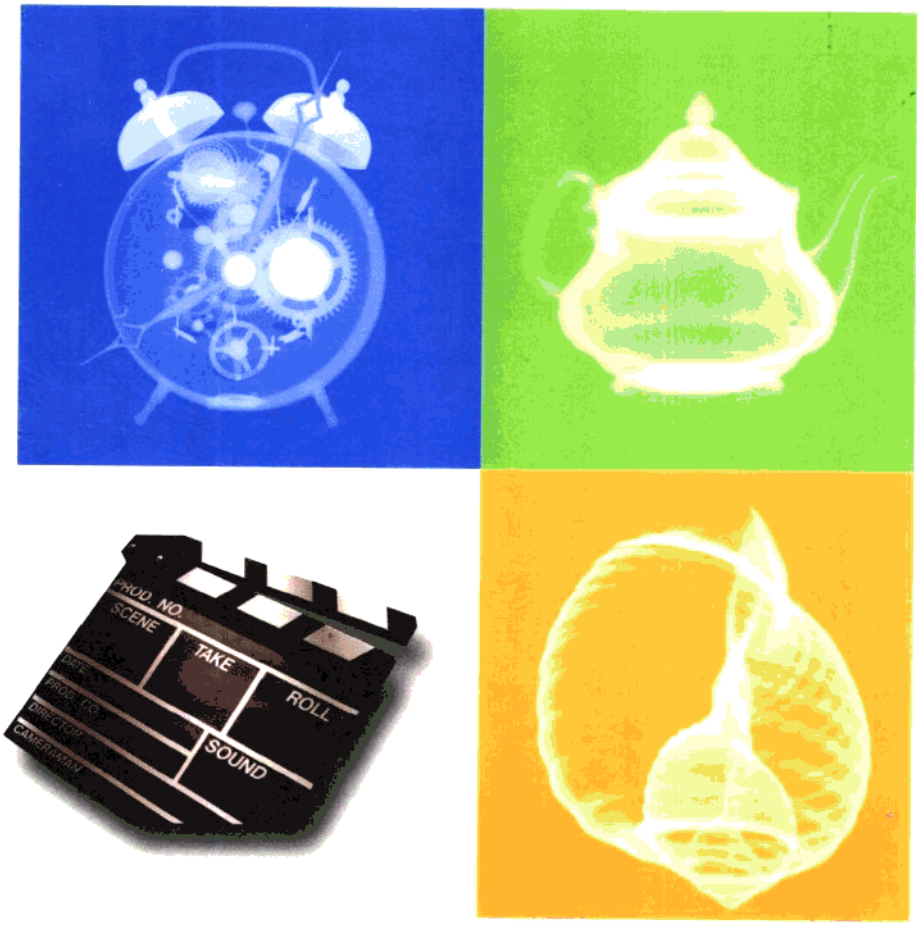
内附光盘



包含 DirectDraw 、 Direct3D Retained Mode
使用 DirectX7.0 SDK 、 VC++6.0 、 MFC 编译

李建汉 编著

DirectX 实用技巧

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

出版说明

DirectX 是微软公司为了 Windows 游戏所设计的一套开发工具程序软件（SDK），这套 SDK 包含了 2D、3D、音效、网络、音乐、输入设备等函数，充分与 Windows 操作系统结合，避免了程序设计师直接与硬件设备、I/O、低级核心程序设计打交道的麻烦，也加快了游戏的执行效率。

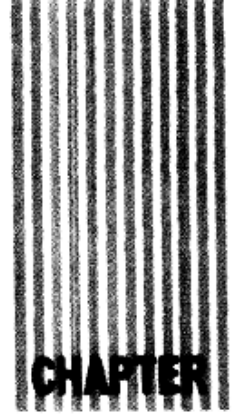
由于本书所附光盘为繁体版，若出现乱码，请用“东方快车”等汉化软件进行转换，敬请读者谅解。

本书由松岗电脑图书资料股份有限公司提供版权，中国铁道出版社计算机图书项目中心审选，费广正、王清、邓雄、李丽等同志完成了本书的整稿工作。廖康良、陈贤淑、肖志军、孟丽花等同志完成了本书的排版工作。

中国铁道出版社

2000 年 11 月

目 录



1

DirectX 与 COM

第一节 DirectX 简介

看这本书的人都应该已经知道 DirectX 的功能，但我还是概略地提一下好了。

一般来说 DirectX 有两种意义，第一是代表 DirectX runtime，也就是一些 dll，这是提供玩 game 的人，也就是所谓的 end-user 用的。另一个意义是 DirectX SDK（software developerkit），其中包含了 runtime，公用程序，lib，VC++中必须的 include 文件，和许多帮助文件，以及示范程序，一般开发者就是以这套 SDK 来开发的。

DirectX 只是一个格式（COM 格式），由微软制定，当我们调用 DirectX 函数或是 method 的时候，其实是调用一些 DirectX DLL，然后这些 DLL 再调用显卡厂商所写的驱动程序，真正驱动硬件的是厂商所写的驱动程序，因此你会发觉，同样的游戏，在不同的显卡上跑起来会有差异，不论是外观或者是性能。

DirectX DLL 也会绘制图形，如果你没有使用 HAL driver 的话。这时候纯粹就是以软件来执行绘图，速度上当然就比硬件慢很多。

DirectX 的优点

以 DirectX 发展 game 的最大优点是你再也不用和一大堆恼人的硬件打交道。以显卡为例，users 所安装的显卡种类繁多，不同的显卡有不同的性能（capabilities）表现，以下是执行 DXSDK 中的应用程序 DirectDraw Editor 测试我的系统时显示的画面，如图 1-1 所示。

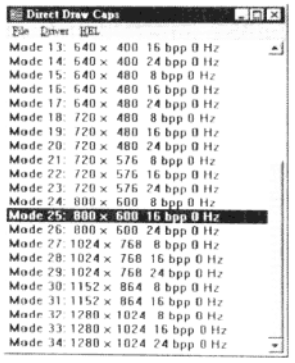


图 1-1

点取 Driver / Caps，出现以下的 window，如图 1-2 所示：

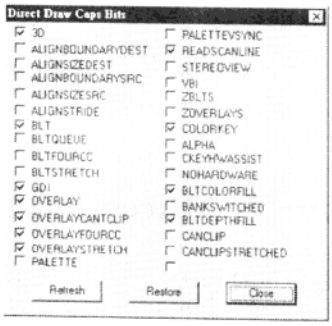


图 1-2

从上图我们可以了解显卡所支持的 capabilities 包括 3D 硬件加速功能，BLT（block transfer），一种硬件提供的功能（functionality），用来将系统内存中的图像数据传送到显示内存（display memory），于是你就看得到图像（image），这样比使用传统的 mov 或 rep movs 命令来将数据搬运（move）到 video memory 要快得多，Overlay（一种特殊的显示效果，就像在屏幕上的一个小窗口），Color key 等，再点取（click）Driver / Driver Surface Caps，则出现以下的 window，如图 1-3 所示：

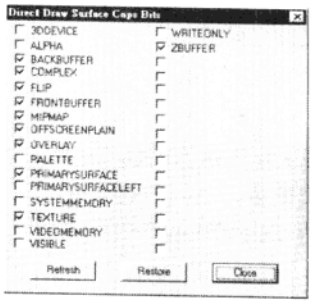


图 1-3

上图显示 surface（surface 简言之就是一个 block 的 video memory，以后会再介绍）的性能（capabilities），告诉你 display card 的 surface 可以规划为 back buffer，complex surfaces，flip-chain，overlay 等等，不同的显卡有不同的支持程度。

通过 DirectX，我们可以轻易地取得硬件的 capability 数据，根据以上的数据，我们可以针对不同的系统做最佳化（optimization）处理，充分发挥显卡的硬件性能，特别是 3D 加速器（3D accelerator）。

如果没有 DirectX，你必须一样一样地测试 user 的硬件性能，再最佳化，换言之，你手头上必须有所有硬件的使用手册（manual）或驱动程序（driver），光是这个工作就是一件很大的工程。

DirectX 还有一个好处，在 DirectDraw 和 Direct3D 中你大可不顾一切地写你的程序，如果你的程序所要求的性能该卡并不支持，DirectX 有 HEL（Hardware Emulation Layer 硬件仿真层面，其实是一个内建在 SDK Library 中的程序模块）仿真硬件功能。

比方你的系统上并没有 3Daccelerator，但是仍旧希望执行一个 3D game，此时可以使用 HEL driver，以软件仿真硬件性能，当然速度上是远不如 HAL。如果你的硬件支持所要求的性能，那么可以改为 HAL（Hardware Abstraction Layer）执行，大大发挥硬件速度上的优势。

2D 的 video card 有性能差异，3D 加速器也有性能差异。现在的 3D card 更是漫无标准，各家有各家的演算方式和 SDK，DirectX 尝试发挥它龙头老大的优势，迫使各家公司就范，希望厂商能够支持 Direct3D，而不是微软开发不同的 Direct3D 版本来配合厂商，虽然不愿意，事实上硬件厂商不得不俯首就范。因此即使鼎鼎大名的 3DFX 也要支持 DirectX，尽管他们也有一套叫做 Glide 的 SDK。

其他像 DirectSound，DirectInput 和 DirectPlay 分别处理音频，输入设备（如游戏端口）和网络等硬件设备，免除我们直接处理硬件的麻烦。

简单的说，DirectX 最大的好处就是以程序接口处理所有的硬件设备，也就是不必理会外围设备的厂牌，型号等，大大减轻程序设计员的负担。在设备越来越多，越来越复杂的今天，这无疑是大开方便之门。相反的，OpenGL，glide 等 3D SDK 的支持程度就没有如此广泛。

此外，DirectX 也担负起与 Windows 沟通的责任，比如说我们可以用 DirectX 决定产生一个正常的 Window 或是占用整个屏幕。

DirectX 的缺点

DirectX 优点固然值得大书特书，其缺点是绝大多数的人根本看不懂 DirectX 的帮助文件。过去 DirectX 的文件好像是写给已经全盘了解的人看，而不是初学者。不过从 DirectX 6.0 开始，文件越写越仔细，已经不输给市场上的任何参考书籍，但是它的 sample program 稍嫌深了一点，也缺乏详细的说明。另外，对许多只学过 MFC，不熟悉 Win32 SDK 的人也比较难以理解。其次是读者必须了解 COM。

在学习 DirectX 设计之前，你也必须精通（我不是吹牛，是精通）C++，因为 COM 本身就是面向对象。对于 windows 程序设计，也必须有相当的经验。除了编写程序不易，

DirectX 的 debug 也是一个大学问，每次错误的执行或者死机，都是对程序设计师能力的考验。

以上几个原因降低了许多人对游戏设计有兴趣的学习热朝，或者使现役的程序设计师决定转行。我讲的可能有点夸张，但也不会过份到哪里，Windows 程序设计本来就不简单，不然的话，Visual Basic，Delphi 和 C++ Builder 也不会走红地那么快。

不过用 VC++开发的程序，跑起来效率上是比较好，所以，辛苦还是会有代价的。除了 VC、VB 之外，Borland C++、C++ Builder、Delphi 也是可以开发 DirectX 游戏，但是必须偏重 component。

从 DirectX 7.0 版开始，增加了 VB 的函数库，对于习惯用 VB 的人来讲，是一大福音。

第二节 COM component

我们要有一个很重要的概念，那就是 COM 不是 program，也不是对象（object），它是一种以面向对象（object orientation）为主的格式（format）。遵循 COM 格式所写出来的程序模块（procedure module），就是 component。DirectDraw，Direct3D，DirectSound，DirectInput 和 DirectPlay 都是 component。遵循 COM 格式的 compiler 才能制作出 COM component，除非你要写一个支持 COM 的 compiler，或是想开发 COM component，否则并不怎么需要深入了解 COM 格式，但是身为程序设计人员（programmer），你至少要有一点概念，并且知道如何使用 COM component，就像本人。在此仅对 COM 做必要的介绍，有兴趣的人请自行参考相关书籍，如 Inside COM，Essential COM 等。

为何使用 COM Component

COM（Component Object Model）的好处如下：

1. 遵循 COM 格式发展好的软件，在改版或更新（update）时不需要原始 source，只要有 binary codes，也就是二进制的机器码就行。这样的好处实在太多了。比方说你现在写了一个 COM application，然后你很不满你的老板，想一走了之，顺便在走的时候放个病毒毁掉 source，没关系，你的继任者可以用机器码继续发展这段程序。以这种方式写出来的程序，只要 CPU 支持该机器码，再古老的程序也可以加以利用，加以改版，因此很可能现在写的 COM 程序在公元 3000 年的时候还可以推出更新版本。
2. 另一方面，现在的 C++ Compiler 为了让用户发展 class，必须提供基本 class（basic class）的 source，这样很容易就会被别人拿去使用，今后也不必担心了。当然，我们在 compile 程序的时候并不 include CWin 或 CObject 这些基本 class 的.h 文件或是.cpp 文件，这是因为 C++ compiler 将这些文件都在幕后处理的原因，但是在 debug 的时候就会 debug 到 source 中，这一点稍微有经验的 C++用户都知道。
3. 不必理会原来的 COM 程序是用什么程序语言（program language）写成，然而你可以用任何程序语言来发展 COM application。比方说一个 COM application（应该

说 module 比较适合，因为 COM 通常以 component 的形式供 application 使用）是由 VB 开发，你却可以用 VC++来写它的新版本。

4. 一个 application 可以由许多的 COM component（component）组成，比方说你要写一个 application，需要播放 mpeg，那么就可以将一个播放 mpeg 的 component 加到你的程序里面，免除你花三个月时间去琢磨如何写这种程序；需要网络功能，就可以将具有网络功能的 component 放到你的程序，你就不必花六个月的时间学习 winsock；要有播放 CD 的功能，可以插入具有此功能的 component，又省了六个月时间学习和摸索。将来除非你找不到合适的 component，否则很可能你的 application 大部分都是由 component 组合而成的。COM 可以大幅度减轻程序设计师的负担，提高软件生产效率。再加上第一项特性，你的后代在公元 3000 年时仍有可能采用你写的 component。
5. 通过网络，COM component 甚至可以做到跨平台，跨操作系统执行(DCOM)。当然在 COM 和平台或操作系统中间必须有一个媒介 interface 才行。

听起来有点不可思议，不是吗？经过这么多年，COM 并未被淘汰掉，相反地，它越来越普遍，越来越流行，什么，没听过 COM？那 OLE 或 ActiveX 总听过吧？这个东西就是架构在 COM 上面的。因此对 COM 再也没有排斥的理由。当然，要达到以上目的，前题是所有的程序都采用 COM 格式来制作才行。

事实上类似 COM 这种格式还有 OMG 的 COBRA 以及 IBM 的 OpenDOC 等，不过到今天能够大显身手的恐怕也只有 ActiveX。

COM 与 Class

COM component 的组成是数据（data）和 interface（Interfaces）。interface 类似纯抽象基本 class（pure abstract base class），如果你不知道纯抽象基本 class 是什么东西，赶快买一本 C++的书翻翻，一个抽象 class 只有声明（declare）一些 member function，却没有这些 member function 的定义（implementation），如下：

```
class PureAbstractBase
{
  public:
    virtual void func01()=0;
    virtual void func02()=0;
    virtual void func03()=0;
};
```

因为 member functions 没有经过 implement，因此你不能建立（create）纯抽象基本 class 的个体（instance）。Interface 就像纯抽象基本类别，它只声明了一些成员函数（member functions），但是并不实现（implement）这些 function 的内容，function 前的 virtual 语句表示这个 member function 并未 implement，必须由其衍生 class（derived class）来 implement，换言之，implement 所有的 interface function 是 component 的任务。

在 COM 中，一个 component 可以包含好几个 interface，就像一个 class 可以继承（inherit）好几个 parent class，因此在改版的时候，不需要重新改写程序，只要继承新的

interface 即可。然而 COM 并不等于 C++的纯抽象基本 class，因为你要定义一个新的 class 时一定要 parent class 的 source，但是在写新的 COM interface 时并不需要。再者纯抽象基本 class 是 C++规定的，COM 是微软的特产，以它来比拟 interface 是因为两者在 compile 之后产生的内存格式是类似的，如此而已，你千万不要将两者划上等号。

还有，COM component 既有的 interface 永远不能改变，就是说只能加入新的 interface 而不能修改旧的 interface。如果你只需要修改旧 interface 的一小部分，对不起，你还是要重写一个新 interface，因为旧的 interface 只有 machine code，没有 source 可以修改。这是 COM 的不便之处。所以 interface 就是 interface，虽然类似，你终究不能说 COM 是一个 C++ class，为了有别于 C++，interface 中的 member function 叫做 method，而不叫做 function。你只能用 C++写出纯抽象基本 class，但是你可以用任何语言写出 COM component，只要它支持 COM。

IUnknown interface

所有的 COM interface 一定是继承自 IUnknown 这个 interface，因此 component 本身和所有的 interface 都是以指向 IUnknown 的地址为起始地址。为什么要有 IUnknown？因为所有的 interface 都必须靠 IUnknown 来建立（create）或摧毁（destroy），也就是 destroy 和 release 内存。IUnknown 包含三个 methods：

```
Interface IUnknown
{
    HRESULT QueryInterface(REFIID riid,LPVOID* obp);
    ULONG AddRef();
    ULONG Release();
};
```

Interface 是一个相当于结构（structure）的新 data type，之所以不用 class 这个关键字来定义 interface，是因为 interface 的所有的 member 都默认为 public，方便模块（这里的模块，就是由好几个 source 所组成的一个完整的 project）中所有的 function 调用。Class 的 member 若未特别指定，则是 private 属性，这一点与 COM 不同。

Component 通常都是以 DLL 的形式存在，这样的说法不太对，因为一个 DLL 可以包含好几个 component，所以正确的说法应该是 component 通常存在于 DLL 中。在你需要使用 DirectX COM component 的时候，一般是以一个 CreateXXXX 或是 XXXXCreate 命令将 component 加载内存，XXXX 代表 COM component，比如我们就是调用 DirectDrawCreate 来建立 DirectDraw component，将硬盘中的 COM component 机器码加载系统内存。

AddRef()、Release()和 Reference Count

建立 component，就是将 component 由 DLL 中 load 到系统 memory，并不是一次将一整个 component 连带所有的 interface 和数据都加载内存，而是只加载一个 IUnknown 而已，IUnknown 的地址也就是 component 在 DLL 中的起始地址。如果你要使用某个 interface，必须以 QueryInterface()向 component 要求 interface，如果 component 支持该 interface，便会从

DLL 中 load 该 interface，并返回一个指向该 interface 的指针，然后调用 AddRef()，将 IUnknown interface 中的 reference count 加 1。如果 interface 已经不再使用，必须调用 lunknown::Release()，将 reference count 减 1，同时该 interface 所占的 memory 清除掉。如果 referencecount=0，即表示所有的 interface 都已经 release，IUnknown 也会从 memory 中自动清除。

当 component 或 interface 由 dll 中 load 到 memory 时，Component 或 interface 就可以使用，这些 components，interfaces 都是实体（instance），每一个存在 memory 中的 interface 都包含 data 或 method，因此也是一个对象（object）。

凡是有以下的状况，Component 应该自动调用 AddRef()：

1. 调用 CreateXXXX 或 CreateXXXX 建立 component 时。（其实建立 component 有好几种方式，但是不管什么方式，只要建立了 component，也就是取得了一个指向 component 的指针，就要调用 AddRef()）
2. 调用 QueryInterface()取得 interface 时。

在以上的情况中，AddRef()由 CreateXXXX(或 XXXXCreate)和 QueryInterface()自动调用，所以我们不须费心去调用 AddRef()。不过也有一些状况 IUnknown 不会调用 AddRef()：

1. 以等号（assignment）取得一个 interface 或 component 时。
2. 以 component 或 interface instance 作为函数参数（function parameter）。

除非你为 assignment 写了一个 overload，否则 compiler 一律以 bit-wise 的方式 copy 原有的 interface 中所有的 member，member pointer 的内容也照 copy 不误，结果新旧 interface 的 member pointers 事实上都指向同样的地址，修改任一 interface 的 pointer member 会影响到另一个。两个 interface 其中一个调用 Release()时，只会 delete 其中的一个，剩下的一个 interface，其 member pointer 所指的地址因为另一个 interface 已经被 release，内容就会被改掉。在这种状况，compiler 不会调用 AddRef()。

在作为函数参数（function parameter）时，compiler 一样会以 bit-wise 的方式 copy 一份 interface 到 function 中，function 也许会修改 copy 版 interface 的内容，结束时再返回这个 interface，但是这个 interface 并不会 release 掉，也就是真正从内存中清除。若是你在函数中调用 release()，也只有 copy 版的 interface 被 release，函数外的 interface 不会被 release()，但是 member pointer 内容会改变。同样的，compiler 也不会调用 AddRef()。

你会发现 DirectX COM 中的 method，如果必须以 interface 为 parameter，经常传递指向 interface 的地址的 pointer：如下例：

```
LPDIRECTDRAW7 pDD2;
DirectDrawCreate(IID_IDirectDraw7，(void**) &pDD2);
```

第 2 个 parameter 就是 pDD2 的 address。pDD2 本身是 pointer，因此就是该 pointer 的 address，而非 pointer 本身。

在本书的 samples 中，若是必须以 interface object 为 parameter，经常传递该 object 的 reference，事实上就是传递 object 本身，避免 compiler copy 该 object。

当 object 不再使用时，则应该调用 Release()，将 reference count 减 1，同时摧毁该 object，当 reference count=0 时，component 才会完全由 memory 中 release 掉。如果没有将 interface release 掉，那么该 object 就会占用内存。

在 DirectX 中，Release object 也有先后顺序，比方你必须先 create DirectDraw object 才能再 create Surface objects，若是先 release DirectDraw object 再 release surface object，会导致 error。

其他还有很多状况很容易让你忽略调用 Release()，比如以 loop 的方式 create object，create object 失败或是没有 create object 却还是 release 该 object 等，以后都会在书中看到。

AddRef()和 Release()若是使用错误，会造成 application 无法执行或死机，特别是当我们 create 相当多的 object 的时候。

GUID

在 DirectX 中，QueryInterface()的声明如下：

```
HRESULT QueryInterface(REFIID riid, LPVOID* obp);
```

第一个参数是一个 REFIID，根据 COM 的定义，不管这个参数的名称是什么，它都是一个 GUID。

GUID 是一个长度达 128 位的结构，用来作为 interface 的代号（其实不只 interface 而已，Applications，drivers，components 通通可以有 guid），不但 DirectX 中的 interface 以 GUID 做代号，全世界所有的 interface 也都是以 GUID 这种结构编制代号。微软企图为全世界每一个 interface 都编制一个不同的 GUID，因此 GUID 的产生过程是非常复杂的，但是我们在这个阶段并没有必要了解如何产生 GUID，我们要了解的是如何取得并使用 GUID。

在使用 DirectDraw 时，我们也免不了调用 QueryInterface()来取得 interface 指针，例如：

```
HRESULT Res= pDD->QueryInterface(IID_IDirectDraw7, (LPVOID*) &PointerToInterface);
```

其中的 IID_IDirectDraw7 就是一个代表 GUID 的常数（constant），要引用这个常数，你得在文件中 include ddraw.h 才行。在 ddraw.h 中也没有直接为 IID_IDirectDraw7 定义一个数值，而是调用一个宏来为这个常数产生 GUID：

```
DEFINE_GUID(IID_IDirectDraw7, 0xB3A6F3E0, 0x2B43, 0x11CF, 0xA2, 0xDE, 0x00,
0xAA, 0x00, 0xB9, 0x33, 0x56);
```

DEFINE_GUID 是一个简化许多复杂手续的宏，但是要使用这个宏，你还必须 include objbase.h，因为这个 DEFINE_GUID 是在 objbase.h 中定义的，所以在 include ddraw.h 之前必须先 include objbase.h，不然 DEFINE_GUID 就会找不到定义。在文件中 include objbase.h 和 ddraw.h，结果就会产生 GUID 的声明，要使用 GUID，还必须再加上一个 header 文件：initguid.h，如下：

```
include "objbase.h"
include "initguid.h"
include "ddraw.h"
```

如此一来才会产生 GUID 的定义，实际上也就是配置内存。在 C++中，每一个结构、