计算机图形学的概念与方法

柳朝阳
郑州大学数学系

Contents

Chapter 1

计算机图形的构成及其表示

1.1 点阵图形及其表示

让我们先了解最先看到的计算机图形. 最直接的, 我们首先从显示器上看到计算机产生的图形. 显示器的屏幕由可以发光的像素点组成. 并且从几何位置看, 所用这些像素点构成一个矩形的阵列. 利用计算机控制各像素点按我们指定的要求发光, 就构成了我们需要的图形. 这种方式构成的图形我们可称之为点阵图形. 我们利用计算机控制各像素点按我们指定的要求发光的方法当然需要使用各种各样的计算机图形生成软件或通过计算机语言编程来实现. 像素点的实际发光则是电子学与光电学的技术实现的, 不属于我们所讨论问题的范围. 我们首先需明白的是像素点的形状、位置及其发出的光, 才能正确的控制它们, 构成我们需要的计算机图形.

1.1.1 点阵图形的大小

显示器的最常用的参数之一是其分辨率. 这个参数常写成800×600, 1024×800或640×480的形式. 这实际上说的是全屏幕显示的点阵图形的大小. 它们的具体含义是指显示器的水平方向的一行上有800, 1024或640个像素点, 垂直方向的一列上有600, 800或480个像素点, 全屏幕显示的点阵图形由800×600, 1024×800或640×480个像素点构成. 而实际的点阵图形可以比全屏幕显示的点阵图形小, 也可以比全屏幕显示的点阵图形大. 显示器上像素点的排列方式如图1.1所示, 只是像素点比较小以至肉眼难以分辨.

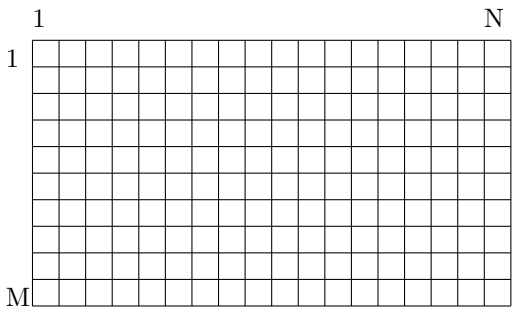


图1.1 点阵图形像素点及其排列方式

一般情况下都明确指出点阵图形每一行每一列像素点的个数N, M, 并用$M \times N$表示相应的点阵图形的大小.

1.1.2 像素点的形状

数学理论中, 一个点是只有位置, 没有大小的. 但无论什么情况下, 要表示出一个点, 这个点一定是有大小有形状的. 只不过不同的情况下, 不同的场合, 这个点的大小形状有所不同. 如果所有的像素点按要求发光或者不发光, 点阵图形的几何形状就是相当于描绘在如图1.1所示的一张方格绘图纸上. 只不过在显示器上, 这个方格非常小, 我们的肉眼很难区分一个一个的小方格.

需要指出的是这个方格可以是一个矩形, 因此其长宽比, 即长度与宽度比不一定是1. 例如，当我们调节计算机显示器的显示区时，可从显示区的长宽比变化中观察到这一变化。这个长宽比在不同的计算机软件系统中也可能不同. 不难想象, 同一个点阵图形, 当像素点的长宽比改变时, 我们看到的点阵图形的长宽也改变了. 如果我们曾经对同一副图形在不同的计算机上上进行处理时, 可能会曾经发现图形自己变了. 像素点的长宽比不同就是可能的原因之一. 这种情况下我们就可能需要知道具体的长宽比以校正图形的这种畸变. 幸好大多数情况下, 我们总是在同一种计算机系统中用相同的计算机软件处理计算机图形, 因而不用关心长宽比到底是多大.

1.1.3 像素点的颜色表示

从图形的色彩看, 黑白图是最简单的图形了. 显示器上显示出一副黑白图图形, 就说明了一些像素点黑, 另一些像素点白. 具体到一个像素点, 仅可以是黑或白二者之一. 这正好与计算机所用存储器里的最小的存储单元的"位"(bit)类似. 存储器里每个位的取值可用关或开来表示. 所以黑白图的每个像素点的颜色只占用存储器的一个存储单元就可以了. 我们用关或开来表示黑或白. 若1为开, 0为关, 则黑或白就用0或1来表示.

如果图形由黑、白、灰三种颜色的像素点构成, 则存储器单元的一位就无法表示了. 这是我们至少需两个位来表示. 两个位可以有00, 01, 10, 11四种组合状态, 我们可以用00, 11, 01分别表示黑, 白, 灰三种颜色. 这时还剩下一个组合10可用来表示一种新的颜色, 同时又没有占用更多的存储单元. 如果我们这时限定只能显示三种颜色, 而不是四种颜色也未尝不可, 但没有充分发挥显示器应有的显示潜力, 是一种资源的浪费. 因此我们见到的最简单的彩色显示器是四色的, 而不是三色的.

如果像素点的颜色用三或四个位来表示, 则可表示出的颜色数目就是$2^3 = 8$或$2^4 = 16$. 于是进一步就有了8色的或16色的显示器. 再进一步的解释就需要提到存储单元的另一个, 也是最常用的计量术语–字节.

计算机的存储单元的最常用的计量术语是字节. 存储单元的8个位组成一个字节(bytes). 顺便提起, 有千字节(Kilobytes, 通常简称为K)及兆字节(Megabytes). 其含义有时被理解为1千字节为1000个字节, 1兆字节为1百万个字节. 但实际上, 1千字节为$2^{10} = 1024$个字节, 其更准确的名称应是$1K$字节. 1兆字节为$2^{10}K = 1024K$字节.

如果像素点的颜色用一个字节, 即8个位来表示, 则可表示出的颜色数目就是$2^8 = 256$. 于是再进一步就有了256色的显示器.

随着科学技术的进一步发展, 计算机所用存储器的容量也越来越大, 现在的显示器可用24位, 即3个字节来表示一个像素的颜色, 能表示的颜色的数目是2^{24}, 超过一千六百万. 如此多种颜色表示出的图形真可称之为真彩色的图形了. 人的肉眼所能分辨的颜色的数目远远低于一千六百万.

1.1.4 像素点的位置

首先我们给所有的点阵图形中的像素点一个固定的顺序:从第一行的的第一个开始, 依次是第二个, 第三个, ... , 直到第一行的最后一个;然后是第二行各像素点, 第三行各像素点, ..., 直到最后一行像素点. 在保存各像素点的信息时, 只要按如上顺序储存, 则在读出点阵图形的信息时, 根据相应信息的顺序位置数, 即可判断出相应的像素点在点阵图形中的位置. 因此, 各像素点在点阵图形中的位置不需要存储空间. 当然, 前提是点阵图形的大小及其每行每列像素点的个数已知的.

1.1.5 点阵图形及相应文件构成

从上面的分析可以看出, 要想正确地显示出一副点阵图形, 计算机必须知道它的这些特征: 点阵图形的大小尺寸, 各个像素点的数据, 单个像素点的数据占用的存储器的位数.

前面对点阵图形是结合显示器来介绍的. 这只是为了方便介绍和理解. 实际上, 点阵图形可以是一个抽象的模型, 用在许多不同的地方. 比如, 打印机上打印出的图形就是按照这种方式组织的. 后面进一步介绍时, 不要把点阵图形局限于显示器上显示出的图形, 事实上, 这也是不可能的.

一个最直接的问题是显示器在工作时显示出图形. 当它停止工作时就不显示图形了. 但我们常常需要把显示出的图形保存下来. 计算机当然可以把图形的信息以文件的形式保存下来. 于是就有了点阵图形的文件保存问题. 最直接的当然就是采用上面介绍的显示器显示点阵图形的数据的构成形式进行保存. 这种格式保存的图形文件就是我们常用的以BMP作扩展名的所谓位图文件. 实际工作中为了适应对点阵图形要求的不同, 以及它所起作用的不同, 就有了多种不同的格式保存点阵图形的点阵图形文件纪录格式. 一般情况下并不需要详细知道各种格式图形文件的精确细节, 但是要知道某个具体的计算机图形处理软件并不能正确处理各种格式的图形文件. 因此可能会碰到一个正确的图形文件不能被正确处理的情况.

使用点阵图形, 经常需要消耗大量的存储器存储空间. 以文件形式保存点阵图形时, 消耗所需存储器空间大小实际上就是文件的大小(文件所占用的字节数). 因而可以说影响点阵图形文件大小的因素主要有三个:

(1). 点阵图形的大小

点阵图形的大小与文件大小之间的关系很清楚, 图形越大, 文件也越大.

(2). 图形颜色的种类数

点阵图形颜色的种类越多, 表达每个像素点颜色占用的存储器的位数就越多, 因而文件也就越大. 同样大小的点阵图形, 24位真彩色图文件的大小是黑白图的24倍, 256色点阵图形文件的大小是黑白点阵图形的8倍. 因为对每个像素点所需存储器的位数而言, 黑白点阵图形需1位, 24位真彩色点阵图形需24位, 为黑白点阵图形的24倍; 256色点阵图形需8位, 为黑白点阵图形的8倍. 这就说明, 当我们只需要黑白图形时, 最好不要把点阵图形作为彩色图形来保存, 以免无为的占用存储空间. 要知道, 计算机, 或者更确切的说, 是具体的某个计算机图形处理软件本身, 常常并不自行判定要处理的图形是彩色图或黑白图, 而是需要由用户来明确指定的. 而黑白图作为彩色图来保存, 并没有什么益处, 只是多占用存储空间而已.

(3). 文件格式

点阵图形文件的格式对占用存储器储存空间, 因而对文件的大小也有很大影响. 在上面提到的两个影响存储空间大小因素相同的条件下, 有些格式

的文件存储时, 利用了图像的压缩技术, 因而文件很小. 有些没有利用这一技术, 文件就比较大. 但压缩后的图形一般都需要复原, 复原后的图形常常有一定程度的失真, 造成图形质量的一定程度的下降. 所以即让是利用了压缩技术, 由于对图形质量要求的不同, 压缩的比例也会有很大的不同. 显然是图形质量要求越高(低), 压缩的比例越小(大). 一般来说,大图形缩小图形不失真; 反之,小图形放大,或多或少会有一定程度的失真.

1.1.6 点阵图形的坐标系统

虽然前面提到像素点在点阵图形中的位置不需要存储空间, 但是我们还是应当注意到存在着一个坐标系统. 各像素点也确实有一个坐标唯一指定了它的位置. 如果点阵图形的大小是$N \times M$, 那么它的点阵共有M行N 列, 每个像素点的位置就由它所在的行和列的位置所唯一确定. 这个行和列的位置就给出了点阵图形的坐标系统. 按照前面的顺序, 第m行, 第n列的像素点顺序数就是$m + (n-1)N$. 反之, 顺序数为s的像素点在第$s\ Mod\ N$行, 第$Int(\frac{s}{N}) + 1$列, 这里的$s\ Mod\ N$是s除以N后的余数, $Int(\frac{s}{N})$ 是$\frac{s}{N}$的整数部分.

需要注意的是第m行, 第n列的像素点的坐标可能不是(m, n), 而是$(m-1, n-1)$. 这是因为有时为了在计算机中处理的方便, 像素点的行列的排序不是从1, 而是从0开始的. 我们常用的显示器的像素坐标就是如此.

1.1.7 点阵图形的精度及相关问题

笼统地说点阵图形的精度, 大多数情况下是指我们看到的或得到的点阵图形的精度. 但我们所得到的点阵图形却是计算机经过多次加工处理后的图形. 加工处理的过程会制约着看到或得到图形的精度. 所以要解释清楚点阵图形的精度, 有必要了解计算机对图形的加工处理过程.

在用计算机处理图形时, 我们大多数情况下看到的是显示器上的图形. 显示器的分辨率就是在给定屏幕面积内可以显示和区别得开的点数, 它制约着我们在显示器上看到的图形的精度。但一幅点阵图形可以在出现于显示器上之前存在. 比如用扫描仪扫描生成的图形, 用其它计算机系统生成的或其他任何方法生成的点阵图形, 可以以点阵图形文件的格式存在于当前使用的计算机上. 这个点阵图形文件本身在生成时一定有一个精度. 点阵图形生成时的精度就由其生成的方法所决定. 比如用扫描仪扫描生成的图形, 就由扫描仪的精度所决定. 用其它计算机系统生成的的点阵图形的精度就由该计算机系统生成图形的精度所决定. 这其中可包括图形生成软件的精度(或更确切的, 软件的图形生成算法的精度); 或相应的这个计算机的显示器的精度(分辨率), 如果图形是根据显示在显示器上的图形生成的话.

如果我们用其他图形输出设备输出图形, 也有一个输出图形的精度. 比如用打印机或绘图仪输出图形时, 输出的图形的精度就受打印机或绘图仪精度的制约.

设想我们有图形文件, 有显示器, 有打印机或绘图仪. 它们的精度不一致. 如果图形文件的精度高于显示器, 打印机或绘图仪的精度, 那么显示器上显示出来的图形的精度仅取决于显示器的精度;打印机或绘图仪输出的图形的精度仅取决于打印机或绘图仪的精度. 这也就是说如果打印机或绘图仪的精度高于显示器的精度, 尽管我们可能看到了一副较粗糙的图形, 但仍然有可能通过打印机或绘图仪得到一副较精细的图形. 反之, 如果打印机或绘图仪的精度低于显示器的精度, 我们只能在显示器上看到, 而不能得到一副较精细的图形.

当然, 也很显然地如果图形文件本身的精度很低, 无论显示器, 打印机或绘图仪的精度多高, 我们都无法看到或得到一副比较精细的图形.

1.2 向量图形及其表示

1.2.1 向量图形的表示

很明显地, 用点阵图形表示出一幅图形时, 图形的结构显得非常简单: 图形是由许许多多的具有不同颜色的点构成的. 绘出一幅图就是在指定的图形显示介质上指定各个点的颜色. 指定一幅图形的大小后, 表示图形的文件(占用存储空间)的大小和复杂程度是完全一致的, 如一条直线段构成的图形和一幅有山水人物的风景画图形, 不因图形本身的简单或复杂程度而有所改变. 所不同的只是生成相应图形时方法的复杂程度.

对于由一条直线段构成的图形, 生成非常简单. 我们只需要知道直线段的起点和终点就可以了. 直线段上的所有点可以通过直线段的数学模型由计算机去自动给出. 这种方式给出了一种不同于点阵图形表示的图形生成方法:给出一些基本的数据, 并依此为依据由计算机用一定的命令程序去生成图形. 此种方式生成的图形就称为向量图形.

向量图形有不同于点阵图形的描述方式: 描述图形的几何形状的数学模型及依此模型生成几何图形的计算机命令. 这个不同之处既是向量图形的优点也是向量图的缺点产生的根源. 后面会对此作进一步解释.

工程用图大多是由一些基本的几何图形构成的, 如平面上的点、线、圆以及椭圆等等, 空间的长方体、球体等等, 当然也有一些复杂的数学模型描述的图形, 如现在很常见的各种各样的流线型物体. 复杂图形的描述可以通过足够多的基本几何图形的组合给出.

当图形比较简单时, 向量图的优点十分明显: 绘一条直线段, 只需要两个端点; 绘一个圆, 只需要圆心和半径. 如果用点阵图表示, 就需要绘出直线段或圆上的每一点. 但对于一个复杂的图形, 向量图形会变得非常复杂, 需要给出许许多多的基本图形, 许许多多的生成各种基本图形的命令. 而点阵图形的表示并没有变得更加复杂.

1.2.2 向量图形的颜色

点阵图形由许许多多的点构成, 每个点可以有各自独立的颜色. 类似地, 向量图形由各个基本图形构成, 因此一般来说, 各个基本图形有各自独立的颜色. 这也就是说, 在我们用点阵图形的形式表示出一个向量图形时, 构成向量图形的某个基本图形(如直线段, 圆, 正方形等等)的所有点应有一个颜色. 基于这样一种考虑, 在描述一个基本图形(如直线段, 圆, 正方形等等)时, 同时要描述其相应的颜色.

由此看来, 无论采用什么样的颜色模式, 描述向量图的文件的大小没有变化(除非是单色图, 可以省略对图形颜色的描述). 相比较而言, 点阵图形的颜色信息是加在每一个像素点上的, 颜色模式越复杂, 颜色信息越多, 表示点阵图形的文件就越大.

1.2.3 向量图形DXF格式文件构成

AutoCAD为CAD软件事实上的标准，作为主要的CAD应用已有许多年，自从DOS 问世，AutoCAD就差不多开发好了。最新版本的AutoCAD已可用于Windows及Macintosh操作系统。

DXF(绘图互换格式)是作为一种元图语言（Metafile Graphics Language），可允许用户把绘出的图用于其它应用或是其它操作系统。大多数的CAD软件，在各种平台上，可以在不同程度上操作DXF格式。即使不是CAD专业应用时，如向量编辑程序，一些桌面出版软件等，也都使用DXF格式。

AutoCAD功能强，并且生成的图形精确度高，就是互换格式的操作比较困难。AutoCAD中有些功能不是所有其它软件都具有的。有些相同的功能在不同的软件的操作中也可能会有完全不同的表示方式。下面我们以介绍DXF文件结构为例来说明向量图形文件的构成.

DXF文件可以是以ASCII字符的形式给出的, 有如下几部分给出.

成对的数据组（Group Pairs）: DXF文件中的物体的描述是由成对的数据组构成的，每一对数据有一个代码和代码项目的具体内容两个部分构成.一组命令码后跟着一组数据。一组命令码用来指明随后一组值是用来画什么的。

分段（Sections）：DXF文件中的数据分成如下所述的四段。

(1). 头部段（Headers Section）: 总文件中的信息(所画物体的颜色及缺省线宽，总体尺寸等)。向量编辑及桌面出版软件会把它省略掉，因为这是只适用CAD软件的变量。

(2). 列表段（Tab1e Section）: 主要供其它CAD程序应用。用来处理坐标系统及物体分层，可以分组形成相近邻的层次。

(3). 块（B1ock section）: 把一些基本图形组合成组并已赋予名字即称之为块，在AutoCAD中为实体（entities）。它与列表段信息层不太相像，块把实体处理为成组物体，它可以作为一个整体被修改, 相当于用户自己定义的基本几何图形。

(4). 实体部分（The Entities Section）: 它含有大量向量实体的叙述，DXF实体的例子为点，线，圆，实心体，三维面文本及形状。实体是由给定的数值及ASCII字符码构成。首先为实体名称码，随后是处理该实体所在的层，实体颜色码(如需要可有其它供选择的码)，最后是叙述该实体的数据。

例如，一条直线段的实体叙述可以用下列格式：

0 [0起头，标志开始实体叙述]
LINE
8 [接着是分层名]
LayerName
62 [接着是颜色]
0
10 [接着是起点X轴坐标]
5. 0000000000
20 [接着是起点的Y轴坐标]
5. 0000000000
30 [接着是起点Z轴坐标]
15. 0000000000
11 [接着是终点X轴坐标]
5. 0000000000
21 [接着是终点的Y轴坐标]
25. 0000000000
31 [接着是终点Z轴坐标]
15. 0000000000

再例如，一个圆的实体叙述可以用下列格式：

0 [0起头，标志开始实体叙述]
CIRCLE
8 [接着是分层名]
LayerName
62 [接着是颜色]
0
10 [接着是中心X轴坐标]

5. 0000000000
20 [接着是中心的Y轴坐标]
5. 0000000000
30 [接着是中心Z轴坐标]
5. 0000000000
40 [接着是圆的半径]
1. 0000000000

图形中同类型的实体（entity）的数据类型及数量是可以改变的。一个实体的开始总是用代码0表示, 随后是实体的名称; 第一个点总是用10，20及30来表示它在X，Y及Z方向上的位置代码。跟着每个代码后面的实际位置数据, 是可变的. 需要多个点时, 点的各个坐标分量的代码依次增加1, 如对直线的终点的各个坐标分量的代码就是11, 21, 31.

DXF文件中只有基本图形的几何信息, 能够使用DXF文件的有关软件中包含着利用这些信息生成几何图形的命令.

1.3 点阵图形和向量图形的特点

1.3.1 图形的整体放大

向量图形最基本的优点是它本身是由精确的数据给出的, 所以可以充分利用各种输出图形的设备的分辨率因可能精确地输出图形。也正因为如此，向量图的尺寸可以任意变化而不损失图像的质量。

向量命令只是简单地命令输出设备创建一个给定大小的图形物体，而采用尽可能多的“点”。换句话说，输出设备输出的“点”越多，同样大小的物体图形就更光滑细腻。

相反，输出图形的设备上输出一个点阵图时，其光滑度就会受到输出图形的设备分辨率和点阵图形本身的分辨率的双重限制。因为点阵图详细地规定生成多少个像素点，像素点数不随输出设备的分辨率而有可能被改变。我们考虑如下几种情形:或者让点阵图形中的点对应地作为打印机的一个点打印输出, 则打印机分辨率越高时，打印的点阵图圆越小(用于像素点的单个点变得更小，圆就收缩了)。或者是图形的大小尺寸不变，分辨率越高的打印机将用更多的点来打印每一个像素点(所以高分辨率的打印机也不能打印出比原图更细腻的图形);而分辨率越低的打印机将用更少的点来打印每一个像素点, 甚至于当打印机的分辨率很低时, 点阵图形的多个点要合并为一个点来打印(打印出的图形就会失去原图的一些细节上的精确性)。

换句话说，不管向量图的大小如何选择，在输出时，总是由输出打印机的分辨率来决定图的平滑或尖锐程度。出现这种差别的原因在于实际的图形可以看作是由无数多没有大小的点构成的. 向量图形可以根据图形的构成原理复制出图形上的任何一个点, 而点阵图形只是用有限个点去近似地表示出这个图形. 因此, 当打印输出向量图形时我们就可以尽可能的利用打印机的能力去近似原图;当打印输出点阵图形时我们就只能尽可能的利用打印机的能力去近似点阵给出的图形而不是近似原图.

上述讨论也说明一个问题, 即点阵图形不能任意放大, 把点阵图形放大实际上是要求构成点阵图形的每个点应以更大的尺寸表示出来. 当点阵图形放大到足够使人的肉眼分辨出来放大后的单独的一个像素点时, 就不会是一幅视觉效果上还不太差劲的图形了, 从这样放大后的图形甚至于不能得到原始图形的外貌轮廓(这样的技术在某些特定场合当然也是十分有用的, 比如现在的电视节目中为了隐藏某些画面及人物的细节而经常采用的马赛克画面就属于这样的技术处理). 而向量图形可以依据具体的图形输出设备, 如打印机的分辨率, 或者说是

其能绘出的点的大小来调整所需要的点的多少, 因而就不会出现这种结果. 对于向量型的图形输出设备, 如绘图机, 绘图笔可以沿直线轨迹或圆弧轨迹运动而绘出相应的图形, 就更不会产生这样的问题了.

1.3.2 图形的缩小

向量图形是通过数据描述的, 因此向量图的放大缩小首先表现为数据的放大缩小. 除非放大缩小的倍数为0这一无意义的特殊情况, 放大缩小后的数据是可以复原的. 这就意味着放大缩小后的向量图形是可以复原的. 图形具有这样的性质似乎是理所当然的. 但点阵图形却不具有这样的性质. 设想一个有点极端的例子, 一个点阵图形收缩得很小, 以至于一个像素点就可以把它表示出来了. 这时收缩后的图形就是由一个单色点给表示出来了. 如果这时我们把它放大, 也只是一个大的单色区域, 而不是原来的具有丰富色彩的图形.

所以在实际应用重要注意点阵图形放大缩小后不能完全复原的特点, 在需要时注意保留原始的点阵图形, 避免带来不必要的麻烦.

1.3.3 图形的局部放大

向量图还有另一个基本优点：在向量图中可以只编辑其中某一个单个物体而不影响图中的其它物体。例如，想缩小或放大向量图中的某个物体，只要选中它，进行缩放则可。向量图物体间可以互相覆盖而不会互相影响。在点阵图中要编辑它的单个物体时很困难，原因是点阵图内无物体元素。这就是说如果你在点阵图形中看到了一个圆, 你只是碰巧看到有某些共同特征的点是在一个圆上. 点阵图形只是分别地表示出了各个点的特征, 但并没有总结出这些点有共同组成某个圆的特征, 因此也并不把它们视为一个圆而进行整体的处理. 如果要缉点阵图某一部分时，就要首先选择定义该区域内的像素点，然后才能修正该区域内的像素点组成的图像。

1.3.4 点阵图形与向量图形的自身及其相间互转换

图形可以分成点阵和向量图形两大类型, 相应的图形文件可以分成点阵和向量图形两大类型. 具体实用时由于应用目的的不同, 因而会有技术处理上的不同, 带来点阵图形本身有许多不同的实现方式, 给出各种不同格式的点阵图形文件. 对向量图形来说也如此. 有很多原因需要把一种图形文件格式转换成另一种文件格式，如在应用中有的计算机不懂该图形文件格式，或要给中心服务器或另一个计算机用户提供特定的图形文件格式，各种技术性说明书中会介绍文件格式转换的技术细节，这里只是介绍一些基本常识。

1.3.4.1. 不同格式的点阵图形文件的相互转换

在不同的点阵图形文件格式中相互转换是比较容易的。点阵图形文件是用一种指定方法组织起来的一列像素点数据，内容结构简单，这种文件转换时，只要从原点阵图形中读出像素点的数据，换个新格式再写入文件就可，新格式仅仅是组织一列像素点数据的新方法而已。能够提供种种格式转换的软件当然需要清楚这两种格式的点阵图形文件组成的技术细节.

需要注意的是能表示的颜色多的文件格式被转换成颜色少的文件格式时，颜色数据的损失会导致图形质量的下降, 甚至面目全非，如把24位的颜色格式转换为仅仅只有8位颜色的格式时，损失了大多数颜色信息，结果是所有像素点的颜色都要改变。如果转换为黑白两色图, 颜色改变就更大.

1.3.4.2. 由点阵图形文件转换成向量图形文件

有两种完全不同的方法可把点阵图形文件转换成向量图形文件。其一是把点阵图形转换成点阵物体, 即把点阵图形作为一个基本图形嵌入在向量图形中. 这种处理类似于我们通常把一幅画粘贴在不同的背景中, 是一种较简单的处理方式。大多数向量文件格式里能包括点阵图形物体、而点阵图形大部分为一堆像素点数据及如何处理这些数据的指令。

要注意在这种转换中类似于点阵文件转换为点阵文件，可能会有一些点阵信息的损失. 如果向量格式不能保持原点阵格式的图形的分辨率，那么图形尺寸就会与要求的不一样。再就是类似于如点阵图互相转换为点阵图文件时减少所用的颜色的情况一样、将会产生不堪设想的改变。但用户对这类问题是无能为力的。用户能做到的只是另选用一种功能更强的向量格式来转换点阵图形文件。

把点阵图形文件转存为向量图形文件后，还有一个问题就是不能够再编辑它。大多数的向量图形编辑程序没有工具能编辑点阵物体中的单个元素。解决的方法之一使用点阵图形编辑软件编辑原始的点阵图形, 再重新转换.

把点阵文件转换成向量文件时一个完全不同的方法是可以把点阵图形中用某种特征划分的各组像素点转换成与基本几何图形类似的向量图形。如可以选用CoreITRACE软件跟踪点阵图形。该程序能自动跟踪一组颜色相同或相似的像素点，然后建立起相同颜色组像素点的向量轮廓线。这样转换的向量图形才是一个真正意义的向量图, 就能用向量图形编辑程序对图形进行编辑。

点阵图形不会总保证产生一个合适的向量图形。如果原点阵图中轮廓分明, 线条清晰, 这也就是说点阵图中像素点组之间有明显的边界时，向量跟踪法的效果好，可以得到一个合适的向量图形。而像照片一类的图像的点阵图形，由于像素点之间的颜色变化平滑，通过向量跟踪生成的物体的几何图形很难辩认。这时用向量法跟踪的结果就不好。

1.3.4.3. 由一种向量格式转换成另一种向量格式

不同格式的向量图间的相互转换问题太多。向量格式是由对各向量物体的描述构成的：直线，曲线，填充图案及文本等。问题在于不同的格式描述向量结构的基本几何图形时甚至对最简单的图形的描述方法都会完全不同。

当一个程序作向量图形格式转换时，很象一般语言翻译，在此种向量图形文件中用它自己的向量语言读出命令及对物体描述，进行翻译，把译出的内容用新的向量图形格式描述。这种翻译过程自然会产生许多毛病。若该程序读出一个物体描述后没有直接对应的向量格式作翻译，程序会选用新向量格式来近似地描述该物体图形，例如，翻译程序读出一个圆的描述，但新格式中却没有对圆的描述，程序就会设法用许多短折线的多边形来描述这个圆，于是圆变成了一个适当的多边形。如果无法较好地近似该物体图形, 程序也许会放弃该物体图形。

无论如何，一种向量格式转成另一种向量格式时，难免有部分图形被改变了，图形越复杂。物体被丢失或被改变的就越多。唯一的解决方法是用向量图形编辑程序来打开转换过的新文件，校正这些错误。

1.3.4.4. 向量图形转换成点阵图形

把向量图形转换成点阵图形是用得最频繁的图形转换方法. 我们在显示器上看到向量图形时, 看到的就是已经转换成点阵图形的向量图形. 点阵打印机输出的也是已经转换成点阵图形的向量图形。

把向量图形转换成像素点集合，软件首先要解释文件中的向量命令，确定向量图形的整体构成（换句话说，要确定哪一个物体在前面哪一个物体在后面），然后再建立该图的点阵格式的图形。成功与否与解释软件有关。如果软件不能解释向量格式的某一物体的叙述，它就可以忽略该物体或者就退出转换。

转换的成功与否及转换的速度直接与向量物体的复杂程度有关，另外也与所需要的点阵图形的分辨率有关，分辨率越高，则转换成的点阵图越复杂，结果是转换所需的时间也越长，文件也越大。不管向量图形中有复杂的三维图形或者仅有一块黑色区域图，只要转换后的点阵图的尺寸和分辨率相同，最后生成的点阵文件大小总是相同的。

向量图形转换成点阵图形时需要指定点阵图形的分辨率，理论上这个分辨率越高越好, 因为分辨率高, 转换成的点阵图形的质量就好. 但分辨率高也有不利的一面, 那就是转换时间长, 生成的点阵图形文件占用较多的存储空间. 所以分辨率的确定常常是根据实际需要综合决定的, 如应该与输出设备的输出分辨率一致。比如激光打印机分辨率为300dpi，点阵图的分辨率就没有理由要高于300dpi，因为比300dpi高时，产生多余的像素点，而低于300dpi就会产生“锯齿”形成不自然的图形。如果用半色调图像打印机输出的话，要想输出的点阵图质量好，应调节点阵图形的分辨率来适合打印机的容量及限制(实际工作中常碰到分辨率过高的点阵图形不能为打印机打印输出, 好在高分辨率的图形可以通过大多数图形软件重新编辑, 产生一个分辨率较低的图形, 然后用打印机打印输出)。

习题1

1. 一幅黑白点阵图形文件转存为16色的彩色阵图形文件后，图形质量是否得到加强？文件的大小是否有变化？为什么？
2. 一幅工程图型适合用哪种图形保存？为什么？一幅彩色照片呢？
3. 为什么点阵图形放大后可能失真？缩小呢？
4. 同类图形转换后会失真吗？
5. 什么是显示器的分辨率？
6. 一幅尺寸为300×400的图形实际尺寸为每米一个像素点，显示在显示器上是每厘米150个像素点。其实际尺寸和显示尺寸各是多少？
7. 一个圆的点阵图显示在像素点的高宽比不是1的显示器上，看上去还是一个圆吗？
8. 一个人的肖像照片的点阵图显示在像素点的高宽比不是1的显示器上。高宽比为何值时，此人看起来比实际的人本身要胖一些？何时看起来会瘦一些？
9. 一幅彩色照片需要50种不同的颜色，每个像素点的颜色值最少需要用多少位表示才能分辨出不同的颜色？

Chapter 2

点阵图形的基本算法

在前一章我们讨论了点阵图形构成原理。点阵图形构成的最小单元是像素点.但通常我们绘图, 包括绘精确的工程图和绘艺术创作图, 很少会单独的去描绘一个一个的像素点.我们通常作的是绘出一条条直线、曲线、圆、圆弧, 涂抹一块块各种颜色的区域, 或者由各种线条围成的区域.这也就是说, 常常是每一次对具有某种共同特征的大量的点作同样的图形处理.因此, 必须采用专门算法以绘出一条条直线、曲线、圆、圆弧, 涂抹一块块各种颜色的区域.本章我们就讨论点阵图形显示系统中的一些基本算法，包括直线、圆、多边形的生成算法和区域填充算法。虽然我们大多数人很少有机会如此直接生成这些图形，计算机内部却是大量的按照这样的方式产生图形，甚至文字信息。因此，了解这些不必然经常直接应用的知识对我们正确理解计算机处理和表示的图形是大有帮助的。

2.1 引言

2.1.1 基本图形的点阵转换

在点阵图形显示系统中，图形信息是以各个像素点值的形式存入存储器的。因此每一幅图形都可以看成是一个二维的像素点阵。确定，或者说生成一幅图就是要确定像素点阵中每一点的颜色信息。现实中我们要讨论的图形却常常是用几何参数来描述的.因此，在点阵图形显示系统中，就必须根据图形的几何描述，确定二维像素点阵中，哪些像素点是正好在图形上或最靠近图形，以便通过这些像素点描绘出相应的几何体的图形。这一过程就称之为给定图形的点阵转换。例如，一直线段可用其两个端点的坐标来描述，对它进行点阵转换就是要确定在二维像素点阵中，哪些像素点位于该直线段上，或最靠近它，以便显示出该直线段。这样的算法在点阵图形系统中使用得相当频繁，每建立或修改一幅图形，一般要用数百次甚至数千次。而且，转换过程中涉及比较和计算的像素点数量是相当大的，因此这些算法不仅必须要建立视觉效果比较好的图形，还必须要执行得尽可能地快。评价一个转换算法的优劣可以通过如下三个方面来进行:

(1). 所显示图形的精度. 转换出的点阵图形毕竟只是对原始图形的近似, 有一定的误差, 这个误差的大小可根据实际需要而定.

(2). 算法的时间复杂性, 也就是算法的速度.

(3). 算法的空间复杂性, 即算法运行过程所需要的内存空间的大小。

上述要求之间常常是相互冲突的, 主要表现为算法的速度和所显示图形的质量等，这两者的处理始终是一个权衡折衷的问题。有些算法速度快但产生的几何体有锯齿形的边缘, 因而效果不好; 而另一些算法速度较慢但产生的图形画面细腻边缘比较光滑，因而效果好. 通常我们都是在满足一定的图形画面质量的要求下追求尽可能快一些的算法。速度快慢不同的两类算法可根据需要, 应用在不同的地方. 在要求实时响应时(如图形的实时编辑处理过程中, 此时重要的是快速地显现出图形画面的大概轮廓和效果), 应使用速度快的算法; 在要求质量高而愿意多花时间时(如最终的结果图形)应使用生成图形质量高的方法, 即让是处理速度慢一些也可以.

2.1.2 描绘线条图形的要求

在计算机绘制图形时，直接地要用到大量的直线线段, 间接用到的直线线段更多。实际上许多曲线也需要利用一系列短直线段有效地逼近而得到。因此，直线描绘质量的好坏，将从根本上影响计算机图形设计的质量和水平。因此我们必须设计良好的直线扫描转换算法. 这样的方法有以下一些要求和准则:

(1) 线段端点位置应该准确

由于算法的累积误差和设备的精度导致端点位置偏移，使得在直线和由直线组成图形的绘制中产生间隙和不连接的情况，如可能造成相连接的一条线段的终点和另一条线段的起点之间留有间隙，从而使输出图形的质量和应用受到很大影响。一个显明的失误是这类问题可能导致绘出一条线段后, 再反方向绘出它, 是两条不同的线段。所以应采取适当的方法, 尽可能精确地保证直线段端点的精确度.

(2) 线段亮度均匀

以在显示器上显示的图形为例，显示出的直线段的亮度正比于单位线段长度内所显示的像素点的个数。如果输出的像素点密度不均匀，就会从直观上给人以一段亮(密度高)一段暗(密度低)的感觉。要保持亮度均匀，像素点就应该沿直线段, 而不是$x-$或$y-$坐标轴方向均匀分布。否则, 如果显示的是不同宽度及转角的直线时，点的密度难以保持均匀，因此会造成线的亮度不均。例如，由于斜向线上所包含的像素点比垂直和水平线上的像素点要稀少，发光点分布不均匀，并由此造成斜向直线的亮度自然比垂直和水平线的亮度低。

要要达到上述要求, 通常需要采用特殊的方式进行处理和补偿。但比较一般的方法是这样的: 直线上或曲线上的一点产生一个位移时, 相应的在两个坐标分量上各自也产生一个位移. 始终要求直线上或曲线上的位移是等距的比较麻烦, 但我们可要求直线上或曲线上的位移相应的两个坐标分量位移的较大者为1, 以保证直线或曲线段是由相邻的像素点构成的.这样输出的像素点密度就比较均匀, 线段亮度也就比较均匀.后面介绍的一些算法就是体现了这样的想法.

(3) 转换算法速度快

图形具有的信息虽很大，而在点阵图形中描绘直线是以像素点的形式进行的，要使画线的速度快，则必须在画线算法、显示及处理系统硬件等方面给予支持。在理想情况下，有关描绘直线的计算应该由专用硬件来完成。

2.2 直线点阵转换算法

直线点阵转换算法的目标是:当已知一条直线的两个端点坐标时，确定二维点阵图形中位于或最靠近这条直线的像素点的位置，即理论直线上的点所接近的像素点的坐标值。这一过程也称之为直线光栅化。

2.2.1 增量DDA算法

2.2.1.1 增量DDA算法思路

设直线的起点坐标为(x_1, y_1), 终点坐标为(x_2, y_2), 则直线的方程为

$$y = mx + b \tag{2.2.1}$$

其中直线的斜率为

$$m = (y_2 - y_1)/(x_2 - x_1) \tag{2.2.2}$$

在$y-$轴上的截距为

$$b = (y_2x_1 - y_1x_2)/(x_2 - x_1) \tag{2.2.3}$$

那么画直线的最直观算法是: 给定直线的两个端点坐标后，求得m和b;然后在$x_1 \le x \le x_2$范围内对x均匀取整数，利用式(2.2.1)进行浮点乘法和加法运算，求得y值后再取整数值即可得到需要的直线上的像素点.

这一方法通过$x-$坐标求得$y-$坐标, 因此很容易实现对$x-$坐标轴求得的像素点分布是均匀的, 但对$y-$轴, 对直线段方向却未必. 不妨认为相邻像素点之间$x-$坐标轴方向的间距是1, 则不难求得相邻像素点之间在$y-$轴和直线段方向的间距分别是$|m|$和$\sqrt{m^2+1}$. 当$|m| > 1$比较大时, 可以看出表示直线段的相邻两个像素点相距甚远.对于几乎是垂直的直线段, 由于此时x_1与x_2的差别仅1或2的, 并且$|m|$也非常非常大, 用这种算法给出的直线只是用两个端点表示的, 如此生成的直线段图形显然难以令人接收.

最简单的改进算法是: 给定直线的两个端点坐标后，求得m和b; 当$|m| \le 1$时，在$x_1 \le x \le x_2$范围内x取整数，利用式(2.2.1)进行乘法和浮点加法运算，求得y值后再取整数值;当$|m| > 1$时，则y先取整数，利用式(2.2.1)进行浮点乘法和加法运算，求得x值后再取整数值。

这时不难看出, 当$|m| \le 1$时，$x-$坐标轴方向的间距是1, 则$y-$轴, 直线段方向的间距分别是$|m| \le 1$, $\sqrt{m^2+1} \le \sqrt{2}$. 当$|m| > 1$时，取$y-$坐标轴方向的间距是1, 则$x-$轴, 直线段方向的间距分别是$1/|m| < 1$, $\sqrt{(1/m)^2+1} < \sqrt{2}$.

这种方法计算方法的缺点是计算量大。考虑到在计算机上实时编辑修改图形时, 连续不断的需要在显示器上显示出大量的直线段, 画线的速度就会非常地慢。因此需要对上述结果详细分析已给出较快的算法。可以认为直线图形上的点是由有先后顺序的一列像素点构成的，相邻的两点应满足:

$$\frac{y_{i+1} - y_i}{x_{i+1} - x_i} = m \tag{2.2.4}$$

于是有

$$y_{i+1} = y_i + m(x_{i+1} - x_i) \tag{2.2.5}$$

其中(x_i, y_i)是第i步求得的像素点坐标，(x_{i+1}, y_{i+1})是第$i+1$步求得的像素点坐标. 类似前面的分析，我们应要求

$$\begin{cases} |x_{i+1} - x_i| \le 1 \\ |y_{i+1} - y_i| \le 1 \end{cases} \tag{2.2.6}$$

并且要求其较大者就是1.也就是说，如果$|m| \leq 1$，则要求

$$\begin{cases} |x_{i+1} - x_i| = 1 \\ |y_{i+1} - y_i| \leq 1; \end{cases} \tag{2.2.7}$$

如果$|m| > 1$，则要求

$$\begin{cases} |x_{i+1} - x_i| < 1 \\ |y_{i+1} - y_i| = 1. \end{cases} \tag{2.2.8}$$

事实上，式(2.2.5)表示所求直线上y值的逐步递推关系，此式称为数字微分分析器(DDA)。于是，画直线的DDA算法可分两种情况描述为为:

(a). $|m| \leq 1$

在$x_2 - x_1 \geq 0$时,

$$x_{i+1} = x_i + 1, y_{i+1} = y_i + m \tag{2.2.9}$$

在$x_2 - x_1 \leq 0$时,

$$x_{i+1} = x_i - 1, y_{i+1} = y_i - m \tag{2.2.10}$$

(b). 若$|m| \geq 1$

在$y_2 - y_1 \geq 0$时,

$$y_{i+1} = y_i + 1, x_{i+1} = x_i + 1/m \tag{2.2.11}$$

在$y_2 - y_1 \leq 0$时,

$$y_{i+1} = y_i - 1, x_{i+1} = x_i - 1/m \tag{2.2.12}$$

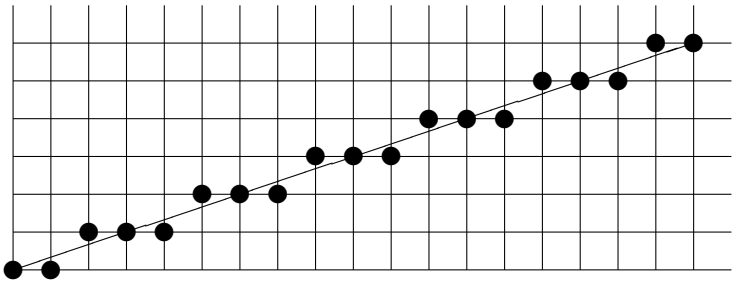
2.2.1.2 线段DDA算法伪代码描述

下面用伪代码给出DDA算法。

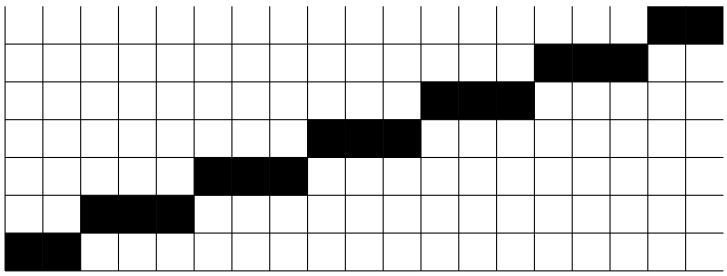
```
Procedure DDA-line(x1, y1, x2, y2)
BEGIN
  /求线段在两坐标轴方向改变量的较大者/
  IF abs(x2-x1)>=abs(y2-y1) THEN
    length=abs(x2-x1)
  ELSE
    length=abs(y2-y1)
  ENDIF
  /定义dx或dy中的较大值为1/
  dx=(x2-x1)/length
  dy=(y2-y1)/length
  x=x1+0.5*Sign(dx)
  y=y1+0.5*Sign(dy)
  i=1
  WHILE (i<length)
    PLOT(Integer(x), Integer(y))
    x=x+dx
    y=y+dy
    i=i+l
  END WHILE
END
```

其中函数Sign()是符号函数，其表达式为:

$$Sign(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases} \tag{2.2.13}$$



(a).实际要求的直线及其近似点



(b).离散化后用像素点表示的的直线

图2.1 DDA算法表示的直线

在此DDA算法中，求Δx或Δy要做一次除法，$x+\Delta x$或$y+\Delta y$又是浮点数的加法，对每个输出像素点坐标还要使用求整函数，因此计算量仍比较大。本算法的另一个缺点是有方向性，也即从(0, 0)到(-8, -4)画线与从(-8, -4)到(0, 0)逆方向所画的线略有不同，此时，当用背景色重画以擦去原直线时，可能会留下一些像素点, 使图形画面显得杂乱。下一节介绍一个新的算法。

2.2.2 Bresenham直线算法

2.2.2.1 Bresenham直线算法思路

Bresenham直线算法最初是为数字绘图仪而设计的。它的目标是选择表示直线的最佳像素点位置. 为此，算法根据直线的斜率确定在x或y方向上每次递增一个单位，而另一方向上根据理论直线段与最近像素点的距离每次的增量为0或1。我们首先讨论直线斜率$m \in [0,1]$且$x_2 > x_1$时的整数Bresenham算法，然后再推广到画任意线段的算法。

当直线斜率$m \in [0,1]$，且$x_2 > x_1$时，根据式(2.2.9)有:

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + m \end{cases}$$

又由于显示直线的像素点只能取整数值坐标，可以假设直线上第i个像素点坐标为$(\overline{x_i}, \overline{y_i})$，它是直线上点$(x_i, y_i)$的最近似并且$\overline{x_i} = x_i$。于是，要表达的直线上下一个$(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + m)$的最近似的像素点的可能位置是$(\overline{x_i} + 1, \overline{y_i})$或$(\overline{x_i} + 1, \overline{y_i} + 1)$。

在$x = \overline{x_i} + 1$处, 直线上点的坐标$y = m(\overline{x_i} + 1) + b$. 该点与点$(\overline{x_i} + 1, \overline{y_i})$和$(\overline{x_i} + 1, \overline{y_i} + 1)$的距离分别是$d_1$和$d_2$:

$$\begin{cases} d_1 = y - \overline{y_i} = m(\overline{x_i} + 1) + b - \overline{y_i} \\ d_2 = (\overline{y_i} + 1) - y = (\overline{y_i} + 1) - m(\overline{x_i} + 1) - b \end{cases} \tag{2.2.16}$$

这两个距离的差为:

$$d_1 - d_2 = 2m(\overline{x_i} + 1) - 2\overline{y_i} + 2b - 1 \tag{2.2.17}$$

这个差有如下几何意义:

(1)当此值为正时，真正的直线上点离像素点$(\overline{x_i} + 1, \overline{y_i} + 1)$较近，说明下一个直线上的像素点应取$(\overline{x_i} + 1, \overline{y_i} + 1)$。

(2)当此值为负时，真正的直线上离像素点$(\overline{x_i} + 1, \overline{y_i})$较近，说明下一个直线像素点应取$(\overline{x_i} + 1, \overline{y_i})$。

(3)当此值为零时，真正的直线上离上、下两个像素点的距离相等，我们规定取$(\overline{x_i} + 1, \overline{y_i})$作为下一个直线像素点。

因此，只要利用$(d_1 - d_2)$的符号就可以决定下一个像素点的选择。如果我们定义一个新的判别式:

$$p_i = \Delta x * (d_1 - d_2) = 2\Delta y \cdot \overline{x_i} - 2\Delta x \cdot \overline{y_i} + c \tag{2.2.18}$$

因此式中的$\Delta x = (x_2 - x_1) > 0$，$p_i$与$(d_1 - d_2)$有相同符号; $\Delta y = y_2 - y_1$;常数$c = 2\Delta y + \Delta x(2b - 1)$。$p_i$的一个优点是省去了$(d_1 - d_2)$中为了计算$m$所需要的除法运算。我们知道除法运算用硬件实现是比较复杂的。

现在我们要进一步化简上述误差判别式以得到递推公式，消除常数c. 以$i + 1$代换此式中的i，得到:

$$p_{i+1} = 2\Delta y \cdot \overline{x_{i+1}} - 2\Delta x \cdot \overline{y_{i+1}} + c \tag{2.2.19}$$

与前式相减, 并利用$\overline{x_{i+1}} = \overline{x_i} + 1$，可得，

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x \cdot (\overline{y_{i+1}} - \overline{y_i}) \tag{2.2.20}$$

再假设直线的初始端点恰好是其像素点的坐标，即满足:

$$\overline{y_1} = m\overline{x_1} + b \tag{2.2.21}$$

于是可得p_i的初值

$$p_1 = 2\Delta y - \Delta x \tag{2.2.22}$$

这样，利用误差判别变量，并注意到每一步x的增量为$x_{i+1} - x_i = 1$就可得到如下算法表示:

(1).

$$\begin{cases} p_1 = 2\Delta y - \Delta x \\ x_{i+1} = x_i + 1 \end{cases} \tag{2.2.23}$$

(2).如果$p_i \geq 0$,

$$\begin{cases} y_{i+1} = y_i + 1 \\ p_{i+1} = p_i + 2(\Delta y - \Delta x) \end{cases} \tag{2.2.24}$$

(3).如果$p_i < 0$,

$$\begin{cases} y_{i+1} = y_i \\ p_{i+1} = p_i + 2\Delta y \end{cases} \tag{2.2.25}$$

从式(2.2.20)可以看出，第$i+1$步的判别变量p_{i+1}仅与第i步的判别变量p_i、直线的两个端点坐标分增量Δx和Δy有关，计算也很简单，只用整数相加和乘2运算，没有四舍五入处理，而乘2可利用左移一位来实现，因此这个算法速度快并易于硬件实现。

该算法的主要步骤如下:

(1)输入线段的两个端点分别存于(x_s, y_s)和(x_e, y_e)中，

(2)将第一点作为起始点, 即有$(x_1, y_1) = (x_s, y_s)$，

(3)分别计算Δx、Δy及p_1，若$p_1 < 0$，下一点为$(x_1 + 1, y_1)$，否则，取$(x_1 + 1, y_1 + 1)$，

(4)以单位步长增加x坐标，按式((2.2.24))或((2.2.25))计算p_i。若$p_i < 0$，下一点的y坐标不变，否则y坐标加1。

(5)重复步骤(4)直到x逐步增加到x_e为此。

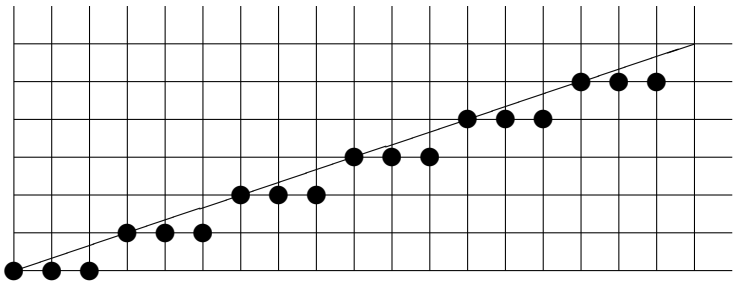
在前面的处理中, 我们假设$m \in [0, 1]$。这一假设的几何意义是要绘的直线段与$x-$轴的夹角不超过与$y-$轴的夹角。或者更直观地, 直线段的方向更靠近$x-$轴方向，如果有$m > 1$, 则说明直线段的方向更靠近$y-$轴方向。如果我们互换两个坐标轴, 则直线段就满足前面的假设条件. 因此, 只要将算法中的x和y对换，则上述两个公式依然有效。如果$m < 0$,则相应的只是x或y的方向的改变后, 上述两个公式依然有效。这时实际上只是改变Δx或Δy的符号即可.

2.2.2.2 Bresenham直线整数伪代码描述

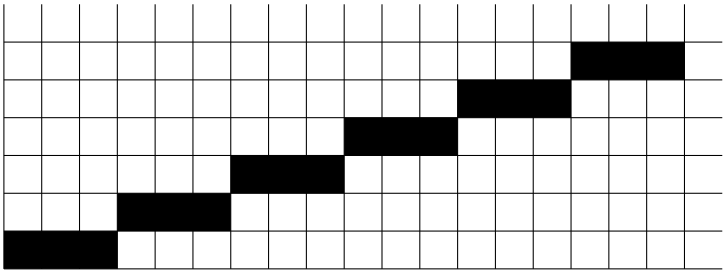
下面给出的是完整的整数坐标的直线Bresenham算法。

```
Procedure Bresenham-line(xs, ys, xe, ye)
BEGIN
  dx=ABS(xe-xs);dy=ABS(ye-ys);x=xs;y=ys
  s1=SIGN(xe-xs), s2=SIGN(ye-ys)
  If dy>dx THEN
     temp=dx;dx=dy:dy=temp
     interchange=1
  ELSE
     interchange=0
  ENDIF
  p=2*dy-dx
  FOR i=1 TO dx
    PLOT(x, y)
    IF p>=0  THEN
      IF interchange=1
        x=x+s1
      ELSE
        y=y+s2
      ENDIF
        p=p-2*dx
    ENDIF
    IF interchange=1 THEN
      y=y+s2
    ELSE
      x=x+s1
```

```
    ENDIF
      p=p+2*dy
  NEXT i
END
```



(a).实际要求的直线及其近似点



(b).离散化后用像素点表示的的直线

图2.2 Bresenham 算法表示的直线

2.3 圆的点阵图形扫描转换算法

2.3.1 一般方法

2.3.1.1.直角坐标方法

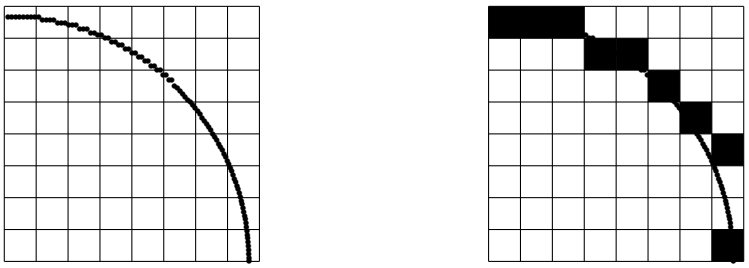
圆也是基本图形之一，为了简单起见，假设圆的圆心在坐标原点，半径为R.于是可得其方程为:

$$x^2 + y^2 = R^2 \tag{2.3.1}$$

解出y, 得到:

$$y = \pm(R^2 - x^2) \tag{2.3.2}$$

我们可先考虑第一象限的四分之一圆弧。让x从0到R以单位步长增加, 在每一步可解出y, 然后调用画点程序即可。但这样做，由于有乘方和平方根运算，并且都是浮点运算，算法效率不高。而且当x接近R值时，由于圆的斜率趋于无穷大导致$y-$方向变化远大于$x-$方向的变化，使得圆周上有较大的间隙，画出的圆如图2.3所示。



(a).圆弧曲线 (b).圆弧的离散表示

图2.3 四分之一圆弧的离散表示

一般地,整圆的算法都可作下述简化: 我们知道圆有对称性,如果点(x, y)在圆周上，那么就可以很容易地计算出圆周上的另外7个点(y, x), $(-x, y)$, $(-y, x)$, $(-x, -y)$, $(-y, -x)$, $(x, -y)$, $(-y, x)$。于是,x的取值范围可限定为$[0, \frac{R}{\sqrt{2}}]$,即只需要计算圆的一段八分圆弧就够了，从而显著地减少计算量，且此时随着x的增加,y是单调减小的.

2.3.1.2. 折线逼近法

圆可以用其内接正多边形来逼近。更广泛地说，曲线也可用折线来通近，每段折线越短曲线逼近越好，但执行时间也越长。我们从折线逼近一段圆弧出发来讨论这种逼近的基本思想。假设圆弧的起始和终止角分别为t_s和t_e,半径为R，用折线代替圆弧容许的最大误差为ϵ，

如果用n段等长折线来逼近圆弧，则每段圆弧曲线对应的圆弧角度为$\delta = (t_e - t_s)/n$。当δ充分小时，圆弧和对应弦的最大误差是:

$$e = R[1 - cos(\delta/2)] = 2Rsin^2(\delta/4) \approx (1/8)R\delta^2 \tag{2.3.3}$$

为了使$e \leq \epsilon$,把δ代人上式后得到:

$$n \geq \frac{t_e - t_s}{2} \cdot \sqrt{\frac{R}{2\epsilon}} \tag{2.3.4}$$

这个式子表明半径越大，要求的误差越小，则所用的折线段数也越多。

2.3.1.3. 实例分析

假设要画一个半圆弧，其半径$R = 256$个像素点单位，于是圆弧的角度为$t_e - t_s = \pi$。若要求误差不超过两个像素点单位,即$\epsilon \leq 2$,则所需要的折线段n为:

$$n \geq \frac{\pi}{2}\sqrt{\frac{256}{4}} \approx 13 \tag{2.3.5}$$

即用13段折线来遇近这个半圆弧。

那么如何求折线段的端点位置呢?我们知道圆的参数方程为

$$\begin{cases} x = R\cos t \\ y = R\sin t \qquad t \in [0, 2\pi] \end{cases} \tag{2.3.6}$$

对圆弧来说，参数$t \in [t_s, t_e]$，而每一折线端点处的参数为

$$t_i = t_s + i\delta \qquad i = 0, 1, 2, \cdots, n \tag{2.3.7}$$

因此,各折线段的端点坐标为

$$\begin{cases} x = R\cos t_i \\ y = R\sin t_i \qquad t \in [0, 2\pi] \end{cases} \tag{2.3.8}$$

显然直接利用公式(2.3.8)求出折线的端点的计算是很费时的，为此，我们根据相邻折线段的端点间的递推关系来递推地计算折线段的端点:

$$\begin{cases} x_{i+1} = R\cos(t_i + \delta) = x_i\cos\delta - y_i\sin\delta \\ y_{i+1} = R\sin(t_i + \delta) = x_i\sin\delta + y_i\cos\delta \end{cases} \tag{2.3.9}$$

在这个公式中, 三角函数值$\cos\delta$,$\sin\delta$是常量. 于是,我们就可以从$x_0 = R\cos t_s$和$y_0 = R\sin t_s$ 开始进行逐点递推计算, 并且在递推计算的过程中每计算一个端点，只需要四次乘法和两次加法,因而计算速度比较快,但缺点是计算过程中的前面的舍入误差却随着递推的过程逐步向后传递,使得越靠后的点,误差会越大。

2.3.2 Bresenham圆弧算法

2.3.2.1 Bresenham圆弧算法思路

Bresenham提出了一种比上面提到的绘圆方法都更为有效的圆弧增量算法。与Bresenham 直线算法一样，其基本的方法是利用判别变量来判断选择最近的像素点，判别变量的数值仅仅用一些加、减和移位运算就可以计算出来。为了简便起见，考虑一个圆心在坐标原点的圆，而且只计算八分圆周上的点，其余圆周上的点利用对称性就可得到。算法沿x轴方向取单位步长，从$x = 0$开始到$x = R/\sqrt{2}$结束。

假定像素点(x_{i-1}, y_{i-1})是在$x = x_{i-1}$时最靠近圆周的像素点，那么在$x = x_{i-1}+1$时要得到的最靠近圆周的像素点只可能是: $H(x_{i-1}+1, y_{i-1})$或$L(x_{i-1}+1, y_{i-1}-1)$, 因为这一段圆弧曲线随x的增加,y单调减小,且x方向改变量大于y方向的改变量。由于x方向改变量为1, y方向的改变量只能在0和1之间, 因此取整后y坐标的改变两只能是0或1。设$x = x_{i-1} + 1$时圆周上点的y坐标为y,则必有下述三种关系之一成立: $y < y_i - 1$,或$y_i - 1 \le y \le y_i$, 或$y > y_i$. 我们下面分别加以讨论.

(1). $y_i - 1 \le y \le y_i$

此时圆弧曲线从H和L两点之间穿过, 定义

$$\begin{cases} d_H = y_{i-1}^2 - y^2 \ge 0 \\ d_L = y^2 - (y_{i-1} - 1)^2 \ge 0 \end{cases} \tag{2.3.10}$$

为$x = x_{i-1}+1$时圆周上点的y坐标为分别与H和L点的y坐标平方之差,注意$R^2 = (x_{i-1}+1)^2 + y^2$,则有

$$\begin{cases} d_H = y_{i-1}^2 - R^2 + (x_{i-1}+1)^2 \\ d_L = R^2 - (x_{i-1}+1)^2 - (y_{i-1}-1)^2 \end{cases} \tag{2.3.11}$$

由此也可以看出这两个量的几何意义是从圆心(原点)到H(或D)距离的平方与到圆周距离的平方之差。如果$d_H \geq d_L$，则L比H更接近于实际的圆。反之，$d_H < d_L$，则H比L更接近于实际的圆。故我们把判别变量定义为d_H和d_L的差，用p_i表示:

$$p_i = d_H - d_L = 2(x_{i-1}+1)^2 + (y_{i-1}-1)^2 + y_i^2 - 2R^2 \tag{2.3.12}$$

并且在当$p_i < 0$时选择的像素点为$H(x_i, y_{i-1})$; 否则选择的像素点为$L(x_i, y_i)$。

(2). $y < y_i - 1$

此时圆弧曲线从H和L两点下方通过，显然的选择是H和L中y坐标为$y_{i-1}-1$的点, 即L。此时式(2.3.11)中定义的$d_H \geq 0$，$d_L < 0$，故同时有有$p_i > 0$.

(3). $y > y_i$

此时圆弧曲线从H和L两点上方通过，显然的选择是H和L中y坐标为y_{i-1}的点, 即H。此时式(2.3.11)中定义的$d_H \leq 0$，$d_L > 0$，故同时有$p_i < 0$.

总之, 无论哪一种情况成立,我们的判别规则都是一致的,即当$p_i < 0$时选择的像素点为$H(x_i, y_{i-1})$; 否则选择的像素点为$L(x_i, y_i)$。

要决定下一个像素点，就要求出下一步的判别式p_{i+1}。用i+1代换(2.3.12)中i可得：

$$p_{i+1} = 2(x_i+1)^2 + (y_i-1)^2 + (y_i)^2 - 2R^2 \tag{2.3.13}$$

这一计算公式需要作数次乘法运算，结合(2.3.12)，可把上式重写为：

$$\begin{aligned} p_{i+1} &= p_i + 4x_i + 6 + 2({y_i}^2 - y_{i-1}^2) - 2(y_i - y_{i-1}) \\ &= \begin{cases} p_i + 4x_i + 6, & p_i < 0 \\ p_i + 4x_i + 10, & p_i \geq 0 \end{cases} \end{aligned} \tag{2.3.14}$$

算法每次从p_1开始，以后则按照(2.3.14)式计算下一个判别变量。把点$(x_1, y_1) = (0, R)$代人式(2.3.12)，可得到

$$p_1 = 3 - 2R \tag{2.3.15}$$

因此，算法可表示为：

(a).

$$\begin{cases} p_1 = -3 - 2R \\ x_i = x_{i-1} + 1 \end{cases} \tag{2.3.16}$$

(b). 如果$p_i \geq 0$, 则有

$$\begin{cases} y_i = y_{i-1} - 1 \\ p_{i+1} = p_i + 4(x_{i-1} - y_{i-1}) + 10 \end{cases} \tag{2.3.17}$$

(c). 如果$p_i < 0$, 则有

$$\begin{cases} y_i = y_{i-1} \\ p_{i+1} = p_i + 4x_{i-1} + 6 \end{cases} \tag{2.3.18}$$

2.3.2.2 Bresenharn画圆算法的伪代码描述

下面我们假设圆心在坐标原点，给出完整的Bresenham画圆算法。

```
Procedure Bresenham-circle(radius)
Begin
/ radius 为圆的半径，圆心在坐标原点 /
  y=radius
  p=3-2*radius
  WHILE x<y DO
  Begin
    PLOT(x,y)
    IF p<0 THEN
      p=p+4*x+6
    ELSE
      p=p+4*(x-y)+10
      y=y-1
    ENDIF
   END
END
```

图2.4中利用Bresenham 画圆算法只画出了四分之一的圆，其余的部分可用对称关系求得。

(a).圆弧曲线　　(b). 圆弧的离散表示

图2.4 Bresenham 画圆算法四分之一圆弧的离散表示

2.4 椭圆点阵图形扫描转换算法

对于一个标准椭圆，

$$\frac{x^2}{a^2}+\frac{b^2}{b^2}=1 \tag{2.4.1}$$

我们可以用如下隐函数形式的方程表示：

$$F(x,y)=b^2x^2+a^2y^2-a^2b^2=0 \tag{2.4.2}$$

2.4.1 椭圆弧算法思路

下面讨论如何用增量算法实现椭圆图形的点阵扫描转换。

不同于圆,椭圆只具有四分对称性，我们需要转换位于第一象限的四分之一椭圆, 其它部分可利用四分对称性获得.在第一象限的四分之一椭圆弧当x从0变到a时, 椭圆弧的变化率是变化的，从0到-1，从-1到$-\infty$，因此我们应该基于变化率为-1的点将第一象限的四分之一椭圆弧分成两部分, 然后确定边界上的像素点。

第一象限内变化率为−1的点应满足

$$-\frac{\frac{\partial F(x,y)}{\partial x}}{\frac{\partial F(x,y)}{\partial y}} = -1 \tag{2.4.3}$$

即有

$$\frac{a^2 y}{b^2 x} = 1 \tag{2.4.4}$$

结合椭圆方程,不难求得此点的$x-$坐标是:

$$x_c = \frac{a^2}{\sqrt{a^2+b^2}} = \frac{a}{\sqrt{a^2+b^2}}\ a \tag{2.4.5}$$

于是当$0 \le x \le x_c$时，椭圆弧曲线上$x-$方向的变化大于$y-$方向的变化，令$\Delta x = 1$，求解Δy；当$x_c \le x \le a$时，则令$\Delta y = -1$，求解Δx。

在$0 \le x \le x_c$时，如果当前像素点是(x_i, y_i)，下一步像素点(x_{i+1}, y_{i+1})的选择方法为: $x_{i+1} = x_i + 1$，因此$x-$方向的变化为1, 由于y在减小，且改变量不超过x的改变量, 因此取整后的y的改变量只能是0或1, 下一个要选择的像素点只能是是$H(x_i+1, y_i)$或$L(x_i+1, y_i-1)$。我们可以根据H 和L之间中点的函数值进行判别，如$F(x_i+1, y_i-1/2) < 0$，表明中点位于椭圆内部,则像素点H靠近椭圆，应该选择像素点H；反之则应该选择像素点L。基于中点判别规则，我们也可以比较容易地区分$x \le x_c$成立否: 如果$a^2(y_i - 1/2) > b^2(x_i+1)$即$-\dfrac{a^2(y_i-1/2)}{b^2(x_i+1)} < -1$，则$x > x_c$。

下面我们分两种情况来推导中点判别函数的增量公式。首先考虑$0 \le x \le x_c$时的椭圆弧曲线.

给定当前点(x_i, y_i)后,相应的判定函数是:

$$p_i = F(x_i+1, y_i-1/2) = b^2(x_i+1)^2 + a^2(y_i-1/2)^2 - a^2b^2 \tag{2.4.6}$$

如果$p_i \le 0$, 则我们应选择的像素点是H，新的判别函数是：

$$p_{i+1} = F(x_i+2, y_i-1/2) = b^2(x_i+2)^2 + a^2(y_i-1/2)^2 - a^2b^2 \tag{2.4.7}$$

其递推计算公式为

$$p_{i+1} = p_i + b^2(2x_i+3). \tag{2.4.8}$$

否则,有$p_i > 0$,应选择的是像素点L，相应的有:

$$p_{i+1} = F(x_i+2, y_i-3/2) = b^2(x_i+2)^2 + a^2(y_i-3/2)^2 - a^2b^2 \tag{2.4.9}$$

其递推计算公式为

$$p_{i+1} = p_i + b^2(2x_i+3) + a^2(-2y_i+2). \tag{2.4.10}$$

当$x_c < x \le a$时，我们可以类似地得到

$$\begin{cases} p_{i+1} = p_i + a^2(-2y_i+2), & p_i \le 0 \\ p_{i+1} = p_i + b^2(2x_i+3) + a^2(-2y_i+2), & p_i > 0. \end{cases} \tag{2.4.11}$$

下面分别确定p_i的初始值.如果椭圆方程的a和b都是整数，显然填充的第一个像素应该是$(x_1, y_1) = (0, b)$，所以

$$p_1 = F(1, b-1/2) = b^2 + a^2(b-1/2)^2 - a^2b^2 = b^2 - a^2(b+1/4) \tag{2.4.12}$$

如果$x_i > x_c$,则(x_{i+1}, y_{i+1})只能从两点$(x_i, y_i - 1)$，$(x_i + 1, y_i - 1)$中选择一个，并且判别时的初始值应重新设置为:

$$p_i = F(x_i + 1/2, y_i - 1) = b^2(x_i + 1/2)^2 + a^2(y_i - 1)^2 - a^2b^2 \qquad (2.4.13)$$

扫描转换算法的终止条件应该是$y = 0$。

2.4.2 椭圆算法的伪代码描述

基于上述规则，我们可以得到如下所示的椭圆扫描转换算法。

```
Procedure  Ellips(a,b)
Begin
  x=0:y=b;
   a=a*a:b=b*b;
   p1=b-a+a/4;
   Plot(x,y);
   x=x+1
   while(a*y-a/2>b*x) do
   Begin
    if p1<0
      p1=p1+2*b*x+b;
    Else
      y=y-1;
      p1=p1+2*b*x+b-2*a*y;
    Endif
    x=x+1;
    Plot(x,y);
   End
   p1=b*(x+1/2)*(x+1/2)+a*(y-1)*(y-1)-a*b;
   While(y>0) do
   Begin
     y=y-1;
     if p1<0
       x=x+1;
       p1=p1+2*b*x-2*a*y+a;
     Else
       p1=p1-2*y*a+a;
     Endif
     Plot(x,y);
   End
End
```

由于椭圆的两个轴长参数a、b在算法中一直以其平方的形式在循环程序段反复出现，我们在程序的开始部分把它们直接调整为其平方的数值结果以减少重复计算。

习题2

1. 给定直线从端点$(0, 0)$到$(4, 20)$。分别求出当x从0到4和y从0到20两种情况下该直线段的点阵表示的图形。

2. 给定直线从端点$(0, 0)$到$(20, 4)$。分别求出当x从0到20和y从0到4两种情况下该直线段的点阵表示的图形。

3. 求出前两习题中直线段用DDA算法产生的点阵表示的图形。

4. 给定直线从端点$(0,0)$到$(20,4)$。分别求出用DDA算法和Bresenham直线算法产生的该直线段点阵表示的图形。直两个图形是否完全一致？

5. 在推导Bresenham直线算法的工程中，我们用d_1和d_2来衡量两个象素点(x_i+1,y_i)和(x_i+1,y_i+1)与给定直线的远近程度。这实际上是两点在y-坐标轴方向与给定直线的远近程度。d_1和d_2的大小关系与这两点到给定直线距离的大小关系是否一致？

6. 给出下列伪代码程序出生的点阵图形.

```
Procedure  Ex(r)
Begin
  x=0:y=r:pl=3-2*r;
   Plot(x,y);
   while(y>x) do
   Begin
    if p1<0
      p1=p1+4*x+6;
    Else
      y=y-1;
      p1=p1+4*(x-y)+6;
    Endif
    x=x+1;
    Plot(x,y);
   End
End
```

7. 在推导Bresenham圆弧算法的工程中，我们用d_H和d_L来衡量两个象素点(x_i+1,y_i)和(x_i+1,y_i-1)与给定圆弧的远近程度。这实际上是两点在y-坐标轴方向与给定圆弧的远近程度。d_H和d_L的大小关系与这两点到给定圆弧距离的大小关系是否一致？

8. 在用折线段逼近法划圆时,如果δ太小，而圆弧上的点通过递推公式计算，会出现什么结果？如果δ太大呢？

9. 写出圆心在原点的整圆的Bresenham圆弧算法。

10. 写出圆心在任一点的Bresenham圆弧算法。

11. 写出边与坐标轴平行的圆角矩形的Bresenham算法。

Chapter 3

区域填充

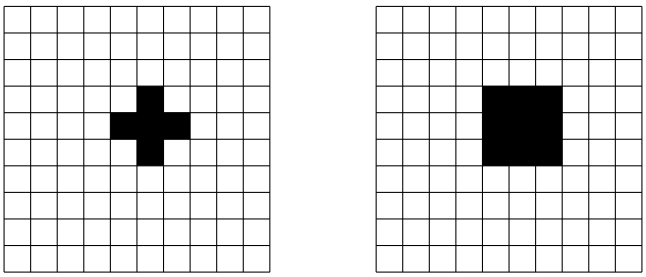
区域填充的目的是要对指定区域内的所有像素点涂上指定的图案或颜色. 这种对图形的处理方法能充分体现点阵图形显示系统的优点之一,就是能够很方便地存储和显示由各种颜色或者浓淡图案组成的区域，填充区域的图案也和颜色、亮度值一样被存放到帧缓冲区中。而在向量图形显示系统中要显示具有浓淡的区域是很困难的，因为区域填充要求在每一刷新周期内在区域内部画出所有线段。而且这样的转换工作的复杂程度随着图形本身的复杂程度的提高而迅速增加.

3.1 区域的定义和类型

3.1.1 区域的连通方式

一个区域是一组相互连通的像素点集合。而像素点之间的连通方式可分为如下两类：

(1)四连通：两个像素点是上下或左右相连的,则称其为四连通的. 一个像素点可通过四连通方式访问的像素点为上、下、左和右四个相邻像素点之一，即一个像素点最多与上、下、左和右四个相邻像素点具有连通关系。



(a). 四连通的像素点　　(b). 八连通的像素点

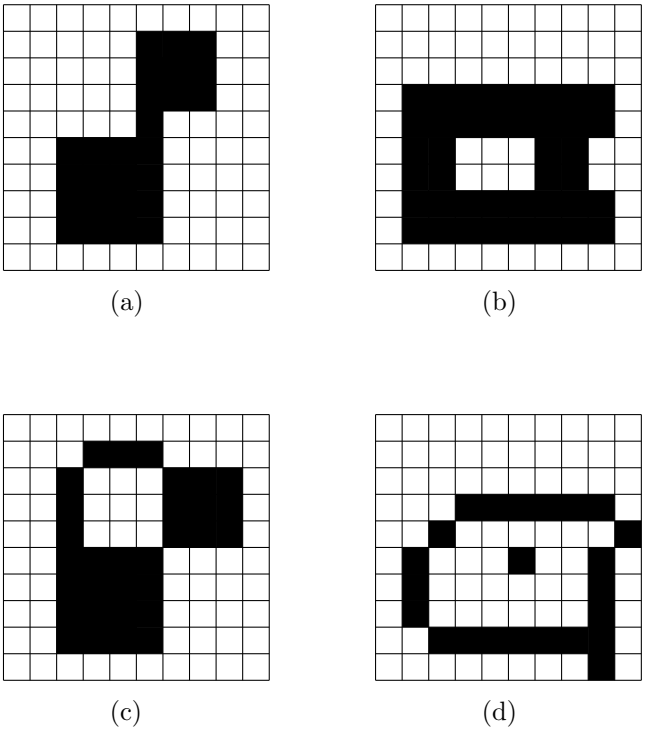
图3.1 一个像素点及与其连通的像素点

(2)八连通：两个像素点是上下或左右相连的,或者对角线方向相连的,则称其

为八连通的. 一个像素点除可访问其上、下、左和右四个近邻像素点之外，还可访问两条对角线方向上四个近邻像素点。即一个像素点最多与上、下、左和右四个及四个对角点共八个相邻像素点具有连通关系。

如果认为像素点是一个个的小方块的话, 则具有公共边的像素点具有四连通关系; 则具有公共边或公共角点的像素点具有八连通关系.

根据像素点之间的连通方式可把区域分为四连通和八连通区域两种.



(a) (b)

(c) (d)

图3.2 区域的定义和分类

3.1.2 区域的定义方式

根据区域的定义和生成方式有以下两类区域：

(1)内部定义区域(interior-defined)：区域内所有的像素点具有单一的值，如图3.2 (a)和(b)所示。

(2)边界定义区域(boundary-defined)：由一个边界来定义的区域，边界部分的像素点具有单一像素点的值，边界包围的部分就是区域本身。如图3.2(c)和(d)所示。通常区域本身部分的像素点具有不同于边界上的像素点的值。

图3.2(a), (b)是四连通区域。图3.2(c)的边界本身是八连通区域。四连通区域是八连通区域的一种特殊情况，在边界定义的区域中如果边界本身作为一个区域和它定义的区域都是连通的，则四连通区域的边界必定是八连通区域，而八连通区域的边界必定是四连通区域。但若有两者之一不连通, 另一个仍是可以连通的,图3.2(d)的边界本身既不是四连通也不是八连通区域, 其内部区域却是四连通的。

3.2 注入填充区域算法

注入填充算法(Flood Fill Algorithm)用于内部定义区域，以改变整个区域的颜色属性，它把区域内的原像素点值改变成另一种像素点值。算法3.2用于填充八连通的内部定义区域。算法中，$read-pixel(x,y)$表示读出像素点(x,y)像素点值。old-value为像素点的原值, new-value为将要填充的新值。

[算法3.2] 注入填充区域算法。

```
Procedure flood-fill-8(x,y,old-value,new-value)
BEGIN
IF  read-pixel(x,y)=old-value  THEN
 BEGIN
   write-pixel(x,y,new-value)
   flood-fill-8(x,y-1,old-value,new-value)
   flood-fill-8(x,y+1,old-value,new-value)
   flood-fill-8(x-1,y,old-value,new-value)
   flood-fill-8(x+1,y,old-value,new-value)

   flood-fill-8(x+1,y-1,old-value,new-value)
   flood-fill-8(x+1,y+1,old-value,new-value)
   flood-fill-8(x-1,y-1,old-value,new-value)
   flood-fill-8(x-1,y+1,old-value,new-value)
  END
 ENDIF
END
```

此算法所采用的基本方法是首先确定(x,y)点的像素点是否在区域内尚未被访问过的那一部分之中，也就是说，如果这个像素点的值是原始值old-value，则需要把它改为填充的值new-value，然后按八连通区域性质先后访问其八个相邻的像素点，当访问其中每一个近邻像素点时，都要进行递归调用。此算法通过在四个方向而不是八个方向上扩展，就可以用来填充一个内部定义的四连通式区域。这时程序只要有前面四个flood-fill-8(...)语句就可以了.

3.3 边界填充算法

边界填充算法(Boundary Fill A1gorithms)把边界包围的区域内所有像素点的值改为所需要的值，设边界上的像素点值为boundary-value，算法3.3用于填充八连通的边界定义区域。

[算法3.3] 边界区域填充算法。

```
Procedure  boundary-fill-8(x,y,boundary-value,new-value)
BEGIN
/ boundary-value为边界像素点的值,new-value为将要填充的新值 /
 IF （read-pixel(x,y)<>boundary-value）
    .AND.（read-pixel(x,y)<>new-value） Then
   BEGIN
     write-p1xel(x,y,new-value)
     boundary-fill-8(x,y-1,boundary-value,new-value)
     boundary-fill-8(x,y+1,boundary-value,new-value)
     boundary-fill-8(x-1,y,boundary-value,new-value)
```

```
      boundary-fill-8(x+1,y,boundary-value,new-value)

      boundary-fill-8(x+1,y-1,boundary-value,new-value)
      boundary-fill-8(x+1,y+1,boundary-value,new-value)
      boundary-fill-8(x-1,y-1,boundary-value,new-value)
      boundary-fill-8(x-1,y+1,boundary-value,new-value)
    END
   ENDIF
 END
```

该算法的基本思想显然同前一算法是一样的，只是在测试(x, y)点的像素点是否处在尚未被访问过的那一部分区域之内时，需要两个条件的判别。首先，与边界值相比较，如果该像素点的值不是边界像素点值，即该像素点不属于边界，而在填充区域内；其次，与新值相比较，以探测该像素点是否已被访问过。此算法也是递归调用的，访问的路径也是按八连通区域设计的。

上述两个递归算法都是深度递归的，当内存空间有限时，可能引起堆栈溢出。我们下面给出两个新的算法. 要填充的区域限于4-连通区域,按给出边界方式定义.

3.4 扫描线算法

在下面的区域填充算法中,我们总是先给定区域内一点，称为种子点。对种子点赋予指定的颜色，并将这种颜色扩充到区域内的所有点。

区域填充的扫描线算法相对于前面的递归算法是一种较好的区域填充扫描线算法。它很好地避免了递归算法由于多层递归、反复进出堆栈操作而造成的费时费内存的缺点。这一借助于堆栈技术实现的种子点入堆栈的区域填充扫描线算法可以由如下几个步骤实现。

步骤1:（初始化）设置一空的堆栈，将预先给定的种子点(x,y)压入堆栈。

步骤2:（种子点出栈）弹出栈顶元素(x,y)作为当前种子点。

步骤3:（向左填充）沿当前种子点所在的扫描线从种子点开始向左侧逐个像素点进行填色，直到区域边界，像素点颜色值置为$new-color$，填充区段左端点记为x_l。

步骤4:（向右填充）类似步骤3的操作向右填充，相应地获得一右端点值记为x_r.

步骤5:（新种子点入栈）分别确定与当前扫描线相邻的上下两条扫描线上位于给定的区段$[x_l, x_r]$ 内含于区域的区段。如果有这样的区段，则取区段的右端点为新的种子点压入堆栈（这样的区段可不止一个）。如果区段内的像素点值为$new-color$或者边界像素点颜色值为$boundary-color$，则不产生新的种子点入栈。

步骤6: (结束判断）如果种子了点堆栈非空则转步骤2;否则算法结束。

在这一算法中，由于有一事先给定的种子点，我们总是从区域内部开始的，区域由置特殊颜色值$boundary-color$的像素点组成的边界给出。当前像素点在区域内外的判断需要不断地获取有关像素点的颜色值并进行相应的逻辑判断，依此我们可以看到在步骤3和步骤4的运算过程中要一个接一个地向种子点两侧获取像素点颜色值，以判定是否达到边界从而结束区段填充。

在步骤5中，寻找新的种子点即要从x_l出发向右逐个查找是否有位于区域内的像素点，找到区域内一点时，还不能立即停止，最少要查找到x_r，以防止有多个区段时产生遗失，整个过程如图9.3所示。执行的结果是$(x_i, y-1), i = 1, 2$作为新的种子点压入堆栈，可以看到为了找到$(x_i, y-1), i = 1, 2$，

从x_l到x_r的像素点颜色值都需要取出和判断，在随后的操作中，当$(x_i, y-1)$ 作为当前种子点弹出时，这一点所在的区段需要步骤3的操作，区段内的所有像素点的颜色值需要再次被获取，因而造成重复操作。

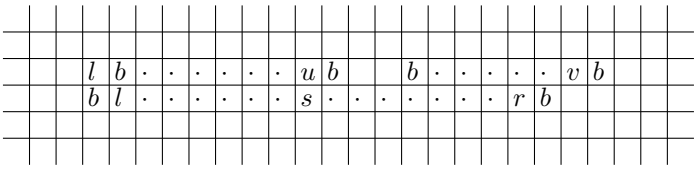


图3.3 步骤5执行过程示意图

图9.3中各标记的含义是：b标示边界像素点，$s=(x,y)$ 是当前种子点、$l=(x_l,y)$ 和$r=(x_r,y)$是当前区段的左右端点，$u=(x_1,y-1)$和$v=(x_2,y-1)$是新种子点。

3.5 压入区段端点的扫描线算法

3.5.1 算法思路

文献[13]分析上述扫描线算法, 发现它包含有大量重复获取像素点颜色的操作以及由此伴随着的逻辑判断等其它运算。可以看到种子点入堆栈除了从递归算法的种子点入堆栈直观继承过来以外，在算法中没有更多的必要性。实际上，由种子点所得到的区段$[x_l, x_r]$ 在算法中具有承前继后、使程序不间断运行下去的重要性. 如果上述算法在寻找新的种子点的同时也给区域内的像素置值new-color，则种子点作为填色的出发点也就没有用处了。而填色后自然就产生了一个区段$[x_l, x_r]$，之后需要的是在上下两条扫描线上相同的区间内寻找含于区域内的新区段，并填色。

依照这一思路，我们以标记区段左、右端点的x_l, x_r及标明区段所在扫描线行值的y 一起作为入栈元素给出一改进的算法。这一算法可以综合为由以下步骤加以实现。

步骤1:（寻找种子点所在区段）从给定的种子点(x,y)出发，沿当前扫描线向左右两个方向填充，直到边界，分别记区段左右端点的列坐标x_l, x_r.

步骤2: (初始化）设置堆栈，将由前面产生的x_l, x_r及事先给定的y值压入堆栈。

步骤3: (出栈）弹出栈顶元素x_l, x_r及y作为当前扫描线及要搜索填充的出发区间。

步骤4: (填充并寻找新区段）分别确定位于当前扫描线上下两条扫描线上区间$[x_l, x_r]$ 内含于区域内的区段并加以填充，如果有这样的区段，则区段左右端点值及所在的扫描线值入堆栈。

步骤5:（结束判断）如果堆栈非空则转步骤3, 否则算法结束。

3.5.2 算法伪代码描述

为使后面的说明清楚起见，先介绍要用到的一些操作及函数。

set-stack-empty：设置堆栈并置为空。

push：将一栈元素压人堆栈。

pop：弹出栈顶元素。

stack-not-empty：检查堆栈状态函数。当堆栈不空时返回True，否则返回False。

getpixel：返回指定点处的像素点颜色值。

setpixel：设置指定点处像素点颜色值为给定的颜色值。

有了上述准备，我们可以给出如下改进后区域填充扫描线算法的算法语言描述。

```
Procedure XXXX(x,y)  /*程序开始*/
BEGIN
set-stack-empty;
xsave:=x;
while  getplxel(x,y)<>boundary-color  do
begin /*向种子点右侧填充*/
setpixel(x,y,new-color);
x:=x+1;
end;
xr:=x-1;
/*类似地，从种子点开始向左填充，并用xl保存左端点*/
push(xl,xr,y);
while  stack-not-empty  do
begin
  pop(xl,xr,y);
  y:=y+1;
  left-need:=False;
  while(getpixel(x,y)<>boundary-color).and.
       (getpixel(x,y)<>new-color)  do
  Begin
     /*向左填充，并用xsave返回填充区段左端点*/
     left-need-fill:=True;
  end;
  while x<xr do
  Begin
    span-need-fill:=False;
    while(getpixe(x,y)<>boundary-color) and
         (getpixel(x,y)<>new-color) do
    Begin
      if not span-need-fill need then
      begin
        span-need-fill:=True;
        if not left-need-fill then xsave:=x-1;
      end;
      setpixel(x,y,new-color);
      push(xsave+1x-1,x-1,)y);
      left-need-fill:=False;
      while(getpixel(x,y)= boundary-colar)  or
           (getpixel(x,y))=new-color) and(x<xr)do
              x:=x+1;
    end;
  end;
```

```
    y:=y-2;
    /*开始在下一条扫描线上检查，运算过程与在上一条扫描线上的过
    程完全一致，故略去详情，在具体实现时这一部分程序可考虑用一
    函数实现，分别代入不同y值调用。*/
  end;
  End./* 算法结束！*/
```

从上述算法描述中可以看到，压区段端点入堆栈的区域填充扫描线算法与原种子点入堆栈的扫描线算法相比，两种算法出入堆栈的次数是一样的，原算法中由于寻找区段种子点与填充是分开进行的，寻找种子点时扫描过的区域内的像素点在填充时又被重新扫描了一遍。只有在弹出栈顶元素时存在向左进行填充的像素点在原算法寻找种子点时才需扫描，因而不存在对这些像素点重复扫描。如果没有向左的这部分像素点，则几乎所有的像素点都被重复扫描，唯一的例外是事先给定的种子点所在的区段内的像素点（仅限于种子点所在的扫描线），这种情况是比较常见的，由算法描述及示意图：都可以看到只要当前扫描区段在区域的最左侧就可以了。对矩形区域，或一般地，左侧对齐的区域这一条件总是满足的。对梯形区域，如果种子点在中部，则有一半的扫描线满足这一要求。所以，定量化一些，对左侧对齐的区域，减少获取像素点颜色值并进行相应判定检查的次数是区域所含像素点的数目。

3.6 多边形扫描转换算法

多边形可用其顶点坐标来表示，因此用多边形可以方便直观地指定其内部作为一个区域. 但这样指定的区域的一个不便之处是没有明确指明某一个像素点是否属于这个区域. 对这一类区域的填充,可以采用先用前面的直线算法产生多边形的边界上的像素点，构成一用边界定义的区域，然后用边界填充算法进行填充，但是这种方法的效率较低。

实际上,多边形的边上的像素点之间有很明确的几何关系,产生出像素点表示的边界后,反倒成了没有相互关联的像素点.下面就从分析多边形边界与其内部点之间的关系入手,试图给出较好的算法以指出哪些像素点是属于这个区域,因而可以赋以指定的颜色.

3.6.1 扫描线上像素点的连贯性

点阵图形的像素点可以认为是由位于一条条水平直线上的像素点构成的.这样的每一条直线可简称为一条扫描线. 图3.4表示出一个多边形和一条穿过该多边形的扫描线，我们要决定扫描线上哪些像素点是在多边形内部，并对这些相应的像素点赋以合适的值表示某种颜色或灰度。对每一条切割多边形的扫描线，都重复这一过程，我们就能对整个多边形进行扫描转换。

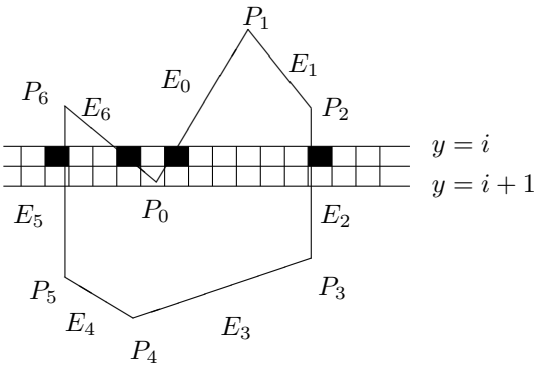


图3.4 多边形与扫描线关系示意图

不妨把一条扫描线设想为两端无限延长的直线,它与同一平面内的一个多边形的关系无外乎两种: 相交或不相交.如果相交,则直线被多边形分割成不同的直线段,其中一些在多边形内,另外一些在多边形外.并且这些多边形内外的直线段大多数情况下相互交替出现.

根据上面的分析,当像素点位于多边形内(外)的直线段上时,它就位于多边形内(外). 也就是说，与多边形相交的一条扫描线上总会有一组相互相连的像素点都位于多边形之内。这个性质称做为扫描线连贯性(scan line coherence)。根据这一性质，我们无需判断每一个像素点，而只需要判断哪些像素点所在的直线段在多边形之内。对这些直线段的确定只需求出扫描线与多边形边的交点即可.

求出扫描线与多边形的所有交点后,位于多边形内的直线段就不难找到了:设想我们从扫描线的一端出发, 这时我们在多边形外,因为多边形总是在有限范围内的.当我们沿直线前进到达第一个交点时, 就开始进入多边形内.由于直线的另一端是在多边形外,然后继续沿直线前进一定会走出多边形内部,于是一定会遇到第二个交点.这时我们可以看到,第一和第二个交点给出的直线段就位于多边形内.

如果没有更多的交点,则该扫描线上只有一段直线段位于多边形内.否则,继续沿扫描线前进, 但这时位于多边形外,直到遇到第三个交点后,重新进入多边形内.类似前面的分析,一定有第四个交点. 于是第三,第四个交点给出的直线段就位于多边形内.

如果还有更多的交点,就重复这样的过程,直到没有新的交点位置.因为交点个数是有限的, 这一过程是一定可以结束的.

也就是说，与多边形相交的一条扫描线上总会有一组相互毗连的像素点都处在多边形之内。这个性质称做为扫描线连贯性(scan line coherence)。因此，我们无需判断每一个像素点，而只需要判断哪些像素点段在多边形之内。

在图3.4中，第i个扫描线y坐标为$y = i$时，有两个像素点段在多边形之内。用以下三步就可以填充这两段区间：

(1)求出这条扫描线与多边形所有边的交点，这条扫描线只是与多边形的边$E_j, j = 0, 2, 5, 6$有交点, 交点的x坐标分别记为$x_{ij}, j = 0, 2, 5, 6$.

(2)对交点排序，从左到右按x坐标增加的顺序对这些交点进行排序，交点的排序表为$(x_{i5}, x_{i6}, x_{i0}, x_{i2})$。

(3)在每一对交点之间填充所有的像素点，即在区间$[x_{i5}, x_{i6}]$和$[x_{i0}, x_{i2}]$内填

充。

然而，仔细分析可以看出这个过程存在有潜在的问题。

假如排序表里某个交点既是走出(进入),同时也是进入(走出)多边形的边界点,这一交点就是一个奇点.如果奇点存在,那么上述填充过程就不能正确地进行，从图中可看出，仅当扫描线与多边形上取得局部极大值或局部极小值的顶点相交时，交点才会是一个奇点。图中多边形顶点P_0是一个取得局部极小值的顶点，而顶点P_6是一个取得局部极大值的顶点。扫描线$y = y_6$与多边形的交点数也是奇点。解决的方法就是把这样的交点作为二重交点处理。例如对于$y = y_0$的扫描线(这里y_0是点P_0的$y-$坐标)，得到的交点表是$(x_{y_0 5}, x_{y_0 6} = x_{y_0 0}, x_{y_0 2})$，填充的区间为$[x_{y_0 5}, x_{y_0 6}]$ 和$[x_{y_0 0}, x_{y_0 2}]$。也可以干脆忽略这样的交点.这时对于$y = y_0$的扫描线，得到的交点表是$(x_{y_0 5}, x_{y_0 2})$，填充的区间为$[x_{y_0 5}, x_{y_0 2}]$。

另外一个潜在的问题与交点的求解方法有关.我们只有通过扫描线所在直线与多边形各边所在直线段求解出交点的方法求出所有扫描线与多边形边界的交点. 如果某个交点恰恰是其中一个边的端点,那么这个交点也一定是另外一个边与扫描线的交点. 例如对于$y = y_5$ (这里y_5是点P_5的$y-$坐标)的扫描线，得到的交点表是$(x_{y_0 4} = x_{y_0 5}, x_{y_0 3})$，按上述方法,就会得出$[x_{y_0 5}, x_{y_0 3}]$不是一个应填充的区间的错误结论。错误的产生是因为$x_{y_0 4} = x_{y_0 5}$被作为两个交点来看待. 解决问题的方法是把当前扫描线$y = y_5$上方的边在端点处临时截掉一小段,就可以保证这样的交点$x_{y_0 5}$不出现,而使得$x_{y_0 4} = x_{y_0 5}$只会出现一个,于是就可以得到正确的结果.

这一解决问题的方法也可以用于对上面提到的奇点$y = y_6$的处理,只是对不同的奇点有不一致的处理方法: 一个取得局部极大值的奇点被认为是两个交点,而一个取得局部极小值的奇点不被认为是一个交点。

顺便提及的是关于水平边,它可能重合于某一条扫描线,因而有无数多交点.实际上,这样的边可以忽略而不予处理,但不影响最后的填充效果(为什么?).

3.6.2 不同扫描线与边的交点在边上的连贯性

前一小节分析了一条扫描线上在多边形内的像素点与交点的关系,最终的结果依赖于扫描线与多边形的边的交点的计算. 我们现在将进一步分析不同的扫描线与多边形边的交点之间的关系.由于多边形的边是连续的,不难看出,当平移一条扫描线时,它与多边形边的交点也在沿多边形的边连续地移动. 如果我们平移的距离不是太大,就可以期望大多数交点移动前后是在同一条边上的. 因此,我们有理由认为与扫描线i相交的许多条边也与扫描线$i + 1$相交。也就是说，相继的扫描线与同一条边相交，对这条边来说，它具有边连贯性(edge coherence)。设第i条扫描线与多边形的某条边E_j的交点的$x-$ 坐标为x_{ij}，第$i + 1$ 条扫描线与这条边的交点的x坐标为$x_{(i+1)j}$，这条边的斜率为m_j，则对于这条边，我们可以利用交点x_{ij}来计算$x_{(i+1)j}$:

$$x_{(i+1)j} = x_{ij} + 1/m_j \tag{3.6.1}$$

注意我们默认了扫描线i的$y-$坐标为i;扫描线$(i + 1)$的$y-$坐标为$(i + 1)$.

该式表明，我们可以利用这种边的连贯性，采用扫描线算法，一条扫描线一条扫描线地从低到顶(或从顶到低)对多边形进行扫描转换。当从一条扫描线转至下一条扫描线时，多边形上同一条边的交点坐标可容易地由前一条扫描线的交点用递推的方法得到，不必逐个解方程。

但毕竟不是所有与扫描线i相交的边和与扫描线$(i + 1)$相交的边是相同的,如下是几种例外情况:

(1). 一个交点恰恰是在扫描线i上下的两条边的公共端点;这时与扫描线$(i+1)$相交的多边形的边是与原来那条边相邻的边;

(2). 一个交点恰恰是在扫描线i下方的两条边的公共端点;这时这个交点是一个局部极大值,此时没有相应与扫描线$(i+1)$相交的多边形的边.即移动后,此处的交点消失了.

(3). 新的交点有可能出现. 如果两条边的公共端点是一个局部极小值,且恰在扫描线$i+1$上, 就增加了两个与新的扫描线$(i+1)$的交点.

具体实现时，对每一条扫描线，我们建立一个称之为活动边表(active edge table)的数据结构，简称AET表。这个表是一个动态表，其内容与扫描线位置有关，它包含与当前扫描线相交的所有边的信息。这些信息包括：

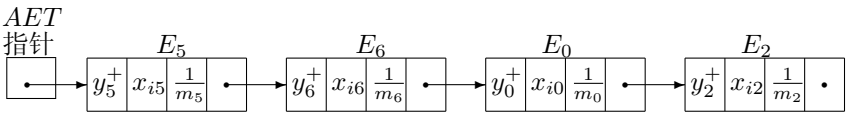
(1) 相交边E_j的上侧端点(y坐标较大的端点)的y_j^+坐标；

(2) 交点的x坐标即x_{ij};

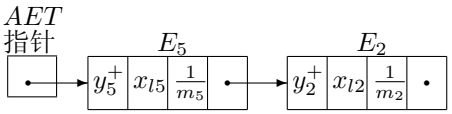
(3) 相交边斜率的倒数即$\frac{1}{m_j}$;

(4) 指向后面一条边的指针。

下面的图3.5画出了图3.4中两条扫描线的$y=i$和$y=i+1$ 的AET表。每一条边的AET表中的给出一组边以这些边与扫描线交点的x坐标递增排序。



(a). 扫描线$y=i$



(b). 扫描线$y=l, l=i+1$

图3.5 活动边表示意图

为了使得向AET表增加新边的效率较高，我们还需建立一个边表(edge table), 简称ET表。边表是一个静态表，包括除水平边外多边形的全部的边的相关信息. 可用桶式排序方法建立此表. 存储桶的数量等于扫描线数量。表中边是按照边的较小的$y-$坐标即下端点的$y-$坐标值递增顺序排列的，对于下端点的$y-$坐标相同的边，存放在同一个存储桶中，并按照下端点的$x-$坐标增加的顺序利用插入排序来组织的。在边表中存储的每条边的信息有与AET表中每条边的相同信息，即边的上端点$y-$坐标和边的斜率倒数以及指向后继边的指针。不同的信息是边表中交点信息被代之以边的下端点的x坐标。这里也对于水平边忽略不作处理。因此两者的数据结构是相同的. 图3.6画出了多边形的桶式排序表。

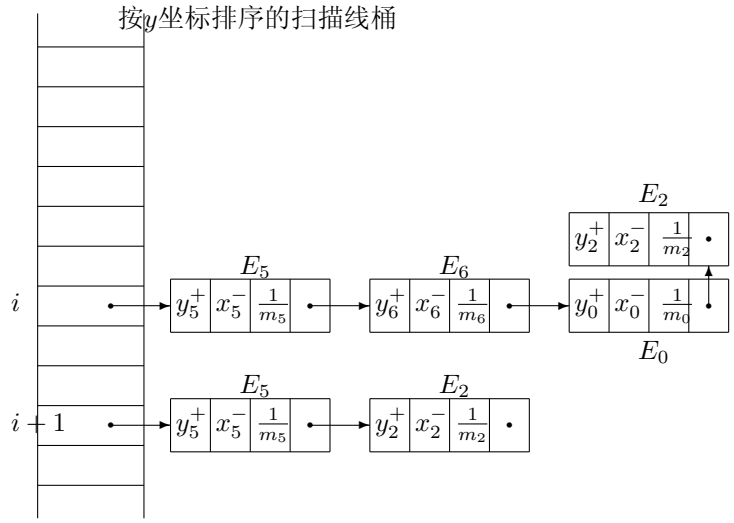


图3.6 桶式排序表示意图

3.6.3 扫描线算法处理步骤

一旦形成了ET表后，那么扫描线算法的处理步骤是：

1. 将y值设置为ET表中所列的最小y坐标值，亦即第一个非空的存储桶；

2. 将AET初始化,使其为空；

3. 重复下列步骤，直至AET为空并处理完ET表中所有桶：

- **3.1.** 把ET中存储桶y的信息与AET合并，并保持AET按x所作的顺序排列；
- **3.2.** 对于扫描线y，从左到右，对AET中每对x坐标值填充所需要的像素点值；
- **3.3.** 从AET表中删去$y = y_{max}$的项；
- **3.4.** 对于仍留在AET中的每一项，求下一条扫描线与边的交点，即以$x + 1/m$代替活动边表中的每个x项；
- **3.5.** 由于上一步有可能破坏AET表中各项按x所作的排序，所以对AET重新排序;
- **3.6.** 使y增加1，成为下一条扫描线的坐标。

需要注意的是，图3.6的桶式排序表中，边e_2和e_5是经过缩短处理的,以保证这些中间的顶点与扫描线只相交一次。

习题 3

1. 关于水平边,它可能重合于某一条扫描线,因而有无数多交点. 实际上,这样的边可以忽略而不予处理,但不影响最后的填充效果. 为什么?

2. 编写一个适用于八连通区域的注入填充算法。
3. 设计一个多边形扫描转换算法，使其边界像素点为某个值，而其内部像素点为另一值。
4. 自己给出一个多边形，并画出其AET表和ET表
5. 对一个左侧边垂直于上、下底，且上、下底比值为$1:2$的梯形区域，分析比较各种区域填充算法的计算量。

Chapter 4

平面图形裁剪

一般来说,图形就是要映像现实或想象中的物体的外表的形状. 在现实或想象的空间中图形可以有任意的,甚至无限大的尺寸. 当我们借助计算机处理图形时,我们总是处理有一定大小限制的图形, 最简单的也最经常要处理的问题是要在显示器上显示图形,而显示器是有限的. 甚至现在计算机中经常采用的窗口技术是我们只能在比满屏更小的显示器的部分区域内显示图形. 因此被显示的图形有落入所指定的窗口之内部分，同样也会有落在窗口之外的部分。由于这部分窗口之外的图形是不应被显示的，有必要对这些不能显示的部分图形进行舍弃。决定图形的哪部分在窗口内，哪部分在窗口外，这样的一个过程就称为对图形的裁剪，它是确定在窗口内可见图形部分的一种处理过程。

对一个图形的裁剪过程实际上就是需要逐个判定构成图形的基本的图形元素,如点,线段,文字等等是否包含在指定的窗口区域内, 对线段,文字还要判定是否仅其部分在指定窗口区域内. 由于图形经常包含大量的点或线段,甚至文字等，裁剪的运算量非常大，裁剪算法的效率就显得十分重要。而裁剪的方法很多, 其效率的高低常和图形的特点、计算机功能等因素有关，因此要根据实际情况来选择裁剪方法。在用软件实现常达不到实时应用环境所提出的对图形的速度要求时，要考虑由相应的硬件实现。

4.1 二维裁剪概念

平面上的图形受该平面上的矩形窗口的裁剪称为二维裁剪，通常窗口是由图4.1中参数x_L, x_R, y_B和y_T所决定的. 因此有窗口的左下角和右上角坐标为(x_R, y_T)和(x_L, y_B).

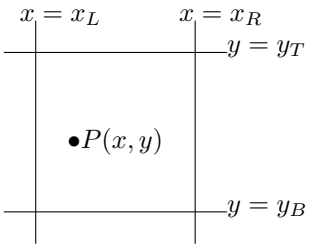


图4.1 矩形裁剪窗口、点可见性

4.1.1 点的裁剪

点的裁剪比较简单. 点(x, y)为可见的充分必要条件是其坐标满足以下两组不等式：

$$\begin{cases} x_L \le x \le x_R \\ y_B \le y \le y_T \end{cases} \tag{4.1.1}$$

其中，等号成立标明点在窗口边界上，也认为是可见的。

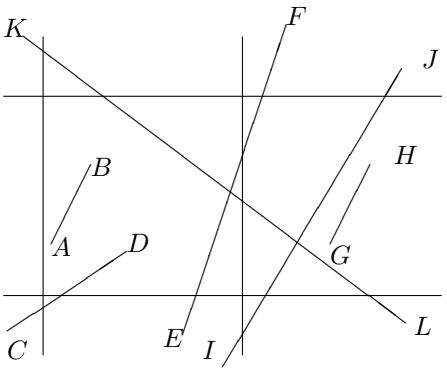


图4.2 线段的可见性

4.1.2 直线段的裁剪

直线的裁剪比点的裁剪复杂，如图4.2所示。此时直线段与窗口的关系有完全可见的、部分可见的或完全不可见的三种。如果是完全可见的，则输出其已知的两个端点坐标并显示这条直线段；如果是部分可见的，则输出可见部分线段的两个端点，并显示这部分线段。要判别一条直线段的可见性，就要根据直线的两个端点与窗口的相对位置关系，分别判定. 依据图4.2可把这些位置关系归纳为如下几种情况：

(1) 如果直线的两个端点都在窗口内如直线段AB，则这样的直线段是完全可见的。

(2) 如果直线的一个端点在窗口内，另一个端点在窗口外如直线段CD，则它只是部分可见的, 需要对它裁剪，即求出它与窗口边框的交点，该交点和窗口内的那个线段端点是可见线段的两个端点。

(3) 如果直线的两个端点都在窗口外，并且是在窗口某边框所在直线的同一侧如直线段GH，则这样的直线完全不可见，可简单地剔除。同一侧是指线段的两个端点同时位于窗口边框线为界的窗口的左面、或右面(如GH)、或上面、或下面。

(4) 如果直线的两个端点都在窗口外，并且不在窗口某一边框所在直线的同一侧如直线段EF、IJ和KL，此时需要求出直线段与窗口边框的交点，并对交点性质进行分析后才能确定是否需要裁剪。这时直线段有可能部分可见如EF,KL也有可能完全不可见如IJ。

可见，为了判别直线的可见性，需要求出直线与窗口边框的交点，然后对交点性质作分析. 求交直线与窗口边框点，可以用直线的参数表示形式或非参数表示形式。

我们首先求出直线与窗口边框的交点, 设直线的两个端点坐标分别为$P_0 = (x_0, y_0)$、$P_1 = (x_1, y_1)$，则易得直线的点斜式方程为

$$y = m(x - x_0) + y_0 \tag{4.1.2}$$

其中m为斜率

$$m = \frac{\Delta y}{\Delta x} \tag{4.1.3}$$

而

$$\begin{cases} \Delta x = x_1 - x_0 \\ \Delta y = y_1 - y_0 \end{cases} \tag{4.1.4}$$

它与窗口左,右,上,下诸边框所在直线的的交点P_L, P_R, P_T, P_B 的坐标分别为

$$\begin{cases} x = x_L,\ y = m(x_L - x_0) + y_0 \\ x = x_R,\ y = m(x_R - x_0) + y_0 \\ y = y_T,\ x = x_0 + \dfrac{1}{m}(y_T - y_0) \\ y = y_B,\ x = x_0 + \dfrac{1}{m}(y_B - y_0) \end{cases} \tag{4.1.5}$$

然后，判别这些交点是否在窗口边框上或边框的沿长线上。如在沿长线上，则剔除该交点(如下图中的P_L,P_T),最后余下两个交点(如下图中的P_B,P_R),给出直线在窗口内部分的直线段。这一直线段与P_0P_1的公共部分(如下图中的P_0P_R)就是P_0P_1的可见部分.

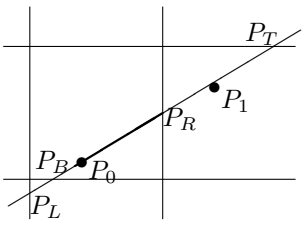


图4.3 直线段的二维裁剪

这一直接求解的方法清楚地解释了线段二维裁剪的含义, 但不是一个有效的直线段裁剪方法，下面一节将介绍一些有效的裁剪算法。

4.2 直线段的裁剪算法

4.2.1 科恩-萨塞兰德算法

1974年Dan Cohen和Ivan Sutherland提出用编码方法来实现裁剪。窗口每条边框所在的直线把平面分成两半:窗口所在的一半(包括边框所在的直线)和窗口所不在的一半(不包括边框所在的直线). 平面上一点可依据它位于两这种的哪一个而有一个编码0或1. 对应于窗口左,右,下,上的四条直线, 平面上每一点有一个四

位的编码.四条直线把平面分成九个区域,一个点的四位编码与窗口的位置关系如图4.4所示.

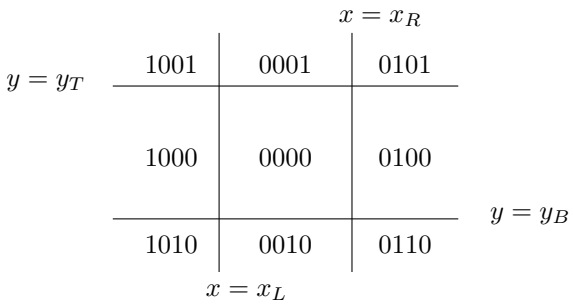


图4.4 点的区域编码对照图

由编码规则可知，

(1). 若线段两端点的编码均为零，两端点均在窗口之内，线段完全可见。

(2). 若将线段两端点的编码逐位取逻辑“与”的运算，结果非零，则该线段两个端点的某一位编码全为1.这说明该线段两个端点都位于那一位都为1的编码相应的边框线的不包含显示窗口的那一侧,因而必为完全不可见线段。

(3). 如果上述判断不能定论，需要计算出直线段和窗口边框所在直线的一个交点，这个交点把直线段分成两段，且其中至少有一条直线段可由上述规则判定为是显然完全不可见的直线段.把这条显然完全不可见的直线段抛弃，而对于另一部分线段转第一步，重复进行上述判断,直到得出肯定的判断为止.

上述分析给出了Cohen-Sutherldnd裁剪算法的思想. 下面的过程Clip描述了这一算法, 其中用一个集合(top, bottom, right, left)来表示端点的编码。

```
var x_L ,x_R ,y_T ,y_B :rea1;
Procedure  clip(x0,y0,x2,y2:real);
label  return;
type  edge=[top,bottom,right,left];
outcode= set  of edge;
var c, c1, c2: outcode;
    x, y: real;
Procedure Code(x, y: real; var c:outcode);
Begin
    c:=[];
    if x<x_L then c:=[left];
    if x>x_R then c:=[right];
    if y<y_B then c:=c+[bottom];
    if y>y_T then c:=c+[top];
End
Begin
    Code(x0, y0,cl);
    Code((x2, y2, c2);
```

```
    While(cl<>[]) or (c2<>[])
    Begin
        if(c1*c2<>[]) then  goto  return;
        c:=cl;
        if c:=[] then  c:=c2;
        if left in c then y:=y+(y2-y0)*(x_L -x0)/(x2-x0);
        if right in c then y=y0+(y2-y0)*(x_R -x0)/(x2-x0);
        if bottom in c then x:=x0+(x2-x0)*(y_B -y0)/(y2-y0);
        if top in c then x:=x0+(x2-x0)*(y_T -y0)/(y2-y0);y:=y_T
        if c=cl then x0=x;y0=y;code(x,y,c1)
        x2:=x;y2=y;code(x,y,c2)
    End;
      {*if we reach here, the line-segment from (x0,y0)
        to (x2,y2) is  visible*}
      if x0<>x2 or y0<>y2 then Showline(x0, y0, x2, y2);
End
```

4.2.2 中点分割算法

上节给出的裁剪算法，需要计算被裁剪线段与裁剪窗口各边的交点。所用的求交点的方法是精确的.考虑到计算机中表示的数据可以是在一定精确度要求下的结果,我们可以用折半分割的方法进行近似求交. 这就是中点分割算法。采用折半分割计算实际上是计算机的移位运算,运算速度快,且便于硬件实现。

该算法思想如下:对不能判定为完全可见或完全不可见的线段用线段中点将其一分为二, 有如下几种情况.

(1) 如果线段的中点在窗口外或边框上,则分割后的两条线段至少有一条直线段是显然完全不可见的直线段.把这段显然完全不可见的直线段抛弃，原直线段与窗口的交点必不在这一分割前的直线段上. 而对于另一部分线段重新作判断, 如果如果两条直线段都是完全不可见的直线段,则分割前的直线段完全不可见.

(2) 如果线段的中点在窗口内,且其中一段完全在窗口内,则原直线段的一个端点是可见的部分的一个段点,且原直线段与窗口的交点必在另一直线段上.这时另一直线段应不能判定为完全可见或完全不可见的.(如若不然,则分割前的直线段完全可见或中点是与窗口边框的交点,都是前面已处理过的情况.)

(3) 如果线段的中点在窗口内,且其中一段不能判定完全在窗口内,则原直线段与窗口的交点必有一个在这一直线段上.对其重新用中点进行分割判断.

(4) 如果线段的中点在窗口内,且分割后的两条直线段都不能判定完全在窗口内, 则原直线段与窗口有两个交点,且分别在这两个直线段上.对其分别重新用中点进行分割法判断.

(5) 若分割后的线段长度小于指定的精度,但仍有不能判定为完全可见或完全不可见的线段,此时即认为这样的线段的中点即为原线段与窗口边框线的交点, 并以此按上述规则进行判定.

本算法实现过程中对直线段可见性的判定一般也采用上节的线段端点编码和相应的检查方法来进行。

可以看出，本算法的思想是用对半分割的方法不断地去除线段的不可见部分，具体的实现方法是用中点逼近线段与窗口边的交点，去除线段的所有不可见部分。反复折半对分有关线段、最终一定会找到直线与窗口边或其延长线的交点. 而点阵图形是由像素点构成的,具有一定的精确度. 故交点计算时的精度达到点阵图形的精度就可以了,更高的精度在这里是无意义的.依此分析,如果直线段上有N个像素，则对分搜索次数最多为log_2N.显然，中点分割算法的执行时间决定于图中所含直线的长度、数量和点阵图形的分辨率，如对要显示在分辨率为1024×1024的显示器上的图形，求一个交点最多对分10次就可以了。

4.2.3 梁友栋-Barsky算法

设要裁剪的线段是P_0P_1，端点P_i的坐标为$(x_i, y_i), i = 0, 1$. P_0P_1 和窗口的边框所在的直线交于A,B，C和D四点，如图4.5所示. 算法的基本思想是从P_B，P_L和P_0三点中找出最靠近P_1的点，图4.5中要找的点是P_0。从P_T, P_R和P_1中找出最靠近P_0的点，图4.5中要找的点是P_R点。因此P_0P_R就是P_0P_1线段上的可见部分。

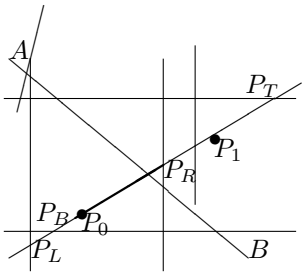


图4.5 有向线段的裁剪

对上述结果进一步分析,可以看到两点:

(1). 同一线段上点的相互位置关系可通过其在直线上的位置顺序关系来确定. 这个顺序位置关系在直线用参数方程表示时可通过其相应参数值的大小顺序关系确定.

(2). 要想从P_B，P_L和P_0三点中找出最靠近P_1的点，和从P_T, P_R和P_1中找出最靠近P_0的点与直线的方向有关. 图4.5中的有向线段P_0P_1是从左下方到右上方的, 于是整条直线可以认为是从左下方到右上方运动产生的,其进入窗口的点只可能是与左,下边框线的交点. 离开窗口的点只可能是与右,上边框线的交点. 当线段方向改变时,这样的关系就会变化.线段的方向可通过$\Delta x = x_2 - x_1$和$\Delta y = y_2 - y_1$的正负来判断.

根据上述分析,首先把P_0P_1写成参数方程的形式

$$\begin{cases} x = x_0 + \Delta x \cdot t \\ y = y_0 + \Delta y \cdot t \end{cases} \tag{4.2.1}$$

则P_0点参数为$t = 0$, P_1点参数为$t = 1$.

其次, 根据直线段的方向把窗口的四条边分成两类：一类称为始边，另一类称为终边。以图4.5中的直线段P_0P_1为例,设想一点在直线段P_0P_1所在直线左下方沿直线方向运动,则始边是该点(在进入窗口之前)先遇到的两条边, 则终边是该点(在进入窗口之前)后遇到的两条边. 根据直线段的方向不同的一般对应关系如下:

(1). 当$\Delta x \geq 0(\Delta y \geq 0)$时, 直线方向是从左(下)到右(上),称$x = x_L(y = y_B)$为始边，$x = x_R(y = y_T)$为终边。

(2). 当$\Delta x < 0(\Delta y < 0)$时，直线方向是从右(上)到左(下), 称$x = x_L(y = y_B)$为终边，$x = x_R(y = y_T)$为始边。

对图4.5的P_0P_1来说，$x = x_L$ 和$y = y_B$ 为始边，$x = x_R$ 和$y = y_T$ 为终边。对AB来说，$x = x_L$ 和$y = y_T$ 为始边，$x = x_R$ 和$y = y_B$为终边。对BA来说，由于方向相反, 就有$x = x_R$ 和$y = y_B$ 为始边，$x = x_L$和$y = y_T$为终边。

对边分类后, 求出P_0P_1分别与$x-$轴和$y-$轴平行的两条始边的交点的参数t_{sx}和t_{sy}，令

$$t_s = max(t_{sx}, t_{sy}, 0) \tag{4.2.2}$$

其中$t = 0$为作为线段始点的P_0点对应的参数值, 则t_s就是图4.5中P_B,P_L 和P_0三点中最靠近P_1的点的参数值。求出P_0P_1分别与$x-$轴和$y-$轴平行的两条终边的交点的参数t_{ex}和t_{ey},令

$$t_e = min(t_{ex}, t_{ey}, 1) \tag{4.2.3}$$

其中$t = 1$为作为线段终点的P_1点对应的参数值, 则t_e就是P_T, D和P_1三点中最靠近P_0的点的参数。当$t_e > t_0$时，方程4.2.1中参数$t \in [t_s, t_e]$对应的线段就是P_0P_1的可见部分。当$t_s \geq t_e$时，整个直线段为不可见，图4.5中为给出的两条未标示的线段就属于这种情况。

为了确定始边和终边，并求得P_0P_1与它们的交点，令

$$\begin{cases} Q_L = -\Delta x,\ D_L = x - x_L \\ Q_R = \Delta x,\ D_R = x_R - x_0 \\ Q_B = -\Delta y,\ D_B = y_0 - y_B \\ Q_T = \Delta y,\ D_T = y_T - y_0 \end{cases} \tag{4.2.4}$$

则交点的参数为

$$t_i = \frac{D_i}{Q_i}, i = L, R, B, T \tag{4.2.5}$$

这里t_L是P_0P_1和$x = x_L$的交点的参数，t_R，t_B和t_T有类似的意义。

分析Q_i和D_i的的取值,可以看出:

(1). 当$Q_i < 0$时，相应的t_i必是P_0P_1和始边的交点的参数。

当

(2). 当$Q_i > 0$时，相应的t_i是P_0P_1和终边的交点的参数。

(3). 当$Q_i = 0$时，直线段P_0P_1和相应于Q_i的窗口边框线平行.

此时若$Di < 0$，则P_0P_1与窗口在和相应于Q_i的边框线的不同侧,因而是完全不可见的(如果有$Q_T = 0$，$D_T < 0$,则有$y_0 = y_1 > y_T$, 因而P_0P_1位于窗口上方而不可见.).

当$Q_i = 0$且相应的$D_i \leq 0$, 则直线段P_0P_1和相应于Q_i的边框线平行,且是夹在相平行的这两条边框线中间.具体结果如下:

如果$Q_L = 0$，$D_L \geq 0$, 则同时有$Q_R = 0$，$D_R \geq 0$(反之亦然.)这时由于P_0P_1和$x = x_L$及$x = x_R$平行，故为了判定直线段上的可见部分不需要求出P_0P_1和$x = x_L$或$x = x_R$的交点。在实际算法实现时可以认为$t_{sx} = 0$,$t_{ex} = 1$.

如果$Q_B = 0$，$D_B \geq 0$, 则同时有$Q_T = 0$，$D_T \geq 0$(反之亦然.)这时由于P_0P_1和$y = y_B$及$y = y_T$平行，故为了判定直线段上的可见部分不需要求出P_0P_1和$y = y_B$或$y = y_T$的交点。在实际算法实现时可以认为$t_{sy} = 0$,$t_{ey} = 1$.

如果四个Q_i同时为0,则直线段退化为一个点.

此算法的代码程序是

```
var xL,xR, yT , yB : real;
Procedure clip(x0,y0,x2,y2:real)
label l;
var  t0, t1, deltax, deltay:real;
procedure  clipt(q,d: real;var t0,t1:real);
var e:real
begin
  if q<0 then
  begin
    r:=d/q;
    if  r>t1  then  goto  l
    if  r>t0 then  t0=r
    if q>0 then
    begin
      r:=d/q;
      If  r<t0  then goto  l
      if  r<t1 then tl=: r
      if  d<0  then  goto l
    end:
  end:
end
begin
  t1:=1;
  deltax:=x2-x0;
  clipt(-deltax,x0-x1,t0, t1);
  clipt(deltax,xR -x0,t0,t1);
  deltay:=y2-y0;
  clipt(-deltay,y0-yB ,t0,t1);
  clipt( deltay, yT -y0, t0,t1);
  y2:=y0+t1*deltav;
  x0:=x0+t0*deltax;
  y0:=y0+t0*deltay;
   showline(x0,y0,x2, y2);
l:
end
```

4.3 多边形逐边裁剪法

多边形的裁剪有它的特殊性。初于此问题容易使人产生一种错觉，把多边形看作是直线段的集合，从而将多边形分解为一条一条的线段, 只要每一条边用对直线段裁剪方法裁剪后，就完成了对多边形的裁剪。其实不然，如此裁剪后，原来封闭多边形变成一个或多个折线段(开口的多边形) 或散开的直线段集合.

这样的结果通常不是我们所需要的结果. 在图形学中的多边形常常是一封闭多边形，它把平面分成多边形内和外.我们需要的并不是多边形本身，而是由多边形作边界而给出的多边形的内部区域.

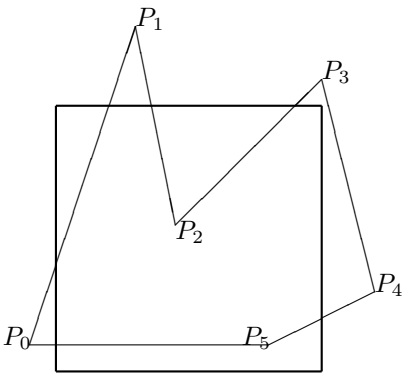


图4.6 裁剪前的多边形 $P_0P_1P_2P_3P_4P_5P_0$

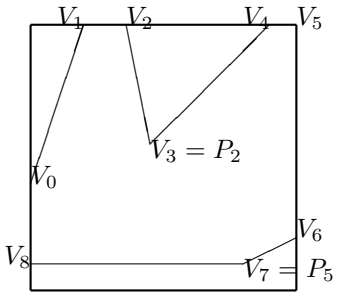


图4.7 裁剪后的多边形 $V_0V_1V_2V_3V_4V_5V_6V_7V_8$

因此对一个多边形的裁剪结果仍要求是多边形，且原来在多边形内的点也在裁剪后的多边形内，反之亦然。一部分窗口的边界可能成为裁剪后的多边形的边界，一个凹多边形裁剪后可能成为几个多边形，如图4.8所示。

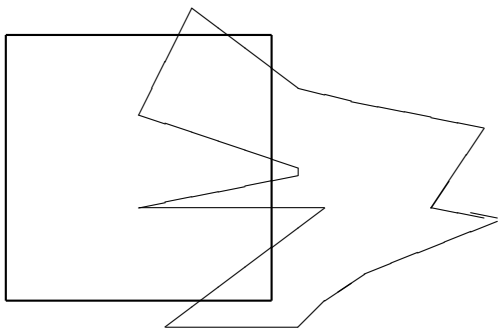


图4.8 裁剪后可能产生多个多边形的情形

对多边形裁剪的Sutherland-Hodgman算法是一种十分简便的方法，只要对多边形用窗口的四条边依次裁剪四次(见图4.9) 便可得到裁剪后的多边形。

在直线裁剪的编码算法中我们提到每条边框所在的直线把平面分成两半:窗口所在的一半(包括相应边框所在的直线)和窗口所不在的一半(不包括边框所在的直线)，分别称为(含)窗口侧和非窗口侧(即不含窗口侧)(当然是相对于指定的边框线而言的). 这样通过这条窗口边界可把落在非窗口侧的图形去掉，只保留在窗口侧的图形，连续用四条边框线裁剪后,就可得到裁剪后包括在窗口内的多边形。

算法的裁剪过程是通过将多边形依次与窗口的每一条边界相比较进行，该算法执行的结果是产生一组顶点，用来定义被裁剪的多边形。设想多边形是从P_0出发, 顺次序沿各边经过各个顶点P_1, P_2, ..., P_{n-1}运动产生的, 最后仍结束于P_0, 并定义$P_n = P_0$. 对指定的一条边框线E, 具体裁剪步骤如下:

(1). 设定当前所在的边为第一条边,即P_iP_{i+1}, $i = 0$. 如果P_0在窗口区,则P_0也是裁剪后的多边形的起点,我们的运动的位置状态当前是在窗口侧. 边号i增加1.

(2). 如果下一条边,即P_iP_{i+1},与E有交点, 则交点成为裁剪后的多边形的一个顶点,新的当前位置状态发生变化: 如果原来的当前位置状态是在窗口侧, 则新的当前位置状态就是在非窗口侧;如果原来的当前位置状态是在非窗口侧, 则新的当前位置状态就是在窗口侧. 边号i增加1, 如果$i < n$, 重复执行本步骤. 如果$i = n$, 执行步骤(4).

(3). 如果下一条边,即P_iP_{i+1},与E没有交点, 新的当前位置状态不发生变化: 如果原来的当前位置状态是在窗口侧, 则新的当前位置状态仍在窗口侧. 同时, P_{i+1}成为裁剪后的多边形的一个顶点;如果原来的当前位置状态是在非窗口侧, 则新的当前位置状态也就仍是在非窗口侧. 边号i增加1, 如果$i < n$, 重复执行步骤(2). 如果$i = n$, 执行步骤(4).

(4). 顺次连接先后产生的各个裁剪后的多边形的顶点, 并闭合它, 就得到裁剪后的多边形, 并把它作为下一次待裁剪的多边形。

(5). 连续用矩形窗口的四条边界对要裁剪的多边形进行裁剪，则原始多边形即被窗口裁剪完毕.

我们用图4.9来说明这一过程。图4.9 (a)是原始多边形和裁剪窗口之间的位置，原始多边形为$P_0P_1P_2P_3P_4P_5P_6P_0$，经过窗口左边框的裁剪后，去掉在非窗口侧的图形，下一次待裁剪的多边形为$P_0P_1ABP_3P_4P_5P_6P_0$，这一步首先要求出窗口左边框和多边形各边的交点，然后把这些点按照一定的原则连成线段，而与窗口左边框不相交的多边形的其它部分保留不动，则可得到下一次待裁剪的多边形

$$P_0P_1ABP_3P_4P_5P_6P_0,$$

如图4.9 (b)所示。

类似，用窗口右边框裁剪多边形

$$P_0P_1ABP_3P_4P_5P_6P_0$$

便可得到下一次的待裁剪多边形

$$P_0P_1ABP_3P_4DCP_6P_0,$$

如图4.9 (c) 所示；经窗口下边框对多边形

$$P_0P_1ABP_3P_4DCP_6P_0$$

的裁剪便得到下一次待裁剪的多边形

$$EP_1ABP_3P_4DCFE,$$

如图4.9 (d) 所示；再经窗口上边框对多边形

$$EP_1ABP_3P_4DCFE$$

的裁剪便可得到多边形

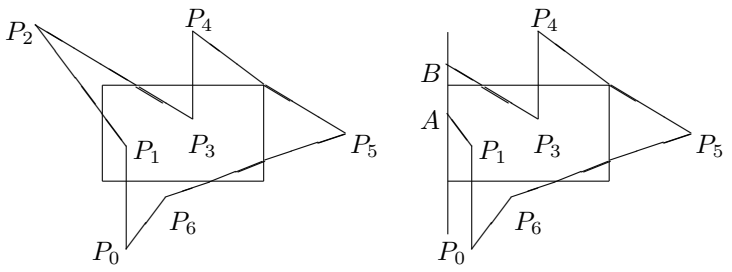
$$EP_1AGHP_3KDCFE$$

如图4.9 (e)所示。

最后裁剪出来的多边形为

$$EP_1AGHP_3KDCFE,$$

如图4.9 (f)所示。



(a). 裁剪窗口与多边形　(b). 用左边框裁剪

(c). 用右边框裁剪　(d). 用下边框裁剪

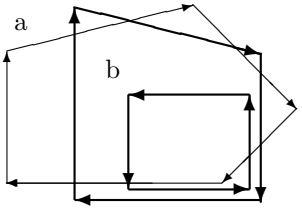
(e). 用上边框裁剪　(f). 裁剪后的多边形

图4.9 多边形的逐边裁剪过程示意图

上述求解过程中也存在着类似于多边形扫描转换算法中交点的奇异情况, 即窗口边框线与边的交点恰恰是边的端点, 在这里也有类似的处理. 另外, 窗口边框线与边重合时, 可以略去重合的边.

4.4 多边形窗口的双边裁剪法

裁剪窗口常常采用矩形窗口直接来源于计算机是在一个矩形的显示器上来显示图形. 当然, 现实中的图形也大多是出现在一个矩形的载体如纸张上. 但毕竟不是全部的图形都出现在一个矩形的区域内. 比如计算机动画中运行于一限定范围内的物体, 应当不显示出其不在限定范围内的部分. 这时限定范围的边界可能不是一矩形, 而是一个个多边形, 同时物体本身也是通过物体的多边形边界表示出来的. 这种情形前面讨论的裁剪算法就不适应了. 由Weiler和Atherton提出的双边裁剪算法可处理这种类型的裁剪问题。



a 标记主多边形外部　　b 标记裁剪多边形内部

图4.10 主多边形与裁剪多边形
边的方向及相互关系

在这一类问题中, 被裁剪的多边形简称为主多边形(Subject polygon, 在图4.10中用细线画出的多边形,为将要被裁剪的多边形区域)，裁剪的多边形区域称为裁剪多边形(clip polygon,在图4.10中用粗线画出的两个多边形围出的环形带)。裁剪的目的是要找出主多边形内部区域包含在裁剪多边形内部的那一部分. 如果不严格区分两者的含义的不同,实际上是要找出两者的公共部分,即其交集.

两个多边形的边界均用顶点的有序环形表来定义，多边形的外部边界取顺时针方向，而其内部边界或内孔取逆时针方向，以保证当遍历环形顶点表时，多边形内部总是位于前进方向的右侧，并以此定义了多边形边界的正方向(在图4.10中用箭头指明的方向)。主多边形与裁剪多边形边的交点总是成对出现，其中一个是主多边形进入裁剪多边形的交点称为进入交点，另一个是从裁剪多边形出来的交点称为离开交点。对于交点要仔细判定交点的有无(即是否为有意义的交点)以及它们的类型。对于如图4.11所示的结果，顺序给出的具体的判定规则为:

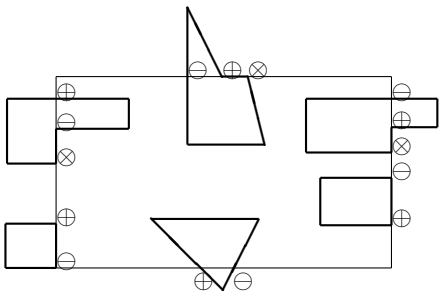
(1). 当主多边形的一条边位于裁剪多边形的一条边上时，这两条边之间应看作没有交点。

(2). 当主多边形的一个顶点位于裁剪多边形的边界上时，作为主多边形的一条边的起点,这个顶点应看作不是一个交点。

(3). 当主多边形的一个顶点位于裁剪多边形的边界上时，作为主多边形的一条边的终点, 这个顶点应看作是一个交点。且其类型为: 如果该边的起点在裁剪窗口外, 则该顶点是进入交点；反之，如果该边的起点在裁剪窗口外，则该顶点是离开交点，

(4). 当主多边形的一条边与裁剪多边形的边的交点不是主多边形的一个顶点时，是一个正常的交点。且其类型为: 如果该边的起点在裁剪窗口外, 则

该交点是进入交点；反之，如果该边的起点在裁剪窗口内，则该交点是离开交点.



⊕是进入交点, ⊖ 是离开交点，⊗不算交点

图4.11 主多边形与裁剪多边形交点分类

裁剪算法从一个进入交点开始，然后沿主多边形的边界正方向遍历，直至遇到与裁剪多边形的交点。在交点处顺时针方向旋转，再沿裁剪多边形的边正方向遍历，直至遇到与主多边形的交点，这表示主多边形又将进入裁剪多边形内部。在交点处顺时针方向旋转，再沿主多边形的边正方向遍历。重复这个过程直至遇到这个裁剪过程的开始点，我们便得到裁剪窗口内的封闭多边形即为裁剪的结果，在下面的算法中还包含外裁剪的过程. 整个算法包括以下步骤：

第一步: 建立主多边形和裁剪多边形各自的环形顶点表，外边界顶点顺时针次序存放，内边界顶点逆时针次序存放，并且每个边界顶点表中最后一个是第一个的重复。

第二步: 计算主多边形各边和裁剪多边形各边的交点，对每个交点标号，并插入到两个多边形的顶点表中，对同一交点，在主多边形和裁剪多边形的顶点表之间建立双向指针。

第三步: 建立两类交点表，即进入交点表和离开交点表。进入交点表存放主多边形进入裁剪多边形时产生的交点，离开交点表存放主多边形离开裁剪多边形时产生的交点，沿着多边形边界，两类交点将交替出现，因此只要测试其中一个即可决定所有交点的类型。

第四步: 建立两个装入表，存放裁剪后多边形的顶点。其中一个称为窗内装入表，存放位于裁剪多边形内部的多边形，另一个称为窗外装入表，存放位于裁剪多边形外部的多边形。建立这两个表，可根据主多边形和裁剪窗口的关系来进行。

首先，当主多边形完全在裁剪窗口外或内时,则只要将主多边形顶点装入窗外装入表或窗内装入表即可。其次、当裁剪窗口在主多边形内部时，裁剪窗口边界将成为裁剪后多边形边界的一部分。因此要将裁剪窗口的顶点装入两个裁剪后的多边形顶点表。例如，当裁剪窗口在主多边形内部时，裁剪后的窗外多边形的外边界由原主多边形顶点决定，而内边界是由裁剪窗的顶点决定的. 此时，位于主多边形内的裁剪多边形边界将构成主多边形的一个内孔。

同时，裁剪窗口的顶点又决定了裁剪后窗内主多边形的外边界。最后，若主多边形和窗口边界相交时，则执行以下的裁剪操作过程. 确定裁剪多边形内的主多边形过程如下：

(1). 从进入交点表中取出一交点，如果该交点没有处理过，则执行以下操作，否则取出下一个交点，直至表空过程结束。

(2). 沿着主多边形顶点表正向查找直到发现下一个交点，将主多边形顶点表中到该交点为止的部分复制到窗内装入表中。

(3). 把连接指针转到裁剪多边形的顶点表中。

(4). 沿裁剪多边形顶点表正向查找直到发现下一个交点，将裁剪多边形顶点表中到该交点为止的部分复制到窗内装入表中。

(5). 把连接指针转回至主多边形顶点表。

(6). 重复步骤(2)到(5)直到回到过程开始的进入交点、此时在窗内装入表中存放了裁剪后在窗内的封闭多边形顶点表。

(7). 转到步骤(1)寻找可能的其它窗内封闭多边形。

位于裁剪多边形外的主多边形可用类似的过程产生，只是起始交点取自离开交点表。在裁剪多边形顶点表中的遍历按反方向进行，并且将有关数据复制到窗外装入表中。

4.5 文本裁剪

在图形中，总免不了要有必要的,适当的文字说明. 因此作为图形的一部分, 就有对文本裁剪的问题. 设有如图4.12 所示处于裁剪多边形中的原始字符串, 文本裁剪有其特殊性, 根据文本字符产生的方法，以及用户对其要求的不同, 可有以下三种不同的裁剪方法,分别叙述如下.

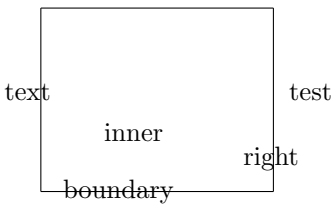
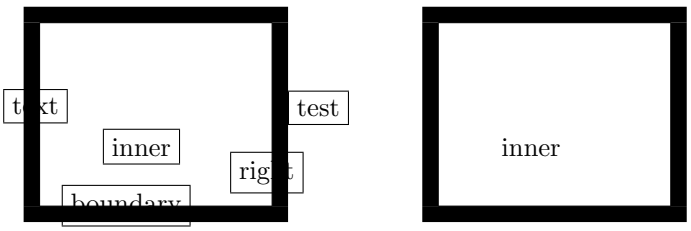


图4.12 原始字符串与裁剪多边形

4.5.1 文本的字符串裁剪法

如图4.13所示，设想有包含整个文本的字符串的矩形边框, 可称其为文本的边框. 如果文本的边框全部含在窗口内, 则整个文本字符串可见; 否则, 则认为整个文本字符串不可见而舍去. 由于是字符串作为一个整体全部可见，或者全部不可见，故称这种裁剪方法为字符串裁剪. 具体实现时, 只需判定文本边框的两条对交线是否全部可见就可以了, 因此这是文本裁剪的最简单方法，而且裁剪的速度也非常地快。如果图形中的字符串是作为文字说明而出现的, 不完整的字符串也就起不到说明的作用, 这时就适合于用文本的字符串裁剪方法.



字符串的边框位置　　字符串裁剪的结果

图4.13 文本的字符串裁剪

4.5.2 文本的字符裁剪法

基本思想类似于文本的字符裁剪法，但在此要分别考虑只包含一个字符的各个边框，要删除的也仅仅是不完全位于窗口内的字符，具体实现时, 需求出文本边框的一条对交线与窗口边框线的交点, 判定处在窗口边框上的字符, 然后把字符串一分为二就可以了. 如图4.14所示，只要字符与裁剪边相交或者在其外面，都将被删除。

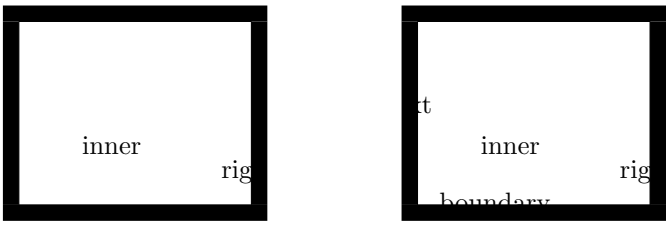


图4.14 文本的字符裁剪　　图4.15 文本的笔划裁剪

4.5.3 文本的笔划裁剪法

这种方法是把每一个字符看成一系列短直线即笔划的集合。因此，字符的裁剪就归结为对组成这些字符的笔划的裁剪。裁剪必须逐条(组成笔划边框的)直线地进行。如果一个字符与裁剪窗口边界重叠，只删除该字符在窗口外的部分，如图4.15所示。这种裁剪方法精度高，如实反映了字符的裁剪结果，但是因为要对字符的每个笔划进行裁剪，因此裁剪过程较长. 当然应用这种方法的前提是对由曲线构勒笔划组成的字符要知道字的笔划结构,对由点阵组成的字要清楚字的点阵结构.

Chapter 5

向量、矩阵概念及其运算

5.1 向量的基本概念

在自然科学和工程技术中所遇到的量可以分为标量和向量两种类型：仅有大小的量叫做标量，如体积、能量、质量、电荷等; 既有大小又有方向的量叫做向量（或矢量），如力、位移、速度、电场强度等。

我们知道, 直线段是最基本, 也是最简单的图形, 是表示出复杂图形的基础图形. 确定一条直线段, 仅知道其长度是不够的, 还必须知道其指向, 即方向. 向量代数研究的是向量，向量在物理学、几何学中具有广泛的应用，它使许多问题的解法简捷而直观。向量代数已经成为计算几何中的一种公认的语言。

向量是有大小和方向的量，常用一条带有方向的线段来表示。有向线段的长度表示向量的大小, 有向线段的方向表示向量的方向。向量的大小又叫作向量的长度或模(长)。一条以A为起点，B为终点的有向线段表示的向量记为$\overrightarrow{AB}$, 它的模记作$|\overrightarrow{AB}|$. 向量也可用一个小写字母加箭头来表示, 如$\vec{a}$, $\vec{b}$, $\vec{c}$等等.

向量是有大小和方向的量，因此确定一个向量就需要确定其大小和方向. 如果两向量$\vec{a}$, $\vec{b}$大小相等, 方向相同, 就称它们是相等的, 记作$\vec{a}=\vec{b}$.

模长为1的向量称为单位向量; 不难看出有无限多个方向不同的单位向量.

模长为0的向量称为零向量，记作$\vec{0}$. 在不致引起混淆时也可记作0. 零向量没有确定的方向，它是唯一一个没有固定方向的向量;

与$\vec{a}$大小相等、方向相反的向量称为$\vec{a}$的逆向量或负向量,反向量, 记作$-\vec{a}$。由向量$\overrightarrow{AB}$的定义, 它的逆向量$-\overrightarrow{AB}=\overrightarrow{BA}$.

根据向量的定义, 平行移动一个向量后, 其大小和方向是没有变化的, 因此移动前后的向量就是相等的向量. 这样的向量可以认为只是任意移动其起点后得到的. 正是由于向量的这一特性, 向量可以称为自由向量.

两个向量之间可以有垂直的关系, 如果两向量$\vec{a}$, $\vec{b}$相互垂直, 就记作$\vec{a}\perp\vec{b}$或$\vec{b}\perp\vec{a}$.

如果两向量$\vec{a}$, $\vec{b}$相互平行, 就记作$\vec{a}\parallel\vec{b}$或$\vec{b}\parallel\vec{a}$. 多于两个向量可两两之间都是相互平行的. 对相互平行的向量, 如果把它们的起点移动到同一点, 则它们都位于同一条直线上, 因此相互平行的向量也称为是共线的向量.

如果把一组向量的起点移动到同一点后, 它们都位于同一个平面上, 则称它们是共面的向量.

向量一般是自由向量, 即平行移动一个向量后, 其大小和方向是没有变化的, 因此仍然是相等的向量. 但根据实际处理问题的不同, 我们需要对处理的向量限制在一定的范围内, 即加上适当的限制条件, 得到有限制条件的的一些向量,常见

的限制条件有如下两种: 起点可以沿指定的直线移动的向量叫做滑动向量；起点固定的向量叫做固定向量.

因此,有时我们也可以说向量可分为自由向量、滑动向量和固定向量三种类型.

对于不同类型的向量，向量相等的概念也有所不同：对于自由向量，只要大小相等, 方向相同, 两个向量即为相等的向量，而对于两个相等的滑动向量它们还必须同一条直线上; 对于两个相等的固定向量它们还必须有相同的起点。

滑动向量和固定向量都可以归入自由向量来研究，因此向量代数只研究自由向量。

下面我们先介绍向量的运算方法及其性质.

5.2 向量的线性运算

向量的线性运算包括向量的加法和减法, 向量的数乘运算, 下面分别予以介绍.

5.2.1 向量的加法

已知向量$\vec{a}$与$\vec{b}$，则它们的和仍然为向量，并且由下面的三角形法则来确定：

(1). 作向量$\overrightarrow{AB}=\vec{a}$, 起点为$A$, 终点为$B$;

(2). 以B为起点作向量$BC=\vec{b}$, 终点为C;

(3). 连接A, C, 得向量$\overrightarrow{AC}$, 则$\overrightarrow{AC}$就是$\vec{a}$ 与$\vec{b}$的和，记作$\vec{a}+\vec{b}$(如图5.1所示)即有

$$\vec{a}+\vec{b}=\overrightarrow{AC} \tag{5.2.1}$$

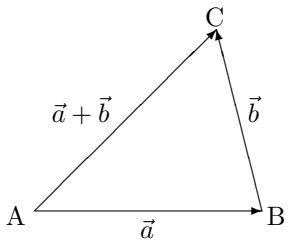


图5.1 向量加法

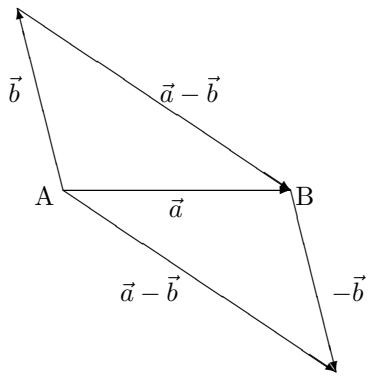


图5.2 向量减法

5.2.2 向量的减法

已知向量$\vec{a}$与$\vec{b}$，则它们的差仍然为向量，记作$\vec{a}-\vec{b}$，定义为

$$\vec{a}-\vec{b}=\vec{a}+(-\vec{b}) \tag{5.2.2}$$

其结果如图5.2的下半部分所示。

向量的差也可由图5.2所示的画图方法直观确定。

按同一尺度画出$\vec{a}$与$\vec{b}$，并使两者的起点重合，从$\vec{b}$的箭头出发指向$\vec{a}$的箭头的有向线段即为$\vec{a}$与$\vec{b}$的差，记作$\vec{a}+(-\vec{b})$ (如图5.2的上半部分所示)。

5.2.3 向量的数乘

已知一纯量(普通数，标量)λ及向量$\vec{a}$,则λ与$\vec{a}$的乘积为一向量, 记作$\lambda\vec{a}$, 它由如下方式确定:其大小$|\lambda\vec{a}| = |\lambda||\vec{a}|$; 其方向在$\lambda > 0$时与向量$\vec{a}$相同, $\lambda < 0$时与向量$\vec{a}$相反. 显然, $\lambda = 0$时,向量$\lambda\vec{a} = 0$为零向量,没有固定的方向.

特别地，在$|\vec{a}| \neq 0$时，如果取$\lambda = 1/|\vec{a}|$, 则$\lambda\vec{a}$就成为与$\vec{a}$同向的单位向量, 也可称之为$\vec{a}$的单位向量。

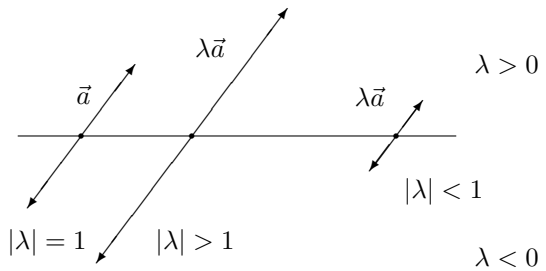


图5.3 向量的数乘

向量$\vec{a}$乘以纯数λ的结果$\lambda\vec{a}$与$\vec{a}$平行。两向量的的长度之比为$|\lambda|$。

5.2.4 向量线性运算的运算规律

设$\vec{a}$, $\vec{b}$和$\vec{c}$是向量，λ, μ 是数，则有：

5.2.4.1. 交换律

$$\begin{cases} \vec{a} + \vec{b} = \vec{b} + \vec{a} \\ \vec{a} - \vec{b} = -\vec{b} + \vec{a} \end{cases} \tag{5.2.3}$$

对于数乘运算,通常的记法总是数在前,向量在后; 但若有必要也可相反,即向量在前,数在后.

5.2.4.2. 结合律

$$\begin{cases} \vec{a} + (\vec{b} + \vec{c}) = (\vec{a} + \vec{b}) + \vec{c} \\ \lambda(\mu\vec{a}) = \mu(\lambda\vec{a}) = (\lambda\mu)\vec{a} \end{cases} \tag{5.2.4}$$

5.2.4.3. 分配律

$$\begin{cases} (\lambda \pm \mu)\vec{a} = \lambda\vec{a} \pm \mu\vec{a} \\ \lambda(\vec{a} \pm \vec{b}) = \lambda\vec{a} \pm \lambda\vec{b} \end{cases} \tag{5.2.5}$$

5.2.4.4. 其它常用的性质

$$\begin{cases} \vec{a} + \vec{0} = \vec{a} \\ \vec{a} + (-\vec{a}) = \vec{0} \\ (1)\vec{a} = \vec{a} \\ (-1)\vec{a} = -\vec{a} \\ 0\vec{a} = \vec{0} \end{cases} \tag{5.2.6}$$

5.2.4.5. 运算规律的证明
向量加法的结合律可以通过图5.4得到证明. 如图所示,

$$\overrightarrow{AD}=\overrightarrow{AB}+\overrightarrow{BD}=(\vec{a}+\vec{b})+\vec{c} \tag{5.2.7}$$

同时有

$$\overrightarrow{AD}=\overrightarrow{AC}+\overrightarrow{CD}=\vec{a}+(\vec{b}+\vec{c}) \tag{5.2.8}$$

于是有向量加法的结合律.

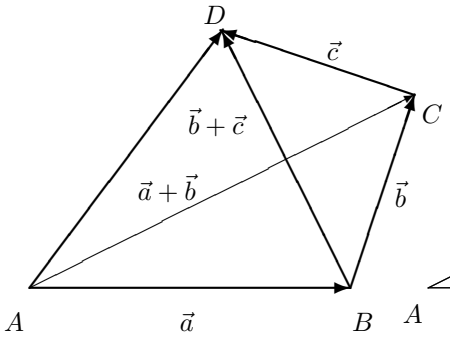


图5.4 向量加法结合律

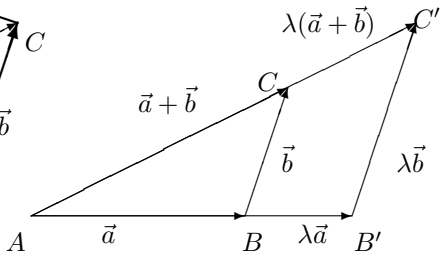


图5.5 向量加法分配律

关于分配律,如图5.5所示,因为

$$\begin{cases} \overrightarrow{AC}=\overrightarrow{AB}+\overrightarrow{BC}=\vec{a}+\vec{b} \\ \overrightarrow{AC'}=\overrightarrow{AB'}+\overrightarrow{B'C'}=\lambda\vec{a}+\lambda\vec{b} \\ \overrightarrow{AC'}=\lambda\ \overrightarrow{AC} \end{cases} \tag{5.2.9}$$

所以有

$$\lambda(\vec{a}+\vec{b})=\lambda\ \overrightarrow{AC}=\overrightarrow{AC'}=\lambda\vec{a}+\lambda\vec{b}. \tag{5.2.10}$$

其他的运算规则可以通过向量的定义，及比较相关向量的长度和方向直接加以验证。

5.3 向量的数量积及向量积

从上面所述可以看到，通过对向量的加、减和数乘运算可以描述空间中的单独的点和直线。但如果要研究两条直线之间的夹角、两条直线之间的最短距离、平面方程以及点在已知平面上的投影等问题，研究线段的长度, 平面区域的面积, 立方体的体积等等, 那就要研究具有三角关系的向量运算了，这就需要本节将要介绍的关于向量的数量积和向量积运算。

5.3.1 向量的数量积

已知向量$\vec{a}$与$\vec{b}$，则它们的数量积是一数量，记作$\vec{a}\cdot\vec{b}$, 也可简记为$\vec{a}\vec{b}$, 定义为两向量长度及其夹角余弦的乘积,即

$$\vec{a}\cdot\vec{b}=|\vec{a}||\vec{b}|\cos<\vec{a},\vec{b}> \tag{5.3.1}$$

其中$0 \leq < \vec{a}, \vec{b} > \leq \pi$为向量$\vec{a}$ 和$\vec{b}$之间的夹角.

显然, $\vec{a} \cdot \vec{b} = 0$等价于如下三个式子之一成立:

$$\begin{cases} |\vec{a}| = 0 \\ |\vec{b}| = 0 \\ \cos < \vec{a}, \vec{b} >= 0 \end{cases} \tag{5.3.2}$$

第三个等式成立说明$\vec{a} \perp \vec{b}$, 前两个等式成立说明两向量$\vec{a}$, $\vec{b}$至少有一个零向量. 而零向量方向不确定, 如果我们规定零向量与任何向量垂直, 则立即有结论: $\vec{a}\vec{b} = 0$等价于$\vec{a} \perp \vec{b}$. 当然也可以说$\vec{a} \perp \vec{b}$等价于$\vec{a}\vec{b} = 0$.

另外, 注意$|\vec{b}| \cos < \vec{a}, \vec{b} >$的几何意义, 它是向量$\vec{b}$在向量$\vec{a}$上的投影, 因此可以说$\vec{a}\vec{b}$是向量$\vec{a}$的长度与向量$\vec{b}$在向量$\vec{a}$上投影的乘积.

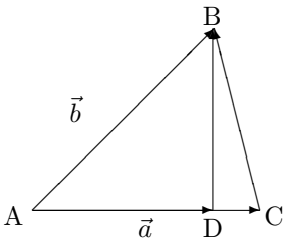


图5.6 向量数量积与投影的关系

由$\vec{a} \cdot \vec{a} = |\vec{a}||\vec{a}| \cos < \vec{a}, \vec{a} >= |\vec{a}||\vec{a}| = |\vec{a}|^2$可知向量$\vec{a}$ 的长度

$$|\vec{a}| = \sqrt{\vec{a} \cdot \vec{a}}. \tag{5.3.3}$$

由数量积的定义可得

$$\cos < \vec{a}, \vec{b} >= \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \frac{\vec{a}\vec{b}}{\sqrt{\vec{a}\vec{a}}\sqrt{\vec{b}\vec{b}}}. \tag{5.3.4}$$

这也是个重要的关系式,在我们后面可以直接确定数量积$\vec{a} \cdot \vec{b}$时, 这个式子就是有意义的了.

5.3.2 向量的数量积运算规律

设$\vec{a}$, $\vec{b}$和$\vec{c}$是向量，λ 是数，则向量的数量积满足如下规律:

5.3.2.1. 交换律

$$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$$

5.3.2.2. 分配律

$$\vec{a} \cdot (\vec{b} \pm \vec{c}) = \vec{a} \cdot \vec{b} \pm \vec{a} \cdot \vec{c}$$

5.3.2.3. 结合律

$$\lambda\vec{a} \cdot \vec{b} = (\lambda\vec{a}) \cdot \vec{b} = \vec{a}(\lambda \cdot \vec{b})$$

但是,一般地有

$$\vec{a} \cdot (\vec{b} \cdot \vec{c}) \neq (\vec{a} \cdot \vec{b}) \cdot \vec{c},$$

因为两端是分别与$\vec{a}$、$\vec{c}$平行的向量.

关于数量积的分配律,如图5.7所示有

$$\begin{cases} \vec{a}\cdot\vec{b} = |\vec{a}||OB'| \\ \vec{a}\cdot\vec{c} = |\vec{a}||OC'| \\ \vec{a}\cdot(\vec{b}+\vec{c}) = |\vec{a}||OD'||OC'| = |BE| = ||B'D'| \end{cases} \tag{5.3.5}$$

于是可得

$$|OD'| = |OB'| + |B'D'| = |OB'| + |O'C'|$$

$$\begin{cases} \vec{a}\cdot(\vec{b}+\vec{c}) = |\vec{a}||OD'| \\ = |\vec{a}|(|OB'| + |O'C'|) = |\vec{a}||OB'| + |\vec{a}||O'C'| \\ = \vec{a}\cdot\vec{b} + \vec{a}\cdot\vec{c} \end{cases} \tag{5.3.6}$$

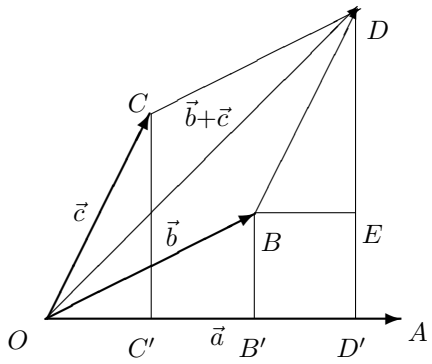


图5.7 数量积的分配律

5.3.3 向量的向量积

已知向量$\vec{a}$与$\vec{b}$，则它们的向量积是一向量，记作$\vec{a}\times\vec{b}$(不可简记为$\vec{a}\vec{b}$), 则定义:

(1). 这个向量的长度为

$$|\vec{a}\times\vec{b}| = |\vec{a}||\vec{b}|\sin<\vec{a},\vec{b}> \tag{5.3.7}$$

其中$0 \le <\vec{a},\vec{b}> \le \pi$为向量$\vec{a}$ 和$\vec{b}$之间的夹角;

(2). 这个向量的方向为$\vec{a}\times\vec{b}\perp\vec{a}$, $\vec{a}\times\vec{b}\perp\vec{b}$, 且$\vec{a}$, $\vec{b}$和$\vec{a}\times\vec{b}$构成右手系. $\vec{a}$, $\vec{b}$和$\vec{a}\times\vec{b}$构成右手系的具体含义是这样的: 伸开右手, 使大拇指与四指垂直并保持在一个平面内, 四指指向$\vec{a}$的方向. 然后旋转四指到$\vec{b}$的方向, 如果能保持大拇指的方向不变, 则大拇指所指的方向就是$\vec{a}\times\vec{b}$的方向. 条件$\vec{a}\times\vec{b}\perp\vec{a}$, $\vec{a}\times\vec{b}\perp\vec{b}$实际上是说向量$\vec{a}\times\vec{b}$ 与两向量$\vec{a}$和$\vec{b}$所确定的平面垂直。

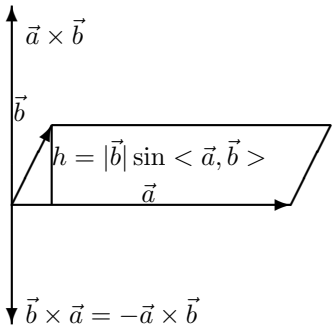


图5.8 向量的向量积

显然, $\vec{a}\times\vec{b}=0$也就是$|\vec{a}\times\vec{b}|=0$, 等价于如下三个式子之一成立:

$$\begin{cases} |\vec{a}|=0 \\ |\vec{b}|=0 \\ \sin<\vec{a},\vec{b}>=0 \end{cases} \tag{5.3.8}$$

第三个等式成立说明$\vec{a}\parallel\vec{b}$, 前两个等式成立说明量向量$\vec{a}$, $\vec{b}$至少有一个零向量. 而零向量方向不确定, 如果我们规定零向量与任何向量平行, 则立即有结论: $\vec{a}\times\vec{b}=0$等价于$\vec{a}\parallel\vec{b}$. 当然也可以说$\vec{a}\parallel\vec{b}$等价于$\vec{a}\times\vec{b}=0$. 因此总有

$$\vec{a}\times\vec{a}=0$$

另外, 注意$|\vec{b}|\sin<\vec{a},\vec{b}>$的几何意义, 它是向量$\vec{a}$, $\vec{b}$所张成的平行四边形在向量$\vec{a}$给出的底边上的高h, 因此可以说$|\vec{a}\times\vec{b}|$是向量$\vec{a}$和$\vec{b}$所张成的平行四边形的面积. 也是向量$\vec{a}$和$\vec{b}$所张成的三角形的面积的两倍.

5.3.4 向量的向量积运算规律

设已知$\vec{a}$, $\vec{b}$和$\vec{c}$是向量，λ 是数，则向量的向量积满足如下规律:

5.3.4.1. 交换律不成立:

$$\vec{a}\times\vec{b}=-\vec{b}\times\vec{a}$$

5.3.4.2. 分配律

$$\vec{a}\times(\vec{b}\pm\vec{c})=\vec{a}\times\vec{b}\pm\vec{a}\times\vec{c}$$

$$(\vec{b}\pm\vec{c})\times\vec{a}=\vec{b}\times\vec{a}\pm\vec{c}\times\vec{a}$$

5.3.4.3. 与数乘运算的结合律

$$\lambda(\vec{a}\times\vec{b})=(\lambda\vec{a})\times\vec{b}=\vec{a}(\lambda\times\vec{b})$$

但是,

$$\vec{a}\times(\vec{b}\times\vec{c})\neq(\vec{a}\times\vec{b})\times\vec{c}$$

与数量积运算的结合运算就是我们下面要介绍的混合积.

上述性质中分配律一般性的几何证明较长,有兴趣可参考有关向量代数的书籍. 特殊情况下, 如果三向量$\vec{a}$, $\vec{b}$与$\vec{c}$共面,这时等式两端向量共线,只需要比较其大小,这时的证明参考图5.7即可.

5.4 三个向量的二重乘积

5.4.1 向量的混合积

已知向量$\vec{a}$, $\vec{b}$与$\vec{c}$，则它们的混合积是一数量，记作$(\vec{a},\vec{b},\vec{c})$(不可简记为$\vec{a}\vec{b}\vec{c}$), 则定义

$$(\vec{a},\vec{b},\vec{c}) = (\vec{a}\times\vec{b})\cdot\vec{c}. \tag{5.4.1}$$

由向量的数量积的定义, 有

$$(\vec{a}\times\vec{b})\cdot\vec{c} = |\vec{a}\times\vec{b}||\vec{c}|\cos<(\vec{a}\times\vec{b}),\vec{c}> \tag{5.4.2}$$

如图5.8所示, 注意到三个向量$\vec{a}$, $\vec{b}$与$\vec{c}$的起点移动到同一点后可构成一个空间中的平行六面体, 这个平行六面体的体积等于它的一个底面的面积乘以该底面上的高. 考察$\vec{a}$, $\vec{b}$所张成的底面, 其面积为$|\vec{a}\times\vec{b}|$. 该底面上的高应该是$|\vec{c}|$与该底面和向量$\vec{c}$夹角的正弦的乘积, 因为该底面与向量$\vec{a}\times\vec{b}$垂直, 该底面与向量$\vec{c}$夹角和夹角$<\vec{a}\times\vec{b},\vec{c}>$互为补角, 因此该底面上的高$h$也等于$|\vec{c}|\cos<(\vec{a}\times\vec{b}),\vec{c}>$的绝对值. 于是, 混合积的绝对值就是给出混合积的三个向量所张成的平行六面体的体积. 而混合积的正负就取决于$\cos<(\vec{a}\times\vec{b}),\vec{c}>$的正负, 这等价于$\cos<(\vec{a}\times\vec{b}),\vec{c}>$是否不超过$\frac{\pi}{2}$, 或者说两向量$\vec{a}\times\vec{b}$和$\vec{c}$是否指向两向量$\vec{a}$和$\vec{b}$所确定的平面的同一侧. 这个关系我们也把它定义为三向量$\vec{a}$, $\vec{b}$, $\vec{c}$是否构成右手系关系.

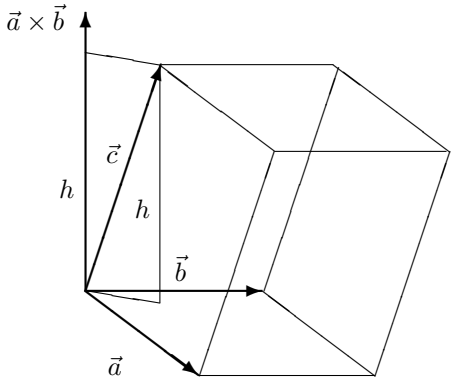


图5.9 三个向量混合的几何意义

根据如上对混合积的几何解释, 不难看出$(\vec{a},\vec{b},\vec{c}) = 0$ 等价于$\vec{a}$, $\vec{b}$, $\vec{c}$三向量共面, 或者说, $\vec{a}$, $\vec{b}$, $\vec{c}$三向量共面的充要条件是其混合积$(\vec{a},\vec{b},\vec{c}) = 0$.

三向量$\vec{a}$, $\vec{b}$和$\vec{c}$的混合积满足如下规律:

5.4.1.1. $(\vec{a},\vec{b},\vec{c}) = (\vec{b},\vec{c},\vec{a}) = (\vec{c},\vec{a},\vec{b})$.

根据如上对混合积的几何解释, 三个混合积确定的是同一个平行六面体的体积, 因此其绝对值相等. 另外, 按如上方式轮换排列三个向量不改变它们之间是否满足右手系关系.

由三个向量$\vec{a}$, $\vec{b}$与$\vec{c}$混合两种向量乘法可定义两个乘积$(\vec{a}\times\vec{b})\cdot\vec{c}$与$\vec{a}\cdot(\vec{b}\times\vec{c})$, 我们有

5.4.1.2. $(\vec{a}\times\vec{b})\cdot\vec{c} = \vec{a}\cdot(\vec{b}\times\vec{c})$.

这是因为$(\vec{a}\times\vec{b})\cdot\vec{c} = (\vec{a},\vec{b},\vec{c}) = (\vec{b},\vec{c},\vec{a}) = (\vec{b}\times\vec{c})\cdot\vec{a} = \vec{a}\cdot(\vec{b}\times\vec{c})$.

5.4.1.3. 拉格朗日等式

作为补记, 我们给出向量的数量积与向量积之间的拉格朗日等式如下.

$$(\vec{a}\times\vec{b})\cdot(\vec{c}\times\vec{d})=(\vec{a}\cdot\vec{c})(\vec{b}\cdot\vec{d})-(\vec{a}\cdot\vec{d})(\vec{b}\cdot\vec{c}) \tag{5.4.3}$$

作为这一等式的特例, 有

$$(\vec{a}\times\vec{b})^2=\vec{a}^2\vec{b}^2-(\vec{a}\cdot\vec{b})^2 \tag{5.4.4}$$

5.4.2 向量的二重向量积

对三向量$\vec{a}$，$\vec{b}$和$\vec{c}$, 可定义两种二重向量乘积, 前面已经提到, 一般地有

$$\vec{a}\times(\vec{b}\times\vec{c})\neq(\vec{a}\times\vec{b})\times\vec{c}$$

显然, 最终的结果都是向量. 我们先看它们的方向: $\vec{a}\times(\vec{b}\times\vec{c})\perp\vec{a}$, $\vec{a}\times(\vec{b}\times\vec{c})\perp\vec{b}\times\vec{c}$. 因为同时有$\vec{b}\perp\vec{b}\times\vec{c}$, $\vec{c}\perp\vec{b}\times\vec{c}$, 三个向量$\vec{a}\times(\vec{b}\times\vec{c})$, $\vec{b}$, $\vec{c}$共面, 或者说向量$\vec{a}\times(\vec{b}\times\vec{c})$, 在向量$\vec{b}$, $\vec{c}$所确定的平面上. 于是, $\vec{a}\times(\vec{b}\times\vec{c})$是在向量$\vec{b}$, $\vec{c}$所确定的平面上与$\vec{a}$垂直的一个向量.

类似地, $(\vec{a}\times\vec{b})\times\vec{c}$是在向量$\vec{a}$, $\vec{b}$所确定的平面上与$\vec{c}$垂直的一个向量.

借助于后面的向量的坐标表示, 我们可以得到

$$\vec{a}\times(\vec{b}\times\vec{c})=(\vec{a}\cdot\vec{c})\vec{b}-(\vec{a}\cdot\vec{b})\times\vec{c} \tag{5.4.5}$$

$$(\vec{a}\times\vec{b})\times\vec{c}=(\vec{a}\cdot\vec{c})\vec{b}-(\vec{b}\cdot\vec{c})\times\vec{a} \tag{5.4.6}$$

5.5 向量的坐标表示及其运算

5.5.1 向量的坐标表示

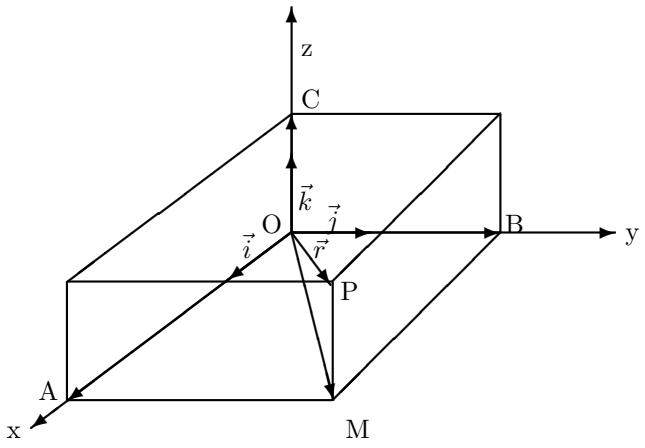


图5.10 向量的坐标表示

以直角坐标系$Oxyz$的原点O为起点, 作出已知向量$\vec{a}$, 其终点为$P(x,y,z)$。作一长方体,它以$\overrightarrow{OP}$为对角线，棱边分别与三条坐标轴平行(见图5.10) ，

与$x-$轴、$y-$轴和$z-$轴平行的棱边依次用向量$\overrightarrow{OA}$, $\overrightarrow{OB}$和$\overrightarrow{OC}$来表示，它们的长度分别为x、y和z。

由向量加法公式得:

$$\overrightarrow{OM}=\overrightarrow{OA}+\overrightarrow{AM}=\overrightarrow{OA}+\overrightarrow{OB} \tag{5.5.1}$$

$$\vec{r}=\overrightarrow{OP}=\overrightarrow{OM}+\overrightarrow{MP}=\overrightarrow{OA}+\overrightarrow{OB}+\overrightarrow{OC} \tag{5.5.2}$$

定义三个单位向量$\vec{i}$, $\vec{j}$和$\vec{k}$，它们的方向分别与$x-$轴，$y-$轴和$z-$轴的正方向相同,于是有

$$\overrightarrow{OA}=x\vec{i},\ \ \overrightarrow{OB}=y\vec{j},\ \ \overrightarrow{OC}=z\vec{k}. \tag{5.5.3}$$

因此向量$\vec{r}$就可以表示为：

$$\vec{r}=x\vec{i}+y\vec{j}+z\vec{k} \tag{5.5.4}$$

于是, 可以认为向量$\vec{r}$有坐标(x,y,z). 反之, 由坐标(x,y,z) 也可以给出一点, 得到从坐标系原点到此点的向量. 向量$\vec{r}$说与坐标(x,y,z)是相互唯一确定的.

$\vec{r}$有坐标(x,y,z)的具体含义是

$$\vec{r}=x\vec{i}+y\vec{j}+z\vec{k} \tag{5.5.5}$$

而x, y, z可分别称为该向量在各个坐标轴上的分量. 此时也可简单地记作$\vec{r}=(x,y,z)$.

5.5.2 向量的坐标运算

设向量$\vec{a}$, $\vec{b}$, $\vec{c}$分别有坐标(x_i,y_i,z_i), $i=1,2,3$, λ, μ都是实数, 则不难得出由坐标表示的向量的运算公式.

5.5.2.1. 向量的加减法运算的坐标表示公式

$$\vec{a}\pm\vec{b}=(x_1\vec{i}+y_1\vec{j}+z_1\vec{k})\pm(x_2\vec{i}+y_2\vec{j}+z_2\vec{k})=(x_1\pm x_2)\vec{i}+(y_1\pm y_2)\vec{j}+(z_1\pm z_2)\vec{k}$$

即有坐标$(x_1\pm x_2,y_1\pm y_2,z_1\pm z_2)$. 用坐标表示为

$$(x_1,y_1,z_1)\pm(x_2,y_2,z_2)=(x_1\pm x_2,y_1\pm y_2,z_1\pm z_2) \tag{5.5.6}$$

5.5.2.2. 向量的数乘运算的坐标表示公式

$$\lambda\vec{a}=\lambda(x_1\vec{i}+y_1\vec{j}+z_1\vec{k})=(\lambda x_1)\vec{i}+(\lambda y_1)\vec{j}+(\lambda z_1)\vec{k}$$

即有坐标$(\lambda x_1,\lambda y_1,\lambda z_1)$. 用坐标表示为

$$\lambda(x_1,y_1,z_1)=(\lambda x_1,\lambda y_1,\lambda z_1) \tag{5.5.7}$$

5.5.2.3. 向量的数量积运算的坐标表示公式

注意到$\vec{i}$,$\vec{j}$,$\vec{k}$是两两相互垂直的单位向量, 则有$|\vec{i}|=|\vec{j}|=|\vec{k}|=1$, $\vec{i}\cdot\vec{j}=\vec{i}\cdot\vec{k}=\vec{j}\cdot\vec{k}=0$, 于是

$$\begin{aligned}\vec{a}\cdot\vec{b}\ &=(x_1\vec{i}+y_1\vec{j}+z_1\vec{k})\cdot(x_2\vec{i}+y_2\vec{j}+z_2\vec{k})\\&=x_1x_2\vec{i}\cdot\vec{i}+x_1y_2\vec{i}\cdot\vec{j}+x_1z_2\vec{i}\cdot\vec{k}\\&+y_1x_2\vec{j}\cdot\vec{i}+y_1y_2\vec{j}\cdot\vec{j}+y_1z_2\vec{j}\cdot\vec{k}\\&+z_1x_2\vec{k}\cdot\vec{i}+z_1y_2\vec{k}\cdot\vec{j}+z_1z_2\vec{k}\cdot\vec{k}\\&=x_1x_2+y_1y_2+z_1z_2\end{aligned}$$

即两向量的数量积等于两向量对应坐标分量乘积之和. 用坐标表示为

$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1x_2 + y_1y_2 + z_1z_2 \tag{5.5.8}$$

需要注意的是此性质依赖于$\vec{i}$, $\vec{j}$和$\vec{k}$为两两相互垂直的单位向量。

5.5.2.4. 向量的向量积运算的坐标表示公式

如果注意到$\vec{i}$, $\vec{j}$和$\vec{k}$构成右手系, 则有$\vec{i} \times \vec{i} = \vec{j} \times \vec{j} = \vec{k} \times \vec{k} = 0$, $\vec{i} \times \vec{j} = \vec{k}$, $\vec{j} \times \vec{k} = \vec{i}$, $\vec{k} \times \vec{i} = \vec{j}$, $\vec{j} \times \vec{i} = -\vec{k}$, $\vec{k} \times \vec{j} = -\vec{i}$, $\vec{i} \times \vec{k} = -\vec{j}$, 于是

$$\begin{aligned}
\vec{a} \times \vec{b} &= (x_1\vec{i} + y_1\vec{j} + z_1\vec{k}) \times (x_2\vec{i} + y_2\vec{j} + z_2\vec{k}) \\
&= x_1x_2\vec{i} \times \vec{i} + x_1y_2\vec{i} \times \vec{j} + x_1z_2\vec{i} \times \vec{k} \\
&+y_1x_2\vec{j} \times \vec{i} + y_1y_2\vec{j} \times \vec{j} + y_1z_2\vec{j} \times \vec{k} \\
&+z_1x_2\vec{k} \times \vec{i} + z_1y_2\vec{k} \times \vec{j} + z_1z_2\vec{k} \times \vec{k} \\
&= (y_1z_2 - y_2z_1)\vec{i} + (z_1x_2 - z_2x_1)\vec{j} + (x_1y_2 - x_2y_1)\vec{k} \\
&= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}
\end{aligned}$$

用坐标表示为

$$(x_1, y_1, z_1) \times (x_2, y_2, z_2) = \left(\begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix}, \begin{vmatrix} z_1 & x_1 \\ z_2 & x_2 \end{vmatrix}, \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} \right) \tag{5.5.9}$$

需要注意的是此性质依赖于$\vec{i}$, $\vec{j}$和$\vec{k}$构成右手系, 且为两两相互垂直的单位向量。

5.5.2.5. 向量的混合积运算的坐标表示公式

根据向量积的坐标运算公式,向量$\vec{a}\times\vec{b}$的坐标为$(y_1z_2-y_2z_1, z_1x_2-z_2x_1, x_1y_2-x_2y_1)$. 由此可得混合积的坐标运算结果为

$$\begin{aligned}
(\vec{a}, \vec{b}, \vec{c}) &= (\vec{a} \times \vec{b}) \cdot \vec{c} \\
&= (y_1z_2 - y_2z_1)x_3 + (z_1x_2 - z_2x_1)y_3 + (x_1y_2 - x_2y_1)z_3 \\
&= \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}
\end{aligned}$$

5.5.2.6. 向量的二重向量积运算的坐标表示公式

根据向量积的坐标运算公式,二重向量积的坐标运算结果为

$$\begin{aligned}
&(\vec{a} \times \vec{b}) \times \vec{c} \\
&= ((y_1z_2 - y_2z_1)\vec{i} + (z_1x_2 - z_2x_1)\vec{j} + (x_1y_2 - x_2y_1)\vec{k}) \times \vec{c} \\
&= ((z_1x_2 - z_2x_1)z_3 - (x_1y_2 - x_2y_1)y_3)\vec{i} \\
&\ +((x_1y_2 - x_2y_1)x_3 - (y_1z_2 - y_2z_1)z_3)\vec{j} \\
&\ +((y_1z_2 - y_2z_1)y_3 - (z_1x_2 - z_2x_1)x_3)\vec{k} \\
&= (z_1x_2z_3 + x_2y_1y_3 - (z_2x_1z_3 + x_1y_2y_3))\vec{i} \\
&\ +(x_1y_2x_3 + y_2z_1z_3 - (x_2y_1x_3 + y_1z_2z_3))\vec{j} \\
&\ +(y_1z_2y_3 + z_2x_1x_3 - (y_2z_1y_3 + z_1x_2x_3))\vec{k} \\
&= (x_2(x_1x_3 + y_1y_3 + z_1z_3) - x_1(x_2x_3 + y_2y_3 + z_2z_3))\vec{i} \\
&+(y_2(x_1x_3 + y_1y_3 + z_1z_3) - y_1(x_2x_3 + y_2y_3 + z_2z_3))\vec{j} \\
&+(z_2(x_1x_3 + y_1y_3 + z_1z_3) - z_1(x_2x_3 + y_2y_3 + z_2z_3))\vec{k} \\
&= (x_2(\vec{a} \cdot \vec{c}) - x_1(\vec{b} \cdot \vec{c}))\vec{i} + (y_2(\vec{a} \cdot \vec{c}) - y_1(\vec{b} \cdot \vec{c}))\vec{j} + (z_2(\vec{a} \cdot \vec{c}) - z_1(\vec{b} \cdot \vec{c}))\vec{k} \\
&= (\vec{a} \cdot \vec{c})\vec{b} - (\vec{b} \cdot \vec{c}))\vec{a}
\end{aligned}$$

由此不难推出

$$\vec{a}\times(\vec{b}\times\vec{c})=-(\vec{b}\times\vec{c})\times\vec{a}=(\vec{a}\cdot\vec{c})\vec{b}-(\vec{a}\cdot\vec{b})\vec{c}$$

5.6 常用几何量的向量表示

5.6.1. 若有两点$A=(x_1,y_1,z_1)$, $B=(x_2,y_2,z_2)$, 则两点之间的向量

$$\overrightarrow{AB}=(x_2-x_1,y_2-y_1,z_2-z_1) \tag{5.6.1}$$

5.6.2. 向量$\vec{a}=(x_1,y_1,z_1)$的长度为

$$|\vec{a}|=\sqrt{\vec{a}\cdot\vec{a}}=\sqrt{x_1^2+y_1^2+z_1^2}. \tag{5.6.2}$$

因此两向量$\vec{a}=(x_1,y_1,z_1)$和$\vec{b}=(x_2,y_2,z_2)$ 所张平行四边形的面积为

$$|\vec{a}\times\vec{b}|=\sqrt{\begin{vmatrix}y_1&z_1\\y_2&z_2\end{vmatrix}^2+\begin{vmatrix}z_1&x_1\\z_2&x_2\end{vmatrix}^2+\begin{vmatrix}x_1&y_1\\x_2&y_2\end{vmatrix}^2}, \tag{5.6.3}$$

所张三角形的面积为这个平行四边形的面积的一半。

5.6.3. 两向量$\vec{a}=(x_1,y_1,z_1)$和$\vec{b}=(x_2,y_2,z_2)$之间的夹角余弦则由数量积的定义可表示为

$$<\vec{a},\vec{b}>=\arccos\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}=\arccos\frac{x_1x_2+y_1y_2+z_1z_2}{\sqrt{x_1^2+y_1^2+z_1^2}\sqrt{x_2^2+y_2^2+z_2^2}}. \tag{5.6.4}$$

两向量相互垂直的充要条件是

$$x_1x_2+y_1y_2+z_1z_2=0. \tag{5.6.5}$$

两向量相互平行的充要条件是

$$\frac{x_1}{x_2}=\frac{y_1}{y_2}=\frac{z_1}{z_2}, \tag{5.6.6}$$

其中分母为零时，分子也为零，则此结论永远成立。

5.6.4. 三向量$\vec{a}=(x_1,y_1,z_1)$、$\vec{b}=(x_2,y_2,z_2)$和$\vec{c}=(x_3,y_3,z_3)$ 所张平行六面体的体积依混合积的意义为$\begin{vmatrix}x_1&y_1&z_1\\x_2&y_2&z_2\\x_3&y_3&z_3\end{vmatrix}$的绝对值。而三向量所张四面体的体积等于所张平行六面体的体积的六分之一。

5.6.5. 我们已经在5.6.1. 中描述了如何由两点定义向量. 一般地, 通过点给出的几何对象可以通过点之间构造向量, 然后通过向量的运算来描述. 例如, 两点间的距离可由两点间向量的长度求得.

三(四)点给出的三角形(四面体)的面（体）积可这样求得: 由其中一点与另外两（三）点之间构造两（三）个向量, 这两（三）个向量所张三角形(四面体)的面（体）积即为所求.

三(四)点是否共线（面）可这样求得: 由其中一点与另外两（三）点之间构造两（三）个向量, 判断这两（三）个向量所张三角形(四面体)的面（体）积是否为零即可. 当然，三点是否共线也可通过判断相应的两个向量是否平行即可。

利用前面的结论，不难写出这些几何量通过相应点坐标给出的数学表达式。

5.7 向量的微分运算

数量依赖于变量变化时产生了函数，通过对函数的微分我们对函数的性质有了更进一步的了解。同样地，向量依赖于变量变化时就产生出向量函数，通过对向量函数的微分我们对向量函数的性质也会有更进一步的了解。图形对象是由空间中的点构成的，即由向量表示的。空间中的曲线曲面一般表现为向量的函数。因此对向量函数的微分做一些简单的介绍。

设有向量$\overrightarrow{P}$是某个变量t的函数。这实际上可以认为$\overrightarrow{P}$有三个分量x, y, z, 并且这三个分量都是t的函数，因此可以表示为

$$P = P(t) = [x(t) \quad y(t) \quad z(t)] \tag{5.7.1}$$

假设$\overrightarrow{P}$的三个分量$x(t)$, $y(t)$, $z(t)$都对t是可导的，则定义

$$\frac{dP}{dt} = \left[\frac{dx(t)}{dt} \quad \frac{dy(t)}{dt} \quad \frac{dz(t)}{dt}\right] \tag{5.7.2}$$

依据这个定义，向量的导数就是两点所联向量与两点间参数值的改变量的比值。如果两点在曲线上，则向量的导数就是曲线上某点处的切向量。

类似于一般的函数，我们可以列出向量函数的到函数的一些性质。

(1). 向量函数导函数的线性不变性，即

$$\frac{d}{dt}(aP(t) + bQ(t)) = a\frac{dP(t)}{dt} + b\frac{dQ(t)}{dt} \tag{5.7.3}$$

(2). 向量函数与数量函数乘积的导函数为：

$$\frac{d}{dt}(\lambda(t)P(t)) = \frac{d\lambda(t)}{dt}P(t) + \lambda(t)\frac{dP(t)}{dt} \tag{5.7.4}$$

(3). 两向量函数数量乘积的导函数为：

$$\frac{d}{dt}(P(t)Q(t)) = \frac{dP(t)}{dt}Q(t) + P(t)\frac{dQ(t)}{dt} \tag{5.7.5}$$

(4). 两向量函数量乘积的导函数为：

$$\frac{d}{dt}(P(t) \times Q(t)) = \frac{dP(t)}{dt} \times Q(t) + P(t) \times \frac{dQ(t)}{dt} \tag{5.7.6}$$

要注意的是这个式子中乘积的顺序是不能改变的。

(4). 单位向量函的导函数与其本身垂直。

设有向量$\overrightarrow{N(t)}$是单位向量，则

$$\overrightarrow{N(t)} \cdot \overrightarrow{N(t)} = 1. \tag{5.7.7}$$

因此，

$$\frac{d}{dt}(\overrightarrow{N(t)} \cdot \overrightarrow{N(t)}) = 0, \tag{5.7.8}$$

依据(5.7.6)展开方程右端得

$$\frac{d}{dt}(\overrightarrow{N(t)}\cdot\overrightarrow{N(t)})=2\,\overrightarrow{N(t)}\cdot\frac{d\,\overrightarrow{N(t)}}{dt}=0, \tag{5.7.9}$$

即有

$$\overrightarrow{N(t)}\cdot\frac{d\,\overrightarrow{N(t)}}{dt}=0. \tag{5.7.10}$$

所以，$\overrightarrow{N(t)}$与$\frac{d\,\overrightarrow{N(t)}}{dt}$垂直。

需要说明的是在以后用到向量时，为了简单起见，我们会在意义比较明确时省略向量的箭头标记⇀。

5.8 矩阵的基本概念

一个$m\times n$矩阵是$m\times n$个量（数字函数或数值表达式）组成的长方形阵列表构成的。一般地．可将一个$m\times n$矩阵表示为：

$$A=A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n}\\ a_{21} & a_{22} & \cdots & a_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ a_{m1} & a_{m2} & \cdots & a_{mn}\end{bmatrix} \tag{5.8.1}$$

我们也可按行数和列数标识矩阵。因此一个$m\times n$矩阵称为一个m行n列的矩阵，并表示为(a_{ij})或$(a_{ij})_{m\times n}$。构成表的这些量为称该矩阵的元素。其中a_{ij}表示矩阵A的i行j列处元素，即其第一个下标指定行数．第二个下标指定列数。下面是矩阵的一些例子.

$$\begin{bmatrix} 3 & 4 & 1.5 & 1.5\\ 2.2 & 7 & 5 & 3.5\\ 0 & 6 & 1 & 1.5\end{bmatrix},\ \begin{bmatrix} \cos x & \sin x\\ -\sin x & \cos x\end{bmatrix},\ \begin{bmatrix} x & y & z\end{bmatrix},\ \begin{bmatrix} 1\\ 0\\ 0\end{bmatrix} \tag{5.8.2}$$

这些矩阵自左到右依次为3x4、2X2、1X3和3x1矩阵.

当行数与列数相同时就称为方阵. 如上述第2个矩阵为2X2矩阵,也成为2阶方阵.

单行或单列的矩阵与一个向量的表示相同。式(5.8.2)的最后两个矩阵分别为行向量和列向量。

矩阵$A=(a_{ij})_{m\times n}$的负矩阵为$-A=(-a_{ij})_{m\times n}$.

所有元素全为零的矩阵称为零矩阵。

矩阵$E_n=\begin{bmatrix} 1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}$ 称n为阶单位矩阵。

矩阵A=B表示两个矩阵的行列数完全一致，并且同行列处的对应元素都是相等的。

5.9 矩阵的运算

5.9.1 矩阵的线性运算

类似于向量的线性，矩阵的线性运算也是指矩阵的数乘和加、减法运算。

5.9.1.1 矩阵的数乘

矩阵的数乘指一个数λ与一个矩阵$(a_{ij})_{m\times n}$的乘法，定义为

$$\lambda(a_{ij})_{m\times n}=(\lambda a_{ij})_{m\times n} \tag{5.9.1}$$

即该数λ乘以每矩阵的每个元素a_{ij}。例如若.

$$10\begin{bmatrix} 3 & 4 & 1.5 & 1.5\\ 2.2 & 7 & 5 & 3.5\\ 0 & 6 & 1 & 1.5 \end{bmatrix}=\begin{bmatrix} 30 & 40 & 15 & 15\\ 22 & 70 & 50 & 35\\ 0 & 60 & 10 & 15 \end{bmatrix} \tag{5.9.2}$$

5.9.1.2 矩阵加、减法

矩阵加法只对有相同行数和相同列数的矩阵有定义。对于任意两个矩阵，其和由对应元素相加得到. 例如

$$\begin{bmatrix} 3 & 4 & 1.5 & 1.5\\ 2.2 & 7 & 5 & 3.5\\ 0 & 6 & 1 & 1.5 \end{bmatrix}+\begin{bmatrix} 0 & 6 & -3.5 & 5.5\\ 2.2 & 7 & 5.5 & 6.5\\ 0 & 9 & 2 & 8.5 \end{bmatrix}=\begin{bmatrix} 3 & 10 & -2 & 7\\ 4.4 & 14 & 11.5 & 10\\ 0 & 15 & 3 & 10 \end{bmatrix} \tag{5.9.3}$$

矩阵的减法可通过加法来定义：

$$A-B=A+(-B) \tag{5.9.4}$$

矩阵的线性运算也具有向量的线性运算完全类似的性质，这里不再一一列举。

5.9.2 矩阵乘法

我们可以把mxp矩阵A和qxn矩阵B相乘，得一mxn的乘积矩阵AB。注意两乘积因子的关系为A的列数等于B的行数。具体方法是乘积矩阵的元素由A的特定行的元素与B特定列的对应元素之积的和来得到，即有

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s}\\ \cdots & \cdots & \cdots & \cdots\\ \underbrace{a_{i1}} & \underbrace{a_{i2}} & \cdots & \underbrace{a_{is}}\\ \cdots & \cdots & \cdots & \cdots\\ a_{m1} & a_{m2} & \cdots & a_{ms} \end{bmatrix}_{m\times s}\begin{bmatrix} b_{11} & \cdots & \{b_{1j} & \cdots & b_{1n}\\ b_{21} & \cdots & \{b_{1j} & \cdots & b_{2n}\\ \vdots & \vdots & & \cdots & \vdots\\ b_{s1} & \cdots & \{b_{sj} & \cdots & b_{sn} \end{bmatrix}_{s\times n}$$

$$=\begin{bmatrix} \sum_{k=1}^{s}a_{1k}b_{k1} & \cdots & \sum_{k=1}^{s}a_{1k}b_{kj} & \cdots & \sum_{k=1}^{s}a_{1k}b_{kn}\\ \vdots & & \vdots & \vdots & \vdots\\ \sum_{k=1}^{s}a_{ik}b_{k1} & \cdots & \underbrace{\sum_{k=1}^{s}a_{ik}b_{kj}} & \cdots & \sum_{k=1}^{s}a_{ik}b_{kn}\\ \vdots & \cdots & \vdots & \vdots & \vdots\\ \sum_{k=1}^{s}a_{mk}b_{k1} & \cdots & \sum_{k=1}^{s}a_{mk}b_{kj} & \cdots & \sum_{k=1}^{s}a_{mk}b_{kn} \end{bmatrix}_{m\times n} \tag{5.9.5}$$

当第一个向量表达为行向量．而第二个向量表达为列向量时，以矩阵表示的向量乘法产生与点积同样的结果。

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = [ax + by + cz] \tag{5.9.6}$$

这个向最的矩阵乘积生成一个包含单个元素的1x1矩阵。

如果我们按相反顺序进行向量矩阵乘法则得到3x3矩阵．

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \begin{bmatrix} a & b & c \end{bmatrix} = \begin{bmatrix} ax & bx & cx \\ ay & by & cy \\ az & bz & cz \end{bmatrix} \tag{5.9.7}$$

这一结果表明矩阵乘法是不满足交换律的．即

$$AB \neq BA \tag{5.9.8}$$

但是，矩阵乘法关于矩阵加法是可以满足分配律的,即有:

$$A(B + C) = AB + AC, \quad (B + C)A = BA + CA. \tag{5.9.9}$$

同时矩阵乘法也满足结合律：

$$A(BC) = (AB)C. \tag{5.9.10}$$

我们仅给出矩阵乘法页满足结合律的证明如下：因为三个矩阵可以相乘，其行列数满足一定要求，故可设

$$A = (a_{ik})_{m\times p}, \quad B = (b_{kl})_{p\times q}, \quad C = (c_{lj})_{q\times n}. \tag{5.9.11}$$

依次假设$A(BC)$和$(AB)C$均为$m \times n$矩阵, 并且有

$$\begin{aligned} A(BC) &= (a_{ik})_{m\times p}((b_{kl})_{p\times q}(c_{lj})_{q\times n})_{p\times n} &&= (a_{ik})_{m\times p}(\sum_{l=1}^{q} b_{kl}c_{lj})_{p\times n} \\ &= (\sum_{k=1}^{p} a_{ik} \sum_{l=1}^{q} (b_{kl}c_{lj})_{m\times n}) &&= (\sum_{k=1}^{p}\sum_{l=1}^{q} a_{ik}b_{kl}c_{lj})_{m\times n} \\ &= (\sum_{l=1}^{q}\sum_{k=1}^{p} a_{ik}b_{kl}c_{lj})_{m\times n} &&= (\sum_{l=1}^{q}(\sum_{k=1}^{p} a_{ik}b_{kl})c_{lj})_{m\times n} \\ &= ((a_{ik})_{m\times p}(b_{kl})_{p\times q})_{m\times q}(c_{lj})_{q\times n} &&= (AB)C \end{aligned} \tag{5.9.12}$$

对于任意mxn矩阵A，我们有

$$E_m A_{m\times n} = A_{m\times n}, \quad A_{m\times n}E_n = A_{m\times n}. \tag{5.9.13}$$

矩阵乘法的直接解释之一是其来自于变换：若定义矩阵

$$\begin{cases} X = X_{m\times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}, Y = Y_{n\times 1} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}, \\ A = A_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \end{cases} \tag{5.9.14}$$

则矩阵X和Y的各个分量所代表的两组变量间的变换关系

$$\begin{cases} x_1 = a_{11}y_1 + a_{12}y_2 + \cdots + a_{1n}y_n \\ x_2 = a_{21}y_1 + a_{22}y_2 + \cdots + a_{2n}y_n \\ \cdots \\ x_m = a_{m1}y_1 + a_{m2}y_2 + \cdots + a_{mn}y_n \end{cases} \tag{5.9.15}$$

就可用矩阵乘法运算简洁地表示为

$$X = AY. \tag{5.9.16}$$

如对第三组变量同时有

$$Y_{m\times 1} = B_{m\times p}Z_{p\times 1}, \tag{5.9.17}$$

则第一组变量可由矩阵乘法的关系直接由第三组变量表示出来：

$$X = AY = A(BZ) = (AB)Z. \tag{5.9.18}$$

5.9.3 矩阵转置

一个矩阵A的转置由A的行和列的交换得来的另外一个矩阵，通常用A'表示。也有表示为A^T或A^τ的。例如,

$$\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times 3}\right)' = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{3\times 2}.$$

矩阵转置可以使为矩阵的一种新的运算，与矩阵的线性运算结合有如下等式：

$$(\lambda A + \beta B)' = \lambda A' + \beta B'. \tag{5.9.19}$$

与矩阵乘积相结合有：

$$(AB)' = B'A', \tag{5.9.20}$$

这里要特别注意等式两端乘积因子顺序的颠倒。

5.9.4 矩阵的行列式

对于方阵A，我们可以组合矩阵单元来生成一个数，称为方阵A的行列式,用$|A|$或$det(A)$表示。对于一、二、三阶方阵的行列式，有表达式：

$$\begin{cases} det\left(|a_{11}|\right) = a_{11}, \\ det\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\right) = a_{11}a_{22} - a_{21}a_{12}, \\ det\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\right) = \begin{array}{r} a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} \\ +a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} \\ -a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32} \end{array} \end{cases} \tag{5.9.21}$$

一般方阵的行列式是递归定义的。对于n阶方阵A，我们可以选择nXn矩阵的任意一列j_0，并按下式计算其行列式：

$$det(A) = \sum_{i=1}^{n} a_{ij_0}A_{ij_0} \tag{5.9.22}$$

其中，$A_{ij} = (-1)^{i+j} M_{ij}$称为$a_{ij}$的代数余子式，而$a_{ij}$的余子式$M_{ij}$是从$A$中删去第$i$行和第$j$列得到的$(n-1)$阶矩阵的行列式：

$$\begin{aligned} M_{ij} &= det\left(\begin{bmatrix} a_{11} & \cdots & a_{1j-1} & \not{a}_{1j} & a_{1j+1} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i-11} & \cdots & a_{i-1j-1} & \not{a}_{i-1j} & a_{i-1j+1} & \cdots & a_{i-1n} \\ \not{a}_{i1} & \cdots & \not{a}_{ij-1} & \not{a}_{ij} & \not{a}_{ij+1} & \cdots & \not{a}_{in} \\ a_{i+11} & \cdots & a_{i+1j-1} & \not{a}_{i+1j} & a_{i+1j+1} & \cdots & a_{i+1n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nj-1} & \not{a}_{nj} & a_{nj+1} & \cdots & a_{nn} \end{bmatrix}\right) \\ &= det\left(\begin{bmatrix} a_{11} & \cdots & a_{1j-1} & a_{1j+1} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i-11} & \cdots & a_{i-1j-1} & a_{i-1j+1} & \cdots & a_{i-1n} \\ a_{i+11} & \cdots & a_{i+1j-1} & a_{i+1j+1} & \cdots & a_{i+1n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nj-1} & a_{nj+1} & \cdots & a_{nn} \end{bmatrix}\right) \end{aligned} \tag{5.9.23}$$

当然，我们也可以选择任意一行i_0并有表达式：

$$det(A) = \sum_{j=1}^{n} a_{i_0 j} A_{i_0 j} \tag{5.9.24}$$

方阵的行列式有如下特性：

$$det(AB) = det(A)det(B). \tag{5.9.25}$$

5.9.5 方阵的逆

一个n阶方阵A的逆矩阵是满足

$$AB = BA = E_n \tag{5.9.26}$$

的矩阵B. 若有方阵A的逆矩阵B_1, B_2, 则

$$B_1 = B_1 E = B_1(AB_2) = (B_1 A)B_2 = EB_2 = B_2. \tag{5.9.27}$$

这说明A的逆矩阵如果存在，最多只有一个，记作A^{-1}。

如果逆矩阵存在，则该矩阵称为非奇异矩阵。否则，矩阵为奇异矩阵。方阵的逆矩阵当且仅当矩阵的行列式非零时存在。对于大多数实际应用，矩阵表示一个物理变换，我们可以期望反向的变换存在。这时逆矩阵就表示了反向变换的矩阵。逆矩阵A^{-1}的元素可以由A的元素计算出：

$$A^{-1} = \frac{1}{det(A)} \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \cdots & \cdots & \ddots & \cdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \end{bmatrix} \tag{5.9.28}$$

其中A_{1n}的为A中元素a_{1n}的代数余子式。需要注意的是它们的排列顺序：原来某行元素的代数余子式现在排在了相应的列上，原来某列元素的代数余子式现在排在了相应的行上。

一般方阵逆矩阵的计算需要专业的计算方法，但对于2阶方阵，上式给出的求逆矩阵的公式还是很简单的：

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}. \tag{5.9.29}$$

方阵经线性运算后再求逆与参加运算的各个方阵没有明显的关系. 但对于乘积和转置, 却有如下关系:

$$(AB)^{-1} = B^{-1}A^{-1}, \tag{5.9.30}$$

$$(A')^{-1} = (A^{-1})'. \tag{5.9.31}$$

5.9.6 矩阵的分块

一个完整的矩阵可以看成是有较小的矩阵构成的,而每一个较小的构成部分由于有明确的来源而具有相对独立的实际意义. 这是我们对各个较小的构成部分相对地单独看待会帮助我们更好的了解完整的矩阵的含义和特性. 比如给定几个行向量表示的点,用来构成一个矩阵. 矩阵就会是由点构成这样很清楚的几何意义.

另外, 在进行矩阵运算中常遇到零集中到某一处或某几处的情况，而这些零在运算中不起大的作用. 再一种情况是在某些实际问题中，会遇到高阶矩阵，即使用计算机去算有时也会受存储量的限制于是我们采取分块的办法，把零集中到某一块或某几块,方便矩阵运算处理.

把一个高阶矩阵按行和列适当分成若干块（称为子块），把每一个子块看作一个元素参加运算（当然，分法要使得矩阵的于块间的运算是有意义的）. 这种方法称为矩阵的分块以子块为元素的矩阵称为分块矩阵.

对于单独一个矩阵,可以在其任意两行或两列之间分开形成矩阵的分块. 对一个矩阵任意两列之间分开就可以看到矩阵是有一些列向量构成的. 一般的一个矩阵可以同时在某些行某些列之间分开. 例如,

$$\left[\begin{array}{cc|c|cc} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{array}\right] = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \tag{5.9.32}$$

涉及到不止一个矩阵是，各个矩阵的分块应保持一定的一致性。如在作加减法运算时要求各个矩阵的行与列的分法分法完全一致. 如

$$\begin{cases} & \left[\begin{array}{c|cc} a_{11} & 0 & 0 \\ \hline a_{21} & 0 & 0 \\ a_{31} & 0 & 0 \end{array}\right] + \left[\begin{array}{c|cc} b_{11} & 0 & 0 \\ \hline 0 & b_{24} & b_{25} \\ 0 & b_{34} & b_{35} \end{array}\right] \\ = & \left[\begin{array}{c|cc} a_{11}+b_{11} & a_{14} & a_{15} \\ \hline a_{21} & b_{24} & b_{25} \\ a_{31} & b_{34} & b_{35} \end{array}\right] \end{cases} \tag{5.9.33}$$

用分块矩阵可以较简单地表示为：

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & 0 \end{bmatrix} + \begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}+B_{11} & 0 \\ A_{21} & B_{22} \end{bmatrix} \tag{5.9.34}$$

在作分块矩阵乘法运算时每个矩阵的列分法须和后需矩阵的行分法一致，以保持前一矩阵中每个子块的列数与后一矩阵中对应相乘的子块的行数相同. 如

$$\left[\begin{array}{c|ccc} 0 & 2 & 0 & 0 \\ 0 & 2 & -1 & 1 \\ 0 & 3 & 1 & 0 \\ \hline 2 & 0 & 0 & 0 \end{array}\right]\left[\begin{array}{cc|c} 4 & 5 & 7 \\ \hline 2 & -1 & 0 \\ 3 & 6 & 0 \\ 0 & -1 & 0 \end{array}\right] = \left[\begin{array}{cc|c} 4 & -2 & 0 \\ 6 & 13 & 0 \\ 9 & 3 & 0 \\ \hline 8 & 10 & 14 \end{array}\right] \tag{5.9.35}$$

用分块矩阵可以较简单地表示为：

$$\begin{cases} \begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix}\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & 0 \end{bmatrix} = \begin{bmatrix} A_{12}B_{21} & 0 \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix} \\ = \left[\begin{array}{cc|c} 4 & -2 & 0 \\ 6 & 13 & 0 \\ 9 & 3 & 0 \\ \hline 8 & 10 & 14 \end{array}\right] \end{cases} \tag{5.9.36}$$

Chapter 6

图形变换

6.1 引言

利用计算机对图形进行处理当然是计算机图形学的最基本的任务. 这其中大量的任务就是控制和修改所显示的图形。作为这门学科在工业自动化生产中的CAD/CAM系统中，显示和输出基本图形和图形的属性，对设计的产品几何模型进行修改或重新设计，由零件、构件装配成产品，在屏幕上显示一幅接一幅连续显示的动画图面，对设计和制造过程进行动态模拟分析和仿真. 这些图形在计算机显示器上处理的过程很多情况下就表现为同一幅图形显示在显示器的不同的位置上. 另外, 图形表示的现实物体的几何图形本身可能有一个坐标系统, 当它被计算机处理时, 就要转换到计算机的设备上去, 设备本身也有一个坐标系统. 两个坐标系统之间的关系也表现为图形变换的关系。所有这些变换都是通过坐标点的几何变换实现的。这些基木变换包括:比例、平移、旋转，以及反射、错切和透视变换等。下面首先介绍变换的原理和方法，然后介绍变换的应用。

6.2 二维图形的基本变换

二维图形变换是指将点、线、面在屏幕上进行平移、比例、旋转、反射以及错切等有关几何位置、尺寸和形状的改变。一般情况下，图形是一个点集。因此，点的几何变换是图形变换的基础。当然, 我们并不是直接对构成图形的所有点直接进行变换. 一个二维图形可以由直线段连接而成，也可以看作是由许多直线段拟合而成，一条直线则由其始末两点相连接而成。所以，直线的变换可以归结为其始末点的变换。

在对直角坐标系中定义的图形实施几何变换时，既可以看作坐标系不动而图形变动，也可以看作坐标系变动而图形不动。前者表现为图形变动引起图形坐标值产生变化，后者表现为图形没有任何变动, 但在新坐标系中具有新的坐标位置，二者本质上相同。两种方式虽然本质相同, 但由于后者表现为图形没有变化, 因此在实际的变换分析过程中更容易理解. 无论那种方式, 最后的变换公式要表示的却总是同一点变化前后坐标的关系, 所以在实际分析求解时要注意变换的参数施加于图形中的点与施加于坐标系对最终变换公式的影响的不同.

6.2.1 平移变换

6.2.1.1 平移图形的平移变换

平移是指点从一个位置到另一个位置的直线移动，即把点$P(x,y)$平移到$P'(x',y')$, 在$x-$方向移动了距离T_x，在$y-$方向移动了距离T_y。于是可得平移公式

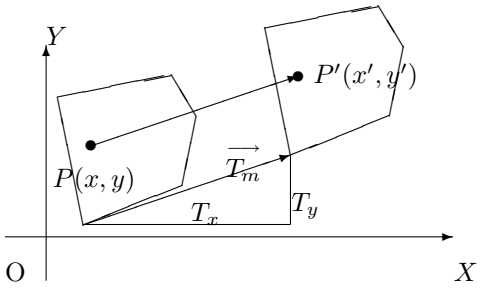
$$\begin{cases} x' = x + T_x \\ y' = y + T_y \end{cases} \tag{6.2.1}$$

平移变换用向量解释更清晰:平面上每一点都对应着一个从坐标原点到这一点的向量, 平移变换实际上就是把整幅图形沿某个方向移动一段距离, 这就是每个点加上一个向量. 用坐标表示出的这个向量就是$\overrightarrow{T_m}=(T_x,T_y)$, 称为平移矢量或位移矢量。用向量表示的平移变换公式就是

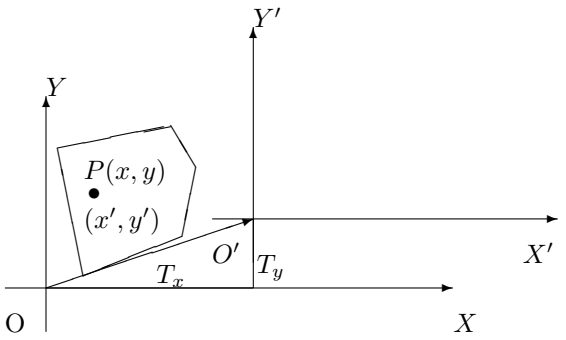
$$\overrightarrow{OP'}=\overrightarrow{OP}+\overrightarrow{PP'}=\overrightarrow{OP}+\overrightarrow{T_m} \tag{6.2.2}$$

或用向量坐标表示即为

$$(x',y')=(x,y)+(T_x,T_y) \tag{6.2.3}$$



(a). 平移图形



(b). 平移坐标系

图6.1 图形的平移变换

显然，多边形的平移变换是由构成多边形的每一条线的始、末点的坐标都加上相应的平移量而得到的。如图6.1所示，多边形从一个位置平移至另一个位置时，其平移量为(T_x, T_y)。由此可见。多边形的平移是通过改变定义其边的坐标值实现的, 不需要对多边形边上的每一点都作变换。而圆和椭圆的平移则需要通过移动其园心坐标和依据其半径来实现。

6.2.1.2 平移坐标系的平移变换

最后要说明的是如果认为坐标系在$x-$方向移动了距离T_x，在$y-$方向移动了距离T_y, 得到新的坐标系$O'X'Y'$, 则在新的坐标系下, 相当于点在$x-$方向反向移动了距离T_x，在$y-$方向反向移动了距离T_y。于是可得平移公式

$$\begin{cases} x' = x - T_x \\ y' = y - T_y \end{cases} \tag{6.2.4}$$

而不是(6.2.1). 用向量表示的平移变换公式就是

$$\overrightarrow{O'P} = \overrightarrow{OP} - \overrightarrow{OO'} \tag{6.2.5}$$

或用向量坐标表示即为

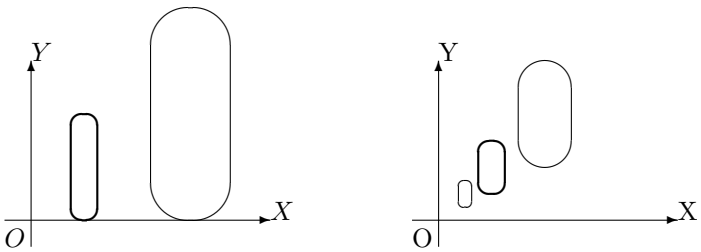
$$(x', y') = (x, y) - (T_x, T_y) \tag{6.2.6}$$

6.2.2 比例变换

6.2.2.1 一个图形的尺寸，变换前、后成比例变化，即称之为比例变换。通过$x-$和$y-$坐标轴方向的比例变换因子S_x和S_y使点$P(x, y)$ 变换为$P'(x', y')$, 则有比例变换公式

$$\begin{cases} x' = xS_x \\ y' = yS_y \end{cases} \tag{6.2.7}$$

比例因子S_x，S_y分别控制$x-$轴和$y-$轴坐标值的缩小和放大。S_x，S_y为大于0的任何数。S_x，$S_y < 1$，图形缩小; S_x，$S_y > 1$，图形放大，$S_x = S_y = 1$，图形不产生变化。$S_x \neq S_y$, 图形被压扁或拉长即产生了拉伸变换. 在图6.2(a)中，$S_x = 3$，$S_y = 2$.



(a). $S_x = 3$, $S_y = 2$ (b). $S_x = S_y = 2$和$S_x = S_y = \frac{1}{2}$

图6.2 图形的比例变换

如图6.2(b)所示,当用式(6.2.7)变换图中的圆角矩形时，如果设定比例因子$S_x = S_y = 1/2$，则属于缩小变换，变换后圆角矩形向坐标原点方向靠拢。设定比例因子$S_x = S_y = 2$，则属于放大变换，变换后圆角矩形向远离坐标原点方向移动。

比例变换的矩阵表示为:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \tag{6.2.8}$$

6.2.2.2 如果选择一个能控制图形比例变换的点，使该点在变换后仍保持不变，则称其为基点。可以选择图形的顶点、中心点或其他任何位置的点为基点。上述比例变换只适合基点在坐标原点的比例变换.

对于一个指定基点(x_F, y_F)(见图6.3)的比例变换, 首先把坐标系原点平移到基点, 在新坐标系下作基点在原点的比例变换, 然后再把坐标系的原点平移回原来的原点。这时的比例变换公式为:

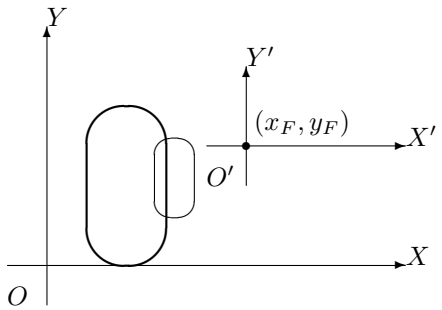


图6.3 图形相对于基点(x_F, y_F)的比例变换

$$\begin{cases} x' = x_F + (x - x_F)S_x \\ y' = y_F + (y - y_F)S_y \end{cases} \tag{6.2.9}$$

将此式整理后，可得到相对于基点的比例变换公式的如下表示式:

$$\begin{cases} x' = xS_x + (1 - S_x)x_F \\ y' = yS_y + (1 - S_y)y_F \end{cases} \tag{6.2.10}$$

对图形中所有点来说，此式的$(1 - S_x)x_F$和$(1 - S_Y)y_F$为常量。因此从几何意义上看, 对任意基点的比例变换相当于对原图形先作了一个基点在原点的比例变换, 再接着作一个平移变换。

对任意基点的比例变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} + [(1 - S_x)x_F \quad (1 - S_y)y_F] \tag{6.2.11}$$

6.2.3 旋转变换

6.2.3.1 图形上的点以坐标原点为中心旋转一定角度产生的变换称为旋转变换. 设点$P(x, y)$旋转至$P'(x', y')$，旋转角为θ (为确定起见设逆时针方向为旋转的正方向)。图6.4中，角ϕ 是$P(x, y)$点在$xy-$平面内与$x-$轴的夹角，用来定义对点$P(x, y)$作旋转变换时的起始位置, 即有

$$x = r\cos\phi, \quad y = r\sin\phi \tag{6.2.12}$$

式中，r是旋转轨迹上的点$P(x,y)$到原点的距离(亦即旋转半径)。根据三角函数关系，可求得旋转变换公式为

$$\begin{cases} x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' = r\sin(\phi+\theta) = r\sin\phi\cos\theta + r\cos\phi\sin\theta \end{cases} \tag{6.2.13}$$

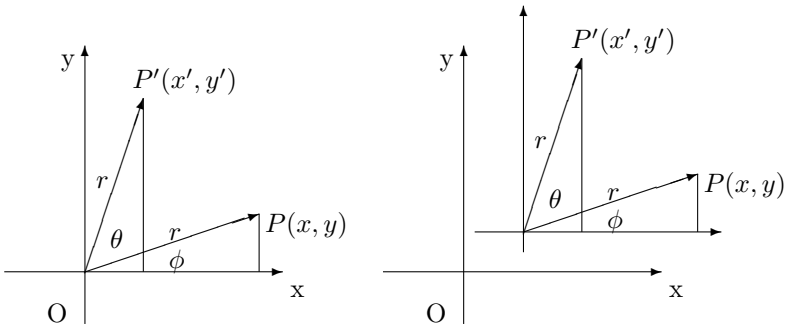
将式(6.2.12)代入式(6.2.13)，则有

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = y\cos\theta + x\sin\theta \end{cases} \tag{6.2.14}$$

根据上面的约定，顺时针旋转时，θ为负值，逆时针旋转时，θ 取正值。

旋转变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{6.2.15}$$



(a). 绕原点　　(b). 饶任意点(x_R, y_R)

图6.4 旋转变换示意图

6.2.3.2　上述变换只适合于以坐标原点为旋转中心的旋转变换. 旋转中心可以选择在图形边界内或边界外的任何地方。当图形需要绕任意点(x_R, y_R)作旋转中心旋转时(如图6.4(b))，类似于以任意点作基点的比例变换的方法, 先平移坐标系使坐标原点平移至旋转中心点, 再作旋转变换, 然后再平移回原来的坐标系原点. 这时的旋转变换公式为

$$\begin{cases} x' = x_R + (x - x_R)\cos\theta - (y - y_R)\sin\theta \\ y' = y_R + (y - y_R)\cos\theta + (x - x_R)\sin\theta \end{cases} \tag{6.2.16}$$

于是可得绕任意中心点的旋转变换的矩阵表示如下:

$$[x' \quad y'] = [x - x_R \quad y - y_R]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} + [x_R \quad y_R] \tag{6.2.17}$$

旋转变换往往包含着三角函数和其他类型的计算。对于动画或某些应用计算，每一个动作都是由小角度的连续运动构成，当旋转角$\theta < 10°$时，三角函数可以用近似值代替，即$\cos\theta$近似等于1，$\sin\theta$则接近于θ (取弧度)值，其误差将随着旋转角的减小而减小。

6.2.4 对称变换

对称变换又可称为反射变换, 镜像变换, 从几何直观上表现为不同的类型: 有关于$x-$轴, $y-$轴的对称变换, 以及关于直线$y=x$, $y=-x$的直线对称变换, 当然也有关于任意一条直线的对称变换; 有关于坐标原点及任意一点的点对称变换. 下面分别加以介绍.

6.2.4.1 关于$x-$轴的对称变换

设对点$P(x,y)$作关于$x-$轴的对称变换后, 得到点$P'(x',y')$, 则变换的特点是x坐标不变, y坐标由正变负, 由负变正, 即改变了符号. 于是变换公式为

$$\begin{cases} x'=x \\ y'=-y \end{cases} \tag{6.2.18}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{6.2.19}$$

6.2.4.2 关于$y-$轴的对称变换

设对点$P(x,y)$作关于$y-$轴对称变换后, 得到点$P'(x',y')$, 则变换的特点是y坐标不变, x坐标由正变负, 由负变正, 即改变了符号. 于是变换公式为

$$\begin{cases} x'=-x \\ y'=y \end{cases} \tag{6.2.20}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.2.21}$$

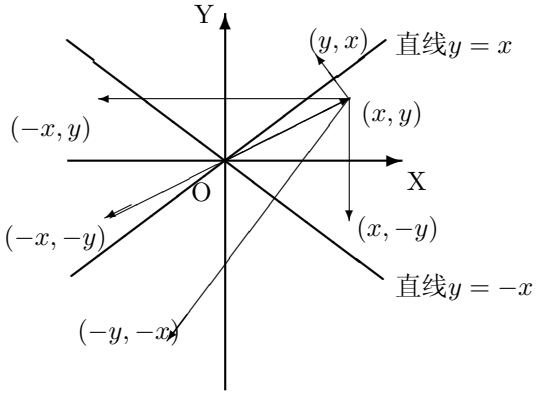


图6.5 点(x,y)的各种对称点位置示意图

6.2.4.3 关于$y=x$直线的对称变换

设对点$P(x,y)$作关于$y=x$进行对称变换后, 得到点$P'(x',y')$, 则变换的特点是x, y坐标互换, 即x坐标变成y坐标; y坐标变成x坐标. 于是变换公式为

$$\begin{cases} x'=y \\ y'=x \end{cases} \tag{6.2.22}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{6.2.23}$$

6.2.4.4 关于$y=-x$直线的对称变换

设对点$P(x,y)$作关于$y=-x$进行对称变换后, 得到点$P'(x',y')$, 则变换的特点是x, y坐标互换, 即x坐标变成y坐标; y坐标变成x坐标, 同时坐标改变了符号. 于是变换公式为

$$\begin{cases} x' = -y \\ y' = -x \end{cases} \tag{6.2.24}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \tag{6.2.25}$$

6.2.4.5 关于坐标系原点的对称变换

设对点$P(x,y)$作关于坐标系原点进行对称变换后得到点$P'(x',y')$, 则变换的特点是x, y坐标同时改变了符号. 于是变换公式为

$$\begin{cases} x' = -x \\ y' = -y \end{cases} \tag{6.2.26}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{6.2.27}$$

6.2.4.6 关于任意直线, 任意点的对称变换

上述对称变换是对特定的直线和点的对称变换. 对于任意点(这一点称对称中心)的对称变换, 类似于前面的一般的比例变换和旋转变换, 先要把坐标系原点平移到对称中心, 再进行坐标系原点的对称变换, 然后把坐标系平移回原来的坐标系原点.

关于以任意指定直线作对称轴的对称变换, 可先把坐标系原点平移到对称轴线上, 再旋转坐标系, 使$x-$轴(或上述其它三种轴对称变换的轴线) 重合于这里的轴线; 这时作相应的轴对称变换; 然后, 反向旋转坐标系到原来的坐标系方向, 并把坐标系原点平移回原来的坐标系原点即可.

把关于坐标系原点及两条坐标轴的对称变换和比例变换结合起来考虑, 我们可以看出比例变换的比例系数也可以是负数, 这时的变换结果实际上相当于同时进行了比例变换和对称变换. 如果两个比例系数都是负数, 则对称变换是关于原点的对称变换; 如果只有一个比例系数是负数, 则对称变换是关于负数比例因子对应的坐标轴的对称变换.

6.2.5 错切变换

错切变换是描述几何形体的扭曲和错切变形的变换，通常用的错切变换是沿$x-$轴或$y-$轴方向产生一个依赖于另一坐标的线性变化, 同时另一坐标保持不变。另外还有在两个坐标轴方向同时产生错切的错切变换.

6.2.5.1 沿$x-$轴方向的错切变换

设点$P(x,y)$沿$x-$轴方向进行错切变换后, 得到点$P'(x',y')$, 则变换的结果是$x-$坐标产生了一个依赖于$y-$坐标的线性变化(线性系数为H_x), 同时$y-$坐标保持不变. 于是变换公式为

$$\begin{cases} x' = x + H_x y \\ y' = y \end{cases} \tag{6.2.28}$$

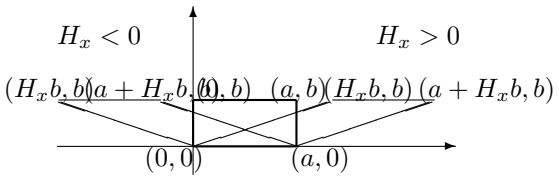


图6.6 沿$x-$轴的错切变换

错切变换参数H_x可以取任意实数值，且只在$x-$方向起作用，而$y-$方向值不变。在图形上表现为每一个点的水平方向位移量与$y-$坐标成比例。如图6.6所示, 如果$H_x > 0$, 图形沿$x-$轴向正方向(右方)的错切; $H_x < 0$, 图形沿$x-$轴向负方向(左方)的错切; 且长方形变换后为一平行四边形，即图形发生了错切变形。

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 1 & 0 \\ H_x & 1 \end{bmatrix} \tag{6.2.29}$$

6.2.5.2 沿$y-$轴方向的错切变换

设点$P(x,y)$沿$y-$轴方向进行错切变换后, 得到点$P'(x',y')$, 则变换的结果是$y-$坐标产生了一个依赖于$x-$坐标的线性变化(线性系数为H_y), 同时$x-$坐标保持不变. 于是变换公式为

$$\begin{cases} x' = x \\ y' = H_y x + y \end{cases} \tag{6.2.30}$$

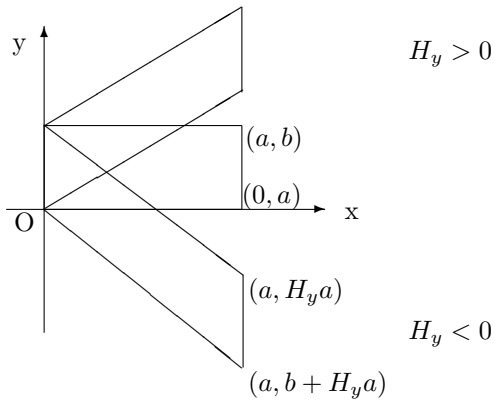


图6.7 沿$y-$轴的错切变换

这种变换使图形在垂直方向错切变形，H_y也可取任意实数值，只在$y-$方向起作用，$x-$方向值不变。如图6.7所示, 在图形

上，每个点垂直方向的平移量与$x-$坐标成比例。如果$H_y > 0$, 图形沿$y-$轴向正方向(上方)的错切; $H_y > 0$, 图形沿$y-$轴向负方向(下方)的错切.

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 1 & H_y \\ 0 & 1 \end{bmatrix} \tag{6.2.31}$$

6.2.5.3 两个坐标轴方向的错切变换

设点$P(x, y)$同时在两个坐标轴方向进行错切变换后, 得到点$P'(x', y')$, 则变换的结果是$x-$坐标产生了一个依赖于$y-$坐标的线性变化(线性系数为H_x), 同时, $y-$坐标产生了一个依赖于$x-$坐标的线性变化(线性系数为H_y). 于是变换公式为

$$\begin{cases} x' = x + H_x y \\ y' = H_y x + y \end{cases} \tag{6.2.32}$$

这一变换的矩阵表示如下:

$$[x' \quad y'] = [x \quad y] \begin{bmatrix} 1 & H_y \\ H_x & 1 \end{bmatrix} \tag{6.2.33}$$

不难看出, 沿$x-$轴方向的错切变换或沿$y-$轴方向的错切变换只是两个坐标轴方向的错切变换在错切系数$H_x = 0$或$H_y = 0$时的特例. 因此也可以认为上述三种错切变换只是同一种.

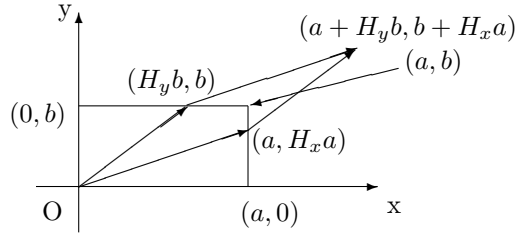


图6.8 两个坐标轴方向的错切变换

类似地, 我们也可以定义以任意点作基点的错切变换.

6.3 齐次坐标与基本变换的矩阵表示

在很多图形变换的实际应用中(例如CAD/CAM设计系统中将一组定义的几何图形构成一张设计对象的装配图时)，需要对每一个子图实施比例、平移、旋转等连续变换，以便使它们能根据技术要求进入图面的恰当位置。这些连续变换是有序而不同步的。一般情况下，一个复杂的变换结果需要分解为前节所述的多个基本变换来实现, 且各个变换顺序是不能随意更改的. 这多个基本变换一般总会包括有比例变换，旋转变换，平移变换等等。这种连续变换的计算方法在求解一个点的最终变换结果时需要应用多个变换公式进行计算, 因而比较费时间。矩阵法是一种更为有效的变换计算方法，用矩阵形式来表示各项基本变换，并用组合矩阵给出最终的整体变换的变换公式进行计算, 就可以根据初始

坐标直接计算出最终变换结果的坐标。同时, 矩阵的运算乘法也是一种非常规则的计算方法, 适合于用硬件来实现, 进一步提高运算速度. 遗憾的是前面的的基本变换中, 平移变换不能通过矩阵乘法的形式来实现. 下面要讨论的齐次坐标系统可以帮助我们解决这一问题.

6.3.1 齐次坐标的概念

在齐次坐标系中,n维空间的点$[x_1, x_2, x_3, \cdots, x_n]$用$n+1$维齐次坐标

$$[x_1h, x_2h, x_3h, \cdots, x_nh, h]$$

表示, 其中$h \neq 0$。反之，若已知$n+1$ 维齐次坐标$[x_1, x_2, x_3, \cdots, x_n, h]$, 则相应的$n$维空间的点为

$$[\frac{x_1}{h}, \frac{x_2}{h}, \frac{x_3}{h}, \cdots, \frac{x_n}{h}].$$

特别是二维空间的点用三维齐次坐标表示。二维点$[x, y]$, 用齐次坐标表示即为$[xh, yh, h]$, 其中h是一个不为零的比例因子。一个具体的二维点$[3, 2]$，若用齐次坐标表示，可以有$[3, 2, 1]$, $[6, 4, 2]$, $[9, 6, 3]$等无穷组齐次坐标，也就是说，其具体的表现形式不唯一。如果分别用比例因子除其余二分量(规范化)，即可得到与之对应的二维点的直角坐标。

二维点$[x, y]$的齐次坐标可表示为一个三维点$[x, y, 1]$。其中, x, y坐标无变化，只是增加了$h = 1$的附加坐标，其几何意义相当于点$[x, y]$落在$h = 1$的平面上。如图6.9所示，xy平面上的$ABCD$，可用各顶点的齐次坐标表示为$h = 1$平面上的$ABCD$。显然，当附加坐标$h = 1$时，图形不发生变化，而仅沿附加坐标$h-$方向平移了一段距离。图6.9是$h = h_0 \neq 1$的情况。

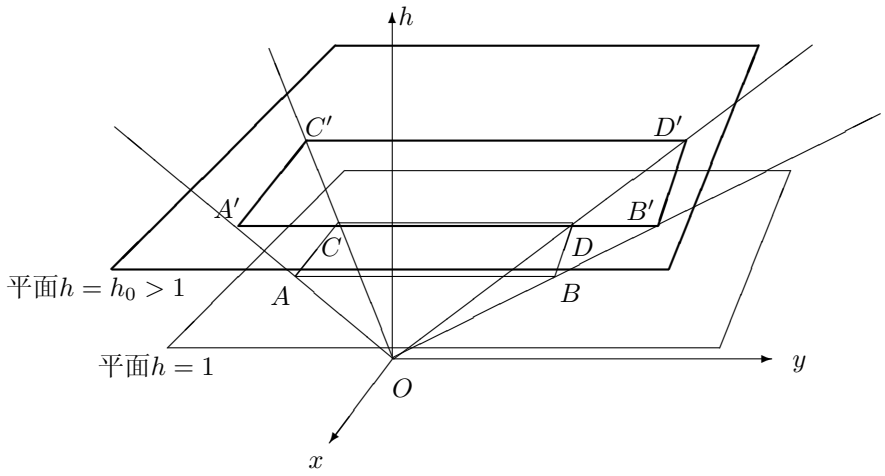


图6.9 齐次坐标的几何意义

根据如上解释，点$[x, y]$的齐次坐标可表示为$[x_h, y_h, h]$，而

$$x_h = xh, \quad y_h = yh$$

我们可以通过研究如下3×3变换矩阵内第三列的数据说明$h \neq 1$的几何意义.

$$[x''\ y''\ \ h] = [x\ \ y\ \ 1] \begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{bmatrix} = [x\ \ y\ \ px + qy + 1] \tag{6.3.1}$$

其中：$x''=x$，$y''=y$，$h=px+qy+1$。这时，齐次坐标h实际上是一个三维空间平面方程。如图6.10所示,这一变换过程可通过一个具体的直线段AB如下变换过程来理解: 位于二维Oxy平面上的直线AB可认为是3维平面$h=1$上的直线段,首先把它平移到3维平面$h=px+qy+1$上，用$A'B'$标识, 然后被以坐标原点为中心的投影变换投射回3维平面$h=1$, 用$A''B''$标识。这相当于将齐次坐标规范化，从而有：

$$\begin{cases} x''=\dfrac{x}{h}=\dfrac{x}{px+qy+1} \\ y''=\dfrac{y}{h}=\dfrac{y}{px+qy+1} \end{cases} \tag{6.3.2}$$

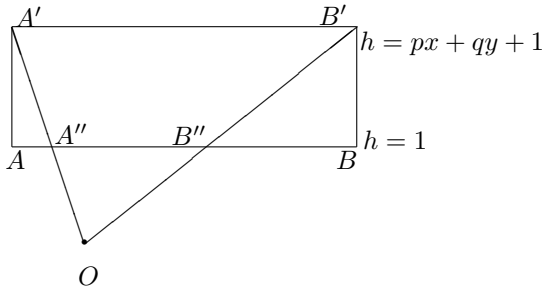


图6.10 齐次坐标变换的几何意义

设直线AB的端点坐标为$A(1,3)$，$B(4,1)$，并假设$p=q=1$，则经齐次坐标h变换$(A\to A'\to A'', B\to B'\to B'')$ 后，得到直线$A''B''$. 其规范化后的坐标为$A''(\frac{1}{5},\frac{3}{5})$, $B''(\frac{2}{3},\frac{1}{6})$。

我们可以简单地取$h=1$, 称为规范化的齐次坐标, 这是我们通常使用的齐次坐标的表示形式. 因此研究二维变换时，二维点的齐次坐标形式通常为$[x\ \ y\ \ 1]$; 后面研究三维变换时，三维点的齐次坐标形式通常为$[x\ \ y\ \ z\ \ 1]$.

6.3.2 基本变换通过齐次坐标的矩阵表示

设点$P(x,y)$进行基本变换后得到点$P'(x',y')$, 分别有齐次坐标$[x\ \ y\ \ 1)$和$[x'\ \ y'\ \ 1]$. 下面给出各个基本变换的矩阵表示变换公式.

6.3.2.1 平移变换

用齐次坐标给出的平移变换的矩阵表示为:

$$[x'\ \ y'\ \ 1]=[x\ \ y\ \ 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \tag{6.3.3}$$

记

$$T=T(T_x,T_y)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \tag{6.3.4}$$

则$T_m=T(T_x,T_y)$称为平移矩阵. 于是上式也可简化为

$$P'=PT(T_x,T_y) \tag{6.3.5}$$

6.3.2.2 比例变换

用齐次坐标给出的比例变换的矩阵表示为:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3.6}$$

记

$$S = S(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3.7}$$

则$S = S(S_x, S_y)$称为比例矩阵. 于是上式也可简化为

$$P' = PS(S_x, S_y) \tag{6.3.8}$$

于是, 绕任意基点(x_F, y_F)的比例变换的矩阵表示如下:

$$\begin{aligned} &[x' \quad y' \quad 1] = [x \quad y \quad 1]T(-x_F, -y_F)S(S_x, S_y)T(x_F, y_F) \\ &= [x \quad y \quad 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ (1-S_x)x_F & (1-S_y)y_F & 1 \end{bmatrix} \end{aligned} \tag{6.3.9}$$

6.3.2.3 旋转变换

用齐次坐标给出的旋转变换的矩阵表示为:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3.10}$$

记

$$R = R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3.11}$$

则$R = R(\theta)$称为旋转矩阵. 于是上式也可简化为

$$P' = PR(\theta) \tag{6.3.12}$$

于是, 绕任意中心点的旋转变换的矩阵表示如下:

$$\begin{aligned} &[x' \quad y' \quad 1] = [x \quad y \quad 1]T(-x_R, -y_R)R(\theta)T(x_R, y_R) \\ &= [x \quad y \quad 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta)x_R + \sin\theta y_R & -\sin\theta x_R + (1-\cos\theta)y_R & 1 \end{bmatrix} \end{aligned} \tag{6.3.13}$$

6.3.2.4 对称变换

关于两个坐标轴和关于坐标系原点的对称变换可归类为广义的比例变换, 只要允许比例变换的系数可以小于零即可. 相应的变换矩阵也可认为是比例变换的矩阵. 下面只列出其余两种对称变换的矩阵.

关于直线$y = x$的对称变换矩阵为

$$D_{y=x} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3.14}$$

关于直线$y=-x$的对称变换矩阵为

$$D_{y=-x}=\begin{bmatrix}0&-1&0\\-1&0&0\\0&0&1\end{bmatrix} \tag{6.3.15}$$

6.3.2.5 错切变换

用齐次坐标给出的关于两个坐标轴同时进行错切的错切变换的矩阵表示为:

$$[x'\ \ y'\ \ 1]=[x\ \ y\ \ 1]\begin{bmatrix}1&H_y&0\\H_x&1&0\\0&0&1\end{bmatrix} \tag{6.3.16}$$

记

$$C=C(H_x,H_y)=\begin{bmatrix}1&H_y&0\\H_x&1&0\\0&0&1\end{bmatrix} \tag{6.3.17}$$

则$C=C(H_x,H_y)$称为错切矩阵. 于是上式也可简化为

$$P'=PC(H_x,H_y) \tag{6.3.18}$$

当$H_y=0$时, 可得沿$x-$轴方向的错切变换矩阵

$$C_x(H_x)=C(H_x,0) \tag{6.3.19}$$

当$H_x=0$时, 可得沿$y-$轴方向的错切变换矩阵

$$C_y(H_y)=C(0,H_y) \tag{6.3.20}$$

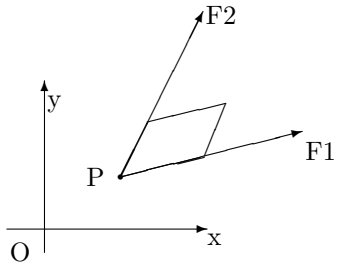
6.3.3 复合变换

前面考虑的都是简单情形的图形变换的变换公式, 对于复杂的图形变换, 可以通过逐步分解的方法把它表示为多个简单变换的复合变换. 考虑如下问题: 一物体被固定在图6.11(a)所示的点$P(a,b)$处, 同时受到来自两个方向的力F_1, F_2的牵引而产生拉伸变形, 拉伸后的变形的比例系数分别为S_1, S_2. 假设这两个力的方向与$x-$轴的夹角分别为α, β. 我们按如下步骤实现这个变换, 求出变换矩阵.

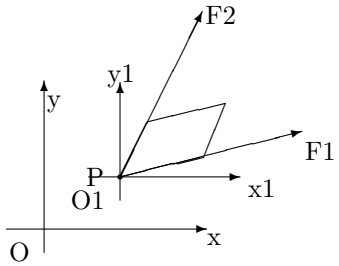
1. 把坐标系原点平移到固定点$P(a,b)$, 于是有平移变换矩阵$T(-a,-b)$, 如图6.11(b)所示;

2. 绕新的坐标系原点旋转α, 使$x-$轴与第一个力的方向一致, 于是有旋转变换$R(\alpha)$,如图6.11(c)所示;

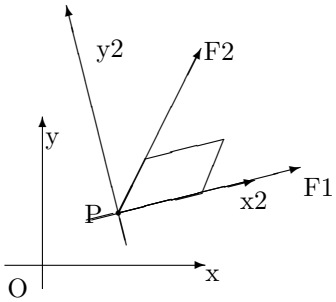
3. 作此坐标系下关于$x-$方向的比例变换, 比例系数为S_1, 于是有比例变换矩阵$S(S_1,1)$,如图6.11(d)所示;



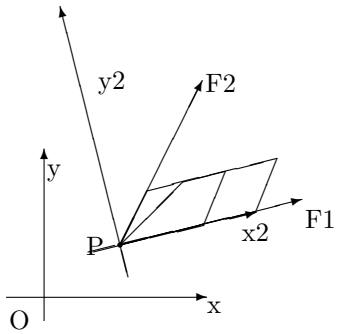
(a). 原始图形



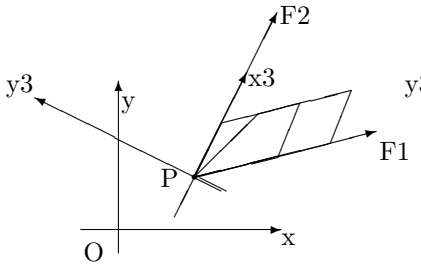
(b). 平移坐标系到点P



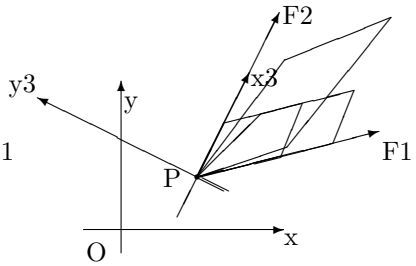
(c). 旋转x轴到F_1方向



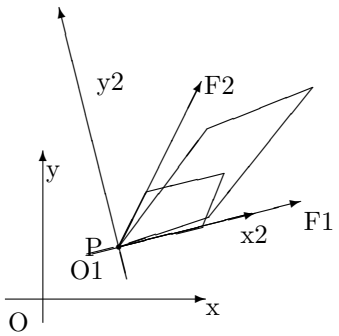
(d). 因子S_1的比例变换



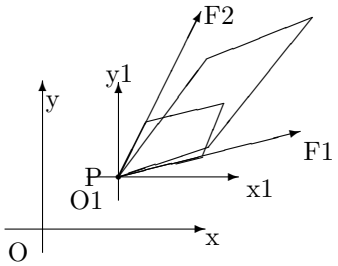
(e). 旋转x轴到F_2方向



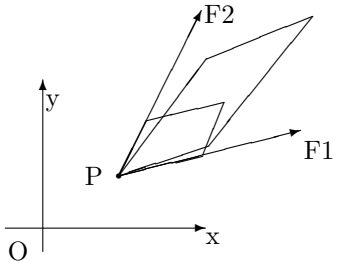
(f). 因子S_2的比例变换



(g). 旋转回F1方向



(h). 旋转回原始坐标方向



(i). 最终变换结果

图6.11 复合变换的变换过程

4. 绕坐标系原点旋转$\beta-\alpha$, 使$x-$轴与第二个力的方向一致, 于是有旋转变换$R(\beta-\alpha)$,如图6.11(e)所示;

5. 作此坐标系下关于$x-$方向的比例变换, 比例系数为S_2, 于是有比例变换矩阵$S(S_2,1)$,如图6.11(f)所示;

6. 绕坐标系原点反向旋转$\beta-\alpha$, 使$x-$轴回到第一个力的方向, 于是有旋转变换$R(\alpha-\beta)$,如图6.11(g)所示;

7. 绕坐标系原点再反向旋转α, 使$x-$轴回到原是坐标系的方向, 于是有旋转变换$R(-\alpha)$,如图6.11(g)所示;

8. 把坐标系原点平移回原始坐标系的原点, 于是有平移变换矩阵$T(a,b)$, 如图6.11(h)所示;

9.上述各个变换矩阵依序相乘, 即可求得复合变换矩阵如下:

$$T(-a,-b)R(\alpha)S(S_1,1)R(\beta-\alpha)S(S_2,1)R(\alpha-\beta)R(-\alpha)T(a,b) \qquad (6.3.21)$$

因为矩阵的乘法满足结合律, 但不满足交换律, 这一结果中各个矩阵的顺序不能改动.

6.3.4 基本变换的一些性质

一般矩阵的乘法不满足交换律,使得矩阵的乘法中各个矩阵的顺序不能改动, 但基本变换矩阵有一些简单性质有助于简化复合矩阵求解过程中的运算, 现列出

如下:

$$\begin{cases} T(T_{x1},T_{y1})T(T_{x2},T_{y2}) = T(T_{x1}+T_{x2},T_{y1}+T_{y2}) \\ R(\theta_1)R(\theta_2) = R(\theta_1+\theta_2) \\ S(S_{x1},S_{y1})S(S_{x2},S_{y2}) = S(S_{x1}S_{x2},S_{y1}S_{y2}) \\ C_x(H_{x1})C_x(H_{x2}) = C_x(H_{x1}+H_{x2}) \\ C_y(H_{y1})C_y(H_{y2}) = C_y(H_{y1}+H_{y2}) \end{cases} \tag{6.3.22}$$

这一组性质说明对于基本的变换，连续作两次完全相同的变换等价于只作一次同类变换，除了比例变换的比例因子需要原比例因子的连乘积外，其他变换的参数皆为原参数之和。

利用这些性质, (6.3.21)中的复合变换矩阵可写为

$$T(-a,-b)R(\alpha)S(S_1,1)R(\beta-\alpha)S(S_2,1)R(-\beta)T(a,b) \tag{6.3.23}$$

6.4 三维图形的基本变换

二维图形, 即平面图形的基本变换可以很直观地推广为三维图形, 即立体空间图形的图形变换. 利用齐次坐标表示后, 三维图形的基本变换可以用4阶矩阵表示出来. 复杂的变换也可以用一系列基本变换矩阵乘积给出的复合矩阵表示出来. 但三维变换由于立体图形的复杂性, 要比平面图形的变换复杂得多. 下面分别介绍各种三维图形的基本变换

设点$P(x,y,z)$进行基本变换后得到点$P'(x',y',z')$, 分别有齐次坐标$[x\ \ y\ \ z\ \ 1]$和$[x'\ \ y'\ \ z'\ \ 1]$. 下面给出各正基本变换的矩阵表示变换公式.

6.4.1 三维平移变换

设三维平移变换在$x-$方向移动了距离T_x，在$y-$方向移动了距离T_y, 在$z-$方向移动了距离T_z。于是可得三维平移公式

$$\begin{cases} x' = x + T_x \\ y' = y + T_y \\ z' = z + T_z \end{cases} \tag{6.4.1}$$

用齐次坐标给出的平移变换的矩阵表示为:

$$[x'\ \ y'\ \ z'\ \ 1] = [x\ \ y\ \ z\ \ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \tag{6.4.2}$$

记

$$T = T(T_x,T_y,T_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \tag{6.4.3}$$

则$T = T(T_x,T_{y,z})$称为三维平移矩阵. 于是上式也可简化为

$$P' = PT(T_x,T_y,T_z) \tag{6.4.4}$$

6.4.2 三维比例变换

设$x-$, $y-$, $z-$三个坐标轴方向的比例变换因子分别为S_x, S_y, S_z, 于是可的三维比例变换公式

$$\begin{cases} x' = xS_x \\ y' = yS_y \\ z' = zS_z \end{cases} \tag{6.4.5}$$

用齐次坐标给出的三维比例变换的矩阵表示为:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.6}$$

记

$$S = S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.7}$$

则$S = S(S_x, S_y, S_z)$称为比例矩阵. 于是上式也可简化为

$$P' = PS(S_x, S_y, S_z) \tag{6.4.8}$$

对于一个任意基点(x_F, y_F, z_F)的三维比例变换, 首先把坐标原点平移到基点, 再作比例变换, 然后再把坐标系的原点平移回原来的原点。这时的比例变换公式为:

$$\begin{cases} x' = xS_x + (1 - S_x)x_F \\ y' = yS_y + (1 - S_y)y_F \\ z' = yS_z + (1 - S_z)y_z \end{cases} \tag{6.4.9}$$

于是, 在任意基点(x_F, y_F, z_F)的比例变换的矩阵表示如下:

$$\begin{aligned} [x' \quad y' \quad z' \quad 1] &= [x \quad y \quad z \quad 1]T(-x_F, -y_F, -z_F)S(S_x, S_y, S_z)T(x_F, y_F, z_F) \\ &= [x \quad y \quad z \quad 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ (1 - S_x)x_F & (1 - S_y)y_F & (1 - S_z)z_F & 1 \end{bmatrix} \end{aligned} \tag{6.4.10}$$

6.4.3 三维旋转变换

平面图形的旋转变换总是绕某一点旋转的, 空间图形的旋转变换就稍稍复杂一些. 我们首先要确定一条直线作为旋转轴; 然后再指定一个旋转角度. 为了确定指定角度的旋转方向, 我们需要指定旋转轴一个方向. 角度的正向规定如下: 用右手握住旋转轴, 大拇指指向旋转轴的方向, 则四指旋转握住旋转轴的方向就是角度的正向.

最简单的三维旋转变换就是以各个坐标轴为旋转轴的旋转变换. 绕$z-$轴的旋转变换是平面情形的直接推广. 这时$z-$坐标不变, $x-$, $y-$坐标的变换公式与平面情形完全一致. 于是有变换公式

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = y\cos\theta + x\sin\theta \\ z' = z \end{cases} \tag{6.4.11}$$

其中, θ为旋转角度.

用齐次坐标给出的旋转变换的矩阵表示为:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.12}$$

记

$$R_z = R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.13}$$

则$R_z = R_z(\theta)$称为绕$z-$轴的旋转矩阵. 于是上式也可简化为

$$P' = PR_z(\theta) \tag{6.4.14}$$

绕$x-$轴的旋转变换矩阵可通过置换式(6.4.11)中的坐标得到，其置换顺序为：用y置换x,用z置换y，用x置换z. 于是对比式(6.4.13)可得绕$x-$轴的旋转公式如下：

$$\begin{cases} y' = y\cos\theta - z\sin\theta \\ z' = y\sin\theta + z\cos\theta \\ x' = x \end{cases} \tag{6.4.15}$$

其齐次坐标形式为:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.16}$$

再次对比式(6.4.13), 经置换坐标后，可以得出绕$y-$轴旋转的公式：

$$\begin{cases} x' = z\sin\theta + x\cos\theta \\ y' = y \\ z' = z\cos\theta - x\sin\theta \end{cases} \tag{6.4.17}$$

其齐次坐标形式为

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.18}$$

6.4.4 三维对称变换

正如平面图形的变换一样, 三维比例变换在比例因子为-1时也给出了三维图形的对称变换. 具体的对应关系如下:

(1). 以坐标原点为对称中心的对称变换:

$S_x = S_y = S_z = -1$;

(2). 以坐标轴为对称轴的对称变换:

$x-$轴: $S_x = 1$, $S_y = S_z = -1$;

$y-$轴: $S_y = 1$, $S_x = S_z = -1$;

$z-$轴: $S_z = 1$, $S_x = S_y = -1$;

(3). 以坐标平面为对称平面的对称变换:

xy平面: $S_x = S_y = 1$, $S_z = -1$;

xz平面: $S_x = S_z = 1$, $S_y = -1$;

yz平面: $S_y = S_z = 1$, $S_x = -1$.

6.4.5 三维错切变换

三维错切变换的结果是$x-$坐标产生了一个依赖于$y-$和$z-$坐标的线性变化(线性系数为H_{xy}和H_{xz}); $y-$坐标产生了一个依赖于$x-$和$z-$坐标的线性变化(线性系数为H_{yx}和H_{yz}); $z-$坐标产生了一个依赖于$x-$和$y-$坐标的线性变化(线性系数为H_{zx}和H_{zy}); 于是变换公式为

$$\begin{cases} x' = x + H_{xy}y + H_{xz}z \\ y' = H_{yx}x + y + H_{yz}z \\ z' = H_{zx}x + H_{zy}y + z \end{cases} \tag{6.4.19}$$

这一变换的矩阵表示如下:

$$[x' \quad y' \quad z'1] = [x \quad y \quad z \quad 1] \begin{bmatrix} 1 & H_{yx} & H_{zx} & 0 \\ H_{xy} & 1 & H_{zy} & 0 \\ H_{xz} & H_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.20}$$

6.4.6 三维复合变换

类似于平面图形的情形, 对于复杂的立体图形变换, 也可以通过逐步分解的方法把它表示为多个简单的三维变换的复合变换.

考虑使图形绕选定的任何轴旋转的三维旋转变换。其复合矩阵由平移矩阵和绕坐标轴旋转的变换矩阵$R_x(\theta)$, $R_y(\theta)$和$R_z(\theta)$构成。正确的连续变换过程是，平移坐标系原点到旋转轴上，旋转坐标系到与三坐标轴之一重合，然后将图形绕该坐标轴旋转。最后用反向变换矩阵使该旋转轴返回原始位置。

假设旋转轴由两点$P_1(x_1, y_1, x_1)$ 和$P_2(x_2, y_2, x_2)$给出, 旋转轴的方向是由P_1到P_2的, 旋转角度为θ.

根据假设, 旋转轴由两个点定义, 其方向与向量$\vec{V} = \overrightarrow{P_1P_2}$相同, 而

$$\vec{V} = (x_2 - x_1, y_2 - y_1, z_2 - z_1) \tag{6.4.21}$$

于是与旋转轴同向的单位向量$\vec{u}$为

$$\vec{u} = \frac{\vec{V}}{|\vec{V}|} = (u_x, u_y, u_z) \tag{6.4.22}$$

其中, 向量$\vec{u}$的各个分量为

$$\begin{cases} u_x = \dfrac{x_2 - x_1}{\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}} \\ u_y = \dfrac{y_2 - y_1}{\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}} \\ u_z = \dfrac{z_2 - z_1}{\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}} \end{cases} \tag{6.4.23}$$

当选择的旋转轴平行于一个坐标轴时，采用一般的三维旋转变换即可以实现所要求的旋转变换。具体实现步骤如下:

(1). 平移坐标系原点到旋转轴上, 使旋转轴与所平行的坐标轴重合;

(2). 根据给定的旋转角θ完成旋转变换;

(3). 平移图形，使旋转轴返回原始位置。

对于一般的情形, 具体实现步骤如下:

(1). 平移坐标系原点到旋转轴上, 使旋转轴过坐标原点;

(2). 旋转图形使旋转轴与坐标轴之一重合; 从理论上讲, 我们可使旋转轴与3个坐标轴的任一个重合，但习惯上往往选择与$z-$轴重合。

(3). 实施旋转变换;

(4). 应用反向旋转变换，使旋转轴返回其原始方位;

(5). 用反向平移变换，使其返回原始位置。

下面给出各步骤详细的解答.

(1). 平移坐标系原点到旋转轴上, 使旋转轴过坐标原点;

我们可以移动坐标系原点到P_1点, 于是有三维平移矩阵

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix} \tag{6.4.24}$$

(2). 旋转图形使旋转轴与$z-$坐标轴重合; 完成这一变换需要分两步:

(a). 首先绕$x-$轴旋转坐标系, 同时也旋转了xz平面，直到旋转轴进入xz平面. 与旋转轴同向的向量$\vec{u}$也就与平面平行. 要建立这个旋转变换矩阵, 必须找出具体的旋转角度α的正弦, 余弦值. 注意前面作的是平移变换, 因此没有改变任何直线和向量的方向, 当然也包括旋转轴和向量$\vec{u}$的方向.

设$\vec{u}_0$为$\vec{u}$在yz平面上的投影向量, 则这一旋转变换的旋转角α就是$\vec{u}_0$和$z-$轴的夹角, 如果把$\vec{u}_0$的起点看作是在坐标系原点, 实际的旋转过程就是旋转$\vec{u}_0$, 使其与$z-$轴重合的过程. 由(6.4.22)易知

$$\vec{u}_0 = (0, u_y, u_z) \tag{6.4.25}$$

如果局限于yz平面来考虑这个旋转角α, 问题就比较容易理解了. 旋转角α就是原点$(0,0)$与点(u_y,u_z)连线和$z-$轴的夹角, 于是

$$\begin{cases} \cos\alpha = \dfrac{u_z}{d} \\ \sin\alpha = \dfrac{u_y}{d} \end{cases} \tag{6.4.26}$$

其中

$$d = \sqrt{u_y^2 + u_z^2} \tag{6.4.27}$$

为$\vec{u}_0$的长度.

由此可以写出这一步绕$x-$轴的旋转变换矩阵矩阵为：

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \dfrac{u_z}{d} & \dfrac{u_y}{d} & 0 \\ 0 & -\dfrac{u_y}{d} & \dfrac{u_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.28}$$

(b). 这时旋转轴在xz平面上, 我们绕$y-$轴旋转，使旋转轴与$z-$轴重合, 并求解这个旋转变换矩阵。经过前面的旋转变换, 向量$\vec{u}$的长度没变, 但方向相对于新的坐标系已经改变了. 设新的向量为$\vec{u'}$. 由于变换是绕$x-$轴的, 所以$x-$分量没有改变. 由于变换后向量$\vec{u}$与xz平面平行, 所以$y-$分量为0. 由于变换后向量$\vec{u}$在yz平面内的投影与$z-$轴重合, 所以$z-$轴分量一定不小于0. 综上所述,

$$\vec{u'} = (u_x, 0, \sqrt{u_y^2 + u_z^2}) \tag{6.4.29}$$

现在绕$y-$轴的旋转就是要使得$z-$轴重合于$\vec{u'}$, 旋转的角度就是$z-$轴和$\vec{u'}$之间的夹角. 局限于xz平面上来考虑这个夹角, 注意到$|\vec{u'}| = |\vec{u}| = 1$, 则不难得出

$$\begin{cases} \cos\beta = d \\ \sin\beta = u_x \end{cases} \tag{6.4.30}$$

由此可以写出这一步绕$y-$轴的旋转变换矩阵矩阵为：

$$R_y(\alpha) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & u_x & 0 \\ 0 & 1 & 0 & 0 \\ -u_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.31}$$

(3). 实施旋转变换; 按指定的旋转角θ绕$z-$轴旋转, 相应的旋转矩阵为:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.32}$$

(4). 应用反向旋转变换，使旋转轴返口其原始方位；
各反向旋转矩阵依次为$R_y(-\beta)$, $R_x(-\alpha)$.

(5). 用反向平移变换，使其返回原始位置。
相应的平移变换矩阵为$T(x_1, y_1, z_1)$.

因此，绕任意轴的旋转变换矩阵就由7个基本变换矩阵的乘积给出：

$$R(\theta) = T(-x_1, -y_1, -z_1)R_x(\alpha)R_y(\beta)R_z(\theta)R_y(-\beta)R_x(-\alpha)T(x_1, y_1, z_1) \tag{6.4.33}$$

6.5 三维投影变换

6.5.1 三维投影变换的概念

现实世界的物体常常是三维的, 即立体的, 如工程设计中产品几何模型, 实际的景物的几何外观等等, 用立体的图形去描述它们们当然是最合适的了. 但我们们描述和观察图形常常却只能在二维介质, 即平面物体上来进行, 如计算机的显示器平面, 绘图机输出的纸平面等等. 投影变换就是帮助我们来解决这一问题的. 它把三维空间中的立体图形变换为二维空间中的平面图形, 从而使我们可以在二维平面介质上显示或输出立体图形。

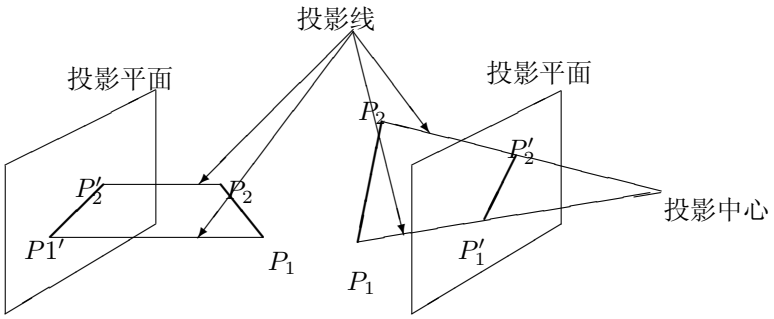


图6.12 不同类型的投影

三维图形实际上是点的集合体. 投影变换可以直观的看作是由一个投影中心发射出的多条的光线给出的投影射线定义的。这些投影射线过图形的每一点并相交于给定的投影平面，从而构成三维图形的投影。空间中一点投影变换的结果就是要求解出过这个点的投影线与投影平面的交点. 图6.12表示同一线段两种不同的投影。由于直线段的投影仍然是直线段，所以实际上是对直线段的端点作投影变换就可以了。

由投影图6.12定义的投影称之为平面几何投影，其特点是投影面为平面，投影线为直线。但是，工程制图领域也有非平面或非直线的投影。

如果投影中心与投影面的距离是是无限的，则投影线相交于无限远点, 因此相互平行, 这时称投影变换为平行投影。如果其间的距离是有限的，则投影线

相交于有限远点, 相互不平行, 这时称投影变换为非平行投影, 通常称为透视投影.

平面几何投影就分为平行投影和透视投影这两种基本类型。当定义透视投影时，需要定义其投影中心。而平行投影只需定义其投影方向即可。

6.5.2 平行投影

平行投影根据它们的投影方向和投影平面法线矢量是否平行, 即投影方向和投影平面是否垂直分为正(平行)投影和斜(平行)投影两类。在正平行投影中，投影方向和投影平面的法线矢量方向是平行的，而斜平行投影的投影方向与投影平面法线矢量方向是不平行的。

6.5.2.1 正(平行)投影

最常见的正投影图实例是工程制图中的主（前）视图、俯（顶）视图和侧视图三种投影图, 合称三视图。正投影的特点是，投影平面垂直于某一个坐标轴，亦即该坐标轴给出投影方向,因而投影平面平行于相应的坐标平面。而三视图更是以三个坐标平面本身作投影平面。正投影被广泛应用于工程制图和构造产品的几何模型。主要的优点是利用这些类型的投影图可以精确测量出立体图中一些真实的距离和角度等几何尺寸。

6.5.2.2 正(平行)投影公式

正平行投影的变换公式是简单的，当设定$z=0$的平面为投影平面时，正投影的方向和投影平面的法线矢量方向相同。因此，$P(x,y,z)$在投影平面上的投影点$P'(x',y',z)$由下式给出.

$$x'=x \quad y'=y \quad z'=0 \tag{6.5.1}$$

这一投影变换的变换矩阵是:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

同理，可以分别写出以$x=0$，$y=0$为投影平面时的正平行投影变换公式和变换矩阵：

$$x'=x \quad y'=0 \quad z'=z \tag{6.5.4}$$

$$x'=0 \quad y'=y \quad z'=z \tag{6.5.5}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

一幅立体图的这三种投影变换图就是最常见工程制图中的俯(顶)视图、主(前)视图和侧视图三种投影图。

6.5.2.3 正轴测投影变换

正投影的另一种应用是生成正轴测投影（axonometric orthographic projection）。轴测投影所用的投影平面一般不垂直于某个坐标轴，因此，一个图形的几个主要侧面可以同时显示出来(在此有一个潜在的假设,即选取坐标系时尽可能使坐标平面为图形表示的物体的主要侧面)。这种投影与投影中心的距离无

关，投影线保持平行，但投影角度可改变，而且沿着每个坐标轴线方向的距离是可以度量的。

等轴测投影（isometric projection）是最常用的轴测投影，其特征是投影平面的法线和每个坐标轴的夹角相等。如果投影平面的法向量设为(a,b,c)，则三个分量应满足要求$|a| = |b| = |c|$, 即$\pm a = \pm b = \pm c$. 考虑投影平面的法向量只是用来指定投影方向的,向量长度的改变不影响其方向, 因而可以认为投影平面的法向量为单位向量,于是有$a^2 + b^2 + c^2 = 3a^2 = 1$, $a = \pm\frac{1}{\sqrt{3}}$. 这样就只有8个方向的向量满足这一条件(即在每一个$\frac{1}{8}$三维坐标卦限区域内有一个方向向量)，但是实际上仅有有4个不同的等轴测投影(除非考虑消除隐藏线问题方向, (a,a,a)和$(-a,-a,-a)$与同一个投影平面相垂直)。故只剩下(a,a,a)，$(-a,a,a)$，$(a,-a,a)$和$(a,a,-a)$是仅有的满足要求的法向量, 其中$a = \pm\frac{1}{\sqrt{3}}$。

等轴测投影具有一些特点，如沿3个坐标轴方向具有相等的变形系数(沿坐标轴方向的量度具有相同的放大缩小的比例系数），此外，主轴(即坐标轴)方向的投影之间具有相等的角度。

对等轴测投影变换(包括一般的投影变换), 其投影公式和变换矩阵可通过如下步骤求解:

(1). 把坐标系原点平移到投影平面上, 得一平移变换矩阵;

(2). 旋转坐标系, 使投影平面重合于坐标平面xy; 实现这一变换结果一般情况下需要绕两个坐标轴作两次基本旋转变换, 产生两个基本变换矩阵.

(3). 作以坐标平面xy为投影平面的正平行投影变换, 得一投影矩阵;

(4). 反序作反向旋转变换, 可得两个旋转变换矩阵;

(5). 把坐标系原点平移回原来的坐标系原点, 得一平移变换矩阵;

(6). 把上述各个变换矩阵依序相乘, 就可得到一般的正平行投影变换矩阵.

6.5.2.4 斜平行投影

斜平行投影变换也可分为斜等轴投影和一般的斜平行投影变换. 对于斜平行投影变换, 不仅要给出其投影平面, 还要给出其投影方向.

我们首先给出投影平面为坐标平面xy, 即平面$z = 0$, 投影方向为向量$\vec{V} = (v_x, v_y, v_z)$ 所指定的投影变换公式和变换矩阵. 对点$P(x,y,z)$, 它的投影点$P'(x',y',z')$就是过点P, 且与投影方向向量$\vec{V}$平行的直线和投影平面$z = 0$的交点. 而直线的参数方程为

$$\begin{cases} z' = x + v_x t \\ y' = y + v_y t \\ z' = z + v_z t \end{cases} \tag{6.5.8}$$

于是可求得相应的斜投影变换公式为

$$\begin{cases} x' = x - \dfrac{v_x}{v_z} z \\ y' = y - \dfrac{v_y}{v_z} z \\ z' = 0 \end{cases} \tag{6.5.9}$$

相应的变换矩阵为

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{v_x}{v_z}z & \frac{v_y}{v_z} & 1 & 0 \end{bmatrix} \tag{6.5.10}$$

其它两个坐标平面的斜平行投影变换与次斜投影变换相似; 斜等轴侧投影变换可用类似于正等轴侧投影的分析的方法得到, 这些斜平行投影变换的变换公式及变换矩阵都不难求出, 在此不再一一介绍.

6.5.3 透视投影

为了得到三维物体几何模型的透视投影（如图6.13所示），首先要定义投影中心和投影平面。投影中心可以选择任何位置, 投影平面可为任意平面。

6.5.3.1 $z-$轴负方向到坐标平面xy的透视投影变换

我们先考虑一个简单情况: 投影平面取为坐标平面xy, 即平面$z = 0$. 投影中心点设置在$z-$轴负方向上，距离投影平面为$d(> 0)$的位置, 因此有坐标$(0, 0, -d)$。

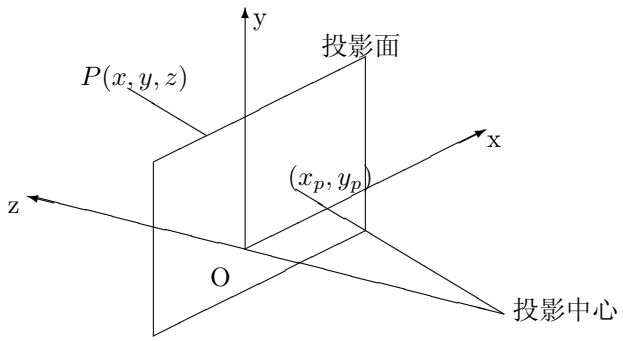


图6.13 透视投影原理图

通过点$P(x, y, z)$和投影中心$(0, 0, -d)$的直线参数方程为：

$$\begin{cases} x' = x - xt \\ y' = y - yt \\ z' = z - (z + d)t \end{cases} \tag{6.5.11}$$

这一直线与投影平面$z' = 0$的交点坐标为

$$\begin{cases} x' = \dfrac{d}{z+d}x = \dfrac{1}{\frac{z}{d}+1}x \\ y' = \dfrac{d}{z+d}y = \dfrac{1}{\frac{z}{d}+1}y \\ z' = 0 \end{cases} \tag{6.5.12}$$

这个交点就是点P透视投影变换后的投影点P'. 此式也就是透视投影变换公式. 这个透视变换的齐次坐标变换矩阵表示为:

$$[x_h \quad y_h \quad z_h \quad h] = [x \quad y \quad z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ & & & \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5.13}$$

这里需要注意式中的

$$h = \frac{z}{d} + 1 \neq 1 \tag{6.5.14}$$

齐次坐标不是按规范化形式给出的, 因此, 投影点P'的坐标和由透视投影变换矩阵给出的齐次坐标的规范化转换关系:

$$x' = \frac{x_h}{h}, \quad y' = \frac{y_h}{h}, \quad z' = \frac{z_h}{h} \tag{6.5.15}$$

由上述公式, 当$d \to \infty$时，将得到平行投影。这与几何直观是相符的.

6.5.3.2 透视投影变换的灭点

考察一束不平行于投影平面的平行线,设其方向向量为$\vec{v} = (v_x, v_y, v_z)$. 由于$\vec{v}$ 不平行于投影平面, $v_z \neq 0$。这一束平行线具有如下形式的表达式:

$$\begin{cases} x = x_0 + tv_x, \\ y = y_0 + tv_y, \\ z = z_0 + tv_z \end{cases} \tag{6.5.16}$$

其中(x_0, y_0, z_0)为直线上的任意已知点. 根据透视变换的投影公式(6.5.12),直线上点(x, y, z)的投影点为:

$$(\frac{dx}{z+d}, \frac{dy}{z+d}, 0) = (\frac{d(x_0 + tv_x)}{z_0 + tv_z + d}, \frac{d(y_0 + tv_y)}{z_0 + tv_z + d}, 0) \tag{6.5.17}$$

当$t \to \infty$时,我们有

$$(\frac{dx}{z+d}, \frac{dy}{z+d}, 0) \to (\frac{dv_x}{v_z}, \frac{dv_y}{v_z}, 0) \tag{6.5.18}$$

只与这一束平行线的方向有关,与具体的某一条直线无关.这说明将立体几何模型用透视投影变换公式投影在投影平面上时，任何一束不平行于投影平面的平行线将相交于一点，这个点称为灭点。不难理解, 当取(x_0, y_0, z_0)为投影中心点时,就得到了这一束平行线中过投影中心的那一条直线. 这条直线上所有点的投影点为同一点:这条直线与投影面的交点. 所以这个交点与这一束平行线的灭点为同一点.

根据上述公式也不难推断,平行于投影面的平行线(这时相应的$v_z = 0$) 将仍然保持平行，即它们只会在无限远处相交，于是在无限远处存在一个灭点。

如果直线束平行于某一个坐标轴，则此时的灭点称为主灭点。对上述透视投影变换而言, 主灭点实际上就是z坐标轴和投影平面的交点$(0, 0, 0)$. 对一般的透视投影,随着投影面的不同, 不平行于投影平面的坐标轴的个数(等于坐标轴和投影平面的交点的个数), 也就是主灭点的个数也不同, 可为1个，2个或3个，因而透视投影变换可以相应地划分为一点透视、两点透视或三点透视。

6.5.3.3 一般的透视投影变换

一般的透视投影变换可按如下步骤求得:

(1). 从投影中心向投影平面作垂线, 求出这条直线和投影平面的交点, 标记为P_2, 把投影中心标记为P_1, 采用三维复合变换一节中使图形绕选定的任何轴旋转的三维旋转变换的方法可把坐标系原点平移到P_1, 并且使$z-$轴重合于P_1P_2。然后再平移坐标系原点到P_2. 这时投影平面就是xy平面, 并且P_1, 即投影中心点在$z-$轴负方向上.

(2). 求出投影中心点到投影平面的距离, 利用前面的简单透视投影变换进行透视投影变换, 可得到一个透视投影变换矩阵.

(3). 反序进行平移, 旋转等变换, 把坐标系变换回原来的位置.

(4). 把各个变换矩阵依序相乘, 即可得一般的透视投影变换矩阵.

当然也可以利用直接求解过投影中心的投影线与投影平面交点的方法得到一般透视投影变换的变换公式和变换矩阵.

6.5.3.4 一般的三维变换矩阵参数含义

一般而言，三维变换矩阵为一4×4方阵，可记为:

$$\begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ T_x & T_y & T_z & s \end{bmatrix} \tag{6.5.19}$$

根据变换功能不同，可以将其元素划分为四部分:

$$\begin{bmatrix} 3\times3|3\times1 \\ ---+--- \\ 1\times3|1\times1 \end{bmatrix} \tag{6.5.20}$$

其中：3×3矩阵元素用以对图形实施比例、旋转、反射、剪切等变换，1×3矩阵元素用以对图形实施平移变换，3×1矩阵元素用以对图形实施透视变唤，1×1矩阵元素用以对图形实施总体比例变换。

对于位置矢量利用上述4×4矩阵实施运算，并将已变换的位置矢量规范化而获得的总变换称之为双线性变换，这个变换给出了一个局部比例变换、旋转、反射、平移、透视、剪切和总体比例变换的组合。

6.6 窗口间的视见变换

6.6.1 图形表示中的坐标系

6.6.1.1 世界坐标系(WC:World Coordinates)

是用户处理自己的图形时所采用的坐标系，现实物体的几何形状本身并没有什么坐标, 坐标是用户为了描述物体的几何形状而附加的. 用户在使用计算机图形系统处理物体的几何形状时，需首先定义其几何形状的坐标表示. 这个坐标系就是世界坐标系, 坐标系的位置, 及尺寸比例, 范围均可由用户根据使用的方便等确定. 它是利用计算机图形处理系统对图形进行定义及描述, 处理的基础。

6.6.1.2 设备坐标系(DC: Device Coordinates)

与一个图形设备相关的坐标系叫设备坐标系。如显示屏就是以分辨率为坐标单位，坐标原点常定义在左上角。绘图仪也有它的坐标系，即以某一角点为坐标原点，以精度为单位。

6.6.1.3 规格化设备坐标系(NDC: Normal Device Coordinates)

规格化设备坐标系是独立于具体物理设备的一种坐标系，它具有的显示空间在x和y方向上都是从0到1。对大多的物理设备而言，NDC与DC 仅仅是坐标值相差一个比例因子。它可以看成是一个抽象的图形设备，要输出到具体的设备时，只需乘上一个比例因子即可。因此，我们在讨论图形输出时，通常是输出到规格化设备坐标系中的。

对于一些具有三维图形处理功能的计算机软件系统, 规格化设备坐标系的显示空间是定义在坐标原点的单位立方体, 在x, y和z方向上都是从0到1。这时NDC与DC就不仅仅是坐标值相差一个比例因子的问题,而是包含着一般的三维图形变换和三维投影变换以及图形的消隐处理等等,这些内容已在前面或将在后面的有关章节介绍, 我们不在这里介绍了。

6.6.1.4 窗口、视口及裁剪

在实际应用中，考察一个图形时，往往采用两种模型。一种是物理模型，它是用户在世界坐标系中描述的; 另一种是逻辑模型，也就是在显示器上呈现的物体的图形，它是在设备坐标系中描述的。

在世界坐标系中描述的实际问题的图形会是相当复杂和庞大的，在具体处理时, 往往需要一部分一部分的研究处理, 当然也需要综合处理完整的图形. 总之, 我们需要考虑局部的图形, 其中的原因可能是用户需要能清晰地观察图形的局部细节; 也可能有时用户只对图形的某一局部区域感兴趣; 因此也只需要处理这一感兴趣的局部区域。这个局部的区域可以由用户在世界坐标系中任意指定，考虑到处理的方便性, 通常是采用矩形区域，称这个矩形区域为窗口(window)，指定或选取这样的一个区域称为开窗口。

当用户指定要显示的内容即开窗口以后，就要把窗口内的图形显示在屏幕上。通常，并不是把整个显示器屏幕都用来显示窗口内的图形，而是在屏幕上指定一个较小的矩形区域，用于显示窗口内的图形，这个在屏幕上的矩形区域就称为视口(viewport)，它是用设备坐标系或规格化设备坐标系进行描述的。

可见，窗口是在世界坐标系中指定待显示内容的区域，视口是在显示器(输出设备)上显示窗口内图形的区域。

窗口和视口可以是多个，而且几个窗口或几个视口可以重叠或嵌套。这一点在利用计算机处理三维的立体图形时是一个经常采用的技术手段. 因为我们无法直接现实处理物体的三维图形, 为了对立体的图形有较好的理解以方便处理, 我们利用多个视口以显示立体图形在不同窗口内的不同的侧面.

窗口和视口并不一定非要矩形区域，有时可以是圆形或多边形的区域。当窗口或视口是一矩形区域时，指定一个窗口或视口就是给出矩形的一对对角顶点坐标值。当视口固定不变，而移动、放大、缩小或旋转窗口时，我们能从视口观察到图形的各个部位，起到类似于照相机镜头的取景的作用。

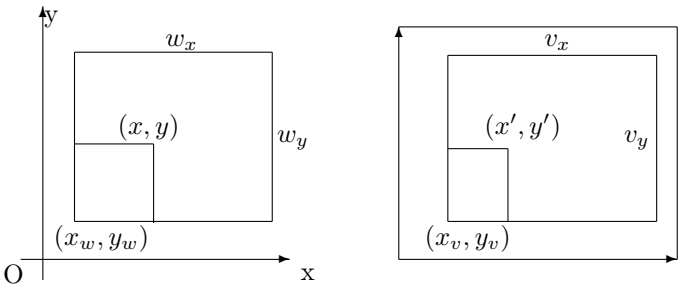
另一方面，在开窗口的时候，图形有可能在窗口内、窗口外或在窗口的边界上。为了正确显示窗口内的全部图形，必须明确地把图形分为窗口内的部分(可见部分)和窗口外的部分(不可见的部分)，尤其是与窗口边界相交的那些图形。区分可见与不可见的过程就称为裁剪(clipping)。相关的内容在图形裁剪一章介绍.

6.6.2 视见变换及其表示

我们只考虑矩形的窗口和视口之间的可见性变换问题.

由于窗口和视口有各自不同的坐标，为了把窗口内的图形正确地显示在视口内，就必须把窗口内的图形进行适当的变换，这一变换过程就称为视见变换。视见变换的具体实现过程如下:

设矩形窗口的左下角坐标为(x_w, y_w), $x-$方向和$y-$方向长度分别为w_x 和w_y; 矩形视口的左下角坐标为(x_v, y_v), $x-$方向, $y-$方向长度分别为v_x 和v_y. 我们的目的是把窗口内的图形在视口内显示出来.



(a).世界坐标系中的窗口 (b). 设备坐标系中的视区

图6.14 窗口,视区及视见变换

很自然地, 首先把窗口的左下角平移到视口的左下角, 只是两个矩形区域的长和宽很可能不一致. 视口是在图形输出设备上, 因此不能改变, 我们只有对图形实行适当的比例变换, 使其能适当地在视口内显示出来. 这只需分别在$x-$和$y-$方向进行比例变换, 调整窗口矩形的长和宽, 使窗口和视口两个矩形区域的长和宽一致即可. 易知$x-$和$y-$方向进行比例变换的比例因子分别为:

$$S_x = \frac{w_x}{v_x}, \quad S_y = \frac{w_y}{v_y} \tag{6.6.1}$$

设(x, y), (x', y')是视见变换前后窗口和视口内的对应点, 则有变换公式

$$\begin{cases} x' = x_v + (x - x_w)S_x = (x_v - x_w S_x) + xS_x \\ y' = y_v + (y - y_w)S_y = (y_v - y_w S_y) + yS_y \end{cases} \tag{6.6.2}$$

这个变换公式的缺点是由于一般情况下$S_x \neq S_y$, 即$x-$和$y-$方向进行比例变换的比例因子不相等, 因此显示出的图形与实际图形相比给人以被拉伸或挤压的感觉. 避免这一缺点的方法应当是要求在$x-$和$y-$方向进行比例变换的比例因子相等, 同时为了保证窗口内所有图形都能显示出来, 一般可取

$$S = min\{\frac{w_x}{v_x}, \frac{w_y}{v_y}\} \tag{6.6.3}$$

于是有变换公式

$$\begin{cases} x' = x_v + (x - x_w)S = (x_v - x_w S) + xS \\ y' = y_v + (y - y_w)S = (y_v - y_w S) + yS \end{cases} \tag{6.6.4}$$

习题 6

1. 严格说来，比例变换示意图有错误。能否看出这个错误？
2. 划出单位正方体在三个不同投影面上的透视图，要求选取的三个投影面分别具有一个灭点、两个灭点和三个灭点。
3. 设投影平面方程为$x + y = 10$, 投影中心为坐标原点。利用求复合变换的方法求投影变换公式。

4. 设投影平面方程为$Ax + By + Cz + D = 0$, 投影中心为投影平面外一点(a, b, c)。利用求投影线与投影平面交点的方法求投影变换矩阵。

Chapter 7

计算机图形中曲线的设计理论

7.1 引言

对曲线的处理是计算机图形学的基本任务之一. 前面我们已经提到了到圆、直线等简单的,当然也是最基本的曲线的处理方法. 但物体的形状可以因为实际工程的需要或者是艺术上审美的需要而成为很复杂很特别的几何形状。在许多计算机图形应用中，表现的物体形状本身是平滑的,光顺的,因而其外形轮廓线就必须是平滑光顺的曲线。首先许多现实世界的目标本身是平滑的，其次计算机图形应用包括模型化实际目标、计算机辅助设计(CAD)的对象、高质量的简字体、数据图表和艺术家的素描等都包含平滑的曲线和曲面，在动画序列中运动的目标路径也应该是平滑的，同样目标表面的强度或颜色空间分布也必须是平滑的。而圆、直线等简单图形不能描述复杂曲线的形状,满足复杂的光顺性要求.

经典数学建立了直线、圆等这些简单几何形状描述和设计的原则及方法。为了得到更逼真更易处理的物体的描述，必须研究那些复杂而特别的几何形状及性质，如光滑、光顺性及连续性等，以便找出一个合适的数学表达式来满足这些要求。上述要求只反映了问题的一个方面,即找出合适的数学表达式以表现出几何形状丰富复杂的曲线.同时也要考虑到计算机图形学是一门应用性很强的科学,设计出的曲线的表示方法应能够使使用这些方法的人能够比较方便地设计出他想要设计出的任意复杂形状的几何曲线.本章即介绍这方面的理论和方法.考虑到曲线的几何形状是一个直观的几何概念,我们尝试从几何直观的观点出发来介绍曲线的设计理论方法,以使理论的结果和方法更直观和易于理解,而不是一开始就引入严格的数学理论和结果. 一些重要结果的数学证明请参阅相关的数学专著,我们只给出直观的几何说明.

7.2 折线段曲线

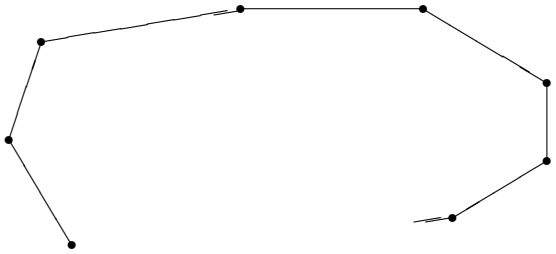


图7.1 折线段连接出的曲线

对于给定的控制点,通过控制点的折线段连接出的曲线是非常自然的一条曲线. 它有许多优点, 这是我们进一步发展曲线设计技术的基础.当然它也有很多缺陷, 这是我们需要进一步发展曲线设计技术的原动力之一.

这一曲线的优点有如下几点:

(1). 它是一条纯粹的几何图形的曲线,不依赖于任何我们在处理这条曲线时必须额外附加给它的几何图形以外的性质.如控制点就表现为空间中的一个几何图形的位置, 折线曲线的每一个直线段就是相邻两点之间的直线段图形. 当我们用计算机处理这个图形时,必须加上一些额外的性质,如我们需要选定一个适当的长度单位,以便决定点之间的距离,然后再选定一个适当的坐标系, 以便决定点的空间几何位置.而长度单位和坐标系依赖于处理这个图形的人的知识的不同,依赖于当时处理这个问题的客观条件的不同会有所不同. 另外, 控制点是有序的,但我们处理时, 它不仅是有序的, 而且还多余要求纯粹是人为因素的一个从先到后或从后到先的顺序. 折线段先于这些额外的性质而存在,说明它不依赖于这些性质,因此不会由于这些性质的改变而改变.用数学化的语言来叙述, 我们称这个折线段曲线具有几何不变性.

(2). 折线段曲线的每个直线段仅存在于两点之间的有限范围内,保证了我们在处理它时总是有结果.具体到计算机的运算处理过程, 就是运算算法总是在一个明确的范围内运算,是可以结束的, 即算法总是收敛的.

(3). 折线段曲线的工作效率是有保障的. 具体的讲, 我们处理一个折线段曲线时,很多时候不是完整的处理全部的这条曲线, 而只是一段一段地设计. 当设计完成时, 可能其中的部分直线段不是很令人满意,需要修改. 但是对其它部分还是比较满意的,并不想作任何改动. 这时我们只需修改相关的直线段的顶点就可以了. 显然,其它的顶点没有改变,相应的折线段也没有改变. 这种性质称为局部性,一个顶点改变时,只有相应的直线段位置改变了,其它的直线段位置没有任何改变.

(4). 其表达式非常简单, 几何意义非常明确. 对于直线段P_iP_{i+1}, 我们可以写出

其方程为:

$$P(t) = tP_i + (1-t)P_{i+1}, \quad t \in [0,1] \tag{7.2.1}$$

点$P(t)$在直线段上, 并把直线段分成两段, 其长度之比为

$$P_iP(t) : P(t)P_{i+1} = t : (1-t).$$

$P(t)$的表达式为两端点的线性组合, 线性组合系数为参数t的一次多项式. 所有直线段组合在一起, 构成折线段曲线.

折线段曲线的缺陷是它仅仅是连续的曲线,每个控制点都是一个角点.这种曲线看起来不够光滑, 即它的光滑程度不够.改进折线段曲线的光滑程度的方法也很直观:切磨各个角点,具体过程如图7.2所示. 图中P_i是角点, $P_{i-1}P_i$和P_iP_{i+1}是角点P_i的两个相邻直线段边.

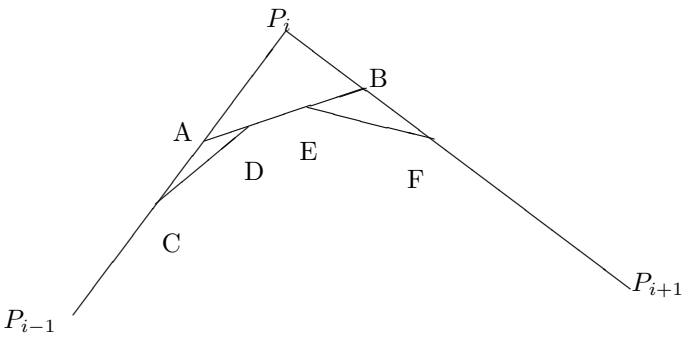


图7.2 磨角法平滑折线段曲线过程示意图

我们首先用AB取代AP_i和P_iB.这里A, B分别在直线段$P_{i-1}P_i$和P_iP_{i+1} 上并且有

$$\begin{cases} |AP_i| : |P_{i-1}P_i| = \alpha_i : 1, \\ |P_iB| : |P_iP_{i+1}| = \beta_i : 1 \end{cases} \tag{7.2.2}$$

其中切磨系数值$0 \le \alpha_i \le 1$, $0 \le \beta_i \le 1$. 于是A, B取代P_i 成为这线段曲线的新的控制点.

如果我们觉得新的折线段仍然不够光滑,可重复上述过程. 于是A, B分别由C, D和E, F取代,相应的切磨系数值表示为α_i^1,β_i^1, α_i^2,β_i^2. 可反复进行上述过程, 直到满意为止.但是, 进一步的理论分析证明,这种方法得到的光滑曲线最好也只能达到切线连续.这当然是不能满足实际需要的.在随后的几节中我们将探讨新的曲线设计方法.无论如何, 在一定意义上, 它们都是本节折线段曲线的某种推广.

7.3 参数三次曲线

7.3.1 参数三次曲线的表示

本节我们用数学的观点来考察生成光滑曲线问题. 给定控制点后, 我们首先想到

的是确定了曲线要经过的位置. 如果要生成光滑的曲线, 就应该给出相应的切线方向. 设每点处有一个切线向量T_i, 点P_i与P_{i+1}之间的一段曲线$P_i(t)$, $t \in [0,1]$就应该满足如下条件:

$$\begin{cases} P_i(0) = P_i \\ P_i(1) = P_{i+1} \\ P_i'(0) = T_i \\ P_i'(1) = T_{i+1} \end{cases} \tag{7.3.1}$$

向量函数$P_i(t)$当然可以为任意类型的函数,考虑到作为前面直线段函数的推广以及计算的方便性,我们要求$P_i(t)$可表为:

$$P_i(t) = F_0(t)P_i + F_1(t)P_{i+1} + F_2(t)T_i + F_3(t)T_{i+1} \tag{7.3.2}$$

并且$F_i(t)$, $i = 0,1,2,3$都是参数t的多项式.结合式(7.3.1),则有

$$\begin{cases} F_0(0)P_i + F_1(0)P_{i+1} + F_2(0)T_i + F_3(0)T_{i+1} = P_i \\ F_0(1)P_i + F_1(1)P_{i+1} + F_2(1)T_i + F_3(1)T_{i+1} = P_{i+1} \\ F_0'(0)P_i + F_1'(0)P_{i+1} + F_2'(0)T_i + F_3'(0)T_{i+1} = T_i \\ F_0'(1)P_i + F_1'(1)P_{i+1} + F_2'(1)T_i + F_3'(1)T_{i+1} = T_{i+1} \end{cases} \tag{7.3.3}$$

注意到式(7.3.3)对任意的P_i, P_{i+1}, T_i和T_{i+1}都成立,我们由此得到

$$\begin{cases} F_0(0) = 1, & F_1(0) = 0, & F_2(0) = 0, & F_3(0) = 0 \\ F_0(1) = 0, & F_1(1) = 1, & F_2(1) = 0, & F_3(1) = 0 \\ F_0'(0) = 0, & F_1'(0) = 0, & F_2'(0) = 1, & F_3'(0) = 0 \\ F_0'(1) = 0, & F_1'(1) = 0, & F_2'(1) = 0, & F_3'(1) = 1 \end{cases} \tag{7.3.4}$$

由(7.3.4),每个函数$F_i(t)$应满足四个等式.因此, 作为参数t的多项式函数,$F_i(t)$的次数最低应为3.作为三次多项式的$F_i(t)$根据式(7.3.4) 不难得到其表达式如下:

$$\begin{cases} F_0(t) = 2t^3 - 3t^2 + 1 \\ F_1(t) = -2u^3 + 3u^2 \\ F_2(t) = u^3 - 2u^2 + u \\ F_3(t) = u^3 - u^2 \end{cases} \tag{7.3.5}$$

函数$F_i(t)$, $i = 0,1,2,3$称为三次埃尔米特(Hermite)函数. 由(7.3.2) 给出的曲线称弗格森曲线, 也是这种曲线的几何形式表示. 这一曲线及其曲面形式由美国波音公司的弗格森(Ferguson J．C，)于1963年首先在飞行器设计中提出.

由(7.3.2), (7.3.1)和(7.3.4)可得

$$\begin{aligned} P_i(t) \quad = \quad & P_i + tT_1 + t^2(-3P_i + 3P_{i+1} - 2T_i - T_{i+1}) \\ & + t^3(2P_i - 2P_{i+1} + T_i + T_{i+1}) \end{aligned} \tag{7.3.6}$$

如果定义

$$\begin{cases} A_0 = P_i \\ A_1 = T_1 \\ A_2 = -3P_i + 3P_{i+1} - 2T_i - T_{i+1} \\ A_3 = 2P_i - 2P_{i+1} + T_i + T_{i+1} \end{cases} \tag{7.3.7}$$

则有

$$P_i(t) = A_0 + tA_1 + t^2A_2 + t^3A_3 \tag{7.3.8}$$

由此式可明显看出$P_i(t)$是参数t的三次多项式,因此是一个三次多项式曲线. 参数三次曲线段简称PC曲线.这也是弗格森曲线的代数形式.

对参数$t \in [0,1]$, $P(t)$表示曲线上任一点的位置向量,有三个分量, 即有：

$$P_i(t) = [x(t) \quad y(t) \quad z(t)] \tag{7.3.9}$$

而A_3、A_2、A_1、A_0有都有三个分量,设为：

$$A_i = [a_{i1} \quad a_{i2} \quad a_{i3}],\ i = 0,1,2,3 \tag{7.3.10}$$

于是(7.3.8)可更清楚的表示为下面的三个三次多项式等式:

$$\begin{cases} x(t) = a_{01} + a_{11}t + a_{21}t^2 + a_{31}t^3 \\ y(t) = a_{02} + a_{12}t + a_{22}t^2 + a_{32}t^3 \\ z(t) = a_{03} + a_{13}t + a_{23}t^2 + a_{33}t^3 \end{cases} \tag{7.3.11}$$

三个多项式中的12个常系数称为代数系数，这组系数唯一地确定了一条参数三次曲线的形状、长短及在空间中的位置。

进一步假设

$$T = [1 \quad t \quad t^2 \quad t^3] \tag{7.3.12}$$

4×3矩阵

$$A = [A_0 \quad A_1 \quad A_2 \quad A_3]^\tau \tag{7.3.13}$$

其中的上标τ是指转置矩阵，则式(7.3.8)又可以写为更简洁的矩阵表示形式形式：

$$P_i(t) = TA \tag{7.3.14}$$

这里的矩阵T称为幂基矩阵，矩阵A可称为代数系数矩阵，通过改变这个代数系数矩阵，总是能够控制由它表示的参数三次曲线的形状及位置。这一表示形式很清楚地说明了参数三次曲线的代数运算性质. 当然，我们不清楚矩阵A的几何含义，因此在控制曲线形状时，无法几何直观地修改系数矩阵以达到修改曲线形状的目的。但是, 参数三次曲线的几何形式(7.3.2)可以解决这个问题。同代数形式一样，如果我们定义

$$\begin{cases} F = [F_0(t) \quad F_1(t) \quad F_2(t) \quad F_3(t) \quad] \\ B = [P_i \quad P_{i+1} \quad T_i \quad T_{i+1}]^\tau \end{cases} \tag{7.3.15}$$

则(7.3.2)又可写成：

$$P_i(t) = FB \tag{7.3.16}$$

我们可称B为几何系数矩阵。B有一个很好的几何解释，它反映了该曲线段的边界条件，即端点的坐标位置及切向量。由B完全确定了参数三次曲线$P(t)$，通过修改B,就可以修改曲线的形状及位置等，也就是说可以改变曲线的两个边界端点位置或端点处切向量方向及大小来直观地修改曲线的形状。在后面7.3，我们会看到，改变端点切向量的大小是如何影响曲线的形状的。弗格森曲线是利用两个端点向量和其切向量共4个向量来表示一曲线段。在实际应用中是难于确定端点切向量的大小的，一种常用的方法是用端点间的距离即弦长作为两端点的切向量长度，这样得到的曲线有一合理的形状。

进一步地,下面我们讨论参数三次曲线的代数系数矩阵A和几何系数矩阵B之间的关系，实际上F可以通过T表示为：

$$\begin{cases} F = [2t^3 - 3t^2 + 1 \quad -2t^3 + 3t^2 \quad t^3 - 2t^2 + t \quad t^3 - t^2] \\ \quad = [1 \quad t \quad t^2 \quad t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \end{cases} \tag{7.3.17}$$

用M代表(8.4.4)式中的4×4矩阵，则F可写成：

$$F = TM \tag{7.3.18}$$

代人(7.3.16)式有：

$$P = TMB \tag{7.3.19}$$

对照(7.3.14)式就有：

$$A = MB \tag{7.3.20}$$

或

$$B = M^{-1}A \tag{7.3.21}$$

其中

$$M^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \tag{7.3.22}$$

是M的逆矩阵.

这个4×4矩阵M称为通用变换矩阵，利用(7.3.20)和(7.3.21)式使得参数三次曲线的代数形式和几何形式之间能容易地相互转化。

7.3.2 参数三次曲线的其它表示形式

任一参数三次曲线总有边界端点、及端点处的切向量，也即任一参数三次曲线均可表示成(7.3.16)和(7.3.19)式的形状，差别仅仅是几何系数矩阵B不相同。但是应当注意切向量量虽然有明确的几何意义,却并不是一个能非常直观地确定的几何量. 作为一个数学问题来处理, 参数三次曲线无论是其代数形式，还是其几何形式，都有12个待定系数，这12个待定系数完全确定了一条参数三次曲线。而确定12个待定系数，只要有12个已知的确定条件就可以了. 前面我们就是知道确定的两个端点和端点处的切向量，并由此确定出12 个已知条件。因此我们可以用多种方法来确定一条参数三次曲线. 但无论如何，得到的曲线表示均可写成(7.3.19)式的几何形式。

另一方面，参数三次曲线也可以通过指定四个曲线经过的点来确定形状，或者指定一个经过的点和三个切向量等多种方法来求出12个常系数，从而确定曲线的表示。

依此观点还有其它很多的方法，如:

1). 给出空间四个点P_0、P_1、P_2和P_3，要求找出一段参数三次曲线$P(t)$,当$t = t_1 = 0 < t_1 < t_2 < t_3 = 1$时,$P(t_i) = P_i,\ t = 0, 1, 2, 3.$

2). 给出曲线两点P_0和P_1，两个单位向量T_0和T_1，及一点C。要求找出一段参数三次曲线$P(t)$,使得$P(0) = P_0$,$P(1) = P_1$. 同时,两个端点P_0和P_1处的切向量分别与T_0和T_1同向,并经过点C.

这两种类型的已知条件都可以确定如上的12个待定系数，或确定几何系数矩阵，我们以第一种类型的已知条为例求解如下。

如果我们希望求得一个4×4的矩阵K，使得当从右边乘上四个点的坐标矩阵时，产生该待构造曲线的几何系数矩阵B即

$$B = [P_0 \quad P_3 \quad T_0 \quad T_3] = K \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \tag{7.3.23}$$

则我们可求解如下. 因为

$$P(t) = TMB \tag{7.3.24}$$

所以

$$P_i = P(t_i) = [1 \;\; t_i \;\; t_i^2 \;\; t_i^3]MB, \quad i = 0, 1, 2, 3 \tag{7.3.25}$$

也就是说

$$\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 1 & t_3 & t_3^2 & t_3^3 \end{bmatrix} MB \tag{7.3.26}$$

于是有

$$B = M^{-1} \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 1 & t_3 & t_3^2 & t_3^3 \end{bmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \tag{7.3.27}$$

代入(7.3.23), 并结合$t_0 = 0$, $t_3 = 1$,即可得到所要求的矩阵K为：

$$K = M^{-1} \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 1 & t_3 & t_3^2 & t_3^3 \end{bmatrix} = M^{-1} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{7.3.28}$$

7.3.3 参数三次曲线的几何形状

折线段曲线可以认为是先有曲线本身,然后才有其参数方程,因此其几何形状非常清楚. 而参数三次曲线则不是这样, 我们只是有了曲线应当要满足的几何条件, 然后根据这些几何条件确定出一个或几个代数方程, 并依此产生出一条曲线,才有曲线的几何形状. 已知的几何条件,如端点处的切线向量这个条件等. 如果我们只是用它来确定端点处的切线的方向,那么它的模长就不应该对曲线的形状产生什么影响. 而通过下面的分析就可以看到, 它的模长对曲线的形状产生了影响.

定义与T_i同向的单位向量为U_i,则有

$$U_i = \frac{T_i}{|T_i|}, \quad T_i = |T_i|U_i \tag{7.3.29}$$

于是由(7.3.16)可得

$$P_i(t) = FB = F \begin{pmatrix} P_i \\ P_{i+1} \\ T_i \\ T_{i+1} \end{pmatrix} = \begin{pmatrix} P_i \\ P_{i+1} \\ |T_i|U_i \\ |T_{i+1}|U_{i+1} \end{pmatrix} \tag{7.3.30}$$

对下面图形中所显示的端点和端点处切向量关系的情形中给出的平面参数三次曲线,在保持切线方向不变, 即U_i和U_{i+1}不变的情况下, $|T_i|$和$|T_{i+1}|$同时增加, 所求参数三次曲线会变得丰满些.但如果太大, 就会出现尖点或环.如图所示, 设点M是P_i点切线的正方向和P_{+1}点切线的反方向部分的交点, 则精确的理论分析可以证明当$|T_i| < |P_iM|$和$|T_{i+1}| < |MP_{i+1}|$时, 就不会出现尖点或环. 当然,如果M不存在,也不会出现尖点或环.

如果仅$|T_i|$增加,而$|T_{i+1}|$比较小且保持不变,则会使参数三次曲线在P_{i+1} 点转向切向量U_{i+1}指定的方向之前有较长的一段靠近切向量U_i指定的方向.反之, 如

果$|T_i|$比较小且保持不变,而$|T_{i+1}|$增加,曲线也会在P_{i+1} 点有较长的一段靠近切向量U_{i+1}指定的方向.

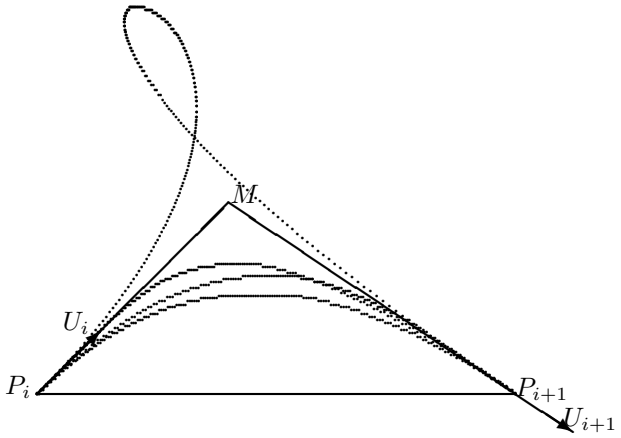


图7.3 两端点处切向量长度同时增加对曲线形状的影响

一般而言,如果把切向量T_i的起点放在点P_i,把切向量T_{i+1}的终点放在点P_{i+1},则连接点P_i,切向量T_i的终点,切向量T_{i+1} 的起点和点P_{i+1}这四点构成的多边形的形状大体给出了参数三次曲线的形状.

7.3.4 参数三次曲线参数值域的变换

前面的讨论中，我们都假设对每一段参数三次曲线参数t是在$[0,1]$区间内变化的。如果我们来考虑多于一条的参数三次曲线段组成一条完整的曲线, 不难发现这种做法是很任意的,因为相邻的每对控制点与控制点之间的差别可以非常大. 现在我们来实际考查参数取值域的变化会对相应的一段参数三次曲线带来什么样的影响.

假定对前面定义的参数三次曲线另有参数u,其值域是$u \in [u_i, u_{i+1}]$, 参数u和t存在着函数关系

$$t = t(u) \tag{7.3.31}$$

并且应同时满足方程:

$$\begin{cases} t(u_i) = 0 \\ t(u_{i+1}) = 1 \end{cases} \tag{7.3.32}$$

因此参数三次曲线$P_i(t)$用参数u可表示为:

$$P_i(t) = P_i(t(u)), \quad u \in [u_i, u_{i+1}] \tag{7.3.33}$$

如果要求$P_i(t(u))$仍然是参数u的三次多项式,函数$t(u)$应该是参数u的一次多项式,即参数u和t存在着线性函数关系,这个表示关系是唯一的,可表示如下:

$$t(u) = \frac{u - u_i}{\Delta u_i}, \quad \Delta u_i = u_{i+1} - u_i \tag{7.3.34}$$

分别用$u = u_i$和$u = u_{i+1}$代入式(7.3.33)可得

$$\begin{cases} P_i(t(u_i)) = P_i(0) = P_i \\ P_i(t(u_{i+1})) = P_i(1) = P_{i+1} \end{cases} \tag{7.3.35}$$

这也就是说,参数三次曲线$P_i(t)$的两个端点保持不变.

再来考察切向量,对参数u求导可得

$$\frac{dP_i(t(u))}{du} = \frac{dP_i(t)}{dt}\frac{dt}{du} = \frac{1}{\Delta u_i}\frac{dP_i(t)}{dt} \tag{7.3.36}$$

由此式可知,相对于另外一组参数u,切向量的方向没有发生变化,但是切向量的模长却发生了变化.参数t和u相应的切向量的模长之间差一个比例因子$\frac{1}{\Delta u_i}$.

上面的方法是通过参数t的$P_i(t)$得到对于参数u的三次多项式曲线,我们完全可以在不知道参数t的$P_i(t)$的情况下直接求得一个对于参数u的三次多项式曲线, 方法就是由参数t求得$P_i(t)$的方法.这时得到的对于参数u的三次多项式曲线的端点当然仍是在点P_i和P_{i+1},但端点处的切向量却是T_i和T_{i+1}, 而不是$P_i(t(u))$的$\frac{1}{\Delta u_i}T_i$和$\frac{1}{\Delta u_i}T_{i+1}$.

这两个关于参数u的三次多项式曲线可以变换到用参数t表示的三次多项式曲线.但是应注意到他们的切向量方向相同,模长不同.根据前一小节的分析, 这两条曲线是完全不同的两条曲线,有可能一条有尖点或环,而另一条却没有.

这说明不同的参数值域产生的三次参数曲线有不同的几何形状.

7.4 Bézier曲线

三次参数曲线是使用纯代数的方法推导出来的,代数参数的几何意义比较明确,这是其优点.但是其产生的曲线的几何形状和存在的范围仍然不容易控制和掌握. 下面介绍的Bézier曲线将会弥补这些不足.

7.4.1 Bézier曲线的de Castéjau定义

Bézier曲线的出发点是利用控制点给出的控制多边形产生曲线,期望获得的曲线的几何形状可以通过控制多边形的形状直观的得以控制.考察折线段曲线的生成, 曲线上某一点是借助于相邻的两个控制点产生的,具体由参数指定的直线的上的比例分位点给出. 这时我们把相邻的两个控制点作为一组产生一段特殊形式的曲线–直线段. 如果我们把相邻的三个控制点作为一组产生一段曲线又该如何做呢?

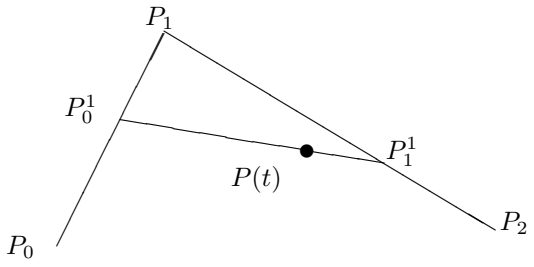


图7.4 三个控制点给出的Bézier曲线上点的生成方法

借助于折线段求点的方法,我们将得到两组相邻的两个控制点,产生出参数指定的两个直线段上的比例分位点. 为了产生出参数对应的曲线上的一个点, 我们不妨再利用一次这个方法,借助的直线段就是刚刚得到的两个分位点产生的直线段. 这一过程如图所示,可用数学的语言描述如下.

我们首先有顺序给出的图7.4所示的三个控制点P_0, P_1, P_2; 对参数$t \in [0,1]$, 要生成的曲线上的一点$P(t)$由如下公式给出:

$$P_0^1 = (1-t)P_0 + tP_1, \quad P_1^1 = (1-t)P_1 + tP_2 \tag{7.4.1}$$

$$P(t) = P_0^2 = (1-t)P_0^1 + tP_1^1 = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2 \tag{7.4.2}$$

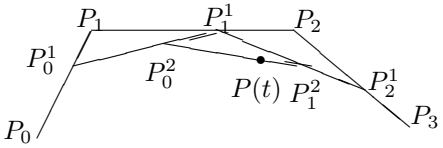


图7.5 四个控制点给出的Bézier曲线上点的生成方法示意图

重复上述方法,对图7.5中顺序给出的四个控制点P_0,P_1,P_2和P_3; 参数$t \in [0,1]$, 要生成的曲线上的一点$P(t)$由如下公式给出:

$$P_0^2 = (1-t)P_0^1 + tP_1^1 = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2 \tag{7.4.3}$$

$$P_1^2 = (1-t)P_1^1 + tP_2^1 = (1-t)^2P_1 + 2(1-t)tP_2 + t^2P_3 \tag{7.4.4}$$

$$\begin{aligned} P(t) &= P_0^3 = (1-t)P_0^2 + tP_1^2 \\ &= (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3 \end{aligned} \tag{7.4.5}$$

此种方式产生的曲线就是Bézier曲线,是由Bézier首先发表的. Bézier曲线的上述几何作图的定义方法是由de Castéjau给出的, 故称为Bézier曲线的de Castéjau定义.

反复重复上述方法,对顺序给出的$n+1$个控制点P_0, P_1, P_2, ..., P_n 和参数$t \in [0,1]$,生成的n次Bézier曲线上的点$P(t)$由如下公式给出:

$$P(t) = \sum_{i=0}^{n} B_{i,n}(t)P_i \tag{7.4.6}$$

其中

$$B_{i,n}(t) = \frac{n!}{(n-i)!i!}(1-t)^n t^i, \quad i = 0, 1, ..., n \tag{7.4.7}$$

为n次伯恩斯坦多项式.

7.4.2 Bézier曲线的性质

根据Bézier曲线的定义，一次Bézier曲线就是直线段. 根据Bézier曲线的数学表示式也不难看出二次Bézier曲线是一抛物线段. 三次或三次以上Bézier曲线的几何形状可以较为复杂。这些Bézier曲线具有很多有用的性质，这些性质取决于其生成方法及伯恩斯坦基函数的性质. 这些性质有：

(1) 端点插值性质

Bézier曲线首末端点正好是控制多边形的首末顶点，即$P(0)=P_0$、$P(1)=P_n$。这使我们能能简单的控制Bézier曲线的起点和终点。

(2) 端点导向量性质

Bézier曲线在首末端点的k阶导向量分别与Bézier多边形的首末k条边有关，与其它边无关。根据Bézier曲线的数学公式

$$P'(u)=n\sum_{i=0}^{n-1}P_i[B_{i-1,n-i}(u)-Bi,n-1(u)] \tag{7.4.8}$$

所以，在起点$u=0$和终点$u=1$处：

$$P'(0)=n(P_1-P_0),\quad P(1)=n(P_n-P_{n-1}) \tag{7.4.9}$$

亦即Bézier在首末端点处分别与控制多边形的首末条边相切，这使我们能直接控制Bézier曲线在首末端点处的切线。同样，因为

$$P''(u)=n(n-1)\sum_{i=0}^{n-2}(P_{i+2}-2P_{i+1}+P_i)B_{i,n-2}(u) \tag{7.4.10}$$

所以，在起点$u=0$和终点$u=1$处：

$$\begin{array}{l}P''(0)=n(n-1)(P_2-2P_1+P_0),\\ P''(1)=n(n-1)(P_n-2P_{n-1}+P_{n-2})\end{array} \tag{7.4.11}$$

(3) 整条Bézier曲线的对称性

根据Bézier曲线的生成方法, 反向排列控制点的顺序的得到的是同一条Bézier曲线，仅曲线方向相反. 反映在Bézier曲线的数学表示式中,就应该是用$(1-t)$代替t, 重新参数化得到的Bézier曲线. 这时我们可以看到有

$$\begin{array}{rcl}B_{i,n}(t) &=& \dfrac{n!}{i!(n-i)!}t^i(1-t)^{n-i}\\ &=& \dfrac{n!}{(n-i)!i!}(1-t)^{n-i}(1-(1-t))^i\\ &=& B_{n-i,n}(1-t)\end{array} \tag{7.4.12}$$

成立。这也就是说Bézier曲线的起点和终点具有相同的性质。

(4) 几何不变性

根据Bézier曲线的生成方法, Bézier曲线的生成不需要借助任何坐标系的选择,Bézier曲线的形状仅与控制多边形的顶点有关，而与坐标系的选择无关. 控制点的坐标只是在我们需要给出Bézier曲线的数学表示时才必须引入的. 要知道实际的曲线是没有坐标的,引入坐标(这却是完全因人而异的)只是为了处理的方便. Bézier曲线几何不变性保证在控制点不变时,生成的Bézier曲线不会因坐标选择的的不同而不同.

(5) 凸包(convex hull)性

Bézier曲线一定落在由其控制顶点组成的凸包内。凸包是指包围一组点集的最小凸多边形或凸多面体。可以这样来想象确定平面上点集的凸包：

在点集的每个点上打上钉子，然后用一根封闭的弹性橡皮绳套在所有钉子的外面，橡皮绳因弹性自然收缩形成封闭的多边形区域，这个多边形区域就是该点集张成的凸包；对于空间中的一组点集，也可以想象用一封闭的弹性橡皮薄膜包住这些点，因弹性收缩所形成的空间区域即为其凸包。

包围一组点集的最小凸包集也可这样生成:对最初的点集连接任意两点生成直线段,从而形成一个包含所有这些直线段上点的扩充的集合;对新的点集合反复进行扩充, 最终得到的就是包围最初给定点集的最小凸包集.根据Bézier曲线的生成方法, Bézier曲线的点就是这样的直线段上的点,因而Bézier曲线一定落在由其控制顶点组成的凸包内。这一性质对Bézier曲线所进行的一些处理如判断两Bézier曲线是否相交等特别有用:如果两Bézier曲线的控制多边形形成的凸包不相交,则它们必然不相交.另外,它可以保证曲线求解算法的收敛性:计算机处理的曲线上的点的坐标值总是在一个可预先确定的范围内.

(6) 变差缩减性质

任意平面与Bézier曲线的交点数目不会超过平面与其$n+1$个顶点构成的控制多边形的交点数目. 如果Bézier曲线在一平面内, 这一性质可表述为平面内任意直线与Bézier曲线的交点数目不会超过直线与其$n+1$个顶点构成的控制多边形多边形的交点数目.这一性质表明Bézier曲线比它的$n+1$个顶点构成的控制多边形的波动更小,即看起来更光顺.

(7) Bézier曲线的保凸性

若平面Bézier曲线的$n+1$个顶点构成的控制多边形为凸多边形，则对应的n次Bézier曲线也是凸的。这也就是说Bézier曲线上没有拐点和奇点。

(8) 控制顶点对Bézier曲线的影响

如果在交互设计曲线时，移动n次Bézier曲线的第i个控制顶点P_i，产生一个位移向量Δ_i则此时新的Bézier曲线$P^*(t)$应是

$$P^*(t) = P(t) + \Delta_i B_{i,n}(t)(3110) \tag{7.4.13}$$

因此有

$$\max_{0\le t\le 1}|P^*(t)-P(t)| = \max_{0\le t\le 1}|\Delta_i B_{i,n}(t)| = |\Delta_i| \max_{0\le t\le 1} B_{i,n}(t) = |\Delta_i| B_{i,n}(\frac{i}{n}) \tag{7.4.14}$$

即对曲线上参数$t=\dfrac{i}{n}$的那一点$P(\dfrac{i}{n})$处发生最大的影响。反之，欲使曲线$P(t)$在$P(\dfrac{i}{n})$处移动ΔP, 则可使顶点P_i偏移一个向量

$$\Delta_i = \Delta P / B_{i,n}(\frac{i}{n}) \tag{7.4.15}$$

这一性质对交互设计Bézier曲线是很有用的，同时也告诉我们，Bézier曲线不会远离控制顶点而作病态的振荡。

(9) Bézier曲线的几何形状

Bézier曲线的表达式使它可以表示多值的形态，如首尾控制顶点重合时，曲线便是闭合的, 并且根据Bézier曲线端点切向量的公式,只要P_1,

$P_0 = P_n$, P_{n-1}三点共线,则闭合的Bézier曲线在端点也是光滑的。另一种情况是有顺序的几个点相重合，这时的曲线朝向这个相重合的控制点靠近。值得指出的是当控制多边形自交时, 也可以产生自交的Bézier曲线. 总之, 控制多边形的形状大致上直观地反映了的Bézier曲线的形状. 但两者毕竟还是有所不同, 特别是较高阶的Bézier曲线更是如此. 我们直观上想象成一条粘在首尾控制顶点间的磁化了的弹性带，磁铁位于所有的控制点上。因此，带子被磁铁吸向每一点。磁铁密集的地方，即相重点较多的地方，带子就被吸引得更靠近那。当控制点较多时,如此相互作用的结果就比较复杂,控制多边形与Bézier曲线的形状就会有较大的差别. 后面的章节我们会对此有更深入的分析.

7.4.3 三次Bézier曲线与Bézier样条曲线

由于一整段的Bézier曲线的次数为控制点的个数−1, 随着控制点个数的增加, 一整段的Bézier曲线会是次数很高的参数曲线，为了避免高次曲线固有的麻烦，工程上运用时，往往采用分段的低次Bézier样条曲线组合出整段的曲线。分段组合Bézier曲线时各段间按一定的连续性要求连接起来，这样的样条曲线更加实用于几何外形曲线的设计。我们知道,一次Bézier曲线是直线段,二次Bézier曲线是抛物线. 因此都不能用来表示复杂形状的曲线.而三次Bézier曲线表示出的形状已经比较丰富,从曲线表示形式应尽可能比较简单来考虑,一般情况下选用C^2 连续的分段三次Bézier曲线.三次Bézier曲线有许多特有的性质,我们较详细的介绍于后.

7.4.3.1. 三次Bézier曲线的矩阵表示

三次Bézier曲线用4个控制顶点给出的曲线的表达式为

$$P(t) = \sum_{i=0}^{3} B_{i,3}(t)P_i = (1-t)^3P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3 \quad (7.4.16)$$

由

$$[B_{0,3}(t)\ \ B_{1,3}(t)\ \ B_{2,3}(t)\ \ B_{3,3}(t)] = [1\ \ t\ \ t^2\ \ t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad (7.4.17)$$

不难得到其类似于参数三次样条曲线的矩阵表示形式如下

$$P(t) = [1\ \ t\ \ t^2\ \ t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad (7.4.18)$$

把上面的表达式又简写为

$$P(t) = [1\ \ t\ \ t^2\ \ t^3]MB \quad (7.4.19)$$

B也称为几何系数矩阵，就是控制多边形的顶点组成的，M称为通用变换矩阵。当曲线阶次不一样时，矩阵M也不一样。

三次Bézier曲线是参数的三次多项式曲线,这一点和前面介绍的参数三次多项式曲线是相同的. 这两种曲线实际上只是表示形式不同, 它们可以互相表示出

来.

$$\begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{7.4.20}$$

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & 0 \\ 0 & 1 & 0 & \frac{1}{3} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix} \tag{7.4.21}$$

用幂基形式(7.3.1)表示的参数的三次多项式曲线也可用三次Bézier曲线的形式表示出来, 这只需注意有

$$[1 \;\; t \;\; t^2 \;\; t^3] = [B_{0,3}(t) \;\; B_{1,3}(t) \;\; B_{2,3}(t) \;\; B_{3,3}(t)] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{3} & 0 & 0 \\ 1 & \frac{2}{3} & \frac{1}{3} & 0 \\ -1 & 1 & 1 & 1 \end{bmatrix} \tag{7.4.22}$$

即可.由此容易看出不同基表示形式下系数向量间的关系。

如前所指，幂基形式的多项式曲线几何意义不明显，并且在数值上不如伯恩斯坦基形式稳走。作为数值上不稳定的一个结果，幂基形式用来表示曲线与曲面就不是很可靠的。这是由于幂基形式本质上是曲线段在始点$P(0)$邻近的泰勒展开，越远离始点，积累的舍入误差将越大，且随着曲线次数的升高，问题将戏剧性地变得越突出。因此，如果出于计算稳定原因，满可不必将伯恩斯坦基形式变换成幂基形式。但在幂基形式下，采用嵌套乘法格式计算曲线上一串点时，其速度要比采用德卡斯特里奥算法快得多。究竟采用那种方法计算，应视实际需要与实际条件决定。

7.4.3.2. 三次Bézier曲线的凸性性质

若平面Bézier曲线的$n+1$个顶点构成的控制多边形为凸多边形，则对应的n次Bézier曲线也是凸的。但反过来,则不一定成立. 当$n = 1, 2$时, 控制多边形总是凸的, 相应的Bézier曲线也总是凸的($n = 1$时为直线段, $n = 2$时为抛物线). 当$n \geq 3$才可能有非凸的Bézier曲线. 我们有如下结论:

三次Bézier曲线凸的充分必要条件是它的控制多边形为凸多边形.

对于$n = 4$,下图所示的凸的Bézier曲线有非凸的控制多边形生成.

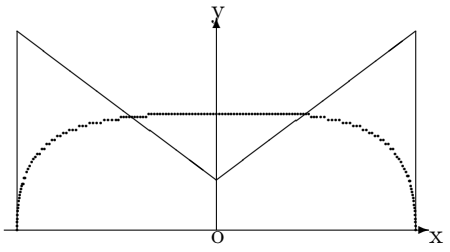


图7.6 非凸的控制多边形生成凸的Bézier曲线

7.4.3.3. 三次Bézier曲线的光滑拼接

下面来分析三次Bézier曲线各在端点处的光滑连续性连接条件。光滑连接时，切向量应方向一致. 设给定了两个控制多边形$P_0P_1P_2P_3$ 和$Q_0Q_1Q_2Q_3$，要求由它们所定义的Bézier曲线段在连接点P_3和Q_0连续或光滑拼接。

如果只是要求两条曲线的端点重合, 只需下式成立即可.

$$P_3 = Q_0 \tag{7.4.23}$$

要求两条曲线在连接点处不仅重合, 且要没有形成尖角,则可要求导向量相等,即有：

$$3(P_3 - P_2) = 3(Q_1 - Q_0) \tag{7.4.24}$$

如果注意到要没有形成尖角只需在连接点处有共同的切线和导向量同向即可,则只需有下式成立:

$$(P_3 - P_2) = \alpha(Q_1 - Q_0), \alpha > 0. \tag{7.4.25}$$

此式的几何意义就是要求P_2, $P_3 = Q_0$和Q_1三点共线，且P_2和Q_1 在$P_3 = Q_0$不同侧. (7.4.24)是此式在$\alpha = 1$时的特例.

要达到C^2-连续性，必须要求二阶导数连续，即下式成立：

$$6(P_3 - 2P_2 + P_1) = 6(Q_2 - 2Q_1 + Q_0)(3 - 121) \tag{7.4.26}$$

结合(7.4.25), 化简可得

$$Q_2 - Q_1 = P_1 - P_2 + (1 + \alpha)(P_3 - P_2) \tag{7.4.27}$$

此式子的几何意义是要求Q_2,Q_1两点在P_1,P_2和P_3三点所决定的平面上. 如果$\alpha = 1$,则有

$$Q_2 - P_1 = 2(Q_1 - P_2) \tag{7.4.28}$$

即直线段Q_2P_1与Q_1P_2平行，且长度为它的两倍。

这样，曲线的设计者为得到一阶连续性而定位控制点就比较容易。实际上，为了保持高阶导数的连续性，只要增加曲线的次数，也就是控制点的数量，这在构造曲线时是容易做到的。

7.5 B-样条曲线

如上节所述Bézier曲线由许多优点.但仍有不足.在三次参数多项式曲线时, 我们看到参数取值范围的变化对三次参数多项式曲线的几何形状有影响,这一问题Bézier曲线依然存在.另外我们知道Bézier曲线作为一类多项式曲线, 其次数会随着控制点的增加而增加.因此不难设想如果我们有无限多个控制点时, 我们无法产生一条Bézier曲线.下面介绍的B-样条曲线将弥补这些不足.

7.5.1 B-样条曲线的de Bool定义

B-样条曲线的出发点仍是利用控制点给出的控制多边形产生曲线,以其获得的曲线的几何形状可以通过控制多边形的形状直观的得以控制. 但与定义Bézier曲线不同的是：

(1). 我们假设有无限多个控制点组成的控制点序列

$$\{P_i : \ i = \ ..., \ -2, \ -1, \ 0, \ 1, \ 2, \ ...\} \tag{7.5.1}$$

(2). 为了克服参数取值范围对曲线的影响, 我们主动把参数取值的因素引入到曲线的设计中来,以便考察这种影响,设计出更合理的曲线.具体的做法是对每一个控制点P_i引入一个参数值u_i,称为节点值,于是有节点值序列

$$\{u_i:\ i=\ ...,\ -2,\ -1,\ 0,\ 1,\ 2,\ ...\} \tag{7.5.2}$$

考察在参数区间$t\in[u_j,u_{j+1}]$上曲线段的生成. 如果只是借助于相邻的两个控制点P_j和P_{j+1}产生曲线段, 只要生成折线段曲线即可,曲线段上某一点,具体由参数指定的直线的上的比例分位点给出.这时我们把相邻的两个控制点作为一组产生一段特殊形式的曲线–直线段.此时两个控制点产生直线段的公式:

$$P(t)=\frac{t-u_j}{u_{j+1}-u_j}P_j+\frac{u_{j+1}-t}{u_{j+1}-u_j}P_{j+1},\quad t\in[u_j,u_{j+1}] \tag{7.5.3}$$

如果我们把相邻的三个控制点P_{j-1}, P_j和P_{j+1}作为一组产生一段曲线又该如何做呢? 同生成Bézier曲线时一样,可以借助于折线段求点的方法,我们将得到两组相邻的两个控制点,产生出指定的两个直线段上的比例分位点.这里需要注意到与生成Bézier曲线时不同的是此时参数的取值是有一定限制的.对于给定的$t\in[u_j,u_{j+1}]$, 该如何利用P_{j-1}, P_j产生出一点?要知道这两点处限定的参数取值为$[u_{j-1},u_j]$.一个直接的方法就是把两个参数区间放在一起来考虑直线段上的比例分位点,也就是在$[u_{j-1},u_{j+1}]$中来考虑某个给定的$t\in[u_j,u_{j+1}]$的比例分位点.于是有利用P_{j-1}, P_j产生出点为

$$P_j^{(1)}(t)=\frac{t-u_{j-1}}{u_{j+1}-u_{j-1}}P_{j-1}+\frac{u_{j+1}-t}{u_{j+1}-u_{j-1}}P_j,\quad t\in[u_j,u_{j+1}] \tag{7.5.4}$$

利用P_j, P_{j+1}产生出$t\in[u_j,u_{j+1}]$对应的比例分位点时,应考虑到与产生前一点的对称性,在参数区间$[u_j,u_{j+2}]$内考虑$t\in[u_j,u_{j+1}]$ 对应的比例分位点,为

$$P_{j+1}^{(1)}(t)=\frac{t-u_j}{u_{j+2}-u_j}P_j+\frac{u_{j+2}-t}{u_{j+2}-u_j}P_{j+1},\quad t\in[u_j,u_{j+1}] \tag{7.5.5}$$

为了产生出参数对应的曲线上的一个点, 我们不妨再利用一次这个方法,借助的直线段就是刚刚得到的两点产生的直线段,只是此时参数的取值范围为前一步涉及到的两个参数区间的公共部分$[u_j,u_{j+1}]$,得到的$t\in[u_j,u_{j+1}]$对应的比例分位点为

$$\begin{aligned}P(t)&=P_j^{(2)}(t)\\&=\tfrac{t-u_j}{u_{j+1}-u_j}P_j^{(1)}(t)+\tfrac{u_{j+1}-t}{u_{j+1}-u_j}P_{j+1}^{(1)}(t),\quad t\in[u_j,u_{j+1}]\end{aligned} \tag{7.5.6}$$

类似地, 如果我们可以把相邻的四个控制点P_{j-2},P_{j-1},P_j和P_{j+1}作为一组产生一段曲线.对于给定的$t\in[u_j,u_{j+1}]$, 利用P_{j-2},P_{j-1}产生出一点时就在完整的参数区间为$[u_{j-2},u_{j+1}]$ 中来考虑某个给定的$t\in[u_j,u_{j+1}]$的比例分位点.于是有利用P_{j-2},P_{j-1}产生出点为

$$P_{j-1}^{(1)}(t)=\frac{t-u_{j-2}}{u_{j+1}-u_{j-2}}P_{j-2}+\frac{u_{j+1}-t}{u_{j+1}-u_{j-2}}P_{j-1},\quad t\in[u_j,u_{j+1}] \tag{7.5.7}$$

利用P_{j-1},P_j产生出一点就在完整的参数区间为$[u_{j-1},u_{j+2}]$ 中来考虑某个给定的$t\in[u_j,u_{j+1}]$的比例分位点.于是有利用P_{j-1},P_j产生出点为

$$P_j^{(1)}(t)=\frac{t-u_{j-1}}{u_{j+2}-u_{j-1}}P_{j-1}+\frac{u_{j+2}-t}{u_{j+2}-u_{j-1}}P_j,\quad t\in[u_j,u_{j+1}] \tag{7.5.8}$$

利用P_j,P_{j+1}产生出一点就在完整的参数区间为$[u_j, u_{j+3}]$ 中来考虑某个给定的$t \in [u_j, u_{j+1}]$的比例分位点,为

$$P_{j+1}^{(1)}(t) = \frac{t-u_j}{u_{j+3}-u_j}P_j + \frac{u_{j+3}-t}{u_{j+3}-u_j}P_{j+1}, \quad t \in [u_j, u_{j+1}] \tag{7.5.9}$$

重复上述过程,利用$P_{j-1}^{(1)}(t)$和$P_j^{(1)}(t)$产生一点时参数的取值范围为前一步涉及到的两个参数区间的公共部分$[u_{j-1}, u_{j+1}]$. 得到的$t \in [u_j, u_{j+1}]$对应的比例分位点为

$$P_{j-1}^{(2)}(t) = \frac{t-u_{j-1}}{u_{j+1}-u_{j-1}}P_{j-1}^{(1)}(t) + \frac{u_{j+1}-t}{u_{j+1}-u_{j-1}}P_j^{(1)}(t), \quad t \in [u_j, u_{j+1}] \tag{7.5.10}$$

利用$P_j^{(1)}(t)$和$P_{j+1}^{(1)}(t)$产生一点时参数的取值范围为前一步涉及到的两个参数区间的公共部分$[u_j, u_{j+2}]$. 得到的$t \in [u_j, u_{j+1}]$对应的比例分位点为

$$P_j^{(2)}(t) = \frac{t-u_j}{u_{j+2}-u_j}P_j^{(1)}(t) + \frac{u_{j+2}-t}{u_{j+2}-u_j}P_{j+1}^{(1)}(t), \quad t \in [u_j, u_{j+1}] \tag{7.5.11}$$

为了产生出参数对应的曲线上的一个点,我们再利用一次这个方法,借助的直线段就是刚刚得到的两点产生的直线段,只是此时涉及的参数的取值范围为前一步涉及到的两个参数区间的公共部分$[u_j, u_{j+1}]$. 得到的$t \in [u_j, u_{j+1}]$对应的比例分位点为

$$\begin{aligned} P(t) &= P_j^{(3)}(t) \\ &= \tfrac{t-u_{j-2}}{u_{j+3}-u_{j-2}}P_j^{(2)}(t) + \tfrac{u_{j+3}-t}{u_{j+3}-u_{j-2}}P_{j+1}^{(2)}(t), \quad t \in [u_j, u_{j+1}] \end{aligned} \tag{7.5.12}$$

一般的,我们可以通过k-个控制点P_{j-k+1},P_{j-k+2},...,P_j 和P_{j+1}定义一段曲线$P(t)$, $t \in [u_j, u_{j+1}]$. 其定义公式为

$$\begin{cases} P_i^{(0)} = P_i \\ P_i^(l) = (1-\alpha_i^{(l)})P_{i-1}^{(l-1)} + \alpha_i^{(l)}P_i^{(l-1)} \\ \alpha_i^{(l)} = \dfrac{t-u_{j-1}}{u_{j+k+l-2}-u_{j-1}} \\ \qquad l = 1, 2, ..., k, \quad i = j-k+l+1, ..., j+1 \end{cases} \tag{7.5.13}$$

而

$$P(t) = P_j^k, \quad t \in [u_j, u_{j+1}]. \tag{7.5.14}$$

需要注意的是节点值的选取. 因为节点值实际上要用来限定参数的取值范围, 我们不应该选取时而增加,时而减少的节点值,以免造成同一参数值处曲线不唯一的困难. 因此节点值应是单调变化的. 但单调增加或减小并没有本质的区别, 习惯上认为节点值应是非单调减小的, 即有

$$..... \le u_2 \le u_{-1} \le u_0 \le u_1 \le u_2 \le ..... \tag{7.5.15}$$

另外还需要注意的是出现节点值相等的情况. 这时运算公式将出现零分母,无法正常运算.但考虑到曲线求解的几何过程仍是有意义的, 并且与如下规定的结果相符:$\frac{0}{0} = 0$.附加此规定后,上述公式就总是可以计算的了.

由此,我们也规定相邻的节点如果相等,就称为重节点,它们相等的最大的次数就称为该节点的重数.相应的,也规定相邻的控制点如果相等,就称为重点.相应地

也有重点的重数.值得一提的是如果不是相邻的控制点是相同的,这两个控制点不认为是重点,除非它们之间的所有控制点都与其相等.

如果我们把所有控制点的下标减去一,同时保持所有节点的下标不变, 则有由$k+1$-个控制点P_{j-k}, P_{j-k+1}, ..., P_{j-1} 和P_j定义一段曲线$P(t)$, $t \in [u_j, u_{j+1}]$, 其定义公式为

$$\begin{cases} P_i^{(0)} = P_i \\ P_i(l) = (1-\alpha_i^{(l)})P_{i-1}^{(l-1)} + \alpha_i^{(l)} P_i^{(l-1)} \\ \alpha_i^{(l)} = \dfrac{t-u_{j-1}}{u_{j+k-l-2}-u_{j-1}} \\ \qquad l = 1,2,...,k, \quad i = j-k+l,...,j \end{cases} \tag{7.5.16}$$

而

$$P(t) = P_j^k, \quad t \in [u_j, u_{j+1}] \tag{7.5.17}$$

这就是通常的B-样条曲线的de Bool公式.

根据B-样条曲线的公式, B-样条曲线可以有如下形式的表示

$$P(t) = \sum_{j=-\infty}^{+\infty} N_{j,k}(t) P_j \tag{7.5.18}$$

其中的$N_{j,k}(t)$就是依据Schoenberg的B-样条理论定义的B-样条基函数. B-样条曲线通常就是由B-样条基函数按这一公式而得到的,B-样条曲线的de Bool公式当然也是由此得到的.

注解一: 依据B-样条曲线的递推公式,如果所有节点值扩大缩小倍数a, 增加一个常数b,则对$t \in [u_j, u_{j+1}]$,有$at+b \in [au_j+b, au_{j+1}+b]$, 并且

$$\alpha_i^{(l)} = \frac{t-u_{j-1}}{u_{j+k-l-2}-u_{j-1}} = \frac{(at+b)-(au_{j-1}+b)}{(au_{j+k+l-2}+b)-(au_{j-1}} + b) \tag{7.5.19}$$

因此递推公式(7.5.17)没有任何改变,说明所有节点值扩大缩小倍数a, 增加一个常数b前后生成的是同一条B-样条曲线. 所以B-样条曲线的节点参数值通常可以取$u_0 = 0$而不失一般性。如果是有限个控制点和节点,u_0通常也是起始节点。

注解二: 依据B-样条曲线的递推公式,如果在递推过程中始终有

$$\alpha_i^{(l)} = \frac{t-u_{j-1}}{u_{j+k-l-2}-u_{j-1}} = t \tag{7.5.20}$$

成立,则上述递推公式与前一节中定义Bézier曲线的递推公式是完全一致的, 这时的B-样条曲线段实际上也就是一段Bézier曲线.

关于B-样条函数理论本身,我们不作进一步的介绍,这方面的内容在关于样条理论以及许多介绍B-样条曲线可以查阅得到. 下面一节主要介绍B-样条曲线的性质.

7.5.2 B-样条曲线的性质

在实际的问题中,一般只有有限个控制点.定义一段k阶B-样条曲线至少需要$k+1$个控制点. 控制点和节点的下标一般从0开始.

在下面的讨论中,若无特别声明,就是讨论参数区间$t \in [u_j, u_{j+1}]$的一段B-样条曲线$P_j(t)$,它由$k+1$-个控制点P_{j-k}, P_{j-k+1}, ..., P_{j-1}和P_j生成.

(1). B-样条曲线的局部性质与定义域

第j段k次B样条曲线$P_j(t)$由$k+1$-个控制点P_{j-k}, P_{j-k+1}, ..., P_{j-1} 和P_j生成,与其它控制点无关。我们知道，Bézier曲线上除两端点外的所有点都与控制多边形的全部顶点有关。另外,与$P_j(t)$有关的节点值为u_{j-k+1}, ..., u_j, u_{j+1}, ..., u_{j+k}共$2k$个. 这就是B-样条曲线的局部性质. 于是，往下标减少方向错过一个顶点，即P_{j-k-1}, P_{j-k}, ..., P_{j-2} 和P_{j-1}那$k+1$ 个顶点就定义了上一曲线段。往下标增加方向错过一个顶点，即P_{j-k+1}, P_{j-k+2}, ..., P_j和P_{j+1}那$k+1$个顶点就定义了下一曲线段。它们的定义区间分别就是上一个与下一个节点区间$[u_{j-1}, u_j]$与$[u_{j+1}, u_{j+2}]$。由此可见增加或减少一个控制顶点，在不含重节点情况下，整条样条曲线相应增加或减少了一曲线段。

从另一方面看，如果我们移动k次B-样条曲线的一个控制顶点也将对曲线有怎样的影响范围? 变动顶点P_j, 由B-样条曲线段的定义, 将使如下参数区间上的曲线段发生变化:

$$[u_j, u_{j+1}],\ [u_{j+1}, u_{j+2}],\ .....,\ [u_{j+k}, u_{j+k+1}] \tag{7.5.21}$$

这些参数区间是前后相邻的, 因此可合并为一个参数区间$[u_j, u_{j+k+1}]$. 于是，我们得到结论: 移动第j个控制顶点至多影响到定义在区间$[u_j, u_{j+k+1}]$上那部分曲线，对B-样条曲线的其它部分将不发生影响。而我们知道，移动Bézier曲线的一个控制顶点，将影响整条曲线的形状。

现在我们可以给出关于B-样条曲线的局部性质的完整表述: k次B-样条曲线上参数为$t \in [u_j, u_{j+1}]$的一点$P(t)$ 至多与$k+1$个控制顶点P_i, $i = j-k, j-k+1$, ..., j和节点u_i, $i = j-k+1$, $j-k+2$, ..., $j+k$ 有关，与其它控制顶点无关; 移动该曲线的第j个控制顶点P_j至多将影响到定义在区间$[u_j, u_{j+k+1}]$上那部分曲线的形状，对曲线的其余部分不发生影响。其中前者所用“至多”是考虑到其中包括重顶点和重节点的情形。后者所用“至多”是考虑到下面要讨论的可能有些节点区间中并不能定义B-样条曲线。

下面讨论B样条曲线的定义域. 如果我们有下标从$-\infty$到∞的无限多个控制点和节点, 显然在每个节点区间上都可以定义一段B-样条曲线(对于重节点,节点区间就收缩为一个点).对于有限个控制点的情形,结果是不同的.

设给定$n+1$个控制顶点P_i, $i = 0, 1, ..., n$，则第一段k次B样条曲线段应由控制顶点P_i, $i = 0, 1, ..., k$定义,所需的节点为u_i, $i = 1, 2, ..., 2k$,定义在节点区间$[u_k, u_{k+1}]$上。最后一段k次B样条曲线段应由控制顶点P_i, $i = n-k, n-k+1, ..., n$定义, 所需的节点为u_i, $i = n-k, n-k+1, ... n+k+1$,定义在节点区间$[u_n, u_{n+1}]$上。于是，给定$n+1$个控制顶点$P_i$, $i = 0, 1, ..., n$，$n+k+1$个节点u_i, $i = 0, 1, ..., n+k+1$，我们得到k次B-样条曲线的定义域的参数区间为

$$t \in [u_k, u_{n+1}] \tag{7.5.22}$$

共含有$n-k+1$个节点区间(包括零长度的节点区间)，若其中不含重节点，则对应B-样条曲线包含$n-k+1$段。

如果我们不额外增加节点,即要求节点与控制点是相互对应的, 则对应B-样条曲线只有$n-2k+1$段。

(2). B样条曲线的可微性或参数连续性

B-样条曲线在每一曲线段内部是无限次可微的即是C^{∞}连续的，在对应节点的曲线段端点处是$k-r$次可微的，即是C^{k-r}连续的，r是该节点的重复度。

(3). 比Bézier曲线更强的凸包性质

B-样条曲线的凸包是定义各曲线段的$k+1$个控制顶点的凸包的并集。这样一来，其凸包区域就比同一组$n+1$个控制点定义Bézier曲线的凸包区域要小，至多相同。B样条曲线恒位于它的凸包内。凸包性质保证顺序$k+1$个顶点重合时，由该$k+1$个顶点定义的B样条曲线段退化到这一重合点; 如果$k+1$-个控制点在一条直线上, 由该$k+1$个顶点定义的B样条曲也是一条直线段.根据一段B-样条曲线的定义,这一性质是非常明显的.

(4). 变差减少性质（即VD性质）

这是对B 样条曲线另一个重要性质。由此可见，B样条曲线保留了Bézier曲线的这一优良性质。

(5). 磨光性质

同一组控制顶点定义的B-样条曲线，随着次数k升高，越来越光滑。这可由B-样条曲线的可微性看出。

(6). 几何不变性

这一性质与Beizer完全类似,B-样条曲线的生成不依赖于坐标系的选取.

(7). B-样条曲线的保凸性

若平面B-样条曲线的$n+1$个顶点构成的控制多边形为凸多边形，则对应的B-样条曲线也是凸的。即此时，B-样条曲线上没有拐点和奇点。

(8). 重顶点重节点对B-样条曲线的影响

前面已经提到,顺序$k+1$个顶点重合即有k重顶点时，由该重顶点定义的B-样条曲线段退化到这一个点. 如果节点为重点时该如何?

在B-样条曲线定义域内的内重节点，重复度每增加1，曲线段数减1，B-样条曲线在该重节点处的可微性或参数连续阶降1。因此，k次B-样条曲线在重复度为r的节点处是C^{k-r}连续的。一条位置连续的曲线，其内节点所取的最大重复度等于曲线的次数k 端节点的最大复度为$k+1$。更高的节点重复度就会产生间断不连续的B-样条曲线.依据这一性质，可以在B-样条曲线内部构造尖点与尖角（注意与重顶点构造尖角的区别）。甚至两条或多条分离的B-样条曲线可以采用一个统一的方程表示。

当端节点重复度为k时，k次B-样条曲线段的端点将与相应的控制多边形的顶点相重，并在端点处与控制多边形相切,即以重点出相应的边为端点处的切线。

当在曲线定义域内有重复度为k的节点时,k次B-样条曲线插值于相应的控制顶点。与设置k重顶点达到插值顶点不同之处在于，此时不引起曲线在该点处切向量消失，保持曲线的正则性。

当端节点重复度为$k+1$时，k次B-样条曲线就具有和k 次Bézier曲线相同的端点几何性质。

k次B-样条曲线若在定义域内相邻两节点都具有重复度k，可以生成定义在该节点区间上那段B-样条曲线的Bézier点。

若端节点重复度为$k+1$的k次B-样条曲线的定义域仅有一个非零节点区间，则所定义的该k次B-样条曲线就是k次Bézier曲线。由此可知，Bézier曲线是B-样条曲线的特例，B-样条方法是Bézier方法的合适的强有力的推广。

7.5.3 常用的B-样条曲线类型

根据B-样条曲线的定义(7.5.14)或者(7.5.17),B-样条曲线一般情况下不仅依赖于控制点的选取,也依赖于节点值的选取.下面我们首先来讨论节点值的选取对B-样条曲线的一些影响.

很显然一次B-样条曲线就是控制多边形自身给出的折线段曲线，与所取节点值无关.现在我们考虑高于一次的B-样条曲线按节点取值的分类，当然对于一次情况也仍然适用，只不过没有意义罢了。

7.5.3.1. 均匀B-样条曲线（uniform B-spline curve）

节点取值为沿参数轴均匀或等距分布，所有节点区间长度$\Delta_j = u_{j+1} - u_j > 0$为常数。这样的节点取值定义出均匀B-样条曲线。当我们选取参数$t = u_j$时,由递推公式(7.5.17)不难看出只要控制顶点互不相同,最后推得的参数区间$[u_j, u_{j+1}]$上的B-样条曲线段的端点一般不等于P_j或其它任何一个相关的控制点. 特别地, 当定义k次B-样条曲线且$j = k$时,就说明均匀B-样条曲线的端点一般情况下不是控制多边形的端点. 这一点与Bézier曲线是不同的. 根据节点值乘以一个常数$(\frac{1}{\Delta_j})$ 加上一个常数$(-u_0)$B-样条曲线不变的性质，均匀B-样条曲线的节点值可以认为是$u_j = j$而不影响其一般性。

7.5.3.2. 准均匀B-样条曲线（quasi-uniform B-spline curve）

如前所述,均匀B-样条曲线的端点一般情况下不是控制多边形的端点. 要想生成端点是控制多边形的端点的B-样条曲线,根据重节点的性质,只要首末两端节点具有重复度$k+1$，即$u_0 = u_1 = ... = u_k$,$u_{n+1} = u_{n+2} = ... = u_{n+k+1}$, 其它节点可以任意取值.当然, 我们可让其它节点均匀分布，具有重复度1，参数定义域为$[u_k, u_{n+1}]$,应均匀分成$n-k$段. 节点这样取值的B-样条曲线与均匀B-样条曲线定义域内节点分布相同，差别仅在于两端节点。这样的节点取值定义了准均匀B-样条曲线。

7.5.3.3. 分段Bézier曲线(piecewise Bézier curve)

考察$k+1$个控制点定义的B-样条曲线在什么样的条件下可以构成一Bézier曲线.

首先它们的参数取值区间应该一致,于是有

$$[u_k, u_{k+1}] = [0, 1] \tag{7.5.23}$$

即$u_k = 0$, $u_{k+1} = 1$. 其次, 因为Bézier曲线的端点就是控制多边形的端点, 根据在定义准均匀B-样条曲线时的分析,两端点处节点应有重复度$k+1$. 这时所有的节点值都给定了: $u_0 = u_1 = ... = u_k = 0$, $u_{2k} = u_{2k-1} = ... = u_{k+2} = u_{k+1} = 1$. 最后, 我们来考察递推公式(7.5.17) 中的$\alpha_i^{(l)}$:

$$\alpha_i^{(l)} = \frac{t - u_{j-1}}{u_{j+k+l-2} - u_{j-1}} = t \tag{7.5.24}$$

因此, 根据递推公式(7.5.17)后面的注解, 这时生成的就是Bézier曲线.

如果我们有$2k+1$个控制点,就可生成在控制点P_k处相连接的两段Bézier 曲线,这只要控制点P_k处相应的节点具有重复度k即可.

一般情况下, 如果有$mk+1$个控制点,就可以定义m段分别在控制点P_k, P_{2k},...,$P_{(m-1)k}$处相连接的Bézier曲线. 相应的节点取值的方式是两端节点重

复度为$k+1$，其它节点重复度都为k. 比如两端节点可以取0, m,其它节点取1, 2, ..., $m-1$.

7.5.3.4.一般非均匀B-样条曲线（general non-unlform B-spline curve）

节点可任意取值，只要递推公式(7.5.17)在数学上成立(其中节点序列非递减，两端节点重复度不超过$k+1$，内节点重复度不超过k)。这样的节点取值定义了一般的非均匀B-样条曲线。前三种类型当然都可作为特例被包括在这种类型里。对于不认为是首尾相连的开曲线，通常为使其具有同Bézier曲线相同的端点几何性质，两端节点取值重复度为$k+1$。

至于其它的节点取值, 我们可以从考察一阶B-样条曲线, 即折线段曲线开始. 记$l_i = |P_i - P_{i-1}|$, 为P_{i-1}和P_i两点的距离, 由于在参数区间$[u_i, u_{i+1}]$上定义的是P_{i-1}和P_i两点之间的这线段, 显然可以认为$u_{i+1} - u_i = l_i$是比较合适的一种选择. 如果定义的是一k阶B-样条曲线, 相关的控制点是P_{i-k}, ..., P_k, 参数区间的长度不妨可采用采用相应控制多边形顺序k条边长的平均值, 即有

$$u_{i+1} - u_i = \frac{1}{k}\sum_{j=i-k}^{i-1} l_j, \quad i = k+1, k+2, ...n+1 \tag{7.5.25}$$

我们知道,整条B-样条曲线定义在参数区间$[u_k, u_{n+1}]$. 通常的习惯是取

$$[u_k, u_{n+1}] = [0, 1],$$

为此, 注意此时有

$$u_{n+1} - u_k = \frac{1}{k}\sum_{s=k+1}^{n+1}\sum_{j=i-k}^{i-1} l_j \tag{7.5.26}$$

对上述参数取值稍作修改即可得到满足要求的参数取值方法

$$u_{i+1} - u_i = \frac{\frac{1}{k}\sum_{j=i-k}^{i-1} l_j}{\frac{1}{k}\sum_{s=k+1}^{n+1}\sum_{j=i-k}^{i-1} l_j} = \frac{\sum_{j=i-k}^{i-1} l_j}{\sum_{s=k+1}^{n+1}\sum_{j=i-k}^{i-1} l_j} \tag{7.5.27}$$

于是有

$$u_i = \sum_{s=k+1}^{i}(u_s - u_{s-1}) = \frac{\sum_{s=k+1}^{i}\sum_{j=i-k}^{i-1} l_j}{\sum_{s=k+1}^{n+1}\sum_{j=i-k}^{i-1} l_j} \tag{7.5.28}$$

这个公式就是Hartley和Judd(1978)给出的节点值的选取公式.

7.5.4 均匀B-样条曲线

7.5.4.1. 均匀B-样条曲线的基函数表示

均匀B-样条曲线是比较简单,也最常用的的一种B-样条曲线。它具有计算简单,几何外形容易控制等优点.为方便起见，均匀B-样条曲线的节点值常取成$u_0 = 0$的整数序列值，于是有均匀节点取值公式

$$u_i = i, \quad i = 0, 1, ..., n+k+1 \tag{7.5.29}$$

相应曲线定义域为$t \in [u_k, u_{n+1}]$。

这一条B-样条曲线是有多段B-样条曲线段构成的, 我们首先考察定义在节点区间$[u_j, u_{j+1}] = [j, j+1]$的一段B-样条曲线段.为表示标准起见,我们想把参数取值限定在区间$[0,1]$上,根据递推公式(7.5.17)的注解,此时节点取值公式认为是

$$u_i = i - j, \quad i = 0, 1, ..., n+k+1 \tag{7.5.30}$$

即可. Clark(1985)推出了基函数表示公式(7.5.18)中基函数的一般表达式:

$$\begin{aligned} f_{k,i}(t) = \quad N_{i-k+i,k}(t) = \frac{1}{k!}\sum_{m=0}^{k-i}(-1)^m C_{k+m}^m (k-i-l+t)^k \\ t \in [0,1], \quad i = 0, 1, ..., k \end{aligned} \tag{7.5.31}$$

由此公式不难看出此时的基函数与标定节点区间$[u_j, u_{j+1}]$的j无关, 这说明当我们要计算由$k+1$个控制点P_{j-k}, ..., P_j生成的一段B-样条曲线时只要用如下公式即可:

$$P_j(t) = \sum_{i=0}^{k} f_{k,i}(t) P_{j-k+i} \tag{7.5.32}$$

或一般地, 任意给出$k+1$个控制点P_0, ..., P_k生成的一段B-样条曲线为

$$P(t) = \sum_{i=0}^{k} f_{k,i}(t) P_i \tag{7.5.33}$$

这一公式也说明, 当我们只是简单的去使用均匀B-样条曲线时, 可以不去关心节点值选取的问题, 甚至不必知道B-样条曲线递推公式就可以直接计算了.

7.5.4.2. 均匀B-样条曲线的矩阵表示

均匀B-样条曲线的基函数$f_{k,i}(t)$可改写成幂函数的表示形式

$$f_{k,i}(t) = \sum_{l=0}^{k} a_{l,i} t^l, \quad t \in [0,1] \tag{7.5.34}$$

其中

$$a_{l,i} = \frac{C_n^k}{k!}\sum_{m=0}^{k-i}(-1)^m C_{k+m}^m (k-i-m)^{k-i} \quad l, i = 0, 1, ..., k \tag{7.5.35}$$

用矩阵表示上述关系, 则有

$$[f_{k,0}(t) \;\; f_{k,1}(t) \;\; ... \;\; f_{k,k}(t)] = [1 \;\; t \;\; t^2 \;\; ... \;\; t^k] M_k \tag{7.5.36}$$

其中的M_k是$(k+1)\times(k+1)$的常系数矩阵, 几个低阶的系数矩阵为

$$M_1 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad M_2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix} \tag{7.5.37}$$

$$M_3 = \frac{1}{3!}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \tag{7.5.38}$$

$$M_4 = \frac{1}{4!}\begin{bmatrix} 1 & 11 & 11 & 1 & 0 \\ -4 & -12 & 12 & 4 & 0 \\ 6 & -6 & -6 & 6 & 0 \\ -4 & 12 & -12 & 4 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix} \tag{7.5.39}$$

于是由$k+1$个控制点P_{j-k}, ..., P_j生成的一段B-样条曲线时只要用如下公式即可:

$$P_j(t) = [1 \;\; t \;\; t^2 \; ... \; t^k] M_k \begin{bmatrix} P_{j-k} \\ P_{j-k+1} \\ \vdots \\ P_j \end{bmatrix}, \quad t \in [0,1], \; j = k, ..., n \tag{7.5.40}$$

7.5.5 三次均匀B-样条曲线

7.5.5.1. 三次均匀B-样条曲线的性质

给定控制顶点$P_i, i = 0, 1, ..., n$，可以定义一条三次均匀B-样条曲线，其方程为

$$P_j(t) = \frac{1}{6}[1 \;\; t \;\; t^2 \; t^3] \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_{j-3} \\ P_{j-2} \\ P_{j-1} \\ P_j \end{bmatrix}, \tag{7.5.41}$$
$$t \in [0,1], \; j = 3, ..., n.$$

它的每一段是参数三次曲线，可以是平面的，也可以是空间的，取决于定义它的顺序四顶点是否共面。B-样条曲线在分段连接点处是C^2连续的。在不含内重顶点情况下，样条曲线的切线与曲率是连续的。移动一个顶点P_j，至多将影响到定义在四个相邻的参数节点区间$[u_{j-3}, uj-2]$, $[u_{j-2}, uj-1]$, $[u_{j-1}, uj]$和$[u_j, uj+1]$ 上的三次均匀B-样条曲线段的几何形状。从方程又可知，三次均匀B-样条曲线的分段连接点及其各阶导向量为

$$\begin{cases} P_{j-1}(1) = P_j(0) = \dfrac{1}{6}(P_{j-2} + 4P_{j-1} + P_j) \\ P'_{j-1}(1) = P'_j(0) = \dfrac{1}{2}(P_j - P_{j-2}) \\ P''_{j-1}(1) = P''_j(0) = P_j - 2P_{j-1} + P_{j-2} \end{cases} \tag{7.5.42}$$

记

$$M = \frac{P_{j-2} + P_j}{2} \tag{7.5.43}$$

则上式可以重写为

$$\begin{cases} P_{j-1}(1) = P_j(0) = P_{j-1} + \dfrac{1}{3}(M - P_{j-1}) \\ P'_{j-1}(1) = P'_j(0) = \dfrac{1}{2}(P_j - P_{j-2}) \\ P''_{j-1}(1) = P''_j(0) = 2(M - P_{j-1}) \end{cases} \tag{7.5.44}$$

如图7.7这个关系式与三角形$\Delta P_{j-2}P_{j-1}P_j$具有数十分密切的关系: 首先,考察三次均匀B-样条曲线的端点的位置. 由于M表示三角形的边$P_{j-2}P_j$上的中点,这

个端点就在这条边的中线上距点P_{j-1}的$\frac{1}{3}$处; 再看端点处的切向量,它平行于$\Delta P_{j-2}P_{j-1}P_j$ 的底边$P_{j-2}P_j$,且长度为这个边长的$\frac{1}{2}$; 而二阶导向量就是这条边的中线长度的2倍,指明了端点处三次均匀B-样条曲线的弯曲方向和弯曲程度.

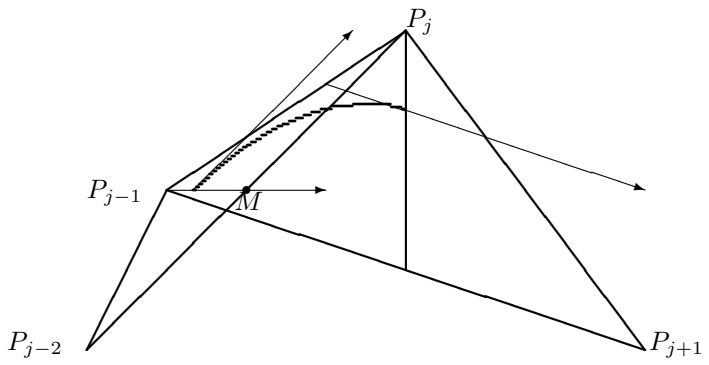


图7.7 三次均匀B-样条曲线与其控制多边形

7.5.5.2. 与其它三次参数多项式曲线的关系

三次均匀B-样条曲线也是一种参数的三次参数多项式曲线,与前面介绍的三次参数曲线, 三次Bézier曲线只是表示方式的不同,它们之间可以相互表示.

根据上面给出的三次均匀B-样条曲线段的端点及其切向量公式,可以写出其参数三次曲线的表示形式为:

$$P_j(t) = [1\ t\ t^2\ t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_j(0) \\ P_j(1) \\ P_j'(0) \\ P_j'(1) \end{bmatrix}, \quad t \in [0,1] \tag{7.5.45}$$

而

$$\begin{bmatrix} P_j(0) \\ P_j(1) \\ P_j'(0) \\ P_j'(1) \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \end{bmatrix} \begin{bmatrix} P_{j-2} \\ P_{j-1} \\ P_j \\ P_{j+1} \end{bmatrix} \tag{7.5.46}$$

反解这一矩阵关系式也可得

$$\begin{bmatrix} P_{j-2} \\ P_{j-1} \\ P_j \\ P_{j+1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \end{bmatrix}^{-1} \begin{bmatrix} P_j(0) \\ P_j(1) \\ P_j'(0) \\ P_j'(1) \end{bmatrix} \tag{7.5.47}$$

这也就是说,如果我们有了三次参数曲线,也可以把它表示为三次均匀B-样条曲线的形式.

根据Bézier曲线的端点的性质,如果我们定义

$$\begin{aligned} D_{3j-2} &= P_j(0), \quad D_{3j-1} = P_j(0) + \frac{1}{3}P'_j(0) \\ D_{3j} &= P_j(1), \qquad D_{3j+1} = P_j(1) + \frac{1}{3}P'_j(1) \end{aligned} \tag{7.5.48}$$

则三次均匀B-样条曲线段等价于Bézier曲线

$$\sum_{i=0}^{3} B_{i,3}(t) D_{3j-2+i}, \quad t \in [0,1] \tag{7.5.49}$$

其中$B_{i,3}(t)$为三次伯恩斯坦基函数,前式定义的Bézier点与控制顶点的直接关系可用矩阵表示为:

$$\begin{bmatrix} D_{3j-2} \\ D_{3j-1} \\ D_{3j} \\ D_{3j+1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} P_{j-2} \\ P_{j-1} \\ P_j \\ P_{j+1} \end{bmatrix} \tag{7.5.50}$$

反解这一矩阵关系式也可得

$$\begin{bmatrix} P_{j-2} \\ P_{j-1} \\ P_j \\ P_{j+1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix}^{-1} \begin{bmatrix} D_{3j-2} \\ D_{3j-1} \\ D_{3j} \\ D_{3j+1} \end{bmatrix} \tag{7.5.51}$$

这也就是说,如果我们有了三次Bézier曲线, 也可以把它表示为三次均匀B-样条曲线的形式.

7.5.6 准均匀B-样条曲线

均匀B-样条基函数在曲线定义域内各个节点区间上具有用局部参数$t \in [0,1]$表示的统一的表达式，使得计算与处理简单方便。但用它定义的均匀B-样条曲线有个缺点，就是没有保留Bézier曲线的端点几何性质。样条曲线的首末端点不再是控制多边形的首末顶点。高于二次的均匀B-样条曲线在端点处也不再与控制多边形相切。采用准均匀B-样条曲线类型可以解决这个问题，使得对曲线在端点的行为有较好的控制。

k次准均匀B-样条曲线的节点取值中两端节点具有重复度$k+1$，所有其它节点成均匀分布，由此决定的准均匀B-样条基函数所定义的准均匀B-样条曲线就具有同次Bézier曲线的端点几何性质。但不同于k次均匀B-样条曲线的是这时各个节点区间段上的k次准均匀B-样条曲线段的基函数将不相同,带来了计算处理方面的一些麻烦.

在曲线定义各个区间段上，两端各自有$k-1$个节点区间上定义的k次准均匀B-样条曲线依赖于端点处的节点值. 依起始的首端点为例,顺序定义$k-1$个节点区间上的k次准均匀B-样条曲线依赖于端点处的节点值的重数分别是$k+1$,k,...,2重，因此其基函数是各不相同的.

如果有除两端各自有$k-1$个节点区间以外的其它可以定义k次准均匀B-样条曲线的节点区间,则它们依赖的节点就都是均匀分布的了,因此有着与k次均匀B-样条曲线相同的基函数.

实际上二次准均匀B-样条曲线基函数有不同的四组,三次准均匀B-样条曲线基函数有不同的九组, 一般的k次准均匀B-样条曲线基函数有不同的k^2组,因此应

用起来并不方便. 实际计算时可直接采用递推公式(7.5.17),并按如下方式选取节点值.

设有$n+1$个控制点$P_j,\ j=0,1,...,n.$ 两端的$k+1$重节点分别是$u_0=u_1=...=u_k=0$和$u_{k+n}=u_{k+n+1}=...=u_{k+n+k}=n$,则曲线定义的各个节点区间段为$[u_j,u_{j+1}]=[j-k,j-k+1],\ j=k,k+1,...,k+n-1.$

7.5.7 一般的B-样条曲线

本小节的讨论中,我们要定义一个新的参数

$$\Delta_j=u_{j+1}-u_j \tag{7.5.52}$$

显然,$\Delta_j\geq 0$,为节点区间$[u_j,u_{j+1}]$的长度.通过下面的讨论我们可以看到这个参数值在一般的B-样条曲线计算中起着十分非常重要的作用.

7.5.7.1. 一般的二次B-样条曲线及其性质

根据递推公式(7.5.17),不难写出节点区间$[u_j,u_{j+1}]$上的二次B-样条曲线段的计算公式为

$$\begin{aligned}P_j(t)=&\frac{u_{j+1}-t}{u_{j+1}-u_j}(\frac{u_{j+1}-t}{u_{j+1}-u_{j-1}}P_{j-2}+\frac{t-u_{j-1}}{u_{j+1}-u_{j-1}}P_{j-1})\\&+\frac{t-u_j}{u_{j+1}-u_j}(\frac{u_{j+2}-t}{u_{j+2}-u_j}P_{j-1}+\frac{t-u_j}{u_{j+2}-u_j}P_j)\end{aligned} \tag{7.5.53}$$

也就是

$$\begin{aligned}P_j(t)=\ &\frac{(u_{j+1}-t)^2}{\Delta_j(\Delta_{j-1}+\Delta_j)}P_{j-2}\\&+(\frac{u_{j+1}-t}{\Delta_j}\frac{t-u_{j-1}}{\Delta_{j-1}+\Delta_j}+\frac{t-u_j}{\Delta_j}\frac{u_{j+2}-t}{\Delta_j+\Delta_{j+1}})P_{j-1}\\&+\frac{(t-u_j)^2}{\Delta_j(\Delta_j+\Delta_{j+1})}P_j\end{aligned} \tag{7.5.54}$$

由此可以求得$P_j(t)$的两个端点为

$$\begin{cases}P_j(u_j)=\dfrac{\Delta_j}{\Delta_{j-1}+\Delta_j}P_{j-2}+\dfrac{\Delta_{j-1}}{\Delta_{j-1}+\Delta_j}P_{j-1}\\P_j(u_{j+1})=\dfrac{\Delta_{j+1}}{\Delta_j+\Delta_{j+1}}P_{j-1}+\dfrac{\Delta_j}{\Delta_j+\Delta_{j+1}}P_j\end{cases} \tag{7.5.55}$$

由此式可知,前后相连的二次B-样条曲线段是整体位置连续的. 二次B-样条曲线段的端点就是相应的控制多边形的边上的比例分位点, 具体的比例值就是相对应的相邻的两个节点区间的长度的比值$\Delta_{j-1}:\Delta_j$ 和$\Delta_j:\Delta_{j+1}$.

两个端点处的切向量为

$$\begin{cases}P_j'(u_j)=\dfrac{2}{\Delta_{j-1}+\Delta_j}(P_{j-1}-P_{j-2})\\P_j'(u_{j+1})=\dfrac{2}{\Delta_j+\Delta_{j+1}}(P_j-P_{j-1})\end{cases} \tag{7.5.56}$$

由此式可知,二次B-样条曲线段在两端与相应的控制多边形相切,前后相连的二次B-样条曲线段在端点处的切线是连续的.

现在的参数取值区间为$t \in [u_j, u_{j+1}]$,作替换

$$u = \frac{t - u_j}{\Delta_j} \tag{7.5.57}$$

则可定义曲线关于参数u的表达式

$$P_{j,u}(u) = P_j(\Delta_j u + u_j), \quad u \in [0, 1] \tag{7.5.58}$$

对于新参数u,曲线段两个端点处的切向量为

$$\begin{cases} P'_{j,u}(0) = \dfrac{2\Delta_j}{\Delta_{j-1} + \Delta_j}(P_{j-1} - P_{j-2}) \\ P'_{j,u}(1) = \dfrac{2\Delta_j}{\Delta_j + \Delta_{j+1}}(P_j - P_{j-1}) \end{cases} \tag{7.5.59}$$

如果我们定义

$$B_{2j-2} = P_{j,u}(0) = P_j(u_j),\ B_{2j-1} = P_{j-1}B_{2j} = P_{j,u}(1) = P_j(u_{j+1}) \tag{7.5.60}$$

则可以验算出

$$P'_{j,u}(0) = 2(B_{2j-1} - B_{2j-2}),\ P'_{j,u}(1) = 2(B_{2j} - B_{2j-1}) \tag{7.5.61}$$

这说明二次B-样条曲线段是由三点B_{2j-2}, B_{2j-1}和B_{2j}生成的二次Bézier曲线.

7.5.7.2. 一般的B-样条曲线的导向量

一般的B-样条曲线上一点$t \in [u_j, u_{j+1}]$处的r 阶导向量$P^{(r)}(t)$可按如下步骤计算:

第一步: 递推定义

$$D_i^{(m)} = \begin{cases} P_i & m = 0 \\ (k + m - 1)\dfrac{D_i^{(m-1)} - D_{i-1}^{(m-1)}}{u_{j+k+1-m} - u_j} & \begin{array}{l} m = 1, 2, ..., r; \\ i = j - k + m, ..., j \end{array} \end{cases} \tag{7.5.62}$$

第二步: 对第一步由控制顶点$P_i, i - j - k, j - k + 1, ..., j$经过$r$级递推计算出第$r$级中间顶点$D_i^{(r)}, i = j - k + r, ..., i$当作控制顶点,产生的$k - r$次B-样条曲线上的点即为节点区间$[u_j, u_{j+1}]$上参数值$t$处的$r$ 阶导向量$P^{(r)}(t)$.其中产生$k - r$次B-样条曲线时所用的节点值仍是原来的节点值.

与求解B-样条曲线上点的递推公式不同的是，求导向量的顶点递推公式与所给参数$t \in [u_j, u_{j+1}]$的具体值无关。但导向量与具体参数值t有关，这反映在第二步产生$k - r$次B-样条曲线的求解过程中。

7.5.8 插值三次B-样条曲线

在前面的讨论中，我们知道B-样条曲线一般不通过控制多边形的控制顶点，利用相重顶点或节点方法仅能使曲线通过个别的控制点, 同时导致曲线在这些点处的的光滑性降低了,甚至可能出现奇异点。但在实际工作中，往往是给出一组离散的数据点, 要求生成的曲线通过这些点，并要求整条曲线具有一致的光滑性. 求解满足这些要求的曲线，就是所谓的插值问题。利用B-样条曲线解决插值问题，必须从给定的型值点求出相应控制多边形的顶点,然后才能求得具有插值性质的B-样条曲线。这一过程常称为反算拟合。相对而言，把前几节介绍的给定控制顶点定义B-样条曲线及其计算，都称为正算。

反算问题可以叙述为：设给定$n+1$个曲线上的数据点$D_i, i=0,1,...,n$，要求解出一条通过这$n+1$个数据点的三次B-样条曲线。为此，可设所要求的三次B-样条曲线的控制点为$P_i, i=0,1,...,n$.

这条插值曲线$P(t)$的可认为是每两个相邻的数据点D_j和D_{j+1}之间有一段三次B-样条曲线$P_j(t)$, 定义在节点区间$[u_{j+3}, u_{j+4}]$上.因此有

$$P(u_{j+3}) = \sum_{i=j}^{j+3} N_{i,j}(t)P_i = D_j, j=0,1,2,...,n \tag{7.5.63}$$

进一步的讨论需要区分开曲线和闭曲线两种情况.

第一种情况: 要求插值曲线是闭曲线,并且处处二阶连续.

这时一般认为数据点连出的折线段也是闭的折线段, 因此有$D_n = D_0$. 不计$j=0$和$j=n$的重复, 方程组(7.5.63)就只有n个方程.进一步扩展定义数据点$D_{n+1}=D_1$和$D_{n+2}=D_2$,则根据三次B-样条曲线的连续性,整条三次B- 样条曲线就是处处二阶连续.于是n个控制点就可以从方程组(7.5.63)的n个方程中解出. 这个方程组等价于如下更直接的矩阵方程.

$$\begin{bmatrix} b_1 & c_1 & & & & a_1 \\ a_2 & b_2 & c_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ a_n & & & & b_n & c_n \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ \vdots \\ P_{n-1} \\ P_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_{n-1} \\ B_n \end{bmatrix} \tag{7.5.64}$$

其中, 对$i=1,2,...,n$有

$$\begin{cases} a_i = \dfrac{(\Delta_{i+2})^2}{\Delta_i + \Delta_{i+1} + \Delta_{i+2}} \\ b_i = \dfrac{\Delta_{i+2}(\Delta_i + \Delta_{i+1})}{\Delta_i + \Delta_{i+1} + \Delta_{i+2}} + \dfrac{\Delta_{i+1}\Delta_{i+2} + \Delta_{i+3}}{\Delta_{i+1} + \Delta_{i+2} + \Delta_{i+3}} \\ c_i = \dfrac{(\Delta_{i+1})^2}{\Delta_{i+1} + \Delta_{i+2} + \Delta_{i+3}} \\ B_i = (\Delta_{i+1} + \Delta_{i+2})D_{i-1} \end{cases} \tag{7.5.65}$$

第二种情况: 插值曲线是开曲线(包括首尾处仅相连,但不要求二阶光滑的闭曲线)

对于这样的三次B-样条曲线方程组(7.5.63)中$n+1$个方程不足以决定其中n段三次B-样条曲线段所需要的$n+3$个未知控制顶点，还必须增加两个通常由边界条件给定的附加方程。边界条件可以根据实际问题的需要选取。首尾端点可取相同的或不同的边界条件, 由此建立相应的附加方程。

如果没有特别的要求,我们可取两端节点重度度$r=3$，于是相应的三次B-样条曲线的首控制顶点就是首未数据点，即$P_0 = D_0$，$P_{n+2} = D_n$，且曲线在首未端点处分别有切向量

$$\begin{cases} P'(u_3) = \dfrac{3}{\Delta_3}(P_1 - P_0) \\ P'(u_{n+3}) = \dfrac{3}{\Delta_{n+2}}(P_{n+2} - P_{n+1}) \end{cases} \tag{7.5.66}$$

如果我们指定首未数据点处的切向量(即曲线端点处的切线方向)T_0,T_n, 就有附加方程

$$\begin{cases} P_1 - P_0 = \dfrac{\Delta_3}{3}P'(u_3) = \dfrac{\Delta_3}{3}T_0 \\ P_{n+2} - P_{n+1} = \dfrac{\Delta_{n+2}}{3}P'(u_{n+3}) = \dfrac{\Delta_{n+2}}{3}T_n \end{cases} \tag{7.5.67}$$

联合方程组(7.5.63),可得如下线性方程组

$$\begin{bmatrix} b_2 & c_2 & & & & \\ a_3 & b_3 & c_3 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} P_2 \\ P_3 \\ \vdots \\ \vdots \\ P_{n-1} \\ P_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_{n-1} \\ B_n \end{bmatrix} \tag{7.5.68}$$

其中B_3,...,B_{n-1}定义同前,而

$$\begin{cases} P_0 = D_0,\ P_1 = P_0 + \dfrac{\Delta_3}{3}T_0 \\ B_1 = D_0 + \dfrac{\Delta_3}{3}T_0 \\ B_2 = (\Delta_3 + \Delta_4)D_1 - \dfrac{(\Delta_4)^2}{\Delta_3 + \Delta_4}P_1 \\ \\ P_{n+2} = D_n,\ P_{n+1} = P_{n+2} - \dfrac{\Delta_{n+2}}{3}T_n \\ B_n = (\Delta_{n+1} + \Delta_{n+2})D_{n-1} - \dfrac{(\Delta_{n+1})^2}{\Delta_{n+1} + \Delta_{n+2}}P_{n+1} \\ B_{n+1} = D_n - \dfrac{\Delta_{n+2}}{3}T_n \end{cases} \tag{7.5.69}$$

根据具体情况,分别求解上述方程组即可求出全部控制点，满足要求(7.5.63)的三次B-样条插值曲线就完全确定了。

7.6 非均匀有理B-样条曲线

B-样条曲线在曲线的设计中表现出非常优良的性质,唯一的不足是不能精确地表示一段或完整的椭圆弧或双曲线线段,这其中当然也包括不能表示一段或完整的圆弧曲线. 这是包括三次参数曲线及Bézier曲线在内的所有多项式参数曲线具有的缺陷. 而一段或完整的圆弧曲线是非常常见的曲线,特别是在工业产品的外形设计中, 有许多部件的外形就是一段或完整的圆弧曲线. 用多项式曲线近似表示时会带来不必要的设计误差. 不用B-样条曲线表示时就要引入新的,可以精确表示圆弧曲线方法. 新的方法本身可能并不复杂,但对利用计算机处理图形将会带来许多麻烦. 较为可取的方法是寻找能包含现有结果为特例,同时又能克服上述缺陷的曲线表示方法. 满足这一要求的方法就是非均匀有理B-样条方法. 根据其英文名称Non-Uniform Rational B-spline 而简单称为NURBS方法. 相应产生的曲线也可称为非均匀有理B-样条曲线或NURBS曲线,

7.6.1 非均匀有理B-样条曲线

一条k阶非均匀有理B-样条曲线可定义为

$$P(t)=\frac{\sum_{i=0}^{n}N_{ik}(t)\omega_iP_i}{\sum_{i=0}^{n}N_{ik}(t)\omega_i} \tag{7.6.1}$$

可以看出$P(t)$为有理多项式表示的向量函数.这也是"有理"一词的由来. 其中$\omega_i,i=0,1,\cdots,n$称为权数或权因子,各自与相应的控制点$P_i,i=0,1,\cdots,n$ 对应.$N_{ik}(t)$是相应的B-样条基函数, 确定基函数当然需要有一列节点取值$u_i,i=0,1,\cdots,n+k+1$. 为保证曲线继续具有保凸性质, 及分母不为0,一般情况下总要求成立

$$\omega_0>0,\omega_n>0;\ \omega_i\geq 0,i=1,...,n-1 \tag{7.6.2}$$

由公式(7.6.1)可以看出B-样条的递推定义依然可用:对乘上加权因子后的加权点列$\omega_iP_i,i=0,1,\cdots,n$ 利用B-样条的递推定义算出式(7.6.1)的分子, 再把$\omega_i,i=0,1,\cdots,n$ 看作一维点列利用B-样条的递推定义算出式(7.6.1)的分母.

权因子对曲线形状的影响可通过下面几个简单事实反映出来.

(1). 如果某个权因子$\omega_i=0$, 则非均匀有理B-样条曲线与该权因子相应的控制顶点P_i无关;

(2). 如果某个权因子$\omega_i\to\infty$, 则只有当$t\in[u_i,u_{i+k+1}]$时, $P(t)$才与P_i有关, 且这时有$P(t)\to P_i$.

(3). 当所有的权因子都相等时, 非均匀有理B-样条曲线就成了一般的非均匀B-样条曲线.

通过以上事实可以看出, 单独增加或减小某个权因子ω_i,具有使非均匀有理B-样条曲线靠近或远离与该权因子相应的控制顶点P_i的作用.

式子(7.6.1)有如下等价的基函数表示:

$$\begin{cases}P(t)=\sum_{i=0}^{n}R_{ik}(t)\omega_iP_i\\ R_{ik}(t)=\dfrac{\omega_iN_{ik}(t)}{\sum_{j=0}^{n}N_{jk}(t)\omega_i}\end{cases} \tag{7.6.3}$$

该式表明非均匀有理B-样条曲线也可通过基函数的形式表示出来,不同于前面曲线的是只不过此时的基函数$R_{ik}(t),i=0,1,\cdots,n$为有理函数而已.

(4). 由于$N_{i,k}(t)\geq 0,\sum N_{i,k}(t)=1$,因此有

$$R_{i,k}(t)\geq 0,\quad \sum R_{i,k}(t)=1 \tag{7.6.4}$$

结合公式(7.6.3)可知, 非均匀有理B-样条曲线仍然具有凸包性质.

前面提到公式(7.6.1)的分子可以看作对乘上加权因子后的加权点列$\omega_iP_i,i=0,1,\cdots,n$ 利用B-样条的递推定义算出. 如果添加一个分量ω_i, 则得到

$$D_i=(\omega_iP_i,\omega_i).$$

这是点P_i的齐次坐标表示. 于是我们可以看到, 非均匀有理B-样条曲线可以认为是按如下方法产生的: 对齐次坐标表示的点列$D_i, i = 0, 1, \cdots, n$按通常的方法生成B-样条曲线,得到的就是曲线上点的齐次坐标,变换回非齐次的坐标表示,就得到了非均匀有理B-样条曲线的点. 如此解释的好处是有关通常的B-样条曲线理论的一切结论都可在非均匀有理B-样条曲线理论和实践中参照应用.

7.6.2 有理Bézier曲线

有理Bézier曲线可以作为特定类型的非均匀有理B-样条曲线而定义,也可直接可定义为

$$P(t) = \frac{\sum_{i=0}^{n} B_{in}(t)\omega_i P_i}{\sum_{i=0}^{n} B_{in}(t)\omega_i}, \quad t \in [0,1] \tag{7.6.5}$$

其中$\omega_i, i = 0, 1, ..., n$为权数或权因子,各自与相应的控制点$P_i, i = 0, 1, ..., n$ 对应.$B_{ik}(t)$是相应的伯恩斯坦基函数,一般情况下总要求成立

$$\omega_0 > 0, \omega_n > 0;\ \omega_i \geq 0, i = 1, ..., n-1 \tag{7.6.6}$$

同样地,Bézier曲线的递推定义也仍然可以参照使用.

有理Bézier曲线的端点仍然是首末控制点:

$$P(0) = P_0,\ P(1) = P_n; \tag{7.6.7}$$

两端点处的切向量为:

$$P'(0) = n\frac{\omega_1}{\omega_0}(P_1 - P_0),\quad P'(1) = n\frac{\omega_{n-1}}{\omega_n}(P_n - P_{n-1}). \tag{7.6.8}$$

与原来的Bézier 在端点处的切向量方向相同, 只是向量长度有所变化.

7.6.3 二次有理Bézier曲线与二次曲线

二次有理Bézier曲线的表达式为

$$P(t) = \frac{(1-t)^2\omega_0 P_0 + 2t(1-t)\omega_1 P_1 + t^2\omega_2 P_2}{(1-t)^2\omega_0 + 2t(1-t)\omega_1 + t^2\omega_2}, \quad t \in [0,1] \tag{7.6.9}$$

如果作参数替换

$$t = t(u) = \frac{au + b}{cu + d} \tag{7.6.10}$$

并要求

$$t(0) = 0,\quad t(1) = 1 \tag{7.6.11}$$

则有

$$b = 0,\quad a = c + d \tag{7.6.12}$$

于是有

$$t = t(u) = \frac{au}{cu + d},\quad 1 - t = \frac{d(1-u)}{cu + d} \tag{7.6.13}$$

代入(7.6.9),有

$$\begin{aligned} P(t) &= P(t(u)) \\ &= \frac{(1-u)^2d^2\omega_0P_0 + 2t(1-t)ad\omega_1P_1 + u^2a^2\omega_2P_2}{(1-u)^2d^2\omega_0 + 2u(1-u)ad\omega_1 + u^2a^2\omega_2}, u \in [0,1] \end{aligned} \tag{7.6.14}$$

如果我们选取$a = \frac{1}{\sqrt{\omega_2}}$,$d = \frac{1}{\sqrt{\omega_0}}$,则有

$$\begin{aligned} P(t) &= P(t(u)) \\ &= \frac{(1-u)^2P_0 + 2t(1-t)\sqrt{\frac{\omega_1^2}{\omega_0\omega_2}}P_1 + u^2P_2}{(1-u)^2 + 2u(1-u)\sqrt{\frac{\omega_1^2}{\omega_0\omega_2}} + u^2}, u \in [0,1] \end{aligned} \tag{7.6.15}$$

把此式的右端与式(7.6.9)作比较可以看出, 这是一个关于参数$u \in [0,1]$的二次有理Bézier曲线的表达式, 其权因子为

$$\omega_0^* = 1, \quad \omega_1^* = \sqrt{\frac{\omega_1^2}{\omega_0\omega_2}}, \quad \omega_2^* = 1 \tag{7.6.16}$$

这说明任何一个一般的二次有理Bézier曲线都等价于一个权因子满足(7.6.16) 的二次有理Bézier曲线. 我们称$\omega_0 = \omega_2 = 1$的二次有理Bézier曲线表达式为二次有理Bézier曲线的标准型或标准表示. 不失一般性, 下面只讨论具有标准表示的二次有理Bézier曲线.

标准二次有理Bézier曲线有如下性质:

(1). 二次有理Bézier曲线表示的二次曲线类型依ω_1^*的取值依次为:

$\omega_1^* = 0$时为首末控制点的线段P_0P_2;

$0 < \omega_1^* < 1$时为一段椭圆弧(包括圆弧);

$\omega_1^* = 1$时为一段抛物线弧(具有一般的多项式表示的二次Bézier曲线);

$1 < \omega_1^* < \infty$时为一段双曲线弧;

$\omega_1^* = \infty$时为控制多边形的两条边乘积形式的退化二次曲线. 曲线的几何形状即为控制多边形本身.

(2). 二次有理Bézier曲线为圆弧曲线的充要条件是

$$\begin{cases} |\overrightarrow{P_0P_1}| = |\overrightarrow{P_1P_2}|, \\ \omega_1^* = \dfrac{\overrightarrow{P_0P_1} \cdot \overrightarrow{P_0P_2}}{|\overrightarrow{P_0P_1}|\,|\overrightarrow{P_0P_2}|}. \end{cases} \tag{7.6.17}$$

这也就是说三个控制点构成的三角形是以直线段P_0P_2为底边的等腰三角形,权因子ω_1^*等于三角形底角的余弦值.

根据有理Bézier曲线的凸包性质, 圆弧曲线应被包含在等腰三角形内. 这说明二次有理Bézier曲线只能表示出小于半圆的圆弧曲线,要想表示出一个完整的圆至少需要三段二次有理Bézier曲线.

习题7

1. 设计三次参数曲线$(-1,\ 0)$, 终点为$(1,\ 0)$, 两点处的切线与经过这两点并且圆心在坐标的圆相同.调整切向量的长度, 使得设计的三次参数曲线尽可能的接近半圆弧.两者是否可以重合?

2. 写出满足如下要求的三次参数曲线表达式：起点位置(-1,0)，切向量(0,1); 终点位置(1,0)，切向量(0,-1)。

3. 互换前题中起点、终点的位置、切向量，求这个三次参数曲线的表达式，并作图比较这两条曲线。

4. 设计二次Bezier曲线,要求其起点为$(0,\ 1)$, 终点为$(1,\ 0)$, 两点处的切线与经过这两点的二次曲线$x^2-2xy+y^2-2x-2y+1=0$的切线相同，切向量长度为1。求的是否为二次曲线？是否为二次曲线$x^2-2xy+y^2-2x-2y+1=0$上的一段？如不是，是否可通过调整切向量长度使其是指定二次曲线的一段?

5. 找出四个点P0,P1,P2,P3,使得有它们产生的三次Bezier曲线就是习题2的三次参数曲线；找出四个点P4,P5,P6,P7,使得有它们产生的三次Bezier曲线就是习题3的三次参数曲线。

6. 用习题5中的八个点顺次产生三次均匀B-样条曲线，有几段三次均匀B-样条曲线？用计算机画出这些曲线，并与前面的三次参数曲线、三次Bezier曲线进行相互比较。

7. 找出四个点P0,P1,P2,P3,使得有它们产生的三次B-样条曲线就是习题2的三次参数曲线；找出四个点P4,P5,P6,P7,使得有它们产生的三次B-样条曲线就是习题3的三次参数曲线。

8. 求经过$(1,\ 0)$, $(0,\ 1)$, $(-1,\ 0)$, $(0,\ -1)$, $(1,\ 0)$ 的封闭三次均匀B-样条曲线。

9. 习题6中的三次均匀B-样条曲线是否是封闭曲线? 如不是,如何生成一封闭的曲线?

10. 证明三次参数曲线无法表示出一段圆弧或完整的圆.

11. 证明标准二次有理Bézier曲线表示的二次曲线类型依ω_1^*的取值的分类结果.

Chapter 8

计算机图形中曲面的设计理论

8.1 引言

对曲面的处理是计算机图形学(在处理三维图形时)的基本任务之一. 物体的形状可以因为实际工程的需要或者是艺术上审美的需要而成为很复杂很特别的几何形状。要真实的表现出这个几何体,最基本的要求就是能描述它们的表面,这一般情况下就是一个复杂的曲面.

本章我们就讨论各种曲面表示方法和技术。在计算机出现以前的工业生产中就有进行曲面设计的需要，那时对曲面的设计描述，一般是用二、三族平面去切割要设计的曲面，以所得到的很少几族截交线来表示这张曲面, 实际上是把曲面的设计转化成曲线的设计,然后利用设计好的曲线通过一些简单的方法设计出曲面。在计算机出现后，有可能用更好的,当然也比较复杂一些的方法来设计与描述曲面，也因此可以研究对曲面设计的理论和方法。与曲线设计的理论和方法相对比较简单成熟而言,曲面设计的理论和方法却非常复杂,也不很成熟. 本章我们讨论各种曲面的表示的理论和方法。

8.2 双线性孔斯曲面

曲面被截线分割后,实际上就成了以截线为边界线的一个个曲面片.如果每个曲面片被设计出来了,整个曲面也就被拼接出来了.(当然,由曲面片拼接成的完整的曲面要想有好的性质,用于拼接的曲面片之间应满足一定的连续性条件.) 这一思路的最大好处是，在设计某一物品的几何外形时，设计人员可以从一个或较少的几个曲面片开始作初步设计，利用交互的方法，考察所得到的曲面是否符合设计要求，如果不符，可以修改或增加新的边界曲线，分割成更小的曲面片重新合成一张新的曲面，直到生成的曲面达到设计要求为止。

最简单的曲面片(Patch)可以看作是由四条边界曲线定义的，当其中的某一条边界线退化成一点时就成了三角形的曲面片。每条边界曲线可以是具有一定连续性要求的任何设计好的曲线。因此,我们的问题开始是这样提出的：给定由四条参数曲线围成封闭的空间曲边四边形，要求找到一张以这四条曲线为边界曲线的曲面. 一个简单的方法是在某一对曲线边界的起始点上放一根长短可以伸缩的直线段,当直线段的两端沿着一对曲线边界运动时就生成了一个以这一对曲线为边界线的曲面片.

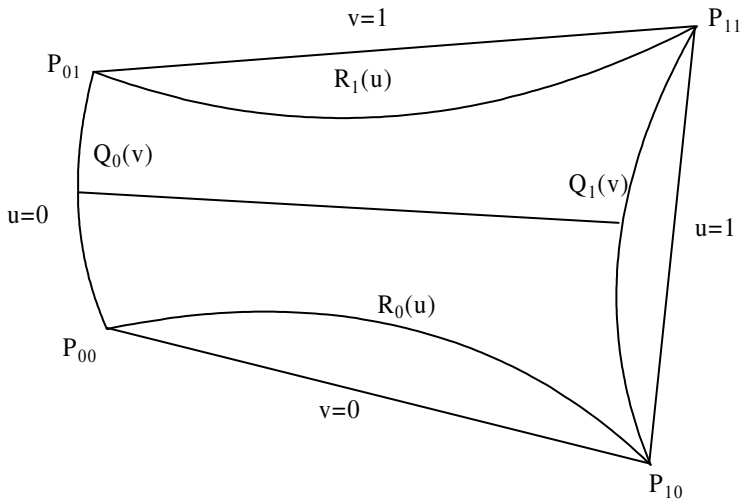


图8.1 曲线边界的四边形及其角点示意图

假设这一对曲线边定义在参数区间$u \in [0,1]$上, 即有参数方程$Q_0(u)$和$Q_1(u)$, 而$u \in [0,1]$, 且满足

$$\begin{cases} Q_0(0) = P_{00},\ Q_0(1) = P_{10}; \\ Q_0(1) = P_{01},\ Q_1(1) = P_{11} \end{cases} \tag{8.2.1}$$

则可写出如上运动生成的曲面片方程为

$$Q(u,v) = (1-v)Q_0(u) + vQ_1(u), \quad (u,v) \in [0,1;0,1] \tag{8.2.2}$$

完全对等地, 对另一对定义在$v \in [0,1]$上的曲线边, 有参数方程$R_0(v)$和$R_1(v)$, 满足

$$\begin{cases} R_0(0) = P_{00},\ R_0(1) = P_{01}; \\ R_1(0) = P_{10},\ R_1(1) = P_{11}. \end{cases} \tag{8.2.3}$$

我们也可写出如上方式生成的曲面片方程:

$$R(u,v) = (1-u)R_0(v) + uR_1(v), (u,v) \in [0,1;0,1] \tag{8.2.4}$$

但这两个曲面片都不符合以四条曲线为边界曲线的条件. 如果把两者迭加起来却适得其反，$Q(u,v)+R(u,v)$不再插值任一对边界。对曲面片$Q(u,v)+R(u,v)$而言, 在参数$u=0$时的边为

$$Q(0,v) + R(0,v) = (1-v)P_{00} + vP_{01} + R_0(v), \quad v \in [0,1] \tag{8.2.5}$$

与指定的曲线边$R_0(v)$相比, 多出的一部分是$(1-v)P_{00} + vP_{01}$. 这一部分的几何意义非常明显: 为连接曲线边界两端点的直线段. 如果$R_0(v)$本身就是直线段边,则这两部分是相同的. 其它三条边的情况也完全类似. 不难看出,在四条曲线边恰好都为相应的直线段时$Q(u,v)$和$R(u,v)$定义出同一个曲面片

$$\begin{aligned} S(u,v) = &\quad (1-u)(1-v)P_{00} + (1-u)vP_{01} \\ + &\quad u(1-v)P_{10} + uvP_{11}, \quad (u,v) \in [0,1;0,1] \end{aligned} \tag{8.2.6}$$

亦即

$$S(u,v)=[1-u\ u]\begin{bmatrix} P_{00} & P_{01} \\ P_{1,0} & P_{11} \end{bmatrix}[1-v\ v],\quad (u,v)\in[0,1;0,1] \tag{8.2.7}$$

为由曲面片四角点决定的一张双线性张量积曲面,插值于四条直线段边,正好是$Q(u,v)+R(u,v)$要插值于四条曲线边时多余的那一部分. 为得到要求的插值曲面，曲面片$Q(u,v)+R(u,v)$减去$S(u,v)$即可,于是有插值于四条曲线边的曲面

$$P(u,v)=Q(u,v)+R(u,v)-S(u,v),\quad (u,v)\in[0,1;0,1] \tag{8.2.8}$$

如此定义的曲面片就是双线性混合孔斯曲面片.根据$P(u,v)$的插值性质,四条边界曲线的等式关系为

$$\begin{cases} Q_0(u)=P(u,0),\ Q_1(u)=P(u,1), \\ R_0(v)=P(0,v),\ R_1(u)=P(1,v) \end{cases} \tag{8.2.9}$$

四个角点为

$$\begin{cases} P_{00}=P(0,0),\ P_{01}=P(0,1), \\ P_{10}=P(1,0),\ P_{11}=P(1,1) \end{cases} \tag{8.2.10}$$

于是,$P(u,v)$可用矩阵形式表示为

$$P(u,v)=-[-1\ 1-u\ u]\begin{bmatrix} 0 & P(u,0) & P(u,1) \\ P(0,v) & P(0,0) & P(0,1) \\ P(1,v) & P(1,0) & P(1,1) \end{bmatrix}\begin{bmatrix} -1 \\ 1-v \\ v \end{bmatrix} \tag{8.2.11}$$

其中右端三阶方阵包含了曲面的全部边界信息，因此可称为边界信息矩阵。这一矩阵的各行元素是曲面片函数$P(u,v)$由参数v分别取可变值v及端点值0和1,同时各列相应地由参数u分别取可变值u及端点值0和1而得到,但其中的第一行第一列处由于是$P(u,v)$本身为未知,用0替换了. 如果把上式看作是由下式解得的,则上式右端的负号就不奇怪了.

$$[-1\ 1-u\ u]\begin{bmatrix} P(u,v) & P(u,0) & P(u,1) \\ P(0,v) & P(0,0) & P(0,1) \\ P(1,v) & P(1,0) & P(1,1) \end{bmatrix}\begin{bmatrix} -1 \\ 1-v \\ v \end{bmatrix}=0 \tag{8.2.12}$$

当然,此式中只有$P(u,v)$是未知的,其它各项都应是已知的. 在式子的三阶方阵中,右下角二阶子块中四个点向量即曲面片的四个角点，第一行的后两列分别为参数$v=0$和$v=1$时的边界线,第一列的后两行分别为参数$u=0$和$u=1$时的边界线。

双线性混合孔斯曲面片很简单地实现了插值四条边界曲线的要求。用它来进行曲面拼合，由边界曲线的公用性自然地保证了整张曲面的位置连续性。但要实现相邻曲面片沿公共边界的光滑连接就不是很简单的事情了. 由(8.2.11)考察公共边界线上的的跨界切向量。

将(8.2.11)式对v求偏导后代入$v=0$可得公共边界线$P(u,0)$上的的跨界切向量为

$$P_v(u,v)=[1\ 1-u\ u]\begin{bmatrix} P(u,1)-P(u,0) \\ P_v(0,0)+P(0,0)-P(0,1) \\ P_v(1,v)+P(1,0)-P(1,1) \end{bmatrix} \tag{8.2.13}$$

式中的下标v表示对参数v求偏导. 由此式可知跨界切向量不仅与该边界端点切向量有关，还与该边界曲线有关。因此，即使两相邻曲面片在公共边界线两端点处光滑，而沿公共边界所有其它点处，一般地因跨界切向量不连续，仅是位置连续的。这就达不到到曲面片之间的光滑连接，不能用来拼合构造具有良好性质的复杂曲面。

8.3 双三次孔斯曲面

8.3.1 双三次孔斯曲面的定义

前节曲面片的生成过程中,我们在一对相对曲线边上各取一点,然后简单地把两点用直线段连接起来.连续的重复这个过程,就得到了一个曲面片.那里用到的只是一些位置信息,最后得到的曲面片自然也就难以保证是光滑的了.如果考虑两点处的切线信息,用连接两点的三次参数曲线代替直线段,我们应该可以预期得到光滑的曲面.下面就根据这一思路,结合前面产生孔斯曲面的方法产生能够光滑拼接在一起的曲面片.

三次参数插值曲线不仅要求端点位置信息，而且还要求端点切向量信息。注意这里位置信息是整条边界曲线边上的各个点，而不是某个单独的点，相应要求的切向量信息也应沿整条曲线边上的各个点，而不是某个单独的点处的。因此，我们应该有的已知数据至少包括：

1. 四个角点:

   P_{00}, P_{10}, P_{01}, P_{11};

2. 四条边界曲线:

   $Q_0(u)$, $Q_1(u)$, $R_0(v)$, $R_1(v)$;

3. 四条边界边上的跨界切向量:

   $Q_0^v(u)$, $Q_1^v(u)$, $R_0^u(v)$, $R_1^u(v)$.

于是,对固定的$v \in [0,1]$根据两点$P(0,v)$, $P(1,v)$及两点处的切向量$Q_0^v(u)$, $Q_1^v(u)$产生的三次参数曲线为

$$\begin{aligned} Q(u,v) = &\quad F_0(u)P(0,v) + F_1(u)P(1,v) \\ + &\quad F_2(u)Q_0^v(u) + F_3(u)Q_1^v(u),\ u \in [0,1] \end{aligned} \tag{8.3.1}$$

其中四个函数$F_i(u), i = 0,1,2,3$为三次参数曲线定义中的三次埃尔米特函数. 把$Q(u,v)$看作是矩形参数区域$(u,v) \in [0,1;0,1]$上的函数, 就是一个曲面片.

类似地，对固定的$u \in [0,1]$根据两点$R_0(v)$, $R_1(v)$及两点处的切向量$R_0^u(v)$, $R_1^u(v)$产生的三次参数曲线为

$$\begin{aligned} R(u,v) = &\quad F_0(v)R_0(v) + F_1(v)R_1(v) \\ &+F_2(v)R_0^u(v) + F_3(v)R_1^u(v),\ v \in [0,1] \end{aligned} \tag{8.3.2}$$

$R(u,v)$也是矩形参数区域$(u,v) \in [0,1;0,1]$上的一个曲面片.

如同在前节一样,我们将要定义仅仅利用角点数据信息的插值曲面. 这也就是说，假定四条边界线现在是未知的。

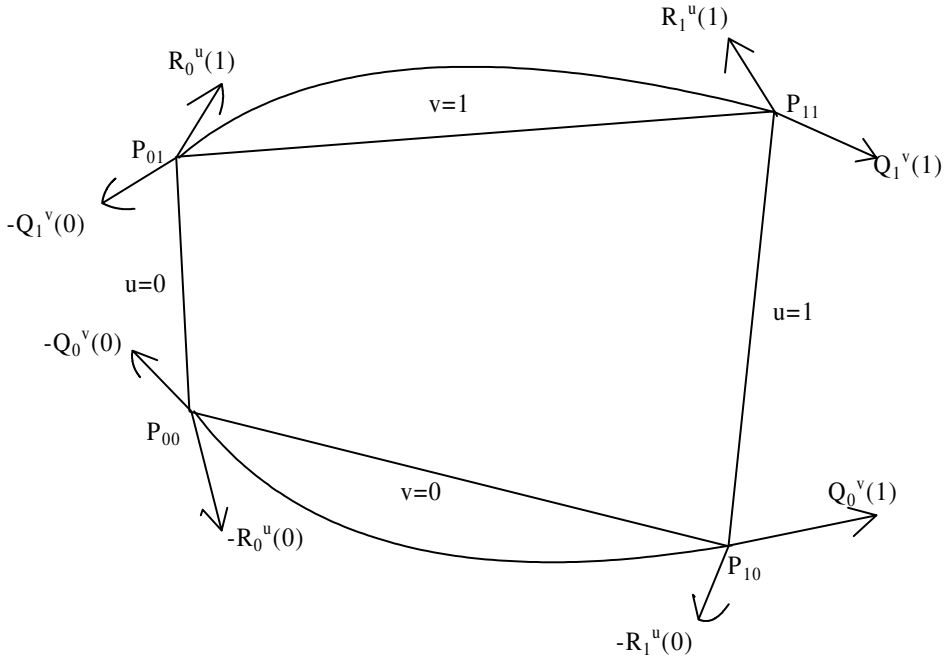


图8.2 定义插值曲面需要的四个角点数据信息

首先, 应给出由两角点P_{00}和P_{10}及两角点处的切向量$Q_0^v(0)$和$Q_0^v(1)$产生的$v=0$对应的曲线边界线:

$$F_0(u)P_{00}+F_1(u)P_{10}+F_2(u)Q_0^v(0)+F_3(u)Q_0^v(1),\quad u\in[0,1] \tag{8.3.3}$$

由两角点P_{01}和P_{11}及两角点处的切向量$Q_1^v(0)$和$Q_1^v(1)$产生的$v=1$对应的曲线边界线为:

$$F_0(u)P_{01}+F_1(u)P_{11}+F_2(u)Q_1^v(0)+F_3(u)Q_1^v(1),\quad u\in[0,1] \tag{8.3.4}$$

应注意的是利用这一对生成的曲线边上的点求三次参数曲线时还需要各点处的切向量, 即关于参数v的一阶导向量信息. 端点处的一阶导向量已经给出, 但要利用三次参数曲线的方法产生曲线边上各点处的切向量就需要端点处一阶导向量的关于参数v的一阶导向量,即二阶混合导向量(通常也称为扭矢)，为此设四个端点处的二阶混合导向量为$P_{uv(0,0)}$, $P_{uv(0,1)}$,$P_{uv(1,0)}$和$P_{uv(1,1)}$,于是我们可以给出$v=0$ 对应的曲线边界线各点关于参数v的切向量为:

$$F_0(u)Q_0^v(0)+F_1(u)Q_0^v(1)+F_2(u)P_{uv(0,0)}+F_3(u)P_{uv(1,0)},\quad u\in[0,1] \tag{8.3.5}$$

和$v=1$对应的曲线边界线各点关于参数v的切向量为:

$$F_0(u)Q_1^v(0)+F_1(u)Q_1^v(1)+F_2(u)P_{uv(0,1)}+F_3(u)P_{uv(0,1)},\quad u\in[0,1] \tag{8.3.6}$$

现在我们就可以定义仅仅利用四个角点数据信息的插值曲面

$$\begin{aligned} S(u,v) &= [F_0(u)\ F_1(u)\ F_2(u)\ F_3(u)\ ]\cdot \\ &\cdot\begin{bmatrix} P_{00} & P_{01} & Q_0^v(0) & Q_0^v(1) \\ P_{10} & P_{11} & Q_1^v(0) & Q_1^v(1) \\ R_0^u(0) & R_0^u(1) & P_{uv(0,0)} & P_{uv(0,1)} \\ R_1^u(0) & R_1^u(1) & P_{uv(1,0)} & P_{uv(1,1)} \end{bmatrix}\begin{bmatrix} F_0(v) \\ F_1(v) \\ F_2(v) \\ F_3(v) \end{bmatrix} \end{aligned} \tag{8.3.7}$$

于是可得双三次混合孔斯曲面片

$$P(u,v) = Q(u,v) + R(u,v) - S(u,v) \tag{8.3.8}$$

此式定义的$P(u,v)$满足

$$\begin{cases} P(0,0) = P_{00},\ P(1,0) = P_{10}, \\ P(0,1) = P_{01},\ P(1,1) = P_{11}; \\ \\ P(u,0) = Q_0(u),\ P(u,1) = Q_0(1), \\ P(0,v) = R_0(v),\ P(1,v) = R_1(v); \\ \\ P_v(u,0) = Q_0^v(u),\ P_v(u,1) = Q_1^v(u), \\ P_u(0,v) = R_0^u(v), P_u(1,v) = R_1^u(v) \end{cases} \tag{8.3.9}$$

利用这些关系式可整理(8.3.7)和此式为

$$\begin{aligned} S(u,v) &= [F_0(u)\ F_1(u)\ F_2(u)\ F_3(u)\ ]\cdot \\ &\cdot\begin{bmatrix} P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1) \\ P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1) \\ P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1) \\ P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1) \end{bmatrix}\begin{bmatrix} F_0(v) \\ F_1(v) \\ F_2(v) \\ F_3(v) \end{bmatrix} \end{aligned} \tag{8.3.10}$$

和

$$\begin{aligned} P(u,v) &= [-1\ F_0(u)\ F_1(u)\ F_2(u)\ F_3(u)\ ]\cdot \\ &\cdot\begin{bmatrix} 0 & P(u,0) & P(u,1) & P_v(u,0) & P_v(u,1) \\ P(0,v) & P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1) \\ P(1,v) & P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1) \\ P_u(0,v) & P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1) \\ P_v(1,v) & P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1) \end{bmatrix}\begin{bmatrix} -1 \\ F_0(v) \\ F_1(v) \\ F_2(v) \\ F_3(v) \end{bmatrix} \end{aligned} \tag{8.3.11}$$

类似地,此式也可写为如下形式.

$$\begin{aligned} &[-1\ F_0(u)\ F_1(u)\ F_2(u)\ F_3(u)\ ]\cdot \\ &\cdot\begin{bmatrix} P(u,v) & P(u,0) & P(u,1) & P_v(u,0) & P_v(u,1) \\ P(0,v) & P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1) \\ P(1,v) & P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1) \\ P_u(0,v) & P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1) \\ P_v(1,v) & P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1) \end{bmatrix}\begin{bmatrix} -1 \\ F_0(v) \\ F_1(v) \\ F_2(v) \\ F_3(v) \end{bmatrix} = 0 \end{aligned} \tag{8.3.12}$$

对此曲面片,当沿公共边界具有公共跨界切向量的两相邻双三次混合孔斯曲面片沿公共边界拼合时，两曲面片沿整条公共边界就达到C^1连续,即一阶光滑拼接。

8.3.2 双三次孔斯曲面扭矢的估计

要想确定一双三次混合孔斯曲面片除已知四条边界曲线外，还要求提供四边界曲线上各点的跨界切向量及四个角点处的扭矢。这里先考虑扭矢。在没有扭矢(即四个角点处的二阶混合导向量和偏导向量)时,我们总可利用公式(8.2.11)产生一插值四条边界的双线性孔斯曲面.因此可考虑采取这一曲面的扭矢,如此定义的扭矢就是通常采用的所谓阿迪尼(Adini)扭矢。下面推导这一扭矢的计算公式.

求方程(8.2.11)的混合偏导向量，可得

$$P_{uv}(u,v)=P_{vu}(u,v)=P_v(1,v)-P_v(0,v)+P_u(u,1)-P_u(u,0)-C \quad (8.3.13)$$

其中

$$C=P(0,0)-P(0,1)-P(1,0)+P(1,1) \quad (8.3.14)$$

是由四个角点决定的张量积双线性曲面,即式(8.2.11)所用曲面$S(u,v)$ 的扭矢。分别用$u,v=0,1$代入上式可得四个角点处的扭矢为

$$\begin{cases} P_{uv}(0,0)=P_v(1,0)-P_v(0,0)+P_u(0,1)-P_u(0,0)-C \\ P_{uv}(0,1)=P_v(1,1)-P_v(0,1)+P_u(0,1)-P_u(0,0)-C \\ P_{uv}(1,0)=P_v(1,0)-P_v(0,0)+P_u(1,1)-P_u(1,0)-C \\ P_{uv}(1,1)=P_v(1,1)-P_v(0,1)+P_u(1,1)-P_u(1,0)-C \end{cases} \quad (8.3.15)$$

8.3.3 扭矢相容性

孔斯曲面片存在一个相容性问题。对于双线性孔斯曲面片来说，前已提及，四条边界曲线必须构成封闭曲边四边形,这就要求方程(8.2.11)中四个角点不是独立的。譬如$P(0,0)$，它既是边界曲线$P(u,0)$的端点，同时也是边界曲线$P(0,v)$的端点，或者说是两者的公共端点。因此,当独立的选择$v=0$和$u=0$对应的边界曲线时,为保证产生的曲面的连续性,这两条曲线边必须有一个公共端点. 其它三个角点也是这样。这就是相容性条件。这个位置连续条件较容易得到满足。

同样地，双三次混合孔斯曲面片也有四个角点的位置相容性问题，且另外还有更困难的四个角点的扭矢相容性问题。从方程(8.3.7)中的边界信息矩阵看，以角点$P(0,0)$ 处扭矢为例，它既应是$P_v(u,0)=Q_0^v(u)$对u求偏导后置$u=0$得到

$$P_{uv}(0,0)=\left.\frac{\partial Q_0^v(u))}{\partial u}\right|_{u=0} \quad (8.3.16)$$

也应是$P_u(0,v)=R_0^u(v)$对v求偏导后置$v=0$得到

$$P_{uv}(0,0)=\left.\frac{\partial R_0^u(v)}{\partial v}\right|_{v=0} \quad (8.3.17)$$

如果$P(u,v)$是二次连续可微的，则求导次序是可交换的，即有$P_{uv}=P_{vu}$, 然而这里的困难在于$P_v(u,0)=Q_0^v(u)$与$P_u(0,v)=R_0^u(v)$都是预先单独给定的，一般地情况下$P_{uv}(0,0)\neq P_{vu}(0,0)$。比如,当几个原来分离的物体拼接在一起时,有可能产生了一个需要修补的洞,这时各条曲线边是给定的,各条曲线边的位置可以产生变化,但各自的扭矢却不会变化,除非让物体产生几何形状的变化. 应用其中一个扭矢值所得曲面仅仅局部地插值给定的数据。孔斯没有意识到这个问题，这是后来由格里戈里(Gregory,1974)发现这一问题的。它表明角点扭矢不能独立地取某个指定的值。

有两种解决这一问题的方式: 一是实际情况许可时调整所给原始数据，以使不相容性消失.

再就是在原始数据不能改变的情况下，可以采用称之为格里戈里正方形的方法. 该方法使用可变扭矢替代前面(8.3.11)中的固定扭矢. 可变扭矢定义如下:

$$\begin{cases} P_{uv}(0,0) = \dfrac{u\frac{\partial P_u(0,0)}{\partial v} + v\frac{\partial P_v(0,0)}{\partial u}}{u+v} \\ P_{uv}(0,1) = \dfrac{-u\frac{\partial P_u(0,1)}{\partial v} + (v-1)\frac{\partial P_v(0,1)}{\partial u}}{-u+v-1} \\ P_{uv}(1,0) = \dfrac{(1-u)\frac{\partial P_u(1,0)}{\partial v} + v\frac{\partial P_v(1,0)}{\partial u}}{1-u+v} \\ P_{uv}(1,1) = \dfrac{(u-1)\frac{\partial P_u(1,1)}{\partial v} + (v-1)\frac{\partial P_v(1,1)}{\partial u}}{u-1+v-1} \end{cases} \tag{8.3.18}$$

如此生成的曲面在角点处的扭矢当然是不连续地. 如沿$u=0$对应的曲线边界线接近角点$P(0,0)=P_{00}$时, $P_{uv}(0,0) = \left.\dfrac{\partial Q_0^v(u))}{\partial u}\right|_{u=0}$; 沿$u=0$对应的曲线边界线接近角点$P(0,0)=P_{00}$时, $P_{uv}(0,0) = \left.\dfrac{\partial R_0^u(v)}{\partial v}\right|_{v=0}$.

无论如何, 切线连续是可以保证的. 角点处的扭矢不连续只是说明角点处不是二次连续可微的. 二次连续可微的性质在有些情况下也并非是必不可少的.

8.3.4 跨界切向量的确定

如果曲线边界线上各点处的切向量已经确定,就可以求得一个双三次孔斯曲面了. 否则,就必须寻找求解曲线边界线上各点处的切向量的方法. 切向量的确定远不如位置向量那么直观和易于确定. 因此要确定沿曲线边界线上无数个点处的切向量就更困难了. 但是, 在确定曲面$S(u,v)$时给定曲线边界线上各点处的切向量方法值得借鉴. 我们只需要有端点处的切向量和扭矢就可以了, 只是要注意每一点处有两个方向的切向量和扭矢. 这也就是说, 只要有$P_u(0,0)$, $P_u(0,1)$, $P_u(1,0)$, $P_u(1,1)$和$P_v(0,0)$, $P_v(0,1)$, $P_v(1,0)$, $P_v(1,1)$, 再结合$P_{uv(0,0)}$, $P_{uv(0,1)}$, $P_{uv(1,0)}$和$P_{uv(1,1)}$,我们就可以定义:

$$\begin{cases} Q_0^v(u) = F_0(u)P_v(0,0) + F_1(u)P_v(1,0) + F_2(u)P_{uv(0,0)} + F_3(u)P_{uv(1,0)}, \\ Q_1^v(u) = F_0(u)P_v(0,1) + F_1(u)P_v(1,1) + F_2(u)P_{uv(0,1)} + F_3(u)P_{uv(1,1)}, \\ R_0^u(u) = F_0(u)P_u(0,0) + F_1(u)P_u(0,1) + F_2(u)P_{uv(0,0)} + F_3(u)P_{uv(0,1)}, \\ R_1^v(u) = F_0(u)P_u(1,0) + F_1(u)P_u(1,1) + F_2(u)P_{uv(1,0)} + F_3(u)P_{uv(1,1)} \end{cases} \tag{8.3.19}$$

如此确定的跨界切向量还有一个优点就是不存在扭矢不相容问题,拼接出的多个这样的曲面片之间自动实现光滑拼接, 即达到C^1连续.

有一点需要注意的是所谓的双线性或双三次孔斯曲面并非真的是对两个参数分别为线性或三次函数,原因是生成曲面的曲线边界线函数可以为任意函数, 但是混合这些曲线边界线函数以生成曲面的函数对两个参数分别为线性或三次函数. 真正的对两个参数分别为线性或三次函数, 即所谓的双线性或双三次曲面是我们下面要讨论的问题.

8.4 双线性与双三次曲面

前两节讨论的孔斯曲面是基于已知曲线边界函数的条件进行的,但实际上也给出了仅知道四个角点处信息,而不知道曲线边界函数时生成曲面的方法. 这就是由式(8.2.7)和(8.3.7)生成的曲面$S(u,v)$.

8.4.1 双线性曲面定义及其表示

(8.2.7)中的$S(u,v)$对两个参数分别为线性,式子(8.2.7)的几何意义也非常明确,只需要知道四个角点的位置就可以了. 写成一般形式时有如下表达式:

$$\sum_{i=0}^{1}\sum_{j=0}^{1} A_{ij}u^i v^j, \quad (u,v)\in[0,1;0,1]. \tag{8.4.1}$$

该表达式清楚的表明确定(8.2.7)中的双线性曲面$S(u,v)$需要四组条件. 但常系数向量A_{ij} 的实际含义不清楚,因此不易确定. 故一般表达式不如式子(8.2.7)好用.

8.4.2 双三次曲面定义及其表示

(8.4.6)中的$S(u,v)$对两个参数分别为三次多项式,写成一般形式时有如下表达式:

$$S(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3} A_{ij}u^i v^j, \quad (u,v)\in[0,1;0,1] \tag{8.4.2}$$

可用矩阵表示为

$$\begin{aligned} S(u,v) \quad &= [1\ u\ u^2\ u^3]\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03}\\ A_{10} & A_{11} & A_{12} & A_{13}\\ A_{20} & A_{21} & A_{22} & A_{23}\\ A_{30} & A_{31} & A_{32} & A_{33}\end{bmatrix}\begin{bmatrix} 1\\ v\\ v^2\\ v^3\end{bmatrix}\\ &(u,v)\in[0,1;0,1]\end{aligned} \tag{8.4.3}$$

该表达式清楚的表明确定(8.3.7)中的双三次曲面$S(u,v)$需要十六组条件.但常系数向量A_{ij} 的实际含义不清楚,因此不易确定. 故一般表达式不如式子(8.3.7)好用.

下面我们推导两种表达式之间的关系. 注意到有

$$\begin{aligned} F(t) \quad &= [2t^3-3t^2+1 \quad 2t^3+3t^2 \quad t^3-2t^2+t \quad t^3-t^2]\\ &= [1\ \ t\ \ t^2\ \ t^3]\begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ -3 & 3 & -2 & -1\\ 2 & -2 & 1 & 1\end{bmatrix}\end{aligned} \tag{8.4.4}$$

其中的4×4矩阵用M表示,则有式子(8.3.7)可重新表示为

$$\begin{aligned} S(u,v) \quad &= [1\ \ u\ \ u^2\ \ u^3]M\cdot\\ &\cdot\begin{bmatrix} P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1)\\ P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1)\\ P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1)\\ P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1)\end{bmatrix}M'\begin{bmatrix} 1\\ v\\ v^2\\ v^3\end{bmatrix}\\ &(u,v)\in[0,1;0,1]\end{aligned} \tag{8.4.5}$$

反过来, 式子(8.4.3)也可用式子(8.4.6)的形式表示出来:

$$\begin{aligned} S(u,v) &= [F_0(u)\ F_1(u)\ F_2(u)\ F_3(u)\ ]M^{-1}\cdot \\ &\cdot \begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix} (M^{-1})' \begin{bmatrix} F_0(v) \\ F_1(v) \\ F_2(v) \\ F_3(v) \end{bmatrix} \end{aligned} \tag{8.4.6}$$

而应用式子(8.3.7)求解双三次曲面$S(u,v)$的最大的困难在于四个角点处扭矢的确定. 扭矢虽解释为相应点处曲面的扭曲程度的度量,扭曲程度却不易直观度量, 扭矢也就不易直观确定. 基于这样的原因, 扭矢常常取为0,这就是弗格森使用的双三次曲面. 但如此以来又造成了在角点处曲面过于平坦的“伪平”点. 下面我们将讨论用其它特定方式确定双三次曲面片的方法,期望在一定情况下能避免四个角点处扭矢的确定,甚或切向量确定的困难.

8.4.3 双三次曲面的其它形式

前面提到用式子(8.4.3)确定一双三次曲面$S(u,v)$需要十六组条件,但并不指定为什么样的十六组条件. 这样一来,我们就可以按照认为方便的方式指定十六组条件. 最简单的方式是指定曲面片通过已知的十六个点. 需要注意的是十六个点可以各自独立指定,要把各个点考虑为曲面片上一点,还需要各个点有相对应的一对参数值.

如果参数值也作为未知量进行求解,问题将非常复杂: 对空间曲面,每一点有x, y, z三个坐标的等式关系要成立,但同时又提供两个参数值作为自由变量,实际上每一点只提供了一个限制条件,为确定原来的十六组共四十八个条件,实际需要给出四十八个独立的点才能确定一个双三次曲面片. 同时求解的等式是多项式等式,多解的情况也不可避免. 由于点太多,每个点对曲面的影响也非常复杂,不利于对曲面片的修改.

基于上述原因, 我们一般认为十六个点形成网状分布, 参数取值为$u = u_0, u_1, u_2, u_3$ 和$v = v_0, v_1, v_2, v_3$, 即有$P_{ij} = S(u_i, v_j), i = 0,1,2,3; j = 0,1,2,3$. 于是由(8.4.2)可得十六组方程

$$\begin{cases} A_{00} + v_j^1 A_{01} + v_j^2 A_{02} + v_j^3 A_{03} + \\ u_i^1 A_{10} + u_i^1 v_j^1 A_{11} + u_i^1 v_j^2 A_{12} + u_i^i v_j^3 A_{13} + \\ u_i^2 A_{20} + u_i^2 v_j^1 A_{21} + u_i^2 v_j^2 A_{22} + u_i^i v_j^3 A_{23} + \\ u_i^3 A_{30} + u_i^3 v_j^1 A_{31} + u_i^3 v_j^2 A_{32} + u_i^3 v_j^3 A_{33} = Pij \\ i,j = 0,1,2,3 \end{cases} \tag{8.4.7}$$

定义16维向量

$$A = [A_{00}\ A_{01}\ A_{02}\ A_{03}\ \ A_{10}\ A_{11}\ A_{12}\ A_{13}\ \ A_{20}\ A_{21}\ A_{22}\ A_{23}\ \ A_{30}\ A_{31}\ A_{32}\ A_{33}] \tag{8.4.8}$$

和

$$P = [P_{00}\ P_{01}\ P_{02}\ P_{03}\ \ P_{10}\ P_{11}\ P_{12}\ P_{13}\ \ P_{20}\ P_{21}\ P_{22}\ P_{23}\ \ P_{30}\ P_{31}\ P_{32}\ P_{33}] \tag{8.4.9}$$

定义16×16矩阵,其$ij+1$行由(8.4.7)的P_{ij}对应的方程中各个未知量A_{mn}前面的系数依次组成.

$$B=\begin{bmatrix}
1 & v_0 & v_0^2 & v_0^3 & u_0 & u_0v_0 & u_0v_0^2 & u_0v_0^3\\
1 & v_1 & v_1^2 & v_1^3 & u_0 & u_0v_1 & u_0v_1^2 & u_0v_1^3\\
1 & v_2 & v_2^2 & v_2^3 & u_0 & u_0v_2 & u_0v_2^2 & u_0v_2^3\\
1 & v_3 & v_3^2 & v_3^3 & u_0 & u_0v_3 & u_0v_3^2 & u_0v_3^3\\
1 & v_0 & v_0^2 & v_0^3 & u_1 & u_1v_0 & u_1v_0^2 & u_1v_0^3\\
1 & v_1 & v_1^2 & v_1^3 & u_1 & u_1v_1 & u_1v_1^2 & u_1v_1^3\\
1 & v_2 & v_2^2 & v_2^3 & u_1 & u_1v_2 & u_1v_2^2 & u_1v_2^3\\
1 & v_3 & v_3^2 & v_3^3 & u_1 & u_1v_3 & u_1v_3^2 & u_1v_3^3\\
1 & v_0 & v_0^2 & v_0^3 & u_2 & u_2v_0 & u_2v_0^2 & u_2v_0^3\\
1 & v_1 & v_1^2 & v_1^3 & u_2 & u_2v_1 & u_2v_1^2 & u_2v_1^3\\
1 & v_2 & v_2^2 & v_2^3 & u_2 & u_2v_2 & u_2v_2^2 & u_2v_2^3\\
1 & v_3 & v_3^2 & v_3^3 & u_2 & u_2v_3 & u_2v_3^2 & u_2v_3^3\\
1 & v_0 & v_0^2 & v_0^3 & u_3 & u_3v_0 & u_3v_0^2 & u_3v_0^3\\
1 & v_1 & v_1^2 & v_1^3 & u_3 & u_3v_1 & u_3v_1^2 & u_3v_1^3\\
1 & v_2 & v_2^2 & v_2^3 & u_3 & u_3v_2 & u_3v_2^2 & u_3v_2^3\\
1 & v_3 & v_3^2 & v_3^3 & u_3 & u_3v_3 & u_3v_3^2 & u_3v_3^3
\end{bmatrix}$$

$$\begin{bmatrix}
u_0^2 & u_0^2v_0 & u_0^2v_0^2 & u_0^2v_0^3 & u_0^3 & u_0^3v_0 & u_0^3v_0^2 & u_0^3v_0^3\\
u_0^2 & u_0^2v_1 & u_0^2v_1^2 & u_0^2v_1^3 & u_0^3 & u_0^3v_1 & u_0^3v_1^2 & u_0^3v_1^3\\
u_0^2 & u_0^2v_2 & u_0^2v_2^2 & u_0^2v_2^3 & u_0^3 & u_0^3v_2 & u_0^3v_2^2 & u_0^3v_2^3\\
u_0^2 & u_0^2v_3 & u_0^2v_3^2 & u_0^2v_3^3 & u_0^3 & u_0^3v_3 & u_0^3v_3^2 & u_0^3v_3^3\\
u_1^2 & u_1^2v_0 & u_1^2v_0^2 & u_1^2v_0^3 & u_1^3 & u_1^3v_0 & u_1^3v_0^2 & u_1^3v_0^3\\
u_1^2 & u_1^2v_1 & u_1^2v_1^2 & u_1^2v_1^3 & u_1^3 & u_1^3v_1 & u_1^3v_1^2 & u_1^3v_1^3\\
u_1^2 & u_1^2v_2 & u_1^2v_2^2 & u_1^2v_2^3 & u_1^3 & u_1^3v_2 & u_1^3v_2^2 & u_1^3v_2^3\\
u_1^2 & u_1^2v_3 & u_1^2v_3^2 & u_1^2v_3^3 & u_1^3 & u_1^3v_3 & u_1^3v_3^2 & u_1^3v_3^3\\
u_2^2 & u_2^2v_0 & u_2^2v_0^2 & u_2^2v_0^3 & u_2^3 & u_2^3v_0 & u_2^3v_0^2 & u_2^3v_0^3\\
u_2^2 & u_2^2v_1 & u_2^2v_1^2 & u_2^2v_1^3 & u_2^3 & u_2^3v_1 & u_2^3v_1^2 & u_2^3v_1^3\\
u_2^2 & u_2^2v_2 & u_2^2v_2^2 & u_2^2v_2^3 & u_2^3 & u_2^3v_2 & u_2^3v_2^2 & u_2^3v_2^3\\
u_2^2 & u_2^2v_3 & u_2^2v_3^2 & u_2^2v_3^3 & u_2^3 & u_2^3v_3 & u_2^3v_3^2 & u_2^3v_3^3\\
u_3^2 & u_3^2v_0 & u_3^2v_0^2 & u_3^2v_0^3 & u_3^3 & u_3^3v_0 & u_3^3v_0^2 & u_3^3v_0^3\\
u_3^2 & u_3^2v_1 & u_3^2v_1^2 & u_3^2v_1^3 & u_3^3 & u_3^3v_1 & u_3^3v_1^2 & u_3^3v_1^3\\
u_3^2 & u_3^2v_2 & u_3^2v_2^2 & u_3^2v_2^3 & u_3^3 & u_3^3v_2 & u_3^3v_2^2 & u_3^3v_2^3\\
u_3^2 & u_3^2v_3 & u_3^2v_3^2 & u_3^2v_3^3 & u_3^3 & u_3^3v_3 & u_3^3v_3^2 & u_3^3v_3^3
\end{bmatrix} \tag{8.4.10}$$

如果定义矩阵

$$V=\begin{bmatrix}
1 & v_0 & v_0^2 & v_0^3\\
1 & v_1 & v_1^2 & v_1^3\\
1 & v_2 & v_2^2 & v_2^3\\
1 & v_3 & v_3^2 & v_3^3
\end{bmatrix}$$

则矩阵B有如下的分块表示:

$$B = \begin{bmatrix} V & u_0V & u_0^2V & u_0^3V \\ V & u_1V & u_1^2V & u_1^3V \\ V & u_2V & u_2^2V & u_2^3V \\ V & u_3V & u_3^2V & u_3^3V \end{bmatrix} \tag{8.4.13}$$

有了上述记号,式子(8.4.7)可用矩阵表示为

$$BA = P \tag{8.4.14}$$

现在求解双三次曲面就是求解一个线性方程组,解为

$$A = B^{-1}P. \tag{8.4.15}$$

实际的情况常常是比较简单一些:其中有十二个已知点是在曲面片的四条边界线上的,这时参数的取值为$u_0 = v_0 = 0;\ u_3 = v_3 = 1$,矩阵$V$和$B$可简化为:

$$V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & v_1 & v_1^2 & v_1^3 \\ 1 & v_2 & v_2^2 & v_2^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

则矩阵B有如下得分块表示:

$$B = \begin{bmatrix} V & 0 & 0 & 0 \\ V & u_1V & u_1^2V & u_1^3V \\ V & u_2V & u_2^2V & u_2^3V \\ V & V & V & V \end{bmatrix} \tag{8.4.18}$$

如果进一步选取$u_1 = v_1 = \dfrac{1}{3};\ u_2 = v_2 = \dfrac{2}{3}$为固定值, 则矩阵$V$和$B$就是常数矩阵了,求解过程也可以进一步简化.

需要注意的是用给定十六个点定义的曲面片拼接的曲面只能保证是连续的,不能保证是光滑的,即不能保证是C^1连续的.

8.5 Bézier曲面

8.5.1 Bézier曲面片的定义

由式子(8.4.6)定义双三次参数曲面的方式可以用来定义Bézier 曲面. 设有$(m+1)\times(n+1)$个点$P_{i,j}, i = 0,1,...,m;\ j = 0,1,...,n$.先固定$i$, 我们即有$(n+1)$个点$P_{i,j},\ j = 0,1,...,n$,可以生成一条$n$阶Bézier 曲线:

$$Q_i(u) = \sum_{j=0}^{n} B_{j,n}(u)P_{ij}, \qquad u \in [0,1] \tag{8.5.1}$$

按照此种方式,我们共定义出了$(m+1)$条Bézier曲线.

对任意固定的$u \in [0,1]$, 我们可以得到取自这$(m+1)$条Bézier曲线的$(m+1)$个点$Q_i(u)$, $i = 0, 1, ..., m$, 以这些点作控制点, 我们又可以得到一条m阶Bézier曲线:

$$\sum_{i=0}^{m} B_{i,m}(v)Q_i(u), \quad v \in [0,1] \tag{8.5.2}$$

把此式看作两个参数$(u,v) \in [0,1;0,1]$处的函数,就得到

$$\begin{aligned} P(u,v) = & \sum_{i=0}^{m} B_{i,m}(v)Q_i(u) \\ = & \sum_{i=0}^{m}\sum_{j=0}^{n} B_{i,m}(v)B_{j,n}(u)P_{ij}, \quad (u,v) \in [0,1;0,1] \end{aligned} \tag{8.5.3}$$

于是就定义了一个$m \times n$次Bézier曲面片。我们当然也可以按先固定j的方式生成一个Bézier曲面片方程, 其几何结果是完全相同的一个Bézier曲面片。直观上, Bézier曲面是一条Bézier曲线在空间按另一条Bézier曲线运动所形成的轨迹。

在此，我们先定义曲线，再通过“线动成面”的方法来定义Bézier曲面的，这种方式定义的曲面称为张量积曲面或笛卡尔积曲面。上式定义的是张量积Bézier曲面，它的两组基函数都是伯恩斯但基函数。

如同Bézier曲线有一个控制多边形一样，类似地Bézier曲面也有一个控制多面体，上面给出的P_{ij}是控制多面体的顶点，称为控制顶点，控制顶点沿i方向和j方向分别构成$m+1$个和$n+1$个控制多边形，它们一起组成曲面的控制多面体,也称控制网格。

式(8.5.3)中的$B_{i,m}(u)$和$B_{j,n}(v)$分别是m次和n次伯恩斯坦基函数，即

$$\begin{cases} B_{i,m}(u) = \dfrac{m!}{i!(m-i)!}u^i(1-u)^{m-i}, \\ B_{i,n}(v) = \dfrac{n!}{j!(n-j)!}v^j(1-v)^{n-j} \end{cases} \tag{8.5.4}$$

8.5.2 Bézier曲面片的性质

Bézier曲线的大部分性质都可应用于Bézier曲面, 这里介绍几条主要的性质.

1. Bézier曲面片的四个角点正好是相应的Bézier控制网格的四个角点, 即有

$$P(0,0) = P_{00}, P(1,0) = P_{m0}, P(0,1) = P_{0n}, P(1,1) = P_{mn}. \tag{8.5.5}$$

2. Bézier曲面片具有几何不变性.

3. Bézier曲面片具有凸包性质.

4. Bézier曲面片在角点处的切平面为由该角点及其相邻的两个点共三个点决定的平面. 如在角点P_{00}处的切平面为由P_{00},P_{01}和P_{10}三个点决定的平面.

5. $m \times n$次Bézier曲面片的四条边界曲线分别是m次和n次Bézier曲线.

对于低阶的Bézier曲面片我们还有如下一些结论:

6. 1×1的Bézier曲面片就是由第一节的式子(8.2.7)定义的曲面片$S(u,v)$,其边界为四条直线段.

7. 2×2的Bézier曲面片共有九个控制点确定, 周围的八个控制点确定了Bézier曲面片在四个角点处的切平面, 也确定了Bézier曲面片的四条边界线;中间的一个控制顶点P_{11}则指明了Bézier曲面片中间部分凸起或凹陷的方向即凸凹的程度.

值得一提的是不同于Bézier曲线，Bézier曲面不具有保凸性，即当控制多面体网格为凸时，相应的Bézier曲面可以不是凸的曲面。

8.5.3 双三次Bézier曲面

在实际应用中，用得最多的是3×3的Bézier曲面片，这时我们就称相应的Bézier曲面片为双三次Bézier曲面片。双三次Bézier曲面片的表达式为：

$$P(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}B_{i,3}(v)B_{j,3}(u)P_{ij},\quad (u,v)\in[0,1;0,1] \tag{8.5.6}$$

写成矩阵表达式为：

$$P(u,v)=[1\ u\ u^2\ u^3]APA'\begin{bmatrix}1\\ v\\ v^2\\ v^3\end{bmatrix},\quad (u,v)\in[0,1;0,1] \tag{8.5.7}$$

其中,

$$\begin{cases} A=\begin{bmatrix}1 & 0 & 0 & 0\\ -3 & 3 & 0 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{bmatrix},\\ P=\begin{bmatrix}P_{00} & P_{01} & P_{02} & P_{03}\\ P_{10} & P_{11} & P_{12} & P_{13}\\ P_{20} & P_{21} & P_{22} & P_{23}\\ P_{30} & P_{31} & P_{32} & P_{33}\end{bmatrix}. \end{cases} \tag{8.5.8}$$

矩阵P所包含的是控制点的位置向量，它们确定了一个多面体，同时也确定了一个Bézier曲面片。显然，只有四个角点真正在曲面上，边界线上的控制点确定了四个角点的切平面,同时也确定了边界曲线的切线。中间的四个控制点P_{11}、P_{12}、P_{21}和P_{22}将影响着曲面片四个角点处的混合偏导向量, 即扭矢。

双三次Bézier曲面片显然也是一种双三次曲面片. 两者相比较, 双三次Bézier曲面片有很多优点, 主要有如下几点: 双三次Bézier曲面片直接用十六个点给出表达式, 避免了确定切线向量和扭矢这些难以确定的量; 十六个点给出的控制多面体大致反映了Bézier曲面片的形状; 通过对控制顶点的直观修改、调整, 就能实现对Bézier曲面片修改、调整.

双三次Bézier曲面片虽有很多优点,但一般来说在曲面的设计中，一张双三次Bézier曲面片还不能完整描述一形状复杂的曲面，这时也需要把多张双三次Bézier曲面片拼接起来，组成组合曲面。此时，需要解决相邻两个Bézier 曲面片的光滑拼接问题。根据Bézier曲线连接的条件可以看到两个Bézier曲面片要求达到光滑连接，就需要在公共边界处有连续变化的法向量，即有连续变化的切平面，具体应满足：

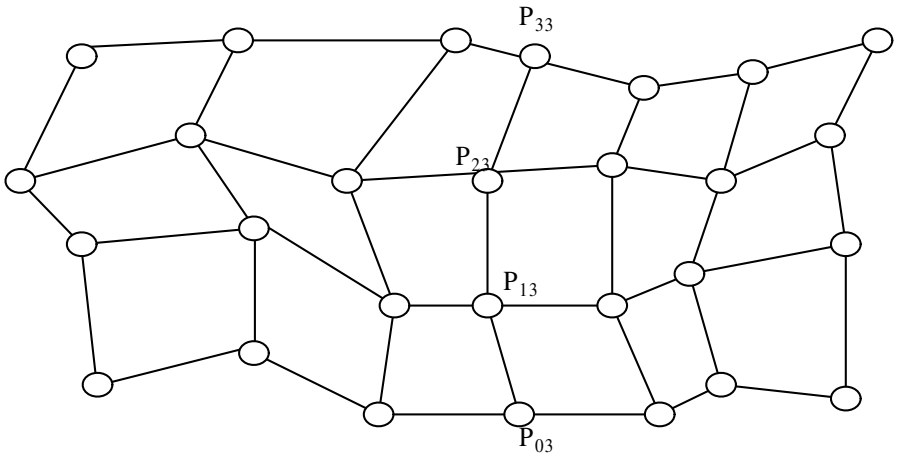


图8.3 两片Bézier曲面片的连接

(1). 如图8.3，公用一条边界曲线，即共同使用定义公共边界曲线的四个控制点:

$$P_{03},\ P_{13},\ P_{23},\ P_{33}. \tag{8.5.9}$$

这一条件保证了两个曲面片是位置连续的,即有共同的边界线;

(2). 公共边界两侧八个控制顶点和定义公共边界的四个控制顶点分为四组，每组三点，则每组中的三点共线。而且，定义公共边界的四个控制顶点分别把共线线段分成等比例，即有

$$\frac{P_{02}P_{03}}{P_{03}P_{04}} = \frac{P_{12}P_{13}}{P_{13}P_{14}} = \frac{P_{22}P_{23}}{P_{23}P_{24}} = \frac{P_{32}P_{33}}{P_{33}P_{34}} \tag{8.5.10}$$

8.6 B-样条曲面

8.6.1 B-样条曲面片的定义

类似于借助Bézier曲线生成Bézier曲面, 我们也可以借助于B-样条曲线生成B-样条曲面。设给定$(m+1)\times(n+1)$个控制顶点

$$P_{ij},\ i=0,1,...,\ m; j=0,1,...,n, \tag{8.6.1}$$

以及两列节点取值

$$u_0 \le u_1 \le ... \le u_{m+k+1} \tag{8.6.2}$$

和

$$v_0 \le v_1 \le ... \le v_{m+l+1}, \tag{8.6.3}$$

则可定义一个$k \times l$阶B-样条曲面

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} N_{ik}(u) N_{jl}(v) P_{ij}, \quad (u,v) \in [u_k, u_{m+1}; v_l, v_{n+1}] \tag{8.6.4}$$

其中$N_{ik}(u)$和$N_{jl}(v)$分别是由两列节点定义的$k-$阶和$l-$阶B-样条曲线时的B-样条基函数. 而$(m+1) \times (n+1)$个控制顶点构成的多面体称为这个B-样条曲面的控制网格或称控制多面体。控制多面体的形状大体上反映了B-样条曲面的形状.

类似于B-样条曲线的分类，B-样条曲面沿任一参数方向按节点序列取值不同可以划分成四种不同类型:均匀、准均匀、分片Bézier与非均匀B-样条曲面四种。沿两个参数方向也可选取不同类型。特殊地，若两个节点序列取值分别为

$$u_0 = u_1 = ... = u_{k+1} = 0, \quad u_{k+2} = u_{k+3} = ... = u_{2k+2} = 1 \tag{8.6.5}$$

和

$$u_0 = u_1 = ... = u_{l+1} = 0, \quad u_{l+2} = u_{l+3} = ... = u_{2l+2} = 1, \tag{8.6.6}$$

则所定义的B-样条曲面就是$k \times l$阶Bézier曲面。

8.6.2 双三次均匀B-样条曲面片公式

3×3阶均匀B-样条曲面片也称双三次均匀B-样条曲面片，当我们把其参数由一般长方形参数区间变换到标准单位正方形参数区间上时,其方程可表示为:

$$P(u,v) = [1 \ u \ u^2 \ u^3] APA' \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}, \quad (u,v) \in [0,1;0,1] \tag{8.6.7}$$

其中

$$\begin{cases} A = \dfrac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}, \\ P = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}, \end{cases} \tag{8.6.8}$$

而矩阵P所包含的是控制点的位置向量，它们确定了一个多面体，同时也确定了一个B-样条曲面片。控制顶点一般不在曲面片上，边界线上的控制点决定了曲面片的边界曲线为三次均匀B-样条曲线。

8.6.3 B-样条曲面片的优点

与Bézier曲面相比，B-样条曲面有以下主要优点：

(1). 多个B-样条曲面片的连接不需要考虑连接条件。当第一个曲面片被计算以后，不需要考虑连接条件，即可计算第二个曲面片，只是控制顶点矩阵P中的元素有部分改变。在双三次B-样条曲面上处处具有一阶和二阶连续性。因此各个相邻的双三次B-样条曲面片之间自动实现二阶连续性,也就不需要考虑双三次B-样条曲面片之间的光滑拼接问题. 利用双三次曲面片和双三次Bézier 曲面片解决这一问题都是很困难的.

(2). B-样条曲面的控制网格的顶点数量不受限制，因此，B-样条曲面可表示比Bézier曲面复杂得多的曲面。同时B-样条曲面的阶数不因控制顶点数目的增加而增加,保证了当控制顶点数目增加时, 不增加计算的复杂程度.

(3). B-样条曲面具有局部控制性质。当改变某个控制顶点时,仅那些与该顶点相关的几个B-样条曲面片的形状会发生变化, 其余的B-样条曲面片的形状不会发生任何变化.

(4). 像B-样条曲线一样, B-样条曲面也具有比Bzier曲面更强的凸包性质。

8.7 非均匀有理B-样条曲面

我们可以完全类似地定义非均匀有理B-样条曲面。设给定$(m+1)\times(n+1)$ 个控制顶点和每点对应的权因子

$$P_{ij},\ \omega_{ij}>0,\quad i=0,1,...,m;\ j=0,1,...,n, \tag{8.7.1}$$

以及两列节点取值

$$u_0\le u_1\le ...\le u_{m+k+1} \tag{8.7.2}$$

和

$$v_0\le v_1\le ...\le v_{m+l+1}, \tag{8.7.3}$$

则可定义一个$k\times l$阶非均匀有理B-样条曲面

$$P(u,v)=\frac{\sum_{i=0}^{m}\sum_{j=0}^{n}N_{ik}(u)N_{jl}(v)\omega_{ij}P_{ij}}{\sum_{i=0}^{m}\sum_{j=0}^{n}N_{ik}(u)N_{jl}(v)\omega_{ij}},\quad (u,v)\in[u_k,u_{m+1};v_l,v_{n+1}] \tag{8.7.4}$$

其中$N_{ik}(u)$和$N_{jl}(v)$是由两列节点定义非均匀有理B-样条曲线时的非均匀有理B-样条基函数. 而$(m+1)\times(n+1)$个控制顶点构成的多面体称为这个非均匀有理B-样条曲面的控制网格或称控制多面体。控制多面体的形状大体反映了非均匀有理B-样条曲面的形状.

特殊地，若两个节点序列取值分别为

$$u_0=u_1=\cdots=u_{k+1}=0,\ u_{k+2}=u_{k+3}=\cdots=u_{2k+2}=1 \tag{8.7.5}$$

和

$$u_0=u_1=\cdots=u_{l+1}=0,\ u_{l+2}=u_{l+3}=\cdots=u_{2l+2}=1, \tag{8.7.6}$$

则所定义的非均匀有理B-样条曲面就是$k\times l$阶有理Bézier曲面。

3×3阶非均匀有理B-样条曲面片也称双三次非均匀有理B-样条曲面片，当我们把其参数由一般长方形区间变换到标准单位正方形区间上时,其方程可表示为:

$$P(u,v)=\frac{[1\ u\ u^2\ u^3]APA'\begin{bmatrix}1\\v\\v^2\\v^3\end{bmatrix}}{[1\ u\ u^2\ u^3]AWA'\begin{bmatrix}1\\v\\v^2\\v^3\end{bmatrix}},\quad (u,v)\in[0,1;0,1] \tag{8.7.7}$$

其中

$$\begin{cases} P = \begin{bmatrix} \omega_{00}P_{00} & \omega_{01}P_{01} & \omega_{02}P_{02} & \omega_{03}P_{03} \\ \omega_{10}P_{10} & \omega_{11}P_{11} & \omega_{12}P_{12} & \omega_{13}P_{13} \\ \omega_{20}P_{20} & \omega_{21}P_{21} & \omega_{22}P_{22} & \omega_{23}P_{23} \\ \omega_{30}P_{30} & \omega_{31}P_{31} & \omega_{32}P_{32} & \omega_{33}P_{33} \end{bmatrix}, \\ A = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}, \\ W = \begin{bmatrix} \omega_{00} & \omega_{01} & \omega_{02} & \omega_{03} \\ \omega_{10} & \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{20} & \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{30} & \omega_{31} & \omega_{32} & \omega_{33} \end{bmatrix}, \end{cases} \tag{8.7.8}$$

其中矩阵P所包含的是控制点的位置向量与相应权因子的乘积.

与双三次非均匀有理B-样条曲面片相对应的, 我们也可定义出有理双三次Bézier曲面, 其表达式完全类似于双三次有理均匀B-样条曲面, 仅需把上述公式中的矩阵A用下面的矩阵代替即可:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}. \tag{8.7.9}$$

控制顶点一般不在非均匀有理B-样条曲面片上，但四个角点总是相应的有理Bézier曲面的角点。

与多项式的这一类曲面相比, 我们可以用有理B-样条或Bézier曲面精确地表示出球面, 以及其他类型的包含圆弧线的曲面, 如圆柱面, 圆锥面及旋转体面等等. 这些都是利用计算机进行辅助设计时经常用到的图形.

8.8 三角域上的Bézier曲面

8.8.1 三角域内的重心坐标

在直线上，若已知两点A, B, 则两点连线上任一点D可表示为

$$D = (1-t)A + tB, \quad 0 \le t \le 1. \tag{8.8.1}$$

现在假设在平面上已知三点A, B, C, 则三点所连三角形内任一点P 该如何表示呢？设O点为坐标原点，回忆直线的情况，上式可用向量形式表示为

$$\overrightarrow{OD}=\overrightarrow{OA} + \overrightarrow{AD}= (1-t)\ \overrightarrow{OA} + t\ \overrightarrow{OB}, \quad 0 \le t \le 1, \tag{8.8.2}$$

此时，参数t页有清楚的几何意义：向量$\overrightarrow{AD}$与向量$\overrightarrow{AB}$长度的比值。

当一点P在三角形$\triangle ABC$内时，类似可得

$$\overrightarrow{OP}=\overrightarrow{OA} + \overrightarrow{AP}=\overrightarrow{OA} + \overrightarrow{AD} + \overrightarrow{DP}\ . \tag{8.8.3}$$

于是，存在$0 \le s \le 1$和$0 \le t \le 1$使得

$$\begin{cases} \overrightarrow{AD} = s\ \overrightarrow{AB} = s(\overrightarrow{OB} - \overrightarrow{OA}), \\ \overrightarrow{DP} = \overrightarrow{AE} = t\ \overrightarrow{AC} = t(\overrightarrow{OC} - \overrightarrow{OA}). \end{cases} \tag{8.8.4}$$

代入(8.8.3)得

$$\overrightarrow{OP} = (1 - s - t)\ \overrightarrow{OA} + s\ \overrightarrow{OB} + t\ \overrightarrow{OC}\ . \tag{8.8.5}$$

如果把$\triangle ABC$同时在理解为一个数值时，看作是这个三角形的面积，则

$$s = \frac{|\ \overrightarrow{AE}\ |}{|\ \overrightarrow{AC}\ |} = \frac{\triangle APC}{\triangle ABC},\quad t = \frac{\triangle ABP}{\triangle ABC},\quad 1 - s - t = \frac{|\ \overrightarrow{AD}\ |}{|\ \overrightarrow{AB}\ |} = \frac{\triangle PBC}{\triangle ABC} \tag{8.8.6}$$

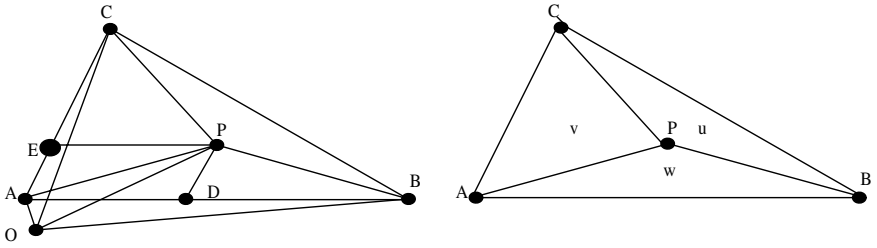


图8.4 三角域内的重心坐标定义

通常，我们用点坐标的形式表示出上述关系：

$$P = uA + vB + wC, \tag{8.8.7}$$

其中

$$0 \le u \le 1,\quad 0 \le v \le 1,\quad 0 \le w \le 1,\quad u + v + w = 1. \tag{8.8.8}$$

并称(u, v, w)为点P关于三点A, B, C的重心坐标。这个坐标从力学原理上可理解为三点A, B, C各有质量u, v, w时，其重心在P. 依据式(8.8.6), 只要关于三点A, B, C不共线，点P的重心坐标(u, v, w) 就存在。

另外，当$\triangle ABC = 1$时，点P重心坐标三个分量u, v, w就是如图8.4所示点P与$\triangle ABC$三个顶点连线形成的三个小三角形的面积。

限制条件$0 \le u \le 1,\quad 0 \le v \le 1,\quad 0 \le w \le 1$是为了保证点$P$位于三角形内。不满足此条件的重心坐标$(u, v, w)$仍可定义出平面上一点，但位于指定的三角形之外。根据重心坐标的几何意义不难给出其由三点表示的表达式。由于涉及到线段长度，其表达式很容易遇到开方运算。如果点是平面坐标系内的点，有二维坐标表示：

$$A = (a_x, a_y),\quad B = (b_x, b_y),\quad C = (c_x, c_y) \tag{8.8.9}$$

则三角形的有向面积为

$$\triangle ABC = \frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ a_x & b_x & c_x \\ a_y & b_y & c_y \end{vmatrix}. \tag{8.8.10}$$

对于这个有向面积计算公式，无论点P位于三角形内或外，始终成立如下表达式。

$$u = \frac{\triangle PBC}{\triangle ABC}, \quad v = \frac{\triangle APC}{\triangle ABC}, \quad w = \frac{\triangle ABP}{\triangle ABC} \tag{8.8.11}$$

8.8.2 三角域上的Beinstein函数

单变量的n次Beinstein函数由n次二项式$(t + (1 - t))^n$的展开式的各项构成。类似地，双变量的n次Beinstein函数可以由n次三项式$(u + v + (1 - u - v))^n$的展开式的各项构成。为方便起见，令$w = 1 - u - v$, 得展开式

$$(u + v + w)^n = \sum_{i=0}^{n} \sum_{j=0}^{n-i} \frac{n!}{i!j!(n-i-j)!} u^i v^j w^{n-i-j} \tag{8.8.12}$$

由此定义三角域上的n次Beinstein函数

$$B_{i,j,k}(u, v, w) = \frac{n!}{i!j!k!} u^i v^j w^k, \tag{8.8.13}$$

其中，

$$\begin{cases} 0 \le u, v, w \le 1, \quad u + v + w = 1, \\ 0 \le i, j, k \le n, \quad i + j + k = n. \end{cases} \tag{8.8.14}$$

依据i, j, k满足的条件可以推得三角域上的n次Beinstein函数共有$\frac{1}{2}(n + 1)(n + 2)$个。这些函数可以直观的认为分布在如图8.5所示的三角阵列中。

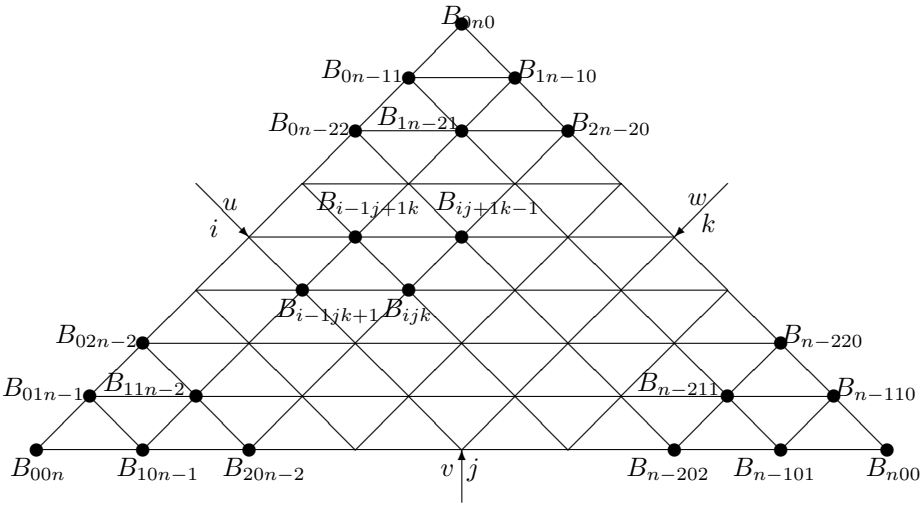


图8.5 三角域上的n次Beinstein函数的分布

不难看出，三角形的三条边上分别对应着$u = 0$, $v = 0$和$w = 0$. 分别与$u = 0$, $v = 0$和$w = 0$相应的边平行的直线上，相应的u, v或w 保持不变，因此称为等参数线。相应于某条边上的所有n次Beinstein函数正好是单变量的所有n次Beinstein函数。三角域上的n次Beinstein函数具有类似于单变量的n次Beinstein函数的如下性质。

1. 规范性

$$\sum_{i=0}^{n}\sum_{j=0}^{n-i}\frac{n!}{i!j!(n-i-j)!}u^iv^jw^{n-i-j}=(u+v+w)^n=1. \tag{8.8.15}$$

2. 非负性

$$B_{i,j,k}(u,v,w)=\frac{n!}{i!j!k!}u^iv^jw^k\geq 0. \tag{8.8.16}$$

3. 递推性

$$\begin{aligned}B^n_{i,j,k}(u,v,w)=\quad & uB^{n-1}_{i-1,j,k}(u,v,w)+\\ & vB^{n-1}_{i,j-1,k}(u,v,w)+\\ & wB^{n-1}_{i,j,k-1}(u,v,w).\end{aligned} \tag{8.8.17}$$

三角域内直线的交点称为节点，用i, j, k标示，即认为节点具有坐标i,j,k。

8.8.3 三角域上的Bézier曲面

类似于Bézier曲线的定义，我们在三角域内个每节点i,j,k对应一个控制点$P_{i,j,k}$，则可立即写出一个曲面片表达式：

$$P(u,v,w)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}B_{i,j,k}(u,v,w)P_{i,j,k}, \tag{8.8.18}$$

其中，

$$\begin{cases}0\leq u,v,w\leq 1,\quad u+v+w=1,\\ 0\leq i,j,k\leq n,\quad i+j+k=n.\end{cases} \tag{8.8.19}$$

这就是三角域上的Bézier曲面。当所有分配在节点处的控制点$P_{i,j,k}$按照图8.5所示的节点连接方式连接在一起时，就形成了一张由三角平面块构成的一个多面体，这就是三角域上的Bézier曲面的控制多面体，或称控制网格。下图显示了二次和三次的三角域上的Bézier曲面及控制多面体。

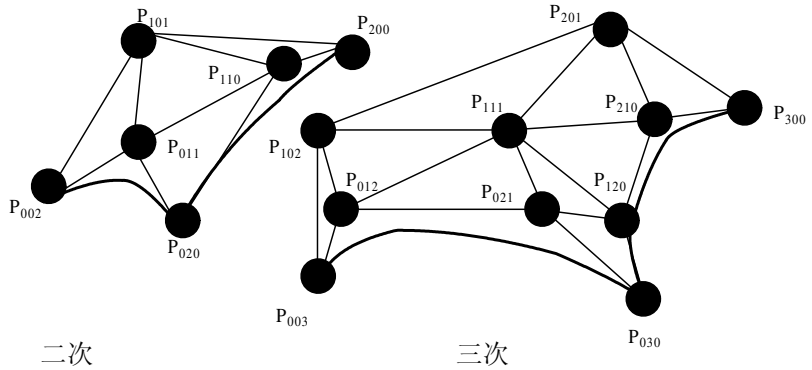


图8.6 三角域上的Bézier曲面及控制多面体

由此图可以看出，虽然从坐标平面看这里的Bézier曲面定义在严格的三角形区域内，实际的控制多面体可以有复杂的形状，相应的Bézier曲面也并不局限于一个严格的三角形区域内。

特别地，假设平面上三个点$A=(a_x,a_y),\ \ B=(b_x,b_y),\ \ C=(c_x,c_y)$, 控制点$P_{i,j,k}=(x_{i,j,k},y_{i,j,k},z_{i,j,k})$满足

$$(x_{i,j,k},y_{i,j,k})=\frac{i}{n}A+\frac{j}{n}B+\frac{k}{n}C, \tag{8.8.20}$$

则曲面上任一点$P(u,v,w)=(x(u,v,w),y(u,v,w),z(u,v,w))$满足

$$\begin{aligned}(x(u,v,w),y(u,v,w)) &= uA+vB+wC\\ &= (ua_x+vb_x+wc_x,ua_y+vb_y+wc_y).\end{aligned} \tag{8.8.21}$$

这也就是说此时参数平面重合于坐标平面Oxy。相应的Bézier曲面可以认为是在三角域内的每个节点处设定具有给定数值的控制点，然后依据这些点产生一个曲面。如此产生的曲面称为函数型三角域上的Bézier曲面。

8.8.4 三角域上的Bézier曲面的方向导向量

由于三角域上的Bézier曲面的三个参数相互依赖，对某个参数求导在这里没有明确的意义。为了明确求导的实际含义，这里指定求某个指定方向的导向量。设有参数三角形一点$P_0=(u_0,v_0,w_0)$, 及方向向量$R=(r_u,r_v,r_w)$. 由此确定过点P_0，具有指定方向R的直线上一点(u,v,w)可表示为为

$$(u,v,w)=L(\lambda)=P_0+\lambda R=(u_0+\lambda r_u,v_0+\lambda r_v,w_0+\lambda r_w) \tag{8.8.22}$$

于是

$$\begin{aligned}P(u,v,w) &= P(u_0+\lambda r_u,v_0+\lambda r_v,w_0+\lambda r_w)\\ &= \sum_{i=0}^{n}\sum_{j=0}^{n-i}B_{i,j,k}(u_0+\lambda r_u,v_0+\lambda r_v,w_0+\lambda r_w)P_{i,j,k},\end{aligned} \tag{8.8.23}$$

对λ求导可得方向到向量

$$\frac{dP(u,v,w)}{d\lambda}=n\sum_{i=0}^{n-1}\sum_{j=0}^{n-1-i}B_{i,j,k}^{n-1}(u_0+\lambda r_u,v_0+\lambda r_v,w_0+\lambda r_w)P_{i,j,k}^{1}, \tag{8.8.24}$$

其中，

$$\begin{cases}P_{i,j,k}^{1}=r_uP_{i+1,j,k}+r_vP_{i,j+1,k}+r_wP_{i,j,k+1}\\ i+j+k=n-1,\ \ 0\le i,j,k\ge 0.\end{cases} \tag{8.8.25}$$

需要提醒的是由于重心坐标的三个分量之和式中为1，作为方向向量的$R=(r_u,r_v,r_w)$的三个坐标分量之和必然为零，即有$r_u+r_v+r_w=0$。

反复求导，对$m=1,2,\cdots,n$可得一般的m阶方向到向量

$$\frac{d^mP(u,v,w)}{d\lambda^m}=\frac{n!}{(n-m)!}\sum_{i=0}^{n-m}\sum_{j=0}^{n-m-i}B_{i,j,k}^{n-m}(u,v,w)P_{i,j,k}^{m}, \tag{8.8.26}$$

其中，

$$\begin{cases}P_{i,j,k}^{0}=P_{i,j,k}\\ P_{i,j,k}^{m}=r_uP_{i+1,j,k}^{m-1}+r_vP_{i,j+1,k}^{m-1}+r_wP_{i,j,k+1}^{m-1}\\ i+j+k=n-m,\ \ 0\le i,j,k\ge 0.\end{cases} \tag{8.8.27}$$

利用此公式可求得对u, v, w任意参数方向的导向量。如为了求u参数方向的导向量，可取$R = (r_u, r_v, r_w)$中的$r_u = 1$，另外两个参数由一个保持不变，即有一个分量为零。由于三个坐标分量之和必然为零，第三个分量一项为-1。因此可取$R = (r_u, r_v, r_w) = (1, 0, -1)$得到

$$P^m_{i,j,k} = P^{m-1}_{i+1,j,k} - P^{m-1}_{i,j,k+1} \tag{8.8.28}$$

为求沿$v = 0$对应的三角形的边方向的导向量需要的点的递推计算公式。取$R = (r_u, r_v, r_w) = (1, -1, 0)$得到

$$P^m_{i,j,k} = P^{m-1}_{i+1,j,k} - P^{m-1}_{i,j+1,k} \tag{8.8.29}$$

为求沿$w = 0$对应的三角形的边方向的导向量需要的点的递推计算公式。

8.8.5 三角域上的Bézier曲面的性质

与定义在矩形域上的Bézier曲面相比, 三角域上的Bézier曲面有如下不同：

1. 参数平面上的定义域结构不同，定义方式也不同。矩形域上的Bézier曲面是由两参数的Bézier曲线做张量乘积得到的，三角域上的Bézier曲面是依据重心坐标得到的，与某个参数的Bézier曲线没有直接关系。

2. 控制点间的相互关系不同，因而控制网格也不同。矩形域上的Bézier曲面的控制点在两参数方向是相对独立变化的，而三角域上的Bézier曲面控制点实际上也只有两个独立的参数，控制点在各个参数方向变化方式是完全相同的。

3. 矩形域上的Bézier曲面对两参数的依赖关系是不同的，三角域上的Bézier曲面对各个参数是完全对等的。

但是两者还是有一些共同的性质的。这一点可以从下面给出的三角域上的Bézier曲面片的性质看出。

1. Bézier曲面片的三个角点正好是相应的Bézier控制网格的三个角点, 即有

$$P(1,0,0) = P_{n00}, P(0,1,0) = P_{0n0}, P(0,0,1) = P_{00n}. \tag{8.8.30}$$

2. Bézier曲面片具有几何不变性. 这可由n次Beinstein函数的规范性得出。

3. Bézier曲面片具有凸包性质. 这可由n次Beinstein函数的规范性和规范性得出。

4. Bézier曲面片在角点处的切平面为由该角点及其相邻的两个点共三个点决定的平面. 如在角点P_{n00}处的切平面为由P_{n00},P_{n-101}和P_{n-110}三个点决定的平面.

5. n次Bézier曲面片的三条边界曲线分别都是n次Bézier曲线，其控制点就是沿相应边分布的，用来定义角域上的Bézier曲面的那些控制点。

值得一提的是一般的三角域上的Bézier曲面也不具有保凸性，即当控制多面体网格为凸时，相应的Bézier曲面可以不是凸的曲面。但是，特殊的函数型的三角域上的Bézier曲面具有保凸性。

习题8

1. 求出如图所示两条半圆弧线即退缩为两点的两条曲线给定的四条插值边界曲线的双线性孔斯曲面.

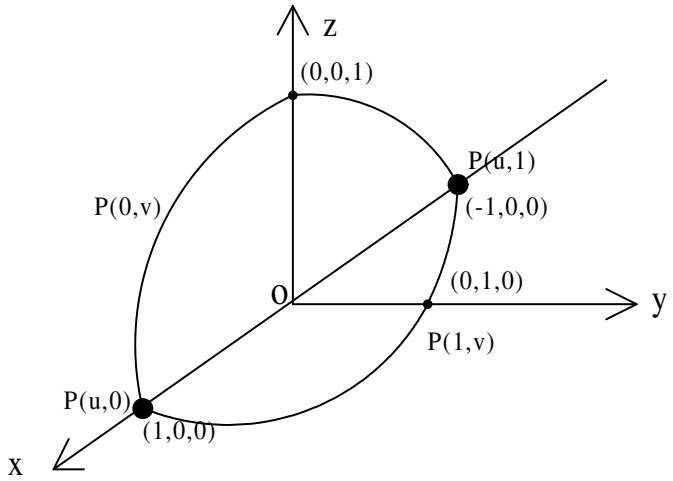


图8.7 习题1四条插值边界曲线示意图

2. 编写一段程序,计算并显示出任意给定四点的的双线性孔斯曲面. 观察其与平面有什么不同.

3. 在半球面上指定四点,并以连接角点间的球面圆弧线作插值边界曲线.求出相应的双线性孔斯曲面.

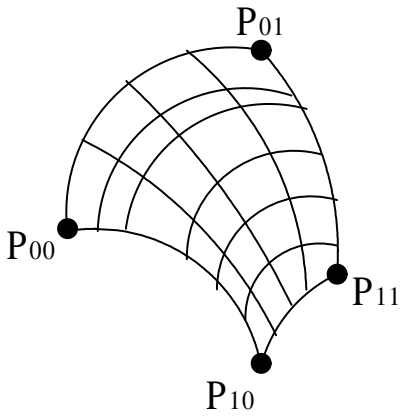


图8.8 球面示意图

4. 求出习题3相应的双三次孔斯曲面.其中计算所需要的当向量与球面的导向量相同.

5. 求出前面习题中相应的双线性和双三次曲面,它们与相应的双线性和双三次孔斯曲面是否相同?

6. 求出与前面习题中的双三次曲面相同的双三次Bézier曲面。求出生成双三次Bézier曲面的16个控制点。

7. 求出与前面习题中的双三次曲面相同的双三次均匀B-样条曲面。求出生成双三次均匀B-样条曲面的16个控制点。

8. 给定三个点$A(-1,\ 0)$,$B(1,\ 0)$,$C(0,\ 1)$, 并定义$P_{i,j,k} = (x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$满足

$$(x_{i,j,k}, y_{i,j,k}) = \frac{iA + jB + kC}{n}, z_{i,j,k} = \sqrt{1 - x_{i,j,k}^2 - y_{i,j,k}^2} \qquad (8.8.31)$$

分别求$n = 1,\ 2,\ 3$时定义在三角形ABC上的n次三角形Bézier曲面。

Chapter 9

计算机图形中消隐处理

9.1 引言

现实的物体是三维立体的, 有不同的侧面. 而我们在指定的时间只能观察物体的一个侧面. 计算机内部描述一个物体时, 可以完整的描述三维物体的各个详细的构成部分,包括其各个侧面,其内部结构,以及我们通常看不到的外部结构. 但当我们要借助计算机直观地去考察这个物体时,同考察物体的实物一样,在指定的时间只能观察物体的一个侧面. 作为计算机的处理结果,就是只能显示出我们能观察到的实物的侧面. 否则, 显示就会是不真实的,杂乱无章的.

当我们从一个侧面观察物体时, 看不见的部分就称为隐藏部分, 因此就有隐藏线, 隐藏面等等. 消隐处理就是要查找出物体所有的隐藏线或隐藏面,并进行相应的处理. 通常有两种处理方式: 把物体上看不见的隐藏线或面从画面中消去或者用虚线画出。但主要的工作都是一样的,那就是查找出物体所有的隐藏线或隐藏面.

消隐处理是三维物体的计算机图形显示技术中的一个基本问题，其原理并不复杂. 对于消去隐藏线，只要把表示三维物体的每一条线与每一个组成物体的不透明面进行可见性判断，把不可见线段部分与可见线段部分区别开来，画出可见线段部分，不画不可见线段部分。对于消去隐藏面，则把每一个组成物体的面与每一个不透明面进行可见性判断，把不可见面部分与可见面部分区别开来，画出可见面部分，不画不可见面部分。

虽然消隐的原理很简单，但实际上消隐算法却是十分复杂的。特别是在大多数情况下,我们可能需要观察物体的不同的侧面。随着要观察的物体侧面的不同,物体的不可见部分,即隐藏线和隐藏面也在不断变化,并且这种变化一般是前后没有明显关联规律的。因此每改变一个侧面,就需要重新查找相应的隐藏线和隐藏面,这导致了消隐处理常常需要耗费较多的机时.

9.2 凸多面体的消隐

所谓凸多面体是指由多个平面凸多边形包围而形成的, 没有凹陷的物体。直观来看, 也就是表面任何部分都没有凹陷下去的多面体. 每个凸多边形平面都有各自的法线，这些法线的方向规定为：由物体内部指向外部空间。通常我们认为这个多面体是不透明的,因此其内部是永远看不到的.

规定了各个凸多边形平面的方向后, 我们可以看到这个平面块正面, 在其法线方向指向观察者时是可见的, 否则是不可见的, 即为隐藏面, 具体的细节随后介绍.

单独一个凸多面体的消隐处理比较简单：对凸多面体的每个平面凸多边形进行可见性检测，把可见平面凸多边形与不可见平面凸多边形区分开，只画出可见的平面凸多边形即可。

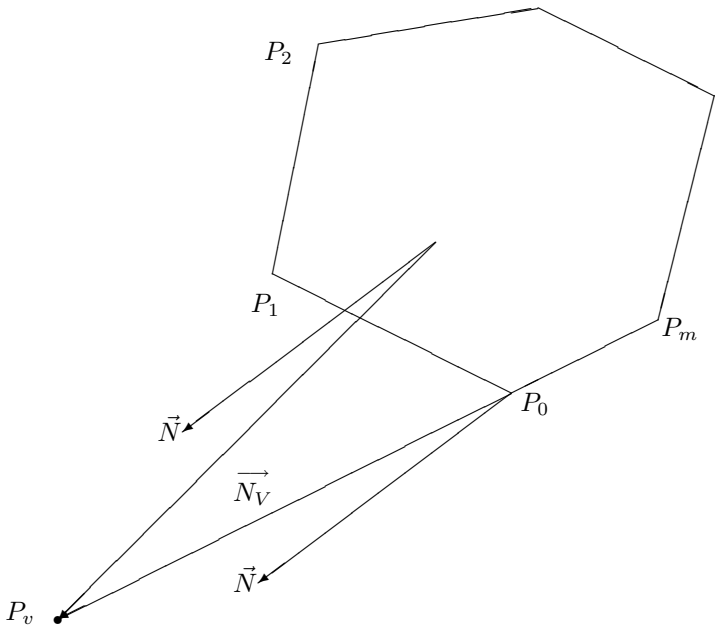


图9.1 凸多面体的一个面及其法向量, 观测方向向量

组成凸多面体的平面凸多边形的可见性检测可用每个平面凸多边形的两个向量来确定：一个是平面凸多边形的法向量, 另一个是观察方向向量。

观察方向是从要观察的物体表面指向观察者的方向。对观察者需要确定的是其所在的空间位置,即观察点$P_v(x_v, y_v, z_v)$. 于是, 一个具体的观察方向向量可以是要观察的那部分平面上任一点指向观察点的向量。

通过法向量和观察方向向量这两个向量的夹角的大小，我们可对相应的平面凸多边形的可见性作出判断：平面是否可见等价于这两个向量是否指向平面的同一侧或者其夹角是否不超过$\frac{\pi}{2}$。

下面首先考虑一片平面凸多边形的法线方向和观察方向向量的计算.

图9.1所示为凸多面体的一个面—平面凸多边形。对于这个平面凸多边形，设有$n+1$个顶点

$$P_i,\ i = 0, 1, \cdots, n. \tag{9.2.1}$$

规定$n+1$个顶点排列顺序满足如下要求: 当观察者在多面体边界上时, 观察者向上站立的方向与平面法向量同向, 并且按顺序

$$P_0 \to P_1 \to P_2 \to \cdots \to P_n \tag{9.2.2}$$

沿边界走动的过程中多边形的内部始终在观察者的右侧.

在上述假设下, 就可以求解法向量了。结合两个向量的向量积的意义,此平面凸多边形的法向量$\overrightarrow{N}$ 与向量$P_0\overrightarrow{P_n} \times P_0\overrightarrow{P_1}$同向.

理论上有很多对向量的向量积与法向量同向. 对实际问题而言,多面体的每一个面都可能很小, 因而边很短. 这导致了用向量$P_0\overrightarrow{P_n} \times P_0\overrightarrow{P_1}$ 或任何其它一对向量的向量积来判定相应面的法向量时有可能由于计算误差而造成法向量方向的错误判定. 为避免此问题, 我们可把多个这种理论上应当同方向的向量求和, 因此法向量通常可用下式计算.

$$\overrightarrow{N} = \sum_{i=1}^{n-1} P_0\overrightarrow{P_{i+1}} \times P_0\overrightarrow{P_i} \tag{9.2.3}$$

观察方向向量则容易求得。不妨取平面凸多边形起点P_0作为观察方向向量V的起点，则

$$\overrightarrow{N_V}=P_0\overrightarrow{P_v}\ . \tag{9.2.4}$$

平面凸多边形的可见性检测规则为：

(1). 当观察者位于平面凸多边形的法向量所指的一侧时, 该平面凸多边形可见. 观察者位于该平面的法向量N所指的一侧等价于$\overrightarrow{N}$和$\overrightarrow{N_V}$指向该平面的同一侧. 因此这两个向量的夹角不超过$\frac{\pi}{2}$.

(2). 当观察者不位于平面凸多边形的法向量所指的一侧时，则该平面凸多边形不可见。此处的条件等价于$\overrightarrow{N}$和$\overrightarrow{N_V}$这两个向量的夹角超过$\frac{\pi}{2}$.

根据两个向量的数量积的定义,$\overrightarrow{N}$和$\overrightarrow{N_V}$这两个向量的夹角是否超过$\frac{\pi}{2}$等价于它们的数量积$\overrightarrow{N} \cdot \overrightarrow{N_V}$是否大于0. 于是可见性的判定规则为

(1). 若$\overrightarrow{N} \cdot \overrightarrow{N_V} \geq 0$，则相应的平面凸多边形可见；

(2). 若$\overrightarrow{N} \cdot \overrightarrow{N_V} < 0$，则相应的平面凸多边形不可见。

若观察方向向量采用坐标轴方向为观察方向，那计算就更加简单了。虽然消隐的物体是立体的,但消隐后总要在二维的平面上显示出来. 伴随着消隐处理, 就总是存在着三维物体投影到平面的投影变换.

为了处理的方便, 通常取投影平面为Oxy平面, 投影方向就是z坐标轴的负方向,这时的观察方向就是z坐标轴的正方向, 因此可取$\overrightarrow{N_V} = (0, 0, 1)$. 在此假设下, $\overrightarrow{N} \cdot \overrightarrow{N_V}$ 等于向量$\overrightarrow{N}$的z坐标分量. 于是，平面凸多边形是否可见等价于向量$\overrightarrow{N}$的z坐标分量是否非负。

上述结论只对单独一个凸多面体组成的物体成立. 如果物体由两个以上的凸多面体组成,对上述方法判定为可见的任何一平面凸多边形,还需要判定其是否被属于另一凸多面体的可见部分遮挡,是被部分遮挡,还是被完全遮挡. 不是凸的多面体也可以被分解为多个凸多面体进行处理. 更详细的内容请参阅相关的书籍.

9.3 函数曲面的消隐

曲面的消隐处理起来比多面体复杂得多。曲面消隐有两种方法：一种是用平面来逼近曲面，然后进行消隐处理，其方法与多面体的消隐处理一样，但是这种方法计算量和数据存贮量很大。

另一种是直接对曲面进行消隐处理。本节将介绍函数曲面消隐和参数曲面消隐。

9.3.1 函数曲面消隐

本小节仅讨论显式方程

$$y = f(x, z) \tag{9.3.1}$$

所表示的曲面的消隐处理. 这种类型的曲面经常出现在实际问题和科学研究中. 这种曲面通常用一或两组平行平面截得的截面线表示.

9.3.1.1. 一组平行的截平面的曲面消隐

这里取一组平行的截平面

$$z_i = z_0 + ih,\ i = 0, 1, \cdots, n;\ h > 0, \tag{9.3.2}$$

与曲面相切割，得到一组截面曲线，用这组曲线来表示曲面，然后将这组曲线投影至屏幕(我们假定对应着平面$z = 0$,观察者在z轴正向充分远处)，计算其遮挡关系后，画出可见曲线部分，即可得到曲面的消隐结果。

下面详细叙述这一过程。

一组平行的截平面

$$z_i = z_0 + ih, i = 0, 1, \cdots, n; h > 0 \tag{9.3.3}$$

中, $z = z_n$是要显示出的曲面的最大z坐标值处, 最靠近观察者. 由$z = z_n$可得曲线

$$y = f(x, z_n) \tag{9.3.4}$$

从这条曲线开始，水平方向每个像素点的对应x坐标值x_j有

$$y_{jn} = f(x_j, z_n) \tag{9.3.5}$$

对其余截平面$y_i, i = 0, 1, \cdots, n-1$，也有

$$\begin{cases} y = f(x, z_j) \\ z_{ji} = f(x_j, y_i) \end{cases} \tag{9.3.6}$$

这样就得到一系列曲线，从离观察点最近的曲线开始，对于屏幕上当前曲线段上每一个x值(即每个像素点)，若其对应的y 值都大于屏幕上先前曲线上同一x对应的y值，则这段曲线可见，从而画出它。即

(1). 如果$y_{ji} > y_{max}(j)$，则点(x_j, y_{ji}, z_j)可见，画出，且将y_{ji}的值重新赋给$y_{max}(j)$，使得$y_{max}(j)$为已经处理过的第$i+1, \cdots, n$条曲线上对应x_i的y值中最大者。即

$$y_{max}(j) = max\{f(x_j, z_k) :\ i + 1 \le k \le n\}; \tag{9.3.7}$$

(2). 如果$y_{ji} \le y_{max}(j)$，则点(x_j, y_{ji}, z_j)不可见，不画出。

根据上述处理方法, 随着i减小,相应的z坐标减小,相应的截面线离观察者. 愈来愈远. 截面线上的点要想可见, 就必须愈来愈高. 这有点像远处的山, 只有高处才能被看到. 上面的处理方法也只是对山形的函数曲面才是正确的. 如果是可以悬在半空中物体的曲面, 其远处的, 向下露出的部分也可以是可见的, 比如口朝上的帽子的表面所代表的曲面就是如此. 就两条截面线而言, 如图9.2所示.

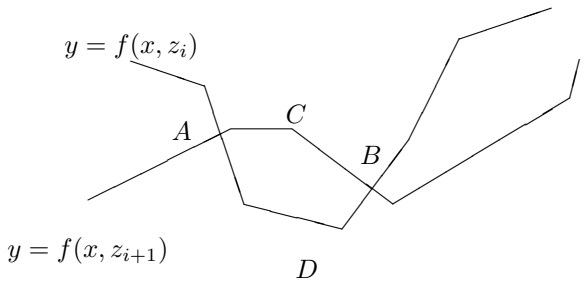


图9.2 两条截面线的前后遮挡关系

图9.2中的D点附近部分应是可见的，但按上面规则，对应于y_{i+1}时的点C附近部分的y坐标比较大,导致对应于y_{i+1}时的D点附近部分不可见。所以要修改上述方法,不但要考虑$y_{max}(j) = max\{f(x_j, z_k) : \ i+1 \le k \le n\}$；而且还应考虑$y_{min}j = min\{f(x_j, z_k) : \ i+1 \le k \le n\}$,不仅使愈远愈高的部分可见, 也要使愈远愈低的部分可见. 于是,修改上述规则为：

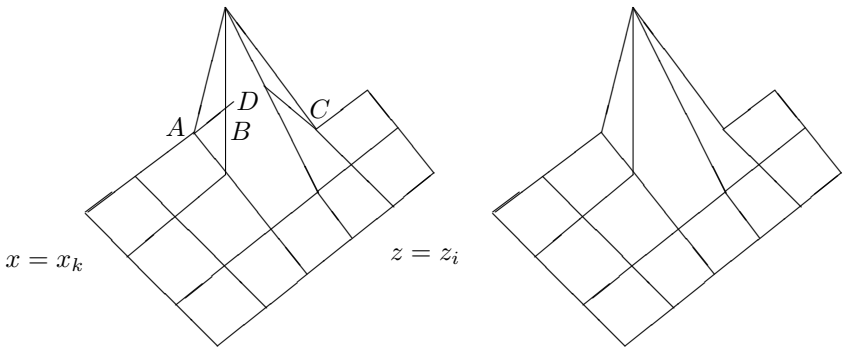
(1). 如果$y_{ji} > y_{max}(j)$或$y_{ji} < y_{min}(j)$，则点(x_j, y_{ji}, z_j)可见，画出，并且在$y_{ji} > y_{max}(j)$时将y_{ji}的值赋给$y_{max}(j)$，在$y_{ji} < y_{min}(j)$时将y_{ji}的值赋给$y_{min}(j)$。即

$$\begin{cases} y_{max}(j) = max\{f(x_j, z_k) : i+1 \le k \le n\}, \\ y_{min}(j) = min\{f(x_j, z_k) : i+1 \le k \le n\}; \end{cases} \tag{9.3.8}$$

(2). 如果$y_{min}j \le y_{ji} \le y_{max}j$，则点$(x_j, y_{ji}, z_j)$不可见，不画出。

9.3.1.2. 两组平行的截平面的曲面消隐

如果同时有$x = x_j$和$z = z_i$两组截面曲线来表示相应的曲面,分别对两组截面曲线按上述方法消隐后把各自的可见曲线简单叠加在一起,有可能使一些隐藏线不能被消除,得不到正确的消隐处理曲面,如下图所示的线段AB和CD.



(a). 不正确消隐处理 (b). 正确消隐处理

图9.3 两组曲线表示曲面的消隐处理

正确的做法是对两组曲线同时进行处理. 即作完截平面$x = x_k$上$z = z_{i-1}$和$z = z_i$之间的一段曲线的消隐处理后,接着作截平面$z = z_i$上$x = x_{x-1}$ 和$x = x_k$之间的一段曲线的消隐处理.

9.3.2 参数曲面消隐

参数曲面一般具有如下表示式

$$x = x(u, v), \quad y = y(u, v), \quad z = z(u, v), \tag{9.3.9}$$

这种形式表示的曲面的消隐处理比较复杂. 如果采用类似于前一小节的显式函数曲面的处理方法, 设计到大量求解形如

$$x(u, v) = x_k, \quad z(u, v) = z_i, \quad y = y(u, v) \tag{9.3.10}$$

方程,计算过程十分复杂.

采用参数网格$u = u_i$, $v = v_k$给出的等值参数曲线表示曲面, 避免了解方程(9.3.10)的复杂计算。但这时也不能采用前一小节的方法进行消隐处理, 因为此时的网格曲线不像前一小节采用的截面线曲线那样, 具有空间位置上的前后顺序关系.

我们需要采用不同的方法处理由参数网格线表示的曲面。可以设想当网格充分密时，每个参数网格对应的小曲面片近似为平面片，对平面片进行编号。将自由曲面投影到屏幕上，然后将屏幕平面均匀分割为正交的光栅网格线。对每个网格区构造一个覆盖了这个网格区的曲面片表，然后进行最大最小测试以决定一个具体的曲面片潜在地覆盖了哪些网格区，再进行深度测试，画出最靠近观察者的曲面片。具体过程，本节不详细叙述，有兴趣者可查找有关参考书。

9.4 z 缓冲器算法

z缓冲器算法是最简单的消除隐藏面算法之一.

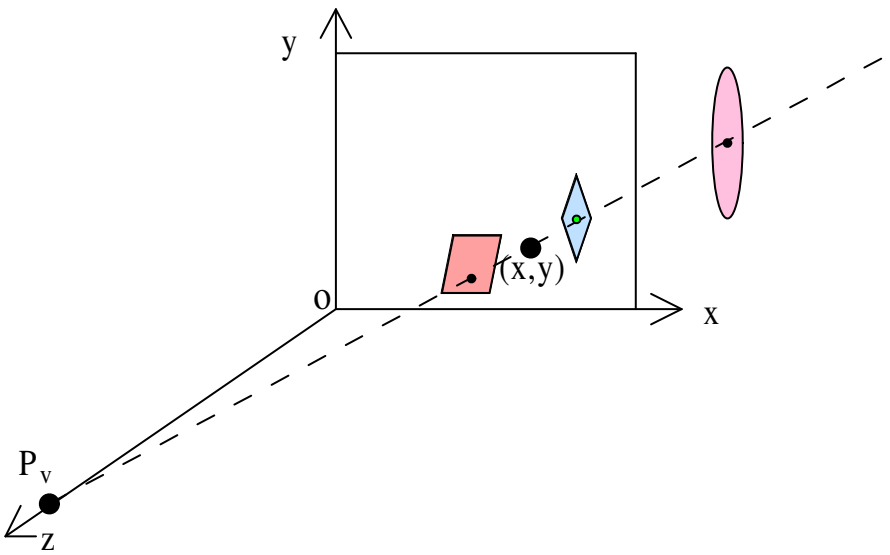


图9.4 z 缓冲器算法原理示意图

缓冲器可理解为存放一些动态变化的对象的数据存储空间,即一组存贮单元. z缓冲器是具有如下特点的一组存贮单元，其单元个数和屏幕上像素的个数相同，也和帧缓冲器的单元个数相同，而且它们之间是一一对应的。z缓冲器中每个单元的位数取决于图形在z方向的变化范围，一般取$20bits$就可以了。

z缓冲器中每单元的值是对应像素点所反相应的对象上点的z坐标值。z缓冲器中每个单元的初值取成z的极小值，帧缓冲器每个单元的初值可放对应像素点背景颜色的值。图形消隐和生成的过程就是给帧缓冲器和z缓冲器中相应单元填值的过程，在把显示对象的每个面上每一点的属性（颜色或灰度）值填入帧缓冲器的对应单元前，要把这点的z坐标值和z缓冲器中相应单元内的值作比较。

只有前者大于后者时才改变帧缓冲器的那一个单元的值，同时z缓冲器中相应单元的值也要改成这点的z坐标值。如果这点的z坐标值小于z缓冲器中相应单元的值，则说明对应像素已显示了对象上一个点的属性，该点比要考虑的点更接近观察者。这样，无论帧缓冲器或z缓冲器相应单元的值均不应改变。对显示对象的每一个面上的每一个点都做了上述处理后，便可得得到消除了隐藏部分的图形。

上述算法的优点是简单、可靠，不需要对显示对象的面预先进行排序。缺点之一是需要很大的z缓冲器。值得说明的是如果显示对象的每个面上每一点的属性（颜色或灰度）值不变(如仅仅显示三维物体的轮廓, 每一点颜色或灰度不变），则不需要建立这个z缓冲器了。无论如何，显示对象的表面和像素对应的每一个点处都要计算它的z值，因而显示对象的表面构成复杂时计算量较大。

为了克服第一个缺点，可把整个平面分成若干个区域，一个区域一个区域来显示，这样z缓冲器的单元数只要等于屏幕上一个区域的像素个数就可以了。

如果把这个区域取成屏幕上一行，就得到了行扫描线z缓冲器算法。这时z缓冲器的单元数就可以取成和一行上的像素数目相同。从最上面的一条扫描线开始工作，向下对每一条扫描线作处理。

对每一条扫描线来说，把相应的帧缓冲器单元置成底色，在z缓冲器中存放z的极小值。对每个多边形检查它在Oxy平面上的投影和当前的扫描线是否相交，若不相交，则不考虑该多边形。如果相交，则扫描线和多边形边界的交点一定是成对地出现. 对每对交点中间的像素点计算多边形所在平面对应点的深度(即z值),并和z缓冲器中相应单元存放的深度值作比较。若前者大于后者，则z缓冲器的相应单元内容要被求得的平面深度代替，帧缓冲器相应单元的内容也要换成该平面的属性。

对所有的多边形都作上述处理后，帧缓冲器中这一行的值便反应了消隐后的图形。对帧缓冲器每一行的单元都填上相应内容后也就得到了整个消隐后的图形。

为了克服第二个缺点，需要分析大量的计算中间的相互关系，尽可能减少不必要的重复运算。从上述算法可知，在处理每一条扫描线时，要检查各多边形是否和该线相交，还要计算多边形所在平面上很多点的z值，这都要花费很多时间去计算。为了使这些工作能高效率地进行，就需要分析不同的扫描线上的交点之间的关系,找出其间的规律. 相关内容可以参阅计算机图形方面的书籍.

习题9

1. 给定四个点$A(1,\ 2,\ 0)$, $B(3,\ 6,\ 20)$, $C(4,\ 8\ 20)$和$D(0,\ 0,\ -10)$，以及观察点$P_v(2,\ 4,\ 6)$. 判定这几个点的相互遮挡关系。

2. 假设观察点$P_v(10,\ 3,\ 20)$.判定一凸多面体上由三点$A(1,\ 2,\ 0)$, $B(3,\ 6,\ 20)$, $C(4,\ 8\ 20)$决定的面的可见性。

3. 假设观察点$P_v(10,\ 3,\ 20)$.判定一凸多面体上由三点$A(3,\ 6,\ 20)$, $B(1,\ 2,\ 0)$, $C(4,\ 8\ 20)$决定的面的可见性。

4. 假设观察点$P_v(0,\ 0,\ -100)$. 通过计算判定四点确定的四面体各个侧面的前后遮挡关系: $A(4,\ 3,\ 1), B(3,\ 2,\ 1), C(0,\ 0,\ 1), D(3,\ 3,\ 2)$.

5. 假设观察点$P_v(a,\ b,\ c)$.编程序实现用两组平行的截平面线的方法画出半球面$y = \sqrt{1 - x^2 - z^2}$.改变观察点P_v的坐标，比较半球面的显示效果。

6. 假设观察点Z轴负无穷远处.编程序实现用两组平行的截平面线的方法画出函数$y = 3\dfrac{\cos(1.2\sqrt{x^2 + z^2})}{1 + \sqrt{x^2 + z^2}}$所表示的曲面。

7. 简述Z缓冲器消隐算法。

Chapter 10

计算机图形中真实感图形设计

10.1 引言

真实感图形, 顾名思义, 就是看起来具有真实感的图形. 而前面所讨论的图形主要是讨论其几何外形的. 具有真实感的图形就不仅包括图形对象的几何外形, 还应包括图形对象的表面色彩方面的视觉效果(颜色和明暗色调)的真实感.

真实感图形设计是计算机图形学的一个重要组成部分. 真实感图形设计需要数学、物理学、计算机科学和其它科学知识在计算机图形设备上生成象彩色照片那样的真实感图形。真实感图形设计技术在各个领域中有非常广泛的应用.

利用计算机设计真实感图形有很大的实用价值。例如在产品外形设计中，常需制作实物模型来检查设计效果。特别是那些对外形美感要求较高的产品，往往需要根据模型反映的问题不断修改设计方案，以获得最佳造型效果，这就需要反复制作模型，耗费大量人力物力。而采用计算机真实感图形设计技术，就可方便地在屏幕上显示产品各种角度的真实感视图，并在屏幕上直接对外形进行交互式的修改，应用设计技术可以代替实物模型的制作.

同样，建筑师在建筑设计时,也可不必制作精致的模型，只需将他的设计通过各种真实感视图表现出来。这种技术大大节约了人力和物力，并使设计周期得以缩短，质量得到提高。

在动画以及影视广告制作中, 需要模拟真实的场景和画面,甚至包括许多实际上有很大危险或不可实现的场景和画面,通过具有真实感的计算机图形就可以很方便地得到. 除此之外,真实感图形设计技术在飞行训练、战斗模拟、分子结构研究、医学等领域都有广阔的应用前景.

真实感图形的设计必须要考虑物体表面的颜色和明暗色调,以便用来表现物体的几何形状、空间位置以及表面构成的材料，因此我们需要光学物理的有关知识, 用数字的方式表示光的大小, 强弱, 及其色彩组成以便借助于计算机进行处理。

产生真实感图像涉及到有关光的物理学和心理学两个方面。光在和周围环境相互作用后到达人的眼睛，并在人的眼睛里进一步发生物理的和化学的变化，最终生成人脑能够感知的电脉冲，人脑因此而获得对环境的了解。关于视觉心理学方面的问题已有广泛的研究，有关内容已超出本书的范围。这里在介绍光线在特定环境中的传播方式的基础上，重点探讨在给定物理环境下光的数字化描述方法,由此给出生成具有真实感的图像的数字化方法。

10.2 光与颜色的基本知识

现实世界中的物体有色彩是因为有了光的存在,物体把光反射到我们的眼睛中然后我们才感觉到了物体的色彩. 最简单的例子就是黑暗中我们什么也看不见, 因此所有的物体也失去了其色彩. 所以要弄清物体的色彩,首先要弄清光的色彩.

作为客观存在的一种物理的量,光是一种电磁波，一束单色光具有一定的频率(因而具有一定的颜色)和波长(因而具有一定辐射功率)。

人眼的视觉特性主要表现在：对不同波长的光具有不同的敏感度,对于具有相同辐射功率的不同单色光，人眼也会觉得它们并不同样明亮。就如我们所感觉到的一样,光具有颜色和明亮程度两种特性. 下面分别介绍光的这两个特性.

10.2.1 光的明亮度

实验表明，人眼对波长为555纳米的绿光在亮度上最敏感，而对波长较长的红光和波长较短的紫光则不敏感。人眼对于波长大于780纳米和波长小于380 纳米的光就失去了亮度感觉。这些波长的光就是我们通常熟知的红外光和紫外光.

光的明亮程度是人的眼脑视觉系统通过眼睛对光的一种感受,因此同一束光的明亮程度是可以因人而异的. 但光的辐射功率却不是因人而异的,是可以客观测量的. 因而我们可以通过光的辐射功率来反映光的明亮程度. 试验表明,光的辐射功率越大,相应的光越不容易为人的眼睛所感觉得到;光的辐射功率越小,相应的光就越容易为人的眼睛所感觉得到.

对于波长为λ的光,用$p(\lambda)$表示其辐射功率,由于人眼对光的亮度的感觉与光的辐射功率成反比,因此我们可用其倒数来度量光的亮度,用一个标准的术语,称为光的视敏度,用$k(\lambda)$来表示. 于是有

$$k(\lambda) = \frac{1}{p(\lambda)} \tag{10.2.1}$$

由于视敏度是一个相对的概念,可以设定最大的敏感度为1. 我们已知对$\lambda = 555$纳米的光具有最大敏感度,因此相对视敏度就是视敏度$k(\lambda)$与$k(555)$之比. 记作$\nu(\lambda)$:

$$\nu(\lambda) = \frac{k(\lambda)}{k(555)} = \frac{p(555)}{p(\lambda)} \tag{10.2.2}$$

光也是一种能量,向四周空间传播. 在原点的光能量传播到半径为r的地方后,一般可以认为这部分光能量就均匀的分布在半径为r的球面上了,而球面上每一点的能量就只能是原来的能量除以球面的面积$4\pi r^2$. 因此,光的亮度传播到远处后,其大小于与距离的平方成反比.

10.2.2 光的颜色

由人眼来处理所觉察到的彩色是一种生理和心理现象，对此至今还没有完全搞清楚，但借助于实验的和理论的方法，我们还是能够确切地表示彩色的物理性质，这对于我们在计算机上生成理想的图像是非常有用的。

10.2.2.1. 彩色视觉及三基色原理

就人眼的视觉感受而言，各种颜色单从其频率的不同来区分是不完全的，还应该同时使用色调(hue)，饱和度(saturation)和亮度(brightness)三个量。

色调由彩色光的光谱成分决定，反映彩色光在质方面的特征。

饱和度由彩色中混入白光的数量决定。彩色光中纯光谱色的含量越高，其饱和度也越高。例如高饱和度的深红色光，在搀入白光后被冲淡而变成低饱和度的淡红色。饱和度反映某纯光谱色的纯度。

亮度由彩色光的强度决定，是彩色光引起视觉刺激的程度，反映彩色光在量方面的特征。

通常所说的色度是色调和饱和度的合称，它说明了彩色光的颜色类型和颜色的深浅程度。

对于人眼的彩色视觉机理提出了这样一种假设：人眼视网膜上的锥状细胞有三种类型：红敏细胞，绿敏细胞和蓝敏细胞，顾名思义，它们对不同色光的亮度感觉有强烈的选择性。三种光敏细胞对光的综合感应形成了我们主观上对光的颜色和亮度的感应.三种光敏细胞对光的感觉一致,主观上光的彩色感觉(色度和亮度)就相同. 由此可以得到下述的三基色原理:

> 选择三种基色（例如红、绿、蓝），把它们按不同的比例混合，就可以得到自然界中几乎所有的彩色。

10.2.2.2. 物理三基色和RGB色系数

选择三基色的原则是：获得方法简单，色度稳定且准确，能配出尽可能多的颜色。为此，CIE(国际照明委员会)规定了RGB三种光为基色，并规定了基色单位当量。其中,R为红光,波长为700.0纳米,基色单位当量为1; G为绿光,波长为546.1纳米,基色单位当量为4.5907; B为蓝光,波长为435.8纳米,基色单位当量为0.0601. 当量是指用指定三基色配出标准白光时，RGB三基色的光的比例，这里为$1 : 4.5907 : 0.0601$.

这样，对于任意给定颜色(光)F，可以将它表示为：

$$F = [R, G, B] \tag{10.2.3}$$

称此方程为颜色(光)F的配色方程，R, G, B称为三色系数.三基色R, G, B的系数比例决定所配彩色光的颜色，它们的数值决定所配彩色光的亮度。R, G, B也称为颜色F的三基色分量或三基色坐标。

10.2.2.3. XYZ三色系数

物理三基色具有清楚的物理意义，但在使用上相应的三色系数R, G, B 存在一些计算上的不便之处，例如：

(1). 不能直接反映某一彩色光的光亮度，而要通过公式计算出来；

(2). 配置最常用的白颜色时系数复杂难记;

(3). 在配制高饱和的蓝色光或绿色光时，会出现负数。

为了克服这些缺点，CIE另外规定了计算三基色，其基色单位为X, Y, Z。它们虽然不代表真实的颜色，但在计算上可以克服上述物理三基色的缺点。计算三基色与物理三基色之间可以通过一个矩阵A联系起来：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = A \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{10.2.4}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = A^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{10.2.5}$$

其中，

$$A = \begin{bmatrix} 0.4185 & -0.0912 & 0.0009 \\ -0.1587 & 0.2524 & -0.0025 \\ -0.0828 & 0.0157 & 0.1786 \end{bmatrix} \tag{10.2.6}$$

$$A^{-1} = \begin{bmatrix} 2.7689 & 1.0000 & 0.0000 \\ 1.7518 & 4.5907 & 0.0565 \\ 1.1302 & 0.0601 & 5.5943 \end{bmatrix} \tag{10.2.7}$$

这两个式子反映了RGB色系数与XYZ三色系数之间的联系。

如此定义的XYZ三基色系数避免了RGB三基色系数的上述不便之处,具体结论如下.

(1). 某一分量的大小直接反映了相应彩色光的光亮度;

(2). 配置最常用的白颜色时三个坐标分量系数相等即可; 分量的大小反映了白颜色的强(亮)度,取最大值时为最亮的白颜色,取最小值时为最暗的白颜色. 最小值为0时就是黑颜色;

(3). 在配制需要的颜色时,各个坐标分量总是非负的。

在计算机的图形处理中就采用XYZ三基色系数,并且在实际应用中实数坐标系统取各分量的最大值为1,最小值为0.

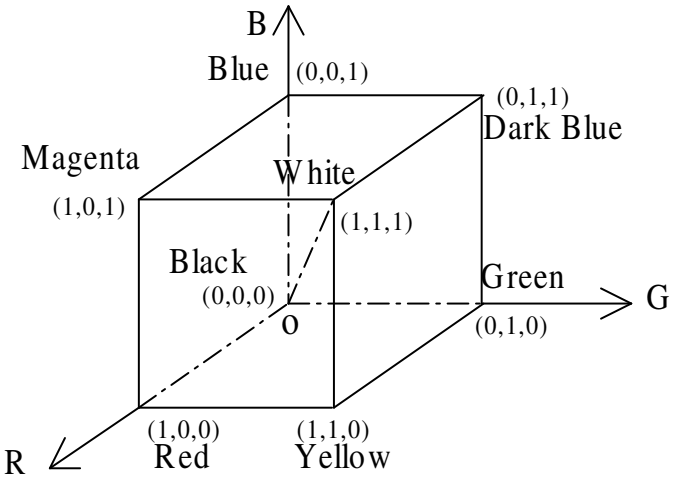


图10.1 RGB颜色模型示意图

如果是采用整数坐标表示, 如24位彩色显示,则各分量用8位表示, 最大值为$2^8 - 1 = 255$.

10.2.2.4. 三基色混合起来的方法:

如下各种方法都可实现对三基色进行混合,生成所需要的颜色.

(1). 把三种光同时投射到一个全反射的表面上合成。

(2). （时间混合法）把三种光轮流投射到一个全反射的表面上，只要每种颜色的光在相应的表面上停留的时间小于人眼的视觉暂留时间，人眼的彩色感觉就跟同时投射的情况相同;

(3). （空间混合法）同时把三种光投射到同一表面的三个相邻点上，只要这三个点相距足够近，超出了人眼的视觉分辨率，人眼就可以得到三种光混合的彩色感觉。

(4). （生理混合法）两只眼睛同时分别看用两种颜色表示的相同图像，可以产生混色效果。

10.3 光的传播规律

10.3.1 光的来源

光由光的强度I, 方向D和颜色(R, G, B的比例)决定，在光照模型中不必特指是哪一种颜色，因为彩色光是由三种原色光按比例配出来的.

要考察物体表面上一点的光亮及其色彩,就首先要弄清楚这一点的光来自那里. 如果没有任何来源的光,这一点就是黑暗的.

10.3.1.1. 光源

光源这里狭义地指可以自身发光的物体. 有了光源,才能照亮一片区域,照亮一片空间,相应空间的物体也就被照亮了,显现出其色彩.

光源发出的光可以有色彩, 有强弱,这些是其光学特性. 发光的物体本身有一定的几何形状,占据一定的几何区域或空间区域如通常的长条或环形日光灯管,称为分布式光源. 但也有可能发光的物体占据的几何区域或空间区域不存在或可以忽略不计如太阳, 我们就可以称其为点光源. 基于点光源的光的处理是比较简单的,基于分布式光源的光的处理相对比较复杂,通常可视为有无数个点光源组成而通过对点光源累加积分来实现.

光是一种电磁能量，从光源发出后就以自然界的极限速度在光媒介质中传播。当光线照射到一个物体表面时，光能量转化为三种形式：被相应物体表面吸收而变成热能，被在相应物体表面透射而进入另一媒介质中传播,被在相应物体表面反射而回到原来的媒介质中继续传播。

光的这这三部分能量的比例与光的波长, 相应物体表面的光学性质和光线射向相应物体表面的角度有关。我们对光的热能不感兴趣，光的热能就是光传播过程中的损耗部分。

10.3.1.2. 反射光

有些物体表面本身不发光,但如前所述,有光线照射到该物体表面时,会被反射而重新进入光媒介质中传播并照亮一定的空间区域,进而照亮物体. 这时的光就是反射光,照射到该物体表面的光相应的称为入射光. 例如月亮本身不发光,但反射太阳光. 因此月光就是典型的反射光.

如果物体的表面是理想的镜面,光的反射满足严格的几何法则：入射角等于反射角，而且入射光线与反射光线共面。

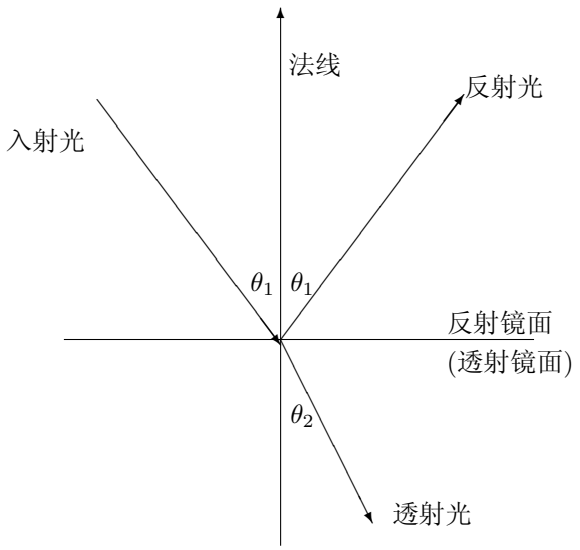


图10.2 光的镜面反射与透射的几何规则

10.3.1.3. 透射光

有些物体表面本身不发光,但如前所述,有光线照射到该物体表面时,光线会穿透该物体继续前进, 这时的光就是透射光. 只有透明或半透明的物体才需要考虑透射光。有透射光产生时,光总是要在两种不同的媒介之中传播,因此传播速度不同,传播方向也不同. 这也就是说入射光和透射光的方向不同.

反射光和透射光是可以同时存在的. 日常见到的情况如太阳镜就是有适量的太阳光透过镜片进入眼内，同时又反射掉大部分的光。

10.3.1.4. 漫反射和漫透射

物体表面的光滑性将影响光的反射和透射. 如果物体的表面是理想的镜面,光在镜面上的反射满足严格的几何法则。物体的表面大多数情况下不是理想的镜面,而只是粗糙的表面. 可以把粗糙的表面看成是由无数(肉眼能够分辨或无法分辨的) 微镜面组成的。这些微镜面的法线方向是随机分布的,表现在物体表面上某一点的即让是非常非常小的范围内的光的反射方向也是随机分布的,因此产生了漫反射光。

如果这些微镜面的法线方向在物体表面外侧以均匀的概率随机分布（这种表面称为完全漫反射表面），那么在看得见表面的任何地方观察它时会有相同的亮度。在理想的镜面表面上，所有的微镜面的法线与表面的法线平行，因此只有在反射光线方向才能看到表面上是亮的，在其它方向观察物体表面时，看不到相应的光因而感觉物体表面是暗的。

由于同样的原因,也可能产生漫透射光. 同一物体表面对不同光线的上述光学性质往往是不一样的：当白色光照射到对红色光有较大的反射率的表面时，表面就呈红色。当白色光照射到对红色光有较大的透射率的表面时，相当于对光进行过滤,透射产生出红色的光。

10.3.1.5. 环境光

在一定的空间区域中,并没有明显的光源,但是周围的物体仍是清晰可见的. 这可能是远离指定的空间区域的光源的光经过无数多感觉到或未感觉到的各种各样的物体表面如房间里的墙壁,天花板,地板以及各式各样的家具的表面,户外环境中的山,水以及植物的枝叶的表面等反反复复地反射后形成的. 这样的光的产生可以是非常复杂的,但结果并不复杂,只是在一定的空间区域中表现为亮度均匀的某种光. 这样的光就是环境光.

广义地讲,能发光的点就是一个点光源,无论发出的光是这一点本身产生的,或是在别处产生而传播到该点后,再经由该点传播出去的.

10.3.2 光的传播的计算模型

光照模型适用于每一种原色光以及白光。这也就是说,在光的传播过程中实际上可以把一束光看作由三原色的R, G和B三束光在同时传播. 光照模型适合于对三束光分别进行传播计算,再叠加得到最后的光.

光有强弱之分,并且在均匀的光传播媒介质中向四周空间均匀传播. 因此传播得越远,光的强度就越小. 于是,离光源越远的物体就显得越暗. 更精确地说,光源发出的光传播到远处后,其强度与距离的平方成反比. 但是实际计算的结果表明这一计算公式导致光的强度减弱得太快,给人的感觉是不同的物体的明暗差别太大. 为避免这一缺点,通常采用强度与距离成反比的公式. 如果点光源的强度为I,则离点光源距离为d的点的光的强度为

$$\frac{I}{d+d_0} \tag{10.3.1}$$

其中$d_0>0$为常数,用以防止当d很小时分母趋于0,其大小可以根据实际情况进行调整直到获得比较满意的图形的明暗效果为止. 这一公式适合于各种点光源发出的光的传播. 根据经验，如果使光强按距离(而不是距离的平方)的线性关系来衰减，得出的结果比较符合实际。

10.3.3 各类光传播的计算

10.3.3.1. 环境光的计算

环境光在空间中是均匀分布的,处于稳定状态. 假定任何物体表面都会受到强度为I_a的环境光的照射。而我们看到的环境光是通过环境的反射而觉察到的,因此看到的环境光就不是全部的环境光,而是

$$k_aI_a \tag{10.3.2}$$

其中k_a是环境光的反射系数,通过调整k_a可以改变空间区域的背景的明暗程度,也正因为如此,环境光也被称为背景光。

10.3.3.2. 反射光的计算

物体表面的光学性质包括物体表面的光滑性, 对光的反射率和透射率。通常把表面的光滑性分为完全漫反射表面和镜面两种形式。

光在镜面上的反射满足严格的几何法则：入射角等于反射角，而且入射光线与反射光线共面。由此得到的光只能在特定的方向被观察到,这与实际情况不符. 实际上,在靠近反射光方向周围的方向上仍然可以观察到反射的光,形成了反射光方向周围的一个高光区.

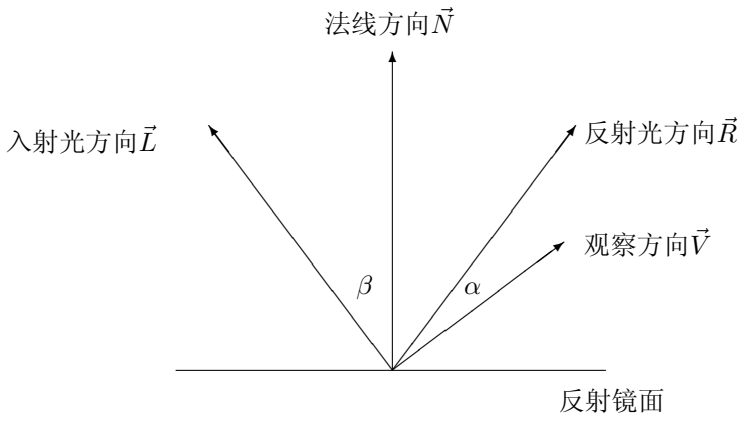


图10.3 反射光的计算

比较光滑的表面,高光区比较狭窄;比较粗糙的表面,高光区比较分散. 反射光不仅存在于特定的反射方向,也存在于邻近的方向. 因此观察方向与反射光方向的夹角是一个重要的参数. 随着夹角的增加,指定的观察方向的光的强度将迅速减弱, 直到$\frac{\pi}{2}$时为0. 于是有光的一个简单的镜面反射模型:

$$\frac{k_s \cos^n \alpha}{d + d_0} \tag{10.3.3}$$

其中k_s为表面的反射系数($0 \le k_s \le 1$)，不仅与构成物体表面的材料的光学性质有关,而且与入射角(入射光和反射表面法线的夹角)有关. 设$\overrightarrow{R}$和$\overrightarrow{V}$分别表示反射光和观察方向的单位方向矢量, α是反射光线和观察方向的夹角,则上式也可表示为

$$\frac{k_s(\vec{R} \cdot \vec{V})^n}{d + d_0} \tag{10.3.4}$$

10.3.3.3. 漫反射光的计算

不同于反射光的是漫反射光均匀地向四周发散. 对于不是镜面的物体表面,朗伯(Lambert)余弦定律总结了一入射光照射到一个完全漫反射的物体表面上时光的反射规律。如果光线的入射强度为I_1，则漫反射光的光强I为：

$$\begin{cases} I = I_1 k_d \cos^n \beta = I_1 k_d (\vec{L} \cdot \vec{N})^n \\ 0 \le \beta = < \overrightarrow{L} \cdot \overrightarrow{N} > \le \dfrac{\pi}{2} \end{cases} \tag{10.3.5}$$

其中k_d为表面的漫反射系数($0 \le k_d \le 1$) ，$\overrightarrow{L}$和$\overrightarrow{N}$ 分别为入射光和反射表面法线的单位方向矢量, β是入射光线和反射表面法线方向的夹角。幂次n用来控制反射光在空间的分布，n越大，漫反射光就越集中在镜面反射光线周围。通常n在1到2000之间取值.

10.3.3.4. 透射光的计算

当光线传播到不同的媒介质的界面上时，不仅会发生反射，往往还会发生透射。我们知道反射光与入射光在同一种媒质中传播，透射光与入射光在不同的媒介质中传播,在进入新的媒介质中时光的方向会发生变化,这一现象称为光的折射。

我们首先了解一下有关折射的光学定律。参考图10.2, 假设入射光I投射到一个既光滑又透明的表面上，这时会产生反射光R和透射光T。关于入射光与反射光的光强关系和方向关系在上一节已经介绍，现在来看一下入射光与透射光的强度关系和方向的关系。透射光的强度T与入射光的强度的关系为：

$$T = k_t I \tag{10.3.6}$$

其中k_t 是个与表面两侧材料的光学性质有关的实验常数，称为透射率，$0 < k_t < 1$. 对于给定的两种材料，光线在透明的界面上投射时，入射光与透射光相对于法线的角度(分别用θ_1和θ_2表示)满足关系：

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2 \tag{10.3.7}$$

其中η_1和η_2各自由表面两侧的的不同材料决定的实验常数。应注意到通常情况下透光材料是放置在空气中的。因此空气大多数情况下是其中一种材料。

为使用方便起见定义

$$k_n = \frac{\eta_2}{\eta_1}, \tag{10.3.8}$$

称为相对透射系数，是由表面两侧的材料共同决定的实验常数。当已知单位法向量$\overrightarrow{N}$和单位入射光线矢量$\overrightarrow{L}$时，透射光的方向矢量$\overrightarrow{T}$可以用下述公式求出：

$$\overrightarrow{T} = k_n(\overrightarrow{L} - (\vec{L}\cdot\overrightarrow{N}) + \sqrt{(\overrightarrow{L}\cdot\overrightarrow{N})^2 - 1 + k_n^2})\ \overrightarrow{N} \tag{10.3.9}$$

当式中的根号部分取虚数时，实际透射光方向$\overrightarrow{T}$不存在，这表示没有透射光，也就是发生全反射。

10.4 一个简单的光照模型

有了上述准备,我们现在来考察物体表面的光的亮度,或者说强度. 显然指定环境中的任何物体表面都会受到环境光的照射. 如果有发光的光源,物体表面也会受到光源的照射,并把光反射到观察者的眼中,因此被观察到. 假设环境光的强度为I_a,点光源的强度为I_1。则有光照模型：

$$I = I_a k_a + I_1 \frac{k_d(\overrightarrow{L}\cdot\overrightarrow{N})^n + k_s(\overrightarrow{V}\cdot\overrightarrow{R})^m}{(d_0 + d)} \tag{10.4.1}$$

在计算机图形学中，上述模型称为明暗函数，用于决定物体上的一点或屏幕上一像素点处的光强或明暗色调。如果要生成彩色图像，只要对三原色的每一分量使用上述明暗函数进行计算即可。通常，反射光具有与入射光相同的颜色组成，因此计算三原色时可以用相同的k_s 值。

如果有m个光点，可以将它们累加求和，因此多光源的明暗函数为：

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_{1j}}{d + d_0}(k_d(\overrightarrow{L_j}\cdot\overrightarrow{N})^n + k_s(\overrightarrow{V_j}\cdot\overrightarrow{R_j})^m) \tag{10.4.2}$$

上述模型中的镜面反射方向的单位向量$\overrightarrow{R}$是根据入射光线向量$\overrightarrow{I}$、法线向量$\overrightarrow{N}$决定的. 下面根据光学上的反射定律，即入射光线、法线及反射光线在同一平面内而且入射角等于反射角，求出当已知入射光线向量$\vec{L}$、法线向量$\vec{N}$时，计算反射光线方向向量$\vec{R}$的方法。

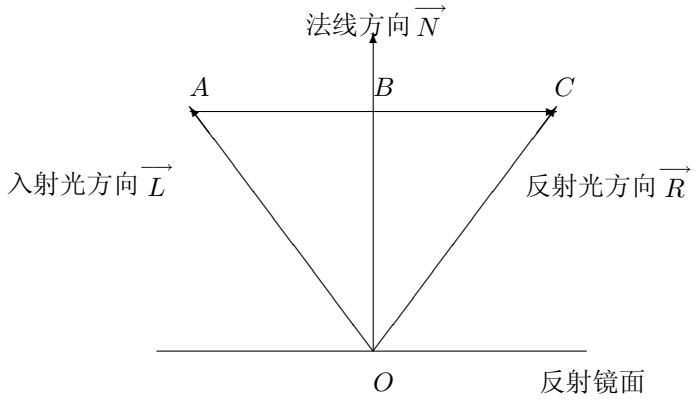


图10.4 镜面反射光线方向向量的计算

假设$\overrightarrow{L}$, $\overrightarrow{N}$, $\overrightarrow{R}$都是单位向量。如图10.4所示, 向量$\overrightarrow{OB}$为向量$\overrightarrow{L}$在向量$\overrightarrow{N}$上的投影向量,则有

$$\overrightarrow{OB}=(\overrightarrow{L}\cdot\overrightarrow{N})\ \overrightarrow{N} \tag{10.4.3}$$

$$\overrightarrow{AC}=2\ \overrightarrow{AB}=2(\overrightarrow{OB}-\overrightarrow{OA})=2((\overrightarrow{L}\cdot\overrightarrow{N})\ \overrightarrow{N}-\overrightarrow{L}) \tag{10.4.4}$$

因此可得反射光线方向向量

$$\overrightarrow{R}=\overrightarrow{OC}=\overrightarrow{OA}+\overrightarrow{AC}=2(\overrightarrow{L}\cdot\overrightarrow{N})\ \overrightarrow{N}-\overrightarrow{L} \tag{10.4.5}$$

10.5 明暗处理

根据上述光照模型，计算物体表面上各点处的光强的关键是求出物体表面在这些点处的法线方向$\vec{N}$。如果物体表面上各点的法线很容易求出，就可以很快得到真实感图像了，如由平面多边形围成的多面体的表面,每个平面多边形上的点共有一个法向量.

但实际物体的表面并不总是那么简单。对于表面包含有曲面的物体，如果曲面方程不是以显式给定的，那么计算每一点的法向量，再按光照模型计算每一点的亮度，计算量是非常大的。于是人们又想到了用平面多面体来逼近给定曲面的方法以使计算简化。

如果用来逼近曲面的平面多面体被划分得很细，而且曲面上各点的曲率半径都比较大时曲面弯曲程度小,即看起来比较平坦，用上述方法可以得到比较理想的真实感图像。

但如果曲面上存在曲率半径较小即曲面弯曲程度大的区域，那么处于这些区域内用来逼近曲面的相邻小平面的法向量(的方向)会变化很快，从而使得到的这些逼近小平面的光亮程度有很大的光强差，由于光学里面的马赫带效应，导致这些部分看起来很不光滑，降低了图像的真实感。

为避免上述问题，对于表面为曲面的部分通常采用以下两种补救方法(对于本来就是平面多边形的表面，当然就不必作此处理了)。

10.5.1 Gouraud的光强度插值法

假设物体表面的曲面部分已经用平面多面体做了很好的逼近，平面多面体上各个顶点的位置以及每个小平面的法向量都已确定。

对$k=0,1,\cdots,n$,先计算顶点P_k处的平均法向量N_k：

$$\overrightarrow{N_k}=\frac{1}{m}\sum_{j=1}^{m}\overrightarrow{N_j} \tag{10.5.1}$$

其中，$\overrightarrow{N_j}, j=1,2,\cdots,m$是所有以$P_k$为公共顶点的$m$个小平面的单位外法向量。

基于这一点求出所有法向量即可利用光照模型求出各顶点处的光强I_k。

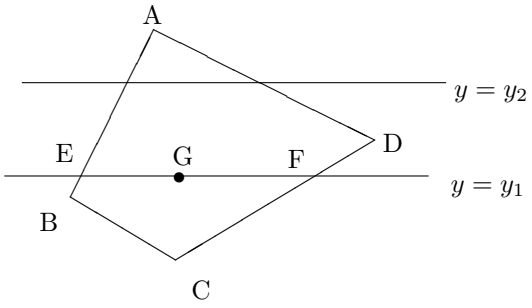


图10.5 Gouraud的光强度插值法

有了各顶点处的光强，就可以用各种插值法求出每个多边形边上各点处的光强。例如对投影到平面$z=0$上的一多边形可采用图10.5所示的线性插值法:先利用公式(10.5.1)计算出每个角点处的平均法向量并进而计算出角点A, B, C和D处的光强I_A, I_B, I_C和I_D，再根据角点处的光强用线性插值求出所有边界线上各点处的光强，例如, 欲求多边形内在扫描线$y=y_1$上一点G处的光强度, 先找出它与边界线的交点E在边AB上, F在边CD上. 点E作为边AB上的一点,其光强为：

$$I_E=tI_A+(1-t)I_B \tag{10.5.2}$$

其中，$t=EB/AB$。同理也可以求出作为边CD上的一点F处的光强为：

$$I_F=wI_C+(1-w)I_D \tag{10.5.3}$$

其中，$w=FD/CD$。

最后,把G看作EF上的点,根据F和E两点处的光强求出G点处的光强度：

$$I_G = uI_E + (1-u)I_F \tag{10.5.4}$$

其中，$u = EG/EF$。在实用中，为了进一步减少计算量，我们定义

$$I_u = I_G = uI_E + (1-u)I_F, \quad \Delta I = I_E - I_F, \tag{10.5.5}$$

则对同一扫描线上各像素点的光强计算通常采用下述的增量公式：

$$I_{u+\Delta u} = (u+\Delta u)I_E + (1-u-\Delta u)I_F = I_u + \Delta I \Delta u \tag{10.5.6}$$

Gouraud明暗处理用于完全漫反射表面时效果比较理想，消除马赫带效应的效果比较明显，而且也很容易把它结合到上一节中的扫描线消隐算法中去。它的主要不足之处是在处理镜面反射时，会使镜面上的高光亮区域的形状和位置发生变化，甚至模糊不清及此时不能有效消除马赫带效应。下面的Phong明暗处理较好地解决了这个问题。

10.5.2 Phong的法向插值法

Gouraud在求出各个顶点的平均法向量后就立即求出各个顶点上的亮度，然后以插值的方法求出各边界以及每一小平面内各点的亮度。Phong的明暗处理方式也是先求出各个顶点上的平均法向量，但他没有立即去求顶点上的亮度，而是用与Gouraud所采用的相同的数学方法求出各边界以及每一小平面内各点的法向量，然后再根据各点的法向量用明暗模型求出各点的亮度。也就是说，只要把公式(10.5.2)～(10.5.6)中的所有光强变量，换成相应点的法向量，就得到了Phong明暗处理。在这里，光照模型被调用的次数大大增加了，因而计算工作量也增加很多，但由此可以改善镜面反射效果。

Gouraud和Phong的明暗处理方法在处理静态画面时可以得到很好的效果，但在处理动画显示时，就出现了下述问题：当画面逐帧更新时，明暗变化太快。这是由于这两种方法都是在图像空间中进行的，即对投影后的图形,而不是对原始空间中的三维物体本身的图形进行明暗处理,因此这两种方法得到的处理结果都不具备对物体旋转的不变性，这样当三维空间中的物体在动画过程中发生旋转时，画面的明暗变化前后可能相差较大而显得不真实。

10.6 光线追踪法

在前一节介绍的光照模型没有考虑直接光源的光线的被折射问题，对于光线的反射也只是采取了简化处理方式。因此这种光照模型通常称为局部光照模型。在下面要介绍的整体光照模型中将考虑光线的折射，并对镜面反射作更深入的处理。光线追踪算法是运用整体光照模型计算像素点光强的一种明暗处理方法，用它可以得到物体的逼真的真实感图像。

10.6.1 整体光照模型

下面通过图10.6所示物体的空间位置关系实例探讨在考虑了反射以后所出现的新情况。假定图10.6中的球体,长方体和三角块均不透明，但它们的表面具有镜面反射能力。

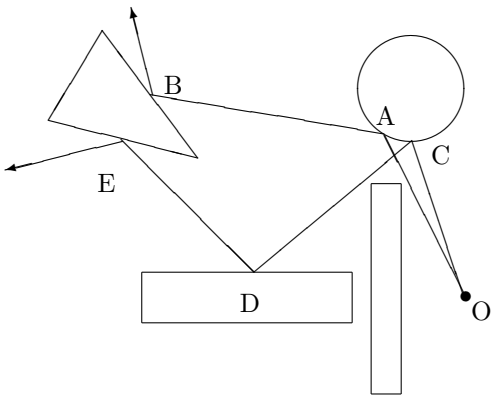


图10.6 考虑反射光后的观察模型

当位于视点O处的观察者注视球面上的点A时，他所看到的不仅有球面，还能看到三角块上的点B在球面上的映象。可见，三角块本来是被两长方块挡往而看不见的，但通过球面的反射就可以被看得见了。

不仅如此，观察者在球面上的点C处还可以看到三角块上的点E，这是光线通过左侧的长方块的一次反射后再通过球面反射而到达人眼的。

因此，观察者可以直接看到长方块背面的点D和球体上的点E；通过球面的直接反射还可以看到倒立的三角块；通过长方体背面的反射还可以看到光线较弱的正立的三角块；而且在三角块的反射光和环境光的作用下，通过球面的反射还可以看到立方体的背面。

从上面的讨论可以看到，消隐算法中首先消除自隐面的做法，已不再适用于整体光照模型，对可见面的查找在此也已失去意义。这也就是说不能直接看到的隐藏面也可能影响要显示的物体表面的光亮度和色彩。因此不能对三维物体的显示采用先消隐的方式进行而要直接考虑三维物体本身的空间位置,求解计算物体表面的光亮度和色彩。同时,在光亮度及色彩的处理过程中还要考虑到物体的透明度,即考虑到透射光的作用.

10.6.2 光线追踪算法

光线追踪法的基本思路可通过图10.7来说明。为了计算观察者视线与表面1上交点的亮度，我们可反向追踪进入视点的那条光线，该光线是某光源的光线到达交点处的反射光和透射光形成的，查找形成反射光和透射光两部分光的入射光.为简单可称其为反射光r_1和透射光t_1，其中反射光r_1 与画面中的其它物体没有交点或跑出画面，可以停止跟踪。

对于透射光t_1，继续追踪，它与表面2相交。该交点处同样可以同样分解出两条光线，其中反射光线r_2跑出画面，可以停止追踪，对透射光线t_2又继续追踪，它与表面3相交，也分成r_3 和t_3两条光线，这两条光线均跑出画面之外，停止追踪。

光线追踪法主要是通过在可能产生光线的方向做射线的方式实现的，因此这一方法也可称为射线追踪法。

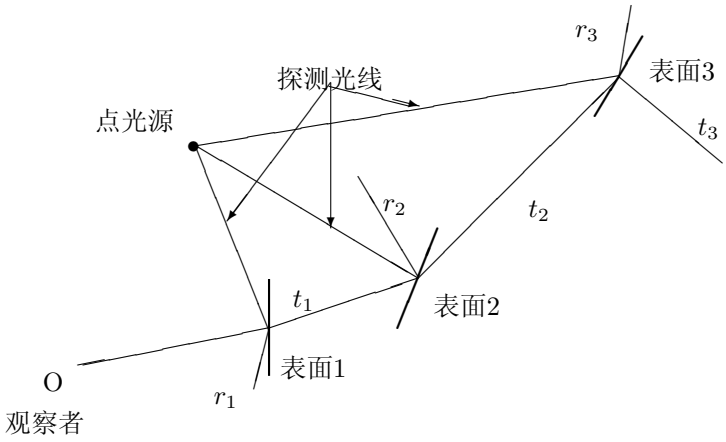


图10.7 射线追踪算法

这一追踪过程可用图10.8的树形结构来表示，树的每一结点表示一光线与表面的交点，在每一结点处分叉成两个子树，左边子树代表该点的反射光线，右边子树代表透射光线，当某一光线停止追踪即跑出画面时，相应的分叉过程终止。

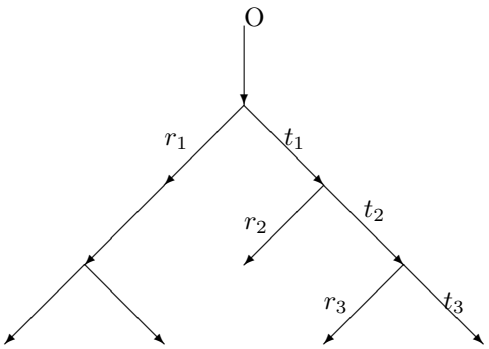


图10.8 射线追踪算法二叉树

从理论上来讲，光线追踪树可以无限地伸展下去，如两个面对面的镜面的互相反射就是如此。但实际的分叉过程可在离原始点较远时自动终止,这是因为光在不断的反射和透射的传播过程中不断地减弱, 因此较远的光起的作用也比较小而可以忽略不计. 这也就是说为了简化计算二叉树分叉的级别可不必太多.

得到二叉树后按后序遍历算法遍历这棵光线追踪树，在二叉树的每一结点处，递归地调用整体光照模型,计算出该点的光亮度并存入该结点的数据结构中，为父结点计算光亮度作好准备。当整棵二叉树遍历完毕时，即按所得的亮度显示表面1上交点的亮度。

在计算时，对于每一个交点，我们还必须首先要确定它是否能为点光源所直

接照射以便确定点光源对该交点处的光亮度的影响(主要是漫反射光的作用). 为此，只要将该点与点光源所在点连一称为探测光线的直线，如果这一探测光线不与画面中任何物体相交，表明点光源的光线能直接照到该点，如果有多个点光源，必须一一判别，并记录下所有直接照射该交点的点光源。反之，若直线与画面中的某个物体相交，表示该交点受不到光源的直接照射，或说它处于阴影之中。

若所遇到的物体不透明，点光源的光线无法透过此物体形成透射光，若为透明面，则点光源的光线可以形成透射光照亮该交点. 透射光的亮度与点光源的亮度相比有一定程度的衰减。这样得到的结果具有阴影效果。

由上述分析可见，光线追踪算法原理并不十分复杂，但是，它的计算量却是相当大的,需要追踪的光线可能相当多，而且对每一条光线都要计算与物体的交点。特别是有多个点光源,以及非点光源时这个算法的计算量是相当大的.

10.6.3 提高光线追踪算法的效率

光线追踪算法是可以产生质量最好的真实感图像的算法，但同时也是耗费时间最多的算法。因此提高光线追踪算法的效率是个重要的研究课题。多年来在这一领域已经取得了一些有意义的成果，这里对此作一简要介绍。

(1) 包围体方法

在光线追踪算法中最耗时的工作是计算光线与物体的交点。利用物体的最小包围体可以尽早排除可能与光线不相交的物体，从而减少计算工作量。

当然对包围体的选择也有一些限制，例如要根据形体的形状特点来选择包围体，使得包围体内尽量由被包围的物体充满, 使得包围体在包围物体的前提下尽可能的小以便尽可能多地排除与被包围的物体不相交的光线。同时，包围体本身要求几何形状简单,一般为球体或长方体, 与光线是否相交的判断比被包围的物体与光线是否相交的判断要简单得多才行.

(2) 预处理法

在计算从像素发出的射线与物体的第一次相交的交点时，可以把所有的隐藏面排除在外。

(3) 空间分剖法

这也是通过减少判断射线是否与物体相交的次数而提高算法效率的，有八叉树方法和体素方法两种实现方法。

(4) 综合方法

指对空间分剖法和包围体法的综合运用，是先对包围所有物体的最小立方体进行分割，直到每个于立方体中包围的物体数不超过两个为止，再对于立方体中的物体作包围体处理。

(5) 射线分类法

如果两条射线的起点和方向都比较接近，那么在它们的连续反射和透射过程中所遇到的表面很可能大多数是相同的。因此在计算出第一条射线的射线追踪树后，与之相近的射线可以先参照这第一条射线的追踪树来建立自己的射线追踪树。例如射线A在某次反射中与物体R相交，那么与射线A相似的射线B在与A同一深度的反射中可以首先判断是否与物体R相交，而不必从第一个物体开始逐个判断计算。

10.7 阴影处理

当光源从一个方向照射到某物体上时，物体的某些表面会挡住一部分光线，这些表面相对于光源就是可见面,且对光源隐藏了在这些可见面背后的物体的表面由此这些表面也可称为物体相对于光源的自隐面。设想这部分光线继续沿原来的方向前进，但起点已改成光线与物体自隐面的交点，那么这些设想的光线所能到达的空间就是该物体在指定光源下形成的阴影空间。可见，阴影空间只与光源和物体有关，而与观察者的位置无关。

但人们所看到的阴影与观察者的观察位置有关，如果只有一个光源,在观察者站在光源处这种特定情况下时，观察方向与光线方向一致，观察者就看不到物体的阴影，而当观察方向逆着光线方向时，观察者看到的全是阴影。一般的情况是观察方向与光线不一致，因此存在着由光源、物体和视点共同决定的阴影。

在画面中设计客观存在的阴影可以强化画面上景物之间的远近深浅的效果，从而极大地提高画面的真实感。确定阴影区域在一些仿真应用中也是重要的，例如建筑物的采光设计、太阳能设施的配置等。

在设计阴影时，我们感兴趣的是湮没在阴影空间中而又为观察者看得见的表面，而不是阴影所占据的实际空间。画面上的阴影就是处于阴影中的表面，它们可以分成两类：自身阴影和投射阴影。

自身阴影是由于物体自身的遮挡使光线不能到达其某些表面而形成的阴影，利用与求自隐面类似的方法可以求出这些阴影（假设视点在光源处）。投射阴影是由于较靠近光源的物体遮挡光线使高光源较远的物体上的部分或全部表面不能被光线照射而形成的阴影。

假设视点在光源处而求出的非自隐隐藏面（相对于光源的不可见面）即为投射阴影面。实际出现在画面上的阴影则是自身阴影和投射阴影相对于观察者的可见面。因此，画面中阴影的生成过程基本上相当于相对于光源和相对于观察者作两次消隐。

一种简单而直观的阴影产生方法是利用光线追踪法，从视点出发，作通过像素点的射线，找出与它相交的第一个物体及交点。这是一个可见点，从此点出发向光源作射线，如果这条射线与不透明物体相交，则此点在阴影中。另外的一些提高光线追踪算法的效率的方法，如预处理方法，阴影体方法，和空间分剖法等等也可以用来产生阴影。

10.8 纹理映射

纹理是对物体表面细节的总称。计算机图形学中的纹理既包括通常意义上物体表面的纹理即使物体表面呈现凹凸不平的沟纹,同时也包括在物体的光滑表面上的彩色图案,通常我们更多地称之为花纹.

凹凸不平的沟纹纹理多见于自然物品的表面,精确地说应当是立体的,常常表现为粗糙的表面,花纹则多见于艺术工艺品的表面,主要是光滑的物体表面上绘制的的彩色图案.

对于花纹而言,就是在物体表面绘出彩色花纹或图案,产生了纹理后的物体表面依然光滑如故。

对于沟纹而言,实际上也是要在表面绘出彩色花纹或图案，同时要求视觉上给人以凹凸不平感即可. 凹凸不平的图案一般是不规则的,这只要在绘制规则图案的过程中加入一个随机扰功函数即可实现。

在计算机图形学中,这两种类型的纹理的生成方法完全一致, 这也是计算机图形学中把他们统称为纹理的原因所在. 所以纹理映射就是在物体的表面上绘制彩色的图案.

10.8.1 图案型纹理映射

先考虑物体表面图案类型的纹理映射。假设花纹图案已经定义好,称其定义在一个纹理空间上,具有坐标(s,t);物体的表面有坐标(u,v). 纹理映射实际上是从一个二维的纹理空间到另一个物体表面的二维空间的映射. 有了两个空间中点的对应关系,图案如何绘制在表面上和表面上的点应取图案中的什么色彩就确定了. 这个过程相当于把一个有图案的彩纸贴在物体的表面上。

设两个空间中点的映射关系为

$$\begin{cases} u = u(s,t) \\ v = v(s,t) \end{cases} \qquad \begin{cases} s = s(u,v) \\ t = t(u,v) \end{cases} \tag{10.8.1}$$

纹理映射函数可以根据需要任意选取,理论上没有任何附加要求，但通常假设它们是线性函数，因此可表示为：

$$\begin{cases} s = au + b \\ t = cv + d \end{cases} \tag{10.8.2}$$

进一步假设纹理空间中的一个图案为由参数方程

$$\begin{cases} s = s(w) \\ t = t(w) \end{cases} \qquad w \in [w_0, w_1] \tag{10.8.3}$$

定义的曲线给出,要把这一图案绘制在由参数方程

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u,v) \end{cases} \qquad (u,v) \in [u_0, u_1; v_0, v_1] \tag{10.8.4}$$

给定的曲面上.根据映射关系式(10.8.1),图案曲线(10.8.3)绘制在曲面上时,曲面上的图案曲线的参数方程为

$$\begin{cases} x = x(u(s(w)), v(s(w))) \\ y = y(u(s(w)), v(s(w))) \\ z = z(u(s(w)), v(s(w))) \end{cases} \qquad w \in [w_0, w_1] \tag{10.8.5}$$

如果图案是由指定纹理空间中各点的颜色指给出的(如图案就是一个已知的位图文件给出的),颜色值函数为

$$c = c(s,t), \qquad (s,t) \in [s_0, s_1; t_0, t_1] \tag{10.8.6}$$

则这个颜色函数定义在物体表面上就成了

$$c = c(s(u,v), t(u,v)), \qquad (u,v) \in [u_0, u_1; v_0, v_1] \tag{10.8.7}$$

如此处理的问题在于在显示器上显示图形时每个像素点是有大小的,即代表一小块曲面片而不是一个点.对应到纹理空间中相应的也不是一点.因此每个像素点对应着纹理空间的一个小区域.像素点的颜色亮度就不能由纹理空间中的某一个点而是对应的小区域内所有点的颜色亮度来确定.一般我们取为对应的小区域内所有点的颜色亮度的平均值.

10.8.2 凹凸不平型纹理映射

使表面呈现凹凸不平效果的纹理映射有两种方法来实现：一是采用前一小节的方法将具有凹凸不平感的图片用上述的方法映射到物体的表面上，这种方法比较简单，但实际效果很不理想；二是使表面真的凹凸不平起来再生成表面上的光亮度。下面就探讨这种方法。

设物体的表面$P(u,v)$ 上任一点处沿u, v方向的偏导向量分别为P_u、P_v，则该点的单位法向量为：

$$\overrightarrow{N}=\overrightarrow{N}\ (u,v)=\frac{P_u\times P_v}{|P_u\times P_v|} \tag{10.8.8}$$

为了在表面上产生凹凸纹理，使位置矢量$P(u,v)$在表面上该点处的法线上作由纯量随机函数$q(u,v)$施加的随机扰动，这样得到的新位置向量$R(u,v)$为：

$$R(u,v)=P(u,v)+q(u,v)\ \overrightarrow{N} \tag{10.8.9}$$

由此得$R(u,v)$点处沿u, v方向的偏导向量为：

$$\begin{cases} R_u=P_u+q_u\ \overrightarrow{N}\ +q\ \overrightarrow{N}_u \\ R_v=P_v+q_v\ \overrightarrow{N}\ +q\ \overrightarrow{N}_v \end{cases} \tag{10.8.10}$$

通常施加的扰动幅度, 即$|q(u,v)|$都非常小使得我们可以略去上述两式中的最后一项，因此近似地由下面的关系式,

$$R_u\times R_v=\overrightarrow{N}\ +q_u(\overrightarrow{N}\ \times R_u)+q_v(\overrightarrow{N}\ \times R_v) \tag{10.8.11}$$

而新的法向量为：

$$\overrightarrow{N'}=\frac{R_u\times R_v}{|R_u\times R_v|} \tag{10.8.12}$$

在光照模型中用$\overrightarrow{N'}$代替$\overrightarrow{N}$即可得到凹凸不平的纹理。

习题10

1. 如何通过三基色获得更多的颜色表示?
2. 为什么要规定出XYZ三色系数？
3. 光源照射后会产生什么光？如何产生？
4. 什么是环境光？是如何产生的？
5. Gouraud和Phong明暗处理模型的主要不同是什么?各有什么特点?
6. 简单叙述光线追踪算法的过程.
7. 简单叙述两种纹理映射的方法及适用问题.
8. 用纹理映射的方法编程绘出有黑白方格图案的球面.

Chapter 11

图形交互技术和用户界面设计

计算机是根据人的意愿工作的。这就需要使用者，即用户，与计算机及时方便地交流信息，即人机对话。提供两者信息交流的设备即交互设备。交互设计技术即实现两者信息交流的设计方法及软件实现。现在的计算机应用软件常常包含有窗口、下拉式和弹出式菜单、图标，以及用于确定屏幕光标位置的鼠标等设备。常用的X窗口、Window操作系统以及在此基础上开发的各种应用软件都有图形用户界面。这些系统应用涵盖文字处理、报刊系统、数据库、文件管理系统、演示系统和页面布局系统等非图形处理应用的软件；以及工程设计、建筑设计、数据可视化、绘图、商用图表和艺术家画笔程序等图形应用的软件。所有这些软件都设计了基于专门的计算机图形技术的交互对话图形窗口系统。在本章中，我们讨论图形用户界面的基本元素和交互对话技术。讨论在图形软件包中，怎样通过对话来构造和管理图形的对象、选择菜单项、指定参数值，以及选择和定位文字串。一般的图形软件包都能与多种输入设备建立交互界面，并提供多种对话功能。

11.1 逻辑输入设备

图形程序使用多种输入数据。描述一张图形需要确定坐标位置的数值、字符串属性参数的数值、作为变换的各类型参数标量值、指定菜单选项的数值和标识图形属性的数值。常见的任何一种输入设备都可以用来输入各种图形数据，但用于输入指定类型数据时每种设备依据其特点会比其他设备更适合。例如在屏幕上指定位置用鼠标输入坐标位置数值时，比键盘输入更方便。因此图形软件的设计和操作或多或少与某种特定的图形设备有关。为了减少系统对物理设备的依赖性，提高系统的独立性和灵活性，图形数据输入功能按照其处理的数据的逻辑功能进行分类，给出了如下逻辑输入设备。

图形软件通常把各种输入数据功能概括成以下六种逻辑输入设备：

1. 定位设备（LOCATOR DEVICE）：指定单点位置坐标（x，y）的设备;

2. 笔划设备（STROKE DEVICE）：指定一组点位置‘坐标的设备;

3. 字符串设备（STRING DEVICE）：指定文字输入的设备;

4. 定值设备（VALUATOR DEVICE）：指定标量值的设备;

5. 选择设备（CHOICE DEVICE）：选择菜单项的设备;

6. 拾取设备（PICK DEVICE）：选择图形的组成部分的设备.

在实际应用中一个物理设备可能用于实现多个逻辑设备功能，多个逻辑设备如定位和笔划操作也可在软件系统中使用同一种逻辑设备，然后由软件判定需要一个坐标位置还是一串坐标位置而区分是定位或笔划设备。因此有些软件系统把这两种逻辑设备认为是一种而只有五种逻辑输入设备。

下面我们将解释各种物理设备为每一种逻辑分类设备提供数据输入的方法。

11.1.1 定位设备

交互地选择一个坐标位置的直观方法是用屏幕光标进行定位。我们可以通过键盘、鼠标、游戏杆、轨迹球、拇指轮、数字化仪的触笔操作光标以及其他光标定位设备来实现定位。当屏幕光标到达指定位置时，按下特定键将激活对该屏幕点坐标的存储操作。

如果是触摸屏，只要有屏幕就可以了，不需要移动光标的其他设备。触摸屏用户可以直接用手指或其它替代物指点屏幕本身进行定位. 这种直接定位容易导致手臂疲劳，因此触摸屏大多安装在公共场合，供偶尔使用的公众使用。较长时间使用计算机的用户已经习惯于间接定位设备的工作方式。

一个通用键盘一般有四个控制键，控制光标向上、向下、向左和向右移动。增加特定的组合键，就可以将光标沿四个对角线方向移动。持续按下选择的光标键，可以实现光标的快速移动。也可直接用键盘键入坐标值，但这种输入的速度很慢，只有在需要关键点精确的坐标值时可以使用。

其他设备如游戏杆、游戏盘、轨迹球或拇指轮通常装在键盘上，帮助控制光标的移动。

光笔也用来输入坐标位置. 光笔通过检测屏幕荧光体发射的光来进行操作，因此在所选的坐标位置上必须有一定强度的信号出现。正因为如此,使用光笔的软件系统屏幕背景通常不是不发光的黑色区域，以保证光笔可以用于选择任意的屏幕位置。光笔用于定位时，是直接定位设备。

数字化仪、鼠标器、操纵杆和光笔等都是连续移动光标的定位输入设备, 将手的平滑移动变成光标的平滑移动。而键盘上的光标控制键是在特定方向控制光标按一定步长移动的离散定位设备。连续定位设备移动光标比离散定位设备更自然更容易更快速. 并且允许光标向任意方向移动, 而作为离散定位设备的键盘上的光标控制键通常只允许光标向四个或八个方向移动。

11.1.2 笔划设备

笔划输入设备用于输入顺序的一组点坐标。笔划设备的输入相当于多次调用定位设备。

许多用于产生定位输入的物理设备如鼠标、轨迹球、游戏杆等均可以作为笔划逻辑设备。这些设备连续移动定位输入点，并转换为一组坐标位置值。输入的一组点常用于显示折线段。数字化仪是一种典型的笔划设备. 其指示笔的按钮按下后数字化仪即进人连续模式. 当指示笔在数字化板上移动时就产生一组点的坐标值。这种功能在画不规则图形时特别方便，常用于允许艺术家在屏幕上绘画的画笔系统，以及跟踪布局并将其数字化存储用于以后的工程系统。

手写体识别输入也是一种极有前途的字符输入方法。用户使用的手写体识别软件用笔划设备在数字化仪平板上写出字符，由手写体识别软件用模式识别技术提取字符的特征，再找出相应的字符来。

连续定位的笔划输入设备连续定位坐标的点理论上有无限多，实际上我们只采集了有限多。实际应用中我们需要控制这有限多到底是多少。在定位输入设备中我们控制这有限多是一次只需要一个点。

11.1.3 字符串设备

最常用的字符串输入的物理设备是键盘。输入的字符串通常作为图形的标记或注解。另外，手写体识别输入也是一种极有前途的字符串输入方法。在相关物理设备如称为写字板的数字化仪上，在写字模式下手动写出字符图案，并通过笔划或定位逻辑设备在屏幕上逐步绘制字符。然后，有关的模式识别程序将使用存储有预定义图案特征的字典来解释这些字符，输入字符。由于手写体字的不规范性及因人而异, 识别效果不好. 对比较规范的印刷体字, 识别效果很好.

11.1.4 定值设备

定值设备在图形系统中用于输入数量值, 设定各种图形参数，例如旋转角度、缩放系数，还可以在物理机制的模拟图像生成中为对特定物理参数如温度、电压等级、强度系数设置具体的数值等等。

任何一个带有一组数字键的键盘都可以作为定值设备。用户可以用浮点数的格式直接键入数值。这种方法尽管比较简单，且数值准确，但输入速度比较慢。克服输入速度比较慢的方法之一是在屏幕上显示滑动块、按键、旋转式标尺和菜单等，然后通过鼠标、游戏杆、轨迹球等其他交互设备快速移动滑动块、按键、旋转式标尺和菜单等输入相应的数据。

当然鼠标、游戏杆、轨迹球等运动后对应成定值输入设备，由程序在对应的数值范围解释为相应的任何特定含义的物理数值。向一个方向的移动，例如从左向右，就增加输入的数值；反方向移动就减小数值。作为一种反馈机制，可以将某种符号显示在所选择的位置。也可以在屏幕的适当位置回显数值以便确认。

也有直接提供定值输入的物理设备如一组控制旋钮。通过旋转旋钮可以输入预先指定的任意范围内的浮点数。向一个方向旋转旋钮将增加输入值，向相反方向旋转则减小输入值。旋转式电位器就是将旋转钮的转动转换成对应的电压。该电压再转换成预定义的指定范围内的一个实数。而滑动电位器则用来将线性运动转换成数值。

11.1.5 选择设备

选择输入设备的功能是从从一组选项构成的表中输入一个选择项。图形软件使用莱单来选择程序设计选项、参数值和各种图标形状的选项。通常选择输入设备由一组按钮构成，相应的光标定位设备可以是鼠标、跟踪球、键盘和触摸板等。

键盘上的每个键都可以被应用程序定义为功能键，按下某个功能键即执行相应的功能。如果用键盘从菜单中进行选择通常将可选项编号或标上缩写字母. 这样只需按一次或两次键即可完成选择。

我们也可以使用光标控制设备对屏幕上列出的菜单选项进行选择。定位设备首先需要判断当前光标位置落在屏幕上的哪个菜单所在的区域，并以此判断选择了该菜单的哪一项, 实现选择功能。

对每次只显示少量选项的大菜单，特别是在面向非专业的不熟悉计算机操作的用户的公众咨询系统软件中，通常使用触摸屏，使得操作直接、易懂。。

也可通过声音输入选择输入选项的方法实现对命令或菜单选项的选择。对于这类选择输入的方法，将菜单列表编号或给其一个短标识名等缩写格式的编码方法非常有用。声音输入在选项数量较少时比较有用。

11.1.6 拾取设备

拾取输入逻辑设备的功能是选定屏幕上的图形对象，以便对它们进行变换或其

他的编辑处理。

用于选定图形对象的物理设备与处理菜单选择的物理设备相同，都是光标定位设备。用户可以先使用鼠标、键盘或游戏杆将光标定位在显示屏幕上的选择对象的图形上，并按下选择按键记录光标位置后，系统会根据一定的拾取算法在存储所有图形对象的内部数据结构中找出选择的图形对象，并将它着重显示出来，让用户确认选定对象，然后对拾取对象作进一步的操作，如复制、删除、修改等。

最基本的拾取算法有点的拾取、线段的拾取和多边形的拾取，现在分别介绍如下。

11.1.6.1 点的拾取

显然，当光标直接定位在要拾取的点时，就拾取了指定点。注意到理论上点都是没有大小的，让两个没有大小的点完全重合操作起来是比较困难的。因此，一般事先给定一个拾取精度$r(>0)$，则在当前光标位置为$P_l(x_l,\ y_l)$时，图形的某一点$P(x,\ y)$如果满足

$$(x_l - x)^2 + (y_l - y)^2 \le r^2, \tag{11.1.1}$$

即光标点和待拾取点的距离小于事先给定拾取精度r，则点P被拾取。此时表明$P(x,\ y)$包含于以$P_l(x_l,\ y_l)$为圆心，r为半径的圆上。

(a). 点的圆形拾取域　　(b). 点的正方形拾取域

图11.1 点的拾取方法

为了避免平方运算，可以用下面的表达式代替，

$$|x_l - x| \le r, \quad |y_l - y| \le r, \tag{11.1.2}$$

此时表明$P(x,\ y)$包含于以$P_l(x_l,\ y_l)$为中心，边长为$2r$的正方形上。

点的圆形拾取域的优点是其与坐标轴方向无关, 点的正方形拾取域与坐标轴方向有关.

11.1.6.2 线段的拾取

如图11.2所示，假设线段的两个端点为$P_0(x_0,\ y_0)$和$P_1(x_1,\ y_1)$。如果当前光标点$P_l(x_l,\ y_l)$位于图中高为$2r(>0)$的矩形上时，认为直线段P_0P_1 被拾取. 具体的判定方法如下：

(1). 如果直线段P_0P_1平行于X-轴，则$y_0 = y_1$, 图中矩形的四条边所在的直线为

$$x = x_0,\ \ x = x_1,\ \ y = y_0 - r,\ \ y = y_0 + r. \tag{11.1.3}$$

这时光标点$P_l(x_l,\ y_l)$位于图中矩形上的判别条件是

$$\min\{x_0,\ \ x_1\} \le x_l \le \max\{x_0,\ \ x_1\},\ \ |y_l - y_0| \le r. \tag{11.1.4}$$

(2). 如果直线段P_0P_1平行于Y-轴，则$x_0 = x_1$, 图中矩形的四条边所在的直线为

$$x = x_0 - r, \quad x = x_0 + r, \quad y = y_0, \quad y = y_1. \tag{11.1.5}$$

这时光标点$P_l(x_l,\ y_l)$位于图中矩形上的判别条件是

$$|x_l - x_0| \le r, \quad \min\{y_0, \quad y_1\} \le y_l \le \max\{y_0, \quad y_1\}. \tag{11.1.6}$$

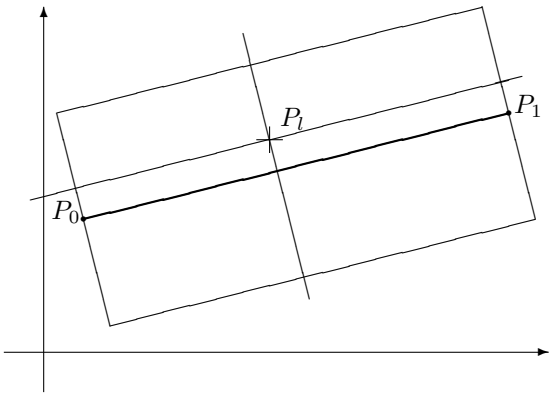


图11.2 线段的拾取方法

(3). 如果直线段P_0P_1和两条坐标轴都不平行，则与直线段P_0P_1 平行的向量为$(x_1 - x_0,\ y_1 - y_0)$, 因此图中矩形的两条与直线段P_0P_1垂直的边所在的直线平行于向量$(y_1 - y_0,\ -(x_1 - x_0))$. 平行于向量$(y_1 - y_0,\ -(x_1 - x_0))$的所有直线的方程可表示为

$$(x_1 - x_0)x + (y_1 - y_0)y = c \tag{11.1.7}$$

的形式。分别将直线的两个端点及光标点的坐标代入这个直线的表达式可得

$$\begin{cases} (x_1 - x_0)x_0 + (y_1 - y_0)y_0 = c_0, \\ (x_1 - x_0)x_1 + (y_1 - y_0)y_1 = c_1, \\ (x_1 - x_0)x_l + (y_1 - y_0)y_l = c_l. \end{cases} \tag{11.1.8}$$

这时光标点$P_l(x_l,\ y_l)$位于过直线段两端点处的两条直线之间的判别条件为

$$\min\{c_0, \quad c_1\} \le c_l \le \max\{c_0, \quad c_1\}. \tag{11.1.9}$$

当前光标点$P_l(x_l,\ y_l)$到线段P_0P_1的垂直距离应不超过r,即满足

$$\frac{[(x_1 - x_0)(y_l - y_0) + (y_1 - y_0)(x_l - x_0)]^2}{(x_1 - x_0)^2 + (y_1 - y_0)^2} \le r^2. \tag{11.1.10}$$

于是光标点$P_l(x_l,\ y_l)$位于图中矩形上的判别条件是

$$\begin{cases} \min\{c_0, \quad c_1\} \le c_l \le \max\{c_0, \quad c_1\}, \\ \dfrac{[(x_1 - x_0)(y_l - y_0) + (y_1 - y_0)(x_l - x_0)]^2}{(x_1 - x_0)^2 + (y_1 - y_0)^2} \le r^2. \end{cases} \tag{11.1.11}$$

当检查到只有一条线段的坐标范围内包含这一光标点时，我们就找到了要拾取的对象。否则，需要进行其他的检查来寻找最接近光标点的线段。避免进行其他的计算来拾取最接近光标点线段的一个方法是高亮度显示候选线段，并让用户自行选择来解决拾取的多义性问题。

11.1.6.3 多边形的拾取

由于多边形是封闭图形，对多边形的拾取实际上就是要判断当前光标位置点$P_l(x_l,\ y_l)$是否在多边形内部。如果在多边形内部，该多边形被拾取；否则不被拾取。

判断点$P_l(x_l,\ y_l)$是否在多边形内部可利用如下的射线法。

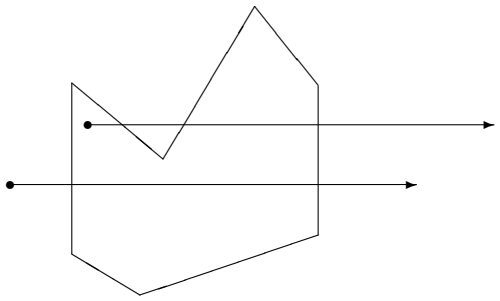


图11.3 射线法示意图

由点$P_l(x_l,\ y_l)$出发向任意方向作射线，计算此射线与多边形的所有边的交点个数。如果交点个数为奇数，则点$P_l(x_l,\ y_l)$在多边形内部（见图11.3中靠上面的射线；如果交点个数为偶数（包括0），则点在多边形外部（见图11.3中靠下面的射线。)

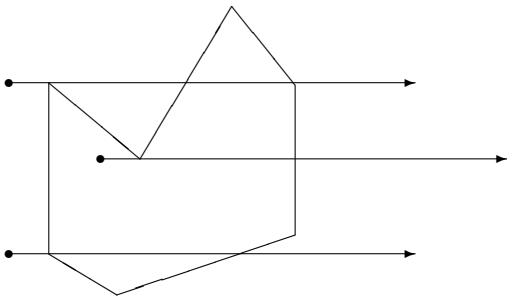


图11.4 射线通过多边形的顶点

通常为简单起见，总取指向X增加方向的水平射线。当射线恰好通过多边形的顶点（即与多边形的边相交于边的端点）时，为保证交点个数的正确性，可作如下规定：水平方向的边可以忽略不计, 如果交点为边的下方端点，则此交点有效，给予计数；否则此交点无效，不给予计数.这样图11.11的上、中、下三条射线的交点个数分别为2、1、2，因此中间射线的端点在多边形内部，其他两条射线的端点在多边形外部。

11.1.6.4 按键拾取

光标定位读出点的坐标，是一种精确的方法，但需要进行复杂的数学分析，一种替代方法是，使用按键输入来高亮度显示屏幕上的待选的图形对象。再用第二个按键在高亮度显示所需对象后停止这一过程。如果这种处理需要搜索太

多的图形对象，则可以增加一个按键来加速这一过程并帮助确认图形对象。第一个按键用来开始快速地依次高亮度显示图形对象。第二个按键用来停止这一过程。第三个按键则用来在所需的图形对象于按下第二个按钮之前就已经高亮度显示过的情况下慢慢地往回退。

我们可以使用键盘来键入图形对象名称或序号，这是一种直接的但交互性较差的拾取选择方法，描述性的名字可以用来在拾取过程中帮助用户。但该方法有一些缺点，它通常比在屏幕上交互地拾取处理得要慢，用户需要提示来回忆各种图形对象的名字。另外，从键盘上拾取图形对象的一部分比在屏幕上拾取更加困难。

这两种替代方法只有在图形不很复杂，待选的图形对象数量较少，且相对固定不变时才较为有效。

11.2 逻辑设备输入模式

用户可指定以下模式设定输入功能.

(1). 使用哪一种物理设备为特定逻辑设备提供数据输入（例如，使用一个数字化仪或鼠标作笔划设备）。

(2). 图形程序和设备如何进行信息交互。程序或设备之一可以要求数据输入还是两者可以同时操作。

(3). 何时输入数据，使用哪一种设备在输入时将特定数据类型传递到指定的数据变量中。

提供数据输入的函数可以按多种输入模式进行组织。每种模式分别指明程序如何与输入设备进行信息交互。可以是程序要求输入，或者程序和输入设备同时提供数据，而且设备也可以要求数据输入。这三种输入模式分别称为请求模式、取样模式和事件模式。

11.2.1 请求模式

在请求模式中，由应用程序请求数据输入。输入过程从提出请求开始，持续到接收到所要求的数据。程序和输入设备交替工作，设备处于等待状态直到程序提出输入请求，然后程序处于等待状态直到收到数据。

这种模式下的输入命令与高级程序设计语言中的标准输入函数相对应。当需要请求输入数据时，程序的其他处理过程在接收到该数据之前暂时停止。正如上前面讨论的那样，在将请求模式指定到一个设备以后，可以使用六个逻辑设备函数之一来对对应的物理设备提出请求输入信号，格式如下：

RequestLDevice(DeviceCode, Status, ...)

LDevice用于区分发出请求的逻辑设备，如用Locator表示定位设备，用Strorke表示笔划设备。该函数的输入值是物理设备代码DeviceCode，可依事先定义好的设备编码赋值。返回值赋给变量Status和对应于请求的逻辑设备的数据参数。

根据获得数据的有效性，变量Sstatus得到返回值“Ok”或“None”。值“None”表示已经激活输入设备但仅产生无效的数据。对于定位设备而言，这说明可能坐标位置超出范围。对于拾取输入，可能是已经激活设备但并没有将其定位到一个图形对象上。也可能是按下了输入设备上的中止键。返回值“None”可作为输入结束信号以终止一个程序的运行。

11.2.2 取样模式

在取样模式下，应用程序和输入设备各自独立地操作、运行。输入设备可以在程序处理其他数据的同时工作。输入设备的新数据取代前面的输入数据而将其存储起来。当程序需要一个新数据时，就从输入设备取得当前值。

一旦为某一个或几个物理设备设定了取样模式，不需要等待程序的指示就开始数据输入。如将一个鼠标或游戏杆设定为一个取样模式下的定位设备，那么就存储该鼠标或游戏杆的当前位置的坐标值。当激活的鼠标或游戏杆位置变化时，当前的位置坐标将不断地替换原来存储的坐标值。

在遇到应用程序中的一个取样命令时，开始对物理设备的当前值取样。定位设备将使用六种逻辑设备对应的某个函数进行取样，格式如下：

SampleLDevice(DeviceCode, ...)

当设备需要时可加一个状态参数，不需要时则没有。

作为取样输入的一个例子，假设要平移和旋转一个选定的对象。对象的最终移动位置可从一个定位设备获得，而旋转角度则由定值设备提供，格式如下列语句所示：

SampleLocator(DeviceCode1, Point)
SampleValutor(DeviceCode2, Angle)

11.2.3 事件模式

在事件模式下，输入设备要求数据输入并交给应用程序。程序和输入设备也是同时工作的，但是输入设备将数据放进一个输入队列中。所有输入数据均存储起来。当程序需要一个新数据时，就从输入队列中取得。

当一个输入设备设定为事件模式之后，程序按照事件模式激活的所有输入设备都可以将数据（称为“事件”）输入到这个事件队列中，每一种设备一旦产生数据，就将其放入队列。任何一个时刻，事件队列可以包含按输入顺序混合的各类数据。进入队列的数据以逻辑设备分类和物理设备码进行标识。

对于任何一个应用函数，可以通过下列函数来检查事件队列中的任何输入：

AwaitEvent(AwaitTime, DeviceClas, DeviceCode)

参数AwaitTime用于设定应用程序的最大等待时间。如果遇到事件队列为空，就将处理挂起直到指定的时间结束或出现一个输入。如果在输入数据到来之前等待时间就已经结束，则参数DeviceClass赋值为“None”。如果给定的时间值为0且队列为空，则程序在检查队列后立即返回到其他的处理。

如果由于AwaitEvent函数使处理转移到事件队列而且队列不为空，则队列中第一个事件就转换为当前事件记录。引起该输入的特定逻辑设备，比如定位设备或笔划设备，将输入数据存入参数DeviceClass中。产生该输入的物理设备存入参数DeviceCode中。

需要时从当前事件记录中取出使用事件模式输入函数输入数据。例如，为了取得一个定位输入，我们可定义函数：

getLocator(Point)

为了快速地交互显示的处理，可以在事件模式下同时使用多个设备。

事件模式下还可以使用某些附加的整理功能。如当一个处理结束而另一个新应用开始时，就需要函数清除事件队列中无用的事件。可以是清除全部事件队列，或是仅仅清除与指定输入设备有关的数据。

11.2.4 各种模式的并行使用及初始化

在请求模式下，一个信号时间段内只能有一个设备可以提供数据输入。在取样模式和事件模式状态下，多个设备可以同时工作。也可以使一些设备处于取样模式状态，而另一些则处于事件模式状态。如使用鼠标将一个选定的对象在屏幕上拖动的过程中，鼠标的位置可以在取样模式下获得，而按下按键事件送入到事件队列。当选定最终位置时，按下鼠标的一个按键可以结束对象的进一步移动。

另外需要注意的是起动设备时需要明确输入设备参数的初值，即对设备初始化，以免带来不需要的结果。有时有必须设定的初始参数，例如：显示光标的类型和大小；拾取操作中高亮度显示的方式；定值输入的范围（最小值和最大值）；定值输入的分辨率（比例）等。当然也有时可以没有, 这是程序有可能随机地给了一个初值. 为确定起见还是应明确设定初值.

11.3 交互式图形设计方法

图形软件中的一些技术方法可以帮助用户进行交互式设计, 提供的各种输入选择方法可以对定位设备和笔画设备直接的坐标输入信息根据要求进行调整和解释。例如，可以限制所有的线段或者是水平的、或者是垂直的。输入的坐标可以作为将要绘制的对象的位置或边界，或者用来重新安排前面已经显示的对象。这些限制性操作的结果不是由用户通过实际操作完成的。用户可以无限制地随意操作，程序经过特定处理自动输出限制性结果。

11.3.1 基本的定位方法

定位输入所提供的坐标值经常与定位方法结合在一起来指定显示一个对象或一个字符串的位置。可以使用一个定位点设备来交互式地选择坐标点，通常的做法是通过定位屏幕光标, 按照选择的功能选项来实现对象或字符串的定位操作。对于字符串，屏幕上的点可作为字符串定位的中心点、起始点、终点，或是其他指定的定位选项。对于直线段，可以在两个选定的屏幕点之间显示相应直线段。

对于具体的定位任务，有时候用户知道需要确定的位置与已有的图形对象在空间上的关系，例如用户要在两个已有的点间画一条线，或者要将一个物体放在另外两个物体中间，此时用户关心的是点的位置而不是点的坐标，因此希望在屏幕上显示出实际位置。另外一些时候，用户知道需要确定的点的坐标值，关心的是点的坐标而不是点的位置，因此希望在屏幕上显示出位置坐标。通过既显示光标位置又显示对应坐标，可以满足两方面的需要。比如，Windows操作系统的附件中的画图工具程序，在其状态栏中就有定位坐标信息的显示功能: 当光标在作图区移动时，实时地显示出光标位置的坐标；当确定了矩形的一个顶点并拖到另一个顶点时，还可以实时地显示出矩形的宽度和高度。

作为定位对象的一种辅助方法，对于所选择的位置的数值可以回显在屏幕上, 方便我们利用回显坐标值来调整选择位置获得精确的定位。

11.3.2 取值任务技术

取值任务是指在给定的范围内,如某个最小值与最大值之间, 指定一个数值。取值任务与指定一个点的定位任务类似，因此也有许多类似的技术方法，有时用户对最终数值只有一个初步的概念, 如通过RGB分量调节颜色时, 用户只有颜色效果的印象, 需要反复尝试才能最终确定所需颜色精确的RGB分量数值，因此希望反复尝试过程中在屏幕上能及时反馈出取某组RGB分量值后的效果。有时

用户知道所需要的精确数值, 但不知道相应对象实际的效果,因此也希望在屏幕上及时反馈出该数值限制下相应对象的实际效果。

可以通过键盘直接键入数值来实现取值任务。也可通过在屏幕上模拟显示出刻度盘、标尺、按钮等辅助工具，再配合定位设备来实现。

11.3.3 标尺、刻度盘、按钮

为了能用比较准确的尺寸来画图，可以使用标尺。用户需要使用标尺、时，可以让系统在屏幕上模拟显示标尺，利用标尺标示的精确数值尺寸来帮助用户决定位置或长度或其它数值的大小，画好图后，再去掉标尺。

在屏幕上模拟显示出刻度盘、标尺、按钮等辅助工具，再配合定位设备, 从鼠标、游戏杆、空间球或其他设备获得定位输入，确定显示器上的一个坐标位置。然后，将该屏幕位置自动转换成屏幕上模拟显示辅助工具对应的一个数值作为输入, 实现取值任务。作为一种反馈机制，可以将某种符号显示在所选择的位置。也可以在屏幕的适当位置回显数值以便确认。

采用图形显示方法时要注意模拟刻度方向含义应与日常习惯保持一致以方便实际操作，如向右、向上移动和逆时针旋转通常表示数值增加方向，而向左、向下移动和顺时针旋转通常表示数值减少方向。

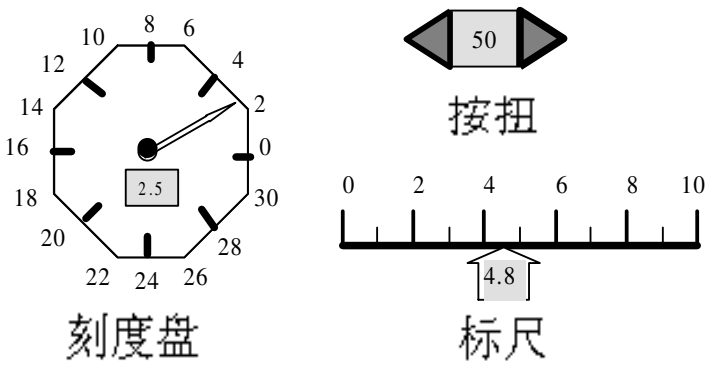
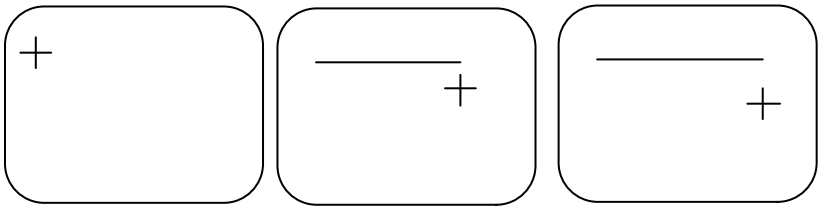


图11.5 几种可能的模拟尺度表示及其数值反馈

11.3.4 约束

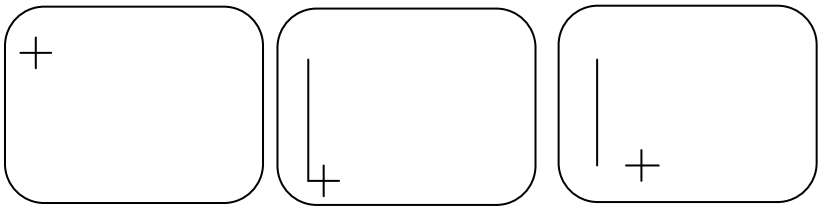
有些应用需要预先设定一些对象的几何特征, 约束就是重新解释，即改变，输入坐标值，以使得得到的新的坐标点满足设计对象事先设定的几何特征. 如要求直线具有特定方向和对齐方式, 则约束就是使任意输入坐标确定的直线能自动修正到指定方向和对齐方式的规则。可以指定多种约束功能，但最常用的约束是直线的水平或垂直方向的对齐。图11.6和图11.7给出的这一类约束，对于形成网络布局是很有用的。使用这个约束，可以在生成水平和垂直线时不必对终点进行精确描述。

实现该约束具体的修正方法是通过判定两个输入的端点接近于水平还是更接近于垂直方向来产生水平或垂直直线。如果两个端点的Y坐标的差值小于X坐标的差值，就显示水平线，否则就显示垂直线。还可以使用其他的约束对输入坐标进行各种对齐处理, 如将直线约束成具有固定角度为45°、30°、60°等；输入坐标也可以约束到预定义的路径，例如在一个圆弧上或一条直线段上。



选择线段初始点　　沿近似水平路径选择线段终点画线

图11.6 水平线约束



选择线段初始点　　沿近似垂直路径选择线段终点画线

图11.7 垂直线约束

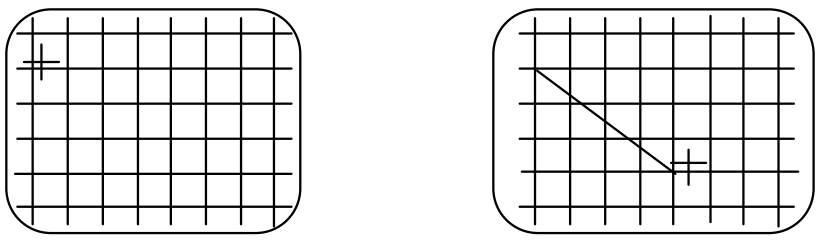
总的说来，约束技术是将光标位置点作为基本的输入，在约束点集中自动找出“最近似”点，作为我们最终需要的点来输出显示的图形对象。

11.3.5 网格

在屏幕上某一部分显示出两组均匀分布的平行直直线的网格是另一类约束。通常是由均匀分布的水平线和垂直线组成的网格。在使用网格时，任何输入坐标位置将移到最近的两根网格线的交点上。

图11.8示例了使用一个网格绘制一条直线段。其中，两个光标位置均移到最近的网格交点上，然后在这两个交点之间画一条直线段。使用网格可以方便地构造图形对象，如一条新的线段可以很容易地与前一条线段相连接. 这时只需在靠近一个线段端点的网格交点附近进行定位即可。

网格线的方向、网格线之间的间距通常可以是用户设定的选项。可以用较低的亮度或较淡的颜色显示网格或网格点，也可以不显示，即网格可以在显示和不显示之间转换，有时还可以使用部分网格，或是在不同屏幕区域有不同大小的网格。



靠近网格交点选择线段初始点　　靠近网格交点选择线段终点画线

图11.8 网格约束画线

11.3.6 引力场

在作图时，有时需要在某线段的端点之间连接另外的线段，由于直接由用户把屏幕光标点精确地定位在连接点是很难的，特别是如果连接点本身就是近似地显示在屏幕上时，精确地定位在连接点根本就做不到。因此，图形软件包可以设计自动将一个靠近特定点的光标点位置转换为特定位置的点。这种转换通过在特定点附近建立引力场来实现。

通常在已有直线段的周围可以想象存在一个哑铃状的引力场（如图11.9所示），一旦光标落在引力场内，就会被吸附到直线段上，自动地被线段上距离最近的一个点所代替。在图11.9示例了一直线段的引力场区域，在阴影区内任意的选择点均被移至线段上的某一位置。。围绕端点的区域相对比较大，是为了用户更易于在端点上连接线段。在引力场的圆形区域内选择的位置都被吸附到该区域的端点上。引力场既要足够大以便能帮助定位，又要足够小从而不和其他直线段的引力场发生重叠。如果显示了许多相互靠近的直线段，它们的引力场将会相互重叠，程序当然比较容易判定距离那条直线段最近，但这时用户可能不容易正确地指定一合适的光标点。

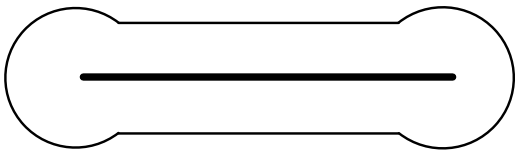


图11.9 围绕一直线段的引力场

图形软件通常不显示引力场,也不提供显示隐藏引力场的转换功能, 因为一般不需要显示出引力场区域或边界。

11.3.7 导向线

如果用户希望某些图形的一端能够对齐，可以让系统在屏幕上显示对齐的边界线，称导向线，利用导向线来作图可以保证图形能够对齐. 在画好图后，再去掉导向线。当然也可以结合标尺、刻度盘实现精确直观的对齐。

11.3.8 选择任务技术

选择任务是指从一个选择集中挑选出一个处理对象，典型的选择集有四种，即：命令、属性、对象类型和对象实例。有些交互技术可适用于全部四种选择集，而有些则适用于某些选择集。不管选择集属于那种类型，利用定位设备指向选择集显示区内其中的一个对象即可选定它.

尽管功能键可以用于选择命令、属性和对象类型，却很难用于选择图形对象本身，原因是用户建立或删除图形对象会引起选择集的经常变化，并且选择集中的被选对象通常也多于可能的功能键的数目。另外,被选对象太多时也使得用户难以清楚所有被选对象含义,直接选择出所需对象.

根据选择集中的选择项数目是否经常变化，可以将选择集分为相对固定（Relatively Fixed－Sized）选择集和可变（Varying－Sized）选择集两类。所谓相对固定选择集，是指选择集中的选择项数目尽管可以变化，但不经常变，而且变化不大，如命令、属性和对象类型选择集就是相对定长选择集；所谓变长选择集，是指选择集中的选择项数目不仅经常变化，而且变化很大，如图形对象本身构成的选择集就是可变选择集。

适用于可变选择集的选择技术有按名字选择和按位置选择两种。适用于相对固定选择集的选择技术有菜单选择和对话框选择两种。当然菜单和对话框中给出的的选择项列出的可能就是选择项的名称。

11.3.9 按名字选择

用户可以直接键入被选对象的名字以选择该对象。这种想法比较简单，但如果存在着几百个图形或非图形对象时，用户经常会不知道或记不住对象的名字，这时候按名字选择就难以奏效。但无论如何，这种技术在下列两种情况下还是唯有的有效方法。

(1). 如果用户知道对象的名字，此时按名字选择可能比按位置选择要快些，特别是在用户需要滚动作图区才能找出所需图形对象时更是如此。

(2). 如果显示的内容杂乱无章，或者图形对象太小又无法放大，此时很难按位置选择，按名字选择可能是唯一的方法。

按名字选择的方法对于那些有经验的用户特别合适。但软件应提供一些辅助性措施帮助用户操作。一般地这些方法措施包括

及时反馈 每当用户键入对象名称的一个字母后，程序自动地立即显示出与当前已经键入的字符串相匹配的对象名称列表。这种反馈形式在用户仅仅能够记住对象名字的前几个字符时，就可以帮助他回忆起对象的完整名字；当只有一个对象的名字与键入的字符串匹配时，该名字会自动着重显示而无须用户键入完整名字，或者由系统自动完成用户尚未键入的部分。

及时校正 拼写校正为按名字选择时常用的另一项技术。如果用户键入的名字与所有被选对象的名字都不匹配，应显示那些与用户键入的名字相近的被选对象的名字以供选择。

命名规则 为对象命名时，要有一定规律，与被命名对象内容相关，即尽可能按照一定的规则命名要处理的对象，以便于记忆。在可能时提供用户自己命名对象或对已有对象重新命名的途径。

11.3.10 按位置选择

菜单或图标、图形显示出要处理的对象，利用定位设备将光标移动到对象上面，再按键确认，即可选择该对象。图形对象由于通常具有层次结构，因此系统还需要确定用户选择的图形对象属于哪一层。图11.10所示为房屋及其层次结构，房屋由房顶和墙组成，而墙又由窗户和墙体组成。当光标落在窗户内时，软件在不清楚用户是要选择窗户还是墙甚至房屋是，是无法作出恰当选择的。

用户有两种指定所需层次的方法：一是用设置选择层次这样的命令直接指定选择对象的层次，如要选择墙，则用该命令将选择层次设为2；二是用上一层和下一层这样的命令来改变当前层次，当选择图形对象时，层次最低的图形对象（如窗户）总是先变为着重显示，用上一层命令可以使得包含该对象的上一层图形对象（墙）变为着重显示，用户可以重复使用上一层和下一层命令直到得到要选的对象为止。

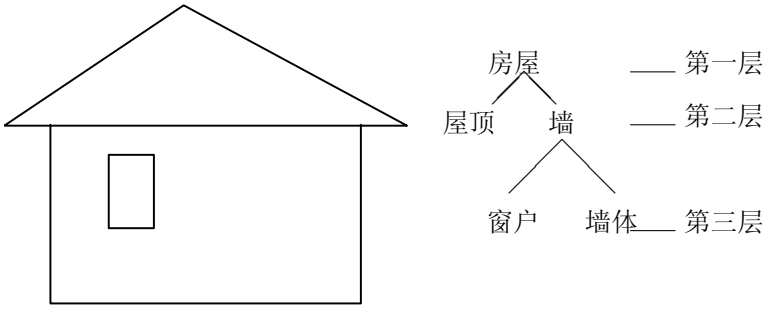


图11.10 房屋及其层次结构示意图

11.3.11 包围盒方法

一种简单的, 但是行之有效的提高拾取和按位置选择效率的方法是包围盒法。对于每个图形对象，都可较方便地得到它的外接矩形. 即图形对象的包围盒。给出了图形对象的包围盒后, 首先对包围盒作拾取检测. 当光标未能拾取包围盒时, 包围盒内的图形对象显然也不可能被拾取. 只有当光标在图形对象的包围盒内部时，才需要进一步判断该图形对象是否被拾取。为了使光标图形对象附近时也能拾取到该图形对象，可以将包围盒适当向外面扩大一点。

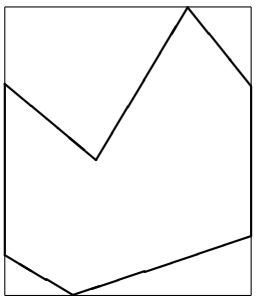


图11.11 多边形的包围盒

对于每个图形对象，也可较方便地得到它的外接圆, 称为图形对象的包围圆。对包围圆, 也有类似于包围盒的结论.

11.3.12 菜单选择

菜单选择主要适合于相对固定选择集的选择技术。设计菜单时需要考虑以下几个方面：

(1) 菜单的层次。当菜单项很多时，需要分析所有菜单项之间相互关系, 用比较自然的、易于理解的方式将它们组织成多级菜单，用分组或下拉式菜单的形式组成菜单。

(2) 菜单项的顺序。同一级的若干菜单项一般先按功能分组，组内再按使用频率或名称的字母顺序排列，组与组之间用分隔符区别开。

（3） 菜单的放置。菜单可以是静态的，自始至终显示在屏幕或窗口的某一固定区域内，如Microsoft Windows程序最上方的菜单栏；也可以是动态的，在用户需要时才显示在屏幕上，用户选择后立即消失，如弹出式菜单、下拉式菜单。

（4） 当前被选项。当前被选项应该用着重、反转、按钮等方式着重表示出来。

11.3.13 对话框

用户有时需要从一个选择集中选择多个元素，比如字符属性集中有倾斜、加粗、下划线等，它们并不是互斥的,而是可以同时起作用的的，用户需要一次选择多个选项。另外，有些不同的选择集是相关的，如字体和字型。采用菜单方法可以从一个选择集中选择一个元素，但是不适合选择多个, 如下拉式菜单和弹出式菜单通常在作出一个选择后就消失了，需要再次激活才能作第二个选择。

使用对话框技术可以解决这个问题. 对话框在用户明确地指示关闭它之前一直是可见的。对话框还可以显示多种选择内容，允许用户从多个选择集中作选择，并且可以提供输入文本和数值等不同类型数值的区域。在对话框中作出的选择在用户确认之前都可以修改。当所有需要的信息都输入到对话框并经用户确定后，对话框就消失了，系统按照多种选择指定的对象及对象属性执行后续操作。

11.3.14 橡皮筋方法

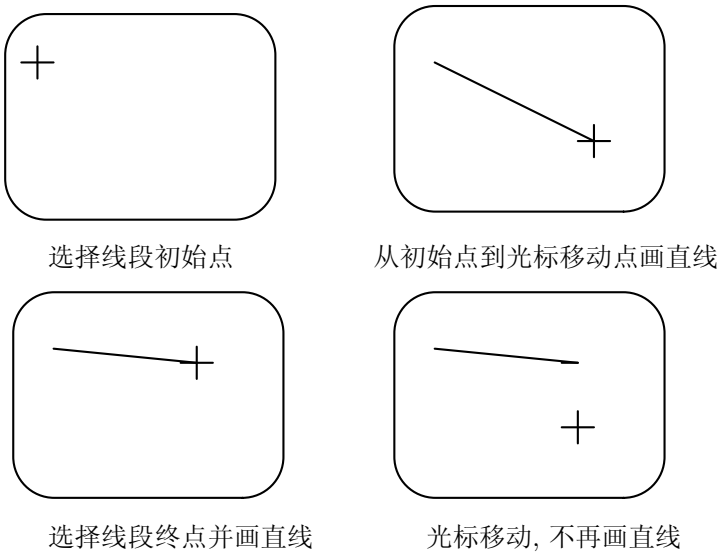


图11.12 绘制一直线段的橡皮筋方法

通过在起始点到不断移动的屏幕光标之间实时地拉出一条直线的方法称橡皮筋方法，用此方法可以构造和定位直线段。图11.12示例了使用橡皮筋技术绘制一直线段的方法。我们先选择作为直线段一个端点的屏幕位置。然后当屏幕光

标移动时，显示从起始点到移动的光标当前位置的一条线段。当最终选择第二个屏幕位置时，得到该线段的另一个端点。

除了直线段以外，橡皮筋方法还可以构造和定位其他对象。图11.13示例了使用橡筋方法构造矩形，图11.14示例了使用橡皮筋方法构造圆的过程。

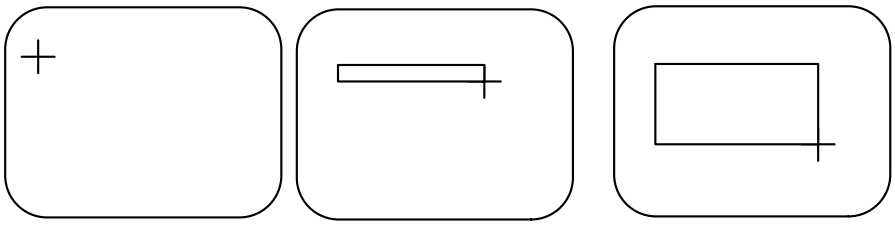


图11.13 绘制一矩形的橡皮筋方法

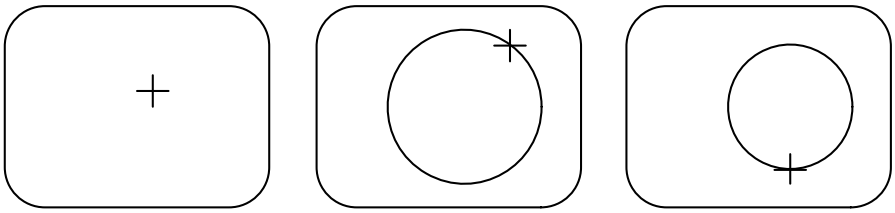


图11.14 绘制一圆的橡皮筋方法

11.3.15 拖动

使用屏幕光标拖动对象来移动该对象，是交互式绘图的常用技术之一。实现的方法是先选择一个对象，然后将光标向所需移动的方向移动，选择的对象就会跟着光标实时移动。对于要在选定最终位置之前分析各种可能性的应用及其图形显示效果，在场景中向各个位置拖动对象是很有用的。

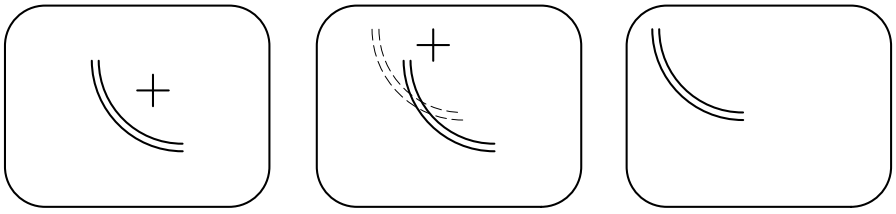


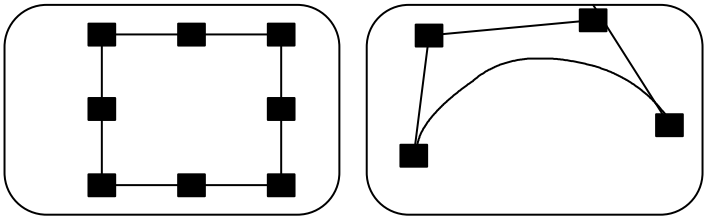
图11.15 光标拖动对象方法

11.3.16 操作柄技术

在图形对象上定义一些点，通过对这些点的移动实现对图形对象进行比例或形状改变的方法称为操作柄技术。相应地我们就可以说在这些点位处定义了一个操作柄，通常用一个黑的小方块表示，操作柄通常并不显示出来。仅在需要时

显示出来，用户通常是移动操作柄实现对图形对象的修改。

图11.16(a)显示了用操作柄技术对一个矩形域内的图形对象进行缩放的操作柄. 用操作柄技术对图形对象进行缩放的方法：先选择要缩放的图形对象，该图形对象的周围会出现八个小方块表示的操作柄，四个在边上，四个在角上。如果用光标拖动某条边上的操作柄标记，则与这条边相对的边固定不动，整个图形对象随着光标的移动而在一个方向上伸缩；如果用光标拖动某个角上的操作柄，则与这个角相对的对顶角点固定不动，整个图形对象随着光标的移动而缩放。这类操作柄技术主要用在非图形处理程序中，用于对嵌入的图形对象从整体上作简单的处理，一般不涉及图形对象的内部构成。这时可以认为图形对象存在于一张弹性薄膜上，弹性薄膜伸缩变化时，薄膜上的图形也相应地伸缩变化了。当光标处在图形对象的非操作柄位置时，一般会不改变图形大小，但平移图形对象改变其位置，即相当于前面的拖动图形技术方法。



(a). 缩放图形对象操作柄　　(b). 修改图形对象操作柄

图11.16　操作柄技术编辑图形对象

拖动和缩放都是针对整个图形对象进行的，有时候用户还需要对图形对象作一些局部调整，比如说移动多边形的某一个顶点的位置，此时也可以使用操作柄技术。图11.16(b)显示了用操作柄技术改动多边形的操作柄标记. 用户先选择要改动的多边形，则多边形的每个顶点上都会显示出一个操作柄；用光标拖动要改动的顶点处的操作柄到指定位置，在此移动过程中，与该顶点相邻的两条边变成橡皮筋，以相邻的两端点为固定点随着光标的移动而改变。同样，如果在多边形的边上定义了操作柄，用户就可以用光标拖动要改动的边到指定位置，并保持其方向和长度不变，在此移动过程中，与该边相邻的两条边变成橡皮筋。如果选择的多边形是用来生成曲线的控制多边形，则相应生成的曲线的形状就会随着多边形形状的改变而改变。

11.3.17　着色和绘图

素描、绘图、着色的选择有多种形式。直线段、多边形和圆可以便用上一节所讨论的方法而生成。曲线绘制可以通过标准曲线形状（例如圆弧和样条）或手绘过程来实现。样条曲线通过指定一组给出曲线大概形状的离散屏幕点而交互地构图，然后系统将使用多项式曲线，并且根据这些点来拟合曲线。在手工绘制中，通过数据板上触笔的路径或视频监视器屏幕光标的路径来生成曲线。一旦显示出一条曲线，设计者可以调整沿曲线路径上选择的点的位置，从而改变曲线形状。

线宽、线型和其他属性选项一般也包括在着色和绘图系统中。这些选择使用菜单方法实现。各种画笔形状、画笔图案、颜色组合、对象形状和表面纹理图案都可在许多系统中、特别是作为艺术家工作站而设计的系统中找到。某些绘图系统根据艺术家的手加在触笔上的压力来改变线宽和笔划。一个绘画系统的

窗口一般设计有菜单等功能，用于让艺术家选择指定对象的形状变化、不同的表面纹理以及场景的各种光照条件。

11.4 交互设计技术图形用户界面

针对某种特定的应用，交互设计方法是对用户与计算机话设计的基础，说明所设计的系统能做什么，应具备什么样的图形操作，指明了可以显示的对象类型以及如何管理对象。例如，作为建筑设计工具的一个图形系统，要如何使用该软件程序定位墙、门、窗和其他建筑构件，从而构造和显示大楼的视图。对于一个设备布局系统，将对象定义为一组家具（桌子、椅子等等），而应具备的操作功能则包括在布局范围内定位一件家具或移动对象。电路设计程序则可以在整个电路设计中使用电子或逻辑元件作为对象，提供增加元件、删除元件及有关的图形操作。

一般要按照实际应用的行业语言来表达用户对话的所有信息。在建筑软件设计中，要使用建筑术语来说明所有交互方法，不显式引入特殊的数据结构或其他建筑设计行业不熟悉的概念。前面我们讨论了常用的交互设计技术与方法，下面将讨论在构造用户对话时一般应注意的问题。

11.4.1 窗口和图标

可视化技术就是要用图形符号表示出计算机的各种功能及处理对象。可视化表示既可用于实际应用中管理的对象，也可以用于对处理对象的操作。

现在常见的窗口系统为用户提供一个窗口管理界面，实现用户显示和管理功能。窗口系统本身的基本功能有打开和关闭窗口、对窗口重定位、缩放功能以及具有内部裁剪和外部裁剪的显示等。一个典型的窗口中包含滑动块、按键、菜单和图标等用于实现各种窗口操作。大部分通用系统，如X窗口系统还可提供多个窗口管理程序，从而使不同风格的窗口可以同时在各自的管理程序控制下实现。窗口管理程序可以按特定的应用要求进行设计。其他的窗口软件则针对一种应用的窗口风格进行设计。

图标是显示在窗口内的标志性符号图形, 代表某个程序或某种功能. 根据所起作用的不同，有不同类型的图标。如用来代表特定处理对象如电路设计中电路元件等的图标称为应用图标。代表旋转、放大、比例变换、裁剪和粘贴等动作的图标称为控制图标或命令图标。

11.4.2 同一功能的多种操作方法

通常，交互式图形界面提供多种选择动作的方法。例如，选项的选择可以通过将光标指向一个图标，然后按下不同的鼠标键而实现；也可以通过下拉或弹出式菜单进行选择；还可以通过键入命令进行选择。这种功能使软件能适应因需要不同而熟练程度不同的用户。

对于经验不足的用户，具有容易理解的操作加上详细提示的界面，要比具有复杂快捷的操作集的界面高效得多。一组简化的菜单和选项比较容易学习和记忆，用户可以把注意力集中在程序的功能及应用上而不是在界面的细节上。对于一个没有经验的应用软件用户，简单的“指点－按键”操作是最容易的。因此，一般的界面都设法掩盖软件的复杂性，从而使初学者在使用该系统时不会因众多的细节而不知所措。

另一方面，有经验的用户需要操作的速度。这意味着较少的提示和来自键盘或多种“鼠标－按键”的复杂的组合形式快速输入。有经验的用户记住了常用动作的缩写，因此经常通过功能键或同时按下组合键来进行选择操作。

同样，也可以分成几个层次来设计帮助功能，使初学者可以进行较为详细的对话，有经验的用户则可以减少或去掉提示和消息。帮助功能还可以包含一个或多个辅导性应用程序介绍软件的功能和使用方法。

11.4.3 一致性

一致性是界面设计中需要考虑的要点之一。一个特定的图标应该始终只有一个含义而不能依靠上下文来代表多个动作或对象。菜单总是放在相同的关联位置，从而使用户不必无目标地寻找特定的选项；总是使用相同的组合键来代表同一个动作；总是使用一种颜色编码方案，从而使相同的颜色总能传递给用户一致的信息。

一般来说，一个复杂的、不一致的模型会使用户难于理解，从而导致工作效率低下。提供的对象和操作应设计成一个尽可能小的和一致的集合，以便使该系统容易学习，但也要注意不能过于简化以至于难于使用。

11.4.4 减少记忆量

界面的操作应该组织得容易理解和记忆。模糊的、复杂的、不一致的和缩写的命令格式会导致软件使用时的混淆和低效。如对于所有的删除操作，使用同一个键比对不同类型对象的删除操作使用不同的键要容易记忆。

图标和窗口系统也可以帮助减少记忆量。不同类型的信息可以分别显示在不同的窗口中，因而在不同的信息重叠显示时不必费力去分辨它们。我们可以简单地在屏幕的不同窗口中保留不同的信息，并轮流在窗口之间转换。通过显示容易辨认的图标（代表各种对象和动作），可以帮助减少记忆量。我们可以简单地通过选择图标来选择它所代表的动作。

11.4.5 回退

在一系列操作过程中，回退和取消机制是用户界面的另一个共同的特点。常常在操作完成之前取消该操作，然而系统将保存操作之前的状态。有了在任一位置回退的功能，就可以放心地使用系统的各种功能。

可以有多种形式来控制回退操作。一个标准的Undo功能可以用来取消一次单个的操作。有的系统可能有回退若干步操作，因此可以把系统回退到某些特定的位置。在具有多步回退功能的系统中，所有输入均被保存，因而可以回退并“重复操作”任意一部分。

11.4.6 删除和出错处理

设计一些好的诊断程序和提供出错消息，可以帮助确定发生错误的原因，尽可能减少程序出错或操作出错造成的损失。

有些操作如出错，损失是不可恢复的，对于这种类型的操作一般可通过再提问确认的方式提醒用户。例如，一旦我们删除了桌面废纸篓中的内容，就不能再找回已经删除的文件，应该在用户给出删除操作后，通过提问提示用户，等待用户确认删除操作之后再进行实际的删除工作。

另外，界面还通过对可能导致错误的一些动作进行预测来减少出错的可能性。例如，当没有选中对象时，不允许作移动一个对象的位置或删除对象的操作；当选中的对象不是某类对象如“线”时，不允许选择相应对象线的属性；当剪贴板上没有对象时，不允许选择粘贴操作。也就是说，某个操作能正常进行的条件不具备时，程序要自动限制该操作。

11.4.7 信息反馈

用户界面有时需要设计成能够完成前后相关的一系列交互对话的复杂形式。因此，必须了解对话过程中每一步动作的进展情况。

系统在接收到每一次数据输入后常常会给出某种类型的响应。如高亮度显示某个对象，出现某个图标或显示某个信息。这不仅告诉用户系统接收到了什么样的输入数据，同时还告诉用户系统正在做什么。如果在几秒钟或更长的时间段之内不能完成某个操作，则可以显示一些信息来告诉用户系统的进展情况。系统也可以在完成操作的过程中逐步显示部分结果，而最终结果是一次一次部分显示结果的综合。系统还可以在执行一条指令的过程中，让用户输入其他的命令。

对于特别长时间才能完成的操作，以及没有中间结果的操作，可以给出一个闪烁的信息，指出系统正在按输入的要求进行正常处理，以免用户茫然无措，不知道程序是否正常运行。

当界面的响应速度很高时，这一点也很重要。如果没有反馈，我们可能会搞不清系统进展情况以及何时应继续输入必要的数据，以保证程序的正常运行。

通常应提供足够清晰的反馈信息，从而使其不易被忽略，但也不能过分突出反馈信息以至于影响用户的注意力。当按下功能键时，可给出听得见的点击声作为反馈，也可以高亮度显示该功能键作为反馈。听觉反馈的优点是其不使用屏幕空间，用户也不必把注意力集中到工作区内以得到反馈信息。如果反馈信息显示在屏幕上，那么使用固定区域，可以使用户知道在什么地方可查看该信息。有时，将反馈信息安排在光标附近有一定的好处。也可以用不同颜色显示反馈，从而使不同的显示对象较容易地相互区别开。

为了提高响应速度，要采用与所用设备操作特性相结合的反馈技术。典型的屏幕反馈技术是反转像素的亮度，尤其是在选择菜单时。也可以对字符采用高亮度显示、闪烁和改变颜色等方法反馈显示。

对于不同类型的反馈，可以设计专门的特色图形、文字符号。例如，一个交叉符号、一张皱眉的脸或一个向下的拇指符号常用来指出一个错误；一个闪烁的“正在工作”符号用来指出处理正在进行之中。这种类型的反馈对于熟练用户是很有效的，对于初学者可能需要提供更详细的信息，不仅要清楚地指出系统正在做什么，还要指出用户接下来要做什么。

某些类型的输入要求回显反馈信息。输入的字符在键入时即回显在屏幕上，使用户可以立即发现并纠正错误。按键和拔号盘输入可以使用相同的方法进行回显。通过拔号盘或显示的标尺而输入的数值通常回显在屏幕上，以便检查输入值的精度。坐标点的选择操作，可以使用光标或其他符号在所选位置进行回显。如果选择的位置要求精确的坐标，可以将坐标值显示在屏幕上特定位置上。

图形交互一般使用窗口和图标进行设计。一个窗口系统提供一个带有菜单和图标的窗口管理界面，允许用户打开、关闭窗口，对窗口重定位以及缩放窗口。窗口系统则包含完成这些操作以及各种图形操作的子程序。通用窗口系统设计成支持多个窗口管理程序。图标是为了应用处理或控制处理的快速标识而设计的图形符号。

用户对话设计的指导思想是易于使用、透明性和灵活性。而且，图形用户界面应设计成能保持用户交互的一致性及支持各种熟练程度的用户。另外，界面应设计成能尽可能减少对用户的记忆要求，提供足够的反馈，并提供适当的回退和错误处理能力。

图形程序的输入可以来自多种不同的硬件设备，多个设备可以提供同一类输入数据。通过对输入设备进行逻辑分类，图形输入函数可以设计成与特定的输入设备无关。即设备按图形输入类型而不是按硬件名称如按鼠标或数字化仪行分类进行分类。常用的六类逻辑设备是定位设备、笔画设备、字符串设备、定

值设备、选择设备和拾取设备。定位设备是任何一种用来输入一个坐标位置的设备。笔画设备输入一串坐标值。字符串设备用来输入文字。定值设备是任何一种用来输入标量值的设备。选择设备输入菜单选择号。拾取设备输入一个图形对象名。

一个图形软件中的输入函数可以定义成三种输入模式。请求模式将输入的控制交给应用程序。取样模式允许输入设备和程序同时工作。事件模式允许输入设备启动数据输入并控制数据的处理。一旦为一个逻辑类和用来输入该类数据的特定物理设备选定了某种模式，那么程序中的输入函数就可以将数据值传递到程序。一个应用程序可以同时使用若于个在不同模式下工作的输入设备。

各种应用，包括设计和绘画软件都使用了一些交互式构图方法。这些方法能让用户定位对象、按预定义的方向和对齐方式约束图形、绘制草图以及在屏幕上拖动对象。网格、引力场和橡皮条方法用来帮助定位和其他构图操作。

习题 11

1. 用射线法编写判断一个给定点是否在多边形内部的程序。
2. 试编写一个用橡皮筋技术画线段的程序，并要求当距离小于10的时候，线段的端点可以自动约束在屏幕上已有线段的端点上。
3. 设计一个程序，该程序允许用户使用一个笔划设备交互式地画图。
4. 设计一个子程序，该子程序可以在屏幕上显示一个线性标尺和一个滑动块，并允许通过沿标尺线定位滑动块来输入数值。选择的数值显示在靠近线性标尺的框内。
5. 设计一个子程序，显示一个圆形标尺和能在圆上移动以选择角度（以度为单位）的指针，或是一个滑动块。选择的角度值回显在靠近圆形标尺的框内。
6. 编写一个绘图程序，该程序允许用户通过指定点来绘制线段，并组成图形。每条线段的坐标由定位设备进行选择。
7. 编写一个通过在指定点间连接直线段而构图的绘图软件。建立一个围绕图中每一条线段的引力场，从而帮助将新的线段与已有线段相连接。
8. 修改习题7中的绘图软件，从而允许对线段进行水平或垂直约束。
9. 开发可以显示一个任选网格图案的绘图软件，使得选择的屏幕坐标能近似移动到网格交点上。该软件能提供画线功能，并且从一个定位设备取得顶点。
10. 编写一个可以使用橡皮条方法绘制直线段来构图的子程序。
11. 编写一个可以用橡皮条方法构造直线段、矩形和圆的绘图软件。
12. 编写一个程序，允许用户通过一个基本形状菜单并使用一个拾取设备，将每一个选取的形状拖动到指定位置来设计图形。
13. 选择几种读者所熟悉的图形学应用，并建立用户模型，能够作为该领域内图形应用系统用户界面的设计基础。
14. 请列出能在一个用户界面中提供的帮助设备，并讨论哪一种帮助对不同层次用户是合适的。

15. 请总结可以处理回退和回显的方法。指出哪一种比较适合于初学者，哪一种比较适合于有经验的用户。

16. 请列出几种向用户表示菜单的格式，并解释每一种在什么情况下是合适的。

17. 讨论对于各种层次用户的反馈的不同选择。

18. 列出一个窗口系统在处理具有多个重叠窗口的屏幕布局时，必须完成的功能。

19. 为一个M层结构的建模软件设计一个用户界面。

20. 为任何一个读者所熟悉的领域，设计一个可以由该领域任何用户使用的图形软件用户界面。

21. 开发一个能使用定位设备在屏幕上定位对象的程序。给出一个几何形状的对象菜单，让用户选择对象及安放位置。该程序应允许对任意多个对象确定位置，直到给出一个结束符号。

22. 对上述程序加以扩充从而使对象在定位前可以缩放和旋转。变换选择及变换参数均以菜单项形式给出。

23. 设计实现请求模式中输入函数的子程序。

24. 设计实现取样模式中输入函数的子程序。

25. 设计实现事件模式中输入函数的子程序。

Chapter 12

计算机图形动画设计

12.1 介绍

动画顾名思义就是会动的画,或者更精确一点,是我们视觉上感到会动的画, 因为画其实是不会动的. 前面各章介绍的二维、三维或带有各种颜色、灰度的真实感图形的理论和方法用以产生图形画面,是计算机图形技术发展和应用的基础. 计算机动画技术则是计算机图形技术更深入一步的发展和应用. 动画技术就是要求相关联成组的图形按照一定的规则移动或活动起来,是借助计算机生成一系列可供动态实时演播的连续图像的计算机图形技术。利用动画就能显示出计算机图形的第四维空间即时间。这一技术将计算机图形学的领域扩展到更深入的程度，成为计算机图形学这一学科中最吸引人的领域之一，具有广泛的应用前景，同时它也将产生重大的经济和社会效益。

计算机动画是一门涉及多学科基础和最新知识的技术，有很多内容有待进一步的研究和探讨并处于不断地发展变化之中。本章仅介绍计算机动画技术的一些概念和一般的实现方法。

12.2 动画原理及制作技术

我们首先了解一些传统的，即人工动画技术的方法和概念.

12.2.1 动画原理

动画本身并不是一个生疏的概念，卡通动画片是动画的最直观明白的实例。动画就是移动的画面组成的,是由相互关联的一系列画面所产生的动态过程, 在这个过程中每一幅画面不断地被新的画面所取代, 使观察者产生连续运动的动态画面的幻觉。

画面本身是静止的。如何使一幅幅密切相关的静止画面移动并形成一种连续的动作，完全取决于设备和技术。卡通片的动画，是将一个连续动作，分解为若干个小动作，然后画出每个小动作的画面，再用摄像机将这些小动作的画面连续拍摄下来，形成一组有联系的照片，然后再以一定的速度使这一组照片通过放映，展现在人们面前。当一幅画面在眼睛前面移去以后，画面的内容并不会在视网膜上马上消失，而是要保留极短的一段时间，这种现象称为视觉残像（视觉的持续性persistence of vision）。如果在这个时间内有另一幅与其相似的画面出现时，就会同时有两幅画面留在视网膜上，其区别在于一个很小的动

作差别。这样当一组连续的动作画面不停地在眼前出现时，就自然形成了一个连续动作的错觉。

电影也是利用了人眼的这一视觉特性。在我们看电视或看电影时，会感到各个画面的动作都是连续的,使人看到动作很流畅的有动感的画面效果。但是，如果我们观察相应部分的胶片，就会发现动作并非是连续的，实际上，电影或电视都是利用许多静止的画面做成的。

电影、电视以及动画等，都是巧妙地利用了视觉上的残像现象的。电影界所使用的标准速度为每秒24幅画面，但实际上，每一幅画面都投影两次。因此，实际的投影速度为每秒48幅画面。这样，产生动画就需要每一秒钟有24幅图。从经济的角度和人的视觉特性两方面来考虑，每秒钟24幅图这个数字是合适的。顺便提起不同于电影的是录像（电视）的图像是每秒钟30幅。

12.2.2 动画的制作

动画片的制作是非常费时间的，一般需要花费大量的人工。比如，如前所述，每一秒钟的影片需要24张图片。要制作一部30分钟的影片，就需要制作43,200幅图片。动画片的每一幅图片都是用手工的方法制作的. 一般来讲，人工制作卡通动画片是按照一个事先规划好的剧本一步一步进行的。大致可分成以下一些步骤：

(1) 初期准备：首先进行总体规划，以脚本的形式将故事写出来。然后对脚本进行分析，用人工的方式在大幅面纸张上，画出各个重要场面景况的一系列草图。这样的一组画板组成了故事板，表现剧本所设计的动画动作，据此概要地列出故事的主要情节。同时，对画面的风格、声音、动作、色彩等各个方面，加以必要的注解。故事板的数量根据影片的长度和此阶段所需要描写的细节而定，有时需要几十张甚至几百张。复杂的动画片比简单的影片所需要的故事板的数量要多。

(2) 分层安排：故事板完成了以后，将其进一步进行分层处理，首先将剧本分成若干集，每一集分成若干组镜头，每一组镜头又分成若干帧，每一帧又可由多层质地透明的图案和背景组成，同组镜头中各帧共用同一个背景。同时这一阶段还要求完成背景画面的草拟，规定前景诸角色的运动。把有关的要求分别通知相应的绘制人员。

(3) 声道记录：在有声道的情况下，在动画实际绘制之前，一般先在录音棚进行录音工作，然后再测定声道，仔细地进行合成工作。也就是说，需要使声道中的声音与画面上发出声音的动作完全吻合起来。

比如声道中录了开门的声音时（当然画面上也必须有开门的场景出场），动画制作人要使得画面上开门的动作发生的瞬间正好声道上发出开门的声音。只有完成了这一步的任务才能保证以后要产生的活动能与同时进行的对话及其它音响密切配合。为此，要把有关对话及分时信息记下来。

(4) 关键画面绘制：各声道的录音工作及其与画面的吻合工作完成了以后，就可以开始画面的处理工作了。为了能对动画的各个阶段的制作工作有较好的了解，有必要先了解一下动画中一幅完整的画面的制作方法。

一幅画面实际上是由若干张“膜”重叠起来，拍成照片得到的。所谓“膜”，指的是画有动画的透明塑料薄膜，一张一张的透明的塑料薄膜画有动画中的人物或人物的一部分，并重叠在一起组成一幅完整的画面。利用这种方法，可以将完整的画面分成若干部分分别来画。这样做的好处是可以减少工作量，节省时间.

比如，动画中要表示一个人站着说话，在制作动画时，只要改变口形和脸部的表情就可以了，这样做无疑是较为合理的。这样，画有身体各部分的薄膜就没有必要修改了。实际上几乎所有的动画在其制作过程中，都要利用这个方法。再比如,利用这种方法,就可以重复使用需要反复用到的背景图, 而不必反复制作背景图.

前面已经讲过，即让是很短的动画片，也需要画几千张薄膜。动画片制作费用的很大一部分都用于制作这些薄膜。一张一张的薄膜都要精心地用手工画出来并着色。由于影片的制作是有一定期限的，所以还需要较为熟练的人来画这些薄膜。这样一来动画片的制作费用就会很高了。

薄膜制作工作的最初阶段是画一些关键画面（keyframe），这些关键画面是用来控制各个画面的构成、出场人物的表情、各种动作的端点及各个重要情节的，对于影片的制作起着很重要的作用。关键画面之间的时间间隔比起故事板之间的时间间隔来要短很多。

这个阶段是动画绘制人员在纸上为前景活动的角色画出画面，一般来说只是画一些草图，由主笔动画师负责创作，限定镜头中动作的极限。绘制的同时，在曝光表格上还要指定分时信息、运动情况等。由于这里的工作量甚大，常常是若干个绘制人员同时进行，分别画不同的场面或不同的角色。

(5) 产生摄制表：经过上面分层按排，声道记录以及关键画面绘制等阶段，还同时综合形成了摄制表。这个表格记录着后面制片所需的各种信息。如各个层次的先后次序、每层的曝光次数、摄像机的运动、背景的运动或它们的合成、摄像方式的指示等。

(6) 中间画面绘制：在两个相邻的关键画面之间插入若干幅画面，使其能形成“动作序列”，这一般是辅助绘画人员的任务。相对于关键画面的制作, 中间画面的制作难度小一些，是依据关键画面而制作的, 细化和丰富关键画面无法确切表达的故事的全部内容。为了能使动画看起来很流畅，动画制作人就必须利用许多方法来生成中间画面.

(7) 检验：通过检查保证上述各部分均已适当地执行，一些需要反复使用的画面也已按摄制表要求准备妥当。然后对已经初步画好的各帧的画面快速地拍摄下来，再依次放映，以便了解制作的画面组成动画的动态效果，进行必要的修改。

(8) 复制及着色：在完成上述检查、试验工作, 确认画面的动态效果以后，就可以对前面已绘制的背景草图可以进一步加工提高质量, 并且按要求着上颜色.

对绘制的前景首先要转换到薄膜片基上，并且进一步勾划清楚。每一薄膜片的背面着上必要的颜色，使它们成为不透明的。如前所述，为了提高使用效率和节省投资，一幅画面往往可能是许多薄膜片迭合而成的。

(9) 拍摄：将背景以及按各层次放好的前景片，按摄制表上的说明合成完整的场景，拍摄记录在电影胶卷或录像带上。

(10) 后期编辑制作：进一步检查声音与景像配合是否一致，决定是否要对某些场景重新拍摄或者重新绘制等。

通常情况下，在这样的一个过程中，前面三个准备阶段约占总开销的23%，前景、背景的绘制与着色约占57%，而拍摄与后期编辑约占10%。因此，主要的任务还是绘制。

12.3 计算机动画技术及应用

在计算机上实现动画，其原理也是一样的，不过它并不是采用电影中的设备与技术，而是利用计算机程序的方法来实现动画的效果, 因此也引进了一些不同于人工制作动画的新方法,新观点。

计算机动画的研究与应用始于60年代，最初的计算机动画由程序实现, 主要应用于科学实验. 从第一部计算机辅助生成的动画片问世至今，计算机动画技术及其应用有了很大的进展，尤其是近几年来发展更是迅猛。

从二维计算机辅助动画制作到高质量、高真实感视觉效果的三维计算机动画的生成，从传统的卡通片动态画面的生成到基于物理的造型和物理的各种属性动态变化的控制，其应用领域也从动画片制作扩展到科学研究，如科学计算的可视化，各种微观物理现象的动态模拟，以及视觉模拟、娱乐、电视广告和片头、计算机辅助教学和训练等等各种领域。

12.3.1 计算机动画的概念

计算机动画的最直接的理解就是借助于计算机实现动画的技术. 从直观上来说，计算机动画技术是将图形、图案或画面的整个或一部分显示在屏幕上，并且按照一定的规律或者预定的要求在屏幕上移动、变换，从而使计算机显示出来的图形能动态地变化的一种技术, 是综合多个学科和技术的高新技术领域。它是以计算机图形学的三维立体造型、消隐、光照模型、表面质感等真实感图像生成技术等为基础的，

同时随着这一技术的发展，其涉及的研究领域和内容也在不断地扩展延伸, 需要诸如运动控制原理，图像处理技术、摄象技术，绘图艺术和广告艺术、计算机视频、音频合成技术等，甚至还涉及到视觉生理学、生物学、机器人学、人工智能、物理学和艺术领域的理论和方法。它综合了传统的绘画系统、运动控制、主框图动画路径设计、基于力学的动画、动画中的图像绘制等技术，近几年来，在动画语言和系统、适用于动画的专门硬件、科学可视化的动画、工程动画、基于人工智能的动画、多媒体系统中音像合成技术、人体造型及动画技术等方面有了很大的进展。

计算机动画属于空间加时间的四维空间的问题，而计算机图形学仅限于三维空间。因此，计算机动画是计算机图形学的综合运用，又由于其本身的特点而逐步形成了一个独立的研究领域。

从产生动画的基本原理来看，动画是计算机重复地进行图形变换操作的结果，在一般的情况下，可由程序来实现这种变换，先显示出一幅图形，然后对其施以变换，擦除原来的图形后，再将变换后的图形显示出来。如此循环，类似于动画片的制作或放映过程，所以，早期也有人直接将计算机动画称为计算机电影。在上述过程中施以变换或运动的图形部分称为动画对象。

12.3.2 计算机动画技术分类

计算机动画技术可以根据动画系统的目标分成两大类: 在人工制作的动画中帮助动画制作者完成一些工作和用计算机自身生成动画. 前一种方式计算机只是起到一个辅助制作的作用, 相应的技术和软件系统也就称为计算机辅助制作技术和软件系统. 后一类型的多用于制作三维空间的物体的具有真实感外形,相应的技术和软件系统也就称为计算机造型技术和软件系统.

12.3.2.1. 计算机辅助动画制作技术

计算机辅助动画也称之为主框图动画，主要是用计算机辅助人工的动画制作，这时计算机在动画制作中的作用体现在以下几个方面：

(1) 关键画面的输入：通常关键画面由数字化输入，背景画面由摄像或扫描输入，也可以使用图形编辑器或采用程序交互生成，如给出描述对象的各种参数，通过程序生成动画对象。

(2) 中间画面的自动生成：利用计算机自动插值生成中间画面，或通过交互式地给出动画对象的物理运动、对象大小、形状等的物理变换、虚拟摄像机的位置、取景范围等自动产生所需的动画画面。

(3) 着色：包括交互式计算机辅助着色或采用真实感图形生成技术由计算机自动成像。

(4) 拍摄作用：由计算机控制物理摄像机，或采用完全由程序处理的虚拟摄像机。

(5) 后期制作：计算机控制编辑音像使两者同步。

12.3.2.2. 计算机模型动画制作技术

为了说明计算机模型动画系统, 我们首先把通常意义上理解的计算机动画系统根据其功能的强弱分成五个等级：

第一级，只用于交互式地产生、绘制、存储、检索和修改画面。不考虑时间因素，设计者只使用图形编辑器。也就是说, 计算机只是帮助动画设计人员制作动画所需要的画面.

第二级，可以考虑时间因素，计算中间画面和沿指定路径移动的对象。

第三级，提供给动画设计人员对动画对象的操作手段，如平移、旋转等，也包括虚拟摄像机的相应操作。

第四级，提供定义动画中各类角色的手段。这些角色(比如人及其它现实的或虚拟的动物)具有自己的运动特色，它们的运动可以是受约束的，包括对其行为的约束和与其他动画对象之间的约束等。

第五级，系统具有智能化的特点, 可自行学习和自行扩充，随着不断地使用，系统的功能会日趋丰富, 使用起来显得更方便灵活。

模型动画主要是三维计算机动画，通常在上述第三级以上。而计算机辅助动画技术基本上属于上述第二级。

12.3.3 人工动画与计算机动画的比较

下表列出了人工动画与计算机动画制作过程的简单对比。

人工动画制作与计算机动画制作过程比较表

人工动画制作	计算机动画制作
设计角色动作关键画面	设计角色动作关键画面
薄膜片上制作黑白画面	在计算机屏幕上制作原画并上色，或用扫描仪输入角色的画面
绘制背景图像	背景图像扫描输入
绘制中间动作画面	制作中间帧画面并上色存储
在特技台上试拍	在监视器上放映并检查
放映并检查动作过程	在屏幕上编辑修改
修改	×
着色	×
在特技台上正式拍摄	录像
冲洗胶片	×

注: 表中的×表示不需要相应处理过程

与人工动画相比计算机动画的主要优点如下：

(1) 计算机动画系统对描线和上色操作方便简单、颜色一致、无裂开现象，上色的界线准确、不用晾干，不会串色，修改方便，也不因层次多而影响颜色质量，上色速度也很快，上色也不需要额外的费用。

(2) 不需要昂贵的薄膜片，不用买颜料、胶片和冲印底片，可直接输出到录象带上。

(3) 工艺环节少，可随时预演，发现问题即时改正，返工率低。

(4) 画面可任意拷贝、粘贴、旋转、放大缩小、移位、镜象，使画面制作简易方便，并可实现传统动画难以制作的特技效果。

总之，计算机动画与人工动画制作相比具有很大的优越性，可以节省投资和时间，提高工作效率。当然, 动画是一门艺术，高质量的计算机动画作品也必然依赖于艺术家的创作和动画美工人员的艺术发挥。计算机动画技术使具有美学基本原理的计算机动画制作人员更好的发挥他们的艺术创造力, 可以设计出更精美的二维和三维的动画作品。

12.4 计算机动画实现方式

计算机生成或实现动画的方式很多，但大体上主要有三种：帧方式、位图传输方式和实时方式。三种方式实现动画的原理不一样，因此实现的特点和使用的场合也不相同。这一节就来讨论这三种方法的基本实现思想和对它们的相互比较。

12.4.1 帧动画

帧方式是指由计算机生成动画中的每一帧画面，并记录下来。然后，根据不同的需要，电影可以按24帧 / 秒、PAL制式的电视按25 帧 / 秒、NTSC制式的电视按30 帧 / 秒的速度播映。

帧动画也称为全屏幕动画或页动画。动画程序建立许多全屏幕图像，并将每幅图像存入单独的页缓冲区中。当所有图像全部建立完之后，程序中的特定程序段将这些已经建立好的图形页以适当的顺序显示在屏幕上，以建立动画效果。图像可以绘制得很大，可占满整个图形页，并可利用大量的时间来建立。因此，帧动画通常用于复杂的、全投影的三维实体模型。三维模型的复杂程度只影响到相应画面的生成速度, 不影响生成后画面的现实速度. 因此无论三维模型多么复杂，帧动画都可以按令人满意的速度实现。

帧动画技术主要采用存储画面重放技术，将所定义的一系列经少量修改的动画图形存储在内存缓冲区中，然后根据需要和动画显示顺序将它们一一调出，送入指定位置显示，由于每一幅图都有差别，所以快速的重放，就形成动画效果，这一点和电影及动画片是完全相同的，所不同的是画面是存放在电影胶片上或是在计算机存储器上。若帧动画较大，可存于扩展内存或扩充内存中，将动画存入这些专用的存储区或从中取出动画都要在专用的软件驱动程序支持下进行。当然，也可以把帧动画存放在固定磁盘上(目前常见的是存放在光盘上)，以便产生更长的动画序列，帧动画的重放管理可以通过将每一帧图像从磁盘装入显示缓冲区来实现。

12.4.2 位图传输动画

位图传输方式是捕获或指定显示缓冲区中的一个映射区域的位图数据，然后移动到显示缓冲区的另一位置，从而得到动画效果。位图传输动画也称为块图形动画、部分屏幕动画或拱形动画等。位图指的是屏幕上显示出的在指定位置的一部分图形，可通过一定的手段从屏幕上获取，再通过变换在屏幕不同的位置输出，就产生了该部分图案的连续运行, 实现动画效果。

例如，在具有图形处理功能的$C++$语言中具有用于获取位图和恢复位图的函数

$$getimage(fname, x1, y1, x2, y2), \quad putimage(fname, x, y).$$

我们用$getimage()$函数将屏幕上一个由$(x1, y1)$、$(x2, y2)$ 定义的矩形区域内的图像采集下来，存储到文件$fname$中. 然后利用$putimage()$函数在屏幕的另一矩形区域内绘制出来，这一矩形区域以点(x, y)为其左下角角点。不断循环，修改其左下角角点就得到该位图块的运动效果。这两个函数还可用于实现图形的剪贴等修改图形操作，而不用关心屏幕的内存地址。在其它一些具有图形处理功能的计算机语言中也具有这两个函数.

在许多情况下，较大、较复杂的位图块传输动画序列可以通过使用空闲存储区并维护一个图形数组或相关的文件来实现。

12.4.3 实时动画

实时动画是指可实时绘制画面并直接在显示器上显示画面的计算机动画。帧动画和位图传输动画实现都需要在启动动画序列之前绘制并保存好所有的图形。这是实时动画与帧动画和位图传输动画的不同之处，实时动画是在动画过程中由计算机程序实时绘制图像的，因此与电影和动画片的原理也是不同的。

在实时动画过程中，CPU的时间划分为图像产生时间和图像显示时间两部分。而在前两种方式中，CPU可以只进行动画显示操作，因为所有图像已经存在于内存中。

实时动画也称为动态页转换动画，它至少需要两个图形页面(一个图形页面实际为计算机的特定的一块内存储区域, 由计算机自动管理)，其中一个页面称为主显示页，其余页面作为图形工作页。显示器总是显示主显示页面上的图形. 在主显示页面显示的同时，下一幅图形可产生并放置在工作页面上，主页面显示完后，再把工作页面切换成主页面，如此反复进行，动画程序在工作页面上绘制图形，显示器不断地显示主显示页面上的图像，每次旧图案都被新图案代替了，从而形成动画效果。

这一计算机的图形处理技术也称为虚屏技术，也就是说，为了快速显示复杂的图形图像，可以在现行内存中开辟一个或者多个图形数据区。这些数据区中数据存放的形式与图形数据在显示缓冲区中的存放形式完全一样，所以，有人也将它们称为虚拟显示缓冲区，或称虚屏。在图形和动画准备过程中将图形存放到虚屏，再将虚屏中的内容传输到实际的显示存储器中，通过这种虚屏和实屏的交换来实现动画。在虚屏和实屏上显示和绘制图形，以及相互交换、操作的程序过程等都可以作为公共子程序调用，也可以由各种高级语言调用。

采用虚屏技术的好处之一是显示器上显示的总是应该完整显示的画面, 而不是产生画面的过程中未完成的画面, 要知道在生成复杂的画面时这个过程是可以被观察到的; 另外一个好处是在虚屏上生成画面时不需要同时处理相应画面的显示问题, 可以加快画面的生成速度, 也可以采用并行处理技术, 实现显示和生成图形同时进行，并可同时生成多个画面, 加快画面的生成速度.

12.4.4 三种实现方式的比较

计算机生成或实现动画的有三种方式的特点为：

(1) 帧动画：占用内存多、重放快。

帧动画是最快速的动画。由于所有图像都已事先建立好，因此，在动画序列中只能使用这些已存在的图像。由于处理的是全屏幕图像，因此，需要大量的RAM或磁盘空间来存放图像。帧动画的优点是动画重放速度非常快，因为计算机只是从一个存储区向另一个存储区复制数据块。

(2) 位图传输动画：局部画面动画。

位图传输动画操作的大多是原画面的一部分, 是原画面的局部画面的动画. 位图传输动画不如帧动画速度快，但是，它节省内存，因为只有屏幕上一小块区域被操作。位图传输动画可以保护背景图案，只需使用位的异或(XOR)及和(AND)这样的逻辑操作即可。这种技术节省了重画背景图像所需的时间和内存。当然程序只能使用那些已经存到图形数组中的图像。

(3) 实时动画：牺牲速度得到多功能性。

实时动画是最灵活的计算机动画形式，但也是相对来说速度最慢的实现方法，这是由于在显示动画画面的同时，计算机还要同时绘制每一帧新画面的图像。也正因为如此，可以随时改变动画序列中下一帧图像的内容。实时动画技术是一种最具交互性的技术。

影响实时动画速度的因素主要是要实时生成的画面的复杂程度和生成画面的程序的及计算机的运行速度。随着计算机硬件速度的提高，实时动画也会达到一个令人满意的速度。但人们可能也会尝试用其生成更复杂、更富于想象力的动画画面，又制约了其速度，使得实时动画总显得不如希望的那样快。

在上述这三种实现计算机动画的方式中, 真正不同于人工制作动画技术的是第三种, 即实时动画方式, 实时动画必须借助于计算机才能实现; 而前两种方式是可以脱离计算机而存在的.

另外，这样三种动画的实现方法并非互相独立的方法。在一个动画系统中，往往将这三种方式任意组合，根据用户的需要进行混合和匹配使用。

12.5 动态设计与动态画面的生成

计算机动画技术中，特别是实时计算机动画技术中, 很重要的一部分内容就是所谓的动态设计。动态设计包括了对物体几何位置、大小、形状以及色彩等随时间而变化的规范描述。在人工的计算机动画中，这种动态变化过程是由动画设计者人为指定的，常用的方法有如下三种：

12.5.1. 运动路径控制法

运动路径控制法首先要求设想画面中有一个坐标系构成的树结构图, 完整的画面定义在根节点对应的世界坐标系中; 画面中的各个物体定义在根节点的子节点对应的坐标系中; 如果某个物体有多个部分构成, 则这个物体的各个部分又定义在其对应的子节点的子节点对应的坐标系中;重复这个过程, 就构成完整的物体, 完整的画面.

这样的画面中物体及物体各部分的运动和变化都是通过其所在的子坐标系相对其父坐标系的运动和变化来实现的. 需要指出的是画面中的物体不仅包括我

们通常认为的物体, 也包括形成完整的画面所需要而假设的物体, 如光源以及为了选取物体某个特定的侧面而设中的摄像机的位置. 它们也定义在相应的坐标系中,对应着坐标系构成的树结构图的某些节点. 光源的位置及光的强度、色彩影响着画面的视觉效果。摄像机的位置及方向影响着我们观察到物体的哪一个侧面。

用这种方法进行动画设计时要求动画设计者绘出物体、摄像机镜头、光源的运动路径和时间的指定。在坐标系构成的树结构图中,父结点坐标系的运动传递给子结点坐标系，子结点坐标系可相对父结点坐标系做各种变换，从而形成较为复杂的坐标系运动，带动定义在各个坐标系中的物体做出复杂的运动和变化. 运动路径的指定通常是先给出物体运动过程中的若干个关键状态点。然后，系统自动通过线性或样条插值来给出具体的路径轨迹。类似地，动画设计者给出时间控制点，从而控制物体运动的速度是需要加速、减速、或匀速。

摄像机镜头在某一时刻的运动状态可由镜头在三维空间中的位置、方向及张开的角度这三个条件确定。光源可根据其位置分为无限远和近距光源。无限远光源在某一时刻的运动状态可由光源的方向、色彩及强度表示，近距点光源则由光源的位置、方向、作用的角度及色彩和强度表示。如同物体运动路径的指定一样，镜头和光源的运动也由动画设计者通过给出运动中的关键状态后，由系统通过插值决定。

12.5.2. 关节动画法

关节动画法是基于逆向运动学的原理而实现的，适用于一些具有关节的物体如人、动物、机器人等的运动描述。一个具有关节动画功能的三维的动画软件应提供两种工具，其一是用来建立关节物体的关节结构, 其二是用来给出关节体的动态运动状态。

关节物体的关节结构是一种树形结构，其顶层为根结点，一般选为运行结点，根据关节物体各关节之间的相对关系，确定树结构. 利用动画软件提供的工具，自顶向下完成整个关节结构的构造，关节结构由关节点、关节连接体和链组成。一个链中包括有若干个相互关联的关节点及关节连接体。

一个关节点的运动将带动整个链的运动。每个关节点可以有6个自由度的变量$(r_x, r_y, r_z, t_x, t_y, t_z)$。其中，$r_x$, r_y, r_z分别表示绕x, y, z轴的旋转角度，t_x, t_y, t_z分别表示在x, y, z方向的位移。它们用来表示对这个关节点的约束。在结构定义完成后，将各关节部件连接到相应的关节连接体上。

给出关节体的动态运动状态就是要求设计者确定关节体运动过程中的若干关键状态，每个关键状态包括此时根结点的位置，各个链上某些关节点在此刻六个自由度的变量具体取值。系统软件据此推导出某个链上其余关节点的状态，从而确定此时刻关节体的运动变化状态，然后系统软件再通过两相邻关键状态的插值，推导出关节体的实际运动变化过程。

12.5.3. 变形动画法

在动画生成过程中，上述两种变化考虑的是一个物体位置和方向的变化。当物体的形状也发生变化时，需要采用新的变形动画方法来实现。假设变化前后物体的形状是已知的, 变形动画技术利用插值的方法来解决这个问题。物体由形状A变形到形状B可通过如下步骤完成：

(1) 在物体的形状A和形状B的造型中，要注意两物体的匹配问题，即构成形状A和形状B的基本图形之间应可建立一一对应关系，对应的基本图形的控制点间也应建立一一对应关系。

(2) 应能按照一定的规则，利用构成物体的形状A和形状B的控制点生成物体的形状A和形状B的图形。

(3) 利用动画软件提供的手段，进行线性或样条插值，在插值过程中加入时

间因子使变形过程可实现加速、减速或匀速控制。插值的目的是给出物体由形状A变形到形状B时形状A的控制点移动到形状B的控制点的轨迹, 由此给出物体由形状A变形到形状B 的过程中某个时刻控制点的位置, 生成相应的图形.

需要指出的是物体的形状A和形状B两者之间可以是没有任何关系的, 比如美女变老虎的动态变形.

12.5.4．物体物理机制动画法

物体物理机制动画法是按照物理的实际客观规律推导出物体的动态运动或变化过程的方法. 前面的三种方法实际上都是某种形式的基于前后关键画面的中间图形画面生成法, 所不同的是中间图形画面生成的方法.

基于物理机制的计算机动画方法与传统的计算机动画方法的区别在于其不需要人为地指定运动过程以生成中间画面. 物体可以有其自身物理的运动规律, 物体之间可以有相互约束关系，物体本身还可受到赋予某种特性的限制等。这时物体变化运动的过程是受这些指定的规则约束的结果. 在有的三维动画软件中，已增加了基于物理的功能计算机动画功能模块。具有图形显示过程的计算机仿真技术可以认为是这方面的一个实例.

当然，这四种方法可以综合运用。如关节物体的运动变化过程中，腿部关节与头部关节在描述自然的人的状态试就永远也不可能重叠在一起。按照物理机制的思路，就要限制这两个关节点处变量的取值。

12.6 动画技术中要注意的问题

在计算机动画技术中要处理大量的图形画面，因此需要占用大量的计算机的时间去生成这些画面,需要占用大量的计算机的内外存储空间去保存这些画面. 而动画的显示速度和计算机内存容量总是矛盾的，这两者也严重地影响和制约着动画显示的质量。动画显示的速度和内存容量既与计算机图形系统的硬件有关，也与图形软件功能有关，为了解决和协调这两个制约因素，需要考虑如下一些问题：

(1) 尽量减少生成图形的计算量

图形画面生成后在显示器上显示出来的时间是固定的，与画面本身的复杂程度无关. 我们所能改进的是生成图形的时间耗用, 而动画过程需要生成大量的图形画面. 因此图形连续显示瞬间的计算时间是一个很重要的参量，从软件方面考虑只有减少图形计算量才能加快图形生成的处理速度，使图形画面的动画处理效果更好，动画图形的连续性更好。要减少计算量，可以选择较快的算法，作为一般性的措施, 需要改进常规算法，尽量减少计算式中的乘法、除法、开方、乘方、三角函数等运算量较大的计算。利用循环、迭代和递归算法从前一值得到后一值而不需要重新计算。对特定的图形在满足要求的情况下要采用尽可能快速的简便算法, 而不是总要采用完善而复杂的方法.

(2) 减少图形闪烁

计算机动画质量的一个重要因素是动画图形画面的闪烁性，如果采用“画–擦除–再画”这种动画方式，会出现图形间断，产生图形闪烁现象，降低动画图形画面的真实感。我们可以利用帧动画的方式，提高图形显示的交替速度，减少图形间断时间，从而减少图形闪烁。

(3) 图形页面操作

图形页面操作与图形显示系统和图形数据在存储器中存放的格式有关，我们知道，显示图形在显示缓冲区中是以一定的格式存放的，在帧动画方式中需要向显示缓冲区送数据，就应该满足一定的格式，经变换后的数据也需要按这种格式填写，才能正常显示。如前面介绍, 图形页面操作技术具有提高图形生成速度, 加快图形画面的完整快速显示,从而也具有减少图形闪烁的作用。

(4) 图形的点阵操作

在动画技术中常用到对图形画面中像素点的操作运算，这种操作是利用了显示缓冲区中的信息与屏幕上对应显示图形的直接映射关系。这种方式可以很方便地利用已有的图形画面形成各种新的图形。

在计算机处理中，对显示缓冲区的字位最快速方便的操作就是逻辑操作，如“与”（AND）、“或”（OR）、“异或”（XOR）和“非”（NOT）等。假设图形画面是单色黑白图, 则每个像素点颜色取值为0或1，因此每一个二进制位代表一个像素点的颜色取值，对应显示缓冲区的一个字节中的一位。要进行处理操作的图形可以是单个像素点，也可以是整屏图形、或按一定方式指定的部分位图。那么根据逻辑操作的含义可知道：

① 图形与自身相“与”，图形保持不变；图形与背景即0相“与”，实现屏幕清除；图形与1相“与”，图形也保持不变；如果图形与另外一个图形相“与”，将产生两个图形的公共部分的点构成的图形，即两幅图形共有的图形点或重叠的图形点都显示在屏幕上，其它点图形将被擦除。

② 图形与自身相“或”，图形保持不变；图形与背景即0相“或”，也保持不变；图形与1相“或”，全屏显亮或构成彩色屏；如果图形与另外一个图形相“或”，将产生图形叠加，即两幅图形所有的图形点都显示在屏幕上。

③ 图形与自身相“异或”，清除图形；图形与背景即0相“异或”，图形保持不变；图形与1相“异或”，图形可形成反视频或者互补色；如果图形与另外一个图形相“异或”，将产生非共点图形，即两幅图形非重叠的图形点都显示在屏幕上，而重叠的图形点被抹掉。

特别是图形与自身相“异或”清除图形的功能非常有用, 这使得我们在放弃对图形的修改, 恢复原来的图形时只要简单的再绘制一遍要作废的图形就可以了.

12.7 计算机辅助卡通动画片制作

计算机动画有着很多的应用领域，计算机卡通动画片是其中之一。用计算机辅助制作卡通动画片，不仅能保证质量，而且还尽可能地降低成本，减少动画制作人员的劳动强度。下面简要叙述了计算机动画技术在卡通动画片制作中发挥作用的四个主要方面，卡通动画片基本上是由二维平面图形的画面组成的。

12.7.1. 画面输入

首先是画面的输入。在用计算机辅助制作卡通动画片时，卡通动画片中的许多基本角色的特征、姿态以及关键画面的绘制、背景的设计与绘制等任务，都仍是动画制作艺术家的任务，很难真正取代艺术家在这一方面的创造性劳动。因此，我们就需要充分地利用这些人工绘制的东西。达此目的的前提条件是把

它们转换成以计算机可以接受的形式，并存储在机器内。这一转换任务可以利用扫描仪或利用交互式的图形输入设备来完成。

此外，若计算机上还配有一定的模式识别及图像处理软件包的话，我们还可能以点、线、折线、多边形、各种曲线等方式来表示同一画面。两种不同的存储方式各有其优缺点。配合起来使用，会大大提高后期制作的效率。

在交互式图形输入设备中，画家甚至可以直接在显示器上作画，实时观察并利用交互软件随时修改所作的画，直至完成并存储起来。交互式图形输入的另一个好处就是方便动画艺术家着色处理。当然系统设计的好坏与用户是否能方便地操作、提高效率、是否愿意使用它等密切相关。为此目的,这类交互软件一般应当具有以下的功能：

1） 交互地指定各种特征点，如线段的端点、多边形的顶点、样条曲线的控制点等;

2） 丰富的拟合算法，使得用户能从给定的控制点得到具有一定光滑性并满足形状特性要求的曲线图形；

3） 各种常用的物体几何形状构成的形状库，这种库里既有如矩形、圆、椭圆等基本形状，又有能使用户生成的或其它拟合方法得到的各种经常要用的形状能比较方便的添加入库或生成新的库. 只有这样才能使用户减少绘制常用的物体形状的重复工作；

4） 具有建立各种类型画笔的功能，也就是说可以对画笔赋以不同的颜色、形状、模式等特性，并能用它来方便地画出各种基本图形，画图形时还应包括能消除阶梯状的处理；

5） 能对物体形状进行多种操作，例如复制或移动、各种几何变换、对指定物体的形状可以缩放，以进行更细致的观察等。

6） 区域填充及轮廓扫描转换功能，也就是说对指定边界的区域以选定的方式着色或填充，如选择不同的颜色及纹理模式。对动画片中连续的画面有一定的自动填色的功能而不必始终需要用户的参与, 这需要软件具有一定的自动分析出某些区域边界的功能.

7） 各种调整功能，主要是对由于连续图形的离散化而引起的各种不一致性的调整。

12.7.2. 动画质量的快速检查

当动画序列已经建立，动画设计人员一般要在初步画好的图形画面的基础上，大致确定动画片中主要角色的动作质量，这些初步画出的画面通常是线条形式的尚未着色的前景角色与背景。利用计算机，可以更方便地实现这一任务，常称之为预视（preview）。

利用预视功能，可实时地播放出初步画出的画面，以检查是否符合设计要求，其视觉效果是否满意，如果不满意，再进行修改。实时的检查需要有很快的处理速度，要占用很大的内存，因此一般能检查到数秒的动画片已是相当不错的了, 但随着计算机技术, 特别是硬件技术的快速发展, 这项工作将变得越来越容易。这种预视方法避免人工设计动画画面时拍摄等阶段的工作，达到快速检查的目的, 同时也节约了相应的投资。

12.7.3. 生产管理

卡通动画片的生产过程中，自然有大量的数据，这些数据不仅仅是图片，还有对各个制作生产环节的管理监督与计划的信息。在动画制作过程中的这些工作是非常繁琐但又非常细致的, 不能有丝毫差错. 利用计算机来辅助管理这些数

据，会使这项工作变得比较容易, 且符合上述要求。这样的一个辅助系统不仅应当能对动画制作过程进行控制与管理，特别对摄像机的运动、分镜头、动画序列可能的多次利用，摄制表的形成等加以控制，而且还应能在主要角色模型库管理、生产规划、成本核算及控制等方面发挥作用。

12.7.4. 中间画面的生成

中间画面的生成是指在动画绘制人员绘制并输入关键画面后，为了关键画面之间动作的连续协调而生成一些中间画面图形，一般希望计算机能自动地产生这些关键画面之间的各个中间画面。图形软件可以用插值或变换两种方法来完成这一自动处理任务，要注意的是由于这两种方法都依赖于各种几何图形的算法，因此要求关键画面应当以各种几何图形为元素来构成。

用插值方法获得中间画面的具体方法是多种多样的，大多包括以下三个方面的内容，第一，在相邻的两个关键画面上，确定要被插值形体之间的一一对应关系，并且选出每个元素中的若干个控制点，这些控制点将能决定几何形体在不同画面中的不同形状，即变形过程。第二，确定一种具体的插值方法，例如线性方法、样条曲线方法等等。最后，定义插值节奏或称为动画分时，即指定在不同的时刻，中间画面应当在已形成的插值路径上的哪个位置, 主要是指定关键画面之间的动画要持续的时间的长短。这是一个以动画制作人员长年积累起来的经验为基础的一个步骤，不同的分时方式，即使对同一几何元素同一路径，也会产生十分不同的动画效果。

用变换方法来得到中间画面也需要经过类似的几个步骤，第一，要把关键画面分解为个别的要作变换的基本几何形状，例如直线、曲线、多边形等。第二，定义对每个基本几何图形要作何种变换如旋转、拉伸压缩等以及与变换有关的参数值，决定是以固定的还是可变的方式来指定参数等，要决定变换轨迹或者说平移的参数。最后，定义插值节奏或动画分时的插值方法，这一点与上面提到的插值方法是一致的。

需要指出的是，中间画面生成这个任务的难度比人们想象的要困难得多，其主要原因是因为丢失了某些必要的信息。动画绘制人员所画的关键画面实际上是利用了物体形状的三维信息，通过绘制人员的想象作了二维投影以后才画出来的，这必然会丢失很多有用的信息。

例如假设输入的两幅关键画面分别是人的侧面和正面，在侧面的那幅关键画面中，只绘出了一只手，另一只手被身体遮挡了，正面的那幅画中绘出了两只手，在这两幅关键画面之间的中间画面，当绘制人员绘制的时侯，很容易判断当人转身到某一角度时在何处显露出另外一只手来。

但这一推断任务如果交给计算机来独立完成的话，单独由两幅二维图形就很难做出判断, 除非引进人工智能的方法，计算机依据有关的信息分析出图形画面中有人的图形, 再依据对人体形状的分析等才能做出必要的推断。如此复杂的分析解答这不是一般的方法所能解决的。

除了研究人工智能的方法解决上述问题外, 也可采用一些辅助性的具体技术或方法解决中间画面的生成。例如，让设计人员通过交互方式指出画面中丢失的信息；或利用角色的骨架形式作为输入和插值的基础；或给出足够多的关键画面以避免某些可能产生麻烦的位置或姿态等等。

但这些技术或方法仍很难令人满意，一是因为这些方法往往大幅增加设计时间，二是这些方法多多少少地限制了角色变化或运动的方式，甚至于禁止某些动作等，从而在一定程度上限制了设计人员艺术创作能力的自由发挥, 影响了动画画面的效果和设计人员的工作效率。

总之，计算机辅助卡通动画片制作已取得了相当多的进展，已有成部的计算机卡通动画片在研制和投产。但是，还需要进一步的努力，以期取得更多的突破，使计算机在这一领域成为越来越方便使用的工具。

12.8 相关问题

本章主要介绍了计算机动画技术的基本概念和一般的实现方法。计算机动画技术也可以分为二维和三维计算机动画技术，因此有些文献专门讨论二维或三维的计算机动画技术。但是实际上，二维动画技术并不比三维动画技术容易。计算机动画技术还有很多的内容并且正在蓬勃发展，应用领域越来越宽广，形成产品的计算机动画系统软件也越来越多。如3D Studio、Animator等等。

Autodesk 的3D Studio是一个具有实体造型、绘图与动画制作等功能的图形软件，它提供了多种动画制作方法，通过光线及相机的移动和修改可产生不寻常的视觉效果。另一个流行的图形系统是Autodesk公司推出的Animator 动画制作软件，它拥有数十种绘图工具如画笔、喷枪等，数十种颜料特性如透明、不透明、垂直及水平渐层等，而且每个画面可从262144种颜色中选取256色使用。这些软件都适合在微机上运行。

习　　题

1. 动画技术的基本原理是什么, 是如何实现的?
2. 计算机动画技术的基本原理是什么, 是如何实现的?
3. 请举例说明动态设计中关节动画法的应用。
4. 计算机动画有哪些实现方法？比较它们的优缺点.
5. 请叙述对一幅图形进行逻辑“非”操作的结果。
6. 什么是中间画面？它起什么作用？计算机自动产生中间画面比绘画人员困难的原因是什么?
7. 设计出从一个圆到一个正方形, 再到一个椭圆的动态变化过程的实现方法.
8. 假设一正方形物体受压而变薄, 变长, 但宽度和体积不变, 设计出这一动态变化过程的程序.
9. 请举例说明在一个图形连续小角度动态旋转中如何减少计算量以提高动态效果。

Bibliography

[1] 吴大任: 微分几何讲义. 人民教育出版社, 1982

[2] 唐荣锡, 彭群生, 汪嘉业, 计算机图形学教程. 科学出版社, 1994

[3] 沈伟烈, 计算机图形学教程. 航空工业出版社, 1995

[4] 胡瑞安, 胡纪阳, 徐树公, 分形的计算机图像及其应用. 中国铁道出版社, 1995

[5] 王莉, 李宗保, 吴志明, 计算机图形及其在工程中的应用. 人民交通出版社, 1992

[6] 王厚祥, 刘孟仁, 夏静, 交互式计算机图形学. 华中理工大学出版社, 1993

[7] 张法荣, 计算机辅助设计(CAD)教程. 中国大地出版社, 1998

[8] 任仲贵等, CAD/CAM原理. 清华大学出版社, 1991

[9] 李建平, 计算机图形学原理教程. 电子科技大学出版社, 1998

[10] 方奎等, 曲线取面设计技术与显示原理. 国防科技大学出版社, 1997

[11] John Corrigan(储留大译), 计算机图形奥秘及解答. 电子工业出版社, 1995

[12] 胡树根, 卓守鹏等, 计算机辅助绘图辅助设计辅助制造. 电子工业出版社, 1995

[13] 柳朝阳, 压入区段端点的区域填充扫描线算法. 计算机辅助设计与图形学学报, Vol.(8), No. 6

[14] 柳朝阳, Theory and application of convex curves and surfaces in CAGD. Press of University of Twente, 2001

[15] 施法中, CAGD & NURBS. 北京航空航天大学出版社, 1994

[16] 王洵, 计算机图形学基础。科学出版社, 2000

[17] Donald Hearn，M. Pauline Baker（蔡士杰，吴春榕，孙正兴等译）, 计算机图形学. 电子工业出版社, 2002