=1$，最大值为 $y_{max}=12$。假定多边形各个顶点的颜色都相同，填充模式为平面着色。

图 3-20 中，扫描线 $y=3$ 与示例多边形有 4 个交点(2.3，3)、(4.5，3)、(7，3)和(9，3)。边界交点的整数坐标为(2，3)、(5，3)、(7，3)和(9，3)。每个跨度的最后一个整数像素分别为(5，3)和(9，3)，根据“左闭右开”规则不予填充。按 x 值递增的顺序对交点进行排序、配对后的填充区间为[2，4]和[7，8]，共有 5 个像素。为了避免填充[5，6]区间，填充时设置一个逻辑变量(初始值为假)进行区间内部外部测试(inside-outside test)。按 x 值递增的顺序，每访问一个交点，逻辑变量就取反一次。如果进入区间内部，逻辑变量为真；如果离开区间内部，逻辑变量则为假。填充逻辑变量为真的所有区间内的像素，这样可以对有多个跨度的扫描线进行正确处理。例如，进入区间[2，4]之内，逻辑变量为真；离开区间[2，4]后，逻辑变量为假。再次进入[7，8]之内，逻辑变量为真；离开区间[7，8]后，逻辑变量为假。

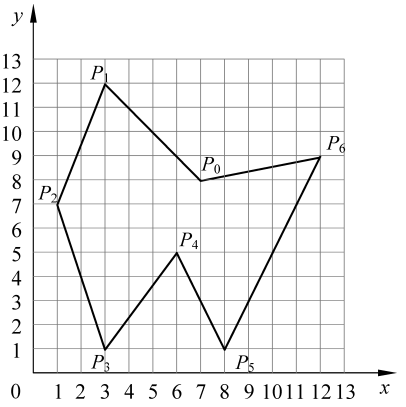


图 3-19 示例多边形

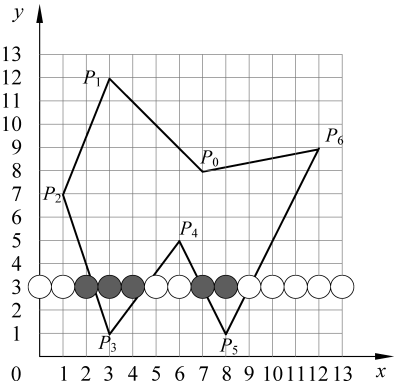


图 3-20 用一条扫描线填充示例多边形

3.4.3 顶点处理规则

示例多边形的顶点可以分为 3 类。局部最高点 P_1、P_6 和 P_4，共享顶点的两条边均落在顶点所在扫描线的下方；普通连接点 P_2，共享顶点的两条边分别落在顶点所在扫描线的

两侧；局部最低点 P_0、P_3 和 P_5，共享顶点的两条边均落在顶点所在扫描线的上方。处理时，常根据共享顶点的两条边的另一端的 y 值大于顶点 y 值的个数来将顶点个数分别置为0、1 和 2。事实上，根据“下闭上开”的处理规则，有效边表算法能自动处理这 3 类顶点。

1. 普通连接点的处理规则

图 3-21 中，普通连接点 P_2 是边 P_3P_2 的终点，同时也是边 P_2P_1 的起点，顶点个数计为 1。按照“下闭上开”的规则，P_2 点作为 P_3P_2 边的终点不予填充，但作为 P_2P_1 边的起点予以填充。

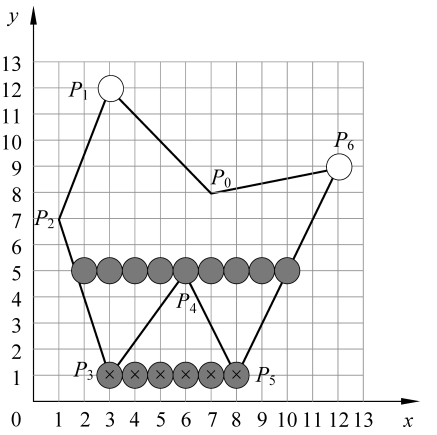


图 3-21　顶点的处理规则

2. 局部最低点的处理规则

P_0 点、P_3 点和 P_5 点是局部最低点。如果处理不当，扫描线 $y=1$ 会填充区间[3,8]，结果填充了 P_3 到 P_5 点之间的像素，如图 3-21 中 $y=1$ 扫描线所示。将局部最低点的顶点个数计数为 2。$y=1$ 的扫描线填充时，共享顶点 P_3 的 P_3P_2 边与 P_3P_4 边加入有效边表，所以 P_3 点被填充两次；同理，共享顶点 P_5 的 P_5P_4 边与 P_5P_6 边加入有效边表，P_5 点也被填充两次；共享顶点 P_0 的 P_0P_1 边与 P_0P_6 边加入有效边表，P_0 点也被填充两次。

3. 局部最高点的处理规则

局部最高点的顶点个数计数为 0。根据“下闭上开”规则，扫描线会自动放弃 P_1 点、P_4 点和 P_6 点。P_1 点与 P_6 点将不予填充，而 P_4 点被经过 P_3P_2 和 P_5P_6 边的扫描线 $y=5$ 予以填充，如图 3-21 所示。

3.4.4　有效边与有效边表

1. 有效边

多边形与当前扫描线相交的边称为有效边。在处理一条扫描线时仅对有效边进行求交运算，可以避免与多边形的所有边求交，提高了算法效率。

2. 有效边表

将有效边按照与扫描线交点 x 坐标递增的顺序存放在一个链表中，称为有效边表。有效边表利用了边的连贯性。

(1) 与扫描线 y_i 相交的边，多数与扫描线 y_{i+1} 相交。

(2) 从一条扫描线到下一条扫描线，交点的 x 值增量相等。有效边表的结点如图 3-22 所示。

x	y_{max}	$1/k$	next

图 3-22　有效边表的结点

图 3-22 中，x 为当前扫描线与有效边的交点；y_{max} 为有效边所在扫描线的最大值，用于判断该边何时扫描完毕后被抛弃而成为无效边；$1/k$ 为 x 坐标的增量，其值为斜率的倒数。

对于图 3-19 给出的示例多边形，扫描线 $y=1$ 至 $y=3$ 的有效边表如图 3-23 所示。

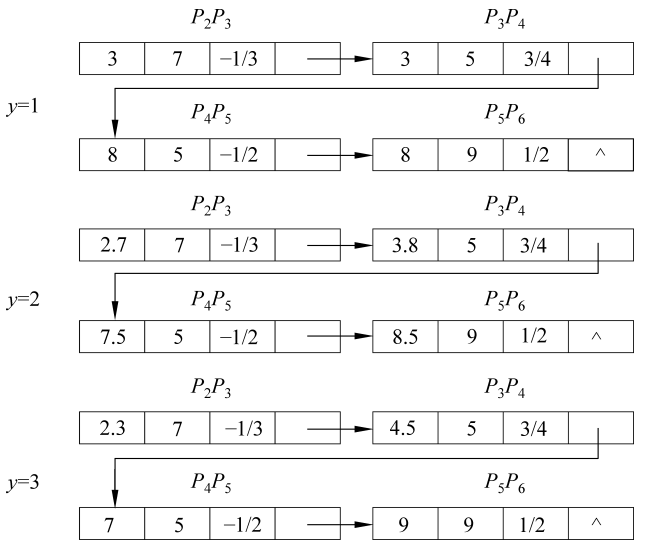


图 3-23　扫描线 $y=1$ 至 $y=3$ 的有效边表

$y=4$ 的扫描线处理完毕后，因为下一条扫描线 $y=5$ 与 y_{max} 相等，根据“下闭上开”的原则，把 P_3P_4 和 P_4P_5 两条边删除，如图 3-24 所示。

$y=6$ 的扫描线处理完毕后，因为下一条扫描线 $y=7$ 与 y_{max} 相等，根据“下闭上开”的规则，把 P_2P_3 边删除。

当 $y=7$ 时，添加新边 P_1P_2，如图 3-25 所示。

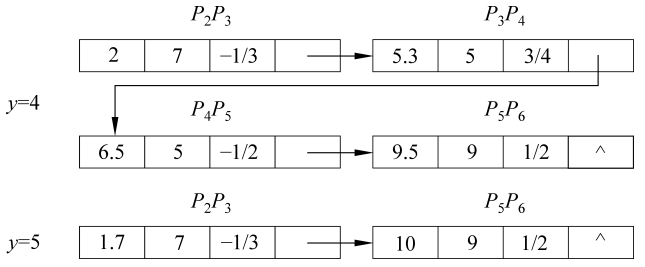


图 3-24　扫描线 $y=4$ 至 $y=5$ 的有效边表

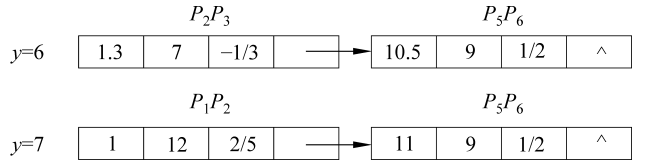


图 3-25　扫描线 $y=6$ 至 $y=7$ 的有效边表

当 $y=8$ 时，添加上新边 P_0P_1 和 P_0P_6，如图 3-26 所示。这条扫描线处理完毕后，因为下一条扫描线 $y=9$ 与 y_{max} 相等，根据“下闭上开”的规则，将 P_5P_6 边和 P_0P_6 边删除，如图 3-27 所示。

$y=11$ 的扫描线处理完毕后，因为下一条扫描线 $y=12$ 与 y_{max} 相等，根据“下闭上开”的

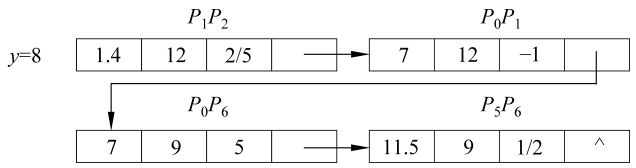


图 3-26　扫描线 $y=8$ 的有效边表

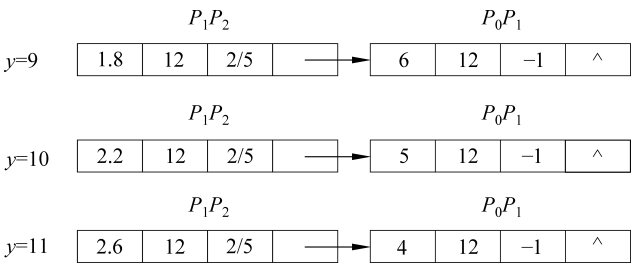


图 3-27　扫描线 $y=9$ 至 $y=11$ 的有效边表

规则，将 P_1P_2 边和 P_0P_1 边删除。

至此，给出了示例多边形的全部有效边表。

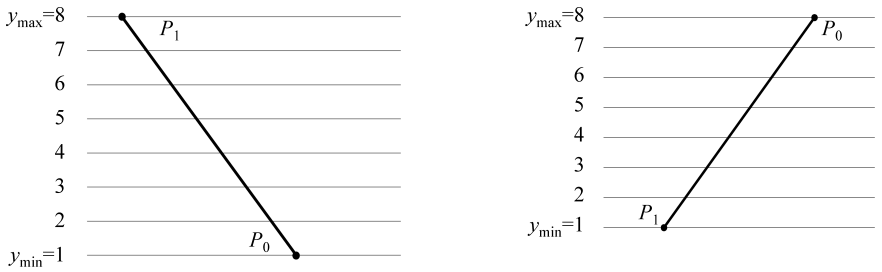
3.4.5　桶表与边表

从有效边表的建立过程可以看出，有效边表给出了扫描线与有效边交点坐标的计算方法，但是并没有给出新边出现的位置。为了确定在哪条扫描线上插入新边，就需要构造一个边表，用以存放扫描线上多边形各边出现的信息。因为水平边的 $1/k$ 为∞，并且水平边本身就是扫描线，在建立边表时可以不予考虑。

1. 桶表与边表的表示法

(1) 桶表(bucket table)是按照扫描线顺序管理边出现情况的一个数据结构。首先，构造一个纵向扫描线链表，链表的长度为多边形所覆盖的最大扫描线数，链表的每个结点称为桶，对应多边形覆盖的每条扫描线。

(2) 将每条边的信息链入与该边最小 y 坐标(y_{min})相对应的桶处。也就是说，若某边的低端点为 y_{min}，则该边就存放在相应的扫描线桶中。边的低端点与高端点的定义如图 3-28 所示。低端点的 y 坐标(扫描线)小，高端点的 y 坐标(扫描线)大。



(a) P_0点为低端点，P_1点为高端点　　(b) P_1点为低端点，P_0点为高端点

图 3-28　按照扫描线大小定义边的端点

(3) 对于每条扫描线，如果新增多条边，则按 $x|y_{min}$ 坐标递增的顺序存放在一个链表中，若 $x|y_{min}$ 相等，则按照 $1/k$ 由小到大排序，这样就形成边表，如图 3-29 所示。

$x\|y_{min}$	y_{max}	$1/k$	next

图 3-29　边表结点

图 3-29 中，x 为新增边低端点的 x 值，表示为 $x|y_{min}$，用于判断边表在桶中的排序；y_{max} 是该边高端点的最大扫描线值，用于判断该边何时成为无效边。$1/k$ 是 x 坐标的增量，即 $\Delta x/\Delta y$。对比图 3-22 与图 3-29，可以看出边表是有效边表的特例，即在该边低端点处的一个有效边表。有效边表与边表可以使用同一个 CAET 类来表示。

2. 桶表与边表示例

为了高效地将边加入到有效边表中，需要在初始化时建立一个包含多边形所有边的边表。边表是按照桶排序的方式建立的，有多少条扫描线就有多少个桶。在每个桶中，根据边的低端的 x 坐标，按照增序的方式排列每条边。对于图 3-19 给出的示例多边形，桶表与边表结构如图 3-30 所示。

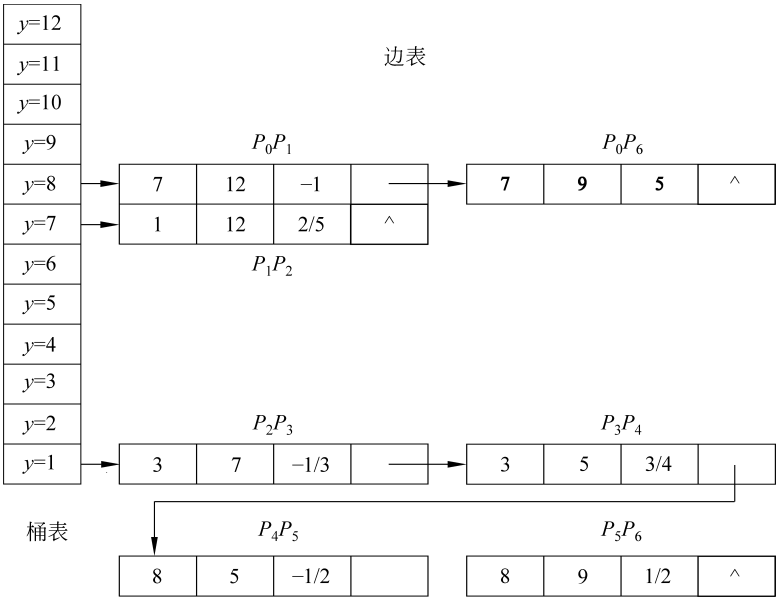


图 3-30　示例多边形的桶表与边表

算法 11：有效边表填充算法

例 3-2　使用有效边表算法填充图 3-31 所示的三角形，写出边表与各条扫描线的有效边表。

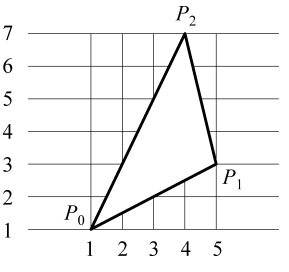


图 3-31　三角形

边表如图 3-32 所示，在第一条扫描线上出现两条边，在第 3 条扫描线上出现一条边。有效边表如图 3-33 所示，在第 3 条扫描线上抛弃 P_0P_1 边，在第 7 条扫描线上抛弃 P_0P_2 边和 P_1P_2 边。

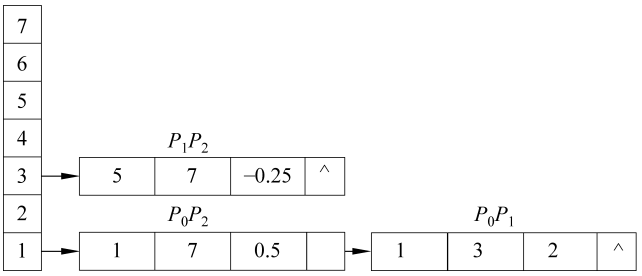


图 3-32　边表

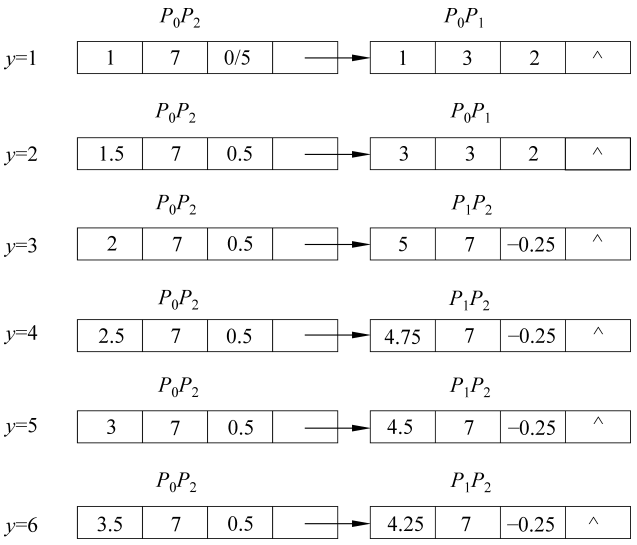


图 3-33　有效边表

3.5　边填充算法

3.5.1　填充原理

有效边表算法填充多边形的优点是多边形内的每个像素仅被访问一次。由于扫描线上每个跨度的两端点像素在填充前就已经确定，因此可以对跨度内的像素进行光滑着色；有效边表算法的缺点是维护和排序各种表的开销太大。

Agkland 和 Weste 提出的另一种填充算法是边填充算法。边填充算法是先求出多边形的每条边与扫描线的交点，然后将交点右侧的所有像素颜色全部取为补色。边填充算法中，边的顺序无关紧要。按某个顺序处理完多边形的所有边后，就完成了多边形的填充任务。

边填充算法利用了图像处理中的“取补”的概念，对于黑白图像，取补就是将白色的像素取为黑色，反之亦然；对于彩色图像，取补就是将背景色取为填充色，反之亦然。取补的一条基本性质是一个像素经过两次取补就恢复为原色。如果多边形的内部像素取补奇数次，则显示为填充色；如果取补偶数次，则保持为背景色。

3.5.2 填充过程

假定边的顺序为 E_0、E_1、E_2、E_3、E_4、E_5 和 E_6，如图 3-34 所示。这里，边的顺序并不影响填充结果，只是方便编写算法的循环结构而已。边填充算法处理过程如图 3-35 所示。

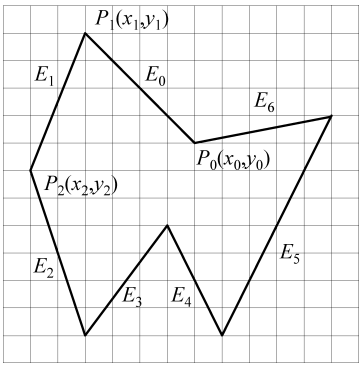


图 3-34　标注了边顺序的示例多边形

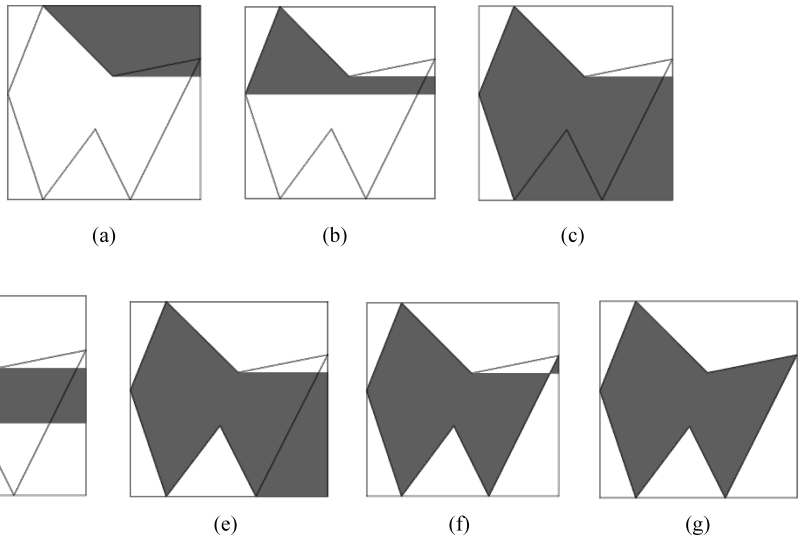


图 3-35　边填充算法执行过程

对于 E_0 边，边的低端点为 $P_0(x_0,y_0)$，高端点为 $P_1(x_1,y_1)$。该边扫描线的最小值为 $y_{\min}=y_0$，最大值为 $y_{\max}=y_1$，斜率为 $k=\dfrac{y_1-y_0}{x_1-x_0}$，边上当前扫描线的坐标为 x_i，边上下一条扫描线的坐标为 $x_{i+1}=x_i+1/k$。在扫描线沿着该边从 y_0 向 y_1 移动的过程中，将该边右侧像素的颜色全部取补，即将 E_0 边右侧的像素全部置为填充色，如图 3-35(a)所示。对于 E_1 边，边的低端点为 $P_2(x_2,y_2)$，高端点为 $P_1(x_1,y_1)$。该边扫描线的最小值为 $y_{\min}=y_2$，最大值为 $y_{\max}=y_1$，斜率为 $k=\dfrac{y_2-y_1}{x_2-x_1}$。在扫描线沿着该边从 y_2 移动到 y_1 的过程中，将该边右侧像素的颜色全部取补，即将 E_1 边与 E_0 边之间的像素置为填充色，而 E_0 边右侧的像素经过两次取补恢复为背景色，如图 3-35(b)所示。按照某个顺序处理完多边形的每条

边后，填充过程就结束了。

边填充算法特别适合于具有帧缓冲的显示器，可以按任意的顺序处理多边形的每条边。当所有的边处理完毕后，按扫描线顺序读出帧缓冲的内容并送显示设备。边填充算法的优点是不需要任何数据存储，缺点是复杂图形的许多边经过同一条扫描线，导致一些像素可能被访问多次。算法的效率受到边右侧像素数量的影响，右侧像素越多，需要取补的像素也就越多。为了减少边右侧像素的访问次数，可以在多边形的包围盒内进行像素取补，如图 3-36 所示。包围盒就是包含该多边形的最小矩形，即用多边形在 x、y 方向的最大值和最小值作为顶点绘制的矩形。为了提高效率，有时也可以在多边形内添加一条边界，称为栅栏，如图 3-37 所示。这便是 Dunlavey 于 1983 年提出的栅栏填充算法。为了计算方便，栅栏通常取过多边形某一顶点的垂线。栅栏填充算法在处理每条边与扫描线的交点时，只将交点与栅栏之间的像素取补。若交点位于栅栏左侧，将交点之右，栅栏左侧的所有像素取补；若交点位于栅栏右侧，将交点左侧、栅栏右侧的所有像素取补。图 3-38 给出了使用栅栏填充算法填充示例多边形的执行过程。

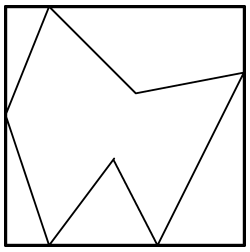


图 3-36　带包围盒的多边形

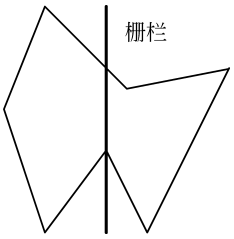


图 3-37　带栅栏的多边形

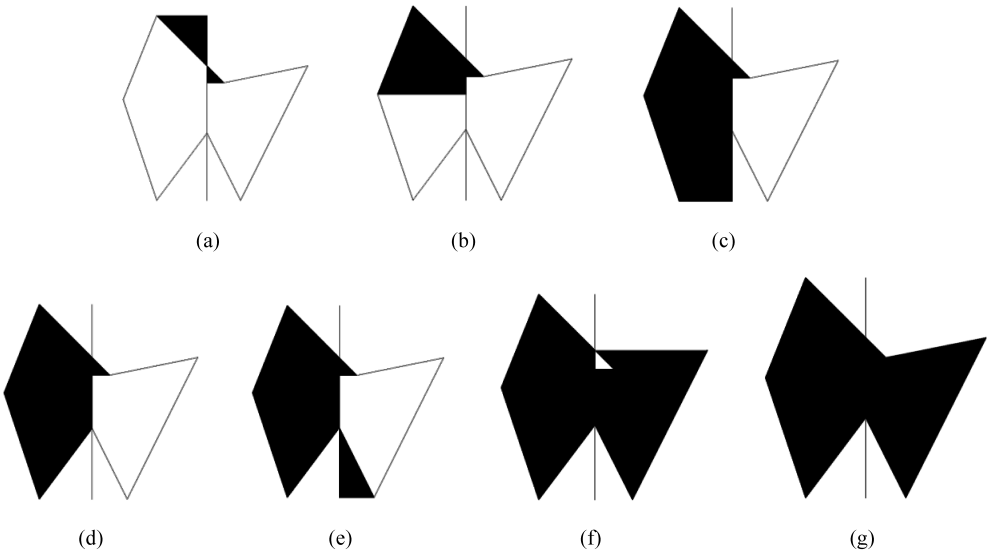
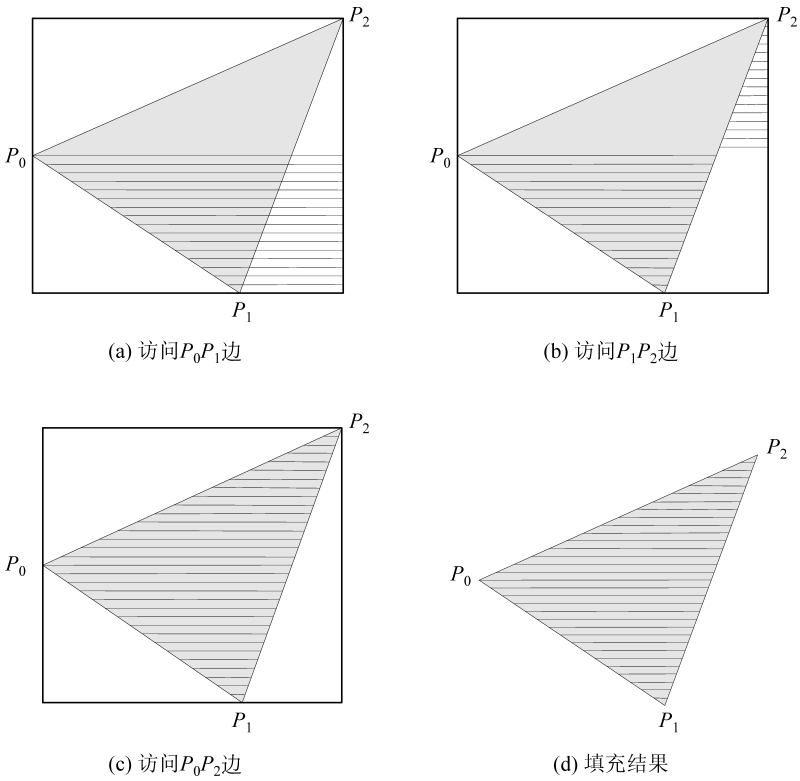


图 3-38　栅栏填充算法执行过程

算法 12：边填充算法

例 3-3　在窗口客户区内，使用边填充算法填充图 3-39 所示的三角形。



(a) 访问P_0P_1边　(b) 访问P_1P_2边

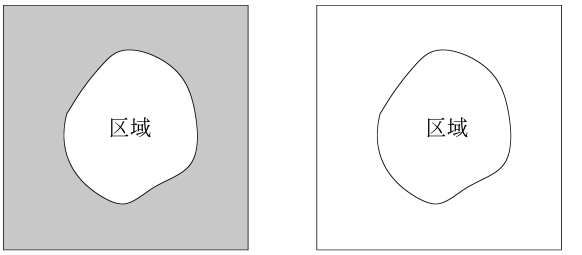
(c) 访问P_0P_2边　(d) 填充结果

图 3-39　边填充算法的原理

3.6　区域填充算法

前面讨论的填充算法都是按照扫描线顺序对多边形进行着色，而区域填充算法则采用了完全不同的策略。区域填充算法假设，区域内部至少有一个像素是已知的，将该像素（称为种子像素）的颜色扩展至整个区域。区域是指相互连通的一组像素的集合，因为只有在连通域内，才可能将种子像素的颜色扩展到其他像素点。区域可以采用内点表示与边界表示两种形式。如果区域是用内点表示的，那么区域内的所有像素具有同一种颜色，区域外的像素具有另一种颜色，如图 3-40 所示；如果区域是用边界表示，区域内部像素与边界像素具有不同的颜色，区域外部的像素可以与内部像素同色或不同色，如图 3-41 所示。基于内点表示的填充算法称为泛填充算法（flood fill algorithm）。基于边界表示的填充算法称为边界填充算法（boundary fill algorithm）。泛填充算法与边界填充算法都是从区域内的一个种子像素开始填充，所以统称为种子填充算法（seed fill algorithm）。无论是采用内点表示还是边界表示，区域均可以划分为四连通域与八连通域。要定义四连通域与八连通域，首先要定义一个像素的四邻接点与八邻接点。

图 3-40　区域的内点表示



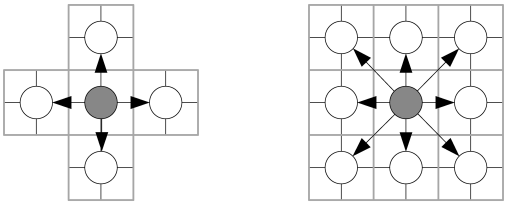
(a) 外部与内部不同色　　(b) 外部与内部同色

图 3-41　区域的边界表示

3.6.1　四邻接点与八邻接点

1. 四邻接点定义

对于区域内部任意一个像素，其左、上、右、下 4 个相邻像素称为四邻接点，如图 3-42(a)所示。



(a) 四邻接点　　(b) 八邻接点

图 3-42　邻接点定义

2. 八邻接点定义

对于区域内部任意一个像素，其左、左上、上、右上、右、右下、下和左下 8 个相邻像素称为八邻接点，如图 3-42(b)所示。

3.6.2　四连通域与八连通域

1. 四连通域定义

如果从区域内部任意一个种子像素出发，通过访问其水平方向、垂直方向的四个邻接点就可以遍历整个区域，则称为四连通(4-connected)区域，如图 3-43 所示。

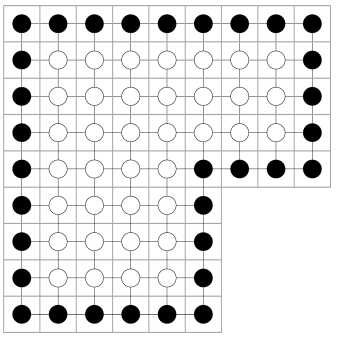


图 3-43　边界表示的四连通域