[DirectX中文手册](http://www.cnblogs.com/skyofbitbit/p/3869780.html" \t "_blank)

第一章 DirectX基础（初级篇）

第一节什么是DirectX？

一、什么是DirectX

微软的DirectX软件开发工具包（SDK）提供了一套优秀的应用程序编程接口（APIs），这个编程接口可以提供给你开发高质量、实时的应用程序所需要的各种资源。DirectX技术的出现将极大的有助于发展下一代多媒体应用程序和电脑游戏。

总的说来，使用DirectX的主要有两个好处：

为软件开发者提供硬件无关性；

为硬件开发提供策略。

    1、为软件开发者提供硬件无关性

微软开发DirectX，其最主要的目的之一是促进在Windows操作系统上的游戏和多媒体应用程序的发展。在DirectX出现以前，主要的游戏开发平台是MS-DOS，游戏开发者们为了使他们的程序能够适应各种各样的硬件设备而绞尽脑汁。自从有了DirectX，游戏开发者们便可以获益于Windows平台的设备无关性，而又不失去直接访问硬件的特性。DirectX主要的目的就是提供象MS-DOS一样简洁的访问硬件的能力，来实现并且提高基于MS-DOS平台应用软件的运行效果，并且为个人电脑硬件的革新扫除障碍。

另一方面，微软公司开发DirectX是为了在当前或今后的计算机操作系统上提供给基于Windows平台的应用程序以高表现力、实时的访问硬件的能力。DirectX在硬件设备和应用程序之间提供了一套完整一致的接口，以减小在安装和配置时的复杂程度，并且可以最大限度的利用硬件的优秀特性。通过使用DirectX所提供的接口，软件开发者可以尽情的利用硬件所可能带来的高性能，而不用烦恼于那些复杂而又多变的硬件执行细节。

一个高表现力的基于Windows平台的游戏将得益于以下几种技术：

专为提高图形运算及快速反应能力而设计的加速卡（Accelerator cards）

即插即用以及其它Windows软硬件

内建于Windows的通信服务, 包括DirectPlay

    2、为硬件开发提供策略

DirectX的另外一个重要的目的是给硬件厂商提供开发策略，他们可以从高性能程序的开发者和独立的硬件供应商（independent hardware vendors IHVs）那里得到反馈。所以，在DirectX 程序员参考书中有时可能会提供那些还不存在的硬件加速设备的技术细节。在很多时候，软件可以模拟这些特性，在另外一些情况下，软件根据硬件的指标判断出其特性，并且可以忽略那些硬件并不支持的性能。

已经和将要实现的显示设备的特性包括：

——覆盖(Overlays)，由于它的被支持，在图形设备接口中，窗口中的换页（page flipping）将成为可用。换页是用来在整个屏幕上显示画面的双缓冲（double-buffer）方案。

——精灵引擎(Sprite engines)，使精灵（不规则图形）覆盖更容易。

——插补延展(Stretching with interpolation)，它可以更有效的保存显示内存，因为它可以使小幅画面延展到整个屏幕。

——Alpha 融合(Alpha blending)，它可以在硬件像素层（hardware-pixel level)上混合颜色。

——带有透视修正（perspective-correct)贴图的3D加速器，它允许你在3D表面上贴图。比如，你可以在用3D软件制作的城堡的走廊里贴上砖块的位图来显示出透视效果。

为3D图象进行的位操作将 Z 方向考虑在内。

标准2兆显示内存，这在3D游戏中是最小需求。

压缩标准，这将允许你在显存中储存更多的信息。这个标准不论是在软件还是硬件中执行都会相当快。它将被用在贴图中并且包含透明压缩。

将要被包括的声音设备的新特性包括：

硬件及其外设可以提供空间环绕立体声效果。

声卡  上内置音频内存。

音频-视频一体卡可共享其上的内存。

另外，视频回放将得益于今后的与DirectX相兼容硬件加速设备。今后的DirectX版本中的一个特性是将支持硬件加速YUV视频解码。

二、DirectX的组件

DirectX SDK为基于Windows平台的应用程序提供了以下几个组件。

DirectDraw ：通过直接访问显示硬件来提供高级的图象处理能力。

DirectSound ：它提供了软硬件的低延迟声音混频（low\_latency sound mixing）和回放（Playback），硬件加速，以及直接访问音频设备的能力。

DirectPlay ：它明确的提供了通用环境连接能力（generalized communication capabilities），来简化你应用程序之间的通讯服务。

Direct3D ：它为主流的桌上型计算机和Internet用户提供实时的、交互的3D技术。

DirectInput ：它简化你的应用程序访问鼠标、键盘和操纵杆设备的能力。

DirectSetup ：一套简单的API向你提供安装DirectX部件的功能。

AutoPlay ：它也是Win95操作系统的一个特性，当你在光驱内放上光盘，指定的应用程序会自动执行。

三、关于DirectDraw

DirectDraw是DirectX SDK大家族中的一员，也是其中最主要的一个部件。DirectDraw允许程序员直接的操作显存、硬件位图映射以及硬件覆盖和换页技术。它在提供这些功能的同时，也使其与现在的基于Microsoft Windows的应用程序和设备驱动程序相兼容。

DirectDraw是一个软件接口，它在提供直接访问显示设备的同时，与Windows图形设备接口（GDI）相兼容。DirectDraw不是一个高层的图形程序编程接口，它为游戏和Windows子系统软件（例如：3D图形包和数字视频编码）提供了一种与设备无关的途径，以获得访问特定的显示设备的某些高级特性的能力。

DirectDraw适用于种类众多的的显示设备，从简单的SVGA显示器到提供裁剪、缩放、和支持非RGB颜色格式的高级硬件实现设备。设计这样的接口是为了让你的应用程序能够列举低层硬件的能力，并且对那些支持的硬件加速特性加以利用。那些在硬件设备中不能实现的特性，DirectX将仿真出来。

DirectDraw提供了以下几个优点，这些好处在以前只有那些专为特定显示设备所写的软件才能利用。

支持双缓冲和换页图形

访问、控制显示卡的位图映射

支持3D z-buffers （z缓存）

支持z方向（z-ordering）硬件辅助覆盖。

访问图形缩放硬件

仿真访问标准的和增强的显示设备内存空间

DirectDraw的任务是用与设备无关的途径来提供依赖于设备的访问显示内存的方法。本质上，DirectDraw管理显示内存。你的应用程序只需要懂得那些一般的关于硬件与设备有关的知识，比如RGB和YUV色彩格式和两条光栅线之间的pitch（宽距） 。在需要利用位转换或操作调色板寄存器时，你不需要为调用过程中的细节而烦恼。使用DirectDraw，你可以方便的操作显示内存，充分的利用不同类型的显示设备的位转换和颜色压缩能力，而不需要依赖于某一个特定的硬件。

DirectDraw给运行于Windows 95和Windows NT 4.0或更高版本的计算机提供了一个高性能的的游戏图象引擎。

四、为什么要使用DirectDraw

毋庸置疑，人们之所以利用DirectDraw来开发各种游戏以及多媒体应用软件，是因为DirectDraw可以为他们的应用程序带来许多强大的功能提升。

DirectDraw可以充分评估视频硬件的能力，只要可能，它就会对其某一特性加以利用。例如，如果你的显卡支持硬件Blit，DirectDraw就会将位图映射这一操作分派给显卡来完成，极大的提升运行速度。此外，当某硬件不支持某项特性时，DirectDraw还提供了硬件仿真层（HEL）以完成这项操作。

DirectDraw的硬件抽象层（HAL）提供了一个统一的接口，通过它，开发者可以直接的操作显示存储器和视频存储器，从系统硬件中获取最大的表现能力。

DirectDraw运行于Windows95操作系统之上，从系统所提供的32位内存寻址和平面内存模型中获益。DirectDraw将视频和系统存储器视为整块的空间，而不是碎片的集合。如果你曾使用过区段偏移寻址，你将很快就喜爱上这种“平面”内存模型。

对于全屏模式的应用程序，DirectDraw使得多后台缓存的换页操作变得极为容易。

支持窗口和全屏模式应用程序的裁减。

支持3-D z缓存。

支持带z轴方向的硬件辅助覆盖。

可访问图象缩放硬件。

可同时访问标准的和增强的显示设备内存区。

其它的特性，包括动态改变调色板、独占访问硬件、和分辨率切换等等。

合理和有效的利用DirectDraw的这些特性将使开发者很容易就能够写出比基于标准的Windows GDI应用程序，甚至MS-DOS应用程序还要优秀的作品。

五、DirectX 5.0的新特性

DirectX5.0版本比3.0版拥有更多的创新和新特性（注意：并不存在4.0版）。尽管这样，你用以前版本写的DirectX应用程序同样可以不加修改的运行于DirectX5.0环境中。以下将介绍的是DirectX 5.0中DirectDraw接口的新特性。

首先，DirectDraw支持Windows98和Windows NT 5.0的多显示器系统。例如，在多显示器环境下，同一台电脑使用两套不同的显示设备和显示器，用户可以将不同的画面显示在两个显示屏幕上，甚至将一个窗口从一个显示器拖拽到另一个显示器上。DirectDraw5.0支持这样一个系统，允许应用程序在这样的环境中直接的访问和操纵硬件加速特性。

其次，DirectDraw已经使视频端口（Video-port）具有新的性能，它允许应用程序直接控制数据流从一个硬件视频端口流向一个位于显存中的DirectDraw页面上。这样，使得视频图象的输入和输出对DirectDraw来说都变得极为便利和快速。

       除此之外，DirectDraw的HEL（硬件仿真层）现在可以充分利用Pentium MMX处理器所带来的多媒体性能的提升。当你第一次创建页面的时候，DirectDraw会测试你的电脑芯片是否支持MMX。而在一台非Pentium MMX级别电脑上，这个测试会在调试程序的时候给出一个良性的异常信息，而这个异常并不会影响到你的应用程序执行的稳定性。

       DirectDraw接口现在还支持离屏页面的宽度大于主页面的宽度。你可以创建这些页面而不用顾及它的大小，而在以前的版本中，你经常会为了不使页面宽度大于主页面而将其分割开来。

DirectDraw接口现在已经能够支持高级图形端口（AGP）构架。在装备了AGP显卡的系统中，DirectDraw程序开发者将从AGP显卡的大容量和高传输速率中获益。

六、部件对象模型（COM）

DirectX是基于COM的一套软件编程接口。

部件对象模型（Component Object Model, COM）是OLE 的基础。COM 为OLE提供了编程模型和二进制标准。COM定义并实现了软部件（如应用程序、数据对象、控件及服务）机制，并把它们统称为“对象”。每个软部件对象由数据以及访问数据的函数组成，访问软部件对象数据的函数的集合称为“接口”。

从这里可以看出，COM 的设计与C++类非常相似，即一个软部件对象具有一个内部数据结构和一组外部接口函数，允许通过接口函数对数据进行访问。因此Microsoft公司把根据COM执行的对象统称为Windows对象。但Windows对象与C++中的对象有明显的差别，Windows对象中没有公共数据，也没有成员函数，因此不能直接访问数据，也就是说数据是全封装的。

对象的提供者或服务器必须指明一个或多个接口的定义，每个接口都是相互关联的一组函数，执行对象的一个特性。每个对象的一个用户（或称为“客户”）必须拥有一个接口指针才能访问对象。当客户有了这个指针后，就可以使用这个对象而无需知道对象的含义，即使客户运行在不同的进程、不同的机器、不同的操作系统、由不同的软件开发而使用不同的语言，或者版本不同。

DirectX SDK接口被创建在COM编程层次表中很基础的一层。每一个代表设备的对象的接口，比如IDirectDraw、IDirectSound、和IDirectPlay，是直接从IUnknown OLE接口中派生下来的。这些基本对象的创建被操作于在该对象的动态连接库中，要比用Win32中专门用来创建COM对象的CoCreateInstance函数要好得多。

特别的是，DirectX SDK对象模型为每一个设备提供一个主要的对象。其它的设备对象是由这个主要对象派生下来的。例如，DirectDraw对象代表显示设备。你可以用它来创建代表显存的主页面（DirectDrawSurface）对象，和代表硬件调色板的调色板（DirectDrawPalette）对象；相似的，DirectSound对象代表声卡，并且创建DirectSoundbuffer对象来代表声卡上的声音数据。

七、自我检测

本章所讲的内容都是关于DirectX的较为抽象的概念。其实，在你真正开始自己编制程序，进入这一领域之前，有些概念现在还是很难接受的。但是，当你在有了一定的实践经验，在回过头看这些文字，你会发现它们其实还是很好理解的。实践出真知，掌握DirectDraw知识的最佳途径只有一个，那就是“编程”。在下一章里，我们将正式开始DirectDraw编程的学习。

通过这一章的学习，你应该对以下内容有所了解：

DirectX是做什么用的？

DirectX包括哪些组件，以及它们各自的作用。

DirectDraw的好处是什么，我们为什么要使用它。

COM的一般解释。

第二节如何安装和使用DirectX

一、编译库和运行库

为了能够使用MS VC++5.0进行DirectX编程，必须拥有一套DirectX的SDK（软件开发工具包），它包括了编译DirectX应用程序所需要的编译库文件（\*.lib）、头文件（\*.h）、示例，还有帮助。MS VC++5.0自己带有3.0版的DirectX SDK，所以只要按照缺省的安装，你的VC就已经可以编制基于DirectX的应用程序了。不过，缺省的安装并不会把DirectX的在线帮助包括在内，没有这个功能强大的在线帮助，你常常会陷入孤立无援的境地。这一章将教你如何安装一个全功能的MS VC++5.0 DirectX编译环境。

如果你有DirectX5.0版或更高版的SDK，本章还将教你如何更新旧版的DirectX编译环境。且慢，如果你辛辛苦苦编了一个漂亮的动画程序，兴高采烈的拷给你的朋友，第二天他们却告诉你根本就运行不了，那将是何等的尴尬与无奈。原来，要让DirectX应用程序能在脱离VC的环境下执行，你的电脑还必须安装有DirectX运行库（Runtime library），这也就是我们常说的DirectX引擎（Engine）或驱动程序（Device Driver）。好在Microsoft为了推行其DirectX标准，将这套驱动程序库免费奉送，你可以从Microsoft公司的Web站点的DirectX页上下载，当然，这显然是个很愚蠢的方法，因为，在很多高速图形游戏如Need For Speed（极品飞车）、Motor Racer（摩托英豪）、Diablo（暗黑破坏神）、Red Alert（红色警戒）等中，都有DirectX驱动程序提供。安装驱动程序只用运行Setup文件就可以了，它默认的包括了在Windows注册表中注册DirectX部件，并且将动态连接库文件（DLL）复制到Windows系统目录中。只不过，这些驱动程序包只包括了DirectX引擎，并无编译DirectX程序所需要的库文件和头文件。

二、安装Visual C++ 5.0

1、电脑配置

本教程所有的DirectX程序都是在Visual C++ 5.0环境中编译完成的，所以，建议用户安装Visual C++ 的5.0版本，请不要使用以前的VC版本，虽然程序可以正常运行，但也会发生不可预知的错误。而且，VC5.0以其强大的功能和基于超文本（HTML）的在线帮助，定会使你[事半功倍](https://www.baidu.com/s?wd=%E4%BA%8B%E5%8D%8A%E5%8A%9F%E5%80%8D&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，大大提高编程效率。

安装Visual C++5.0的DirectX所需要的软、硬件最低配置：

Windows95或WindowsNT 3.0；

IBM PC 极其兼容机，最好具有80486以上的微处理器；

8MB内存；

最小安装需要140MB的可用硬盘空间，典型安装需要200MB的可用硬盘空间，CD-ROM安装需要50MB的可用硬盘空间，完整安装需要300MB的可用硬盘空间；

高密软盘驱动器；

VGA显示器；

CD-ROM驱动器。

以上是Microsoft所给出的最低配置，但在实际情况下，如果你的电脑还是一台486、8MB内存，甚至奔腾100、16MB内存的配置，你完全没有理由犹豫是继续使用这台老牛拉破车的电脑呢，还是该为自己重新配一台（如果你的钱包够鼓的话）。以下是为了更加突出的体现DirectX的硬件加速特性，建议读者至少有以下配置：

Windows95、Windows98或WindowsNT 4.0；

奔腾166、AMDK6-166或Cyrix166以上的微处理器，PII或AMDK6-3D尤佳；

至少16MB内存；

SVGA显示器，至少支持800\*600\*16K，刷新率75HZ以上；

PCI或AGP显示卡，至少2MB显存；

16位声卡；

至少4速CD-ROM驱动器；

拥有一块VOODOO2或G200图形加速卡（鉴于其不菲的价格，这并不是必选配件）。

2、开始安装

运行Visual C++5.0的安装光盘上的Setup.exe文件。

弹出“Welcome”对话框，单击“Continue”按钮，安装程序显示“Registration”对话框，要求用户输入姓名（User）、公司名称（Organization）和10位的CD号。

正确输入注册信息后单击“Continue”按钮，从弹出的画面中单击“Microsoft Visual C++ 5.0”，再从弹出的画面单击“Install Visual C++ 5.0 Enterprise Edition”。

稍后弹出“Welcome”对话框，单击“Next”按钮，安装程序显示软件授权协议。

单击“Yes”按钮，安装程序显示“Installation Options”对话框，可以对Visual C++5.0的安装路径和安装类型进行设置，选择用户定制（Custom）型安装。

单击“Next”按钮，出现如下对话框，如图选择要安装的部件，最后选中“Books Online”项，再单击“Details”，以安装DirectX帮助文件。

在出现的以下对话框中，选中“Win32SDK”,其它项也如图选择。

接着，单击“Next”按钮，安装程序开始安装Visual C++5.0，并显示进度画面。

安装完毕后，请重新启动Windows95。

三、安装DirectX5.0的SDK

很多地方都提供DirectX的SDK，你可以在某些三维游戏的光盘上找到，也可以直接从Microsoft的Web站点上下载（http:\\www.microsoft.com）。

由于本教程的所有示例都是用DirectX5.0编译成功的，所以读者最好也使用DirectX的5.0版。安装方法如下：

运行SDK中的Setup.exe文件，出现Welcome窗口，按Next按钮，出现软件许可协议窗口，按YES按钮。此时出现Setup Type对话框，有三种安装选择，选择其中的Compelete（完全安装），然后按Next。

在出现的下一个对话框中，在Retail or Debug Runtime选项中选择Debug，在Force Install中做上打勾标记，如下图。按Next按钮。

在这里有两个安装选择：一个是Retail（零售）安装，推荐给仅用于单独运行游戏的用户，并不会帮助程序员在编译过程中发现错误；另一个是Debug（调试）安装，推荐给开发者，运行虽然较前者稍慢，但可以帮助调试程序。选中Force Install，表示将覆盖这台电脑上的任何更新的DirectX版本。

在出现的对话框中，让你选择是否安装IE和ActiveMovie，读者请随意。然后按Next按钮。

在余下的步骤中，让你选择安装路径和快捷方式夹名称，按Next即可。设置完毕后，安装程序开始正式安装直至结束。

安装完毕后，你将在你的SDK路径中看到以下目录：

SDK：软件开发工具包。

INC：头文件

LIB：库文件。\*.LIB，用于VC++编译器；\*.LBW，用于Watcom编译器

SAMPLES：例程源代码

BIN：例程的可执行文件

DOCS：帮助和Readme文件。

FOXBEAR：Fox & Bear演示游戏Demo。

ROCKEM：Direct3D演示游戏Demo。

DXBUG：DirectX 调试报告工具。

可以看到，一个完整的SDK已经安装至你的电脑。但是为了让VC能够调用到SDK中的头文件和库文件，你还要做以下的工作以打通路径。有两种方法供你选择：

方法一：将sdk\inc和sdk\lib目录中所有的文件分别复制到VC的include和lib目录中去。这是最简单也是最有效的方法。

方法二：打开VC，选择Tools菜单的Options选项，在Include files列表的开头添加“c:\dxsdk\sdk\inc”，再在Library files列表的开头添加“c:\dxsdk\sdk\lib”。这是微软所推荐的方法。

必须把这两个路径加在最开头是因为：VC在寻找头文件和库文件时是按照列表中的先后顺序，VC以前所打通的路径中已经包含了DirectX3.0的头文件和库文件，如果把这两个路径加在最后，VC在编译时所使用的仍是旧版的DirectX 3.0的文件。

好了，一切就绪，DirectX已经在向你招手。如果你想赶紧实验你的第一个程序，可以跳过下面的部分，直接进入下一章。

四、DirectX5.0文件说明

如果按照缺省路径安装，DirectX将在C:\Program Files\directx\setup目录中加入三个可执行文件：DxInfo.exe、DxSetup.exe、DxTool.exe。对那些谙熟DirectX的人来说，运行这些个文件并加以评价和指点，是很容易在同事中建立起高手形象的。这三个文件用来透视和评价电脑硬件系统2D、3D、音频等加速性能的高低。

DxInfo.exe：显示计算机的软、硬件信息极其驱动程序文件信息，从这里你可以了解到你的计算机的显卡、声卡、芯片、操作系统和DirectX各组件的驱动程序和特性。

DxSetup.exe：DirextX安装程序，可以设置是否启用Direct3D硬件加速以及还原音频、显示器驱动程序。

DxTool.exe：这是DirectX中最有用的一个文件，从它你可以对你的电脑的DirectX硬件加速特性有一个全面而深入的了解，还可以设置是否启用Direct3D硬件加速以及DirectDraw硬件加速。

这三个文件随着版本不同，也会与其上所述不尽相同。

五、卸载DirectX

由于DirectX驱动程序自己不带反安装程序，所以一旦安装之后，是无法将其卸载的，只有一步一步往上升级，否则只能重新安装Windows95，这就象一旦你选择了一条道路，就要贯穿始终的走下去一样。试着打开“控制面板”里的“添加/删除程序”，会发现“DirectX驱动程序”赫然其中，小心翼翼的按过“删除”按钮，弹出的却是DirextX的安装程序，你始终无法用常规的方法将其卸载。

是不是就无能为力了呢？当然不是，本站的软件下载中提供了一个反安装DirectX的软件，可以用来删除你的电脑上的DirectX（它只是卸载DirectX驱动程序，并不会卸载其SDK）。目录位于光盘：\dxuninstall。

应该注意的是，由于某些显卡和声卡，几乎包括现在出的所有显卡或声卡，它们的驱动程序中都已经自带了DirectX驱动程序，所以，除非你重新启动Win95，切换到安全模式下，再运行DirectX的卸载程序，否则，你始终无法将DirectX完全卸载。

第三节一个DirectDraw入门程序

一、一个小测验

在正式进入本章主题之前，先对读者进行一次小小的入学考试，不用紧张，其实几道题都非常的简单。

1、根据你对左边这三个函数的直观感觉，选出它们的正确对应关系。

WinMain()         a、初始化窗口

InitWindow()       b、处理Windows消息

WinProc()          c、应用程序入口

2、找出与其它三个没有共同点的一个。

a、HINSTANCE     b、HWND

c、HBITMAP         d、HELLO

3、HWND之于窗口，相当于什么之于苹果。

a、苹果皮     b、苹果核

c、苹果把儿     d、整个苹果

4、如果要创建一个最普通的窗口，应该用以下的哪一个标志。

a、WS\_OVERLAPPEDWINDOW     b、WS\_STRANGE

c、WS\_BEAUTIFUL              d、WS\_UGLY

如果你有一道题做错了，说明你对Win32编程还不是十分了解，那么你需要事先预习一下。请跳转到本教程的“Win32编程基础知识”一章，学习一下Win32编程的基础知识。（这四道题的答案分别是：cab、d、c、a。其实它们真的是非常简单，只要你仔细阅读一下题目，就是猜也能猜出来。）

如果这些题对你来说不成问题，祝贺你，你可以继续本章的内容了。

二、牛刀小试

只要是介绍编程的书，似乎有一个不成文的规定，即第一个例子由“Hello World”开始，本教程也不例外。那么，如果你早已迫不及待想初尝DirectDraw程序编译成功后的“0 error(s), 0 warning(s)”的喜悦，就让我们开始吧！

在下面的例子中，我们将利用Visual C++5.0来生成一个简单的DirectDraw应用程序。程序的创建将不使用方便的MFC（Microsoft Foundation Class Library，微软的C++基础类库）向导，而是使用最原始的Win32 应用程序开发环境。熟悉VC++的读者可能会问，为什么舍先进的MFC工具不用，而去使用最原始的方法呢？这是因为，MFC主要是用于基于窗口和文档的应用软件的编程，它集成了大量的数据和方法，将许多烦琐的任务，如：应用程序初始化、文档处理、磁盘IO封装起来，虽然这样可以给你的编程带来了极大的便利，但是在你编制基于图形显示和多媒体的应用程序的时候，这却会给你带来极大的麻烦。首先，你无法触及系统的内核，如：你需要自己来处理每一个消息循环时，而MFC并没有为你留出这样一个接口；而且，MFC为你事先建好的类，它们的许多功能对你来说是没用和低效率的，使用它们只会给你的程序带来冗余和不便。

总之，MFC为你隐藏了太多技术细节，而DirectDraw编程需要系统对于开发者具有一定的透明度。 所以，在大多数情况下，我们用最基本的Win32应用程序开发环境来开发我们的DirectDraw应用程序，本教程中几乎所有的例程都是使用Win32开发环境。当然，这并不是说用MFC就不能编制基于DirectDraw的应用程序了，它也是可以的，这将在本教程的“用MFC创建DirectDraw应用程序”一章中做介绍。

使用Win32开发环境表明，你必须从WinMain（）开始编程，自己写每一个消息的处理程序，这的确是一项很繁重的工作。但是当你理解和熟悉了这一套方法时，你会发现它其实是相当直观和容易的。

打开Visual C++ 5.0。

选择File菜单的New，在出现的对话框中，选择Projects栏目（新建工程），并点取其下的Win32 Application项，表示使用Win32环境创建应用程序。先在Locatin（路径）中填入“c:\”，然后在Project Name（项目名称）中填入“Hello”，其它按照缺省设置，使对话框如图所示。单击OK按钮。

此时，一个基于Win32的工程已经创建完毕，但是它还没有包括任何文件。你需要新建一个C++文件增加到工程中。

再次选择File菜单的New，在出现的对话框中，选择Files栏目（新建文件），并点取其下的C++ Source File项，表示新建一个C++源文件。在右边的File栏中输入“Hello”，最后确定让Add to project检查框打上勾，使整个对话框如图所示。单击OK按钮。

在Hello.cpp文件中输入以下源程序代码，当然，你最好的做法是将以下的代码复制到你的文件中去，确保能用。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 工程：hello

// 文件：hello.cpp

// 内容：创建第一个DirectDraw应用程序，

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <windows.h>

#include <windowsx.h>

#include <ddraw.h>

LPDIRECTDRAW lpDD; // DirectDraw对象

LPDIRECTDRAWSURFACE lpDDSPrimary; // DirectDraw主页面

char szMsg1[] = "Hello World, I am DirectDraw boy !";

char szMsg2[] = "按 ESC 退出";

//函数声明

LRESULT CALLBACK WinProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam );

BOOL InitWindow( HINSTANCE hInstance, int nCmdShow );

BOOL InitDDraw( void );

void FreeDDraw( void );

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//函数：WinMain()

//功能：Win32应用程序入口函数。进行初始化工作，处理消息循环

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int PASCAL WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,

                      LPSTR lpCmdLine, int nCmdShow)

{

    MSG msg;

    //初始化主窗口

    if ( !InitWindow( hInstance, nCmdShow ) )

        return FALSE;

    //初始化DirectDraw环境，并实现DirectDraw功能

    if ( !InitDDraw())

    {

        MessageBox(GetActiveWindow(), "初始化DirectDraw过程中出错！", "Error", MB\_OK );

        FreeDDraw();

        DestroyWindow(GetActiveWindow());

        return FALSE;

    }

    //进入消息循环

    while (GetMessage(&msg, NULL, 0, 0))

    {

        TranslateMessage(&msg);

        DispatchMessage(&msg);

    }

    return msg.wParam;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//函数：InitWindow()

//功能：创建主窗口。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static BOOL InitWindow( HINSTANCE hInstance, int nCmdShow )

{

    HWND hwnd; //窗口句柄

    WNDCLASS wc; //窗口类结构

    //填充窗口类结构

    wc.style = 0;

    wc.lpfnWndProc = WinProc;

    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;

    wc.hInstance = hInstance;

    wc.hIcon = LoadIcon( hInstance, IDI\_APPLICATION );

    wc.hCursor = LoadCursor( NULL, IDC\_ARROW );

    wc.hbrBackground = GetStockObject(BLACK\_BRUSH);

    wc.lpszMenuName = NULL;

    wc.lpszClassName = "dxHello";

    //注册窗口类

    RegisterClass( &wc );

    //创建主窗口

    hwnd = CreateWindowEx(

    0,

    "dxHello",

    "",

    WS\_POPUP,

    0, 0,

    GetSystemMetrics( SM\_CXSCREEN ),

    GetSystemMetrics( SM\_CYSCREEN ),

    NULL,

    NULL,

    hInstance,

    NULL );

    if( !hwnd ) return FALSE;

    //显示并更新窗口

    ShowWindow( hwnd, nCmdShow );

    UpdateWindow( hwnd );

    return TRUE;

    }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//函数：WinProc()

//功能：处理主窗口消息

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LRESULT CALLBACK WinProc( HWND hWnd, UINT message,

                             WPARAM wParam, LPARAM lParam )

{

    switch( message )

    {

        case WM\_KEYDOWN://击键消息

            switch( wParam )

            {

                case VK\_ESCAPE:

                    PostMessage(hWnd, WM\_CLOSE, 0, 0);

                    break;

            }

            break;

        case WM\_DESTROY://退出消息

            FreeDDraw();

            PostQuitMessage( 0 );

            break;

    }

    //调用缺省消息处理过程

    return DefWindowProc(hWnd, message, wParam, lParam);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//函数：InitDDraw()

//功能：初始化DirectDraw环境并实现其功能。包括：创建DirectDraw对象，

// 设置显示模式，创建主页面，输出文字。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL InitDDraw(void)

{

    DDSURFACEDESC ddsd; //页面描述

    HDC hdc; //设备环境句柄

    //创建DirectCraw对象

    if ( DirectDrawCreate( NULL, &lpDD, NULL ) != DD\_OK ) return FALSE;

   // 取得独占和全屏模式

    if ( lpDD->SetCooperativeLevel( GetActiveWindow(),

                    DDSCL\_EXCLUSIVE | DDSCL\_FULLSCREEN ) != DD\_OK)

        return FALSE;

    //设置显示模式

    if ( lpDD->SetDisplayMode( 640, 480, 8 ) != DD\_OK) return FALSE;

    //填充主页面信息

    ddsd.dwSize = sizeof( ddsd );

    ddsd.dwFlags = DDSD\_CAPS;

    ddsd.ddsCaps.dwCaps = DDSCAPS\_PRIMARYSURFACE;

    //创建主页面对象

    if ( lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL ) != DD\_OK)

        return FALSE;

    //输出文字

    if ( lpDDSPrimary->GetDC(&hdc) != DD\_OK) return FALSE;

    SetBkColor( hdc, RGB( 0, 0, 255 ) );

    SetTextColor( hdc, RGB( 255, 255, 0 ) );

    TextOut( hdc, 220, 200, szMsg1, lstrlen(szMsg1));

    TextOut( hdc, 280, 240, szMsg2, lstrlen(szMsg2));

    lpDDSPrimary->ReleaseDC(hdc);

    return TRUE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//函数：FreeDDraw()

//功能：释放所有的DirectDraw对象。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void FreeDDraw( void )

{

    if( lpDD != NULL )

    {

        if( lpDDSPrimary != NULL )

        {

            lpDDSPrimary->Release();

            lpDDSPrimary = NULL;

        }

        lpDD->Release();

        lpDD = NULL;

    }

}

为了简化代码，这第一个入门程序没有头文件。在进行编译之前，还得进行最后的设置。选择Project菜单的Settings…，出现工程设置对话框。选择Link栏，在Object/Library modules中添入“Ddraw.lib”。使对话框如图所示。

请注意：这一步骤是将DirectDraw的静态连接库文件连接到工程中，否则，程序虽然可以正常编译，但是在连接时会产生一个“unresolved external symbol”（没有定义的外部符号）的错误。在以后所有的DirectDraw程序中，都必须将与你所用到的DirectDraw组件相应的静态连接库添入到这个设置中。

至此，一个最基本的DirectDraw应用程序已创建完毕，你现在不必去深究这些代码的含义，在下面及以后的章节中我们会对它们进行详细的分析。这虽不是一个最简单的DirectDraw应用程序，但它确实是一个能够实现最基本的输出功能的DirectDraw程序。

按F7编译成功后，按Ctrl+F5，执行该程序，显示器将切换到640\*480\*256色模式，黑屏后，屏幕中央会打印出蓝底黄字“Hello World, I am DirectDraw boy !”，除了输出这些字符外，这个程序什么也不做。按ESC可退出程序。程序运行结果如下图

这就是DirectDraw？有的人也许会对DirectDraw感到很失望，因为它并没有为我们表现出神奇的功能啊？但有的人却会对此感到异常兴奋，他们觉得一扇充满诱惑的房间的大门正向他们打开。这就象透过天窗，有些人只会看到黑暗的夜空，有些人却能看见满天的星星一样。第一个例子，为了使程序不至于太长而让那些初学者[望而生畏](https://www.baidu.com/s?wd=%E6%9C%9B%E8%80%8C%E7%94%9F%E7%95%8F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，所以只能一再简化（尽管这样，整个程序还是占用了相当大的篇幅）。在后续章节的例子中，你们会看到程序会一个比一个更精彩。

三、分析代码

下面，让我们来逐一分析一下这个程序。

1）程序结构

分析程序应该是一个由外而内，逐步求精的过程。首先从大的方面来看，这个程序一共用到了五个函数，如果按照正常顺序，排除程序中出错的可能，它们的调用顺序依次是这样的。

WinMain ----> InitWindow ----> InitDDraw ----> WinProc ----> FreeDDraw

WinMain：所有Win32应用程序的入口函数，它也是应用程序关闭时的出口，一个应用程序的全生命周期就是在它的控制之下。所以，确切的说，其它四个函数是被包括在WinMain之内的。消息循环也是在这个函数中启动。

InitWindow：初始化和创建一个与程序的HINSTANCE（实例句柄）相关联的主窗口，这个窗口的HWND（窗口句柄）在初始化DirectDraw环境时需要用到。

InitDDraw：初始化和创建DirectDraw对象，并执行一定的功能。它里面包括了创建DirectDraw对象，创建页面，设置显示模式，创建主页面，输出文字。

WinProc：是应用程序感知外来动作和产生反应的神经中枢，相当于人的大脑。这是程序中最主要的部件之一，它和在WinMain中所启动的消息循环是一起工作的。

FreeDDraw：释放DirectDraw的各种对象，以使其不再占用内存空间。

上面这段话如果使你感到迷惑，就象是在你小学的时候有人给你讲什么是微积分，那么你仍需要事先预习一下Win32编程。请跳转到本教程的“Win32编程基础知识”一章，复习一下Win32编程的基础知识。

如果你对Win32编程和Windows的消息机制有一定的了解的话，以上概念是比较容易理解的。

2）定义和创建DirectDraw对象

分析完程序的总结构，再让我们从最开头看起。

#include <ddraw.h>

这是把DirectDraw的头文件包含到文件里来。这一步在以后所有的例程中都是必不可少的。

LPDIRECTDRAW lpDD; // DirectDraw 对象

LPDIRECTDRAWSURFACE lpDDSPrimary; // DirectDraw 主页面

接着定义了三个全局指针变量，它们都是指向对象的指针。第一个是DirectDraw对象，表示显示硬件，它包括了显示器和显卡还有显存，用它来代表整个显示系统。第二个是DirectDrawSurface对象，表示页面，你可以在大脑中把它想象成一张矩形的白纸，你可以在上面绘制图象。现在你暂且不用去深究它们的含义，看下去就是了，在后面的课程中，我还会更加详细的介绍。

LPDIRECTDRAW和LPDIRECTDRAWSURFACE是在ddraw.h头文件里预定义的指向DirecctDraw和DirectDrawSurface对象的长型指针，所以前面加了LP，代表Long Point。因为这些DirectDraw对象指针变量经常用到，所以给它起名为lpDD和lpDDSPrimary，DD是DirectDraw的缩写。

给一个变量命名时，在变量名前加上该变量的类型标志，在Windows编程中是一个默认约定，称为匈牙利表示法，该名称来源于微软的一个匈牙利籍资深程序员，因为他惯用此表示法，后来便成了规范。如lpDD表示一个长型指针变量，dwHeight代表一个DWORD型变量。这样的好处就在于你可以一眼就辨认出某变量的类型，而不用去追溯它的定义。

这里所说的“对象”(Object)，并不完全等同于C++中对象的概念，尽管它们使用的是同一个英文单词Object。这里的对象指的是COM，COM是Component Object Model 的缩写，代表“部件对象模型”，它在DirectX中贯穿始终，无处不是它的身影。

       在DirectX SDK中，大多数APIs（应用程序编程接口）由对象和基于COM的接口组成。COM是致力于可重复利用接口资源的面向对象系统的基础，并且是OLE编程的核心模型。它也是一个接口规范，通过它可以设计出许多接口。它是建立在操作系统层次的对象模型。

COM与C++类也有许多相同之处。对一个C++程序员来说，COM接口就象是一个抽象基础类。这就是说，它定义了一套关键符号（signatures）和语法（semantics），但不是执行语句，并且没有与接口相关联的状态数据。在C++抽象类中，所有的方法被定义成纯虚（pure virtual）函数，它们并没有实际的代码。在这一点上，COM和基础类是一致的。

COM对象与C++对象的另一个相似点是：一个函数的第一个引用是接口或类的名称，在C++中叫做this引用。因为COM对象和C++对象是完全二进制兼容的，编译器把COM接口当作C++抽象类来看，而且采取同样的语法。这样就可以减少代码的复杂程度。例如，this引用在C++中被当作可识元素，并且被暗中的操作，COM中也是如此。

COM 是DirectX的基础，虽然它和C++中的类不近相同，但你完全可以把它当成C++的类来看待，在实际编程中，它们的语法和接口也是完全一样的。所以，对于C++程序员，进行DirectX编程并不需要你去学习新的编程方法，继续沿用你所熟知的C++，DirectX也能为你所用。

只要是你要用到DirectDraw接口的特性，都必须创建一个DirectDraw对象，它是DirectDraw接口的核心。它是这样被创建的：

if ( DirectDrawCreate( NULL, &lpDD, NULL ) != DD\_OK ) return FALSE;

DirectDrawCreate()函数是在ddraw.h中定义的，关于这个函数的详细解释，请参看本教程的“DirectDraw参考手册”一章，它的原型如下：

HRESULT DirectDrawCreate(GUID FAR \* lpGUID, LPDIRECTDRAW FAR \* lplpDD, IUnknown FAR \* pUnkOuter);

第一个参数是lpGUID：指向DirectDraw接口的全局唯一标志符（GUID：Global unique identify）的指针。在这里，我们给它NULL，表示我们将使用当前的DirectDraw接口。

第二个参数是lplpDD：这个参数是用来接受初始化成功的DirectDraw对象的地址。在这里，我们给它&lpdd。

第三个参数是pUnkOuter：千万不要追问这个参数是干嘛使的，如果你不想惹麻烦，就给它NULL吧。Microsoft的说明书上是这么写的“考虑到与将来的COM集合特性保持兼容，当前，不管怎样，如果这个参数不是NULL ，DirectDrawCreate将返回一个错误”。

所有的DirectDraw函数的返回值都是HRESULT类型，它是一个4字节（32位）的值，用来代表某个错误或警告。DirectDraw头文件中已经预定义了所有可能的返回值常量，仅函数返回成功的值是用 “DD\_OK”表示，所有的错误值标志开头都为“DDERR”，如：

DDERR\_DIRECTDRAWALREADYCREATED

DDERR\_GENERIC

DDERR\_OUTOFMEMORY

Windows编程中，有一项让中国软件开发者大挠其头的就是函数名、常量名和数据结构名称中往往有长串的字符。这时，就要考验读者的英文断句水平了。如：DDERR\_OVERLAYCOLORKEYONLYONEACTIVE，应该断为：DDERR-OVERLAY-COLORKEY-ONLY-ONE-ACTIVE。这对于老外来说也许并不成什么问题，但对于我们，有时候要看懂一个语句，简直就象在做一道英文题。除非你对一个名称有十足的把握，否则就应该尽量使用CP规则（原是离散数学里的一条规则，现引申为Copy-Paste，即：复制-粘贴）。而且，给读者一点忠告，在起函数、变量或常量名称时，也应该沿用老外的这条原则，以尽量清楚的表达意义为宗旨，不要担心它是否太长（当然，对于那些重复利用率很高的函数、变量或常量，还应该尽量使用缩写），否则，总是以i、n为变量，程序的可读性会变得极差，而在编程的初学者当中，这是很常见的。

3）设置控制程度和显示模式

DirectDrawCreate函数调用成功后，lpDD已经指向了一个DirectDraw对象，它是整个DirectDraw接口的最高层领导，以后的步骤都是在它的控制之下。

if ( lpDD->SetCooperativeLevel( GetActiveWindow(),

DDSCL\_EXCLUSIVE | DDSCL\_FULLSCREEN ) != DD\_OK)

return FALSE;

       这个语句用来设置应用程序对操作系统的控制程度。它的原型如下：

HRESULT SetCooperativeLevel( HWND hWnd, DWORD dwFlags )

第一个参数是hWnd，我们调用Win32的API函数GetActiveWindow获得应用程序主窗口的句柄，这将使DirectDraw对象与主窗口的消息挂上勾。

第二个参数是dwFlags，控制级的标志符。我们给它DDSCL\_EXCLUSIVE | DDSCL\_FULLSCREEN，表示我们期望DirectDraw以独占和全屏方式工作。

这个函数有很多用法，而且它必须在创建DirectDraw对象之后，立即调用。你可以用它来设置应用程序是运行于全屏还是窗口模式，是独占还是共享模式。具体用法将在以后逐步介绍。可参阅“DirectDraw参考手册”。

if ( lpDD->SetDisplayMode( 640, 480, 8 ) != DD\_OK) return FALSE;

显而易见，这是设置显示器的显示模式，它把显示模式设为640\*480，8位色彩模式（即256色）。这是绝大多数显示器所能够支持的显示模式，所以我们不用担心它会出什么问题。在以后的例程中，我们将看到可以调用EnumDisplayModes()来列举出显示器所支持的所有显示模式。绝不要轻易尝试直接设置一个新的显示模式，而应从列举出的显示模式中选择，这也将在后续章节讲到。

要注意的是，只有当DirectDraw对象为独占访问的控制程度时才能改变显示器的显示模式，如果DirectDraw对象运行为窗口模式，调用该函数会返回一个错误。

4）创建主页面

然后的任务是要创建一个DirectDrawSurface对象。

DirectDrawSurface对象代表了一个页面。页面可以有很多种表现形式，它既可以是可见的（屏幕的一部分或全部），称之为主页面（Primary Surface）；也可以是作换页用的不可见页面，称之后台缓存（Back Buffer），在换页后，它成为可见；还有一种始终不可见的，称之为离屏页面（Off-screen Surface），用它来存储图像。其中，最重要的页面是主页面，每个DirectDraw应用程序都必须创建至少一个主页面，用它来代表屏幕上可见的区域，说白了，就是你的显示屏幕。

创建一个页面要分两步走，这里，我们以创建主页面为例，简要介绍一下这两个步骤，其它类型页面的创建也与之类似，在以后的例程中还会着重讲解。

在调用CreateSurface()函数创建一个页面之前，首先需要填充一个DDSURFACEDESC的结构，它是DirectDraw Surface Description的缩写，意思是DirectDraw的页面描述。该结构的详细资料请参看本教程“DirectDraw参考手册”一章。

//填充主页面信息

ddsd.dwSize = sizeof( ddsd ); //结构的大小

ddsd.dwFlags = DDSD\_CAPS; //指定DDSURFACEDESC结构的ddsCaps成员为可用

ddsd.ddsCaps.dwCaps = DDSCAPS\_PRIMARYSURFACE; //指定要创建的是主页面

       这就像似你到银行取款，必须事先填写一张取款单，你要在上面详细描述你的姓名，你的存折序号，以及你要取的钱数等等，然后把它递给银行工作人员。

页面描述填充完毕后，把它传递给CreateSurface()函数即可。

//创建主页面对象

if ( lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL ) != DD\_OK)

return FALSE;

CreateSurface()函数的第一个参数是被填充了页面信息的DDSURFACEDESC结构的地址，为&ddsd；第二个参数是接收主页面指针的地址，此处为&lpDDSPrimary；第三个参数现在必须为NULL，为该函数所保留。

如果函数调用成功，lpDDSPrimary将代表一个合法的主页面对象。由于在前面已经设置了该程序的工作模式为独占和全屏，所以，此时主页面所代表的实际上是你的整个显示屏幕。在主页面上所绘制的图形将立即反映到你的显示屏幕上。

5）输出文字

下面开始在主页面上输出文字。

//输出文字

if ( lpDDSPrimary->GetDC(&hdc) != DD\_OK) return FALSE; //获得设备环境句柄

SetBkColor( hdc, RGB( 0, 0, 255 ) ); //设置背景颜色

SetTextColor( hdc, RGB( 255, 255, 0 ) ); //设置文字颜色

TextOut( hdc, 220, 200, szMsg1, lstrlen(szMsg1)); //输出文字

TextOut( hdc, 280, 240, szMsg2, lstrlen(szMsg2));

lpDDSPrimary->ReleaseDC(hdc); //释放资源

如果你十分熟悉Windows的GDI（图形设备接口），你会发现上面这段程序居然和在窗口中输出文字一模一样。这是因为，DirectDraw页面和GDI的设备环境（DC）其实是兼容的。在调用主页面的GetDC()函数获得页面的设备环境句柄（HDC）之后，就可以使用Win32的API绘图函数来进行绘图操作了。同样，最后也必须调用主页面的ReleaseDC()函数来释放设备环境资源。

6）释放对象

在程序结束之前，DirectDraw还必须做一项扫尾工作，即：把已经创建的所有DirectDraw对象从内存中清除出去。这就是FreeDDraw()函数的作用。

void FreeDDraw( void )

{

if( lpDD != NULL ) //判断DirectDraw对象是否为空

       {

if( lpDDSPrimary != NULL ) //判断主页面对象是否为空

{

lpDDSPrimary->Release(); //释放

lpDDSPrimary = NULL;

}

lpDD->Release(); //释放

       lpDD = NULL;

}

}

每一个DirectDraw接口的对象都有Release()函数，以将其所引用的对象释放。这其实相当于C++中的delete方法。及时的将不用的对象释放掉是每一个优秀的程序员都应当养成的良好习惯。

6）主窗口的类型

以上关于DirectDraw接口的编程似乎与应用程序的主窗口没有一点联系，而且执行后，也看不到窗口的影子，是不是就可以不用创建程序的主窗口了呢？当然不是。

其实，程序执行后，漆黑的背景就是程序的主窗口。在注册窗口类时，已经给hbrBackground成员指定了黑色。

wc.hbrBackground = GetStockObject(BLACK\_BRUSH);

而且，在创建主窗口时，窗口的类型用的是WS\_POPUP，表明创建的是一个没有标题栏，没有边框的窗口；而不是常用的WS\_OVERLAPPEDWINDOW。

hwnd = CreateWindowEx(

……

WS\_POPUP,

……

值得特别注意的是：在设置DirectDraw对象控制级的时候，使用了主窗口的句柄（HWND）。这就是说，DirectDraw将用主窗口来接收各种消息，这样，就可以利用主窗口来实现DirectDraw对用户操作的反馈。如，用户想用光标键来控制游戏图像的运动，光标键按下的消息被发往主窗口，然后在主窗口的消息处理过程中操作DirectDraw，使之对光标键产生反应。所以，在大多数情况下，创建一个主窗口是必不可少的。

if ( lpDD->SetCooperativeLevel( GetActiveWindow() /\*主窗口句柄\*/ ,

DDSCL\_EXCLUSIVE | DDSCL\_FULLSCREEN ) != DD\_OK)

return FALSE;

四、小结

DirectX是一个功能强大而且使用复杂的工具，在这一章里，我们仅学习了一个最简单的利用DirectDraw接口的例子，对于庞大的DirectX来说，它只是冰山一角。要掌握DirectDraw编程是一个复杂的过程，需要从大量的实践中摸索出经验，任何读者想一蹴而就是不可能的。但是，好的开端意味着成功的一半，如果你能理解和掌握本章所学到的内容，继续下去，坚持不泄，你会发现你将一天比一天更加充实。

通过本章的学习，读者应能掌握：

在VC++5.0环境中使用Win32应用程序开发环境新建一个DirectDraw工程以及添加源文件。

用DirectDrawCreate()函数创建DirectDraw对象，并立即调用SetCooperativeLevel()设置其控制级，注意其第一个参数为应用程序主窗口句柄。

只有在独占的控制级下才能调用SetDisplayMode()以改变显示器的显示模式。

在调用CreateSurface ()创建DirectDrawSurface对象之前，必须先填充一个DDSURFACEDESC结构以描述页面的信息。

页面对象的种类，只有主页面代表了显示屏幕，为可见页面。

用GetDC()获得主页面的HDC，然后可以使用Windows API的GDI函数进行绘图或输出文字，最后必须调用ReleaseDC()以释放资源。

DirectDraw接口的对象必须被及时的用Release()函数释放。

第四节 DirectDraw基本图形概念

    在上一章里，我们尝试了一个非常简单的DirectDraw应用程序，可以看到，DirectDraw编程对一个C或C++程序员来说并不是陌生和难以接近的。在这一章里，我们将介绍Directdraw编程中所经常用到的图形方面的知识，在深入进行DirectDraw编程之前，领悟这些最基本的图形概念是非常必要的。

一、像素(Pixel)和分辨率（Resolution)

首先，让我们从电脑显示器屏幕上的基本单位像素说起。对用户来说，像素指的是显示器在某种分辨率状态下所能表达图象的最小单位。你可以把它简单的理解为一种类似于马赛克的东西。像素的大小与你当前的显示分辨率有关，现在所有的SVGA显示器都支持多种分辨率，比如通常所说的640x480、800x600等。许多人错误的将像素的大小与显示器的点间距（Dot Pinch）混淆了，其实它们指的不是同一个东西。当你用放大镜观察显示器屏幕时，你会发现像素实际是由许许多多红、绿、蓝整齐排列的小点组成，你可能会认为这就是像素，那么你就错了。这实际上是显示器光栅的一个扫描点，它是荧光屏后部的三束电子枪发射电子透过一层致密的网打到荧屏反面而发出的荧光，人们常说的0.28的彩显意即：屏幕上相临两个扫描点的平均间距为0.28毫米。一个像素是由若干个这样的扫描点组成的。专业地说，在某种分辨率状态下，显示器的水平（垂直）像素的个数，实际上等于一次水平（垂直）扫描期间，电子束的通短强弱状态能够发生变化的次数。一台彩显所能达到的最大分辨率受到这台彩显的尺寸和点距的限制。

显然，显示器型号越大，点间距越小，则它所能达到的分辨率就越高，那么，它所显示的图像就越清晰，表现得越细腻。

二、RGB色彩

现实生活中，我们所能分辨的所有颜色都能用三原色：红、绿、蓝合成出来，称为RGB色彩模式。用RGB方式合成颜色主要用于发光设备，如计算机显示器。通过分别调节红，绿，蓝三束电子枪的强度，就可产生一个广泛的色彩范围。由于色彩的产生来源于RGB三种色彩光强度的叠加，所以，这种方法被称为彩色叠加。

值得一提但与我们的DirectDraw编程基本上没有什么关系的另一种色彩表示模式是CMYK色彩模式，代表青，粉，黄三种可减基色，加上黑色作对比，常用于打印，感光材料，胶片等。每种基本色从照射到打印页的白光中吸取某些色彩，由于CMYK方式通过从白光中吸取特定的色彩，所以被称为彩色削减。此外，还有HSI（也称HSL)色彩模式，它代表Hue ， Saturation 和 Intensity 。Hue 是我们所认为的色调，如红橙黄绿青蓝紫等，Saturation描述该色调颜色的饱和度，Intensity 描述亮度。

RGB 颜色方式对于显示器来说是唯一适用的彩色混合方法，而CMYK则是适用于打印和印刷的彩色混合方法，HSI 则是一个很直观的定义彩色的方法。现在的许多图形软件中都同时具有这三种定义颜色方法的应用。但是，不论用那种方法给颜色取值，最终都必须转化为RGB方式才能为显示器所认识接收并显示出来。

在几乎所有的编程中，当然也包括DirectDraw，都是采用标准的24位RGB模式定义一个颜色，用三个从0到255的数值来分别代表R、G、B三个分量，每个分量都有256种亮度级别。当三种元素全被置为0，像素显示黑，通常表示为RGB(0，0，0)；当全部置为255，像素显示为白，表示为RGB(255，255，255)。

三、设备无关位图（DIB）

位图图象（也称点阵图象）准确的说是什么呢？它们是数据元素的集合，这些数据元素决定在了在图片的某个具体位置是什么颜色。这就好比我们用许许多多的马赛克来拼图案一样，每个马赛克虽然只有一种颜色，但你看到的只是整个美丽的图案，而不是某一个马赛克。当图象显示在显示器屏幕上时，正如前面提到的那样，这一个个的马赛克就是像素。

Windows中，当然还包括DirectX，都使用设备无关位图DIB（Device-Independent Bitmap）作为其最基本的图象文件格式。

之所以称之为设备无关位图（DIB），其实是为了与设备相关位图（DDB：Device dependent bitmap）相区别。由于只有Windows 3.0以前的版本才广泛使用设备相关位图DDB，Windows 95和Windows NT及以后的版本使用的都是设备无关位图DIB，所以，“设备无关位图”这个名称在当前已没有其现实的含义。在大多数情况下，我们将其简称为位图（Bitmap）。

从本质上说，DIB是一个包含了图象的各种信息的文件，包括图象的尺寸，使用的颜色个数及其颜色值，还有每个像素的颜色数据等。此外，一个DIB中还包含了一些极少使用的参数，象文件压缩类型、重要颜色，还有图象的物理维数等。DIB文件通常具有“.bmp”扩展名，偶尔也会将“.dib”作为其扩展名。

Windows对DIB位图有规定的格式，在我们学习了位深度和调色板的概念之后，对DIB格式再做详细的解释。

与位图格式相对的是矢量格式，由于它是用形状和相互关系来描述图象，WMF文件就是一种典型的矢量图，它的特点是具有极大的灵活性，可以任意缩放不失真，且存储空间相对较小，但矢量图由于其描述图象的方式与位图完全不同，不适用于图象处理的领域。在以后的DirectDraw编程中，我们所指的图象特指位图图象。

四、位深度（Bit depth）

计算机中，一个字节（Byte）是由8个位（Bit）组成的。位深度指的是用来描述某状态值所使用的计算机位的个数。在DirectDraw中，通常用位深度来代表位图中的颜色值所使用的位个数，从另一个意义上讲，位深度表示了位图中颜色的丰富程度。

1字节等于8位二进制数，所以一字节所能表达的十进制整数的范围是从0到255，即一字节能且最多只能反映出256（2的8次方）种不同的状态。在位图中，如果用其中每一种状态代表图像中某一个点的颜色，那么最多可以得到256种不同的颜色，这就是我们常见的8位（256色）的位图。同理，如果用1位二进制数来表示某一点的颜色，那么只能得两种（2的1次方）颜色，这就是一幅两位的黑白位图；如果用四位二进制数则产生16种颜色（2的4次方），这是一幅4位（16色）的位图，用24位产生16M种颜色（2的24次方）等，这就是一幅24位真彩位图。由于每幅图形文件都用确定的位数来代表颜色（1、2、4、8、16、24或32位），所以我们所见的图片颜色数都为2的n次方（n=位深度）。

显然，一幅图象的位深度越高，那么它所能表现的颜色也越多，色彩也就越丰富。通常，在一般情况下，8位（256色）的图片已可满足需要，但在一些要求高质量图象的场合，16或24位的图象是必须的。当然，颜色位深度的增加，也势必带来所需存储空间的膨胀，没有经过压缩的相同大小的24位图象所需的存储空间是8位图象的3倍，这也会带来文件读取和操作速度的降低，所以在需要高速度显示图象的场合，使用低位深度的图象是必要的。

五、抖动处理（Dithering）

在低位深度的图象中，由于颜色总数的限制，有些颜色无法显示出来，为了模拟出那些颜色以提高显示效果，广泛采用了一种称作抖动处理（dithering）的方法，也称半色调处理（Halftoning）。它是指用交替的点图案去模拟在图像中不能使用的颜色的过程。单色图像是最简单的格式，一般由黑色和白色组成，在一些单色图象如黑白照片和有深浅的图案中，会使用各种灰度，这种图像常被称为灰度图像(Grayscale Image)。由于人眼会把一个很细致的黑白相间的图案解释成灰色，所以灰度图象也可使用单色文件格式，数据仍然可以是黑和白。使用黑色或某一种单色的点获得连续的该色灰度的过程就是抖动处理。抖动处理被更多的用在那些低位数彩色图像文件中，与不采用这种处理相比，它具有更好的显示效果。

六、调色板（Palette)

显示器及显示卡与显示器的接口都采用模拟方式来处理色彩，因此，它们都具有几乎无限的色彩显示和传输能力，但主机和显示卡只能用数字方式来表示和处理色彩，在用数字方式表示色彩时，如果要获得更丰富，更细腻的色彩就需要增加色彩的位深度，这就需要更大容量的显示存储器，相应的也就需要更高的处理速度，同时分辨率的提高也对显示存储器的容量提出了很高的要求。

为了尽量降低对显存的需求，在Windows中可以使用一种间接的色彩表示方法，这就是调色板（Palette）表示法。它的含义是：用一个颜色索引（Color Index）来代表各个像素点的颜色，而不是直接用红，绿，蓝三基色的亮度值来确定每个像素点的颜色。色彩表（Color table）是一个包含了若干颜色索引和该索引所对应的真实颜色值的表，这个色彩表就是调色板。

这种方法就好比给学校里的每一个学生规定一个专用的学号，在很多场合，并不需要知道某个同学的实际姓名，只要知道他的学号即可，显而易见，使用学号无疑会给学校的学员管理工作带来了极大的便利。调色板就好比一张记载了学生学号及其姓名的名册，颜色索引值就是学生的学号，RGB颜色值就是学生的姓名。

使用调色板的好处就在于：索引值占用较少的数据位(1、2、4或8位)，而真实颜色值占用较长的数据位(24位，即3字节，分别代表红，绿，蓝三基色的颜色亮度值，从0到255)。由于使用了调色板，既提高了图象显示效率，又减少了对显存的需求。

调色板的颜色索引主要采用4或8两种位深度：4位位深度可有16种不同取值，对应于显示器的16色显示模式，这时一屏最多只能显示16种不同的颜色；8位位深度可有256种不同的取值，对应于显示器的256色显示模式，一屏最多只能显示256种颜色。于是，问题就开始出现了，当你在16色模式下显示一幅256色或更高位深度的的图象时，你将会看到本应色彩鲜艳的图片变得面目全飞，罪魁祸首就是调色板，由于它使得同时显示在屏幕上的不同颜色最多只能有16种（对应于16色模式下），于是Windows首先从图象中挑选了16种使用频率最高的颜色指定给调色板，对于所有其它的颜色，从调色板中挑选一个最接近的颜色显示出来。在256色模式下，要同时显示两幅不同调色板的256色位图时，也会发生这种图象失真的情况。

七、GDI与DirectDraw

在大多数的Windows编程中，开发者们使用的是Win32的函数以获得访问绘图页面的能力，例如，使用GetDC函数，可以获得设备环境（DC?/FONT>Device context）。在获得设备环境之后，你就可以开始进行对屏幕的绘图了。Win32的所有图形函数都是由Windows系统的一个独立完整的模块所提供，这就是图形设备接口（GDI?/FONT>Graphics device interface）。GDI为计算机用户和计算机硬件之间提供了一个抽象层，在此层的基础上，用户可以通过简单的调用Win32的图形函数进行图形显示。

GDI的一大缺憾就是，它不是为具有高表现力的多媒体软件和游戏而设计的，设计者们开发它的主要用途是运行商业应用软件诸如：Word字处理软件、Excel电子表格、Explorer浏览器等。GDI只提供了访问系统主存的能力，而不提供直接访问显存的能力，并不能从具有某些加速特性的显卡中获得其优良特性。简而言之，GDI对绝大多数的商业软件来说是相当完美的，但对于多媒体软件和游戏来说，它却是低速和低效的。

另一方面，DirectDraw可以提供给开发者代表了真实显示内存的绘图页面。这意味着，只要你使用了DirectDraw，你就可以直接操纵显卡上的内存，图形显示变得出奇的快速。而且这些页面代表了显存中连续的内存块，使得在页面中寻址和读写变得非常方便。

八、位块传送（Blit）

“Blit”是“Bit block transfer”的缩写，意为“位块传送”。顾名思义，Blit的作用是：将某一内存块的数据传送到另一内存块，前一内存块被称为“源”，后一内存块被称为“目标”。这里用的是“传送”一词，而不是“复制”，因为在Blit过程中，数据并不是被原封不动的转移，而是经过了一定的转换。

在许多书籍中或程序中经常可以见到的“Bitblt”、“Blt"或“Bltting”，其实也是同一个意思，前者读作['bitb'lit]；后者读作[b'lit]。

在绝大多数情况下，Blit操作是针对位图图象的，因此，源数据代表的是“源位图”，目标数据代表的就是“目标位图”。图象程序开发者使用Blit的函数在内存中将某页面上的一幅位图经过一定的变换转移到另一个页面上。这种变换有很多种，每一种都有一个代码与之对应，我们称该代码为光栅操作代码（ROP?/FONT>Raster operation code）。

Blit操作被广泛的用于图形程序中，如在显示位图、移动、复制位图、位图合成、位图特效以及精灵的实现中都有blit的身影。GDI和DirectDraw中，都提供了blit的函数，如GDI中的BitBlt、StretchBlt、PatBlt等，DirectDraw中的IDirectDrawSurface3::Blt和IDirectDrawSurface3::BltFast等。

下面我们来分析一个典型的Blit的函数，GDI的BitBlt是Win32 API中一个重要的函数，关于它的详细资料可以参阅VC中的帮助，位于：Platform，SDK，and DDK Document\Platform SDK\Reference\Functions\Win32 Functions，它的原型如下：

BOOL BitBlt(

HDC hdcDest， //目标设备环境的句柄

int nXDest， //目标设备环境的矩形区域的左上角的x坐标

int nYDest， //目标设备环境的矩形区域的左上角的y坐标

int nWidth， //目标设备环境的矩形区域的宽度值

int nHeight， //目标设备环境的矩形区域的高度值

HDC hdcSrc， //源设备环境的句柄

int nXSrc， //源设备环境的矩形区域的左上角的x坐标

int nYSrc， //源设备环境的矩形区域的左上角的y坐标

DWORD dwRop //光栅操作符

);

dwRop参数是光栅操作代码（Rop），它是指源位图与目标位图以及图案刷的颜色值进行布尔运算的方式，以下列出了常用的光栅操作符。

光栅操作代码       含义

BLACKNESS       用黑色填充目标矩形区域。

DSTINVERT       将目标矩形图象进行反相。

MERGECOPY      将源矩形图象与指定的图案刷（Pattern）进行布尔“与”运算。

MERGEPAINT     将源矩形图形经过反相后，与目标矩形图象进行布尔“或”运算。

NOTSRCCOPY    将源矩形图象经过反相后，复制到目标矩形上。

NOTSRCERASE   先将源矩形图象与目标矩形图象进行布尔“或”运算，然后再将

得图象进行反相。

PATCOPY        将指定的图案刷复制到目标矩形上。

PATINVERT       将指定的图案刷与目标矩形图象进行布尔“异或”运算。

PATPAINT        先将源矩形图象进行反相，与指定的图案刷进行布尔“或”运算，

 再与目标矩形图象进行布尔“或”运算。

SRCAND         将源矩形图象与目标矩形图象进行布尔“与”运算。

SRCCOPY           将源矩形图象直接复制到目标矩形上。

SRCERASE           将目标矩形图象进行反相，再与源矩形图象进行布尔“与”运算。

SRCINVERT                将源矩形图象与目标矩形图象进行布尔“异或”运算。

SRCPAINT            将源矩形图象与目标矩形图象进行布尔“或”运算。

WHITENESS               用白色填充目标矩形区域。

      表中，提到了三种的基本布尔运算，分别是：反相（NOT）、与（AND）、或（OR）。还提到了异或（XOR），学过数理逻辑的人都知道，异或运算其实也是可以由前三种组合出来的。可以看出，所有的光栅操作代码都是由三种基本布尔运算组合而成。实际上，一共有256种光栅操作代码，但最常用的就是以上这15种。若想使用这15种以外的任何一种时，可以查阅VC的帮助，但在实际中，它们是极少被用到的。

图案刷（英文是Pattern， 也可翻译成模式刷，直接的意思是布料上图案的花样）是Windows资源的一种，属于Brush，它其实是一个固定大小的位图（通常为8x8像素），用来平铺填充设备环境（DC）的某一区域。它的用法与普通的刷子是没有区别的，当把一个图案刷用SelectObject函数选定给某设备环境，再在该设备环境中进行填充操作，那么所填充的不是一个单纯的颜色（选用普通的刷子时的情况），而是连续平铺的图案。

GDI的blit函数还有PatBlt和StretchBlt。它们与BitBlt大同小异，BitBlt具有Patblt的所有功能，StretchBlt除了具有BitBlt的一切功能外，它还可以将位图放大或缩小，实现缩放功能。

DirectDraw最常用的blit函数是IDirectDrawSurface3::Blt，它的原型如下：

HRESULT IDirectDrawSurface3::Blt(

LPRECT lpDestRect， //目标矩形区

LPDIRECTDRAWSURFACE3 lpDDSrcSurface, //源页面

LPRECT lpSrcRect, //源矩形区

DWORD dwFlags, //标志符

LPDDBLTFX lpDDBltFx //光栅操作代码及特效

);

DirectDraw中的Blt函数比GDI中的BitBlt函数虽然在形式上极为相似，但在内容上却有了质的飞跃。首先，从运行速度上来比较，IDirectDrawSurface3::Blt可以利用到一切可以利用的硬件加速特性，而且该操作在缺省情况下是异步执行的，这意味着程序不需要等待blit结束才返回，而是在给系统发出了blit的指令后，立即返回，然后系统在后台进行blit操作，这使程序的运行变得极为高效、快速；其次，它还支持带源和目标关键色以及z缓存和alpha 通道的blit操作，这使得用该blit函数来完成各种效果变得极更为容易。

该函数的目标页面就是调用者本身，而且矩形区域用RECT结构表示。第四个参数dwFlays表示该blit操作的类型，最后一个参数lpDDBltFx是一个包含了光栅操作代码和其它特效的结构，其中的光栅操作代码成员与Win32的是兼容的。

DirectDraw中的blit函数还有IDirectDrawSurface3::BltFast和IDirectDrawSurface3:: BltBatch。前者完成一次显存中的blit操作，在没有硬件加速的情况下，其速度可以较Blt快一些，后者完成一组blit操作。

要得到更多的关于DirectDraw中的blit函数的资料，请参阅本教程的DirectDraw参考手册。

九、翻页（Page flipping）

在多媒体动画、动感游戏等软件中，翻页是一个相当关键的概念。与绘画师绘制动画片相比，计算机中的翻页技术与其具有相似之处。例如：绘画师在一叠相同的纸上绘画，画完一张再画下一张，在每一张上，绘画师使画面有略微的改动，于是，当你在这一叠纸上快速的翻页时，静止的图象便开始运动起来。

DirectDraw中的翻页与上面这个过程极为相似。首先，你得设置好一个翻页链结构，它由一组DirectDraw页面组成，每一个页面都可以被轮流翻页至显示屏幕。当前正好位于显示屏幕的页面，我们称之为主页面（primary surface），其后等待翻页至屏幕的页面，我们称之为后台缓存（Back buffers）。应用程序在后台缓存上进行绘图操作，然后“翻一页”，将此页面翻页成为主页面，原来的主页面成为后台缓存，翻页后，你所进行的修改可以立即显示在屏幕上。与此同时，你可以在下一个即将翻页成为主页面的后台缓存上进行绘图。将这个翻页过程一直持续下去，直到动画结束。

有了DirectDraw，整个翻页的任务并不是一件十分困难的工作。你既可以创建一个简单的双缓存翻页链结构（一个主页面和一个后台缓存），也可以创建一个使用起来更为灵活的多后台缓存翻页链结构。

由于DirectDraw更多的使用的是硬件的特性，使得翻页变得极为快速，这个过程在屏幕上不会产生丝毫的闪烁，其速度可以达到与显示器的刷新率一样的数量级。在后面的DirectDraw教程中，我们将具体介绍如何在DirectDraw程序中实现翻页。

十、矩形（Rectangle）

在Windows编程中，屏幕上的对象都是以一个封闭矩形的形状出现的。一个封闭的矩形可以用两个点来确定，左上角和右下角。绝大多数应用程序使用RECT结构来定义一个封闭矩形，用于blit操作或碰撞检测（hit detection）。RECT结构定义如下：

typedef struct tagRECT {

LONG left; //矩形左上角的X坐标

LONG top; //矩形左上角的Y坐标

LONG right; //矩形右下角的X坐标

LONG bottom; //矩形右下角的Y坐标

} RECT, \*PRECT, NEAR \*NPRECT, FAR \*LPRECT;

应该注意的是：这个矩形区域是排除边线的，并不包括它的右边线和下边线。所以，这个矩形的宽度应该等于right-left，而不是right-left+1；同理，矩形的高度等于bottom-top。

十一、精灵动画（Sprite animation）

精灵动画被广泛的用于多媒体及游戏软件中。从最基本的意义上说，精灵是一个可以在背景屏幕上四处移动的图象，通常，这个图象的形状是不规则的。它的实现方法可以简单的这样来描述：将精灵画在可见的背景页面上，然后将所画的上一个精灵从页面上抹去，再将精灵画在页面的另一个地方，依次类推。于是，对观察者来说，精灵在屏幕上就动起来了。

然而，在绝大多数情况下，我们所使用的精灵图象并不是规则的矩形，它们可能是多边形，也可能是一个支离破碎、完全不规则的图形。这就给我们实现精灵动画带来一个巨大的挑战，因为所有的blit函数所使用的都是规则的矩形区域，只有对于矩形区域，才能使blit操作更高效、流畅和便于调用。

于是，在GDI编程中，使用了一种称为屏蔽的方法，这也跟电影的特级合成用的方法差不多，先让一个人在一块蓝布前表演一些看上去很惊险的动作，然后再将这些影片与事先拍好的背景合成一体，有蓝布的地方成了背景的画面，人就好象置身其中了。

要在计算机中实现这样的动画，得有两幅图：一幅是精灵图(Sprite)，一幅是屏蔽图(Mask：也称掩码图)。精灵图中使要显示的部分颜色保持不变，其余全为黑色，屏蔽图中使精灵的部分都为黑色，其余全为白色。

有了这两幅图，经过两次blit运算，就可以把想要的精灵贴到背景上去了。第一次blit是让屏蔽图与背景进行布尔“与”（光栅操作代码为SRCAND）运算，第二次让精灵图与背景进行“异或”（光栅操作代码为SRCINVERT）运算。

伪代码如下：

BitBlt( hDCdest, 0, 0, width, height, hDCMask, 0, 0, SRCAND );

BitBlt( hDCdest, 0, 0, width, height, hDCSprite, 0, 0, SRCINVERT);

可以看出，在GDI中实现精灵动画是个比较复杂的过程，主要原因就是因为GDI的Blt函数不支持透明方式。在下面我们将看到，在DirectDraw中，由于其blit函数支持关键色，可以实现透明blit，所以，与GDI相比，在DirectDraw中实现精灵动画要变得简单得多。

十二、关键色（Color Key）

DirectDraw的blit函数，支持带关键色（Color key）的透明blit操作。这就是说，如果在源页面上指定了一个颜色为关键色，那么在blit操作中，将视具有这种颜色的区域为透明，不会被传送到目标页面上。

所以，不管你的精灵看起来是否象一个矩形，你必须首先使它们可以放在一个合适大小的矩形区域内。然后在这个包含了精灵图象的矩形内，将不属于精灵图象的部分全部使用同一种颜色，或使其在一个颜色范围之内。这个颜色或颜色范围必须是精灵图象中所没有用到的，它就是关键色。

使用IDirectDrawSurface3::SetColorKey函数，将这个关键色设置给该页面。之后，调用IDirectDrawSurface3::Blt或IDirectDrawSurface3::BltFast函数，将该精灵blit到另一个页面上去，矩形区域中只有属于精灵的像素被映射了，其余带关键色的像素全被视为透明，对目标页面不会产生任何影响。这种在blit操作中，用于源页面表示透明区域的关键色称为源关键色（Source color key）。

除此之外，你还可以使用一种目标关键色，它所影响的只是目标页面。目标关键色定义了在blit操作中，目标页面上可以被覆盖的像素。举例来说，如果你要设计一个有一棵大树的背景，一个兔子在树后活动，当它经过树的时候，会被树遮住它的一部分或全部，以创造一种前后效果。那么，同样，你应该给背景页面上所有除了树的像素使用关键色，表示只有这些像素才能允许在blit中被使用。使用IDirectDrawSurface3::SetColorKey函数，将这个关键色设置给背景页面。然后调用IDirectDrawSurface3::Blt或IDirectDrawSurface3::BltFast函数并且指定了使用目标关键色，那么，当你把包含了兔子的页面blit到目标页面上大树所在的位置，大树总是完整的，兔子的某部分会被大树遮挡住，就象出现在树后一样，前后效果出现了。

十三、补丁（Patching）

在精灵动画的过程中，当你把精灵画到一个新的位置之前，你必须将上一个位置的精灵从背景上抹掉。当然，最简单的做法是莫过于重新刷新整个背景，使其还原，然后再将精灵画上去。但是，这样做会使你失去很多的时间，而时间对于动画来说是非常宝贵的，而且最主要的一点就是，你的屏幕会因为大面积的刷新而闪烁。所以，你应该跟踪精灵所在的上一个矩形位置，然后在画下一个精灵之前，只重画这一个区域。这种方法被形象的称为打“补丁”（patching）”。

要给精灵的上一个位置打上补丁，你必须在上一步绘制该精灵之前，把这个矩形区域的背景内容保存下来，在你要画下一个精灵之前，用这个补丁去还原这块区域。这个过程会进行得非常和谐，因为与刷新整个背景相比，它不会占用太多的时间。

打补丁的方法可以按以下几个步骤进行：

将保存的精灵的上一个位置的背景图象复制到背景上。

把将要绘制精灵的位置的背景图象保存下来。

将精灵blit到背景图象上。

重复步骤1

十四、范围检查（Bounds Checking）与碰撞检测（Hit Detection）

范围检查和碰撞检测是编程中与精灵动画相关联的两个重要任务。

范围检查用来限制精灵的可能的活动范围，例如，在程序中，你可能想让某一个精灵限制在屏幕的某个矩形区域内活动，要完成这一步，你应该在精灵每移动一步之前检查精灵所在的位置，使这个位置的坐标保持在一个矩形范围之内（通常是一个RECT结构），并且阻止精灵移出这个矩形区域。DirectDraw并没有提供范围检查的服务函数，但是你可以用很简单的程序实现这样一个功能。

碰撞检测，或称为冲突检测，用来判断某一时刻是否有多个精灵处于同一位置。大多数的碰撞检测是检查某个精灵的包络矩形是否与另一个精灵的包络矩形有重合部分。因为有太多种不同的方法来实现不同种类的碰撞检测，所以DirectDraw也没有为用户提供碰撞检测的服务函数。用户可以根据自己的需要编制碰撞检测的程序。

第二章DirectDraw核心（高级篇）

第一节 DirectDraw架构

这一章介绍了DirectDraw与操作系统和系统硬件之间的关系。

一、DirectDraw架构概览

多媒体应用程序及游戏需要高表现力的图形引擎。Microsoft公司通过DirectDraw，为广大开发者提供了一个比GDI层次更高、功能更强、操作更有效、速度更快的应用程序图象引擎，与此同时，也努力使其保持了设备无关的优良特性。DirectDraw主要提供了完成以下任务的工具。

管理多页面

直接访问视频RAM

换页（Page flipping）

后台缓存（Back buffering）

管理调色板（Palette）

裁剪（Clipping）

视频端口（Video port）

       除此之外，DirectDraw允许开发者在应用程序运行期测定显示硬件的特性，然后，充分利用主机硬件设备的加速特性为用户提供可能的最优的显示速度和效果。

与DirectX其它组件一样，只要可能，DirectDraw就会最高程度的利用硬件执行某特定功能，并且让那些该硬件还不支持的特性也能用软件仿真的方式加以实现。设备无关性通常是通过硬件抽象层（HAL：Hardware abstraction layer）实现的，要得到更多的关于HAL的资料，请参阅“硬件抽象层（HAL）”。

DirectDraw是通过基于COM的接口提供服务。在DirectX 5.0版本中，这些接口分别是：IDirectDraw2、IDirectDrawSurface3、IDirectDrawPalette、IDirectDrawClipper和IDirectDrawVideoPort。DirectX的这些组件是向下兼容的，它们仍然支持旧版本中的所有功能。要得到更多的关于COM的概念，以有助于理解和创建DirectX应用程序，请参阅“DirectX与部件对象模型（COM）”。

DirectDraw对象代表显示适配器，并且通过IDirectDraw或IDirectDraw2接口将其函数性暴露于开发者。在大多数情况下，开发者使用DirectDrawCreate函数创建一个DirectDraw对象，但也可以通过使用CoCreateInstance COM函数创建之。要得到更的资料，请参阅“用CoCreateInstance创建DirectDraw对象”。

在DirectDraw对象创建好之后，你可以通过使用IDirectDraw2::CreateSurface方法为该DirectDraw对象创建页面。页面代表了位于显示硬件上的内存，但是它既可以存在于视频RAM，也可以存在于系统RAM中。DirectDraw还扩展了对调色板、裁剪（主要用于基于窗口的应用程序）和视频端口（Video port）的支持。

二、DirectDraw的对象类型

你可以将DirectDraw视为由若干个协同工作的对象所组成。DirectDraw所使用的对象有以下五个：

对象                          含义

DirectDraw            DirectDraw对象是DirectDraw应用程序的核心。它是你在建立DirectDraw应用程序时所要创建的第一个对象，再用它来创建所有其它相关的对象。通过调用DirectDrawCreate函数可以创建一个DirectDraw对象。DirectDraw对象通过IDirectDraw和IDirectDraw2接口为开发者提供其函数性。要得到更多的资料，请参阅“DirectDraw对象”。

DirectDrawSurface       DirectDrawSurface对象，通常简称为“页面（Surface）”，代表了内存中的一块区域，它存储了可以显示在显示器上的图象数据。通过调用DirectDraw对象的IDirectDraw2::CreateSurface函数可以创建一个与该DirectDraw对象相关联的页面。DirectDrawSurface对象通过IDirectDrawSurface、IdirectDrawSurface2和IDirectDrawSurface3接口为开发者提供其函数性。要得到更多的资料，请参阅“页面”。

DirectDrawPalette   DirectDrawPalette对象，通常简称为“调色板（Palette）”，代表了一个可以为页面所使用的16或256色的调色板。它包含了一组RGB值的索引，用来描述页面上的像素所使用的颜色值。对于像素位深度大于8的页面，不需要使用调色板。通过调用IDirectDraw2::CreatePalette函数，可以创建一个DirectDrawPalette对象。DirectDrawPalette对象通过IDirectDrawPalette接口为开发者提供其函数性。要得到更多的资料，请参阅“调色板”。

DirectDrawClipper   DirectDrawClipper对象，通常简称为“裁剪器（Clipper）”，帮助开发者使Blit（位块传送）操作限定在页面的某一区域内，或不超出页面的边界范围。通过调用IDirectDraw2::CreateClipper函数可以创建一个DirectDrawClipper对象。DirectDrawClipper对象通过IDirectDrawClipper接口为开发者提供其函数性。要得到更多的资料，请参阅“裁剪器”。

DirectDrawVideoPort DirectDrawVideoPort对象代表了当前某些系统上的视频端口（Vedio

port）硬件。这个硬件允许直接的访问帧缓存，而不需要通过CPU或使用PCI总线。通过对DirectDRaw对象调用QueryInterface函数（指定IID\_IDDVideoPortContainer标志符），可以创建一个DirectDrawVideoPort对象。DirectDrawVideoPort对象通过IDDVideoPortContainer和IDirectDrawVideoPort接口为开发者提供其函数性。要得到更多的资料，请参阅“视频端口”。

三、硬件抽象层（HAL：Hardware Abstraction Layer）

DirectDraw通过硬件抽象层（以后简称为：HAL）来提供设备无关的特性。HAL是由设备生产商提供的指定设备的接口，DirectDraw用来直接操作显示硬件。应用程序从来不会直接与HAL打交道，相反，而是与HAL所提供的下属函数打交道。

DirectDraw HAL可以以16位、32位或在Win95中两者[兼而有之](https://www.baidu.com/s?wd=%E5%85%BC%E8%80%8C%E6%9C%89%E4%B9%8B&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的形式执行。HAL在WinNT中通常以32位方式执行。HAL可以是显示设备驱动程序的一部分，或独立的DLL，通过驱动程序编写者定义的一个私有接口联系显示驱动。

DirectDraw HAL是由芯片制造商、板卡生产商或原始设备制造商（OEM）实现的。HAL只执行硬件有关代码而不进行仿真。如果硬件不能实现某个功能，HAL不会将其反映在自己的硬件特性中。

四、硬件仿真层（HEL：Hardware Emulation Layer）

当硬件抽象层（HAL）不支持某种特性时，DirectDraw会试图进行软件仿真。仿真的函数是由硬件仿真层（HEL）提供的。HEL与HAL一样，代表了DirectDraw的特性，并且应用程序从来不直接与HEL一起工作。结果是，DirectDraw对硬件的主要特性都提供了透明的支持，而不管这个特性是通过HAL硬件支持的还是通过HEL软件仿真的。

很显然，软件仿真不能与硬件所提供的特性等效。可以调用IDirectDraw2::GetCaps函数以查询硬件支持什么特性。在应用程序初始化的时候检查这些特性，你可以调整应用程序的参数以提供优化的性能。

在有些情况下，硬件特性与软件仿真的组合操作反而会比单纯使用软件仿真效率更低。例如，如果显示设备驱动程序支持DirectDraw，但不支持带缩放的Blit操作，在从视频RAM页面进行带缩放的Blit操作时，将导致明显的速度降低。这是因为有些视频RAM的速度要比系统RAM慢，迫使在访问视频RAM页面的时候，CPU进入等待状态。如果你的应用程序使用硬件不支持的特性，某些时候，在系统RAM中创建页面更为合适，这样才能避免CPU访问视频RAM时的效率损失。

五、系统集成

下图展示了DirectDraw，图形设备接口（GDI），硬件抽象层（HAL）和硬件仿真层（HEL）四者之间的关系。

如上图所示，DirectDraw对象与GDI位于同一层次，都通过一个设备相关的抽象层来直接访问硬件设备。与GDI不同的是，DirectDraw会尽可能的利用硬件的加速特性。如果硬件不支持某特性，DirectDraw会使用HEL试图将该特性进行软件仿真。DirectDraw可以以设备环境（DC）的形式提供页面内存，使得开发者可以使用GDI的函数操作页面对象。

第二节控制级（Cooperative Levels）

控制级描述了DirectDraw是怎样与显示设备相互作用的，它如何对系统事件产生反应。使用IDirectDraw2::SetCooperativeLevel函数可以设置DirectDraw的控制级。在很大程度上，开发者使用DirectDraw控制级来决定其应用程序是运行于全屏模式（具有独占的访问视频RAM的特性），还是运行于窗口模式。不管怎样，DirectDraw的控制级具有以下作用。

允许DirectDraw使用Mode X分辨率。要得到更多的信息，请参阅“Mode X 和Mode 13显示模式”。

阻止DirectDraw释放对显示设备的独占控制，或按Ctrl + Alt + Del以重新启动计算机。（仅用于独占模式）

允许DirectDraw对应用程序进行最小化或最大化控制，作为对系统事件的反应。

普通的控制级表明你的DirectDraw应用程序将以窗口的形式运行。在这种控制级下，你将不能改变显示器分辨率，主页面的调色板，或进行换页操作。除此之外，你也不能够调用那些会使视频RAM或视频RAM产生激烈反应的函数，例如：IDirectDraw2::Compact等。

当应用程序为全屏并且独占的控制级时，你就可以充分的利用硬件资源了。在这种控制级下，你可以设置自定义和动态的调色板，改变显示器分辨率，紧凑内存，和实现换页操作等。独占模式（也可称为全屏模式）不会妨碍其它的应用程序分配页面内存，也不会阻止它们使用DirectDraw或GDI的函数性。然而，它的确会阻止除了它自己（为活跃状态时）以外的应用程序改变显示模式或调色板。

因为DirectDraw应用程序可以具有多窗口，所以，在调用IDirectDraw2::SetCooperativeLevel设置控制级时，如果应用程序请求了DDSCL\_NORMAL模式（表明应用程序以普通窗口的形式运行），则不需要提供一个指定窗口的句柄。给窗口句柄参数为NULL，所有的窗口的消息进程都可以同时被使用。

IDirectDraw2::SetCooperativeLevel函数在内部捆绑了消息进程和一个窗口句柄。如果IDirectDraw2::SetCooperativeLevel函数在一个进程中被调用了一次，那么，这个进程就会和一个窗口句柄捆绑起来。如果该函数在同一进程中再次被调用，并且指定了另一个合法的窗口句柄，那么会返回一个DDERR\_HWNDALREADYSET错误。当DirectSound在设置控制级时指定了与DirectDraw不同的窗口时，有些应用程序也可能会返回这个错误值棗它们必须被设为同一个、顶层的窗口句柄。

第三节显示模式

一、关于显示模式

显示模式指的是显示器的当前设置，描述了显示器的分辨率和位深度，这个信息通常是由显示硬件从主页面传递给显示器的。显示模式是由三个特征定义的：宽、高、位深度。例如，大多数的显示器可以显示宽为640像素、高为480像素的图象，每一个像素的位深度是8。通常我们把这个显示模式称作640x480x8。随着显示模式的尺寸和位深度的增加，它所需要的视频RAM也随之增加。

有两种显示模式：调色板式和非调色板式。对于调色板式显示模式来说，每一个像素的颜色值是以一个相关调色板的索引值来代表。显示模式的位深度决定了调色板中可容纳的颜色数量。举例来说，在8位的调色板显示模式中，每一个像素的值从0到255，该调色板可容纳256个颜色入口。

非调色板式显示模式，就象它的名称所表示的那样，不需要使用调色板。在这种显示模式下，像素的位深度为16、24或32，每个像素分别占用2字节、3字节或4字节，用来描述像素的真实颜色。

主页面、以及在换页链中的所有页面必须符合显示模式的尺寸，位深度和像素格式（请参阅“像素格式”）。

二、测定支持的显示模式

因为显示硬件（包括显示卡和显示器）的不同，不是所有的显示设备都支持所有的显示模式。要测定某系统所支持的显示模式，应该调用IDirectDraw2::EnumDisplayModes函数。设置正确的参数和标志符，IDirectDraw2::EnumDisplayModes可以列举出该系统所支持的所有的显示模式，或判断是否支持用户所指定的显示模式。该函数的第一个参数，dwFlags，控制该函数的额外选项，在大多数情况下，你应该设置dwFlags为0以表明忽略额外的选项。第二个参数，lpDDSurfaceDesc，是一个DDSURFACEDESC结构的地址，包含了要被测定的显示模式信息，通常，该参数被设为NULL，以列举出该系统所支持的所有显示模式。第三个参数，lpContext，是你想让DirectDraw传递给其回调函数的一个指针，如果在回调函数中不需要任何数据，给该参数值为NULL。最后一个参数，lpEnumModesCallback，一个应用程序定义的回调函数的地址，在DirectDraw每列举出一个显示模式的时候，该回调函数将被调用。

在调用IDirectDraw2::EnumDisplayModes时所提供的回调函数必须符合EnumModesCallback函数的原型。每当找到一个硬件所支持的显示模式的时候，DirectDraw调用该回调函数，并且传递了两个参数。第一个参数是一个DDSURFACEDESC结构的地址，包含了一个支持的显示模式的描述。第二个参数是一个应用程序定义的数据的地址，是在调用IDirectDraw2::EnumDisplayModes时所指定的第三个参数。

检查DDSURFACEDESC结构中的值以获得它所描述的显示模式，关键的成员是dwWidth、dwHeight、和ddpfPixelFormat。dwWidth和dwHeight成员代表了显示模式的长和宽，ddpfPixelFormat成员是一个DDPIXELFORMAT结构的地址，它包含了显示模式的位深度信息。

DDPIXELFORMAT结构不仅包含了显示模式的位深度，还可以告诉你该显示模式是否使用调色板，以及像素格式。如果dwFlags成员包含了PALETTEINDEXED1、DDPF\_PALETTEINDEXED2、DDPF\_PALETTEINDEXED4、或DDPF\_PALETTEINDEXED8标志，显示模式的位深度为1、2、4或8，并且每个像素是一个相关调色板的索引。如果dwFlags成员包含了DDPF\_RGB标志，那么该显示模式是非调色板式的，并且它的位深度由DDPIXELFORMAT 结构中的dwRGBBitCount成员所提供。

三、设置显示模式

你可以用IDirectDraw2::SetDisplayMode来设置显示器的显示模式。该函数的前四个参数用来描述要设置的显示模式的尺寸、位深度以及显示器的刷新率。函数的第五个参数是用来指定额外的选项，目前，唯一可用的标志是DDSDM\_STANDARDVGAMODE，它将使显示模式被设为Mode 13，而不是Mode X 320x200x8。如果你要设置另一种分辨率，位深度或Mode X模式，不要使用这个参数，并且将其设为0

尽管你可以指定所需要的显示模式的位深度，但是你不能指定显示硬件的像素格式。要测定显示硬件用于该位深度的RGB位掩码，在设置好显示模式之后，调用IDirectDraw2::GetDisplayMode。如果当前的显示模式不是基于调色板的，你可以从dwRBitMask、dwGBitMask和dwBBitMask中获得掩码值。

要正确的测定red、green和blue的所在的位，请参阅“DirectDraw参考手册”中的“像素格式掩码”。

显示模式可以由多于一个的应用程序改变，只要它们共享同一块显卡。只有当应用程序拥有对DirectDraw对象独占的访问，你才可以改变显示模式的位深度。当显示模式被改变的时候，所有的DirectDrawSurface对象将丢失它们的页面内存，并且对任何操作不起反应。这时，一个页面的内存必须被重新分配，调用IDirectDrawSurface3::Restore函数。

必须重声的是：DirectDraw的独占模式并不阻止其它的应用程序分配DirectDrawSurface对象对显示模式或调色板的访问。

四、还原显示模式

在应用程序结束的时候，你可以明确的调用IDirectDraw2::RestoreDisplayMode函数，使显示器还原到原始的显示模式。如果你使用的是IDirectDraw2::SetDisplayMode函数来改变显示模式，并且应用程序具有独占的控制级，那么，当重新设置控制级为普通时，原始的显示模式会自动还原。如果你使用的是IDirectDraw::SetDisplayMode函数，那么你必须明确的调用RestoreDisplayMode以还原显示模式。

五、ModeX与Mode 13显示模式

DirectDraw同时支持Mode 13和Mode X显示模式。Mode 13是一种线性不可换页的320x200x8的基于调色板的显示模式，因为它的16进制BIOS模式编号是13，而被广泛的称之为Mode 13模式。要得到更多关于它的资料，请参阅“Mode 13的支持”。Mode X是从标准的VGA Mode 13模式演化而来的。通过使用VGA显示适配器的EGA多图象平面系统，它允许开发者使用最多可达256K字节的视频RAM（而Mode 13仅为64K）。

在Windows 95系统上，DirectDraw为所有的显示卡提供了两种Mode X模式：320x200x8和320x240x8。某些显卡同样也支持线性低分辨率模式。在这种模式中，主页面可以被锁定和直接访问，这在Mode X模式中是不可能的。

应用程序在调用IDirectDraw2::SetCooperativeLevel函数时，只有使用了DDSCL\_ALLOWMODEX、DDSCL\_FULLSCREEN、和DDSCL\_EXCLUSIVE标志符，才能使用Mode X模式。如果没有指定DDSCL\_ALLOWMODEX标志符，IDirectDraw2::EnumDisplayModes将不会列举出Mode X模式，并且调用IDirectDraw2::SetDisplayMode函数以请求一个Mode X模式，将会失败。

Windows 95和Windows NT不直接支持Mode X模式，因此，当你的应用程序处于Mode X模式时，你将无法使用IDirectDrawSurface3::Lock或IDirectDrawSurface3::Blt以锁定或Blit到主页面。你同样也不能对主页面，或GDI的屏幕设备环境使用IDirectDrawSurface3::GetDC函数。Mode X模式是在DDSCAPS结构中由DDSCAPS\_MODEX标志符指定的，该结构是DDSURFACEDESC结构的一部分（DDSURFACEDESC结构是由IDirectDrawSurface3::GetCaps和IDirectDraw2::EnumDisplayModes函数返回的）。

目前，Windows NT还不能支持Mode X模式和某些线性低分辨率模式。

六、对高分辨率和真彩色的支持

DirectDraw支持显示设备驱动所支持的所有屏幕分辨率和色彩位深度。DirectDraw允许应用程序改变显示模式到计算机显示驱动所支持的任何一个模式，包括24或32位色彩模式（也称为真彩色）。

DirectDraw同样也支持对真彩色页面的硬件仿真层（HEL）的Blit操作。如果显示设备驱动支持这些分辨率的Blit操作，那么硬件Blitter（位块传送器）将被用来进行视频RAM对视频RAM的Blit操作。否则，HEL将被用来完成此项操作。

       Windows 95和Windows NT允许用户指定它们所使用的显示器类型。DirectDraw核对已知的显示模式与已安装的显示器所限制使用的显示模式。如果DirectDraw发现所请求的模式与显示器不兼容，对IDirectDraw2::SetDisplayMode函数的调用失败。当你调用IDirectDraw2::EnumDisplayModes函数时，只有显示器支持的模式才可以被列举出来。

第四节 DirectDraw对象

该节包含了关于DirectDraw对象的信息，以及如何通过IDirectDraw或IDirectDraw2接口对该对象进行操作。

一、什么是DirectDraw对象？

DirectDraw对象是所有DirectDraw应用程序的核心，并且与Direct3D应用程序形成一个整体。它是你要创建的第一个对象，通过它，你可以创建所有其它相关的对象。典型的，通过调用DirectDrawCreate函数可以创建一个DirectDraw对象，它代表了IDirectDraw接口。如果你想使用该接口的另一个更高级的版本（比如：IDirectDraw2接口），以获得更加优秀的性能，你可以请求获得该接口。应注意的是，你可以创建若干个DirectDraw对象，每一个都代表了系统已安装的显示设备。

DirectDraw对象代表显示设备，并且可以利用硬件的加速特性。如果DirectDraw对象所实例化的显示设备具有硬件加速，则该对象是硬件加速的。DirectDraw对象可以创建三种对象：DirectDrawSurface（页面）、DirectDrawPalette（调色板）、和DirectDrawClipper（裁剪器）创建这些对象的函数分别是：IDirectDraw2::CreateSurface、IDirectDraw2::CreatePalette和IDirectDraw2::CreateClipper。

每次，多于一个的DirectDraw对象可以被实例化。最简单的例子是在Windows 95系统上使用两台显示器。尽管Windows 95并不支持双显示器，但为每一种显示驱动程序配置一个DirectDraw硬件抽象层（HAL）是可能的。当缺省的DirectDraw对象被实例化时，Windows 95和GDI将使用它所认识的显示驱动程序。Windows 95和GDI不认识的显示驱动程序可与另外一个设备相匹配，独立的DirectDraw对象必须通过第二个显示驱动程序的全局唯一标志符（GUID）来创建。这个GUID可由DirectDrawEnumerate函数获得。

DirectDraw对象管理它所创建的所有对象。它控制缺省的调色板（如果主页面是8位色彩模式）、缺省的关键色，和硬件显示模式。它跟踪哪些资源已经被分配了，以及哪些资源正有待分配。

二、IDirectDraw2接口的新特性？

IDirectDraw2接口扩展了原先IDirectDraw接口的函数性：增加了一个IDirectDraw2::GetAvailableVidMem函数。该函数可以询问所有可用的视频RAM容量值，和当前可以为某种指定类型的页面所用的空余视频RAM容量值。

DirectX使用COM模型表明，可以通过提供新的接口而给旧的接口加入新的函数特性。在DirectX3的版本中，IDirectDraw2接口取代了原先的IDirectDraw接口。这个新的接口可以通过调用IDirectDraw::QueryInterface方法来获得，如下例所示：

……

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//本例程片段演示如何创建一个IDirectDraw2接口.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LPDIRECTDRAW lpDD;

LPDIRECTDRAW2 lpDD2;

//首先创建一个IDirectDraw接口

ddrval = DirectDrawCreate(NULL, &lpDD, NULL);

if(ddrval != DD\_OK)

return;

ddrval = lpDD->SetCooperativeLevel(hwnd, DDSCL\_NORMAL);

if(ddrval != DD\_OK)

return;

//获得新的接口

ddrval = lpDD->QueryInterface(IID\_IDirectDraw2, (LPVOID \*)&lpDD2);

if(ddrval != DD\_OK)

return;

ddscaps.dwCaps = DDSCAPS\_OFFSCREENPLAIN;

ddrval = lpDD2->GetAvailableVidMem(&ddscaps, &total, &free);

if(ddrval != DD\_OK)

return;

       这个子演示了用C++创建一个IDirectDraw接口的方法，然后再调用IDirectDraw::QueryInterface方法来创建一个IDirectDraw2接口。这个新接口包含了IDirectDraw2::GetAvailableVidMem函数，而试图从IDirectDraw接口调用该函数将会在编译的时候导致一个错误。

IDirectDraw2::GetAvailableVidMem是唯一一个被加入IDirectDraw接口的新方法。而且，IDirectDraw2::SetDisplayMode 和 IDirectDraw2::EnumDisplayModes，这两个方法被修改和扩展。

IDirectDraw::SetCooperativeLevel 和 IDirectDraw::SetDisplayMode之间的相互关系与 IDirectDraw2::SetCooperativeLevel 和 IDirectDraw2::SetDisplayMode之间的相互关系有一些改变。如果你使用旧的IDirectDraw接口，而且应用程序通过调用设置有DDSCL\_EXCLUSIVE标志的IDirectDraw::SetCooperativeLevel取得了独占（全屏）显示模式，用IDirectDraw::SetDisplayMode来改变模式，再调用设置有DDSCL\_NORMAL标志的IDirectDraw::SetCooperativeLevel来释放独占模式，原始的显示模式不会被还原。除非应用程序明确的调用IDirectDraw::RestoreDisplayMode方法或DirectDraw对象被销毁时，原始的显示模式才能恢复。然而，如果你使用新的IDirectDraw2接口，然后按着与上面同样的方法，当DirectDraw对象失去独占模式时，显示器的原始显示模式将会自动被恢复。

因为有些接口可能会因新接口的发布而该动，混合使用一个接口和它的替代者的方法（比如IDirectDraw 与IDirectDraw2）可以导致意想不到的错误。你必须只使用某接口的同一个版本的函数或方法。

三、单进程的多DirectDraw对象

DirectDraw允许一个进程在需要的时候，可以任意多次的调用DirectDrawCreate函数。每次调用后，返回一个唯一的与设备无关的接口。每一个DirectDraw对象可以随心所欲的使用；在对象与对象之间没有相互依赖的关系。每个对象的行为就象它是由一个唯一的进程创建的。

因为DirectDraw对象之间是不相互依赖的，由一个特定的DirectDraw对象创建的DirectDrawSurface、DirectDrawPalette和DirectDrawClipper对象不应该与其它的DirectDraw对象一起使用，因为这些对象会在它的DirectDraw对象销毁时而自动被释放。如果它们与其它的DirectDraw对象一起使用，调用它们的函数将在它们被销毁时退出。

例外的是，DirectDrawClipper对象是由DirectDrawCreateClipper函数创建的。这些对象与任何特定的DirectDraw对象是没有关系的，可以与一个或多个DirectDraw对象一起使用。

四、使用CoCreateInstance创建DirectDraw对象

除了用常规的DirectDrawCreate方法创建一个DirectDraw对象外，你还可以使用CoCreateInstance函数，再调用IDirectDraw2::Initialize来创建一个DirectDraw对象。以下的例程片段描述了这个方法的各步骤。

第一步、在程序的最开始调用CoInitialize来初始化COM对象，参数为NULL。

if ( FAILED( CoInitialize( NULL )))

return FALSE;

第二步、然后，调用CoCreateInstance和IDirectDraw2::Initialize来创建DirectDraw对象。

ddrval = CoCreateInstance( &CLSID\_DirectDraw,

NULL, CLSCTX\_ALL, &IID\_IDirectDraw2, &lpdd );

在CoCreateInstance函数中，第一个参数CLSID\_DirectDraw，是DirectDraw驱动对象类的类标志符；IID\_IDirectDraw2参数指定了要创建的特定的DirectDraw对象；最后的lpdd参数接收创建的对象。如果调用成功，这个函数返回一个没有初始化的对象。

第三步、在你使用这个DirectDraw对象之前，你必须调用IDirectDraw2::Initialize。在此之后，你就可以操作和释放该对象，就象它是用DirectDrawCreate创建的一样。如果在使用之前，你没有调用IDirectDraw2::Initialize，将返回DDERR\_NOTINITIALIZED的错误。

if( !FAILED ( ddrval ))

ddrval = IDirectDraw2\_Initialize( lpdd, NULL );

第四步、在关闭应用程序之前，使用CoUninitialize来关闭COM。

CoUnitialize();

第五节页面

该节包含了关于DirectDrawSurface对象的信息.

一、页面的基本概念

1、什么是页面？

页面，或被我们称作DirectDrawSurface对象，代表了内存里的一个连续的线性的数据区。这个数据区可以被代表显示硬件的DirectDraw对象所识别和确认。通常，DirectDrawSurface对象被置于显卡上的视频RAM中，而这并不是绝对的。除非明确的指定是在视频RAM还是系统RAM中创建DirectDrawSurface对象，DirectDraw可以将其放置在其中任一位置，条件是这样可以获得最佳性能。

DirectDrawSurface对象可以从显卡上的特效处理器上获得好处，不仅仅是通常意义上的加快处理速度，而是可以与系统CPU并行工作，以达到最优的效率和速度。

调用IDirectDraw2::CreateSurface函数可以创建若干类型的DirectDrawSurface对象，包括最简单的单页面对象，复杂的由若干个页面组成的换页链，以及三维页面等等。CreateSurface函数创建我们所请求的页面或换页链，并且返回指向主页面的IDirectDrawSurface接口的指针，通过该接口可以暴露DirectDrawSurface对象的函数性。如果你想使用该接口的较高级的版本，如IDirectDrawSurface3，你也可以询问系统并且得到它。

IDirectDrawSurface3接口通过Blit函数可以使你间接地访问页面内存，例如：IDirectDrawSurface3::BltFast函数。DirectDrawSurface对象可以创建Windows的GDI设备环境句柄（HDC），这样，就可以允许使用Win32的API函数来访问代表DirectDrawSurface对象的页面。GDI识别这些HDC（设备环境句柄），如果它们存在于视频RAM中，那么就可以获得硬件的加速特性。除此之外，你还可以使用IDirectDrawSurface3接口的函数直接访问页面内存。例如：可以使用IDirectDrawSurface3::Lock函数锁定页面内存，并且获得指向该页面上相应区域（用户指定的矩形区域）的内存区的地址。视频RAM上的地址可以指向可见的帧缓存（存储了当前显示画面的缓冲区，也称作主页面），也可以是不可见的缓存（离屏页面或覆盖页面）。不可见的缓存通常被置于视频RAM中，但是如果是受硬件限制或DirectDraw正以仿真模式运行，它也可以被置于系统RAM中。IDirectDrawSurface3接口还扩展了另外一些函数，比如可以用来设置或获得调色板的函数，专门用于某特定类型页面的函数（如换页链或覆盖页面）。

从下面这个例图中，你可以看到所有的DirectDrawSurface页面对象都是由DirectDraw对象创建的，并且与调色板协同工作。尽管每一个页面对象都可以被分配一个调色板，除了像素格式的位深度小于等于8的主页面以外，调色板并不总是必须的。

2、页面接口

前面已经提到过，DirectDrawSurface对象是通过IDirectDrawSurface、IdirectDrawSurface2和IDirectDrawSurface3接口来暴露其函数性的。接口的每一个新的版本与旧的版本相比，除了提供所有原有的函数并且扩充其功能之外，还提供了一些新的函数。

三种接口中，IDirectDrawSurface接口是最早的一个版本，当你调用IDirectDraw2::CreateSurface函数时，系统会为你缺省地创建一个该接口的页面对象。要利用新版接口的函数性，你必须通过调用QueryInterface函数来询问是否存在新版本，并获得它。下面的例程为你展示了这是怎样完成的。

LPDIRECTDRAWSURFACE lpSurf;

LPDIRECTDRAWSURFACE2 lpSurf2;

//填充页面结构

memset(&ddsd, 0, sizeof(ddsd)); //调用Win32 API函数清空ddsd结构

ddsd.dwSize = sizeof(ddsd);

ddsd.dwFlags = DDSD\_CAPS | DDSD\_WIDTH | DDSD\_HEIGHT;

ddsd.ddsCaps.dwCaps = DDSCAPS\_OFFSCREENPLAIN | DSAPS\_SYSTEMMEMORY;

ddsd.dwWidth = 10;

ddsd.dwHeight = 10;

//创建页面，该页面使用IDirectDrawSurface接口

ddrval = lpDD2->CreateSurface(&ddsd, &lpSurf, NULL);

if(ddrval != DD\_OK)

return;

//询问IDirectDrawSurface3接口

ddrval = lpSurf->QueryInterface( IID\_IDirectDrawSurface3, (LPVOID \*)&lpSurf2);

if(ddrval != DD\_OK)

return;

//调用IDirectDrawSurface3接口特有的函数

ddrval = lpSurf2->PageLock(0);

if(ddrval != DD\_OK)

return;

ddrval = lpSurf2->PageUnlock(0);

if(ddrval != DD\_OK)

return;

       上面的例子通过调用QueryInterface函数（指定IID\_IDirectDraw2引用标志符）获得一个DirectDrawSurface对象的IDirectDrawSurface3接口。要得到IDirectDrawSurface3接口，使用IID\_IDirectDrawSurface3引用标志符即可。

3、宽度（Width）和宽距（Pitch）

如果你的应用程序要写视频RAM，内存中的位图并不需要占据连续的内存块。在这种情况下，一条线的width和pitch含义是不同的。width是指内存中位图的一条线的开始和结束位置的内存地址之差。这个距离只代表了内存中位图的宽度，它不包括位图中到达下一条线开始位置所需要的任何额外的内存。pitch是指内存中位图的一条线到下一条线开始位置的内存地址之差。

对矩形内存来说，比如，视频RAM的pitch将包括位图的宽度加上一部分缓存。下面的例图表示了矩形内存中width和pitch的区别。

在这个例图中，前台缓存和后台缓存大小都是640x480x8，高速缓存是384x480x8。要到达下一条线的地址，你必须在640后加上384，得到1024，这就是下一条线的地址。

因此，当直接向页面内存中着色时，一般用IDirectDrawSurface3::Lock（或IDirectDrawSurface3::GetDC）方法返回的pitch值。不要认为pitch只是基于显示模式的。如果你的应用程序在某些显示器上发生显示混乱，这多半是因为pitch使用错误造成的。

4、关键色

DirectDraw支持带源或目标关键色的Blit操作和覆盖页面。这个关键色可以是单个的颜色值，也可以是一个颜色范围。要得到关于关键色的详细介绍，请参阅前一章的“关键色”一节。通过调用IDirectDrawSurface3::SetColorKey函数，可以为一个页面设置一个关键色。

源关键色（Source color key）指定了一个颜色或一个颜色范围，在Blit过程中，不被复制，或在覆盖页面中，对目标层来说是不可见的。目标关键色（Destination color key）指定了一个颜色或一个颜色范围，在Blit过程中，将被替换，或在覆盖页面中，将被目标层所覆盖。源与目标关键色的一个显著的区别就是：源关键色指定了在源页面上什么是可以读和什么是不可以读的；目标关键色指定了在目标页面上，什么是可以写和什么是不可以写的。如果目标页面有关键色，则只有那些符合关键色的像素可以被改变（在Blit操作中），或被覆盖（在覆盖页面中）。

除了与Blit相关的关键色之外，覆盖页面还可以使用覆盖关键色。要得到更多信息，请参阅“覆盖关键色”。

有些硬件只支持YUV像素数据的颜色范围。YUV数据通常用于视频显示的像素格式，并且其透明背景不是一个特定颜色而导致在数值转换过程中发生错误。所以，只要可能，就应该将数据写到一个特定的透明颜色上，而不管它是什么像素格式。

关键色是按页面的像素格式指定的。如果一个页面是调色板格式，关键色是以一个调色板索引或一组调色板索引指定的。如果页面的像素格式是按FOURCC代码指定的，描述了一个YUV格式，YUV关键色是由DDCOLORKEY结构的dwColorSpaceLowValue 和dwColorSpaceHighValue成员的低三位字节指定的。最低的字节包含V数据，下一个包含U数据，第三个包含Y数据。IDirectDrawSurface3::SetColorKey的dwFlags参数指定了关键色是用在Blit操作中还是覆盖页面中，以及它是源还是目标关键色。以下是一些合法的关键色的例子。

8位调色板模式：

//调色板登录项26是关键色。

dwColorSpaceLowValue = 26;

dwColorSpaceHighValue = 26;

24位真彩模式：

//（255，128，128）颜色是关键色

dwColorSpaceLowValue = RGBQUAD(255,128,128);

dwColorSpaceHighValue = RGBQUAD(255,128,128);

FourCC YUV模式：

//只要Y在100和110之间，并且U或V在50和55之间的的任何一个YUV颜色为透明。

dwColorSpaceLowValue = YUVQUAD(100,50,50);

dwColorSpaceHighValue = YUVQUAD(110,55,55);

5、像素格式

像素格式规定了页面内存中的每个像素的数据是怎样进行编码的。DirectDraw使用DDPIXELFORMAT结构来描述各式各样的像素格式（请参阅“DirectDraw参考手册中关于该结构的帮助”）。DDPIXELFORMAT结构中的成员包含了各种像素格式相互区别的以下几个显著的特点：

像素格式是基于调色板的还是非调色板式的

如果是非调色板式，像素是RGB，还是YUV格式

位深度

位掩码

通过调用IDirectDrawSurface3::GetPixelFormat函数，你可以获得关于当前页面的像素格式的信息。

二、创建页面

DirectDrawSurface对象代表了一个页面，调用IDirectDraw2::CreateSurface函数可以创建一个DirectDrawSurface对象，也可以同时创建由若干个页面组成的复杂页面结构，最典型的就是换页链。调用该函数时，需要提供要创建的页面的描述，如页面的尺寸、是单个页面还是复杂页面、所采用的像素格式（如果页面将不使用索引调色板）等等。所有这些描述信息存储在一个DDSURFACEDESC结构中，在调用CreateSurface函数时必须提供。如果硬件不支持所请求的页面特性，或是要创建的页面已经存在，则该函数返回一个错误。

要创建单个的页面或若干个页面其实是一项并不复杂的工作，只需要少许的几行代码即可。主要有以下四种类型的页面可以被创建：

创建主页面

创建离屏页面

创建复杂页面和换页链

创建超宽页面

       在缺省的情况下，DirectDraw试图在本地的视频RAM中创建页面。如果恰好没有足够的本地（local）视频RAM来容纳要创建的页面的话，DirectDraw将试图使用非本地（non-local）的视频RAM（在某些装备了AGP设备的系统上），如果仍旧无法实现，那么DirectDraw将只能将其创建于系统RAM中。当然，在调用CreateSurface函数时，你也可以在相关的DDSCAPS结构中指定适当的标志符向DirectDraw明确的表明你想将页面置于哪种类型的内存中。

1、创建主页面

主页面（primary surface）代表的是在显示器的当前可见屏幕，它在页面描述中具有PRIMARYSURFACE标志符。对于每一个DirectDraw对象来说，你只可能拥有一个主页面。

主页面对用户来说是可见的。当你创建一个主页面时，实际上，你创建的这个DirectDrawSurface对象，访问的是由GDI正在使用的已经可见的页面（即显示屏幕），主页面的大小以及像素格式暗中符合当前显示器的显示模式。因此，尽管创建所有其它类型的页面要求填充DDSURFACEDESC结构的dwHeight 和 dwWidth值以及像素格式，而创建主页面时一定不能自己指定它们，甚至你知道它们与当前屏幕是同样大小，否则，CreateSurface函数将调用失败，并且返回DDERR\_INVALIDPARAMS。要创建一个主页面，DDSURFACEDESC 结构（以下为ddsd )的成员是如下填充的。

DDSURFACEDESC ddsd;

ddsd.dwSize = sizeof(ddsd);

//告诉DirectDraw哪些成员是可用的。

ddsd.dwFlags = DDSD\_CAPS;

//请求一个主页面

ddsd.ddsCaps.dwCaps = DDSCAPS\_PRIMARYSURFACE;

       然后，就可以调用CreateSurface函数并且提供了这个页面描述，如果函数调用成功，可以返回一个指向主页面的指针。如果你想要得到关于这个主页面的尺寸和像素格式信息，应该调用IDirectDrawSurface3::GetSurfaceDesc函数。相关信息，请参阅“显示模式”。

2、创建离屏页面

离屏页面（off-screen surface），通常被用来存储位图，用于后来的将位图图象Blit到主页面或后台缓存上。因为离屏页面是一个相互独立的页面，不与任何对象产生隶属关系，所以你必须指定你所要创建的离屏页面的大小，这是通过在DDSURFACEDESC结构中包含进DDSC\_WIDTH和DDSD\_HEIGHT标志符，并且给dwWidth和dwHeight成员填充正确的参数完成的。除此之外，你还必须包含进DDSCAPS\_OFFSCREENPLAIN标志符以表明创建的是一离屏页面。

在缺省的情况下，DirectDraw将会在视频RAM中创建离屏页面，除非视频RAM容量不够，那么，DirectDraw将会把离屏页面置于系统RAM中。你可以在DDSURFACEDESC结构的dwCaps成员中包含进DDSCAPS\_SYSTEMMEMORY或DDSCAPS\_VIDEOMEMORY标志符，以明确的表明你希望将页面置于何处。如果DirectDraw不能满足你所提供的要求，CreateSurface函数将调用失败，并且返回错误。

下面的例程展示了要创建一个离屏页面，DDSURFACEDESC 结构（以下为ddsd )的成员是如何填充的。

DDSURFACEDESC ddsd;

ddsd.dwSize = sizeof(ddsd);

//告诉DirectDraw哪些成员是可用的

ddsd.dwFlags = DDSD\_CAPS | DDSD\_HEIGHT | DDSD\_WIDTH;

//请求一个离屏页面，大小为100x100

//（这假定了要创建的离屏页面的像素格式将符合主页面的像素格式）

ddsd.ddsCaps.dwCaps = DDSCAPS\_OFFSCREENPLAIN;

dwHeight = 100;

dwWidth = 100;

       除此之外，你也可以创建一个页面的像素格式与主页面不同的离屏页面。然而，在这种情况下，有一个缺点该离屏页面将只能被限制于系统RAM中。下面的例程片段展示了如何准备DDSURFACEDESC结构的各成员，以用于创建一个8位的调色板式页面（假定当前的显示模式不是8位格式）。

ZeroMemory(&ddsd, sizeof(ddsd));

ddsd.dwSize = sizeof(ddsd);

ddsd.dwFlags = DDSD\_CAPS | DDSD\_HEIGHT | DDSD\_WIDTH |

DDS\_PIXELFORMAT;

ddsd.ddsCaps.dwCaps = DDSCAPS\_OFFSCREENPLAIN |

DDSCAPS\_SYSTEMMEMORY;

ddsd.dwHeight = 100;

ddsd.dwWidth = 100;

ddsd.ddpfPixelFormat.dwSize = sizeof(DDPIXELFORMAT);

ddsd.ddpfPixelFormat.dwFlags = DDPF\_RGB | DDPF\_PALETTEINDEXED8;

// 设置页面的位深度为8，但是绝对不要设置任何RGB mask值，

// 因为对调色板式页面来说，该值一定是0。

ddsd.ddpfPixelFormat.dwRGBBitCount = 8;

       在DirectX以前的版本中，离屏页面的宽度的最大值被限制于主页面的宽度值之内。而在DirectX 5.0版中，你可以随心所欲的创建任何宽度的离屏页面，如果显示硬件能够承受的话。在请求一个超宽离屏页面时，要小心的是，如果显卡上的内存不能够容纳该页面，页面将被置于系统RAM中。如果你明确的指定在视频RAM中创建超宽页面，而硬件又无法承受的话，调用失败。要得到更多关于超宽页面的信息，请参阅“创建超宽页面”。

3、创建复杂页面和换页链

除了上面介绍的由单独的一个页面组成的主页面和离屏页面之外，你还可以创建由若干个页面组成的复杂页面（complex surfaces），它同样也是由一步调用IDirectDraw2::CreateSurface函数所创建的。如果你在页面描述中设置了DDSCAPS\_COMPLEX标志符，那么在调用CreateSurface函数后，DirectDraw除了创建你所明确要创建的页面之外，还将暗中为你创建一个或多个附加页面。对复杂页面的管理与对单页面的管理基本上是没有区别的：一步调用IDirectDraw::Release函数将释放复杂页面中所有的页面，并且一步调用IDirectDrawSurface3::Restore函数将恢复所有页面。然而，这些暗中创建的页面不能被脱离，即解除隶属关系，要得到更多的信息，请参阅“DirectDraw参考手册”中关于IDirectDrawSurface3::DeleteAttachedSurface函数的帮助。

你所能创建的最常用的复杂页面之一就是换页链（flipping chain）。通常，一个换页链是由一个主页面以及隶属于它的若干个后台缓存组成。DDSCAPS\_FLIP标志符表明页面是一个换页链的一部分。创建一个换页链的页面描述中，同样必须包含进DDSCAPS\_COMPLEX标志符。

下面的例程片段展示了要创建一个换页链结构，如何填充页面描述的各成员。

DDSURFACEDESC ddsd;

ddsd.dwSize = sizeof(ddsd);

// 告诉DirectDraw哪些成员是可用的

ddsd.dwFlags = DDSD\_CAPS | DDSD\_BACKBUFFERCOUNT;

// 请求一个主页面，以及后台缓存数量为1

ddsd.ddsCaps.dwCaps = DDSCAPS\_COMPLEX | DDSCAPS\_FLIP | DDSCAPS\_PRIMARYSURFACE;

ddsd.dwBackBufferCount = 1;

       上面的例程构造了一个双缓冲区换页（double-buffered flip）环境：单步调用IDirectDrawSurface3::Flip函数可以交换主页面和后台缓存。如果给DDSURFACEDESC 结构的dwBackBufferCount成员设置了2，将会创建两个后台缓存，在每次调用Flip函数时，主页面将在三个页面间循环，这就构成了一个三缓冲区换页（triple-buffered flip）环境。

4、创建超宽页面

DirectDraw允许你在显卡内存中创建超宽离屏页面（页面的宽度大于主页面）。这只在显示设备支持超宽页面的情况下才能实现。

要判断DirectDraw是否支持超宽页面，调用IDirectDraw2::GetCaps函数，检查你所提供的第一个DDCAPS结构的dwCaps2成员中是否存在DDCAPS2\_WIDESURFACES标志。如果存在，表明你的DirectDraw支持超宽页面。

如果你试图在视频RAM中创建一个宽度大于主页面的页面，而DDCAPS2\_WIDESURFACES标志并不存在，函数调用将失败，并且返回一个DDERR\_INVALIDPARAMS错误。

超宽页面通常被系统RAM页面、视频端口页面、和可执行缓存所支持。

三、换页

DirectDraw里的任何页面都可以构造为换页页面（Flipping surface）。一个换页页面是位于内存里的任何一个可以在前台缓存（front buffer）和后台缓存（back buffer）之间交换的页面，这个换页环境就是我们所称的换页链（flipping chain）。通常，前台缓存指的就是主页面，当然，这并不是绝对的。

典型的，当你调用IDirectDrawSurface3::Flip函数以请求一次换页操作，指向主页面和后台缓存的指针相互交换。这就是说，换页的操作，是通过交换显示设备用来代表页面内存的指针，而不是通过相互复制页面的实际内存来完成的。但是，也有例外的时候，那就是当DirectDraw以仿真的方式进行换页操作时，在这种情况下，它所做的就是简单的相互复制页面内存。只有在后台缓存不能容纳进视频RAM，或硬件不支持DirectDraw的时候，DirectDraw才会以仿真的方式进行换页操作，当然，这只是极其少见的情况。

当换页链中包含了一个主页面和一个以上的后台缓存时，在换页操作中，指向它们的指针将按前后顺序依次转换。如下图所示：

隶属到DirectDraw对象上的其它类型的页面，只要不是换页链中的一部分，在换页过程中都不会受到任何影响。

请牢记，DirectDraw进行换页，是通过交换指向DirectDrawSurface对象的指针。而不是交换DirectDrawSurface对象本身。这意味着，在任何类型的换页方案中，如果你想将图象Blit到后台缓存，你所使用的始终是同一个DirectDrawSurface对象，而不用去考虑原先的后台缓存已经换页到哪儿了。同样地，你应该始终使用主页面作为Flip函数的调用者，以完成一次换页操作，而不用去管最开始的主页面换页到哪儿了。

当换页对象是可见的页面，比如主页面换页链或一个可见的覆盖页面换页链，进行换页的Flip函数与系统CPU是异步执行的。这就是说，在这些可见的页面上，调用Flip函数，它只是简单的告诉显示硬件该进行换页了，并不需要等待换页操作在硬件设备中实际完成后才返回。这是因为显示硬件（显示器）只有在完成一次垂直刷新后才能进行一次换页。所以，Flip函数调用成功，并不意味着换页已经完成，在实际的换页操作进行之前，对即将成为主页面的后台缓存是不能锁定和进行Blit操作的，如果在这时调用以下这些函数，调用将失败，并且返回DDERR\_WASSTILLDRAWING的错误，如IDirectDrawSurface3::Lock、IDirectDrawSurface3::Blt、IDirectDrawSurface3::BltFast和IDirectDrawSurface3::GetDC。但是，对于三缓冲区换页环境，最后一个后台缓存仍是可用的。

要让Flip函数成为与系统CPU同步的操作，在调用时指定DDFLIP\_WAIT标志即可。

四、页面丢失

当代表页面内存的DirectDrawSurface对象被不得已的释放时，与该对象相关联的页面内存也会被释放。当一个DirectDrawSurface对象丢失其页面内存的时候，它的许多函数将返回DDERR\_SURFACELOST，并且不进行任何其它操作。

页面可能被丢失是因为：.显示设备（显示器）显示模式的改变，或另一个应用程序获得了对显卡的独占访问模式，并且释放了显卡上当前被分派其它应用程序的所有页面内存。对页面调用IDirectDrawSurface3::Restore方法可以为这些丢失了内存的页面重新分配内存，并且将这些内存与DirectDrawSurface对象联系上。重建内存并不会使以前存在于该页面上的图象重新显现出来，因此，如果你的页面丢失了其内存，在调用Restore函数重建之后，必须亲手重新绘制所有的图象。

要得到更多资料，请参阅“设置显示模式”。

五、释放页面

与所有的COM接口一样，在你不再需要某页面的时候，你可以通过调用Release方法释放它。

每一个单独创建的页面必须逐个的明确的释放掉。然而，如果页面是通过单步调用IDirectDraw2::CreateSurface或IDirectDraw::CreateSurface函数创建一个多页面结构（例如一个换页链）时暗中形成的，那么，你只需要明确的释放前台缓存就可以了。在这种情况下，所有的后台缓存都被暗中的释放了，指向它们的指针将不再合法。

六、更新页面属性

你可以通过调用IDirectDrawSurface3::SetSurfaceDesc函数来更新一个现存页面的属性。有了这个函数，你可以更改页面的像素格式，还可以使该DirectDrawSurface对象指针重定位，使其指向一块应用程序已经明确分配了的系统RAM。这是很有用的，因为它使得你的页面可以直接使用一个已经存在的缓冲区的数据，而不用进行复制操作。新的页面内存是由客户程序所分配，同样的，这些内存也必须由客户程序释放掉。要得到更多关于如何使用SetSurfaceDesc函数的资料，请参阅“DirectDraw参考手册”中关于此函数的帮助。

在调用IDirectDrawSurface3::SetSurfaceDesc函数时，lpddsd参数必须是一个DDSURFACEDESC结构的地址，描述了新的页面内存并且提供了指向该内存的指针。在这个结构中，你只能设置dwFlags成员为反映了页面内存的地址、大小、宽距、和像素格式的标志符。因此，dwFlags只能是以下标志符的集合：DDSD\_WIDTH、DDSD\_HEIGHT、DDSD\_PITCH、DDSD\_LPSURFACE、和DDSD\_PIXELFORMAT。

在向DDSURFACEDESC结构中填充数据之前，你必须为新的页面分配内存。你所分配的内存的大小是非常重要的，它不仅要能容纳满足页面的长和宽所需要的内存，还必须能够容纳页面的宽距，宽距必须是WORD（8位）的倍数。应该注意的是，宽距是以字节为单位，而非像素。

在向DDSURFACEDESC结构中填充数据的时候，lpSurface成员是一个指向你刚分配的内存的指针，并且dwHeight和dwWidth成员描述了页面的大小（以像素为单位）。如果你指定了页面的大小，你还必须填充lPitch成员以反映页面宽距的大小。Pitch必须是DWORD的倍数。同样的，如果你指定了宽距，你还必须为其指定一个宽度值。最后，ddpfPixelFormat成员描述了页面的像素格式。如果你没有给这些成员指定新的值，那么，SetSurfaceDesc函数将使用当前页面的原始值，只有lpSurface成员是例外。

在使用IDirectDrawSurface3::SetSurfaceDesc方法的过程中，你还应当注意到这样一些细节，当然，它们只是常识。举例来说，DDSURFACEDESC结构的lpSurface成员必须是一个指向系统RAM的合法的指针（SetSurfaceDesc函数目前还不支持指向视频RAM的指针）。同样，dwWidth和dwHeight成员的值不能为0。最后一点，你不能为主页面或换页链中的任何页面调用此函数。

你可以将同一块内存设置给若干个DirectDrawSurface对象，但是，你必须注意到，这块内存被所有的页面对象所使用，它不会因为某一个页面的释放而被释放掉。

不正确的使用SetSurfaceDesc函数将导致不可预知的行为。因为DirectDrawSurface对象不会释放并不是它分配的页面内存，因此，当页面内存不再需要的时候，将其及时的释放掉是你的责任。但是，不管怎样，当SetSurfaceDesc函数被调用的时候，DirectDraw将释放掉该页面在创建的时候被暗中分配的原始的页面内存。

七、直接访问帧缓存

一个DirectDrawSurface对象允许应用程序通过调用IDirectDrawSurface3::Lock锁定页面以获得对页面内存的直接的访问。当应用程序调用这个函数的时候，需要给lpDestRect参数提供一个指向RECT结构的指针，描述了页面中你所想要直接访问的矩形区域。如果应用程序需要访问整个页面，设置这个参数为NULL即可。两个线程或进程可以同时锁定同一个页面上的若干个矩形区域，条件是这些矩形区域没有相互重叠。

Lock函数调用成功的话，将填充一个DDSURFACEDESC结构，描述了你要正确的访问页面内存所需要的所有信息。如果页面的像素格式与主页面的不一样，该结构中还包含了关于页面的宽距（pitch）和像素格式的信息。当应用程序结束了对页面内存的访问，可以调用IDirectDrawSurface3::Unlock以解锁页面。

当你锁定了一个页面，你就可以对页面内存中的数据进行直接的操作。以下介绍了一些小技巧，可以避免在页面被锁定的过程中，直接向页面内存进行绘图的时候发生的绝大多数一般的错误。

决不要假想页面的宽距（pitch）为一恒定值，每次调用IDirectDrawSurface3::Lock函数的时候都要检查返回信息中的宽距值。这个值的改变可以有各种各样的原因，包括页面内存在内存中的位置，显卡的类型，甚至是DirectDraw引擎的版本。

确保你要进行Blit操作的目标页面是没有被锁定的。如果对一个锁定的页面调用DirectDraw的Blit函数，调用将失败，并且返回DDERR\_SURFACEBUSY或DDERR\_LOCKEDSURFACES错误。相似的，如果对视频RAM中一个锁定的页面调用了GDI的Blit函数，调用也将失败，但不会返回错误值。

尽量减少你的程序在IDirectDrawSurface3::Lock 和IDirectDrawSurface3::Unlock之间的执行活动。在一个页面被锁定的过程中，DirectDraw通常控制住Win 16锁，于是使得对页面内存的访问得以安全的进行。Win 16锁使对GDI和USER的访问串行化，在Lock和UnLock函数的调用过程中，暂停执行Windows。IDirectDrawSurface3::GetDC函数暗中的调用了IDirectDrawSurface3::Lock以锁定页面，并且IDirectDrawSurface3::ReleaseDC函数暗中的调用了IDirectDrawSurface3::Unlock以使页面解锁。

对齐复制到显示存储器。Windows 95使用了一个页错误处理器棗Vflatd.386文件，来为具有堆交换存储器的显示卡实现虚拟的平面桢缓冲区。该处理器允许这些显示设备为DirectDraw提供线性桢缓冲区。如果复制是跨存储堆的，非对齐复制到显示存储器会导致系统挂起。

锁定页面通常导致DirectDraw控制住Win 16锁。在Win 16锁被控制期间，所有其它的应用程序，包括Windows系统，都会暂停执行。因为这个原因，标准的调试器在此期间都不可能工作，但是只有内核（kenerl）调试器仍可以正常工作。

如果在你调用IDirectDrawSurface3::Lock函数的时候，对于该页面的一次Blit操作还在进行之中，函数将立即返回一个错误值。要防止这种情况的发生，可以在调用Lock函数的过程中指定DDLOCK\_WAIT标志，以表明该函数将等待，直到成功的获得锁定之后才返回。

八、使用非本地视频RAM页面

DirectDraw支持高级图形端口（AGP）结构，所以它可以在非本地的视频RAM中创建页面。在装备了AGP的系统中，如果本地的视频RAM被耗尽，或用户明确的请求非本地的视频RAM，DirectDraw将使用非本地的视频RAM，这依赖于当时的AGP执行模式。

目前，有两种AGP结构的执行模式，这就是通常所说的“执行模式”（execute model）和“DMA模式（DMA model）”。在执行模式中，对于非本地的视频RAM和本地的视频RAM，显示设备支持同样的特性。因此，当你调用IDirectDraw2::GetCaps方法以获取硬件特性的时候，在DDCAPS 结构的dwNLVBCaps、dwNLVBCaps2、dwNLVBCKeyCaps、dwNLVBFXCaps、和dwNLVBRops成员中所包含的与Blit操作相关的标志符，与本地视频RAM的那些标志符是一样的。在执行模式中，如果本地的视频RAM被耗尽，DirectDraw将自动的退回到使用非本地的视频RAM，除非调用者明确的指定使用本地的视频RAM。

在DMA模式执行过程中，对非本地视频RAM的Blit操作和材质贴图（texturing）的支持是有限度的。如果显示设备使用DMA模式，当你询问设备特性的时候，dwCaps2成员将会被设置了DDCAPS2\_NNLOCALVIDMEMCAPS标志。DDCAPS结构中的dwNLVBCaps、dwNLVBCaps2、dwNLVBCKeyCaps、dwNLVBFXCaps、和dwNLVBRops成员中所包含的Blit相关的标志描述了DirectDraw所支持的特性；这些特性通常是本地的视频RAM页面所支持的特性的一个子集。DMA模式下，当本地的视频RAM被耗尽的时候，对于材质页面，DirectDraw将自动的退回到使用非本地的视频RAM，除非调用者明确的请求使用本地的视频RAM。材质页面是唯一的一种会被这样处理的页面，所有其它类型的页面不能被创建在非本地的视频RAM中，除非调用者明确的指定。

DMA执行模式对材质贴图的支持，因非本地视频RAM页面的不同而不同。如果驱动程序支持从非本地视频RAM页面的材质贴图，当你调用IDirect3DDevice2::GetCaps方法询问3-D驱动程序的特性的时候，D3DDEVCAPS\_TEXTURENONLOCALVIDMEM标志符将会被设置其中。

九、色彩和格式转换

非RGB色彩空间的页面格式是由四字符代码（FOURCC codes）定义的。如果应用程序调用IDirectDrawSurface3::GetPixelFormat方法询问页面的像素格式，并且该页面是一个非RGB页面，那么DDPF\_FOURCC标志符将会被设置，并且DDPIXELFORMAT结构中的dwFourCC成员的值是有效的。如果页面的四字符代码（FOURCC code）代表了一个YUV格式，那么DDPF\_YUV标志符也将被设置，并且dwYUVBitCount、dwYBits、dwUBits、dwVBits、和dwYUVAlphaBits成员的值是有效的，它们可以被用来提取出页面像素格式的信息。

如果当前的页面是RGB格式，那么DDPF\_RGB标志符将被设置，并且dwRGBBitCount、dwRBits、dwGBits、dwBBits、和dwRGBAlphaBits成员的值是有效的，它们可以被用来提取出页面像素格式的信息。DDPF\_RGB标志符可以与DDPF\_FOURCC标志符一起使用，如果被描述的页面是非标准的RGB格式。

在色彩和格式转换过程中，两套四字符代码可以为应用程序所使用。一套代表了Blit硬件的特性，另一套代表了覆盖硬件的特性。

要得到更多的信息，请参阅“DirectDraw参考手册”中的“四字符代码”部分的帮助。

十、覆盖页面

该节包含了关于DirectDraw支持覆盖页面的资料。

1）覆盖页面概览

覆盖页面（Overlay），通常也被称作重叠页面或覆盖层，是一种需要特定的硬件支持的页面。覆盖页面通常被用于显示实时视频、视频录制、或静止的位图于主页面之上，而不需要进行Blit操作到主页面上或用任何方法改变主页面的内容。覆盖页面完全是由硬件支持的；DirectDraw支持由显示设备驱动程序报告的任何一种特性。DirectDraw不支持以软件仿真的方式实现覆盖页面。

可以用一张透明的塑料片来比喻一个覆盖页面，你可以将它放在显示器屏幕前，并且在上面进行绘图。同样的道理，当覆盖页面位于显示器屏幕的上层时，你可以同时看到覆盖页面和主页面的内容，当你将覆盖页面挪开时，主页面的内容不会发生任何改动。实际上，正如前面提到的那样，一个覆盖页面的执行机制与一张透明的塑料片基本上是一样的。当你要显示一个覆盖页面的时候，你要告诉设备驱动程序在哪儿和怎样去显示这个覆盖页面。当显示设备在显示器上绘制扫描线的时候，它会检查主页面上的每一个像素的位置，来判断该处是否应该让覆盖页面可见。如果是的话，显示设备从覆盖页面上取得该处的像素颜色，并且显示出来。整个过程如下图所示：

通过这个方法，显示适配器在显示屏幕上产生了一个由主页面和覆盖页面合成的图象，并且在提供透明和缩放特效的同时，不用修改任何一个源页面的内容。这个合成的页面被注入视频数据流中，直接显示在屏幕上。因为这是一个悬空（on the fly-我们暂且这么称呼它）的操作过程，并且像素的转换是在硬件层中完成的，所以对用户来说，在显示覆盖页面的过程中，并没有可以察觉的时间损失。除此之外，这个方法还允许将具有不同的像素格式的主页面和覆盖页面合成起来。

你可以通过调用IDirectDraw2::CreateSurface函数创建一个覆盖页面，在相应的DDSCAPS结构中指定DDSCAPS\_OVERLAY标志符。覆盖页面只能存在于视频RAM中，所以，你还必须包含DDSCAPS\_VIDEOMEMORY标志符。与创建其它类型的页面一样，通过包含进适当的标志符，你可以创建一个单独的覆盖页面，或一个由若干个覆盖页面组成的换页链。

2）DDCAPS结构中的重要成员和标志

通过调用IDirectDraw2::GetCaps方法，你可以获得关于DirectDraw支持的覆盖页面特性的信息。该函数将描述了所有这些特性的信息填充到两个DDCAPS结构中。

当报告硬件特性时，如果硬件对某特定类型的限制加以实施，设备驱动程序会在DDCAPS 结构的dwCaps成员中设置相应的标志符来表明。在获得驱动程序的能力之后，检查dwCaps成员中的标志符，以判断硬件实施了哪些限制。DDCAPS结构中包含了九个描述了硬件对覆盖页面的限制信息的成员。下面的这张表列出了与覆盖页面相关的成员及其相应的标志符。

成员                                        标志符

      dwMaxVisibleOverlays       该成员总是可用的

      dwCurrVisibleOverlays         该成员总是可用的

      dwAlignBoundarySrc           DDCAPS\_ALIGNBOUNDARYSRC

      dwAlignSizeSrc                   DDCAPS\_ALIGNSIZESRC

      dwAlignBoundaryDest          DDCAPS\_ALIGNBOUNDARYDEST

      dwAlignSizeDest                  DDCAPS\_ALIGNSIZEDEST

      dwMinOverlayStretch           DDCAPS\_OVERLAYSTRETCH

      dwMaxOverlayStretch          DDCAPS\_OVERLAYSTRETCH

      dwMaxVisibleOverlays和dwCurrVisibleOverlays成员携带了硬件设备可以显示的覆盖页面的最大个数，以及其中的多少个可以同时被显示出来的数据。

除此之外，硬件设备将覆盖页面矩形的位置和大小限制反映在dwAlignBoundarySrc、dwAlignSizeSrc、dwAlignBoundaryDest、dwAlignSizeDest、和dwAlignStrideAlign成员中。这些成员中的值表明了在显示覆盖页面的时候，你必须怎样控制源和目标矩形的大小和位置。要得到更多的资料，请参阅“源和目标矩形，以及边界和大小限制”。

同样，硬件将缩放系数反映在dwMinOverlayStretch和dwMaxOverlayStretch成员中。要得到更多资料，请参阅“最小和最大缩放系数”。

3）源和目标矩形

要显示一个覆盖页面，调用覆盖页面的IDirectDrawSurface3::UpdateOverlay函数，在dwFlags参数中指定DDOVER\_SHOW标志。该方法需要你在lpSrcRect和lpDestRect参数中指定源和目标矩形。源矩形描述了将显示在主页面上的覆盖页面的区域。要请求该方法使用整个页面，设置lpSrcRect参数为NULL即可。目标矩形描述了在主页面上将被用来显示覆盖页面的区域。

源和目标矩形不需要是同样的大小。通常，你所设置的目标矩形是小于或者大于源矩形的，那么在显示出覆盖页面的时候，硬件将会自动的将其进行缩小或放大。

要成功的显示覆盖页面，你可能需要调整源矩形和目标矩形的大小和位置。这是否必要，取决于你的设备驱动所强加的限制。要得到更多的资料，请参阅“边界和大小限制，以及最小和最大缩放系数”。

4）边界（Boundary）和大小（Size）限制

由于不同的硬件的能力所限，在显示覆盖页面的时候，某些设备驱动对源和目标矩形的位置和大小强加了一些限制。要找出某个设备受到了哪些限制，调用IDirectDraw2::GetCaps函数，并且在DDCAPS的dwCaps成员中检查那些与覆盖页面相关的标志。下面的表展示了表明边界和大小限制的的成员极其标志。

类型                                        标志                     成员变量

      边界（位置）限制              DDCAPS\_ALIGNBOUNDARYSRC  dwAlignBoundarySrc

                                          DDCAPS\_ALIGNBOUNDARYDEST       DwAlignBoundaryDest

      大小限制                            DDCAPS\_ALIGNSIZESRC                     dwAlignSizeSrc

                                  　 DDCAPS\_ALIGNSIZEDEST               DwAlignSizeDest

      有两种类型的限制：边界限制和大小限制。每一种限制都是以像素为单位（而不是以字节为单位），并且可以提供给源和目标矩形。同样，这些限制会因为覆盖页面和主页面的像素格式的不同而不同。

边界限制影响的是你可以放置一个源或目标矩形的位置。dwAlignBoundarySrc和dwAlignBoundaryDest成员所报告的值分别告诉你应如何对齐相应的矩形的左上角。矩形左上角的x坐标（RECT结构的left成员）必须是所报告的该限制值的整数倍。

大小限制影响的是你可以设置一个源或目标矩形的合法的大小。dwAlignSizeSrc和dwAlignSizeDest成员所报告的值分别告诉你应如何对齐相应矩形的宽度（以像素为单位）。源或目标矩形的宽度必须是这个所报告的值的整数倍。如果你按照最小的缩放系数缩放一个矩形，还必须保证缩放后的矩形仍是满足大小限制的。在缩放完成后，用上舍入的方法（只取不舍）调整矩形的的宽度，而不是下舍入，这样才能保证满足最小缩放系数。要得到更多的资料，请参阅“最小和最大缩放系数”。

5）最小和最大缩放系数

由于硬件的能力所限，某些设备对于覆盖页面从源矩形到目标矩形Blit操作的缩放比例存在一定的限制。DirectDraw将这种限制称之为缩放系数。要获得设备缩放系数的数据，调用IDirectDraw2::GetCaps函数，如果DDCAPS成员中设置了DDCAPS\_OVERLAYSTRETCH标志，那么，有关缩放系数的数据就存在dwMinOverlayStretch和dwMaxOverlayStretch成员中。应该注意的是，缩放系数是按照实际数值的1000倍给出的，所以，1300实际上表示缩放系数为1.3，750表示0.75。

对于那些对覆盖页面的缩放不存在限制的设备来说，最小和最大缩放系数的值都被设为0。

最小缩放系数报告的是目标矩形必须至少是源矩形的宽度乘以该最小缩放系数值。如果最小缩放系数比1000大，那么，你所指定的目标矩形必须比源矩形大。举例来说，如果设备报告的最小缩放系数是1300，你必须保证目标矩形的宽度至少是源目标矩形宽度的1.3倍。同样，最小缩放系数小于1000表明目标矩形可以比源矩形小。

最大缩放系数报告了目标矩形必须至多是源矩形的宽度乘以该最大缩放系数。举例来说，如果最大缩放系数为2000，你可以指定目标矩形的大小至多是源矩形大小的两倍。如果最大缩放系数小于1000，那么，你总是必须使目标矩形比源矩形要小，才能正常的实现覆盖页面。

覆盖页面在进行完缩放操作之后，目标矩形还必须满足设备所需要的大小和位置限制。因此，先将目标矩形进行缩放，然后再进行大小和位置的调整是一个好主意。要得到更多的资料，请参阅“边界和大小限制”。

硬件不需要你对目标矩形的高度进行调整。你可以增加一个目标矩形的高度来保证显示正常的外观比例，而不会产生负面影响。

6）覆盖页面的关键色

与其它类型的页面一样，覆盖页面使用了源和目标关键色，以控制完成在不同页面间的透明Blit操作。因为覆盖页面本身在Blit操作时是不可见的，因此，当你调用IDirectDrawSurface3::UpdateOverlay函数的时候，需要一个另外的方法来控制该覆盖页面是如何被显示到主页面之上的。这种要求是通过覆盖页面的关键色来满足的。覆盖页面的关键色，与其它Blit相关的关键色一样，被分为源关键色和目标关键色，可以通过IDirectDrawSurface3::SetColorKey函数来设置。使用DDCKEY\_SRCOVERLAY或DDCKEY\_DESTOVERLAY标志来指定是源关键色还是目标关键色。覆盖页面可以使Blit操作、覆盖页面关键色、以及覆盖页面显示操作同时正常的进行，两种类型的关键色不会发生相互冲突。

IDirectDrawSurface3::UpdateOverlay函数使用覆盖页面的源关键色来指定在覆盖页面上哪些像素被视为透明，以使在这些像素覆盖下的主页面的内容可以被显示出来。同样的，该函数还使用覆盖页面的目标关键色来指定在主页面上哪些部分可以被覆盖页面所占据。产生的这些视觉效果与由Blit相关的关键色的是一样的。要得到更多的资料，请参阅“透明Blit，以及源和目标关键色”。

7）覆盖页面的定位

在调用IDirectDrawSurface3::UpdateOverlay函数初始化显示一个覆盖页面之后，你可以通过调用IDirectDrawSurface3::SetOverlayPositio函数来重新定位覆盖页面的目标矩形。

必须保证你所指定的位置要满足硬件设备所强加的边界限制。要得到更多的资料，请参阅“边界和大小限制”；同时，也要记住IDirectDraw2::SetOverlayPosition函数不会帮你进行裁减操作，函数中使用的坐标值如果有可能使你的覆盖页面超出目标页面的边缘，将会导致该函数调用失败，返回一个DDERR\_INVALIDPOSITION的错误。

8）创建覆盖页面

与其它的页面一样，通过调用IDirectDraw2::CreateSurface函数可以创建一个覆盖页面，条件是必须在DDSCAPS结构中指定DDSCAPS\_OVERLAY标志。

对覆盖页面的支持随显示设备的不同而存在着巨大的差异。因此，绝对不要认为存在一个被绝大多数硬件设备所支持的像素格式，必须作好准备与各种各样不同的像素格式打交道。通过调用IDirectDraw2::GetFourCCCodes函数，你可以得到设备驱动所支持的有关非RGB格式的信息。

当你创建一个覆盖页面的时候，先试着创建一个被大多数设备所认可的像素格式，如果失败，再试着使用其它的格式，这种方法是比较好的。

你可以创建一个覆盖页面换页链。要得到更多的资料，请参阅“创建复杂页面和换页链”。

9）覆盖页面的Z轴排序（Z-order）

覆盖页面在DirectDraw中被认为是位于屏幕上所有内容的最上一层，但是，当你需要显示多个覆盖页面的时候，你需要有一个办法来组织这些页面。为此，DirectDraw支持覆盖页面的Z轴排序，来管理这些覆盖页面层次关系和相互裁减。Z轴排序值代表了从主页面到观察者之间的一个抽象的距离。这些值的可能范围是从0，紧靠着主页面的一层，直到40亿，最靠近观察者的一层，并且没有任何两个覆盖页面可以共享同一个Z轴次序值。通过调用IDirectDrawSurface3::UpdateOverlayZOrder可以设置一个覆盖页面的Z轴次序。

目标关键色只受主页面的影响，而不会受到被另一个覆盖页面所覆盖的覆盖页面的影响。源关键色作用于覆盖页面，而不考虑该覆盖页面是否被设置了Z轴次序。

没有指定Z轴次序的覆盖页面，其Z轴次序被默认为0。没有指定Z轴次序的多个覆盖页面间的层次关系是不可预知的，当它们被映射到主页面上时，会产生混乱。

一个DriectDraw对象不能跟踪由另一个应用程序显示的覆盖页面的Z轴次序。

10）覆盖页面的换页

和其它各类型的页面一样，你可以创建覆盖页面的换页链。在创建好一个覆盖页面的换页链之后，你可以通过调用IDirectDrawSurface3::Flip函数实现换页。要得到更多的资料，请参阅“换页”。

利用覆盖页面的软件视频解码在调用Flip函数实现换页的时候，可以使用DDFLIP\_ODD和DDFLIP\_EVEN标志，以利用减少运动噪声的特性。如果设备驱动支持奇偶（odd-even）隔行换页，在调用GetCaps函数以获取设备驱动能力的DDCAPS结构中将会被设置DDCAPS2\_CANFLIPODDEVEN标志。如果是这样的话，那么，在调用IDirectDrawSurface3::UpdateOverlay函数时，你可以包含进DDOVER\_BOB标志，以告知设备驱动你希望它使用BOB（摆动）算法来使运动噪声达到最小。随后，在你调用Flip函数实现换页时，你可以带上DDFLIP\_ODD或DDFLIP\_EVEN标志，设备驱动将会自动的调整覆盖页面的源矩形来补偿不稳定的噪声。

获取设备驱动能力时，如果DDCAPS结构中没有设置DDCAPS2\_CANFLIPODDEVEN标志，那么，指定了DDOVER\_BOB 标志的UpdateOverlay函数调用将会失败。

要得到更多的关于Bob算法的资料，请参阅“普通视频噪声的解决方案”。

十一、Blit到多窗口

你可以使用一个DirectDraw对象和一个DirectDrawClipper对象Blit到由运行于普通控制级的应用程序创建的若干个窗口中。要得到更多的信息，请参阅“对多窗口使用裁剪器”。

创建多个DirectDraw对象，并且让它们Blit到相互的主页面的作法是不推荐的。

第六节调色板

一、什么是调色板？

基于调色板的页面需要调色板才能真正有意义的显示出来。一个基于调色板的页面，通常也被称作一个“色彩索引”页面，仅仅是一些数字的集合，其中的每一个数字代表一个像素。每一个数字的值都对应于一个色彩表中的项，这个表告诉DirectDraw对这个像素使用什么样的颜色。DirectDrawPalette对象，通常简称为“调色板”，给你提供了一个及其方便的途径来管理调色板。那些使用16位或更高位像素格式的页面并不使用调色板。

提供DirectDrawPalette对象是为了拥有直接操作16和256色调色板的特性（一个DirectDrawPalette对象通常与一个DirectDrawSurface对象相依属）。一个DirectDrawPalette对象保留了一个从0到255的256色调色板的入口；它不保留16色调色板的任何入口。它允许直接对色彩表（color table）的直接操作。一个色彩表是一系列颜色值（典型的是RGB三个一组）。这个表可以包含16或24位的RGB色彩入口，代表与每一个索引相对应的颜色。对16色调色板来说，色彩表可以包含另一个256色调色板的索引。

调色板被材质图、离屏页面、覆盖页面所支持，它们并不需要与主页面拥有同样的调色板。

你可以通过调用IDirectDraw2::CreatePalette函数来创建一个调色板。该函数将返回一个指向IdirectDrawPalette接口的调色板对象的指针。你可以使用该接口的函数来操作调色板入口、获得关于调色板对象能力的信息、或初始化该对象（如果你用的是CoCreateInstance函数创建的它）。

你可以通过调用IDirectDrawSurface3::SetPalette函数将一个调色板连接到一个页面上。一个调色板可以被连接到若干个页面。

通过调用IDirectDrawPalette::GetEntries，应用程序可以获得这些表的入口，并且可以通过IDirectDrawPalette::SetEntries改变这些入口。这个函数有一个dwFlags参数，指定对于调色板的改动是否立即生效。

DirectDrawPalette对象为一个8位的调色板保留了从0到255的入口，除非你指定了DDPCAPS\_ALLOW256标志请求所有这些入口都可以被你所用。

SDK中的Ddutil.cpp源文件包含了一些关于操作调色板的唾手可得的应用程序定义的函数（非库函数）。要得到更多的信息，请参阅源文件中的DDLoadPalette函数。

二、调色板的种类

DirectDraw支持1位（2个入口）、2位（4个入口）、4位（16个入口）、和8位（256个入口）的调色板。一个调色板只能依附于符合其像素格式的页面。例如，一个由DDPCAPS\_1BIT标志创建的2入口（2-entry）的调色板只能被依附于一个由DDPF\_PALETTEINDEXED1标志创建的1位（1-bit）页面。

除此之外，你可以创建一个不包含色彩表的调色板，这就是“索引调色板（indexed palettes）”。一个索引调色板是指：其入口并不包含RGB色彩值，而是另一个调色板的PALETTEENTRY结构的索引值。一个索引调色板的色彩表（color table）是一个2、4、16、256字节的序列，每一个字节是另一个调色板的索引。

要创建一个索引调色板，在调用IDirectDraw2::CreatePalette时指定DDPCAPS\_8BITENTRIES标志。例如，要创建一个4位的索引调色板，指定DDPCAPS\_4BIT |DDPCAPS\_8BITENTRIES。当你创建一个索引调色板，传递一个指向一系列字节的指针，而不是一个指向PALETTEENTRY结构的指针。当你使用IDirectDraw2::CreatePalette时，必须将一个指向字节组的指针转换成LPPALETTEENTRY类型。

注意：在Blit操作的过程中，DirectDraw不会解除索引调色板，而使其成为普通调色板。

三、对非主页面（Non-Primary Surfaces）设置调色板

调色板可以依属于任何一种基于调色板的页面（主页面，后台缓存，离屏平面，或材质图）。只有那些被依属到主页面（primary surface）的调色板才能对系统调色板（system palette）产生影响。注意到这一点是很重要的：DirectDraw位图映射决不会导致色彩转换；任何依属于源或目标页面的调色板在位图映射中被忽略。此外，IDirectDrawSurface3::GetDC函数同样忽略被选入页面的任何DirectDrawPalette对象。

非主页面调色板可以被应用程序或Direct3D（或其它3D渲染引擎）所使用。

四、共享调色板

调色板可以被多个页面所共享。同一个调色板可以设置给一个换页链的前台缓存（front buffer）和后台缓存（back buffer），或在多个材质页面中共享。当应用程序调用IDirectDrawSurface3::SetPalette使一个调色板依属于一个页面，页面将递增该调色板的引用记录（reference count）。当页面的引用记录达到0时，页面将递减依属于它的该调色板的引用记录。除此之外，如果通过IDirectDrawSurface3::SetPalette使调色板脱离页面，则该调色板的引用记录将减1。

注意：如果对同一页面反复调用IDirectDrawSurface3::SetPalette以脱离同一调色板，则该调色板的引用记录只会递减一次。后来的调用不会影响调色板的引用记录。

五、调色板动画

调色板动画指的是：用修改隶属于一个页面的调色板的方法来实现显示页面色彩的变化，而不是通过实际修改页面的内容来实现页面的变化。为了这个目的，调色板动画给你提供了一个不需要改变页面内容，改变页面所显示的图象的途径，而只会产生很少的消耗。

有两种方法可以直接实现调色板动画：

修改调色板的入口

在两个调色板之间切换

你可以使用以下两种方法之一来实现简单的调色板动画。前一个方法：改变为实现动画所需要改变颜色的调色板入口。这种方法，你可以通过调用IDirectDrawPalette::SetEntries来实现调色板的重置。第二个方法需要两个DirectDrawPalette对象。应用程序通过轮流将两个DirectDrawPalette与DirectDrawSurface对象相依属，来实现动画。这种方法，你可以通过调用IDirectDrawSurface3::SetPalette来实现。

没有一种技术会使硬件产生激烈反应，所以，你可以随心所欲的选择适合于你的程序的那种。

第七节裁减器

一、什么是“裁减器（Clipper）”对象

裁减器，或DirectDrawClipper对象，限定你的Blit操作到一个指定的页面区域。一个裁减器对象拥有一个或多个裁减清单。一个裁减清单是一个或一系列封闭的矩形，描述了页面上的一个或一系列区域，只有在该区域内才允许Blit。这些区域是用RECT结构来描述的，按屏幕坐标。

裁减清单是一个颇有价值的工具。它们最通常的作用是阻止你的应用程序在Blit操作时，超出屏幕的边界。举例来说，想象一个子图形从屏幕的一边逐渐进入屏幕。你不会想让你的子图形是突然出现在屏幕上的；你想让它从屏幕的边上一点一点的出现。如果没有裁减器对象，你应该需要在程序中加入一个逻辑判断和运算，逐步修改Blit操作的参数，以保护那些超出屏幕边缘的页面内存，如果没有这些逻辑操作的话，你的应用程序会产生内存访问异常（memory access violations）的错误。，

下面的插图展示了这种类型的裁减。

你可以使用裁减器对象来指定目标页面的特定区域为可写的。DirectDraw在这些区域中裁减Blit操作，以保护那些指定裁减矩形以外的区域不被改写。

下面的例图展示了这种类型的裁减。

二、裁减清单

DirectDraw用DirectDrawClipper对象来管理裁剪表。一个裁剪表是描述页面可见区域的一系列矩形。一个DirectDrawClipper对象可以被依附于任何一个页面。一个窗口句柄同样也可以与一个DirectDrawClipper对象相依附，在这种情况下，DirectDraw更新窗口的DirectDrawClipper对象的裁减表。

尽管在DirectDraw HAL看来，裁减表是可见的，但DirectDraw调用HAL只用于符合裁剪表所需要的矩形区域。比如，如果一个页面的右上方被裁减，并且应用程序指示DirectDraw将这个页面映射到主页面（primary surface）上，则DirectDraw将会使HAL进行两个映射，首先是页面的左上角，其次是页面的下部区域。

通过IDirectDrawClipper::SetClipList函数，你可以将整个裁减清单连接到裁减器上（如果设备支持该操作），而不是多次调用该函数，每次连接裁减清单中的一个矩形。除此之外，通过调用IDirectDrawClipper::SetHWnd函数，并且指定一个窗口的句柄，你还可以将该窗口连接到裁减器上。如果你已经让裁减器使用一个窗口句柄，那么，你将不能给该裁减器再设置另外的矩形。

覆盖页面的裁减只有在覆盖硬件可以支持，并且目标关键色没有被激活的情况下才能使用。

三、共享DirectDrawClipper对象

DirectDrawClipper对象可以被多个页面所共享。例如，同一个DirectDrawClipper对象可以被设置到一个换页链的前台缓存（front buffer）和后台缓存（back buffer）上。当应用程序使用IDirectDrawSurface3::SetClipper，将一个DirectDrawClipper对象依属到一个页面，则页面将递增该DirectDrawClipper对象的引用记录（reference count）。当页面的引用记录达到0时，页面将递减DirectDrawClipper对象的引用记录。除此之外，如果调用IDirectDrawSurface3::SetClipper，并使其裁减接口指针为NULL，则DirectDrawClipper对象将从页面上分离，其引用记录也将递减1。

注意：如果在同一页面多次调用IDirectDrawSurface3::SetClipper以依属同一个DirectDrawClipper对象，则它的引用记录只会递增一次。以后的调用不会影响到该对象的引用记录。

四、独立的DirectDrawClipper对象

你可以创建不直接属于任何特定DirectDraw对象的DirectDrawClipper对象。这些DirectDrawClipper对象可以被多个DirectDraw对象所共享。驱动无关（Driver-independent）的DirectDrawClipper对象是用DirectDraw的新函数DirectDrawCreateClipper创建的。应用程序可以在任何DirectDraw对象创建之前调用此函数。

因为DirectDraw对象不拥有这些DirectDrawClipper对象，所以它们不会在应用程序的DirectDraw对象被释放时自动被释放。如果应用程序没有明确的释放这些DirectDrawClipper对象，DirectDraw引擎将在应用程序结束时释放它们。

你仍然可以使用IDirectDraw2::CreateClipper函数创建DirectDrawClipper对象。这些DirectDrawClipper对象会在创建它们的DirectDraw对象被释放时，被自动释放。

五、用CoCreateInstance创建DirectDrawClipper对象

DirectDrawClipper对象拥有COM所支持的全部类功能。除了使用标准的DirectDrawCreateClipper 和 IDirectDraw2::CreateClipper创建一个DirectDrawClipper对象外，你同样可以使用CoGetClassObject来获得一个类对象，然后再调用CoCreateInstance，或者直接调用CoCreateInstance来创建它。下面的例子展示了如何使用CoCreateInstance 和 IDirectDrawClipper::Initialize来创建一个DirectDrawClipper对象。

ddrval = CoCreateInstance(&CLSID\_DirectDrawClipper,

NULL, CLSCTX\_ALL, &IID\_IDirectDrawClipper, &lpClipper);

if (!FAILED(ddrval))

ddrval = IDirectDrawClipper\_Initialize(lpClipper,

lpDD, 0UL);

       在CoCreateInstance调用中，第一个参数，CLSID\_DirectDrawClipper，是DirectDrawClipper对象类的类标志符，IID\_IDirectDrawClipper 参数指名了当前所支持的接口，lpClipper 参数指向返回的DirectDrawClipper对象。

在使用这种类机制创建DirectDrawClipper后，应用程序必须调用IDirectDrawClipper::Initialize来初始化该对象，否则该对象无法使用。dwFlags参数的值是0UL，在这种情况下，该值为0，因为现在还没有可以支持的标志符。在这个例子种，lpDD是拥有该DirectDrawClipper对象的DirectDraw对象。然而，你也可以提供NULL值，这将创建一个独立的DirectDrawClipper对象。（这相当于使用DirectDrawCreateClipper函数来创建一个DirectDrawClipper对象）

在关闭应用程序之前，用CoUninitialize函数来关闭COM。

CoUnitialize();

六、对系统鼠标使用裁减器

DirectDraw应用程序通常需要为用户提供一个鼠标指针。对于全屏独占模式的应用程序来说，如果使用了换页操作，那么要实现鼠标指针的唯一选择是使用子图，通过接受由DirectInput设备或由Windows鼠标消息传来的数据控制子图的移动。然而，任何应用程序，只要没有使用换页操作，仍然是可以使用Windows系统的缺省鼠标指针的。

当你使用系统的鼠标指针时，在某些情况下，当你在Blit到主页面的某些区域时，你将沦为图形杂点的牺牲品。这些图形杂点象鼠标指针那样出现在屏幕上，就好象是系统留下的。

通过阻止鼠标指针图象在Blit操作过程中“挡道”， 一个DirectDrawClipper对象可以防止这样的杂点出现。这也是一个相对而言较简单的解决办法。步骤是这样的：首先，调用IDirectDraw2::CreateClipper创建一个DirectDrawClipper对象；然后，通过调用IDirectDrawClipper::SetHWnd函数将你的应用程序的窗口句柄连接到该裁减器对象上。一旦一个裁减器对象被连接到一个窗口后，任何后续的对主页面进行的Blit操作（通过调用IDirectDrawSurface3::Blt函数）将不会出现任何鼠标杂点。

注意：IDirectDrawSurface3::BltFast函数，以及它的IDirectDrawSurface和IdirectDrawSurface2接口的版本，将不能对隶属于页面的裁减器进行操作。

七、对多窗口使用Clipper

你可以使用一个单独的DirectDrawClipper对象，对由单独的应用程序创建的多个窗口进行Blit操作。

这个方法是这样的。首先，创建一个单独的DirectDraw对象以及一个主页面。然后，创建一个DirectDrawClipper对象，并且将它连接到你的主页面上（通过调用IDirectDrawSurface3::SetClipper函数）。要使Blit操作仅限于窗口的客户区，在Blit到主页面之前，将该窗口的句柄设置给该裁减器（通过调用IDirectDrawClipper::SetHWnd函数）。只要你需要Blit到另一个窗口的客户区，再次调用IDirectDrawClipper::SetHWnd函数，将另一个窗口的句柄设置给裁减器。

创建多个DirectDraw对象，并且让它们Blit到相互的主页面是不推荐的做法。上面所介绍的技术提供了一个有效和可靠的Blit到多客户区的方法，针对的是单独的DirectDraw对象。

第八节多显示器系统

Windows 98和Windows NT 5.0可以支持在一个单独的系统中存在多个显示设备和显示器。多显示器架构（通常简称为“MultiMon”）使操作系统使用两个或多个显示设备和显示器来建立一个逻辑桌面成为可能。举例来说，在一个有两台显示器的系统中，用户可以在其中的任何一个显示器上显示他的应用程序，或者将窗口从一个窗口中拖到另一个窗口中。这一章的内容包含了如何在一个多显示器系统中使用DirectDraw的信息。

DirectX SDK的\Samples\Misc目录中包含了一些文件，提供了用于多显示器系统中的辅助性的函数。Multimon.h头文件使在Windows 98多显示器系统下编译的源代码能够在使用Windows 95的机器中顺利的编译和执行。除此之外，Ddmm.cpp文件Ddmm.h文件提供了辅助性的函数，允许你从一个窗口句柄或设备字符串中创建或获得DirectDraw对象。

一、在多显示器系统中列举显示设备

使用DirectDrawEnumerateEx函数可以列举出在多显示器系统中的各显示设备，指定一个标志来决定哪种类型的DirectDraw设备可以被列举出来。该函数每列举出一个设备，就调用一次应用程序定义的DDEnumCallbackEx型回调函数。

因为显示设备包括了显示卡和显示器，而且在通常情况下，一个显卡对应一个显示器，为了便于理解，以后我们谈到的显示设备，读者可以简单的将其直接视为显示器。

DirectDrawEnumerateEx函数被Windows 98和Windows NT 5.0（或更高版本）所支持，并且只能通过从动态连接库中获得函数地址来调用。在运行期，它是这样实现的：调用GetProcAddress Win32函数，从Ddraw.dll动态连接库中载入函数的地址。下面的例程演示了这个方法。

HINSTANCE h = LoadLibrary("ddraw.dll");

// 如果ddraw.dll不存在搜索的路径中，

// 那么，可能还没有安装DirectX，返回失败。

if (!h)

return FALSE;

// 注意：你必须知道要获得的函数的版本，

// 在这个例子中，我们使用ANSI版本。

LPDIRECTDRAWENUMERATEEX lpDDEnumEx;

lpDDEnumEx = (LPDIRECTDRAWENUMERATEEX) GetP    ocAddress(h,"DirectDrawEnumerateExA");

// 如果函数存在，调用它，列举出所有连接到桌面的显示设备，

// 以及所有非显示的DirectDraw设备。

if (lpDDEnumEx)

lpDDEnumEx(Callback, NULL,

DDENUM\_ATTACHEDSECONDARYDEVICES |

DDENUM\_NONDISPLAYDEVICES

);

else

{

/\*

\* 我们一定是运行在一个较老版本的DirectDraw中。

\* 因此，该操作系统必定不支持多显示器。

\* 于是，返回到DirectDrawEnumerate函数以列举出单显示器的标准显示设备。

\*/

DirectDrawEnumerate(OldCallback,NULL);

/\* 注意：这里有一个小的技巧，

\* 让OldCallback 回调函数包装进DDEnumCallbackEx回调函数。

\* 最后的回调函数是下面的这个样式：

\* BOOL FAR PASCAL OldCallback(GUID FAR \*lpGUID,

\* LPSTR pName,

\* LPSTR pDesc,

\* LPVOID pContext)

       \* {

              \* return Callback(lpGUID,pName,pDesc,pContext,NULL);

\* }

\*/

       }

       // 最后，必须调用FreeLibrary函数释放动态连接库。

FreeLibrary(h);

上面的例程代码可以在运行期或装载期为连接到Ddraw.dll的应用程序所用。

应该注意的是，你必须获得DirectDrawEnumerateEx 函数的ANSI或Unicode版本中的哪一个，依赖于你的应用程序的字符串类型。当声明相应的回调函数时，为字符串参数使用LPTSTR数据类型。如果你声明了\_UNICODE符号，LPTSTR数据类型在编译时使用Unicode字符串，否则使用ANSI字符串。通过使用LPTSTR数据类型，该函数能够正常工作，而不论你的应用程序使用的是什么样的字符串。

二、焦点窗口与设备窗口

在单独显示器系统中，当你为一个应用程序设置了全屏独占的控制级时（通过调用IDirectDraw2::SetCooperativeLevel函数），你必须指定你的应用程序窗口句柄。DirectDraw使用应用程序的窗口句柄来勾住系统的消息，用来获得应用程序的状态信息、传递键盘输入的消息、和调整窗口大小的消息等。DirectDraw不需要为普通控制级（窗口模式）的应用程序设置一个窗口句柄。

因为多显示器系统必不可少的会用到至少一个显示设备，因此，要用DirectDraw访问到每一个设备， 你必须为每一个显示设备创建一个DirectDraw对象。然而，你所创建的每一个DirectDraw对象需要知道哪一个应用程序的窗口是它可以控制的，以及哪一个是可以接收到击键消息的。换句话说，应用程序需要有一个方法来告诉DirectDraw哪一个窗口将被用于哪些操作。这个特性是通过焦点窗口（Focus window）和设备窗口（Device window）的概念提供的。IDirectDraw2::SetCooperativeLevel函数支持三个标志符使这个特性连接到一个DirectDraw对象：DDSCL\_SETFOCUSWINDOW、DDSCL\_SETDEVICEWINDOW、和DDSCL\_CREATEDEVICEWINDOW。

一个“设备窗口”只是一个简单的可见窗口，DirectDraw可以调整其大小以占据显示器的整个显示区域。你可以手工创建与安装与显示设备一样多的设备窗口，或者你可以让DirectDraw来为你处理这些细节（当你让DirectDraw来管理这些设备窗口的创建时，会存在一些协定，这些协定将在“缺省设备窗口”一节中讨论）。

一个焦点窗口指的是DirectDraw对象可以对其发送击键消息的窗口。一个DirectDraw应用程序只能拥有一个焦点窗口。

要得到更多的资料，请参阅“设置焦点和设备窗口”和“缺省设备窗口”。

三、设置焦点窗口和设备窗口

调用IDirectDraw2::SetCooperativeLevel函数，并且指定DDSCL\_SETDEVICEWINDOW或DDSCL\_SETFOCUSWINDOW标志，你可以设置一个DirectDraw对象的焦点窗口和设备窗口。（SetCooperativeLevel函数同样接受DDSCL\_CREATEDEVICEWINDOW标志，要得到更多的资料，请参阅“缺省设备窗口”。）

注意：只有当你想要获得对多显示器的全屏独占的访问时，才能设置焦点和设备窗口。如果你只想对主显示设备和显示器获得独占访问，你可以调用SetCooperativeLevel函数，在dwFlags参数中仅仅指定DDSCL\_FULLSCREEN和DDSCL\_EXCLUSIVE标志。

设置焦点和设备窗口是一个两步的过程：在为一个显示设备创建了DirectDraw对象之后，你必须首先为它设置焦点窗口。焦点窗口对你的应用程序中所有DirectDraw对象来说都是同一个，并且通过调用SetCooperativeLevel函数来设置。在调用中，第一个参数是获取击键消息的窗口的句柄，第二个参数是DDSCL\_SETFOCUSWINDOW标志。

在设置了DirectDraw对象的焦点窗口后，你应该设置设备窗口和控制级（在这种情况下，为全屏独占模式）。这一步是通过另一次调用SetCooperativeLevel函数完成的。在这一次的调用中，第一个参数是DirectDraw将调整其大小为全屏的窗口的句柄，第二个参数是DDSCL\_SETDEVICEWINDOW、DDSCL\_FULLSCREEN、和DDSCL\_EXCLUSIVE标志的结合。

尽管你必须指定一个焦点窗口，你仍然可以让DirectDraw来操纵你的设备窗口的创建和管理工作。要得到更多的资料，请参阅“缺省设备窗口”。

四、缺省设备窗口

你可以选择让DirectDraw来为你完成创建、管理、和销毁设备窗口的工作。简单的说，DirectDraw管理的窗口被称为是“缺省设备窗口”。尽管使用缺省设备窗口可以把你从创建、管理、销毁设备窗口的繁重任务中解放出来，但是你的应用程序将不能够获得操作系统发送到那些缺省设备窗口的鼠标消息。因此，只有在你的应用程序不需要从那些设备窗口接收鼠标消息的情况下，你才应该选择使用缺省设备窗口。

启用缺省设备窗口的过程是很简单的。在首先调用IDirectDraw2::SetCooperativeLevel函数设置了一个DirectDraw对象的焦点窗口之后，再次调用该函数，将dwFlags参数指定为DDSCL\_CREATEDEVICEWINDOW、DDSCL\_FULLSCREEN和DDSCL\_EXCLUSIVE标志符的结合。

注意：在一个单步调用SetCooperativeLevel函数的过程中，设置一个DirectDraw对象的焦点窗口、告诉DirectDraw创建一个缺省设备窗口、并且设置DirectDraw的控制级也是可能的。下面的例程演示了这是怎样完成的：

/\*

\* 该代码片段只在使用缺省设备窗口时才可用.

\*/

HRESULT hr;

hr = g\_lpDD->SetCooperativeLevel(

// 该窗口句柄仅仅是焦点窗口的

hwndFocus,

DDSCL\_SETFOCUSWINDOW | DDSCL\_FULLSCREEN |

DDSCL\_EXCLUSIVE | DDSCL\_CREATEDEVICEWINDOW);

六、多显示器系统中的显示设备与加速特性

一个DirectDraw应用程序应该列举出所有显示设备，并且从中选择一个显示设备（或让用户来选择），然后通过使用它的硬件全局唯一标志符（GUID）为该显示设备创建一个DirectDraw对象。这种技术可以使无论是在多显示器系统还是单显示器系统中，以及在任何一种控制级上都能获得最好的显示效果。

当前所激活的显示设备也被称为“缺省设备（Default device）”，或“空设备（Null device）。后一个名称的得来是因为当前所激活的显示设备是将NULL作为它的GUID而列举出来的。许多现有的应用程序为缺省设备创建一个DirectDraw对象，并且假定该设备是硬件加速的。然而，在多显示器系统上，缺省设备并不一定是硬件加速的，这取决于当时对DirectDraw所设置的控制级。

在全屏独占模式中，缺省设备是硬件加速的，但是已安装的其它显示设备则不是。这意味着，具有全屏独占模式的应用程序在多显示器系统中可以和任何其它系统一样运行得非常快速，但是不能利用系统内置的跨显示器的图形操作。需要使用到多显示设备（显示器）的全屏独占模式的应用程序，可以为每一个他想要使用的显示设备（显示器）创建一个DirectDraw对象。应注意的是：为一个特定的设备创建一个DirectDraw对象，你必须提供该设备的GUID（在调用DirectDrawEnumerate函数时可以被列举出来）。

当你把控制级设置为普通（窗口模式）时，缺省设备则不具有硬件加速性；此时的缺省设备成为了一个有效的、将两个物理显示设备的资源联合起来的逻辑仿真显示设备，因此，当设置了控制级为普通时，缺省设备是根本没有被硬件加速的。从另一个方面来说，当设置了控制级为普通时，缺省设备自动具有了跨越各显示器（显示设备）的图形操作的能力。典型的实例是，当第二显示器的逻辑地址位于主显示器的左边时，对主显示器进行带负坐标的Blit操作是有效的。

如果了你给应用程序设置控制级为普通模式，却仍需要利用硬件的加速特性，你必须只创建一个单独的DirectDraw对象，使用一个特定显示设备的GUID。应注意的是：如果你不使用缺省设备，你不会获得自动跨越设备的能力。这就是说，对主页面的Blit操作超出了显示器边界的部分将会被裁减掉（如果你使用了裁减器），或操作失败，返回DDERR\_INVALIDRECT的错误。

作为所有系统上的惯例，在创建了DirectDraw对象后，获得其能力或查询其它接口之前，你应该立即设置其控制级。除此之外，还应该避免在多显示器系统中多次设置控制级。如果你的应用程序需要从全屏模式切换到窗口模式运行，最好的做法是新建一个DirectDraw对象。

第九节高级DirectDraw主题

一、对Mode 13的支持

1、关于Mode 13

DirectDraw支持线性非换页的320x200的8位调色板显示模式，人们对它使用更为广泛的名称是Mode 13棗它的16进制BIOS模式编号。DirectDraw将这种模式视为一种Mode X模式，但是它们之间又有着重要的区别，这是由Mode 13模式的物理性质所强加的。

2、设置Mode 13

Mode 13模式与Mode X模式拥有一样的被列举和模式设置的行为特性。如果DDSCL\_ALLOWMODEX 标志被传给IDirectDraw2::SetCooperativeLevel 函数，DirectDraw将只会列举出Mode 13模式。

你可以象列举其它显示模式那样列举出Mode 13模式，但是你必须在调用IDirectDraw2::EnumDisplayModes函数之前做一次页面能力的检测。要完成这一步，调用IDirectDraw2::GetCaps函数，并且检查DDSCAPS 结构中的DDSCAPS\_STANDARDVGAMODE标志。如果该标志不存在，表示不支持Mode 13模式，那么，试图带DDEDM\_STANDARDVGAMODES标志的列举将会失败，返回一个DDERR\_INVALIDPARAMS错误。

       numDisplayModes函数现在支持一个新的列举标志棗DDEDM\_STANDARDVGAMODES，它会让DirectDraw除了列举出320x200x8的Mode X模式外，还可以列举出Mode 13模式。同样，IDirectDraw2::SetDisplayMode函数也有一个新的标志棗DDSDM\_STANDARDVGAMODE，为了将显示模式设置成Mode 13模式而不是320x200x8的Mode X模式，你必须传递此标志。

注意：有些显示卡提供了线性加速的320x200x8模式。在这种显示卡上，DirectDraw将不会列举出Mode 13模式，而列举出线性模式。在这种情况下，如果你试图通过将DDSDM\_STANDARDVGAMODE标志传递给SetDisplayMode函数而设置显示模式为Mode 13，函数会调用成功，但是使用了线性模式。这是低分辨率模式替换Mode X模式的一个相似的方式。

3、Mode 13与页面特性

当DirectDraw调用一个应用程序定义的EnumModesCallback回调函数，相应DDSURFACEDESC 结构的ddsCaps成员包含了反映正被列举模式的标志。在Mode X模式中，它会是DDSCAPS\_MODEX，在Mode 13模式中，它会是DDSCAPS\_STANDARDVGAMODE。这些标志是相互排斥的。如果没有任何一个标志被设置，那么，该模式是线性加速的。这些特征同样可应用于由调用IDirectDraw2::GetDisplayMode函数获得的标志。

4、使用Mode 13模式

因为Mode 13是线性模式，DirectDraw可以给应用程序提供对帧缓冲区的直接访问。与Mode X模式不一样，你可以调用IDirectDrawSurface3::Lock、IDirectDrawSurface3::Blt、和IDirectDrawSurface3::BltFast函数直接访问主页面。

当使用Mode 13模式时，DirectDraw支持一种模拟的IDirectDrawSurface3::Flip函数，这是通过将后台缓存的内容直接复制到主页面来实现的。你可以自己编写这些代码，使用Blt或BltFast函数复制一个后台缓存中的矩形区域到主页面上。

有一个有关锁定和Mode 13的误解。尽管DirectDraw允许对Mode 13 VGA帧缓冲区的直接线性访问，但绝不要认为该缓冲区的地址总是0xA0000，因为DirectDraw可以返回一个指向的地址并不是0xA0000的帧缓冲区的别名虚拟内存指针。同样，绝不要认为Mode 13页面的宽矩（Pitch）是320，因为支持320x200x8加速的显示卡很可能会使用不同的宽矩值。

二、从DMA中获益

本节的内容是关于如何从设备对DMA（直接内存访问）的支持中获得好处，以提高应用程序的运行效果。

1、关于DMA设备支持

某些显示卡可以完成在系统RAM页面中的Blit或其它操作，这些操作一般被称为“直接内存访问（DMA：Direct Memory Access）”。你可以利用DMA支持来加速某些集合操作。举例来说，你可以利用系统设备对DMA的支持，完成一次从系统RAM到显示RAM的Blit操作，而与此同时，处理器正在准备下一帧。为了使用这个功能，你必须承担一些责任，以下将对此进一步详细探讨。

2、对DMA支持的检测

在使用DMA操作之前，你必须检测系统设备是否支持DMA操作，如果支持的话，支持的程度如何。首先调用IDirectDraw2::GetCaps函数查询设备能力，然后检查相应DDCAPS 结构的dwCaps 成员中是否存在DDCAPS\_CANBLTSYSMEM标志。如果该标志存在的话，表示显示设备支持DMA。

如果你已知道显示设备支持DMA，你还需要知道显示设备支持的程度如何。这可以通过检查另一些结构成员来完成，它们提供了有关从系统RAM到视频RAM，视频RAM到系统RAM，以及系统RAM到系统RAM的Blit操作的DMA特性。这些特性由12个DDCAPS结构的成员提供，它们依照Blit操作的DMA方式来命名。下面的表展示了这些成员。

系统RAM到视频RAM 视频RAM到系统RAM 系统RAM到系统RAM

dwSVBCaps dwVSBCaps dwSSBCaps

dwSVBCKeyCaps dwVSBCKeyCaps dwSSBCKeyCaps

dwSVBFXCaps dwVSBFXCaps dwSSBFXCaps

dwSVBRops dwVSBRops dwSSBRops

举例来说，系统RAM到视频RAM的Blit特性标志由dwSVBCaps、dwSVBCKeyCaps、dwSVBFXCaps和dwSVBRops成员提供。类似的，视频RAM到系统RAM的Blit特性是由“dwVSB”打头的成员提供；系统RAM到系统RAM的Blit特性是由“dwSSB”打头的成员提供。检查存在于这些成员中的标志，可以确定硬件对Blit操作的DMA方式的支持程度。

就这些成员的Blit类型来说，这些成员中的标志与在dwCaps、dwCKeyCaps和dwFXCaps成员中Blit相关的标志是一致的。例如，dwSVBCaps成员中包含了常规的Blit操作特性，你可以在dwCaps成员中找到同样的标志。同样，dwSVBRops、dwVSBRops和dwSSBRops成员中也提供了有关特定类型的Blit的光栅操作（Raster operation）的光栅操作码。

从这些成员可以检测到的特性中，有一个关键的特性，用来表明系统对异步DMA Blit操作的支持。如果显示设备支持页面间的异步DMA Blit操作，那么DDCAPS\_BLTQUEUE标志会被设置在dwSVBCaps、dwVSBCaps或dwSSBCaps成员中（一般来说，你将看到的是从系统RAM到视频RAM页面的最好的支持）。如果该标志不存在，表明系统并不报告支持异步DMA Blit操作。

3、典型的DMA方案

从系统RAM到视频RAM的SRCCOPY（光栅操作码之一，表示“复制”）传送是被硬件支持的最基本的Blit操作类型。因此，该操作最典型的用法是将系统RAM页面中的数据传送到视频RAM的页面中，以备后用。系统RAM到视频RAM的DMA传送与CPU控制的传送（比如：HEL的Blit）一样快速，但是又具有极大的利用价值，因为它们可以与主CPU进行并行的操作。

4、利用DMA

硬件的Blit操作使用的是物理内存地址，而不是虚拟地址，后者是应用程序的本地地址。某些设备驱动要求你提供页面的物理内存地址。这种机制是由IDirectDrawSurface3::PageLock函数实现的。如果设备驱动不需要进行页锁定（Page locking），当你调用IDirectDraw2::GetCaps函数以测定硬件特性时，返回值中将会被设置DDCAPS2\_NOPAGELOCKREQUIRED标志。

锁定一个页面可以阻止系统将该页面的物理内存给其它的进程使用，并且保证该页面的物理内存地址始终保持不变，直到调用了相应的IDirectDrawSurface3::PageUnlock函数。如果设备驱动需要进行页锁定，那么，DirectDraw将只能允许系统RAM中页面的DMA操作，该页面是被应用程序锁定的。如果在这种情况下，你不调用IDirectDrawSurface3::PageLock函数，DirectDraw将会使用软件仿真的方式来完成Blit操作。应该注意的是：锁定大块的系统RAM将会导致Windows系统运行阻塞。因此强烈建议，如果你的应用程序不是运行于全屏独占模式，那么，不要调用IDirectDrawSurface3::PageLock函数以锁定大块的系统RAM，而且，当应用程序最小化时，必须及时将这些页面解锁，当然，当应用程序恢复最大化时，你可以再次锁定页面。

管理页锁定的任务完全落在应用程序开发者手中，DirectDraw绝不会来插手来锁定或解锁一个页面。而且，决定究竟将多少页面内存锁定才不会对系统运行产生严重的负面影响，也是对开发者的一项考验。

三、在窗口模式下使用DirectDraw调色板

当应用程序为独占（全屏）模式时，IDirectDrawPalette接口的函数拥有直接写硬件的能力。然而，当应用程序处于非独占（窗口）模式时，IDirectDrawPalette接口的调用的是GDI的调色板函数句柄来与其它应用程序协同工作。

对以下主题的讨论，我们假使桌面是8位调色板模式，并且你已经创建了一个主页面和一个标准的窗口。

1、窗口模式的调色板入口类型

与全屏模式的应用程序不同，窗口模式的应用程序必须与其它的应用程序共享桌面调色板。这就给开发者带来了一些困难，如怎样修改调色板入口，以及如何才能安全的修改等。用于DirectDrawPalette对象和GDI的PALETTEENTRY结构中，包含了一个peFlags成员，它携带了系统如何解释PALETTEENTRY结构的描述信息。

PALETTEENTRY结构是这样定义的：

typedef struct tagPALETTEENTRY { // pe

BYTE peRed;

BYTE peGreen;

       BYTE peBlue;

BYTE peFlags;

} PALETTEENTRY;

前三个成员我们都很熟悉，它们是该入口所代表的颜色的三原色分量的值，分别是红、绿、蓝。最后一个成员peFlags描述了该调色板入口是以下三种类型的哪一种：

Windows静态入口

动态入口

非动态入口

Windows静态入口：

在窗口模式中，Windows保留了调色板的前10个（0到9）和后10个（246到255）共20个入口，将它们作为系统保留色来显示窗口，如：菜单条、菜单文字、窗口边框、按钮底色等。为了使你的应用程序与系统保持一致的外观，以及避免破坏其它的应用程序，你有必要保护主页面调色板的这些入口。通常，开发者调用Win32的GetSystemPaletteEntries函数获得系统调色板的入口，然后，在将用户定制的调色板指派给主页面之前，将系统调色板中这20个保留入口的值明确的匹配给该定制调色板。在定制调色板中复制系统调色板的入口项在应用程序初始化时可以正常工作，但如果用户更改了桌面的配色方案，它会变得无法使用。

要避免你的调色板在用户更改了配色方案之后变得面目全非，你可以通过提供一个指向系统调色板的引用，而不是直接指定一个颜色值来保护这些被保留的入口。用这种方法，不管系统对某一个保留入口使用什么样的颜色，你的定制调色板将总会符合系统颜色，而且不需要做任何更新工作。在peFlags成员中使用PC\_EXPLICIT标志，使你将定制调色板的入口直接指向一个系统调色板成为可能。当你使用这个标志，系统将不会再认为其它的成员中包含颜色信息，你应该设置peRed成员的值为系统调色板的一个索引，并且将其它成员的值设为0。

举例来说，如果你想要确保你定制调色板的保留入口与系统调色板相一致，你应该使用如下的代码：

// 设置前10项和后10项入口与系统调色板相匹配

PALETTEENTRY pe[256];

ZeroMemory(pe, sizeof(pe));

for(int i=0;i<10;i++){

pe[i].peFlags = pe[i+246].peFlags = PC\_EXPLICIT;

pe[i].peRed = i;

pe[i+246].peRed = i+246;

}

你可以强迫Windows仅保留使用调色板入口的第一项和最后一项（0和255），方法是调用Win32的SetSystemPaletteUse函数。在这种情况下，你应该设置0号与255号PALETTEENTRY结构的peFlags成员值为PC\_EXPLICIT。

动态入口：

在相应的PALETTEENTRY结构的peFlags成员中设置PC\_RESERVED标志，表示调色板的该项入口即为用户所保留，Windows将不会允许任何其它的应用程序将它们的逻辑调色板映射到这个物理入口中。因此，只有你的应用程序能够修改这个入口的颜色值，实现调色板动画，而其它的应用程序却无法修改它。

非动态入口：

在相应的PALETTEENTRY结构的peFlags成员中设置PC\_NOCOLLAPSE标志，表示该调色板入口为普通、非动态的调色板入口。PC\_NOCOLLAPSE标志通知Windows不要用另一些已经分配了的物理调色板入口来替换该入口。

2、在窗口模式下创建调色板

下面的例程演示了如何在非独占模式（既窗口模式）下创建一个DirectDraw调色板。为了让你的调色板能够正常工作，一个关键的任务是你要设置256个入口（PALETTEENTRY结构）中的每一个入口，然后将其提交给IDirectDraw2::CreatePalette函数。

LPDIRECTDRAW lpDD; // 假定它已经被初始化

PALETTEENTRY pPaletteEntry[256];

int index;

HRESULT ddrval;

LPDIRECTDRAWPALETTE lpDDPal;

// 首先设置Windows的静态入口

for (index = 0; index < 10 ; index++)

{

// 前10个静态入口

pPaletteEntry[index].peFlags = PC\_EXPLICIT;

pPaletteEntry[index].peRed = index;

pPaletteEntry[index].peGreen = 0;

pPaletteEntry[index].peBlue = 0;

// 后10个静态入口

pPaletteEntry[index+246].peFlags = PC\_EXPLICIT;

pPaletteEntry[index+246].peRed = index+246;

pPaletteEntry[index+246].peGreen = 0;

pPaletteEntry[index+246].peBlue = 0;

}

// 现在，设置用户私有的入口，在这个例程中为前16个

// 这些入口是可以动态修改的

for (index = 10; index < 26; index ++)

{

pPaletteEntry[index].peFlags = PC\_NOCOLLAPSE|PC\_RESERVED;

pPaletteEntry[index].peRed = 255;

pPaletteEntry[index].peGreen = 64;

pPaletteEntry[index].peBlue = 32;

}

// 现在，设置其余的入口，非动态入口

for (; index < 246; index ++)

{

pPaletteEntry[index].peFlags = PC\_NOCOLLAPSE;

       pPaletteEntry[index].peRed = 25;

pPaletteEntry[index].peGreen = 6;

pPaletteEntry[index].peBlue = 63;

}

// 所有256个入口已经被填充，下面将其提交，以创建调色板。

ddrval = lpDD->CreatePalette(DDPCAPS\_8BIT, pPaletteEntry,

&lpDDPal,NULL);

3、在窗口模式下设置调色板入口

在IDirectDraw2::CreatePalette函数中应用于PALETTEENTRY结构的规则，同样适用于IDirectDrawPalette::SetEntries函数。典型的做法是，保存住你一开始创建的PALETTEENTRY结构组，以后你就不必每次都重建它。在必要的时候，你可以修改这个组中的个别项，然后调用IDirectDrawPalette::SetEntries函数以更新调色板。

在大多数情况下，如果你的应用程序运行于窗口模式，你就不应该去触动任何一个Windows静态入口，否则，后果是你所无法预知的。

对于调色板动画，通常你需要修改的只是PALETTEENTRY结构组的一个子集，然后将这些子集入口提交给IDirectDrawPalette::SetEntries函数，这些入口必须具有PC\_NOCOLLAPSE和PC\_RESERVED标志。试图对其它的入口进行调色板动画，后果同样是不可预知的。

下面的例程演示了如何在非独占模式（窗口）下进行调色板动画。

LPDIRECTDRAW lpDD;

PALETTEENTRY pPaletteEntry[256];

LPDIRECTDRAWPALETTE lpDDPal; // 以上三个对象为已初始化的

int index;

HRESULT ddrval;

PALETTEENTRY temp;

// 修改调色板入口，将靠前的16个入口进行循环

temp = pPaletteEntry[10];

for (index = 10; index < 25; index ++)

{

pPaletteEntry[index] = pPaletteEntry[index+1];

}

pPaletteEntry[25] = temp;

       // 更新调色板，不需要传递指向整个调色板的指针，

// 只用提供那些修改了的入口。

ddrval = lpDDPal->SetEntries(

0, // 该标志必须为0

10, // 要更新的第一个入口

16, // 要更新的入口个数

& (pPaletteEntry[10])); // 数据来源

四、获得换页和Blit操作的状态

当你调用IDirectDrawSurface3::Flip函数，你所期望的结果是将前台缓冲区与后台缓存进行交换。然而，这个函数调用并不一定都会成功，而且如果成功也并不是说就会立即进行换页的，举例来说，如果上一次的换页操作还没有完毕或没有成功，那么这次调用的换页函数将返回一个名为DDERR\_WASSTILLDRAWING的错误。对于这个问题，仅仅依靠该函数本身的一个较为简单的解决办法是：用While循环反复调用IDirectDrawSurface3::Flip函数，直到返回DD\_OK为止。还应注意的一点是，即使Flip函数调用成功，也不是说马上就能进行换页，换页被显示系统安排在显示器的下一次垂直回扫发生的时候进行，这就是说，换页一定是与显示器的刷新同步进行的。

然而，上面的这种方法其实是非常低效的，因为在循环的过程中，你将有可能多次调用Flip函数，而该函数不管是否成功，都会占用相对较长的CPU时间，于是就会产生这样一种情况：虽然上一次循环调用Flip函数没有成功，而在调用的过程中正在进行的换页完毕了，所以当再次循环调用Flip函数的时候，此时已经离上次换页结束有一段时间了，那么在这两次的换页中将会存在一个较长的时间差（这里所指的时间其实都相当短，可以说是在瞬间完成，其数量级为毫秒，远远超出人的感觉器官的灵敏度）。

最好的方法就是使用另一个用以检测换页操作行进状态的GetFlipStatus函数。该函数相对于Flip函数来说只占用相当短的时间，所以可以使两次换页的时间间隔达到最小。如果上一次的换页尚未结束，并且返回DDERR\_WASSTILLDRAWING，你的应用程序可以利用这段时间来进行另外的任务，然后再次调用IDirectDrawSurface3::GetFlipStatus函数检测换页是否完毕。一直到函数返回DD\_OK，就表示你被获准可以进行下一次的换页操作了。下面的例程片段演示了这种方法：

while(lpDDSBack->GetFlipStatus(DDGFS\_ISFLIPDONE) ==

       DDERR\_WASSTILLDRAWING)

// 等待上一次的换页操作完毕，

// 应用程序可以在这里进行另外的任务。

ddrval = lpDDSPrimary->Flip(NULL, 0);

       这种方法打个比方说就是在你攻城之前先派个小兵去打探敌情，直到探得条件已经成熟再大军压境，而不必每次都兴师动众却无功而返。

对于基于窗口模式的应用程序，你会取而代之使用Blit函数，这时，你可以使用另一个与GetFlipStatus函数相类似的IDirectDrawSurface3::GetBltStatus函数以检测正在进行的Blit操作的状态，这个函数与GetFlipStatus函数一样，都会占用相当短的CPU时间，并且立即返回，利用它们可以使你的应用程序达到最快的换页或Blit速度，而其间只有很少的时间损失。

五、使用Blit进行单色填充

Blt函数有很多用法，对你来说最熟悉不过的莫过于将图象从一个页面复制到另一个页面，然而，Blt函数也可用于单色填充却是鲜为人知的。不熟悉DirectDraw的程序员经常使用Win32的FillRect函数来实现这一功能，要知道FillRect是GDI的函数，其速度受到GDI模式的限制，其实是非常低效的。

在页面上，你可以将最常用的颜色作为该页面的底色，使用IDirectDrawSurface3::Blt函数可以为该页面“打底”，或者用该函数来实现清屏（使整个页面为黑色）。例如，如果你的应用程序画面的底色是兰色，首先填充一个DDBLTFX结构，并且设置其dwFillColor成员，该值必须与目标页面的像素格式一致。对于基于调色板的页面来说，该值应该是一调色板索引；对于一个16位RGB像素格式的页面来说，该值应该是一个16位的像素颜色值。然后，调用Blt函数，指定其dwFlags成员为DDBLT\_COLORFILL，表示使用该函数的单色矩形填充功能。“打底”之后，你就可以在该页面上绘制图象了。使用DirectDrawSurface的Blt函数是进行单色填充最快捷的一种方法，而调用常规的FillRect函数绘制实心矩形将耗时得多。

应该注意的是，如果页面是8位像素格式的页面，将一个颜色如红色，用RGB（255，0，0）的三原色方式赋给DDBLTFX结构的dwFillColor成员是错误的，你必须给它红色所对应的调色板索引值，可以用DDColorMatch函数获得。DDColorMatch函数是在随微软DirectX SDK附带的Ddutil.cpp文件中提供的，你必须将该文件连结到你的工程中去。如何连结在前面的章节中已有详细的介绍。

下面的例程演示了如何对页面进行清屏，即进行黑色填充：

DDBLTFX ddbltfx;

ddbltfx.dwSize = sizeof(ddbltfx);

ddbltfx.dwFillColor =0 ;

ddrval = lpDDSPrimary->Blt(

NULL, // 目标矩形

NULL, NULL, // 源页面和源矩形

DDBLT\_COLORFILL, &ddbltfx);

switch(ddrval)

{

case DDERR\_WASSTILLDRAWING:

.     case DDERR\_SURFACELOST:

case DD\_OK:

default:

}

六、测定显示硬件的能力

DirectDraw使用软件仿真的方式来实现那些用户的显示硬件不支持的DirectDraw函数。为了加速你的DirectDraw应用程序的图象显示，你应该在创建了DirectDraw对象之后，测定用户显示硬件的能力，然后指挥你的应用程序尽可能的利用这些能力。

通过调用IDirectDraw2::GetCaps函数可以测定这些能力。不是所有的需要硬件支持的特性都可以由软件仿真出来。如果你想要利用那些只有某些高级显示设备（如高级的3D图形加速卡，如VooDoo等）才能支持的特性，在你的用户没有装备这些硬件设备时，你必须给他们提供一个可以选择的方案，是退出还是用软件仿真模式来玩，当然，后者是一个不得以而为之的办法，有时候你的程序会变得跟蜗牛爬一样。

七、在视频RAM中储存位图

从视频RAM到视频RAM的Blit操作通常比从系统RAM到视频RAM更有效。因此，你应该在视频RAM中储存尽可能多的用于显示的图象。

目前，绝大多数显示适配器（显卡）都可以在容纳主页面及其后台缓存之后还有一些富余。调用IDirectDraw2::GetCaps函数以获得用户的显示硬件的能力，其中，DDCAPS结构的dwVidMemTotal和dwVidMemFree成员可以用来测定视频RAM中可用内存的数量。如果你想亲眼看一下它是怎样工作的，使用包含在DirectX SDK中的DirectX Viewer（DirectX 查看器）例子。运行后，打开DirectDraw Devices文件夹，再打开主显示驱动程序文件夹，于是就会显示出视频RAM的总量以及空余视频RAM的数量。每当向DirectDraw对象中增加一个页面，空余视频RAM数量将减少，其减少量为该新增的页面所占用的视频RAM。

八、Triple Buffering（三缓冲）

在某些情况下，那就是，当显示适配器（显卡）拥有足够的视频RAM，那么，你就可以能够使用Triple Buffering（三缓冲区）来加速你的应用程序的显示速度。三缓冲区使用一个主页面和两个后台缓存。下面的例子演示了如何初始化一个三缓冲区方案：

// lpDDSPrimary，lpDDSMiddle和lpDDSBack是全局变量，

// 定义成LPDIRECTDRAWSURFACE型。

DDSURFACEDESC ddsd;

ZeroMemory (&ddsd, sizeof(ddsd));

// 创建一个主页面，带有两个后台缓存。

ddsd.dwSize = sizeof(ddsd);

ddsd.dwFlags = DDSD\_CAPS | DDSD\_BACKBUFFERCOUNT;

ddsd.ddsCaps.dwCaps = DDSCAPS\_PRIMARYSURFACE |

DDSCAPS\_FLIP | DDSCAPS\_COMPLEX;

ddsd.dwBackBufferCount = 2;

ddrval = lpDD->CreateSurface(&ddsd, &lpDDSPrimary, NULL);

// 如果我们成功的创建了换页链，

// 取出指向这些页面的指针，用来进行换页和Blit。

if(ddrval == DD\_OK)

{

// 得到直接连结到主页面的页面（后台缓存）。

ddsd.ddsCaps.dwCaps = DDSCAPS\_BACKBUFFER;

ddrval = lpDDSPrimary->GetAttachedSurface(&ddsd.ddsCaps,

&lpDDSMiddle);

if(ddrval != DD\_OK) ;

              // 在此处显示出错信息。

}

因为在调用Flip函数进行换页的时候，系统会自动将主页面在三缓冲区换页链中的三个页面中循环，所以你并不需要关心三缓冲区换页链中的每一个页面。你只需要保持住指向主页面和紧接主页面的一个后台缓存的指针就可以了。指向主页面的指针用来调用Flip函数进行换页，指向后台缓存页面的指针用来准备将要显示的下一个画面。要得到更多的信息，请参阅“换页”。

建立三缓冲区换页链的好处在于：你可以不用象双缓冲区换页链那样，必须等到换页操作结束后，才能向后台缓存绘制图象（因为换页时页面被锁定，不能被其它读写工作）；相反，你可以在调用完Flip函数之后立即对后台缓存进行新的绘制工作，从而使换页成为一种非同步的事件。如果你只使用一个后台缓存，那么在等待Flip函数返回DD\_OK前，你的应用程序将让这些空闲的时间白白浪费掉。

九、DirectDraw应用程序和窗口风格

如果你的DirectDraw应用程序是运行于窗口模式的，你可以用任何窗口风格来创建你的窗口。然而，全屏独占模式的DirectDraw应用程序的窗口不能设置有WS\_EX\_TOOLWINDOW风格，否则可能会遭遇到不可预知的行为。WS\_EX\_TOOLWINDOW风格原是阻止一个窗口成为顶层窗口，而在DirectDraw的全屏独占模式应用程序中，这却是必须的。

为了正确的显示，全屏独占模式的应用程序最好带有WS\_EX\_TOPMOST扩展窗口风格和WS\_VISIBLE基本窗口风格。这些风格使得DirectDraw应用程序成为所有窗口的上层窗口，并且阻止GDI写向主页面。

下面的例子演示了在全屏独占模式的应用程序中如何正确的初始化一个窗口：

////////////////////////////////////////////////////////

// 注册窗口类，显示窗口，

// 并且初始化所有DirectX和图形对象。

////////////////////////////////////////////////////////

BOOL WINAPI InitApp(INT nWinMode)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

              wcex.hInstance = g\_hinst;

wcex.lpszClassName = g\_szWinName;

wcex.lpfnWndProc = WndProc;

wcex.style = CS\_VREDRAW|CS\_HREDRAW|CS\_DBLCLKS;

wcex.hIcon = LoadIcon (NULL, IDI\_APPLICATION);

wcex.hIconSm = LoadIcon (NULL, IDI\_WINLOGO);

wcex.hCursor = LoadCursor (NULL, IDC\_ARROW);

wcex.lpszMenuName = MAKEINTRESOURCE(IDR\_APPMENU);

wcex.cbClsExtra = 0 ;

wcex.cbWndExtra = 0 ;

wcex.hbrBackground = GetStockObject (NULL\_BRUSH);

RegisterClassEx(&wcex);

g\_hwndMain = CreateWindowEx(

WS\_EX\_TOPMOST,

g\_szWinName,

g\_szWinCaption,

              WS\_VISIBLE|WS\_POPUP,

0,0,CX\_SCREEN,CY\_SCREEN,

NULL,

NULL,

              g\_hinst,

NULL);

if(!g\_hwndMain)

return(FALSE);

SetFocus(g\_hwndMain);

ShowWindow(g\_hwndMain, nWinMode);

UpdateWindow(g\_hwndMain);

return TRUE;

}

十、将真彩色匹配到帧缓冲区的色彩空间

当显示设备不是处于24位真彩模式时，应用程序需要知道一个真彩RGB颜色是怎样映射到一个帧缓冲区的色彩空间中去的。举例来说，如果你正在编写一个可运行于8、16或24位显示模式的应用程序，在设置关键色或用Blit进行单色填充的时候，你所提供的颜色值必须与该页面当前的色彩空间相一致，也就是与帧缓冲区的色彩空间一致。这就是说，如果你的应用程序运行于8位色彩模式，那么你必须提供一个调色板索引；如果运行于16位色彩模式，你必须提供一个相应的16位值；如果运行于24位真彩模式，你必须提供一个24位值。用户可以选择他所期望的显示模式，那么你的程序就必须可以根据当前的显示模式，将一个特定的颜色值（通常是真彩24位值）映射到正确的色彩空间中去。

尽管DirectDraw不会为你自动完成颜色匹配的任务，但是仍有许多方法来解决如何把你的关键色映射到帧缓冲区中去。这些方法是比较复杂的。在绝大多数情况下，你可以使用GDI内建的色彩匹配服务，加上DirectDraw的直接帧缓冲区访问，来解决将一个24位颜色值匹配到不同的色彩空间中去。实际上，DirectX SDK中的Ddutil.cpp源文件包含了一个DDColorMatch函数，用它就可以完成任何色彩空间的匹配工作。它的工作原理十分巧妙，你并不需要了解每一种色彩空间的格式，它用的是“埋种子”的方法，其步骤如下：

调用Win32的GetPixel函数，取页面左上角（坐标为0，0）的像素值。

调用Win32的SetPixel函数，用COLORREF结构来描述你的24位RGB颜色，将该色设置给页面左上角。

使用DirectDraw的Lock函数锁定页面，取得指向帧缓冲区内存的指针。

从帧缓冲区中获得真实的颜色值，调用Unlock函数将页面解锁。

还原页面左上角像素颜色为原始值，使用SetPixel函数。

DDColorMatch函数的速度并不是很快，这是因为它调用了Win32的GetPixel和SetPixel函数。然而，它提供了一个可靠和值得信赖的方法来求解颜色在不同的RGB色彩空间中是如何映射的。