带有一个投影仪的光学装置可以制成彩色的系统。更多细节参见[SHER79]。其他类似的光阀系统利用LCD来调整光线。

表4-3概括了四种主要显示技术的特点，然而，技术更新的步伐很快，在接下来的几年中一些关系可能发生改变。还要注意到，液晶显示的比较是对无源寻址的，使用有源阵列，可以获得灰度等级和彩色。更多关于这些显示技术的细节可参见[APT85；BALD85；CONR85；PERR85；SHER79；TANN85]。

表4-3 显示技术的比较

	阴极射线管	电致发光	液 晶	等离子体板
能耗	中	中~良	优	中
屏幕尺寸	优	良	中	优
深度	差	优	优	良
重量	差	优	优	优
鲁棒性	中~良	良~优	优	优
亮度	优	优	中~良	优
寻址能力	良~优	良	中~良	良
对比度	良~优	良	中	良
每点亮度等级	优	中	中	中
视角	优	良	差	良~优
色彩能力	优	良	良	中
相对费用	低	中~高	低	高

4.3 光栅扫描显示系统

光栅图形系统的基本概念在第1章已经提及，第2章对光栅显示可能的操作类型进行了更深的探讨，在本节中我们讨论光栅显示器的各种元素，重点放在各种光栅系统相互不同的两个基本方面。

第一，多数光栅显示器都有一个专门的硬件，用来帮助进行扫描转换，将输出图元转换成位图，并且执行移动、拷贝、修改像素或者像素块等光栅操作，我们把这个硬件称为*图形显示处理器*。显示系统之间最基本的不同在于显示处理器做多少工作，驱动光栅显示器的通用CPU上执行的图形子程序相对的又做多少工作。注意，有时候图形显示处理器也叫作*图形控制器*（强调它同其他外设的控制单元的相似性）或*显示协处理器*。第二个不同在于像素图与计算机通用内存的地址空间之间的关系，像素图是计算机通用内存的一部分，或者是独立的。

在4.3.1节我们介绍了一个简单的光栅显示器，它由一个CPU和一个视频控制器组成，像素图作为 CPU 内存的一部分，视频控制器驱动CRT。没有显示处理器，CPU既做了应用程序的工作也做了图形的工作。在4.3.2节，介绍了一个有单独像素图的图形处理器，4.3.3节讨论了广阔范围的图形处理器的功能，4.3.4节讨论了有图形处理器存在时，像素图可以集成到CPU地址空间中的方法。

165

4.3.1 简单的光栅显示系统

图4-18所示的是最简单和最普通的光栅显示系统的结构，内存和CPU之间的关系和其他非图形计算机系统的一样，但是，内存的一部分还充当像素图，视频控制器显示帧缓存中定义的图像，按照光栅扫描频率的规定通过一个独立的访问端口访问内存。在一些系统中，固定的一部分内存永久地分配给帧缓存，而一些系统有几个相同功能内存区域（在个人计算机中有时称为*页*），其他的系统则可以指定（通过寄存器）任意一部分内存作为帧缓存，在这种情况下，系统的组织结构可能如图4-19所示，或者整个系统内存可以都是双端口的。

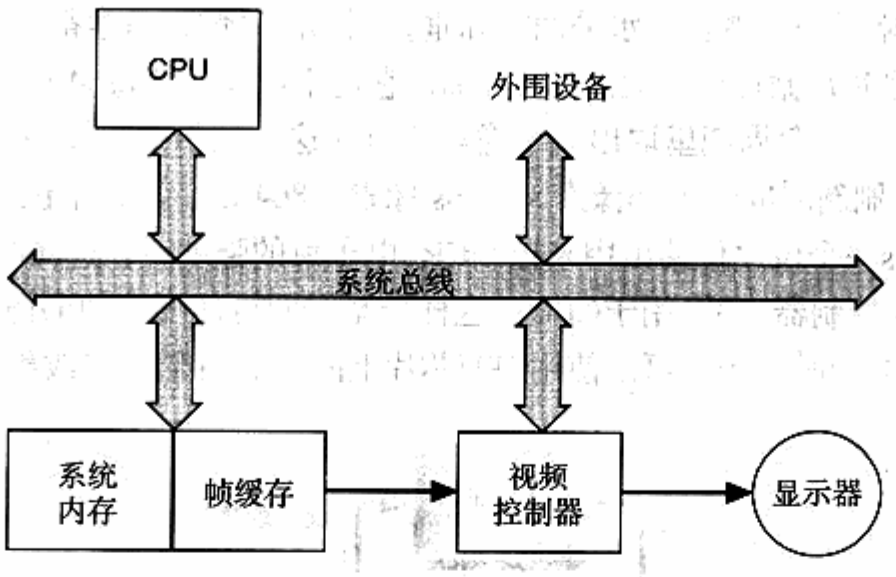


图4-18 普通光栅显示系统的结构，专用的一部分系统内存是双端口的，这样它可以直接被视频控制器访问，而不用中断系统总线的工作

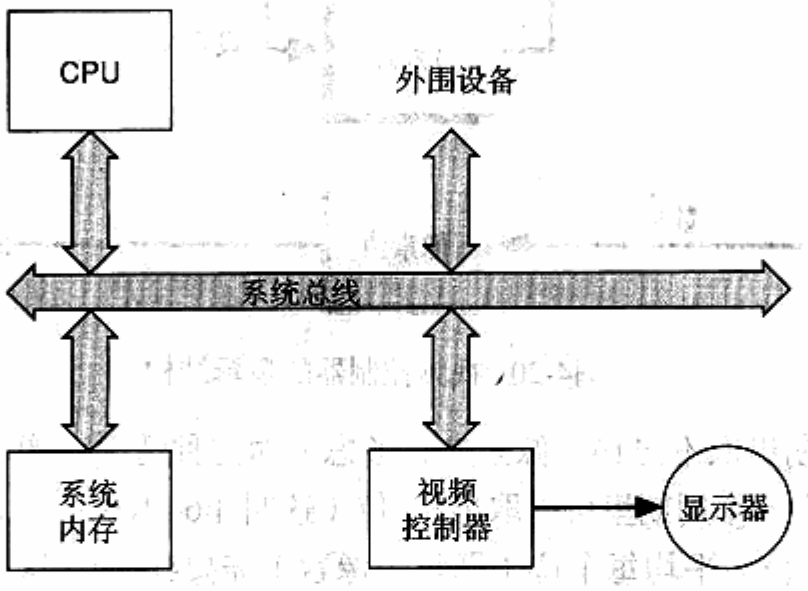


图4-19 简单的光栅显示系统结构。因为帧缓存可以存储在系统内存的任何地方，视频控制器通过系统总线访问内存

应用程序和图形子程序包共享系统内存，并由CPU执行。图形软件包包含扫描转换过程，当应用程序调用子程序时，比如说调用SRGP_lineCoord（x_1, y_1, x_2, y_2），图形软件包能设置帧缓存中适当的像素（关于扫描转换过程的细节参见第3章）。因为帧缓存在CPU的地址空间里，图形软件包可以很容易地访问它来设置像素，并实现第2章里描述过的PixBlt指令。

视频控制器在帧缓存里轮转，每次一条扫描线，通常是每秒钟60次，内存引用地址和光栅扫描同步生成，内存中的内容用来控制CRT电子束的亮度或者颜色。视频控制器的结构如图4-20所示。光栅扫描生成器产生偏转信号，这些信号用来产生光栅扫描，它还控制X地址寄存器和Y地址寄存器，这两个寄存器依次定义了下一个要访问的内存位置。

假设帧缓存的x编址是从0到x_{max}，y编址是从0到y_{max}，那么在一个刷新周期的开始，X地址寄存器设置为0，Y地址寄存器设置为y_{max}（最顶部的扫描线），当第一条扫描线生成的时候，X地址每次增加1直到 x_{max}，每个像素的值被取出，并用来控制CRT电子束的强度，第一条扫描线生成完毕后，X地址寄存器复位为0，Y地址寄存器减1，然后依次处理该条扫描线上的各个像素，整个过程对后面的扫描线重复执行，直到最后一条扫描线（$y=0$）生成。

在这种简单的情形下，对每一个要显示的像素的帧缓存都要进行一次内存访问，对一个640像素×480行的中等分辨率显示器，估计显示一个1位像素所用时间的简单方法是：1/(480×640

×60) = 54 ns。这里忽略了一个事实：水平回扫和垂直回扫的时候，像素都不显示（见习题4.10），但是一般的RAM存储器芯片循环周期大约为200 ns，它们不可能支持每54 ns访问一次的要求！因此视频控制器必须在一个内存周期里取出多个像素值，在这个例子中，控制器必须在一个内存周期里取出16个位，从而刷新时间变为16像素 × 54 ns/像素 = 864 ns。这16个位存放在视频控制器的寄存器里，然后每54 ns一个位逐位移出用来控制CRT电子束的强度，在这864 ns里，大约有4个内存周期：一个用于视频控制器，3个用于CPU。这样分享时间可能要让CPU访问内存的时候等待，就有可能降低CPU速度的25%。当然可以使用CPU芯片上的cache存储器来改善这个问题。

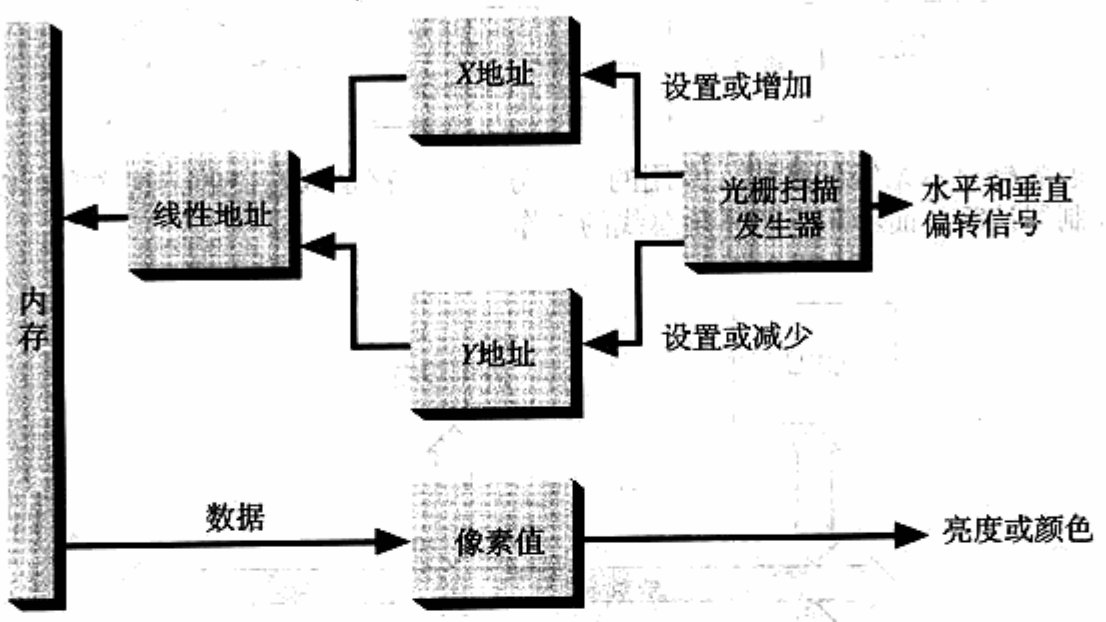


图4-20 视频控制器的逻辑结构

可能一个内存周期里取不出16个像素来，考虑下面这种情况：像素图使用5个64 KB的内存芯片实现，每个芯片在一个周期里可以取出1个位（这叫作64 KB × 1片芯片结构），在200 ns的周期时间里取出一共5个位，平均每个位（即每个像素）需要40 ns，这个速度比54ns/像素的速度快不了多少，几乎没有时间让CPU访问内存（除非是在7 ms的扫描线间回扫时间和1250 ms的帧间垂直回扫时间里），但若使用5个32 KB × 2片的芯片，200 ns内可以取出10个像素，就给CPU留有一半多一点的时间。对1200 × 1600的显示器，像素时间是1/(1600 × 1200 × 60) = 8.7 ns，200 ns的内存周期时间里，每个周期必须取出200/8.7 = 23个像素。1600 × 1200的显示器需要1.92 MB内存，可以由8个256 KB的芯片实现，可是256 KB × 1片的芯片每个周期只能取出8个像素，而32 KB × 2片的芯片每周期可以取出64个像素，就给CPU提供了三分之二的空闲内存周期时间。

CPU对内存的访问和视频控制器对内存的访问明显是一个问题：表4-4说明了问题的数量级。解决的办法是适应光栅显示需要的RAM体系结构，我们将在第18章讨论这些体系结构。

表4-4 描绘图像时处理器可以访问包含位图的内存的时间的百分比

可视区域 像素 × 线	芯片大小	芯片数	每次访问 的像素	视频控制器两次 访问的时间间隔(ns)	处理器访问时间 的百分比
512 × 512	256K × 1	1	1	64	0
512 × 512	128K × 2	1	2	127	0
512 × 512	64K × 4	1	4	254	20
512 × 512	32K × 8	1	8	507	60
512 × 512	16K × 16	1	16	1017	80
1024 × 1024	256K × 1	4	4	64	0

（续）

可视区域像素×线	芯片大小	芯片数	每次访问的像素	视频控制器两次访问的时间间隔(ns)	处理器访问时间的百分比
1024×1024	128K×2	4	8	127	0
1024×1024	64K×4	4	16	254	20
1024×1024	32K×8	4	32	407	60
1024×1024	16K×16	4	64	1017	80
1024×1024	1M×1	1	1	16	0
1024×1024	64K×16	1	16	254	21
1024×1024	32K×32	1	32	509	61

注：假定内存周期200ns，显示频率为60Hz，假定512×512显示器的像素时间为64ns，1024×1024的为16ns，这些时间是随意假设的，因为它们没有包括水平和垂直描绘时间，实际像素时间相应的大约是45ns和11.5ns

迄今为止我们讨论的只是单色的，每个像素一个位的位图。这个假设对一些应用是可以的，但对其他应用却非常的不满意。对每个像素的亮度的附加控制可以通过为每个像素存储多个位来获得：2位可以获得4个亮度等级，等等。这些位不仅可以用于控制亮度，还可以用于控制颜色。需要多少个位才能使存储的图像看上去是具有连续灰度的？通常5或者6位就足够了，但是8位或者更多位可能是需要的。因此，对彩色显示器，一个有些简单的说法提出需要三倍的位数：添加的原色红、绿、蓝中的每种需要8个位（见第13章）。

尽管固态RAM的价格在下降，每像素24位的系统仍然相对昂贵，并且许多彩色应用在一幅图像中不需要2^{24}种不同的颜色（一般只有2^{18}~2^{20}种）。另一方面，在一幅给定的图像或者一个应用中经常需要使用很少的颜色，也需要从图像到图像或者从应用到应用之间改变颜色的能力，还有，在许多图像分析和图像增强的应用中，想要改变图像的视觉效果，但是不改变定义图像的基本数据。比如说，把所有值低于某个阈值的像素显示成黑色，把亮度范围扩大，给单色图像创建出伪彩色显示。

由于这些各种各样的原因，视频控制器经常包括一个视频查找表（video look-up table）（也称为查找表(look-up table, LUT)），查找表的条目数和像素值一样多，像素值不再用来直接控制电子束强度，而是作为索引值来访问查找表，查找表中条目的值再用来控制CRT的亮度或者颜色。一个值为67的像素就表示访问查找表中第67项的内容，并且用这个内容控制电子束强度。在一个显示周期中对每个像素进行这样的查找操作，所以查找表必须能够快速访问，CPU必须能在程序命令的时候装入查找表。

在图4-21中，查找表插入在帧缓存和CPU的中间，帧缓存每个像素有8位，所以查找表有256项。

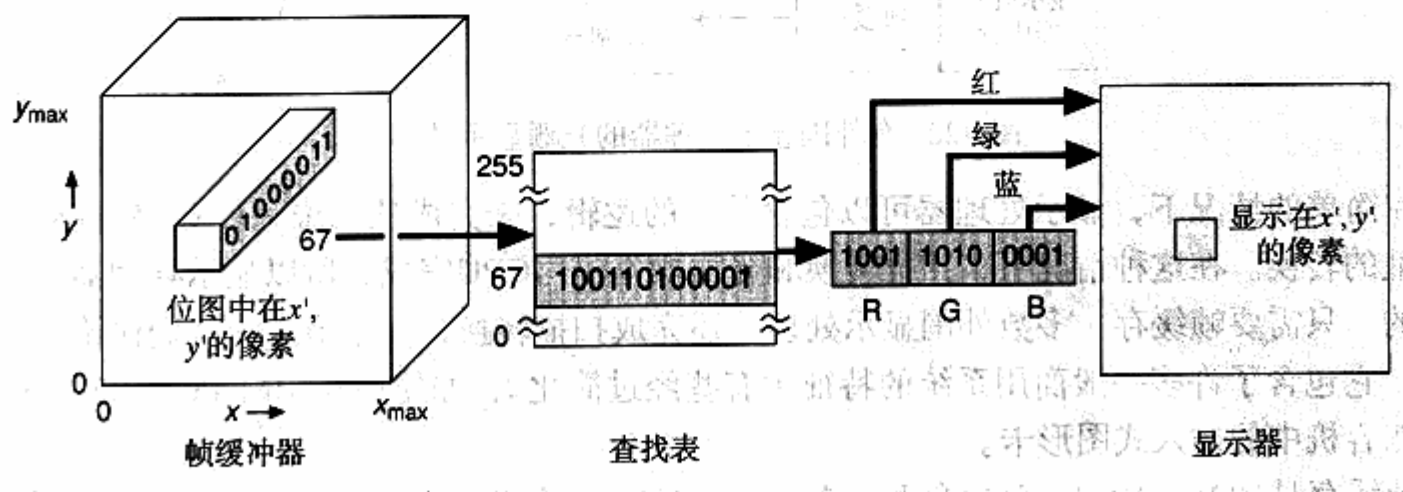


图4-21 视频查找表的结构。值为67（二进制值01000011）的像素显示到屏幕上，红色电子枪的强度为最大值的9/15，绿色的为10/15，蓝色的为1/15，所示的是12位的查找表，通常的会达到24位

图4-18和图4-19显示的简单光栅显示系统的结构使用在很多便宜的个人计算机中。这样的系统造价低，但是有很多缺点。首先，软件扫描转换很慢，例如一条扫描线上的每个像素的（x, y）地址都要计算，然后转换成由一个字节和字节中的位组成的内存地址。尽管每一步都很简单，但是都要重复许多次，基于软件的扫描转换使得应用程序与用户交互的速度全部变慢，可能会使用户不满。

这种结构的第二个缺点是：当寻址能力或者显示刷新速率增加时，视频控制器的内存访问次数也随之增加，因此降低了CPU可用的内存周期的数目，CPU也因此慢下来。特别是图4-19中的结构。在图4-18中，系统内存有一部分是双端口的，当CPU访问帧缓存以做扫描转换或者光栅操作时，速度下降就会发生。在考虑CPU访问帧缓存的方便性和系统的结构简单性时，也还要考虑这两个缺点。

4.3.2 具有外围显示处理器的光栅显示系统

具有外围显示处理器的光栅显示系统是一种避免了简单光栅显示器的缺点的普遍结构（见图4-22），它引入了一个独立的执行诸如扫描转换和光栅操作等图形功能的图形处理器和一个独立的用于图像刷新的帧缓存。现在我们有两个处理器：通用CPU和专用显示处理器。我们还有三块内存区域：系统内存，显示处理器内存和帧缓存。系统内存存放数据以及在CPU上执行的程序：应用程序，图形软件包和操作系统。相似地，图形处理器内存上也存放着数据和进行扫描转换及光栅操作的程序。帧缓存存放扫描转换和光栅操作生成的可显示的图像数据。

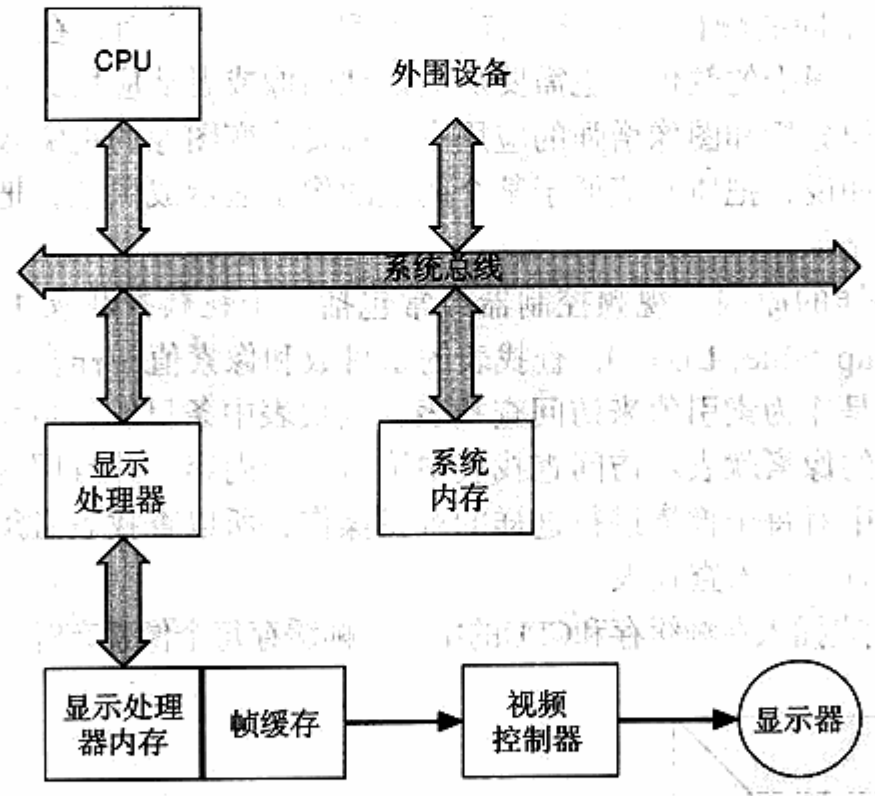


图4-22 有外围显示处理器的光栅显示结构

在简单的情况下，显示处理器可以包含特定的逻辑，来完成从二维（x, y）坐标到线性内存地址的转换。在这种情况下，扫描转换和光栅操作仍由CPU完成，所以显示处理器内存是不需要的，只需要帧缓存。多数外围显示处理器还完成扫描转换，在本节中，我们讨论一个原型系统，它包含了许多一般商用系统的特征（有些经过简化），如使用在IBM PC、XT、AT、PS以及兼容机中的插入式图形卡。

帧缓存是1024 × 1024 × 8位/像素，查找表有256项，每项12位，红、绿、蓝每种颜色使用4位，坐标原点在左下方，仅显示像素图的前768行（y从0到767），显示器有6个状态寄存器，可

以被不同的指令设置，并影响其他指令的执行。这些寄存器是：CP（由X位置寄存器和Y位置寄存器组成），FILL，INDEX，WMODE，MASK和PATTERN。下面将解释它们的操作。

简单光栅显示的指令如下：

- Move（x, y） 将定义当前位置（CP）的X、Y寄存器设置成 x 和 y，因为像素图是1024×1024的，x和y必须在0到1023之间。
- MoveR（dx, dy） dx和dy的值被加到X、Y寄存器，因此定义了新的CP，dx和dy必须在－1024到1023之间，以2的补码表示。所做的加法可能导致溢出，因此X寄存器和Y寄存器环绕式处理。
- Line（x, y） 从CP到（x, y）画一条直线，（x, y）成为新的CP。
- LineR（dx, dy） 从CP到CP +（dx, dy）画一条直线，CP +（dx, dy）成为新的CP。
- Point（x, y） 设置（x, y）处的像素，（x, y）成为新的CP。
- PointR（dx, dy） 设置CP +（dx, dy）处的像素，CP +（dx, dy）成为新的CP。
- Rect（x, y） 在CP和（x, y）之间画一个矩形，CP不受影响。
- RectR（dx, dy） 在CP和CP +（dx, dy）之间画一个矩形，参数dx可以看成是矩形的宽，dy看成是矩形的高，CP不受影响。
- Text（n, *address*） 从CP开始显示内存位置为*address*的n个字符。字符定义在7×9的像素网格中，垂直和水平分别有额外的2个分隔像素用于分隔字符和行。CP更新为第$n+1$个字符将被显示的区域的左下角。
- Circle（*radius*） 以CP为圆心画一个圆，CP不受影响。
- Arc（*radius*，*startAngle*，*endAngle*） 画圆心在CP的一段圆弧，角度单位是十分之一度，从x轴沿逆时针方向增加。CP不受影响。
- CircleSector（*radius*，*startAngle*，*endAngle*） 画一个扇形的封闭区域，直线从CP连接到圆弧的两个端点，CP不受影响。
- Polygon（n，*address*） 地址为*address*的内存存储顶点列表（$x_1, y_1, x_2, y_2, x_3, y_3, \cdots, x_n, y_n$），以（$x_1, y_1$）为起点画多边形，经过所有这些顶点直到（$x_n, y_n$），然后回到（$x_1, y_1$），CP不受影响。
- AreaFill（*flag*） *flag*用来设置光栅显示中的FILL标志，当此标志设置成ON的时候（用一个非零的*flag*值），用命令Rect、RectR、Circle、CircleSector、Polygon创建的区域都会用Pattern命令定义的图案填充。
- PixelValue（*index*） 像素值*index*装入INDEX寄存器，当任何一个之前列表中的输出图元进行扫描转换时将该值填入像素图。
- Pattern（*row*1，*row*2，…，*row*16） 16个2字节的参数定义了填充图案，用来填充由Rect、RectR、Circle、CircleSector、Polygon创建的区域。图案是一个16×16的位矩阵，当创建一个填充区域且FILL标志为ON的时候，如果PATTERN寄存器中的一个位是1，则INDEX寄存器中的像素值填入到像素图中，否则像素图不受影响，当PATTERN寄存器中的所有位都为1时，进行的就是实心填充。
- WBlockR（dx, dy, *address*） 将开始地址为*address*的主内存中存放的8位像素值写入到像素图中从CP到CP +（dx，dy）的矩形区域，从区域的左上角开始向顶向下逐行写入。
- RBlockR（dx, dy, *address*） 将像素映射中从CP到CP +（dx, dy）的矩形区域读入开始地址为*address*的主内存中，从区域的左上角开始向顶向下逐行读入。

166 ~ 171

- RasterOP（*dx*, *dy*, *xdest*, *ydest*） 将帧缓存中从CP到CP +（*dx*, *dy*）的矩形区域和左下角为（*xdest*, *ydest*）同样大小的目标区域结合，目标区域被覆写，结合受WMODE寄存器的控制。
- WMode（*mode*） *mode*的值被装入WMODE寄存器中，该寄存器控制帧缓存中的像素和要写入帧缓存的任何像素值结合的方式，参数*mode*有四个值：**replace**（替换），**xor**（异或），**and**（与），**or**（或）。这些模式进行的操作如第2章所述。注意，为了简洁，前面对命令的描述编写得似乎**replace**（替换）是仅有的一种写模式值。在xor模式下，写入到帧缓存中的新像素值和当前的值逐位地进行异或操作结合到一起。

172

- Mask（*mask*） 8位的*mask*值装入到MASK寄存器中，该寄存器控制帧缓存的哪一位在写帧缓存的时候被修改：1表示允许修改对应的位，0则禁止修改。
- LuT（*index*, *red*, *green*, *blue*） 用给定的颜色装入查找表的第*index*项中。每个颜色参数的低4位装入到查找表中。

表4-5总结了这些命令。注意，MASK寄存器和WMODE寄存器影响所有写入帧缓存的命令。

表4-5 光栅显示命令概括

命令助记符	参数及长度	CP所受的影响	影响命令的寄存器
Move	*x*(2), *y*(2)	CP := (*x*, *y*)	–
MoveR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP
Line	*x*(2), *y*(2)	CP := (*x*, *y*)	CP, INDEX, WMODE, MASK
LineR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP, INDEX, WMODE, MASK
Point	*x*(2), *y*(2)	CP := (*x*, *y*)	CP, INDEX, WMODE, MASK
PointR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP, INDEX, WMODE, MASK
Rect	*x*(2), *y*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
RectR	*dx*(2), *dy*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Text	*n*(1), *address*(4)	CP := next char pos'n	CP, INDEX, WMODE, MASK
Circle	*radius*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Arc	*radius*(2), *startAngle*(2), *endAngle*(2)	–	CP, INDEX, WMODE, MASK
CircleSector	*radius*(2), *startAngle*(2), *endAngle*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Polygon	*n*(1), *address*(4)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
AreaFill	*flag*(1)	–	–
PixelValue	*index*(1)	–	–
Pattern	*address*(4)	–	–
WBlockR	*dx*(2), *dy*(2), *address*(4)	–	CP, INDEX, WMODE, MASK
RBlockR	*dx*(2), *dy*(2), *address*(4)	–	CP, INDEX, WMODE, MASK
RasterOp	*dx*(2), *dy*(2), *xdest*(2), *dest*(2)	–	CP, INDEX, WMODE, MASK
Mask	*mask*(1)	–	–
WMode	*mode*(1)	–	–
LuT	*index*(1), *red*(1), *green*(1), *blue*(1)	–	–

注：每个参数后括号内的数字是参数的长度（以字节为单位），本表还指出了命令对CP的影响以及哪些寄存器影响命令运作的方式。

命令和立即数通过位于CPU地址空间中专用部分的先进先出（FIFO）缓冲区（如队列）传输到显示处理器，图形软件包把命令送入到队列中，显示处理器取得指令并执行。在特定的内存位置还存放着指向这个缓冲区的起始地址和结束地址的指针，供CPU和显示处理器访问。每次移走一个字节时显示处理器修改指向起始地址的指针，每次加入一个字节时CPU修改指向结束地址的指针，要做适当的测试以保证对空的缓冲区不进行读操作，对满的缓冲区不进行写入操作，使用直接内存访问来给指令提供寻址数据。

对于命令传递，队列比使用单个指令寄存器或显示处理器可以访问的位置要更吸引人。第一，变长的指令适合队列的概念；第二，CPU可以超前于显示处理器，把许多显示命令排在队列里。当CPU发布完显示命令后，在显示处理器处理命令清空队列时它可以去处理其他工作，

编程实例

对显示的编程有点类似于第2章所述的SRGP软件包的使用。所以我们这里只给出了一些例子。“Z”表示指定的是十六进制值，“A”表示使用后面括号中的地址列表。

下面一段程序创建了全黑色背景上的一条白色直线：

```
LuT          5, 0, 0, 0                    查找表第5项为黑色
LuT          6, Z'F', Z'F', Z'F'           查找表第6项为白色
WMode        replace
AreaFill     true                          打开FILL标志
Pattern      32Z'FF'                       32个全1的字节，实心图案
Mask         Z'FF'                         使能够写入到所有平面
PixelValue   5                             使用像素值5进行扫描转换
Move         0, 0
Rect         1023, 767                     帧缓存中可视的部分现在是黑色
PixelValue   6                             使用像素值6进行扫描转换
Move         100, 100
LineR        500, 400                      画直线
```

173 ~ 174

下一段程序创建黑色背景上与蓝色三角形交迭的红色圆：

```
LuT          5, 0, 0, 0                    查找表第5项为黑色
LuT          7, Z'F', 0, 0                 查找表第7项为红色
LuT          8, 0, 0, Z'F'                 查找表第8项为蓝色
WMode        replace
AreaFill     Z'FF'                         打开
Pattern      32Z'FF'                       32个全1的字节，实心图案
Mask         Z'FF'                         使能够写入到所有平面
PixelValue   5                             准备好画黑色的矩形
Move         0, 0
Rect         1023, 767                     现在帧缓存中可视部分为黑色
PixelValue   8                             接下来画蓝色三角形，把三角形当成三个顶点的多边形
Polygon      3, A(200, 200, 800, 200, 500, 700)
PixelValue   7                             把圆画在三角形的上面
Move         511, 383                      把CP移动到显示器的中心
Circle       100                           以CP为圆心画半径100的圆
```

4.3.3 显示处理器的附加功能

我们的简单显示处理器只完成一些能被实现的与图形相关的操作。设计者所面临的诱惑是：给显示处理器增加功能，以更多地减轻CPU的负担，比如使用局部内存来存放显示指令列表，完成裁剪和窗口-视口转换，也许还提供拾取相关逻辑和图形元素被拾取后的自动反馈。最终，显示处理器变成了另一个完成通常图形交互工作的通用CPU，设计者又要尝试着增加特定功能的硬件以减轻显示处理器的负担。

Myer和Sutherland在1968年确定了这个“轮回”（wheel of reincarnation）[MYER68]。作者的观点是在通用功能和专用功能之间要有一个折衷，通常，专用硬件完成工作比通用处理器快。另一方面，专用硬件更昂贵且不能用于其他用途。这种折衷在图形系统设计中是持久的题目。

如果裁剪（第3章）增加到显示处理器的功能中，输出图元可以用坐标指定给处理器而不使用设备坐标。这种说明可以在浮点坐标中给定，虽然一些显示处理器只能处理整数（随着低廉的浮点芯片的使用，这种情况正在改变中）。如果只使用整数，应用程序使用的坐标也必须是整数的，或者图形软件包必须把浮点坐标映射到整数坐标。要使映射成为可能，应用程序必须给图形包一个矩形，这个矩形保证包含了指定给图形包的所有输出图元的坐标。然后这个矩

175

形映射到最大的整数范围，这样在这个矩形内的一切都是在整数坐标的范围里的。

如果子程序包是三维的，显示处理器就能够完成第5章和第6章中所述的复杂得多的三维几何变换和裁剪。同样，如果图形包里包含了三维曲面图元，比如多边形区域，显示处理器还能进行第15章和第16章讨论的可见面判定（visible surface-determination）和绘制步骤（rendering step）。第18章讨论了一些能使这些步骤完成得更快的把通用和专用VLSI芯片组织在一起的基本方法。许多商用的显示器都提供了这些特征。

另外一个经常给显示处理器增加的功能是*本地段存储*（local segment storage），也叫*显示列表存储*（display list storage）。显示指令被分组到命名的段中，具有未被裁剪的整数坐标，并存储在显示处理器内存中，允许显示处理器的操作更自主于CPU。

显示处理器对这些存储起来的段能做些什么呢？它可以对它们进行变换和重绘，像缩放和滚动。能够提供这些段到新位置的本地拖动功能。通过让显示处理器对光标位置和所有图形图元（更有效率的方法在第7章讨论）比较来实现本地拾取。当删除一个段的时候，需要用段存储来重新生成以填充产生的洞。段可以被创建、删除、编辑，还可以使段可见或者不可见。

段还可以被拷贝或者引用，这两种操作都减少了必须从CPU传送给显示处理器的信息，也使显示处理器自己的内存使用更经济。例如，创建一个VLSI芯片衬垫外形的显示指令在绘制的时候要多次用到，这些指令只需要给显示处理器传送一次，存储成一个段，然后每次出现同样的衬垫时就传送引用这个段的显示指令。利用这个功能可以建立复杂的层次数据结构，许多具有本地段内存的商用的系统可以拷贝和引用其他段。当显示段时，必须保存显示处理器的当前状态，然后进行另一个段的引用，就像保存CPU的当前状态，然后进行子程序调用一样。引用可以嵌套，导致了*结构显示文件*或*层次显示列表*的出现，如在PHIGS中 [ANSI88]，第7章将进行更深的讨论。在GKS图形包[ENDE87；HOPG86]中使用的是线性无嵌套的显示列表，一个已存在的段可以拷贝到将要创建的段中。

段数据结构不是必须放在显示处理器的内存中（图4-22），它们可以直接由图形包在系统内存中创建，由显示处理器访问。当然在这种方式下需要显示处理器必须直接连接在系统总线上，RS232接口和以太网速度的连接都是不可行的。

如果所有要显示的信息都用段数据结构表示的话，显示处理器还可以实现窗口管理器的操作，如移动、打开、关闭、改变大小、滚动、入栈、出栈等。当窗口平移时，段也进行有效的视点旋转。在某些地方“轮回”又再度出现，但是我们要注意到便宜得令人吃惊的专用VLSI芯片，等到本书过时，窗口管理芯片可能只花几个美元就能买一片。确实，技术更新换代是如此之快，显示处理器中的图形功能将继续惊人地增加，而其价格则继续下降。

尽管和4.3.1节中的简单光栅显示系统相比，具有图形显示处理器和独立帧缓存的光栅显示
176 系统结构有着许多优点，它也还是有着一些缺点。如果显示处理器是作为DMA端口上或者RS232接口上的外设，则每次传送指令给它的时候就会有相当多的操作系统管理开销（显示处理器的指令寄存器映射到CPU的地址空间就不会发生这种情况，因为图形包很容易就可以直接设置寄存器）。

如图4-22所示，也可对内存做有标记的划分。在显示列表内存中建立显示列表是很慢的，因为需要发布增加或者删除元素的显示处理器命令。显示列表可能不得不在主处理器内存中保存一份拷贝，因为它经常读不回来。因此，显示列表由图形子程序包直接建立在主存中的环境更灵活、更快速，对程序更方便。

光栅操作命令特别复杂，从概念上说，它应该有四种可能的源-目的对：系统内存到系统

内存，系统内存到帧缓存，帧缓存到系统内存，帧缓存到帧缓存（在此，图4-22中的帧缓存和显示处理器的内存被认为是相同的，因为它们都在同一个系统地址空间里）。但是在显示处理器系统里，对不同的源-目的对使用不同的方式处理，可能系统内存到系统内存的这种情况不存在。缺乏对称性使得程序员的任务变得复杂，降低了灵活性。比如说，如果位图超出屏幕的部分填充着的是菜单、字体等等，就很难利用主存来作为溢出区域。更进一步，因为像素图的使用是如此的广泛，不支持对存储在主存中的像素图进行光栅操作是不可行的。

另外一个问题是：扫描转换算法的输出必须输出到帧缓存，这个要求排除了双缓存：扫描转换在系统内存中创建一幅新的图像，然后把它拷贝到像素图替换掉存储在那里的原有图像。另外，某些窗口管理器策略和动画技术要求部分或者全部变暗的窗口在屏幕以外的调色板中保持，这又要求扫描转换到系统内存去（第10章和第19章）。

本节前面定义的显示处理器像许多真正的显示处理器一样，通过系统总线上的I/O传输在系统内存和帧缓存之间移动光栅图像。但是，实时操作里这种移动可能太慢，像动画、拖动、弹出窗口和菜单，操作系统初始化传输的时间和总线的传输率是瓶颈所在。通过增加显示处理器的内存以装载更多的屏幕外的像素图可以部分克服这个问题，但那样一来这部分内存就不能用于其他用途了——总之，几乎永远都不会有足够多的内存！

4.3.4 具有集成显示处理器的光栅显示系统

把帧缓存作为系统内存的一部分，我们可以克服上节所讨论的外围显示处理器的许多缺点，这就是图4-23所示的单地址空间（single-address-space, SAS）显示系统体系结构。这里显示处理器、CPU和视频控制器都在系统总线上，因此都可以访问系统内存。帧缓存的起始地址，一些情形下还有大小，都存放在寄存器中，双缓存就变得很简单，只需重新填入寄存器：扫描转换的结果可以送到帧缓存用于直接显示，或者送到系统内存别的地方用于以后的显示。类似地，显示处理器进行的光栅操作的源和目的可以在系统内存的任何地方（现在我们只对内存感兴趣）。这种安排还有一点吸引人之处，是因为CPU可以直接操作帧缓存中的像素，只要简单地读写合适的位就可以。

177

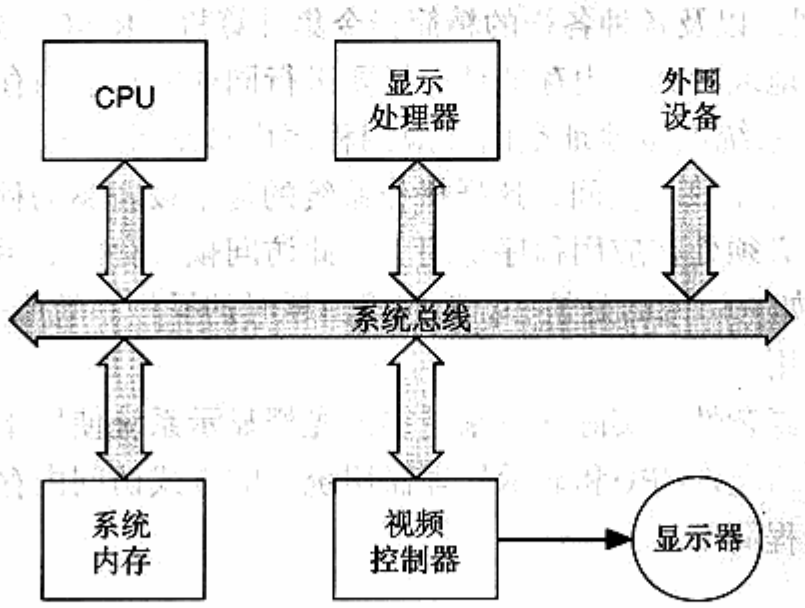


图4-23 单地址空间（SAS）光栅显示系统的体系结构，它有一个集成的显示处理器，该显示处理器可以有自己的用于存放算法的内存和工作存储器

但是，SAS体系结构有许多缺点，对系统内存访问的竞争是最严重的。通过把系统内存的特定部分专用于帧缓存和通过提供从视频控制器到帧缓存的第二个访问端口，至少可以部分解决这个问题，如图4-24所示。另一个解决方法是使用带有指令或数据高速缓冲存储器的CPU，减少CPU对频繁快速访问系统内存的依赖。当然这些方法和其他方法可以巧妙的集成到一起。

第18章将讨论更多的有关细节。由于限制，硬件实现的PixBlt只能在帧缓存上工作，应用程序员看到的一个PixBlt指令可能是分几种不同的情况来对待的，如果硬件不支持源和目的，还要进行软件模拟，一些处理器实际上已经足够快到完成软件模拟，特别是有指令高速缓冲存储器（instruction-cache memory），软件模拟的最内层循环可以保持在高速缓冲存储器里。

和前面提到的一样，帧缓存的非传统内存芯片结构也能避免内存竞争问题。一种方法是在一次访问时间里打开扫描线上所有的像素，因此减少了扫描转换到内存所需要的内存周期，特别是对于填充区域。德州仪器（Texas Instruments）开发的视频RAM（VRAM）结构可以在一
178 个周期里读出一条扫描线上的所有像素。第18章也会给出更多的细节。

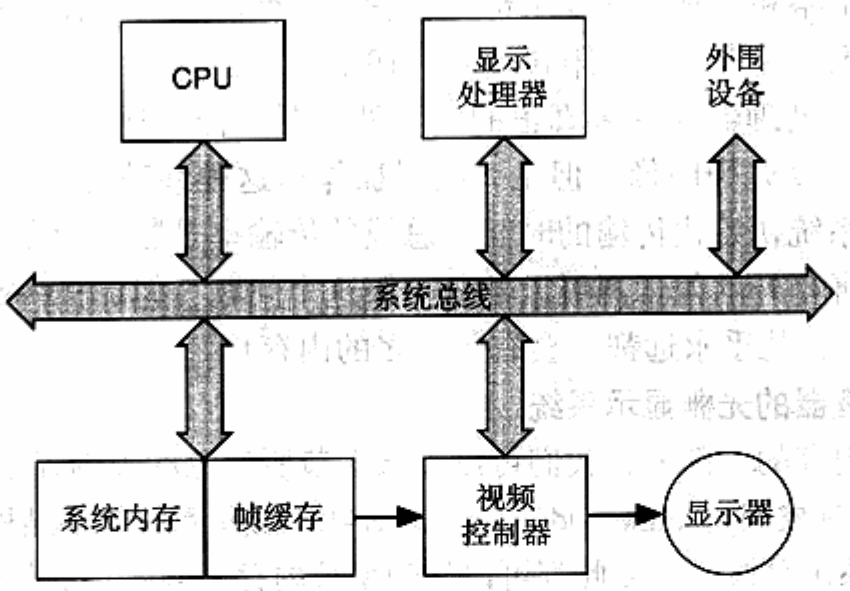


图4-24 公用单地址空间光栅显示系统的体系结构，具有一个集成的显示处理器（和图4-23比较），该显示处理器可以有私有的用于存放算法的内存和工作存储器，系统内存的专用部分是双端口的，以便视频控制器可以直接访问，而不用中断系统总线的工作

如果CPU具有虚拟地址空间时，出现了另一个设计上的复杂因素，像普遍应用的Motorola 680x0和Intel 80x86系列，以及各种各样的精简指令集计算机（RISC）处理器。在这种情况下，显示处理器生成的内存地址和其他内存地址一样要进行同样的动态内存地址转换。另外，许多CPU结构区分核心操作系统虚拟地址空间和应用程序虚拟地址空间，经常也需要帧缓存（在SRGP术语中是画布 0) 处于核心空间，这样操作系统的显示设备驱动程序可以直接访问。但是应用程序所分配的画布必须处在应用程序空间。因此访问帧缓存的显示指令必须区分核心地址空间和用户地址空间。如果访问的是核心，就应该由耗时的操作系统服务调用来调用显示指令，而不是由简单子程序调用。

尽管有这些潜在的复杂性，实际上越来越多的光栅显示系统使用单地址空间结构，一般是图4-24所示的那种类型。允许CPU和显示处理器用统一的方式访问内存的任何部分这一灵活性非常引人注目，并且编程简单。

4.4 视频控制器

视频控制器的最主要任务是持续地刷新显示。有两种基本的刷新方式：交错的和不交错的。前者使用在广播电视和设计用来驱动正规电视的光栅显示中。其刷新周期被分成两个部分，每
179 部分持续1/60秒，一次完整的刷新持续1/30秒，所有奇数行扫描线在第一部分时间里显示，所有偶数行扫描线在第二部分时间里显示。隔行（交错）扫描的目的是每次以60 Hz的频率在屏幕的所有区域放置一些新的信息，因为30 Hz的刷新频率容易导致闪烁。交错显示的净效果是

产生的图像其有效刷新频率更接近60 Hz而不是30 Hz。在相邻扫描线实际上显示相似信息时这一技术较为有用；在交替的扫描线上有水平线的图像将有严重的闪烁。大部分视频控制器以60 Hz或者更高的刷新频率刷新并且使用非交错扫描。

视频控制器的输出分为以下三种模式：RGB、单色和NTSC。对RGB（red，green，blue），用单独的电缆来传输红、绿、蓝信号，控制荫罩式显示器的三支电子枪，另外一根电缆传输标志开始垂直回扫和水平回扫的同步信号。RGB信号的电压、波形和同步时间都有标准，对480扫描线的单色信号，RS-170是其标准；彩色信号的标准是RS-170A；高分辨率单色信号的标准是RS-343。同步时间经常和绿色信号包含在同一根电缆里，这种情况下信号叫作复合视频（composite video）信号。单色信号使用同样的标准，但是只有亮度和同步信息，或者仅仅只是一根传输亮度和同步信号的复合电缆。

NTSC（国家电视系统委员会）视频是北美电视商品中使用的标准。颜色、亮度和同步信息都合并在一个带宽为5 MHz的信号中，以525条扫描线进行广播，分为两个262.5条扫描线的部分，只有480条扫描线是可见的，剩下的扫描线在每个部分结束进行垂直回扫的时候出现。单色电视使用亮度和同步信息，彩色电视还使用彩色信息控制三支电子枪。带宽的限制可以允许在分配给电视的频率范围内使用许多不同的频道进行广播。不幸的是，这个带宽限制了图像的质量，使得它的有效分辨率只有350 × 350。不过，NTSC是录影带录制装置的标准。问题总会改善，现在对用于录像带和卫星广播的1000线高清晰度电视（HDTV）的兴趣越来越大。欧洲和前苏联电视广播和录影带标准是SECAM和PAL，两个625扫描线、50 Hz标准。

一些视频控制器加入了可编程光标，光标的形状存储在16 × 16或者32 × 32大小、位于帧缓存顶部的像素图中，这样就避免了每次刷新周期里都需要把光标形状PixBlt到帧缓存中去。类似地，一些视频控制器在帧缓存的顶部增加了几个小的固定尺寸的像素图（叫作精灵（sprite）），这个特征经常用于视频游戏。

4.4.1 查找表动画

光栅图像的动画效果可以从几个途径获得。要显示一个旋转的物体，我们可以逐一地从略有不同的位置把物体的图像扫描转换到像素图中，扫描转换至少必须每秒钟10次（最好是15次到20次）才能获得平滑的效果，因此一幅新的图像必须在不多于100 ms的时间内创建出来，但是如果对物体进行扫描转换占用了这100 ms中的大部分，比如说75 ms，那么完整的物体只能在剩下的25 ms中显示，然后必须删除它，重新绘制，效果很差。使用双缓冲可以避免这个问题。帧缓存分为两个图像，每个图像拥有整个帧缓存中每个像素一半的位，我们把像素图的两半分别称为*image*0和*image*1，下面描述动画的生成：

180

```
装入查找表，用背景色显示所有像素；
物体扫描转换到image0；
装入查找表，仅显示image0
do {
  物体扫描转换到image1
  装入查找表，仅显示image1
  旋转物体的数据结构描述
  物体扫描转换到image0
  装入查找表，仅显示image0
  旋转物体的数据结构描述
} while（不结束条件）
```

当然，如果旋转和扫描转换物体的时间多于100 ms，那么动画看上去就在跳动，但从一幅图像到另外一幅图像的过渡是立即的，因为装入查找表通常只花不到1 ms的时间。

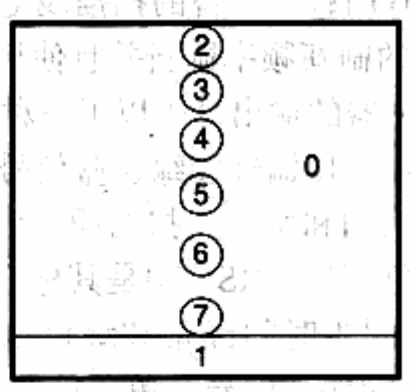


图4-25 弹跳的球的动画所对应帧缓存的内容

查找表动画的另外一种方式是显示短的重复的图像系列[SHOU79]。假设我们要显示一个弹跳的球。图4-25说明了帧缓存的装入方式，标志像素值的数字放置在帧缓存的每个区域。图4-26演示了如何在每一步装入查找表的内容，在颜色为0的背景上显示其中一个小球。循环查找表中的内容，我们可以获得动画效果。不仅是弹跳的小球，电影帐篷中移动的光线的效果、管子中流出的水流、旋转的车轮等等都可以模拟出来。21.1.4节将更深入地讨论这个主题。

通道号	动画每一步装入查找表的颜色										
	1	2	3	4	5	6	7	8	9	10	11
0	white	white	white	white	white	white	white	white	white	white	white
1	black	black	black	black	black	black	black	black	black	black	black
2	red										red
3		red								red	
4			red						red		
5				red				red			
6					red		red				
7						red					

图4-26 白色背景，在黑色表面上拍红球的查找表，未标明的项均为白色

对更复杂的循环动画（比如旋转一个复杂的线框架物体）可能在帧缓存中存放的就是独立的图像，而不是交迭的图像。在此情况下，一些显示的图像会有“洞”，少数这样的洞是没有影响的，特别是不追求完美时。但是出现更多的“洞”时，动画就失去了它的效果，这时候双缓存就显得更吸引人。

4.4.2 位图变换和窗口技术

在一些视频控制器里，像素图和观测表面之间的联系被切断，也就是说，帧缓存中的位置和观测表面上的位置不再有固定的对应关系，而由图像变换来定义对应关系。图像变换把帧缓存变换到观测表面，变换通常包括平移、放缩、旋转和裁剪操作。

图4-27演示了一些光栅显示系统中发现的变换的类型，帧缓存中由裁剪区域定义的一部分放大到充满整个观测表面。裁剪窗口与观测表面大小的比率必须是整数（图中为3）。没有使用帧缓存中裁剪窗口以外的像素，也没有修改任何像素的值，视频控制器以刷新速率进行变换。

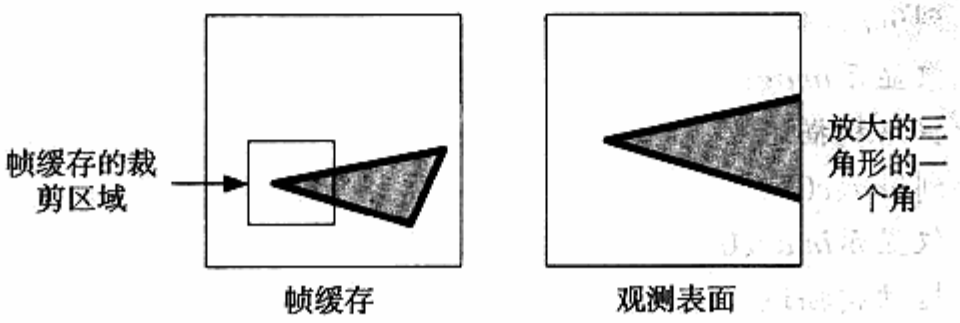


图4-27 在观测表面放大的帧缓存的一部分

图像变换每秒钟可以进行许多次，造成图像的翻卷或者放大的实时动态效果。还能够快速显示任意图像的系列，把它们装入到帧缓存中的不同区域，定时进行图像变换，首先显示第一个区域，然后显示下一个，等等。

181 ~ 182

放大一幅图像需要的比例变换在显示图像时可简单地通过重复窗口中的像素值来实现。比例因子为2时，每个像素值使用4次，在两条相邻的扫描线上每条两次。图4-28显示了以2为比例因子放大一个字母及和它相邻的一条直线的效果。放大并不能展现更多的细节，除非存储的图像分辨率比它显示的分辨率高。图形放大后有了更明显的锯齿形的外观。因此这种动画效果牺牲了空间分辨率，但是保持了全部的颜色范围。前面一节所述的双缓存技术保持了图像的空间分辨率，但是却减少了任一个图像中可使用的颜色数。

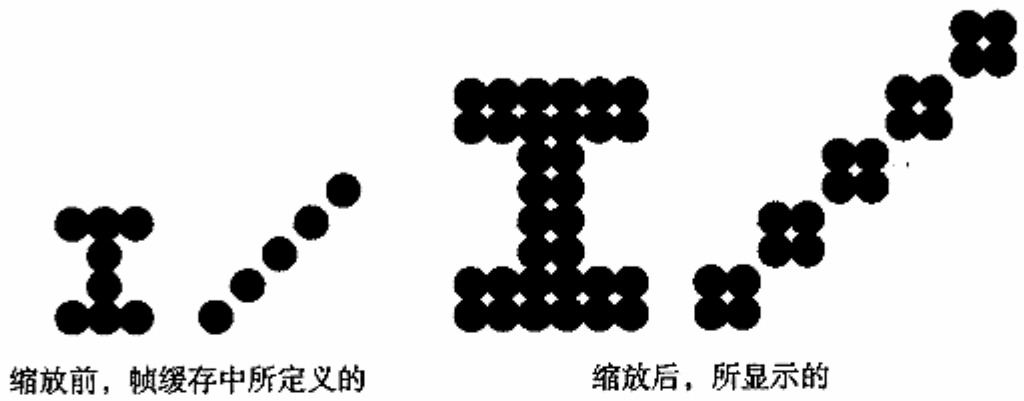


图4-28 因子为2，按比例放大一幅像素图的效果

在一些更普通的图像变换应用中，按比例变换的图像只覆盖由视口所定义的观测表面的一部分，如图4-29所示。现在我们必须给系统定义应该在视口以外的区域显示什么。一种可能是显示一种恒定的颜色或亮度，图示的另一种可能是显示帧缓存自己。后一种选择的硬件实现很简单，将存放视口边界坐标的寄存器与定义光栅扫描当前位置的X、Y寄存器比较，如果电子束处于视口中则从帧缓存中窗口区域取得像素，并且按照需要复制，否则像素从帧缓存中与电子束相同的坐标（x, y）位置取出。

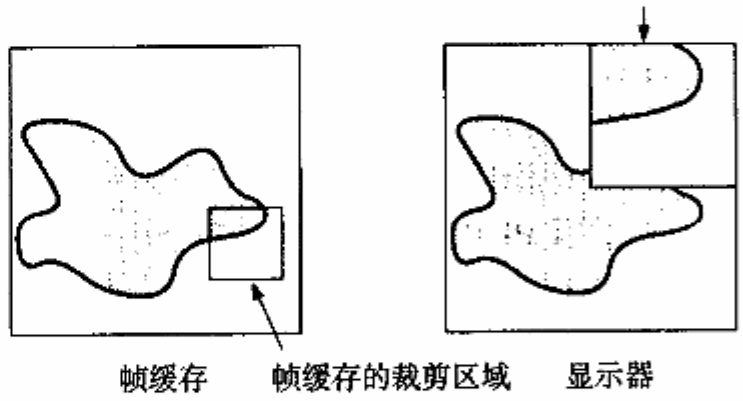


图4-29 帧缓存的一部分，由帧缓存的裁剪区域所指定，放大两倍之后叠放在未放大的帧缓存上

183

VLSI芯片已经实现了许多这样的图像显示功能，这些各种各样的变换是通用窗口管理操作需要的特定的例子。已经有了在视频控制器中使用的窗口管理芯片[SHIR86]。每个窗口在系统内存中有一块独立的像素图和一个描述窗口大小和原点的内存中的数据结构。显示扫描线的时候芯片知道哪个窗口是可见的，因此知道从哪个像素图中取得像素值。这些问题和其他高级的硬件问题将在第18章中讨论。

4.4.3 视频混合

视频控制器另一个有用的功能是视频混合。一幅定义在帧缓存中的图像可以和一个来自电视摄像头、录像机或者其他来源的视频信号混合一起形成一个复合图像。这种合成的例子在电视新闻、体育运动节目和天气预报中经常可以看到，图4-30描绘了通常的系统结构。

有两种类型的合成，一种是把图形图像置到视频图像中去，新闻报告员肩上显示的图表或者图形就是这个类型的典型例子，这种合成可以由硬件实现，它把帧缓存中指定的像素值作为

一个标志，指示应该显示视频信号，还是显示帧缓存中的信号，通常这个指定的像素值和帧缓存图像的背景颜色相对应，但是使用其他像素值能够获得感兴趣的效果。

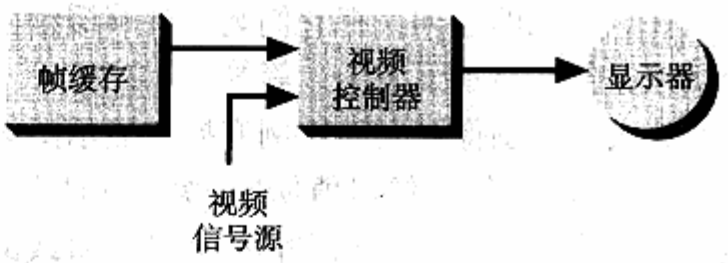


图4-30 把帧缓存的图像和视频信号源图像合成到一起的视频控制器

第二种合成是把视频图像放置在帧缓存图像的上面，就像天气预报员站在全屏幕的天气图前面那样，天气预报员实际上是站在一个背景前，背景的颜色（通常为蓝色）用来控制合成：当视频信号是蓝色时，显示帧缓存图像，否则显示视频图像，只要天气预报员不穿蓝色衬衫或者系蓝色领带，这种技术会很好地工作。

4.5 随机扫描显示处理器

图4-31演示了一个典型的随机（向量）显示系统，它一般和4.3.2节中基于显示处理器的光栅显示系统类似。当然刷新显示的时候没有像素图，显示处理器也没有扫描转换算法使用的本地内存，因为这个功能通常使用可编程逻辑阵列或者微代码来实现。

184

随机扫描图形显示处理器经常被称为显示处理单元（display processing unit，DPU），或者称为图形控制器。DPU有一个指令集和一个指令寄存器，并和任何计算机一样经过经典的取指、解码和执行周期。因为没有像素图，显示处理器必须每秒钟执行30~60次程序才能避免闪烁。DPU执行的程序存放在主存中，可以由通用CPU和DPU共享。

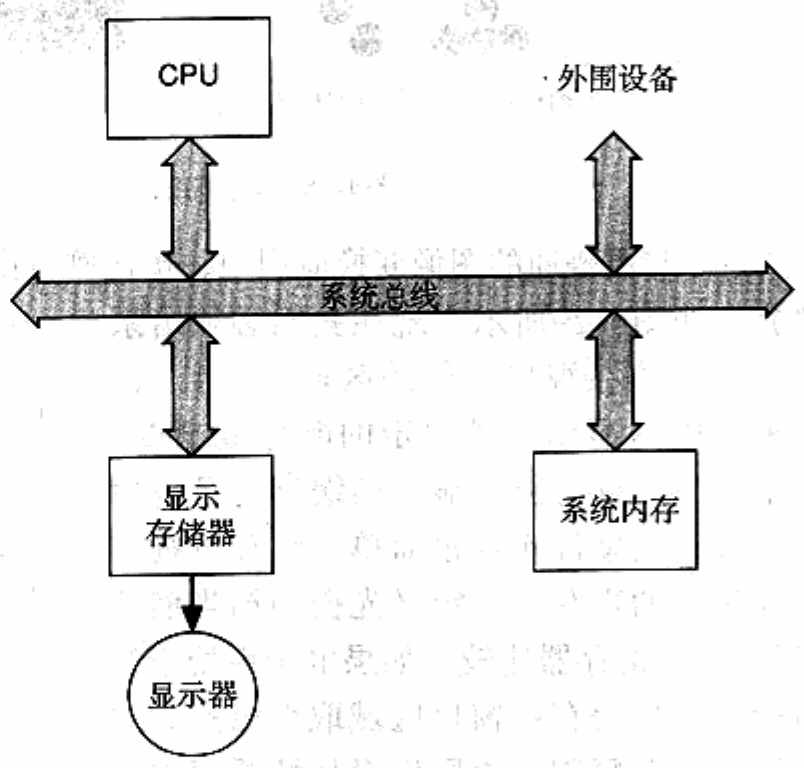


图4-31 随机显示系统的结构

应用程序和图形子程序包也驻留在主存中，在通用CPU中执行，图形包生成一个DPU指令的显示程序并告诉DPU从哪里开始执行程序，然后DPU异步地执行这个显示程序，直到图形包让它停止。程序结束处的JUMP指令把控制转回到程序的开始处，这样不用CPU干预，显示就可以继续刷新。

图4-32展示了一个简单随机扫描DPU的指令集和助记符，处理器有X、Y寄存器和一个指令计数器，指令定义为16位字长，LD指令中的R/A（relative/absolute）修饰符指明下面的地址是按照11位的相对地址对待还是按照10位的绝对地址对待。前一种情况下，11位的值加到X或者Y寄存器上，后一种情况下，10位的值替换寄存器中的内容。（相对移动需要使用11位是因为可以让2的补码表示的值在 – 1024到 + 1023的范围内。）

JUMP指令使用同样的R/A修饰符，只是修改的是指令计数器，从而影响控制流程的改变，SM、SP和SL指令提供了轮廓线、散布图等等的紧凑表示。

185

图4-33是一段简单的DPU程序，以汇编语言的风格写成，使用了许多指令。注意正方形和菱形是如何绘制的：第一个move指令是绝对的，而其他的是相对的，这样可以帮助在屏幕上拖动对象。如果不使用相对指令，拖动就需要修改用于显示对象的所有指令中的坐标值。最后

一条指令跳回到程序的开始。因为由L修饰符指定设置了帧锁定位，跳转的执行要延缓直到30 Hz时钟的下一次触发，这样可以允许DPU以30 Hz的频率刷新，但是却避免了短小程序更频繁的刷新，那样会烧坏荧光物质。

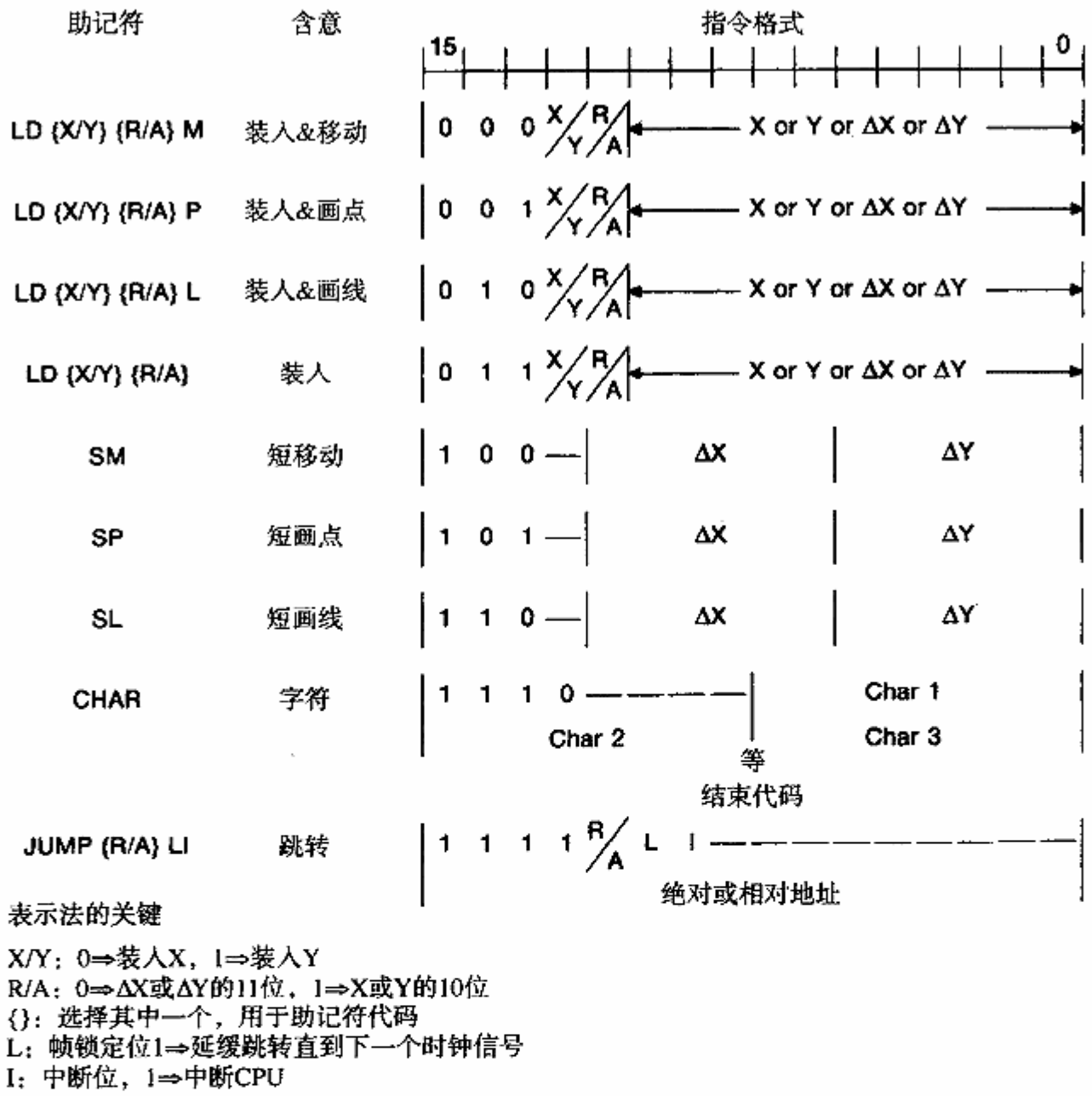
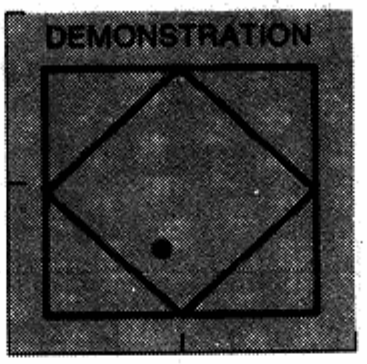


图4-32 随机扫描显示系统的指令集

注意，设置了这个指令后，只需要一个load命令（助记符为LD）就可以绘制或者垂直、水平移动图像。因为其他坐标都是固定的。但是间接的移动需要两个load命令，这两个命令可以先*x*再*y*，也可以先*y*然后*x*，两个中的第二个通常指定移动、画点或者画线操作。字符命令（助记符为CHAR）后面跟随一个字符串，最后以一个非显示字符终止代码结束。 186

DPU指令和通用计算机中使用的指令集有两个主要的区别。除了JUMP指令以外，其他所有的指令都是特定用途的，寄存器可以装入或者加上某个数，但是不能保存结果，寄存器值控制CRT电子束的位置。第二个区别是，除了JUMP指令以外，所有数据都是立即数，即它们是指令的一部分。LDXA 100意味着“把100这个值装入X寄存器”，而不是像计算机指令中那样“把地址为100的内存中的内容装入到X寄存器”。在第18章中描述的一些更高级的DPU中已经去掉了这个限制。

一般的随机处理器和光栅处理器指令集只有很小的差别，随机处理器没有区域填充、位操作和查找表的命令，但是因为有指令计数器，随机显示处理器有控制转移的命令，随机扫描显示器可以在比光栅扫描显示器更高的分辨率下工作，能够绘制出平滑的没有锯齿边的直线，最快的随机显示器可以在一个刷新周期里绘制大约100 000个短向量，允许极度复杂形状的实时动画。




```
SQUARE:   LDXA      100      准备好画正方形
          LDYAM     100      移到(100, 100)
          LDXRL     800      画线到(900. 100)
          LDYRL     700      画线到(900, 800)
          LDXRL    -800      画线到(100, 800)
          LDYRL    -700      画线到(100, 100)，正方形的起点
POINT:    LDXA      300
          LDYAP     450      在(300, 450)画点
DIAMOND:  LDXA      100
          LDYAM     450      移到(100, 450)
          LDXR      400
          LDYRL    -350      画线到(500, 100)
          LDXR      400
          LDYRL     350      画线到(900, 450)
          LDYR      350
          LDXRL    -400      画线到(500, 800)
          LDXR     -400
          LDYRL    -350      画线到(100, 450), 菱形的起点
TEXT:     LDXA      200
          LDYAM     900      移到(200, 900)画文字
          CHAR      'DEMONSTRATION t'   t是终止代码
          JUMPRL    SQUARE   重新生成图像，锁住帧
```

187 图4-33 随机扫描显示处理器的程序

4.6 用于操作者交互的输入设备

在本节中我们描述最普通的输入设备的工作。我们简单并高层次地讨论可用的设备是如何工作的。在第8章我们将讨论各种输入设备的优缺点，还描述一些更先进的设备。

我们的介绍是围绕着*逻辑设备*（logical device）的概念进行组织的，逻辑设备在第2章中已有介绍，并在第7章和第8章中进行更深入的讨论。有五种基本逻辑设备：*定位设备*，用来指明位置或者方向；*拾取设备*，用来选择显示的实体；*定值设备*，用来输入一个实数；*键盘*，用来输入字符串；*选择设备*，用来选择可能的行为或者选项集合中的一个或者多个。根据设备提供给应用程序的信息种类，逻辑设备的概念定义了这些输入设备的等价分类。

4.6.1 定位设备

1. 输入板

输入板（或数据输入板）是一个平板，其尺寸从6英寸×6英寸到48英寸×72英寸或者更大。它可以探测用户手中可移动触笔或者手持光标定位器的位置。图4-34显示了一个同时具有触笔和光标定位器（我们今后将主要只提及触笔，尽管讨论对两者相关）的小输入板。大部分输入板使用一种电感应的机制来确定触笔的位置，在一种设计中，网格宽度为1/4英寸或者1/2英寸的矩形网格线嵌在输入板的表面，沿金属线生成电磁脉冲系列，激励触笔中的线圈，感应出电信号，每个脉冲感应出来的电信号的强度可以用来确定触笔的位置。这个信号强度还粗略地用于确定触笔或者光标距离输入板有多远（“远”、“近”（比如说离数据板大约1/2英寸）或者