Windows 2000 应用系列

Windows 2000 编程

李多多 等 编著

人 民 邮 电 出 版 社

需要全本请访问 www.ertongbook.com

第 14 章　DirectX 程序设计

DirectX 主要被用来在 Windows 下开发多媒体程序。在 Windows 2000 下，DirectX 是开发游戏的主要工具。

本章主要介绍 DirectX 程序设计的基本知识，并在最后附有利用 DirectX 设计的应用例程。

14.1 DirectX 基础

DirectX 的主要功能是在 Windows 下进行游戏程序设计。大多数电脑玩家都记得几年以前玩 DOS 游戏的经历。确实，在 Windows95 诞生之前，大部分游戏都是在 MS-DOS 下编写的。此时，程序员需要知道硬件的具体情况，对于不同的硬件需要进行很多重复开发工作。DirectX 是专为 Windows 设计的游戏开发系统，使用 DirectX 可以实现游戏开发平台从 DOS 向 Windows 的转移。

14.1.1　DirectX 的特点

DirectX 的一大特点是其相对于硬件的独立性。程序员不需要具体考虑硬件的细节部分。与此同时，为了实现游戏的实时性，DirectX 必须提供对硬件的直接访问。也就是说，DirectX 实现了本该在 Windows 下面实现的东西——直接访问硬件、独占资源，事实上也正是因为这样，DirectX 程序才可能具有不同于典型的 Windows 应用程序的高质量的游戏表现。

为了达到以上目的，DirectX 具有灵活的底层接口，它提供了所有游戏所必须的共同的接口，这使得游戏设计者能够更加充分地利用系统的性能。

DirectX 的结构包括两个驱动程序：一个是硬件抽象层（HAL），另一个是硬件模拟层（HEL）。这两者组合起来共同响应 DirectX 的请求。在创建 DirectX 对象时，它将查询相应的硬件并将硬件所提供的功能和参数记录下来，这样，在进行一些特殊的操作，如进行图像的拉伸操作时，如果图形处理器提供支持，DirectX 会直接调用硬件所提供的功能，否则，与其等价的某 HEL 命令将会被调用。

实际上，硬件层被 HAL 层包起来了，而 HEL 层则是提供给程序员的一组必要的、互相独立的接口。这样的结构使得 DirectX 具有硬件自动更新特性：当更先进的硬件生产出来以后，只要能获得最新版本的 DirectX，那么以前编写的软件就能够自动利用最新硬件的先

进特性，从而使程序获得更佳的图形效果和运行速度。这样，DirectX 就可以利用专为提高效率而设计的各种加速卡，尤其是各种即插即用的硬件和软件。

14.1.2　COM 技术基础

DirectX 运用了 COM（对象组件模型）技术为基础。COM 技术是当前许多 Windows 产品的基石，其自身相当复杂。在此仅仅介绍与开发 DirectX 应用程序有关的部分。

1．COM 技术的发展背景

OO（面向对象）技术经过变形和发展，衍生到软件体系结构设计和代码设计之中，出现了部件化软件和 Design Pattern 等新的技术概念。过去的十年里，计算机硬件和软件技术水平的持续提高，软件工业界给 PC 机用户桌面和互连网络提供了大量功能强大而结构复杂的应用程序。正是由于如此复杂的程序功能和结构，给软件开发者、用户和软件供应商带来了难以想象的困难。

复杂软件的设计和实现非常昂贵，并且容易出现错误。许多的努力和消耗都集中于相同的设计概念和代码部件的重复开发。而硬件结构的多样化、操作系统的纷繁复杂和/ 杂的通信平台使得从零开始来设计正确、可移植、高效而低代价的应用变得非常困难。

硬件的飞速发展使得旧的软件设计方法不能适应。程序员需要一种新的软件开发模式，它必须具有分布式的计算、客户机/服务器结构、模块化和部件化的特点。

一种方法就是使用和开发可重复利用的软件部件。所谓软件部件，是指在软件系统设计中能够重复使用的模块。部件把一系列互相关联的操作和服务包装起来，形成一个整体。

具体的包装手段则是由部件标准（Component standards）来规定的。部件对外部世界的表现完全不受本身内部实现的约束。这种对外部接口和交互协议的强调，使得部件标准和传统通信协议相区别。很好的部件标准应该使得部件的交互、替代、升级、定制以及组合成更大粒度的部件或应用成为可能。因此，部件标准是软件开发工作从可复用部件技术得到应有效果的必要保证。

部件软件的发展非常迅速，现在已经形成了统一的标准。而且由于涉及分布式服务部件的使用，部件概念已经发展出分布式部件模型（DOM, Distributed Component Model）。它继承了部件软件的所有功能优点，并且引入了强大的网络特性。分布式部件模型不仅会改变应用程序本身的结构，同时也会改变网络的运行和管理机制。这一点应该引起充分的重视。

由于 DOM 技术的采用，许多的智能可以分散于不同地点的部件之中，使得部件越来越自动化。网管员能够在各地加载强大的网管工具，使得各个路由节点能够自动地动态调整带宽，以适应本地的网络条件的变化。不同地点的 object agent 能够对网管员的查询作出更为细致的反应，这就能够帮助问题的定位和解决。我们也能够发现，switched net 和 virtual LANs 的概念也和 DOM 很相容。

在此基础上，操作系统可能并不限于连接的建立，而应该引入更加高层的能力。这种功能现在只能依靠手动调整和设置。下面将具体介绍 DOM 的标准。

2．COM 的特点和优点

COM 作为面向对象的编程模型，其目的是提高软件的交互协同工作能力，而不应该受具体环境的影响。从某种意义上说，COM 就像传统的系统服务 API，提供某些操作以使服务的客户能够和多个服务器建立连接。但是一旦连接建立，COM 就立刻退出这个通信框架，而完全由客户和服务器直接进行交互，从而避免了因为强制通过中央处理 API 产生的许多不必要的附加代价损失。传统的操作系统主要以 API SDK 的形式向应用程序提供服务。而 COM 力图将对象化的概念延伸到系统中，零散的 API 函数被组织成逻辑上比较紧凑的形式。所有的客户、服务器和系统间都通过一组函数（在 COM 中称为方法或接口）实现交互。所谓接口（interface）就是软件部件间严格格式化了的协议，它由较少量但又有用的逻辑相关的操作组成。接口严格地说明了操作实现的功能和期望的响应，而操作间的相关性使得设计者能够顺利地以整体的方式实现所需功能。

COM 使用接口具有以下优点：

(1) 客户机/服务器都能独立的完成升级。

(2) 对于同一地址空间的客户服务器，能够实现简单高效的对象交互。

(3) 对象设计完全实现地点的透明性。

(4) 不依赖任何设计语言。

3．COM 接口的二进制标准

COM 协议规定了接口的二进制标准，它包含有：

- 通过 QueryInterface 实现基本接口交互；
- 通过 refefence counting 机制管理对象及其资源，包括和多个客户连接；
- 内存在独立开发的部件间传递交换时的分配释放原则；
- 一致性和错误诊断工具。

值得注意的是，在作为二进制标准的同时，COM 也包含部分的系统代码。

4．对象和接口

所谓接口，就是一组逻辑相关的操作的集合，在各种资料中，接口也可以是接口规范定义的代码实现。实际上，这是两个不同的概念，应该加以区分。接口的实现，是指对象使用接口的用代码实现的所有方法（成员函数），并将接口的 COM 二进制兼容的指针传递给 COM Library。而接口的类型定义只是一组相关的函数原型，它们仅仅定义了形式而不包含实现。由上可知，接口不是对象，而是不可更改一种协议。

COM 对象则是若干个接口的集合。具有多个接口的对象，能够同时提供多个函数表指针。这样的对象应该能提供一些基本操作，使得客户能够自由地得到它所包含的任意一个接口指针，并且控制对象本身的生存周期。

任何 COM 对象都必须包含并实现 IUnknown 接口，而任何定制的接口也必须从 IUnknown 继承而来。这可以通过以下三种方法完成：

- QueryInterface；
- AddRef；

- Release。

QueryInterface 主要完成 COM 对象所含多个接口的遍历浏览。而 AddRef 和 Release 方法则完成 COM 对象生存期的控制。

一个对象一般只需 IUnknown 对象的一个实现。对于支持聚集的对象来说，情况稍微有些不同。这时需要为 IUnknown 接口和其他接口分别实现不同的方法函数。

14.1.3　DirectX 的主要组成部分

DirectX 主要包含 7 个部分，本小节将简单讲述这 7 个部分的内容（其中 DirectDraw 和 DirectSound 是编程时主要用到的部分，在下文里还将详细介绍）。

1．DirectDraw

DirectDraw 是 DirectX 中的视频输入输出的基本部分，其目的在于提供一个基于 Windows/Windows NT 游戏图形系统，它很有可能在未来的版本中与 Direct 3D 合并，组成一个强大的支持 3D/2D 操作的系统。不过微软似乎也有可能在 DirectDraw/Direct3D 的基础上再建立一个层面，如 DirectX 7.0 中新提供的 Direct3DX，来完成对 3D 的支持。

在 Windows 的 32 位环境中，DirectDraw 可以完成对显存的访问，并据此编制出高效的视频处理程序，完成复杂的功能。

下面简单地列出 DirectDraw 提供的功能：

- 内存直接读写功能；
- 内存块移动、复制功能；
- 访问屏外显示内存中的位图；
- 利用硬件的位块传输和缓冲区翻转等软硬件加速技术实现快速直接存取；
- 直接操作显存；
- 支持显卡的位块操作；
- 硬件覆盖；
- 翻转内存影射的平面。

此外，DirectDraw 还支持许多种不同的显示硬件，它提供的 API 接口不但可以使程序应用各种硬件特性，还能模拟硬件尚不支持的特性。

2．DirectSound

DirectSound 提供低延时的声音混合系统，硬件加速并支持对声音设备的直接访问。同时，它还允许进行属性设置，以便应用程序开发员充分利用声卡和相关的外设驱动所提供的扩展服务。DirectSound 给程序员提供了一个独立于硬件的编程接口，使程序员能够非常有效地应用硬件所提供的特性而不必了解硬件的复杂的细节。

DirectSound 充分利用了声卡的各种优越性能以尽可能地提高其运行速度，从而减少 CPU 的占用时间。

与 DirectDraw 一样，DirectSound 也是与 Windows API 相兼容的。一般地，对于复杂的、需要有特殊音效的应用程序，建议使用 DirectSound。

需要全本请访问 www.ertongbook.com

3．DirectPlay

DirectPlay 的目的在于简化游戏通信设备的入口，并构成游戏外设的标准接口。DirectPlay 所提供的通信功能自成一体，其所实现的功能独立于用户之间实现通信的介质、协议和其他在线控制功能。

微软建立了一套称为 Direct Protocol 的互联协议。在 DirectX 6.0 以后的版本中，DirectPlay 支持各种信令——包括 TCP/IP、IPX 等，只要取得建立的服务器的认可。其一系列新的特性甚至包括了各种的通信的同步/异步控制、信令控制，并支持建立在 TCP/IP 协议基础上的防火墙。

4．Direct3D

Direct3D 的功能是开发基于 Windows 的多语言的游戏接口和实现互动的图形显示。它提供了如下两种工作模式：

- 保留模式（RetainedMode）。
- 立即模式（Immediate-Mode）。

保留模式是建立在即时模式基础上的，它使程序易于建成一个完整的三维图形系统；立即模式使用户能完全控制着色管道,它提供了一个与设备无关的底层硬件加速系统。

Direct3D 的目的在于给程序员们提供一个底层的高性能 3D 接口，使得用户程序可以与硬件相结合，从而获得独立于设备的灵活性。在 DirectX 7.0 中，微软提供了一个新的接口－Direct3DX 接口，用来简化对 Direct3D 设备的操作。

5．DirectInput

DirectInput 的主要功能是提供基于 Windows 游戏输入的 API 和驱动程序，它支持键盘、鼠标和游戏操纵杆等设备，并能兼容将来的输入外设。

DirectInput 也是基于 COM 模型的，它提供了一种利用设备驱动程序直接访问输入设备的方法，可以替代传统的 Windows 消息机制。DirectInput 所提供的扩展服务和优越的性能是游戏开发的强大的工具，并为实现模拟和实时交互系统提供了强大的支持。

6．DirectSetup

DirectSetup 的主要功能是提供基于 DirectX 应用程序的一次性安装过程，并在此过程中对使用 DiretPlayLobby 对象的应用程序提供自动的 Windows 注册表注册。

DirectSetup 现在提供了用户自定义接口的能力，用户可以通过声明一个函数原型 DirectSetupCallbackFunction 来达到对用户接口的扩展定义。

7．DirectMusic

DirectMusic 的主要功能是支持 MIDI 音频。

14.2 DirectDraw

DirectDraw 是微软专为游戏开发商提供的一个基于 Windows 的软件子系统，它支持许

多种不同的显示硬件，从你可以想象的最简单的 SVGA 到支持剪切、伸展和各种颜色格式的高级显卡。DirectDraw 提供的 API 接口不但可以使你的程序应用各种硬件特性，还可以模拟硬件尚不支持的特性。使用 DirectDraw 只需程序员遵守一些标准硬件约定，如: RGB 或 YUV 颜色格式及解析度。DirectDraw 对显存的移动操作也非常简单，无须调用特殊的过程。

DirectDraw 是 DirectX 中视频输入输出的基本部分。可以说，在整个 DirectX 技术，中，DirectDraw 都处于核心地位，它是 DirectX 的基石。DirectX 的其他组件，如 Direct3D 和 DirectShow，都是建立在它的基础上的。

如上所述，DirectDraw 提供的功能非常多，其中最主要的是对内存的直接读写功能和内存的位块移动、复制功能。同时，DirectDraw 还支持访问屏外显示内存中的位图，你可以通过利用硬件的位块传输和缓冲区翻转等软硬件加速技术实现快速直接存取。总而言之，DirectDraw 能让你直接操作显存。

DirectDraw 最重要也是最基本的功能就是通过绘图表面（Drawing Surface）提供对内存的直接访问。绘图表面一般对应于显示内存中的一个内存块，它接受视频数据并将其作为图像在屏幕上显示出来。为了获得更好的图形效果，并且减轻 CPU 的负担，在计算机系统中引入了图形协处理器。图形协处理器同 CPU 并行工作，主要用于处理图形显示。在这里，我们通过对显示内存的直接操作来加快图形显示速度。

对于图形显示来说，最频繁的操作就是把内存中一个位块的内容移动到内存的另一个位置，再在屏幕上显示出来，所以表面（Drawing Surface）最常用的功能就是对位块移动操作(Blitting)的支持。如果显示硬件支持，DirectDraw 还能完成收缩、拉伸、格式转换等一系列功能。当然，我们也可以通过 HEL 层来模拟实现这些功能，但是这会大大降低图形的显示效果。

为了加快图形的显示速度，现在的显示卡上都带有 1MB 以至 100MB 的显示内存。DirectDraw 在一般情况下通过对显示内存的直接操作，大大提高了显示的速度，而在显示内存不够时，DirectDraw 又能够利用系统内存，从而获得良好的性能。为了优化对内存的使用，DirectDraw 还提供在指定的内存区域中创建 DirectDraw 对象的功能，从而使得程序员对内存的操作更加得心应手。

DirectDraw 通过硬件操作层（Hardware Abstraction Layer [HAL]）来实现硬件独立性。HAL 是一个由设备制造商提供的设备特定的接口，这个接口提供了对显示硬件的直接操作能力，并且能够对硬件的性能给予最大的优化。另外，DirectDraw 还可以通过硬件模拟层（Hardware Emulation Layer [HEL]）来模拟实现硬件现在还不支持的新特性。

DirectDraw 是基于 Windows 操作系统的，所以 DirectDraw 采用操作系统提供的 32-Bit 线性内存模式，把系统内存和显示内存看成大的内存块，而不再是以前的那种分段分页的内存模式。DirectDraw 不再采用过时的段地址：偏移地址结构来寻址，而是采用线性寻址，因此使寻址更为快捷方便。

对于全屏应用程序，DirectDraw 还提供了依靠后台缓冲（Back Buffering）实现的页翻转（Page Flipping）功能。由于具有了页翻转功能，从而使游戏和动画的画面变得更加流畅，使画面真正地动起来。对于窗口程序，DirectDraw 则提供了剪切功能，从而获得类似的效

果。

14.2.1 DirectDraw 结构

在出现 DirectDraw 之前，Windows 开发人员都是使用图形设备接口（GDI）来开发应用程序。GDI 通过显示驱动接口（DDI）的驱动程序来操作具体的图形硬件，它提供了和多设备无关性的图形接口，这些接口包含了从字体处理到位图操作的几乎所有功能。在这种机制下，所有的硬件功能都被隔离了，GDI 的这种结构布局能够在绝大多数的情况下很好地工作，但是对于一些需要访问硬件的底层接口的情况，GDI 提供的设备无关性将导致这些底层的操作不被支持。不但如此，实际上对于一些诸如覆盖图等游戏开发人员感兴趣的高层功能也不被 GDI 所支持。

当应用程序调用 DirectDraw 时，它将根据各组件的能力确定是要调用与硬件直接相关的 HAL 功能还是调用 HEL 功能来进行模拟，DirectDraw 的 HEL 功能基本上是保持不变的，而 HAL 功能与具体的硬件相关。这样就能尽可能地发挥硬件性能。

图形界面问世以来，图形编程技术有了长足的进步，图形设备接口（GDI）使得程序员完全脱离了复杂多样的硬件，获得了硬件独立的好处。对于很多软件，使用图形设备接口（GDI）已经足够了。但是随着多媒体技术的飞速发展，多媒体软件需要高品质的图形显示，要求能够在高速显示的时候仍然能够提供很高的画面质量，图形设备接口（GDI）对此就显得无能为力了，此时就需要 DirectDraw 来完成相应的程序设计了。

DirectDraw 使得基于 Windows 的应用程序仍然能够获得高速高效的图形操作。它能够尽可能地使用硬件所提供的优良性能，并且能够软件模拟硬件功能。

DirectDraw 通过硬件操作层（Hardware Abstraction Layer [HAL]））来实现硬件独立性。系统将优先使用硬件操作层(HAL)，因为使用硬件模拟层来模拟硬件特性会对显示速度造成严重影响。

14.2.2 DirectDraw 的对象类型

DirectDraw 应用程序是通过 DirectDraw 对象所提供的接口来完成功能的调用的。DirectDraw 应用程序是由几个一起工作的对象组成的，通常有一个父对象和由该父对象派生的几个子对象。一个进程中可以创建多个 DirectDraw 对象，所有的这些对象互相独立。一般，与每一个对象相关的接口和其他子对象在 DirectDraw 对象之间是不能共享的，但是裁剪器在需要的时候可以在不同对象间进行共享。

下面将介绍 DirectXDraw 中的相应对象。

1．DirectDraw

在所有的对象中，“DirectDraw”对象有着非常重要的地位，它是所有 DirectDraw 应用的核心。使用 DirectDraw 时要做的第一件事就是创建 DirectDraw 对象，然后才能使用 DirectDraw 对象来创建其他的子对象，每个子对象都提供用于访问某种 DirectDraw 特性的方法。要想使用 DirectDraw，最先要做的就是创建 DirectDraw 对象。DirectDraw 对象提供了创建和操作其他对象的方法，比如说操作图面和调色板。

在一个进程中，可以创建多个 DirectDraw 对象，一般而言，对象是相互独立的，因此接口和其他子对象在 DirectDraw 对象之间是不能共享的。这里有一个例外，那就是裁剪器，在必要的时候裁剪器是可以共享的。

可以用 DirectDrawCreate（） 函数来创建 DirectDraw 对象。函数原型如下：

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

下面将说明函数的参数。

lpGUID 所要使用的 DirectDraw 唯一设备标识的地址，其值意义如下：

- NULL—则使用主设备；
- DDCREATE_EMULATIONONLY—系统仅使用 HEL；
- DDCREATE_HARDWAREONLY—仅使用默认设备的 HAL。

后两种情况只用于测试和调试时。

LplpDD 用于存放指向 IdirectDraw 接口指针的地址。

PUnkOuter 给未来的 COM 集合特性保留，其值为 NULL。

若该函数调用成功，它将创建一个带有 IDirectDraw 接口的 DirectDraw 对象。由于 COM 对象允许一个接口的几个版本同时存在，因此还可以使用 IDirectDraw2 接口。下面的代码创建了一个 DirectDraw 对象并且设法获得了 IDirectDraw2 接口。

```
// 创建 DirectDraw 对象
If(FAILED(DirectDrawCreate(lpDevice,&lpDD,NULL)))
{
    MessageBox(NULL,“creating DirectDraw Object fail”,NULL,NULL);//创建失败信息
    return FALSE;
}
// 申请 IDirectDraw2 接口
if(FAILED(lpDD->QueryInterface(IID_IDirectDraw2,(LPVOID*)&lpDD2)))
{
    MessageBox(NULL,“ query the Interface fail”,NULL,NULL);//申请失败信息
    return FALSE;
}
// 释放不需要再使用的 lpDD 接口
lpDD->Release( )；
```

2．DirectDrawSurface

DirectDrawSurface 对象对应于显示卡上的一块内存区域，它保存着要在屏幕上作为图形显示的数据，本对象使得对显示内存进行直接访问成为可能。

在创建好 DirectDraw 对象后，我们就可以用 IDirectDraw7::CreateSurface 方法来创建一个表面。通过表面对象所提供的 IDirectDrawSurface、IDirectDrawSurface2、IDirectDrawSurface3、IdirectDrawSurface4 和 IDirectDrawSurface7 等接口，我们就可以对这一块内存进行我们所希望的操作，如在这一块内存中调入位图然后显示在屏幕上。

3．DirectDrawPalette

DirectDrawPalette 对象表示了一个表面所要使用的 16 色或 256 色索引的调色板，它通过 IDirectDrawPalette 接口来提供对调色板的管理。调色板能够节约时间和空间。使用调色板时，不需要在显示缓存中直接存储每个像素的颜色，而是存入了调色板条目指针来提供颜色的索引，在要调用颜色时，通过查找颜色索引来调用存储在调色板中的对应颜色，从而提高内存的利用率。

在不采用调色板时，显示缓存中的数值直接代表了每个像素的颜色。使用的颜色数目越多，存储每个像素的颜色状态所需的内存空间也就越大。在采用 256 色显示设置时，用 8bit 就能表示每个像素可能的颜色。而使用对于 16,000,000 色的真色彩来显示时，每个像素需要用 24 位才能表示。显然，随着使用的颜色数目增多时，所需要的存储空间就越大，而且由于数据位的增加，进行数据操作时的时间也延长。所使用的颜色越多，画面的显示质量就越好，但存储图像的显示内存就要加大，画面刷新所需要的时间也就越长，有什么方法能够解决这个问题呢？

调色板就是为解决这个问题而引入的。调色板化的显示格式是一种能够节省时间和显示内存的机制。采用调色板时，在显示内存中并不直接存储每个像素的颜色，而是存入调色板条目指针来提供要使用的颜色的索引，实际的颜色保存在调色板中。在应用程序要调用某一种颜色时，通过查找该颜色的索引来调用存储在调色板中的对应颜色，从而提高内存的利用率和显示速率。

可以用 IDirectDraw7::CreatePalette 来创建一个调色板。其函数原形为：

```
HRESULT CreatePalette(
DWORD  dwFlags,
LPPALETTEENTRY  lpDDColorArray,
LPDIRECTDRAWPALETTE FAR *lplpDDPalette,
IUnknown FAR *pUnkOuter                    );
```

其参数说明如下：

DWORD dwFlags 控制标志，用来控制要创建的调色板的类型。

LPPALETTEENTRY lpDDColorArray 保存用来初始化调色板对象的颜色入口数组的地址。

LPDIRECTDRAWPALETTE FAR *lplpDDPalette 一个指针变量，该变量在函数调用成功时，保存指向。

IdirectDrawPalette 接口的指针的地址。

IUnknown FAR *pUnkOuter 本参数是为将来的兼容性而保留的，现在必须设置为 NULL。

在创建调色板时，还可以通过控制参数来选择要使用的调色板的类型。可选控制标志如下：

DDPCAPS_1BIT 表明调色板采用 1 位索引，在颜色表中有 2 个入口。

DDPCAPS_2BIT 表明调色板采用 2 位索引，在颜色表中有 4 个入口。

DDPCAPS_4BIT 表明调色板采用 4 位索引，在颜色表中有 16 个入口。

DDPCAPS_8BIT 表明调色板采用 8 位索引，在颜色表中有 256 个入口。

DDPCAPS_8BITENTRIES 表明索引指向一个 8 位的颜色索引，只有在使用了 DDPCAPS_1BIT、DDPCAPS_2BIT 或者 DDPCAPS_4BIT 标志后该标志才有效。每一个颜色入口为 1 字节长度，为一个使用 256 色的目标表面提供颜色索引。

DDPCAPS_ALPHA 表明与 PALETTEENTRY 结构相关的 peFlags 成员用一个简单的 8 位 alpha 值来解释。

DDPCAPS_ALLOW256 表示调色板可以定义所有的 256 个颜色入口。

DDPCAPS_INITIALIZE 用 lpDDColorArray 传递过来的颜色数组中的颜色来初始化调色板。

DDPCAPS_PRIMARYSURFACE 表示所创建的调色板将直接与主表面相联系。

DDPCAPS_PRIMARYSURFACELEFT 表示所创建的调色板将直接与左视主表面相联系。

DDPCAPS_VSYNC 表示调色板的改变将与监视器的刷新率同步。

4．DirectDrawClipper

DirectDrawClipper 对象提供了裁减功能。裁剪能够使程序的输出限制在一个被用户定义的区域内，从而不必担心在位块移动时移动到不希望的区域或者超出边界。从某种意义上说，裁剪就像是照相机的取景框，它使我们能够选取我们感兴趣的区域。通过调用 DirectDraw 对像的 CreateClipper 方法，能够创建一个 DirectDrawClipper 对象，并且生成的 Direct DrawClipper 对象自动和 DirectDraw 对象相联系。

一个运行 DirectDraw 的窗口最有可能遇到的情况是被其他的窗口遮住了一部分，可能是一个角，也可能是大半个窗口，一般需要避免把当前程序中的一部分画到别人的文本编辑器里去，这就需要用到裁剪。

所谓裁剪就是把被遮住的元素从用户的图像中移开的过程。当窗口被其他的应用程序窗口遮住以后，用户就不能随心所欲地在整个窗口区域里绘图了，被遮住的区域必须经过裁剪。裁剪的操作是通过裁剪器的使用来完成的。裁剪器在内部的裁剪列表中保留了一列有效的输出区域，并通过这个列表来控制程序的输出。DirectDraw 提供了 DirectDraw Clipper 类来实现裁剪的功能。

裁剪器需要按以下三步来操作：首先创建裁剪器对象，把它与目标窗口相关联，把它与目标图面相关联。

CreateClipper（）函数将创建一个 DirectDrawClipper 对象。该函数的原型如下：

```
HRESULT CreateClipper(
  DWORD  dwFlags,
```


需要全本请访问 www.ertongbook.com

```
    LPDIRECTDRAWCLIPPER FAR *lplpDDClipper,
    IUnknown FAR *pUnkOuter
);
```

下面将介绍其参数。

DwFlags **未使用，其值必须为零。**

LplpDDClipper **一个用于存放所得到的裁剪器指针的地址。**

PUnkOuter **未使用，其值必须为** NULL**。**

在获得了 DirectDrawClipper **对象后，接下来需要把一个** DirectDrawClipper **对象与一个窗口进行关联，当一个** DirectDrawClipper **对象与一个窗口进行关联后，裁剪列表就会自动地被操作系统刷新以反映桌面上窗口排列的变化。**

SetHWnd（）**函数可以完成** DirectDrawClipper **对象与一个窗口进行关联的工作。函数原型如下：**

```
HRESULT SetHWnd(
    DWORD dwFlags,
    HWND hWnd
);
```

下面介绍其参数。

DwFlags **未使用，其值必须为零。**

HWnd **所要关联的窗口句柄。**

关联完成以后，裁剪列表就会自动地被操作系统刷新以反映桌面上窗口排列的变化，但是为了让 Blt **和** UpdateOverlay **等方法的输出自动根据裁剪列表作出正确的调整，还必须将裁剪器与图面相关联。**

SetClipper（）**函数可以完成这一操作，其函数原型如下：**

```
HRESULT SetClipper(
    LPDIRECTDRAWCLIPPER lpDDClipper
);
```

下面将介绍其参数。

LpDDClipper **为该图面的** Blt **和** UpdateOverlay **方法所要使用的裁剪器。**

5．DirectDrawVideoPort

DirectDrawVideoPort **对象用来表示某些系统中的视频端口的硬件，这些硬件允许不通过** CPU **或者** PCI **总线来对结构缓冲区进行加速。通过在调用** DirectDraw **对象的** Query Interface **方法时声明** IID_IDDVideoPortContainer **标志就能够获取** IDDVideoPortContainer **接口，然后从该接口可以创建** DirectDrawVideoPort **对象。**

14.2.3 硬件操作层和硬件模拟层

DirectDraw **通过硬件操作层（**Hardware Abstraction Layer [HAL]**） 来实现硬件独立性。**HAL **是一个由设备制造商提供的设备特定的接口，**HAL **所能够提供的功能与具体的硬件实**

现有关。这个接口提供了对显示硬件的直接操作能力，并且能够对硬件的性能给予最大的优化。

当应用程序调用 DirectDraw 时，应用程序会根据各组件的能力自动确定是要调用与硬件相关的 HAL 功能还是调用 HEL 功能。如果没有指定要调用 HAL 功能还是 HEL 功能，只要硬件支持，DirectDraw 通常会调用 HAL 功能，因为 HAL 功能能够最大程度地利用硬件所提供的优良特性。

通过 HAL 提供的基础结构，DirectDraw 提供一组相容的接口和方法来让应用程序进行图形显示，应用程序并不需要与 HAL 打交道。HAL 可以是显示卡驱动程序的一部分，也可以是一个单独的动态链接库(DLL)，通过驱动程序设计者定义的一个私有接口来和驱动程序进行通信。HAL 完全依靠设备来实现功能，而不进行模拟。如果硬件不支持某种功能，HAL 没有进行报告的能力。此外，HAL 不验证传递的参数，DirectDraw 在 HAL 调用之前完成这些工作。

为了能够更好地利用硬件特性，可以使用相应的硬件模拟层（Hardware Emulation Layer [HEL]）功能代替。硬件模拟层有时会调用 GDI 的一些功能，而有时则直接访问硬件以完成一些简单的并与具体硬件无关的任务，如内存访问等。

在利用硬件特性时，如果指定的 HAL 功能没有找到，DirectDraw 还可以通过硬件模拟层来模拟实现硬件现在还不支持的新特性。

当通过硬件操作层(HAL)而硬件无法支持某些特性时，DirectDraw 就试图对其进行模拟。显然，硬件模拟在性能上与硬件特性所支持的性能无法相提并论。我们可以通过调用 IdirectDraw7::GetCaps 方法来查询硬件所支持的特性。通过初始化时的查询，我们可以通过适当调整参数来对不同性能层次的硬件进行优化。

不过有时硬件操作与硬件模拟的组合反而比用纯软件模拟更慢。例如：显示驱动程序支持 DirectDraw 但不支持拉伸位块移动（Stretch Blitting）的情况下，在用显存进行拉伸位块移动来移动位图时将明显损失性能。因为通常显存都要比系统内存慢，所以在访问显示内存时会迫使 CPU 处于等待状态，从而延长了操作时间。

14.2.4　协作级别

协作级别主要可用于控制应用程序和系统以及与其他应用程序之间的交互程度。

可以使用 SetCooperativeLevel（）来设置 DirectDraw 的协作级别。函数原形为：

```
HRESULT SetCooperativeLevel(
  HWND  hWnd,
  DWORD  dwFlags
);
```

函数的参数说明如下：

hWnd 应用程序窗口的句柄。如果使用 DDSCL_NORMAL 协作级别，则它可以为 NULL。

DwFlags DirectDraw 的协作级别。下面将介绍该参数各个值的含义。

DDSCL_ALLOWMODEX

允许在 X 模式下显示，该标志必须和 DDSCL_EXCLUSIVE 与 DDSCL_FULLSCREEN 标志组合使用。

DDSCL_ALLOWREBOOT

DDSCL_FULLSCREEN 与 DDSCL _EXCLUSIVE 标志被设定时，允许用户用 Ctrl+Alt+Delete 重新启动计算机。

DDSCL_FULLSCREEN

当与 DDSCL_EXCLUSIVE 一起使用时确定独占级别。

DDSCL_EXCLUSIVE

必须与 DDSCL_FULLSCREEN 一起使用。

DDSCL_NORMAL

应用程序将以一般的 Windows 程序运行。

DDSCL_NOWINDOWCHANGES

DirectDraw 不能自动最小化或恢复应用程序窗口。

14.2.5 显示模式

DirectDraw 应用程序可以分为两类：全屏模式的和窗体化模式的。编写全屏模式的 DirectDraw 应用程序时所遇到的问题是显示模式的设置，而编写窗体化模式的应用程序所需要处理的是 DirectDraw 裁剪器。下面我们将分别对这两类 DirectDraw 应用程序进行讨论。

先看设置了 DDSCL_FULLSCREEN 与 DDSCL _EXCLUSIVE 标志的全屏模式的 DirectDraw 应用程序，这类应用程序可以进行任意显示模式的操作，而且也必须进行相应的操作，与之相比，仅设置了 DDSCL_NORMAL 标志的应用程序虽然可以改变分辨率，但是颜色深度是不允许改变的。

在 DirectDraw 中，显示模式包括如下几个数值：

- 显示的高度；
- 宽度；
- 颜色深度；
- 刷新频率。

上面这几个数值是相互牵制的：高的分辨率和颜色深度将会导致低的刷新频率（也就是动画帧速率）。

DirectDraw 提供的模式选项包括 Windows 提供的模式和 DirectDraw 提供的其他模式。如果想知道系统支持何种显示模式，可以调用 EnumDisplayModes （）函数来查看。

要设置显示模式，就需要调用 SetDisplayMode（）函数。SetDisplayMode（）函数的原型为：

```
HRESULT SetDisplayMode(
    DWORD  dwWidth,
    DWORD  dwHeight,
    DWORD  dwBPP,
    DWORD  dwRefreshRate,
```

```
    DWORD  dwFlags
);
```

下面介绍其参数：

dwWidth 显示的宽度，以像素为单位。

DwHeight 显示的高度，以像素为单位。

DwBPP 颜色的深度。

dwRefreshRate 参数为刷新频率。将该参数设为 0 可以使用默认的刷新率。

DwFlags 此参数被设置为 DDSDM_STANDARDVGAMODE 时，应用程序可以使用 320*240 的 Mode 13 显示模式。

DirectDraw 应用程序可以进行模式自动切换。当 DirectDraw 应用程序关闭后，DirectDraw 对象被释放，此时显示模式会自动恢复到原来的设置值。另外如果在应用程序运行时切换到了其他的应用程序，模式也将被自动切换，而之后如果用户又从其他的应用程序切换回来，模式又将自动切换为应用程序所设定的值。当然如果调用 RestoreDisplayMode()函数的话，也可以手动恢复原先的显示模式，但还是建议让 DirectDraw 自动完成这些工作。

当设置了 DDSCL_NORMAL 标志以后，虽然可以改变窗体化模式应用程序的分辨率，但是颜色深度是不允许改变的。而且对于窗体化模式的的 DirectDraw 应用程序来说，也是不推荐运行时改变分辨率的。

14.2.6　图面和位转换函数

如上所述，DirectDraw 在很大程度上是一个内存管理器，它允许开发人员创建和管理图像数据缓冲区，并以一定的形式显示缓冲区中的数据。而实际上 DirectDraw 中所说的图面即是指这些缓冲区。图面是 DirectDraw 的主要成分，而图面对象也是 DirectDraw 中最重要的对象之一。任何有实际意义的 DirectDraw 操作都不可能脱离图面。

把图面理解成一块以矩形方式来管理的内存缓冲区可能会更清楚。可以根据图面的作用和特性把它分为以下几类：

1．主图面

主图面是最主要的图面。一个庞大的 DirectDraw 应用程序中可能有几十甚至上百个图面，而主图面只能有一个，这是用来显示给用户看的图面。

2．后台图面

除去主图面，应用程序中的其他图面大多属于后台图面。在本章最后的游戏例程中，大家可以看到，后台图面被用来在后台进行各种用于最终显示的图面元素的操作，如拉伸、粘贴等等。

3．组合图面

组合图面实际上就是由 DirectDraw 创建并维护的图面的集合。

4．简单图面

简单图面是由 DirectDraw 创建的组合图面结构的组成部分。

5．纹理图面

纹理图面实际上是后台图面的一种，它被专门用于存储 Direct3D 中的纹理信息。

6．覆盖图面

覆盖图面也是后台图面的一种，通过显示硬件，覆盖图面可以自动地与主图面进行组合。

7．切换图面

切换图面是一种复杂的图面，它可用于生成平滑的动画。

这么多种类的图面，实际上在一个 DirectDraw 应用程序中使用得最多的还是前两种：主图面和后台图面。

可以用 DirectDraw 对象的 CreateSurface（）函数来创建一个图面。函数原型如下：

```
HRESULT CreateSurface(
   LPDDSURFACEDESC  lpDDSurfaceDesc,
   LPDIRECTDRAWSURFACE  FAR  *lplpDDSurface,
   IUnknown  FAR  *pUnkOuter
);
```

下面将介绍其参数：

LpDDSurfaceDesc 为一个指向 DDSURFACEDESC 结构的指针。

LplpDDSurface 为一个指向创建得到的图面的指针的地址。

PUnkOuter 未使用，其值必须为 NULL。

CreateSurface（）的第一个参数为一个指向 DDSURFACEDESC 结构的指针，DDSURFACEDESC 结构是 DirectDraw 定义的用于存储图面信息的一种结构，该结构的定义如下：

```
typedef struct _DDSURFACEDESC {
    DWORD           dwSize;          // 结构的大小
    DWORD           dwFlags;         // 控制标志
    DWORD           dwHeight;        // 图面的高度
    DWORD           dwWidth;         // 图面的宽度
    union
    {
        LONG        lPitch;                  // 图面的跨度
        DWORD       dwLinearSize;
    };
    DWORD           dwBackBufferCount;  // 后台缓冲区数
    union
```

```
    {
        DWORD           dwMipMapCount;      // 未被使用
        DWORD           dwZBufferBitDepth;  // 未被使用
        DWORD           dwRefreshRate;      // 刷新频率
    };
    DWORD               dwAlphaBitDepth;    // 未被使用
    DWORD               dwReserved;         // 未被使用
    LPVOID              lpSurface;          // 图面内存的指针
    DDCOLORKEY          ddckCKDestOverlay; // 覆盖图目标颜色值
    DDCOLORKEY          ddckCKDestBlt;      // 位转换的目标颜色值
    DDCOLORKEY          ddckCKSrcOverlay; // 覆盖图源颜色值
    DDCOLORKEY          ddckCKSrcBlt;       // 位转换的源颜色值
    DDPIXELFORMAT       ddpfPixelFormat;    // 像素格式结构
    DDSCAPS             ddsCaps;            // 图面能力标志
} DDSURFACEDESC;
```

可以使用 DirectDrawSurface 对象的 GetSurfaceDesc（）函数来得到一个 DDSURFACEDESC 的结构，该结构中填充着图面的信息。在调用 CreateSurface（）创建图面的过程中，要正确地填充它的第一个参数，一个 DDSURFACEDESC 结构并不是一件很简单的事，很多人会在这里遇到麻烦。如果参数填写错误，将会有很多无法预料的事情发生。

有了主图面和后台图面后，DirectDraw 应用程序的框架就可以搭建起来了。在一个典型的 DirectDraw 应用程序中，关于图面的主要操作是位转换。位转换操作就是一个拷贝操作——把内存中的数据从一个地方移到另外一个地方。而图面实际上就是一块被假想成矩形的用来存放图像数据的内存区域，基于图面的位转换操作，也就是将一个图面中的全部或是部分数据复制到另外一个图面中。DirectDraw 提供了如下两种位转换方法：

- Blt（）方法；
- BltFast（）方法。

Blt（）的函数原型如下：

```
HRESULT Blt(
    LPRECT  lpDestRect,
    LPDIRECTDRAWSURFACE3  lpDDSrcSurface,
    LPRECT  lpSrcRect,
    DWORD  dwFlags,
    LPDDBLTFX  lpDDBltFx
);
```

下面将介绍其参数：

LpDestRect 一个 RECT 结构的指针。该 RECT 结构用于定义目标矩形，如果该参数为 NULL，则整个目标图面都将被使用。

需要全本请访问 www.ertongbook.com