83750—2

'84 INTEL
微型计算机系统器件手册

(二)

上海交通大学 微机研究所
科技交流室

1986.2

本册译者：王建中、张光耀、赵正校、李慕靖、唐长钧、
谢康林、李治柱

总 审 校：徐子亮、杜毅仁

'84 INTEL 微型计算机系统器件手册(二)

目　录

iAPX86/20 数值处理机在三维图形显示中的应用

引　言

随着微型计算机性能的不断提高，其应用领域日益扩大，特别是在十六位微处理器问世之后，由于CPU性能的提高，尤其是引进了整数乘法指令，从而使得三维物体的图形显示所需执行的操作变得比较容易了。当然，因为硬件仅支持整数乘法，所以，用户只可以用一些整数值去定义图形。倘若系统提供了一些浮点数处理程序，那么，即便是对最简单的浮点操作，一般的通用微型计算机也都必须先调用这种子程序，显然，其速度将受到限制。

由于缺乏高性能的浮点运算功能，以及受到仅可使用整数表达方法的限制，这就严重地限制了能够定义的三维物体的规模和类型，难以想象，假如把整个宇宙空间中的所有事物的长、宽、高都限制在32000毫米以内，那么我们的世界将会成为什么样子，这一限制能够将那些用于模拟现实世界目标的系统集中于一个很有限的空间内，计算机辅助设计(CAD)系统就是这类应用的一个典型例子。但是，如果利用实数表示方法，那就几乎可以定义任何物体(一光年也不过9,397,728,000,000,000,000毫米，它一般总在浮点数的表示范围之内)。由于iAPX 86/20的问世，终于使得微型机能够满足三维图形的浮点表示的处理需要，这从经济上来讲是非常合算的。

iAPX 86/20具有Intel 8086 CPU及8087数值协处理器的特色，两者的结合可以完成高性能的、高精度的数值运算操作，而这在本应用报告给出的图形子程序中是特别重要的，因为对划出的每根线段，都需要进行大量的浮点运算操作，故对浮点运算速度的要求较高；此外，为了保证图形的显示质量，又要求有较高的数值表示精度。

本应用报告中给出了三维图形显示的一些基本的子程序段，正如前面指出的那样，若想以物体的实际大小来描述物体，就必须采用浮点表示方法。由于执行这些操作所需要的乘法和除法次数一般都比较多，故高性能的浮点运算部件就成为系统的必要条件了。值得注意的是，由这类软件完成的操作不同于所谓的"位图"控制器，完成这种特殊任务的单片设备不久将投入实用，由于它们是一些专用设备，故有可能比较快地执行这类任务，且有可能让通用的微型机并行地执行其它的工作。除此之外，由于位图控制器中可使用的存贮空间受到限制，故只需进行整数运算操作。本图形显示软件包由一类非常高级的子程序所组成，其输入是三维空间的画线命令。

本应用报告中给出的三维图形软件包允许输入三维图形，可对这些图形实施操作，也可设置观测者的位置及可见到的图片的大小，还可以设置图片在图形输出设备上的位置，此外，该软件包还将完成视图变换、窗口裁剪及投影变换。其中的所有图片都用浮点数定义，因此，任何图形都可用其实际规模来定义，而无需先取比例因子，也就是说，本软件包内定义的图形的大小可以是物体的实际尺寸，即目标的规模不受机器的限制，小至微观粒子，大至宇宙天体都可表示出来。

iAPX 86/20硬件概述

iAPX 86/20是一个以Intel 8086 CPU为基础的16位微型处理机。8086 CPU具有八个通用的16位内部寄存器，存贮结构采用分段方法，还具有其它一些能从高级语言编译程

序获得紧凑、高效代码的特色。在将它与8087数值协处理器结合起来时，它就成为一种进行高速数值处理的有力工具，8087在原8086的基础上，又增加了八个80位的内部浮点寄存器，以及一个浮点算逻部件(ALU)，从而使得浮点操作的速度与用软件方法实现的浮点处理仿真器相比提高了近100倍之多。

8086和8087执行的是同一个指令流，其中，8087专门负责处理数值指令，当正在译码的是一条数值指令时，8086就将为8087产生它所需要的存贮器地址，然后，8087将自动地开始执行该数值指令，此时，不再需要任何其它的软件接口，这是与现行的其它浮点处理器所不同的，因为在这些浮点处理器中，主处理器必须显式地将浮点数以及浮点命令送给浮点数处理部件。但在8086与8087所组成的系统当中，在8087开始执行某条数值指令之后，8086将继续执行非数值指令，直到它又碰到了一条8087指令，这时，8086就必须等待8087执行完前面的数值指令。8086与8087的这种并行工作方法被称为并发性，在理想的条件下，系统的吞吐能力是这两个处理器吞吐能力的总和，当然，即便在系统中不存在并发性时、8087的数值处理速度也要比单独的8086高得多。

8086与8087之间的硬件接口也同样非常简单，其间的硬件协议通过两组引脚即可实现。当8087需要传送操作数、状态信息或控制信息时，它就得使用RQ/GT引脚来实现与存贮器之间的信息交换，这是因为8087可以独立于8086去访问存贮器，它必须能够有机会成为"总线主"，即8087要能够获得所有地址、数据及状态线的读/写控制权。

还有一组引脚就是TEST/BUSY引脚时，它专门用于管理上面提到的并发性。具体工作原理是：当8087正在执行某条数值指令时，它将把BUSY引脚置为高电平，8086的一条WAIT指令将测试该引脚的状态，若该引脚处于高电平状态，WAIT指令将把8086置于踏步等待状态，直到BUSY引脚变为低电平时，等待过程才告结束。因此，为了确保在8087还在处理前面的数值指令时，8086不会取出另外的一条数值指令，就必须在绝大多数数值指令之前加上一条WAIT指令(只有那些访问8087的控制寄存器的指令之前无需加上WAIT指令)。除了所有的INTEL编译程序之外，80 86/87/88汇编程序也将自动地在大多数数值指令之前自动地插入这种WAIT指令。倘若不需要硬件交换协议，就可使用软件查询方法来确定BUSY引脚的状态。

8086和8087之间的大多数其它引线(地址线、状态线等等)都可直接相连，又有的一个例外是8087的中断引脚，该信号线必须送至某个外部中断控制器。图1给出了一个iAPX 86/20系统的例子。

除了8087硬件之外，8086还受到Intel的Pasca l及FORTRAN编译程序的有力支持，由这些编译程序所产生的代码可以很容易地与其它编译程序产生的代码联接起来，也能够容易地与Intel 8086/87/88宏汇编程序或IntelPL/M编译程序产生的代码连接起来。此外，在需要数值操作时，这些编译程序将为8087产生所需要的代码，而不是调用浮点子程序，这样，就去除了不必要的过程调用及返回所耗费的软件开销。

iAPX 86/20中的硬件协处理器与软件支持结合起来，使得系统的性能变得很高，并为更快速、更方便地开发软件产品提供了可能。

三维图形显示基础

三维图形显示软件包的关键在于：把物体在三维空间中的透视图转换为在一般图形输出设备上能够显示的二维形式，并且要求这一转换是精确的。为了满足这些要求，其图形

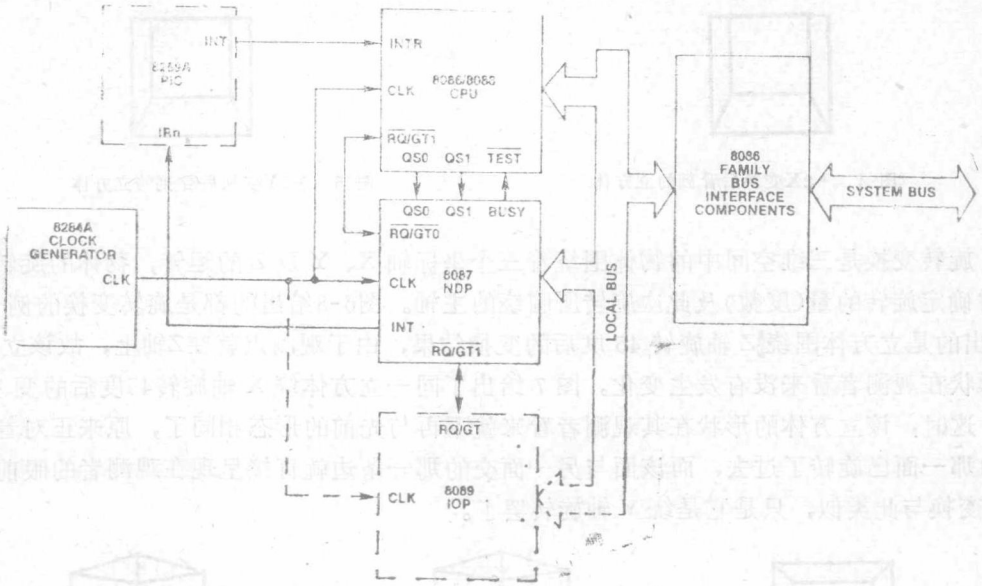

图 1　86/20系统结构示意图

显示软件包就必须：

·允许输入三维空间中的数据。因为显示软件包内的所有图形都是由三维空间中一系列的点所表示的，显然，系统中必须提供一种方式，以把这些图形输进计算机。

·完成适当的变换工作。显示软件包在三维空间中对三维目标图形的变换工作包括：旋转(rotate)、移动(translate)以及图形的放大或缩小(取比 例 scale)，图 2 到图 11 都 说 明了三维图形的这类变换操作形式。与三维空间的坐标定义类似，在所有的图形中，给出的第一个坐标为 X，第二坐标是 Y，第三个坐标为 Z。所谓观测点就是相对于事先任 意选定的坐标原点，观测者在三维空间中的位置。

移动(translation)变换是指物体在三维空间的移动，图3到图 5 是移动变换的 几 个 例子。具体讲，图 3 的变换是沿着 Z 轴增加的方向移动两个单位，由于观测 点 位于 Z 轴 上的 10 单位处，故这一移动变换就使该立方体与观测者之间的距离缩短了五分之一，也就是说，立方体好象变大了。图 4 说明同样的立方体沿X增加的方向移动两个单 位的变换情况，因为立方体的边长为 4，故这一移动变换就将使观测者往下正好看到立方 体 的 一 个面。与此类似，图 5 中的观测者往下也正好看到立方体的一个垂直面。

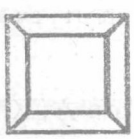

图 2　从(0, 0, 10)所看 到的以(0, 0,0)为中心的2×2×2的立方体

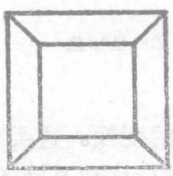

图 3　+2Z变换后看 到的立方体

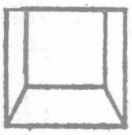

图 4　+2X变换后看到的立方体

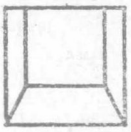

图 5　+2Y变换后看到的立方体

旋转变换是三维空间中的物体围绕着三个坐标轴 X、Y 及 Z 的运转，物体的旋转必须同时确定旋转的量(度数)及此次旋转所围绕的主轴。图6-8给出的都是旋转变换的例子，图6给出的是立方体围绕Z 轴旋转 45 度后的变换结果，由于观测点就在Z轴上，故该立方体的形状在观测者看来没有发生变化。图 7 给出了同一立方体绕 X 轴旋转45度后的变换结果，这时，该立方体的形状在其观测者看来就不再与先前的形态相同了，原来正对着观测者的那一面已旋转了过去，而该面与另一面交的那一条边就直接呈现在观测者的眼前。图 8 的变换与此类似，只是它是绕 Y 轴旋转罢了。

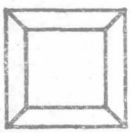

图 6　绕Z轴旋转45度

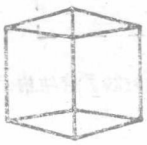

图 7　绕X轴旋转45度

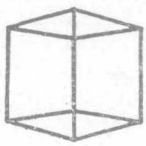

图 8　绕Y轴旋转45度

图形的所谓放大缩小(取比例尺)是指，用一常数去与定义某个图形的点的所有坐标相乘，以使目标图形显得更大或更小了，图 9-11 给出了几个取比例尺的例子。当然，这种取比例尺的操作不需要是物体的所有方向都作同样的变换，例如，若对立方体的所有Z 坐标取比例尺，使其Z坐标值两倍于原来的值，就将形成图9这样的结果，应该注意，在此，X及Y 的坐标没有发生任何变化，只有 Z 坐标是原坐标的两倍，也就是说，这一变换相当于使立方体的正面更靠近观测者，而背面更远离观测者了。图 10 的变换与此相似，只是这一操作是对X轴执行的，而图 11 则是对 Y 轴进行类似变换的结果。

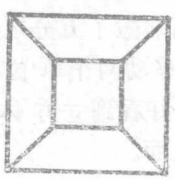

图 9　2×Z的坐标值

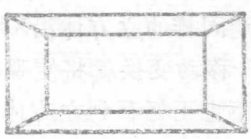

图 10　2×X的坐标值

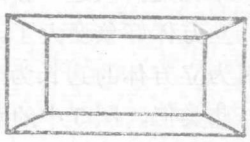

图 11　2×Y的坐标值

·进行观测点的变换。这类变换是根据观测点在空间中的位置和方位(观测者注视的方向)，来对图形进行移动及旋转之类的变换。图 12 给出的例子是说明观测点发生变化时，对三维图形产生的影响，再强调一遍，观测位置或观测点是观测者相对于事先任意选定的坐标原点的位置。

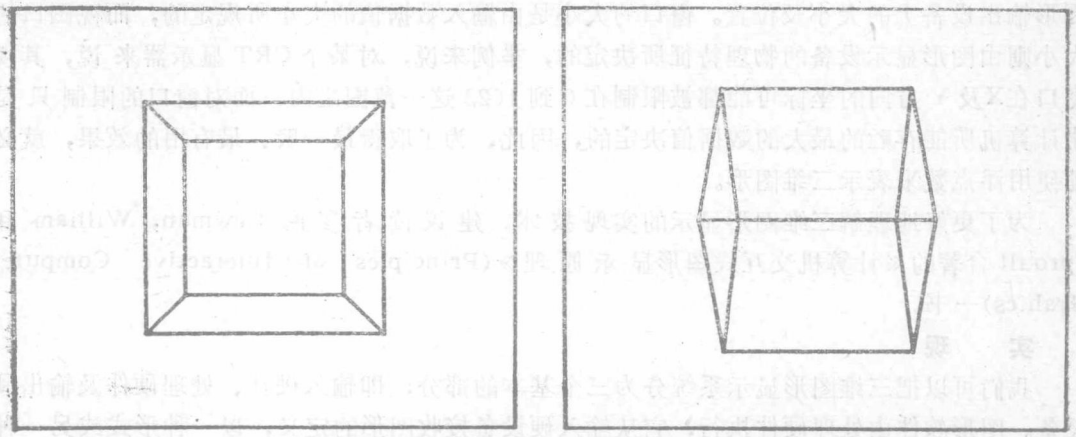

图 12　以(0，0，0)为中心的2×2×2的立方体先从(0，0，10)处看，再从(10，10，0)处看

·对三维数据执行Z裁剪。这一变换可以确保仅有那些位于观测者眼前的数据才在图上显示出来，此外，它也允许离观测者"太远"(超过一定的距离)的那些物体不在图上显示出来。

·将三维空间中的数据投影到一个二维平面上。物体必须根据透视图的规律投影到二维平面，通过不断改变那些逐渐消失的点("vanishing point")。也可以获得很有趣的效果。图13是投影操作的一个例子，其中，第一个图是放大了的一个透视图(即立方体正面与背面观测面之间的差被放大了)，而第二个图给出的是同一物体的调和透视图(观测到的立方体正面与背面大小之间的差距比头一个图小多了)，当物体离观测者较近时，就形成了夸大透视图；而当物体离观测者较远时，也就得到了所谓的调和透视图。值得注意的是，对于同样的一个图，从相同的方向，却得到了两个不同的透视图。

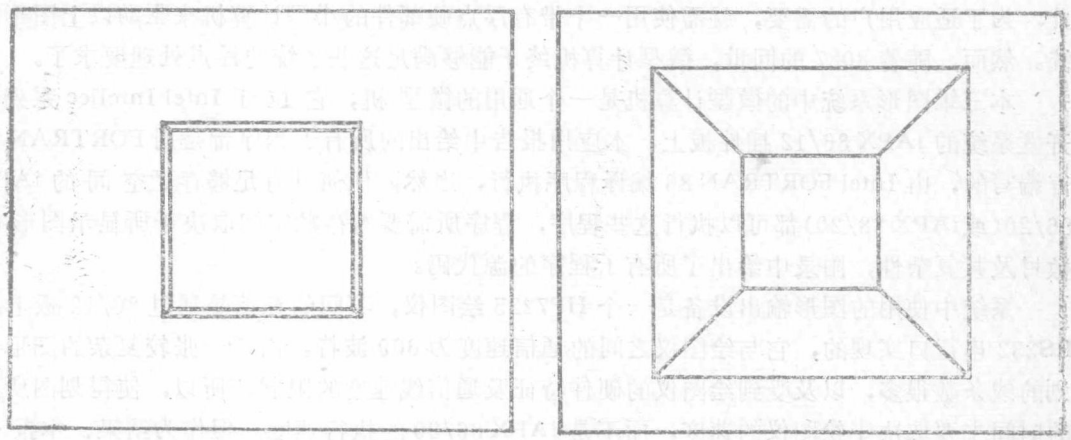

图 13　夸大透视图与调和透视图

·对投影数据进行X-Y裁剪。这一操作剪去了位于给定"窗口"之外的投影数据上的那些线段。

·完成窗口(window)到视图口(viewport)的变换。它根据"窗口"与"视图口"的相对大小，取二维投影值，并取适当的比例因子。

这里为"窗口"描述了观测者所看到的数据区域的大小，而"视图口"则描述了该区域在

图形输出设备上的大小及位置。窗口的大小是由输入数据值的大小所规定的，而视图口的大小则由图形显示设备的物理特征所决定的，举例来说，对某个CRT显示器来说，其视图口在X及Y方向的坐标可能都被限制在0到1023这一范围之内，而对窗口的限制只是由计算机所能存贮的最大的数据值决定的，因此，为了取得最一般、最有用的效果，就必须使用浮点数来表示三维图形。

为了更好地理解三维图形显示的实现技术，建议读者参阅Newman, William和Sproull合著的《计算机交互式图形显示原理》(Principles of Interactive Computer Grahics)一书。

实　现

我们可以把三维图形显示系统分为三个基本的部分，即输入硬件、处理硬件及输出硬设备。图形软件由处理硬件执行，它从输入硬设备接收图形的定义，以一种形式或另一种形式将图形存贮起来，并对之进行处理，以使其能在输出硬设备上显示出来。

输入硬设备可以是通用的键盘，也可以是复杂的三维输入设备；输出硬设备可以是绘图仪、存储管终端，也可以是位映射光栅扫描显示器或向量绘图CRT。

处理硬设备可以是通用的小型计算机，也可以是速度很快的专用图形处理硬设备。我们之所以使用一些通用计算机，主要是因为考虑到这样可以用高级语言来编制应用程序。在需要十分高的处理速度时，就要使用一些专用的硬设备，例如在飞行模拟器的实时图形显示应用中，就存在着这一需要，这类专用硬设备能完成完整的矩阵变换。当然，有许多应用并不需要实时地显示图形（每1/30秒显示一幅画面），对这些应用来讲，利用通用计算机就可以解决问题，一种典型的应用就是计算机辅助设计(CAD)系统。但是，也应该看到，由于这些图形显示系统大都用于交互环境当中，故若对简单图形的处理延时就达几秒，而对非常复杂图形的处理可能要花几分钟，这对交互用户来讲显然是不能接受的，因此，为了适应用户的需要，就需使用一个带有浮点硬部件的小型计算机来驱动以上图形系统。然而，随着8087的问世，微型计算机终于能够满足这些系统的浮点处理要求了。

本三维图形系统中的微型计算机是一个通用的微型机，它位于Intel Intellec系列Ⅲ开发系统的iAPX 86/12插件板上，本应用报告中给出的所有子程序都是用FORTRAN语言编写的，由Intel FORTRAN 86编译程序执行，当然，任何具有足够存贮空间的iAPX 86/20(或iAPX 88/20)都可以执行这些程序，程序所需要的存贮空间取决于所显示图形的数目及其复杂性，附录中给出了所有子程序的源代码。

系统中使用的图形输出设备是一个HP7225绘图仪，其间的通信是通过86/12板上的RS232串行口实现的，它与绘图仪之间的通信速度为600波特。由于一张较复杂的图形要划的线条数很多，以及受到绘图仪的硬件特征及通信线速度的限制，所以，使得划图所花的时间主要取决于绘图仪的速度，而不是iAPX 86/20的执行速度。但作为结果，本报告中给出的时间不反映绘图时间，而只到将ASCII字符放至串行通信片的缓冲区为止，为了更充分地发挥iAPX 86/20的优点，可以采用速度更高的显示设备。

系统采用的图形输入设备是附在开发系统上的标准字母数字键盘，这就允许用它来输入图形及该图形系统的控制信号。当然，为了在定义较大图形时获得更高的速度也可以从存贮器(磁盘)取得输入信息。整个系统的硬件结构框图如图14所示。

所有子程序都可以用8087及8087软件仿真器来运行，8087软件仿真器是一个软件

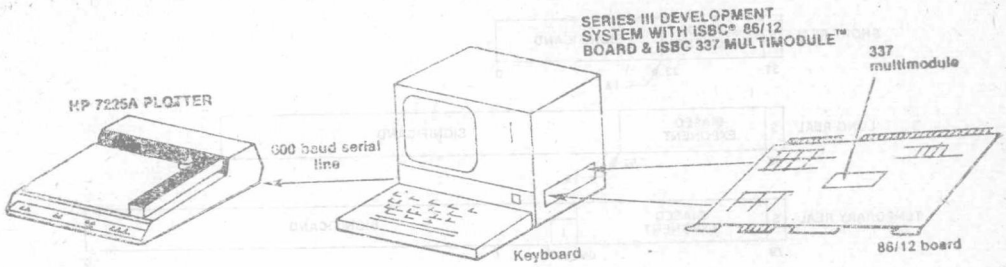

图 14　该图形系统的计算机系统

包，它能通过利用 8086 的指令来精确地模拟 8087 的内部操作，该软件包可 从 Intel 公司作为 8087 支持库的一个部分而获得，但 8087 硬件的性能要比其软件仿真 器 高 得 多，当然，作为专用的一个硬浮点处理部件，这也是起码的要求。

8087 可以支持多种数据格式类型，单就实型数来讲，就有短实数(单精 度)、长 实 数(双精度)及临时实数(扩大的精度)之分，这三种实数类型之间的区别仅仅 在 于，为 了 表示某个给定的浮点数，所需要的位数是不相同的。

在所有的实数当中，数据都被分为三个字段，即符号位、指数字段及尾数字段，符号位用于标志数据的正负，指数字段与尾数字段一起确定了数值的大小，具体讲，指数字段的内容为该数值的二的幂次方，而尾数字段的内容则是该数的规格化值。

所谓"规格化数"是一种在规定范围内取值的数。通过用某个二的幂去除一个数，在绝大多数情况下，任一数值除以某个二的幂次方，总可以将它调整到 1 到 2 的范围内，这里作为除数的二的幂次方就是该数值(被除数)的指数部分，而除法操作所得到的结果就是该数值的尾数部分。当然，这种操作并非对所有的数值都能适用(例如，用任一数值去除零，得到的结果总是零)，因此，数值系统必须对一些特殊情况进行处理。

随着指数部分所占位数的增加，其数值的表示范围也将相应增大，也就是说，它能够表示出一些更大或更小的数值来。当尾数部分的位数增多时，其数 值 的 表 示 精 度 就 将随之增高，这就意味着两个相邻数值之间的差距缩小了，换句话说，即可更为精确地表达数值了。对于短实数来讲，其指数部分有八位，尾数部分有 23 位；长实数格式的指 数字段由 11 位组成，尾数部分有 52 位；而临时实数则具有 15 位的指数部分和 64 位数的尾数部分。这些数据格式如图15所示。由此可见，在以上的三种实数数据格式中，短实数的表示精度最低，临时实数的表示精度最高。其实，精度级的这种区别只有在数据的存贮格式中才真正反映出来，而在 8087 的内部，所有数据都是以临时实数格式表示的。在把数 据 装入 8087 时，8087 将自动地将这些数据转换为临时实数格式；而在将 数 据 从 8087 写入存贮器时，8087 又将自动地把数据从临时实数转换为指定的实数格式，除了实型 数 之 外，对于 16 位，32 位或 64 位整数以及80位的二进制编码的十进制数(BCD)，8087 也将进 行类似的转换。

随着数值表示精度的提高，所需要占用的存贮空间也将相应增大。由上面的介绍可以看出，一个短实数仅仅需要占用四个存贮器字节 (32位)，每个长实数需要占用八个字节，而每个临时实数则需占用十个字节(80 位)之多。在许多浮点处理器中，其处理 时 间 也将随着数值表示精度的提高而动态地增加，但是，8087当中，处理短实数、长实数及临时实

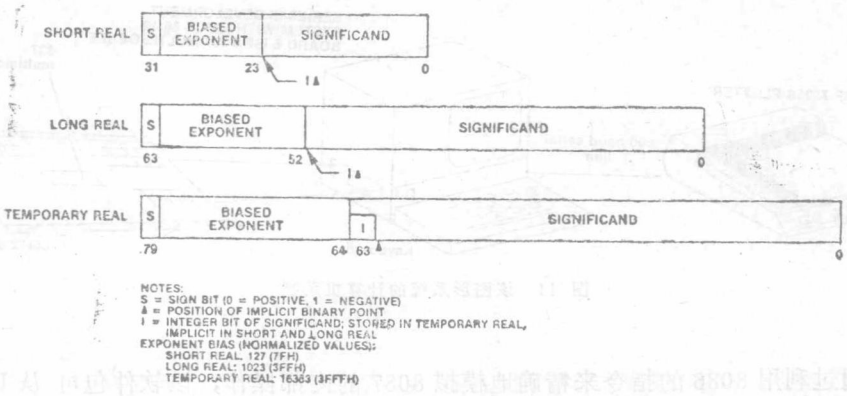

图 15 浮点数据格式

数所需时间之间的差异是微乎其微的。因此，在 iAPX 86/20 系统当中，选择数值表示精度时要考虑的主要因素仅仅是存贮空间的限制及对精度的要求。

本图形显示系统选择的是双精度实数，这是因为考虑到这种数据格式的取值范围较大，且具有较高的数值表示精度，这一点是非常重要的，因为本软件包允许用户将图形的某一很小的部分放大，在这种情况下，若没有达到双精度数的表示精度，显示的结果肯定是难以令人满意的。

三维图形描述及用户接口

本报告中给出的图形用户接口是比较基本简单的，它不需要使用任何专用的三维图形输入硬设备，所有输入数据都是通过键盘键入的。

该软件包允许在其图形包内定义图形，以供将来使用，在要求产生某个物体的多个视图时，这一特色就相当有用，它要求开始就定义该物体，然后，允许用户从任一位置出发观察此物体，并可进行旋转、取比例尺或移动等变换操作。

图形显示包的命令由一组字母数字命令，后跟一些必需的数值常量组成。若要给图形显示包输入命令，则只要输入用单引号括起来的一条字母数字命令，后跟适当的数值参数。这种命令所需要的参数数目不会超过六个，如果在同一行中输入了不到六个参数，那么该行必须以字符'/'结束，之所以存在这些要求(用单引号把命令括起来及显式指出一行的结束)，是因为使用了 FORTRAN 的直接列表输入格式的缘故。

本图形显示处理器可以识别以下命令：

comment arg1：该命令指示图形显示处理器略去后面的arg1行，利用它可在图形命令内插入一些注释信息。

define arg1：本命令通知图形处理器后N行(直到 enddef 命令)应送至内部缓冲区，以供将来用于图 arg1 当中。这些图形命令在此不需解释，也就是说，它们不会在此把图形划出来。利用这种方法，就可以定义三维物体，并可将它们放至内部显示表之中。利用现有的程序版本，最多可以定义十个三维物体，当然，这一数目也可以增加，但不可超过存贮空间的限制。当前，系统中有一个内部存贮器空间，它可以存放多达 500 条的图形命令，将这些命令适当地组织起来，一般可以定义十个图形。显然，当内部存贮器空间大小发生变化时，可定义的图形数目也将随之改变。

cnddef：本命令结束一次图形定义，并把控制返回到主图形显示处理机。

call arg1：该命令将使图形处理机从内部缓冲区取出一些图形命令，这些命令是先前某个时候用来定义图形 arg1 的。

line arg1 arg2 arg3 arg4 arg5 arg6：这是一条在三维空间划线的命令，即划一条从点 arg1，arg2，arg3)到点 (arg4，arg5，arg6) 的直线，当前目标的旋转，取比例尺、移动，及观测者的位置、窗口、视图口等概念都可使用。

plot arg1 arg2 arg3 arg4：该命令也执行划线功能，只不过这时它是从上条线的末端出发，使用"笔码"(pencode)arg4，划到点(arg1,arg2，arg3)，目前,系统支持两种笔码，一为'2'，表示划实线；一为'3'，表示不划线，即仅用来改变画笔的位置。当然，还能够定义其它一些笔码，以实现划虚线、点线等功能。

ident：本命令将把"当前"矩阵置为等阵，即把所有旋转置为零，把所有移动置至始点，而把比例因子置为 1。

push：该命令将把当前矩阵压入矩阵堆栈中，而当前矩阵并不发生变化。

pop：本命令执行弹出矩阵堆栈操作，弹出的信息送至当前矩阵中。

rotate arg1 arg2 arg3：该命令引起观测者看到的三维图形旋转一定的角度，它绕 X 轴、Y轴、Z 抽的旋转角度分别由 arg1、arg2、arg3 这三个参数确定，其单位为度。执行该命令不会引起对目标定义的改变。

translate arg1 arg2 arg3：此命令将引起观测者看到的三维图形移动一定的距离，它沿 X、Y、Z 方向移动的距离分别为由参数 arg1、arg2、arg3 所确定。同样，目标的定义也不发生变化。

scale arg1 arg2 arg3:本命令将使得观测者看到的三维图形的大小发生变化，它在 X、Y、Z方向所取的比例因子分别由参数 arg1、arg2、arg3 所确定。

window arg1 arg2：该命令设立窗口参数，这些参数确定了投影图形的可视区域部分，它把一个无穷高的锥体放在三维空间中，而观察点定在锥体的顶点处(从上往下看)。位于该锥体内部的所有物体都可看到，而位于此锥体之外的所有物体就见不到了。

viewport arg1 arg2 arg3 arg4：本命令设置可视口参数，这些参数决定了以上的窗口在绘图设备表面上的大小及位置,该区域在绘图设备表面上的中心由参数 arg1、arg2 确定，至于参数 arg3、arg4 则分别规定了此区域在X及Y方向尺寸的一半。这里，我们假定绘图设备在 X 方向的变化范围是 0 到 12 之间，在 Y 方向的变化范围为0到10之间。通过一个较低级的绘图设备接口子程序，可以进行一些变换，以将图形调整到绘图设备所能够识别的区域范围之中，由于是在较低的软件级上执行的上述操作，故使该软件包变得更为通用。

viewpoint arg1 arg2 arg3 arg4 arg5 arg6：此命令建立所谓的"观测"变换，其中，参数 arg1、arg2、arg3 表示观测者在三维空间中的位置，而 arg4、arg5、arg6 则表示被观测目标在三维空间中的位置，两者结合起来就形成了一个向量，它指向欲观测的区域。

zclip arg1 arg2：本命令设置Z方向的裁剪参数，这些参数确定了观测者前面的可视距离，其中，arg1确定了观测区域的最近边界，而 arg2 则规定了该区域的最远边

界。若把该命令与窗口命令(window)结合起来使用，则可以把可视物体与不可见物体很明显地区分开来。

cube arg1 arg2 arg3 arg4 arg5 arg6：这是一条划立方体的命令，该立方体的中心点在(arg1，arg2，arg3)处，而参数arg4、arg5、arg6则分别规定了此立方体在三个方向的宽度之半(长、宽、高的一半)。

arrow：本命令从(0，0，0)往(1，0，0)画一个箭头。

pyramid arg1 arg2 arg3 arg4 arg5 arg6：该命令会画出一个四面的锥体，该锥体的底部中心位于点(arg1，arg2，arg3)处，其半宽分别为arg4、arg5、arg6，其中，X方向的半宽arg4用作此锥体的高度。

current：本命令在终端上打印出当前矩阵。

printdef：本命令打印给定图形的定义。

startt：本命令启动iSBC 86/12单板机上的10mS时钟。

readt：该命令中止iSBC 86/12单板机上的10mS时钟，并在终端上打印出10mS计数。

end：此命令将中止本图形显示软件包的执行，并在终端上打印出画出的点数及"success!!!"，最后，把控制权归还给ISIS。

图形包的内部操作

本软件包中的所有内部操作都用到1×4或4×4的双精度实数矩阵，每个点都用1×4的双精度实数向量表示，其中，前三个值分别代表某点的X、Y、Z坐标值，第四个元素的值总为1，当将该点投影到某个二维平面上时，就将用到这一元素。一般说来，大多数子程序所完成的任务可从相应的子程序名上看出来(FORTRAN规定子程序名字的长度不能超过六)。

当前变换

倘若每执行一次移动、旋转或放大缩小变换，都要对物体进行修改的话，则图形包的性能肯定是相当低劣的，除此之外，它还将使得该图形的原始定义消失。假如存在一种方法，它既能一次完成以上三种类型的变换操作，又能保持某物体的原始定义不变，则将提高图形包的性能及容易使用程度。

能把这些操作结合起来的一种方法就是通过使用所谓的"当前"矩阵，这里，当前矩阵是一个4×4的双精度实数矩阵，它从数值的角度反映了任意顺序的旋转、移动及取比例尺变换的结合情况。在为各图形点定位时，应该用该矩阵与各点的1×4定义向量相乘，乘法操作的结果就是经过旋转、移动、取比例尺诸变换后所形成的各点的新位置。若这一矩阵是一恒等矩阵，则表示各点的坐标位置不发生变化，也就是说，恒等矩阵表示不要进行旋转、移动或放大缩小变换。以上介绍的乘法操作是在子程序pline的第20和21行中完成的。

当碰到某个旋转、取比例尺或移动变换命令时，当前矩阵就将与另一个仅反映这一变换的4×4矩阵相乘。但应注意，由于矩阵乘法不具有可交换这一特色，故乘法操作的执行顺序必须与变换操作的先后次序相一致，这一点非常之重要，因为我们知道，先移动后旋转与先旋转后移动所产生的影响是不一样的，这是由于所有的旋转操作都是围绕原点这一支点进行的这一缘故(参见图16)。开始，我们总将当前矩阵设置为恒等矩阵，以后的所有操作都将以当前矩阵作为出发点。

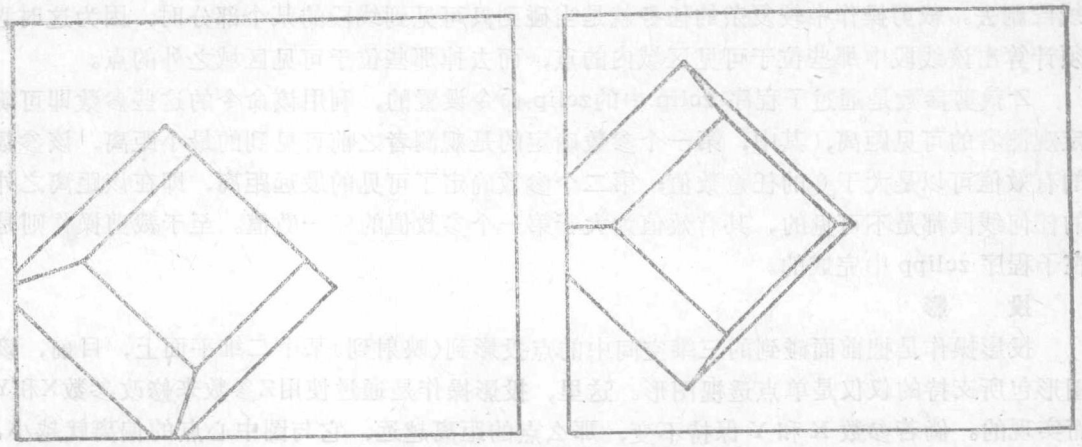

图 16　从点(0，0，10)处看到的立方体
左图：先旋转后移动　右图：先移动后旋转

当前矩阵中的参数值是通过 rotate、scale、translate、ident、push、pop 诸操作建立起来的，这些英文名字说明了它们所执行的操作功能，完成这些任务的子程序分别是 rotate、scale、transl、ident、push 及 pop。因为有了 Ident，故允许把所有的旋转及移动置为 0，而把所有的比例因子置为 1。引入push 与 pop 操作的主要目的在于，使得各图形可以将当前矩阵的状态保护起来，以防后继操作改变当前矩阵的内容，当把一个较大的图形定义为许多部分时，这种操作就显得相当重要。

观测变换

在绘制某一物体的图形之前，必须先知道观测者的位置(观测点)，即要求确定观测者在三维空间中的位置，以及他的观测方向。把这些信息结合起来形成了所谓的"观测"矩阵，它也是一个4×4的矩阵，实际上，它相当于对该物体所进行的另一种旋转变换，以使物体能从某个适当的角度被观测。因此，在所有的图形点都通过了当前矩阵之后，还将要通过这一观测矩阵。通过这两次变换，就将得到一组点，由这组点就可画出物体的三维视图，因为这些点是经过在当前矩阵上实现的旋转、移动、放大缩小变换，及在观测矩阵上进行了适当的旋转变换之后所得到的结果，显然，它们应该是目标图形点。

观测矩阵是由 viewpoint(观测点)命令建立起来的，该命令将在观测矩阵中放入适当的旋转变换系数，以使目标图形能够正确地反映用户的三维视图。完成以上任务的子程序是 viewpn。

Z 裁　剪

由上面的讨论知道，经过当前矩阵及观测矩阵的所有图形点，都将在三维空间的适当位置处被定位。然而，对观测者来讲，并非能看到所有的图形点，他能见到的只是该空间中的某个部分，比如说，位于观测者背后的那些物体他是看不到的，但是，物体的每一点（包括那些位于观测者背后的点）都被映射到了观测空间之中。为了解决这一问题，我们通过所谓的Z裁剪操作，就可以将这些"不可见的点"从图中删去。其实观察方法也很简单，即通过检查各点的 Z 参数值，看它是位于观测者之前还是背后，从而决定是否应画出这一点。此外，人们有时也不希望显示那些离观测者太远的线段，故可利用类似的方法将这些

线段删去。裁剪操作中较复杂的任务就是当碰到只可见到线段的某个部分时，因为这时必须计算出该线段中那些位于可见区域内的点，而去掉那些位于可见区域之外的点。

Z裁剪参数是通过子程序 zclip 中的zclip 命令设置的，利用该命令的这些参数即可确定观测者的可见距离，其中，第一个参数确定的是观测者之前可见到的最小距离，该参数的有效值可以是大于 0 的任意数值；第二个参数确定了可见的最远距离，即在该距离之外的任何线段都是不可见的，其有效值为大于第一个参数值的任一数值。至于裁剪操作则是在子程序 zclipp 中完成的。

投　影

投影操作是把前面碰到的三维空间中的点投影到(映射到)某个二维平面上，目前，该图形包所支持的仅仅是单点透视图形。这里，投影操作是通过使用Z参数来修改参数X和Y而实现的。倘若参数 X 和 Y 保持不变，那么点的距离越远，它与图中心点的偏离就越小，大多数人也都熟悉这个道理。举例来说，若你在往下看一条铁道，尽管事实上两根铁轨之间的距离保持不变，然而，在你看来，两根铁轨在远处好象会聚于同一点（参见 图 17）。

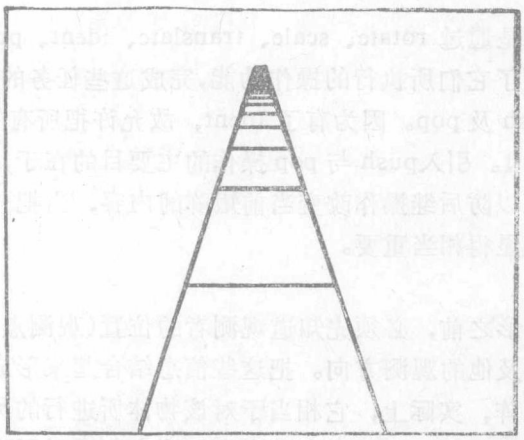

图 17　两根铁轨将逐渐消失

这样的一种投影操作在图形显示包中，是通过用一个 4×4 的"投影"矩阵与 1×4 的点位置向量相乘来实现的，该矩阵很象一个恒等矩阵，只是在矩阵中的(3，4)位置处放的是视图值。

该值是由观测点参数值计算出来的，在执行完矩阵乘法操作之后，1×4 的点定义向量中发生变化的元素仅是最后的那个(其值假定为 1)，即在乘法操作完成后，该位置中的值代表的是改变量，它是为了体现投影操作的影响，而在点向量的 X 及 Y 参数上所应执行的修正，当该向量是一"规范化"向量时，则将使用单点视图规则将点投影。这里的规范化是通过用向量的最后一个元素去除向量中的每一个元素而获得的，因此，原向量的Z 分量已经改变了 X 和 Y 分量。若需要两点或三点视图，则必须做的工作仅仅是把视图 值放到投影矩阵的位置(1，4)和(2,4)处，而其余工作都是完全相同的。执行这些操作的子 程序是 Viewpn(建立透视图的消失点)、projct(建立投影矩阵，并执行透视图乘法)和 norm (规范化该点向量)。

X-Y 裁剪

一旦图形的数据点投影到了某个二维平面上，就必须进行X-Y裁剪操作，当然，该裁剪操作也可以对三维空间中的数据进行，但若把该动作推迟到投影操作之后，则需要的计算工作将比较简单。这一点对Z裁剪就不再成立了，因为一旦把数据投影到某个平面，其Z参数就不再是它原来的样子了。

通过将参数X及Y与窗口值进行比较，即可完成X-Y裁剪任务，其中的窗口值是由窗口命令(window)所设定的。当然，这里的比较操作要比Z裁剪中所要求进行的比较操作稍微复杂一些，因为它要包括两个裁剪参数。总起来讲，对每根线段的两个端点来说，它们可能位于九个不同的区域之中。例如，这些区域可能是：在X和Y的窗口区域内、小于X的窗口区域边界但在Y的区域边界内、小于X的窗口区域边界且小于Y的窗口区域边界等等。假如线段的一个或两个端点位于可见区域当中，那么至少可以见到该线段的某个部分，即不可能完全看不见这条线段，但应注意，即使线段的两个端点都不在可见区域内，有时仍可看到该线段的某个部分，当然有时也可能完全看不到这根线段，这就要求我们能够确定这类线段是否可见，完成这一任务的一种简单方法就是：对小于或大于X及Y窗口限制的每种情况，都用一个二进制位来表示，故共需要四个二进制位，然后，将参数X及Y的值分别与窗口边界进行比较，倘若其值超出了窗口的边界，则将该点描述子相应的二进制位置为1，在线段两个端点的这种代码(描述子)都确定了之后，则将这两个代码值进行逻辑"与"操作(FORTRAN 86允许执行这种操作)，若相与的结果为0，则说明可以看到该线段的某个部分；否则，即在"与"操作结果不等于0时，则说明整个线段都位于可见区域之外，即该线段是不可见的。若只可见到线段的某个部分，那么，还必须计算出那些边界点，即算出从哪一点开始线段可见，所采用的计算方法是利用"相似三角形"的原理，这也是与Z裁剪使用的计算方法相一致的。

执行这些操作任务的子程序有wtovp(它用适当的参数去调用子程序xyclip)、xyclip(执行实际的裁剪操作)、code(取得该点位置相对于窗口的二进制代码)和ppush(计算出线段离开可见区域的那一点)。

窗口到视图口的变换

在图形点都通过了以上所有的处理步骤之后，现在，它终于可以显示出来了，因为它们构成的所有线段经过了裁剪处理，并且都落在给定的窗口之内。然而，应该记住，这种窗口的值是以"现实世界"为单位的，它们的大小尺寸的单位可能是英时或英里，一般来讲，这些不适宜于在图形输出设备上描绘出来。为了解决这一问题，以使该窗口可在图形输出设备上显示出来，必须进行另外的一种变换，即窗口到视图口的变换，视图口代表了图形输出设备上的物理位置及其大小，视图口命令(viewport)为这种变换建立了适当的参数。它共需要四个参数，这几个参数允许视图在图形显示表面上移动，也允许设置视图口的大小。值得注意的是，视图口与窗口的方位比例不一定要求相同，也就是说，窗口和视图口的竖直尺寸与水平尺寸之间的比率不需要相同。当然，若两个比率真不相同，则图形的形状将发生变化。以上的变换操作实现起来也非常简单，只要给窗口值取适当的比例因子，以填满视图口即可，执行这种变换操作的代码包含在子程序wtovp当中。

绘图仪接口

本图形软件包与Hewlett-Packard 7225A平板绘图仪相接口，两者之间的通信是通过RS232进行的，其传输速率为600波特，具体讲，这是通过使用Intellec Series III内的

iSBC 86/12 单板机上的8251串行通讯控制器实现的。采用的绘图仪具有一个较好的接口，它接收以 ASCII 码表示的命令，其命令的形式可以是：降低笔的高度("lower the pen")及从笔的当前位置到笔的另一位置画一线段("draw a line from the current pen position to another pen position")。完成这些操作的子程序为plot(确定需要送到绘图仪的那些字符)、ponum(将浮点数转换为相应的整数值，并用ASCII字符表示出来，采用的是截尾舍入方法)、putout(负责处理与8251串行通信控制芯片的接口)及 plots(初始化 iSBC86/12 单板机上的串行控制器芯片及波特率发生器)。

性能分析

上述的所有子程序段都是用Intel FORTRAN 86 编译程序进行编译的，且都在 Intellec Series III 开发系统上运行。其 8086 的硬件环境由一个 Inteli SBC 86/12 单板机组成，它带有一片位于 iSBC 337 卡上的 8087。该 iAPX 86/20(8086 加 8087)所采用的时钟频率为 5MHz，单板上的存贮器(64KBRAM)在每次存贮器访问操作中都可能插入一到三个等待状态，此外，根据存贮空间的大小、程序的大小以及 Series III 的存贮器空间需求量，还有可能需要使用单板之外的存贮器，以运行这些程序。

表中给出的时间并不代表真正的绘图时间，它仅仅给出了为了形成要送到绘图仪的那些输出数据而要花的时间，这是因为绘图仪物理运动速度的限制将使得 iAPX 86/20 系统不可能以其最高的运算速度来发出绘图命令，实际上，画一个不算很复杂的图形，就差不多要花去绘图仪半小时到四十五分钟的时间(图 19)。

对于画出的每根线来讲，都必须执行五次1×4与4×4矩阵的乘法操作，还要执行一些其它的浮点操作，例如除法、比较操作等等。举例来说，在执行裁剪操作的时候，就必须把线的端点值与裁剪参数进行比较，倘若只可见到线的某个部分，那末，还要求算出该线段离开可视区域的那一点，这又需要进行十二次浮点运算。还有一个例子是在进行窗口到视图口的转换时，对于画出的每根线段，都必须进行四次浮点乘法、四次浮点除法及四次浮点加法操作。

此外，在进行旋转、取比例尺、移动变换或在观测点发生变化时，也必须执行4×4矩阵的乘法操作，除此之外，还必须运行各种三角函数子程序，如正弦函数、余弦函数等等，以把旋转参数值送至以上的矩阵之中。

表 1 给出了部分性能测试结果。

表 1　　性能测试

	图形编号	
	一	二
图上的点数	117	9131
实际画出的点数	117	6114
86/20的执行时间(秒)	2.84	188
使用8087仿真器时8086的执行时间(秒)	144.77	9801
PDP11/45的执行时间(秒)	1.7	120