返回总目录

第一篇

DirectX

目　　录

第一章　DirectX 简介

到目前为止，**Microsoft Windows** 下的计算机游戏还没有一个辉煌的历史——它的成功还受到多媒体技术方面的限制。**Windows** 所提供的应用程序和 **PC** 平台之间的设备独立性使得游戏和多媒体开发者备受压力，这是因为设备独立性技术使得软件和硬件之间增添了许多中间层次，因此要想在 **Windows** 平台上生成平滑、快速的动画和紧凑、实时的输入和声音是非常困难的。**Windows** 的中心思想就是要把开发者和应用程序从硬件中分离出来，但这一点对于那些想直接操作硬件而获得最大速度的游戏开发者来说是致命的。市场需要的是高性能的游戏，因此，对于那些想把 **Windows** 作为计算机游戏平台的推广者来说，“**DOS!DOS!DOS!**”是他们经常遇到的对 **DOS** 游戏的赞歌**!**

1.1　DOS 已经过时

然而，**MS-DOS** 也有它自己的问题，其中最棘手的是硬件设备的支持。**PC** 机的游戏开发者是不能享受到游戏机开发者的那种平台一致性的。对于游戏机软件开发者，他们晚上可以睡得很香，因为白天所写的代码将在上百万台同样的机器上运行。而 **PC** 机的开发者却不能这样，他们老是梦见新

的图形协处理器、数字游戏杆、**3D** 加速卡和实时的输入设备，他们自己也知道，在下一个游戏中将需要支持更多的硬件。

所有的 **PC** 游戏都要利用目前最好的硬件以获取最佳性能，这使得那些小游戏软件开发公司很难跟上硬件发展的步伐。在他们的游戏刚放上柜台不久，新的硬件就出现了。由于这些硬件之间缺乏一致的接口，这就使得游戏开发者们不得不花费很多时间和精力来编写自己的硬件驱动程序以获取新硬件的最佳性能。但不幸的是，这种做法在大多的情况下会使得游戏软件难以安装以致于顾客们都会很恼火，同时也会增加维护费用。

一个像 **Windows** 这样的操作系统能够提供许多 **MS-DOS** 世界所不具备的性能。**Windows** 提供了很多优越性，这使得有关安装问题的咨询电话大幅度地下降。例如，**Plug & Play**（即插即用）功能使得用户安装最新的显示卡、输入设备或声卡都变得非常简单。在安装过程中，**Windows** 系统的注册器会询问配置信息，而且 **Windows** 的自动播放系统使得安装变成真正的自动化。但是直到目前为止，使用 **Windows** 这样的平台对游戏开发者来说是意味着严重的性能损失。

所有的这些情况都随着 **DirectX** 的推出而改变了。**DirectX** 向游戏开发者提供了一个健全并有良好支持的平台，使他们能够开发出高性能的游戏或多媒体应用程序。**DirectX** 又打开了 **Windows** 的一扇大门，使得开发者们不用损失机器的性能就可以获得高质量的游戏，同时他们也不用费尽心思支持各种硬件了。

1.2 加速 DirectX

在开始使用 **DirectX** 之前，我们有一个目标：就是使 **Microsoft Windows** 成为一个理想的游戏开发平台。这看起来很可笑，一台具有高级 **CPU**、巨大内存、图形卡、声卡和 **CD-ROM**，价值达 **2000-3000** 美元的 **PC** 机居然要竭力追赶价值仅有 **300** 美元的游戏机的显示质量。虽然在 **Windows** 也有诸如 **Myst** 这样的好游戏，但是我们需要做得更好——流畅的画面和动人的音响—— 一种在 **Windows** 桌面环境下的真实体验。

很显然，要达到上述效果，必须改变某些基本的东西。如果能够用 **DirectX** 技术来满足用户、软件开发人员和硬件供应商的需要，就可抓住这个千载难逢的机会。我们将和其它许多 **Windows** 用户一起享受高质量的游戏；开发人员将能够拥有一个功能更加强大的开发平台，可以获取系统的最佳性能同时降低维护成本；而对于硬件供应商则可以占有更广阔的市场份额。当然，这一切也使得 **Microsoft Windows** 成为一个更理想的平台。

我们意识到，如果能够在游戏设计中也引入原来 **Windows** 的设备独立性分层技术，并且在设计人员的创造性和游戏的运行效率之间进行很好的折衷的话，那么游戏程序的设计将变得很容易。在这种思想的指导下，我们开发了 **DirectX** 的如下指标：

- **DirectX** 必须由快速、底层的库组成并且不能给游戏设计增添很多约束。
- **DirectX** 的框架结构必须把硬件驱动程序的任务交给硬件厂商而不

是游戏软件开发者，因为硬件厂商更懂得怎样高效地使用自己的硬件。另外，在硬件厂商提供的驱动程序中，必须包含对当前硬件最先进技术的支持。

- **DirectX** 能够使开发人员开发出“真正”的桌面应用程序，它将与操作系统默契地配合。**DirectX** 组件必须能与其他已有的 **Windows** 组件协同工作。另外，**DirectX** 还必须允许用户程序把当前游戏窗口最小化并切换到另一个窗口以观察当前的金融趋势预测，而当用户切换回游戏窗口时，程序能够继续执行。
- **DirectX** 除了提供上述功能之外，还必须提供比 **MS-DOS** 更好的性能。

1.3 加速计算机工业

计算机工业常常会陷入类似于“鸡和鸡蛋”的困境——新的软件直到新的硬件出现之后才出现，而新的硬件只有当新的软件需要时才生产。我们想通过制定一种软件框架来打破这种死循环，从而把软件开发人员和多媒件硬件设备制造商分开。

我们不仅想要提供已有的硬件设备接口，而且还想扩展这些接口以满足未来的需要。这样做的好处是可以使硬件厂商明白自己所需提供的新功能。**DirectX** 现在已经开始着手提供这样功能，而有战略眼光的开发人员将利用

它的这些特性。一旦某个硬件厂商提供了框架中的某项新功能，这个新功能就可以被 **DirectX** 自动使用。同样，硬件厂商也可以通过分析已有的游戏软件而对它们提供更好的支持。

在上述情况下，用 **DirectX** 编写的游戏将能够运行于不同档次的硬件平台。想象一下，如果你买了一个游戏并且良好地运行了几个月，然后某一天你买了新的图形卡和声卡，用 **Windows** 的 **Plug & play** 功能把它轻易地安装到计算机上，然后你会发现自己的游戏已经变得更加生机盎然——它具有更加动人的动画、纹理和 **3D** 声音——所有的这一切都是自动完成的。

1.4 Directness 原理

在设计 **DirectX** 规范时，我们尽量用原理的方式来表达它的各部分特征，我们敲定原理的名称叫 **Directness** 原理。下面分三个部分详细讨论这三个规则：

- 更快速
- 最短延迟
- 底层接口

1.4.1 最快速

其中最重要的一点是，要使 **DirectX** 的操作速度尽可能地快。为了做到这一点，我们在各种可能的情况下寻求硬件厂商的协助。

下面举一个例子来说明我们是怎样用异步的方法来实现这一点的：对于那些硬件所提供的特定的功能，如内存数据的移动，**DirectX** 先设置执行这一操作，然后立即返回以执行下一条指令。这一功能将使得软件开发者能够从这种多媒体的并行结构中获得最大的性能。

1.4.2 最短延迟

Windows 是一个高层的操作系统，它已经把人们所能想象到的功能进行了抽象。这种抽象使得软件能够从特定的运行环境中独立出来。但遗憾的是，这一技术也意味着当你要从 **Windows** 中申请一项服务时，**Windows** 将要进行一系列“思考”。这个“思考”过程对于打印文档这类的应用程序来说是无关紧要的，但对于游戏软件来说是不可接受的。假如游戏中一方击中了另一方的鼻子，那么你肯定想要在击中鼻子的一瞬间让扬声器发出“啪”的一声。在这种情况下，时间延迟是很重要的。对于一个好游戏来说，时间延迟越短越好。

1.4.3 底层接口

当生成一个软件库之后，你总会按一定的“标准”套路去使用它。在某些情况下这当然是正确的——这种使用软件库的标准方法可以使开发者和用户都从中受益。但这种情况对于游戏开发者来说是不适用的，游戏开发者们总是用创造性和革新性把它们的游戏组合在一起，而正是这种创造性和革新性才使得他们的游戏软件与众不同。我们现在的主要任务不是引入游戏设计的高层接口，而是引入具有很大灵活性的底层接口，以使游戏开发者们开发出更好的多媒体和游戏应用程序。

其中的一个例子就是 **DirectPlay API**，它为游戏提供了与介质无关的通信接口。许多软件开发者对 **DirectPlay** 的最初反应是：“什么？它所做的仅仅是发送消息。”但事实上，它要做的事情要多得多。通信接口对于象象棋之类的游戏来说功能已经足够了，但对于赛车游戏却不行。因此，我们决定提供所有游戏程序所必须的共同接口。这将使得游戏开发人员能够更加充分地利用系统的性能，而且第三方也可以给 **DirectX** 添加接口。

1.5 Direct 结构

在我们刚使用名词“**Direct**”的时候，曾经引起了人们一些骚动。它实现了本不应该在 **Windows** 底下实现的东西——直接访问硬件，独占资源。

但事实上，我们的目标不是要造成混乱，而是要以设备无关的方法来提供设备相关的性能。

我们在这里所提出的结构是这样的两个驱动程序：硬件抽象层（**HAL**）和硬件模拟层（**HEL**），它们两者组合起来一起响应 **DirectX** 的请求。当 **DirectX** 对象创建时，它将查询相应的硬件并把参数填入“兼容表”。如果硬件能够提供某种特定的功能——例如，某图形协处理器可以处理拉伸操作——那么 **DirectX** 将会直接调用硬件所提供的功能；如果硬件不支持某项功能，那么与其等价的某 **HEL** 命令将会被调用。

如图 **1-1** 所示，**HAL** 层紧紧地包围了硬件层，而 **HEL** 层则给程序开发人员提供了一组必要的、互相独立的接口。当更先进的硬件生产出来之后，以前编写的软件将能够自动地利用硬件的先进特性。例如，当 **Intel** 推出 **MMX** 技术之后，**DirectX** 马上实现了对它的支持，这样，用 **DirectX** 开发的游戏程序都能自动地利用 **MMX** 技术，使程序运行更快。这时你所要做的仅仅是从 **Microsoft** 或游戏供应商那里下载最新版本的 **DirectX**，这样你的游戏就成为了 **MMX** 技术的优化版本。

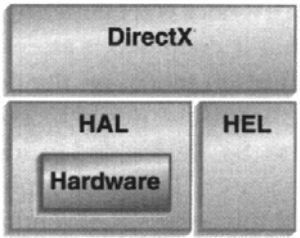

图 **1-1　DirectX HAL/HEL** 结构图

1.6　DirectX 组件

随着时间的推移，**DirectX** 组件越来越多。**DirectX 5** 包括如下组件：

DirectDraw　使用页面切换的方法提供动画，直接访问图形协处理器，内存的管理。**DirectDraw** 是 **DirectShow** 和 **Direct3D** 的基础。

Direct3D　在本书中仅提到它的部分内容。它提供了高层和底层的 **3D** 硬件接口。

DirectSound　提供了立体声和 **3D** 声音效果，同时管理声卡的内存使用。

DirectPlay　为多人游戏软件提供了消息服务，同时还提供启动和组

织多人游戏的功能。

DirectInput 为大量的设备提供输入服务，同时还支持输出设备。

DirectSetup 自动安装 **DirectX** 驱动程序。

1.7 小结

- **DirectX** 以设备无关的方式提供了访问多媒体的低层接口。
- 通过 **HAL** 层和 **HEL** 层，**DirectX** 使用大量的硬件配置文件为大范围的硬件提供一致的支持。
- 一旦新的硬件出现，**DirectX** 应用程序可以立即从硬件的新特性中受益。
- **DirectX** 不仅仅用于游戏软件开发人员，还可以把它用于诸如动画播放器，声音编辑器等应用程序。

第二章　基　础

在编写 **DirectX** 的代码之前，我们必须做一些准备工作。在本章中，将要讲解几个专题，为以后打好基础。

2.1　期望什么

第一版的 **DirectX** 也叫做 **Game SDK**（游戏 **SDK**）。我们之所以要开发 **DirectX** 是因为 **Windows** 底下还有很多未开发的潜能——包括快速运行游戏的潜能。当时我们开发小组中的大部分人都认为自己所做的努力将仅仅只有益于游戏行业。虽然有这种想法，但由于我们要全身心地投入代码编写中，所以也没有时间考虑别的。后来当 **Microsoft** 的 **Office** 小组向我们请教“**Game SDK** 是什么东西”时，我们才意识到自己所做的事情确实是具有重大意义的。我们原来一直没想到流畅的动画也可以用于画“饼图”和宇宙飞船等。后来，我们的工作从原来的游戏转变到了更为广泛的多媒体世界。**DirectDraw**、**DirectPlay** 和所有的其他“**DirectSomthing**”都统统称为 **DirectX**。

就像 **DirectX** 对于大多数应用程序都有用一样，我们希望本书也能适用于大量的应用程序。在本书中，将避免针对某类特定的游戏。在这里你是不能找到怎样创建自己的 **3D** 引擎、侧滚引擎等的原代码的。如果你需要这方面的书籍可以找另外的参考资料。但一旦你读完本书，你可以使 **DirectX** 按照你发出的指令运转，无论是要开发游戏软件或股票市场分析软件，你都会感到得心应手的。

2.1.1 Windows 要素

由于很多 **PC** 游戏是基于 **MS-DOS** 的，因此好多游戏开发者对于 **Windows** 是陌生的。关于怎样在 **Windows** 下编程这个问题已经超过了本书的范围——这里没有这么多的空间。如果你发现自己对 **Windows** 下的编程技术不很了解，那么可以参考一下其他的参考书。一般情况下，**Charles Petzold** 的《**Programming Windows**》是这个领域内的一本好书。

2.1.2 编程语言

一般情况下，用 **C** 或 **C++**语言访问 **DirectX** 是最容易的，但 **C** 与 **C++** 不是唯一的语言。例如，**Microsoft** 为 **Java** 提供了 **SDK**，这样 **Java** 应用程序就可以访问 **DirectX** 的服务。另外，第三方提供的符合 **Microsoft ActiveX** 标准的控件也可以使网页或其他支持 **ActiveX** 的程序访问 **DirectX**，每个编

程人员都可以选择自己最喜欢的语言。对于我们这些操作系统的开发人员来说，主要使用 **C** 语言来开发。以后你浏览 **SDK** 所提供的范例子程序时，你会发现它的绝大部分代码都是 **C** 风格的，其中也有一些是 **C++**风格的。

我们之所以选择 **C** 语言作为 **SDK** 的开发语言主要是因为我们对 **C** 语言最熟悉——**DirectX** 本身也是用 **C** 语言写成的。当然，对于开发你自己的应用程序来说，使 **C++**将比使用 **C** 语言更具有优势。我们之所以用 **C** 语言的方式提供大部分 **SDK** 源代码是因为 **C++**的面向对象的特性会使我们本来要表达的意思变得晦涩难懂，另外，如果你能读懂 **C++**语言，那么读懂 **C** 语言肯定没有问题，但反过来却不一定。

2.1.3 源代码

在本章的前面曾经提到过，我们的目标不是教你如何编写游戏程序，而是教你怎样使用 **DirectX**。当然，由于 **DirectX** 主要是用于设计 **PC** 游戏的，所以如果用某种游戏效果来演示 **DirectX** 的概念，效果将会更好。当然，在本书提供的例子中，我们还是削减了那些与 **DirectX** 无关的代码，比如说 **Windows** 的框架代码、用户界面代码以及复杂的游戏逻辑代码，而仅保留能说明 **DirecctX** 概念的代码，所以本书讲解的基本上是纯的 **DirectX**。

2.2 COM（对象组件模型）入门

如果一见到 **COM** 就想回避它，那你就错了。**Microsoft** 的 **COM** 现在可以说是无所不在——它是许多 **Windows** 产品的基础，甚至于 **Windows** 本身都包含了 **COM** 组件。**COM** 原先是为了支持对象链接与嵌入而开发的，但现在它的应用范围已经远不止这些了。

COM 为模块软件的开发方式提供了基石，它不仅便于使用，而且还有其他许多优点。由于 **COM** 最初是为 **OLE** 提出的，所以它的名誉可能会因此而受到许多损失。**OLE** 虽然是一种灵活、功能强大的技术，但同时它也是复杂、速度缓慢、难于掌握的代名词。而相反的是，**COM** 技术却显得非常简单、灵活，它仅在应用程序前面添加很少的头部标识。现在，**COM** 和 **OLE** 这两种技术已经各自独立发展了。

如果你对 **COM** 技术感兴趣，甚至于想做自己的软件模型，那么可以参考附录中列出的有关 **COM** 的书籍。幸运的是，如果我们仅要开发 **DirectX** 应用程序，仅掌握 **COM** 某些基本功能就可以了，而不必去实现它。正因为这样，我们对 **COM** 仅做简单的介绍。

COM 是一种标准，它定义了软件对象或组件之间的交互规则。一个 **COM** 对象通过接口的方式提供自己的功能。所谓接口是指一组永不改变对象方法。现在 **COM** 技术已经成为许多 **Windows** 产品的基石（包括 **OLE** 和 **DirectX** 在内）。**DirectX** 的 **COM** 模型实现就象 **COM** 本身一样简单，所以现在还称其为 **COM** 技术——但一些纯化论者却不同意这种观点。这样，

为了符合 **COM** 对象的规则，所有的对象必须有一个公用的接口 **IUnknown**，也就是说接口至少必须要实现 **IUnknown**，请参考表 **2-1**。表中的三种方法用于控制对象的生命周期和访问对象的接口。

表 **2-1** **IUnknown** 的接口方法

方法	目标
AddRef	增加对象的引用计数器
Release	减少对象的引用计数器
QuerygInterface	获取某个特定对象接口的引用

2.2.1 生命周期管理

COM 模型可以同时支持多个用户或客户。由于 **COM** 的多个用户不知道对方是否存在，因此用户是无法确定到底什么时侯应该把 **COM** 删除——因为其他的用户很可能正在使用 **COM** 对象。这就意味着 **COM** 对象本身应该具有管理自己生命周期的功能。这个功能是通过对象内部的一个计数器和 **AddRef** 和 **Release** 方法来实现的。当一个新的用户连到 **COM** 对象后，对象内部的计数器就加 **1**。

用户是不能删除 **COM** 对象的。当一个用户断开与 **COM** 对象的连接之后，可调用 **Release** 方法，使对象内部的计数器减 **1**。当 **COM** 对象内部的

计数器数值减到 **0** 之后，对象就把自己从内存中删除了。在大多数情况下，当一个用户连到 **COM** 上后，对象内部计数器是自动加 **1** 的，但在有些情况下，比如当用户拷贝一个连接时，必须手工调用 **AddRef** 方法使对象内部计数器加 **1**。

所有的这些加数和减数操作听起来很繁琐，但事实上这些过程几乎完全是自动的。你所要做的就是在连接结束时调用 **Release** 方法，而在建立一个 **COM** 对象不能感知的连接时调用 **AddRef** 方法。有时候系统可以提供给你某些连接，这时你就可以认为对象内部的计数器已经加 **1** 了（事实上也是这样的）。以后当我们再遇到 **COM** 对象的操作时，再进一步讲解对象内部计数器的工作过程。

2.2.2 接口管理

为了让 **COM** 对象能正常地工作，参考计数就像是一项繁琐的簿记工作，与此不同的是，接口和版本管理却要清晰明了得多。**COM** 规范中定义了一套机制，使得用不同语言编写的 **COM** 对象可以互相访问，同时还可以在修改 **COM** 对象时不影响它原来的接口定义。**COM** 的这些性质使它具有了极大的优越性。使用 **COM** 之后，即使我们 **Microsoft** 发行了更新版本的 **DirectX**，只要新版 **DirectX** 与旧版兼容，那么程序在使用新版 **DirectX** 时照样能正常工作。

使用接口这一概念，**COM** 使得程序编制变得更加简单。在前面已经提

到过，所谓接口就是一组能够用于操作 **COM** 对象的方法。这些方法通常是按照它们的功能进行分类的。一旦定义好了一个接口之后，这个接口将永远不再改变。如果某个已经定义的方法要改变参数或实现方法，那么必须重新建立一个新的接口。

例如，你为某个零售店开发了一个软件包。在应用程序中，用到了另一个公司提供的一个组件，而这个组件与你的应用程序是分开发行的——最终用户必须从另一个公司购买这个组件，而组件的功能又与用户所付的价格有关。其中的一个对象如图 **2-1(a)**所示——一个“**Cash Register**”对象，它允许用户为商品和服务付款。这是一个基本的对象，它只支持一个接口 **ICash**。图 **2-1(b)**所示的对象是一个更高级一点的“**Cash Register**”对象，它还支持 **IPlastic** 接口，用于处理信用卡。最后，在图 **1-2(c)**中列出了图 **1-2(b)**的修正版，它支持两个版本的 **ICash** 接口，其中新版的 **ICash** 接口可以用于处理消费卡。购买你的软件的客户可以安装其中的任何一个对象。通过使用 **COM** 模型，你的程序可以支持上述所有的对象。

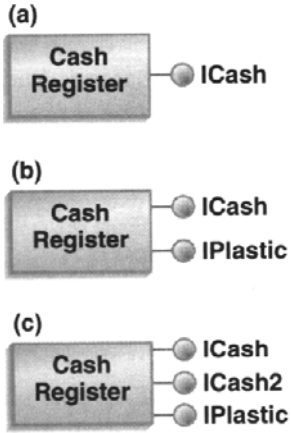

图 **2-1** “**Cash Register**”对象的三种不同版本

COM 的 **QueryInterface** 方法使上述功能得以实现。**QueryInterface** 通过一个唯一的接口标识（由开发者提供）来获取对某个特定接口的引用。如果这个函数调用成功，表明对象支持这个接口（我们就可以使用这个接口）；如果函数调用不成功，表明对象不支持这个接口，这样应用程序就必须恰当地处理这种情况。在你给零售店提供的软件包中，可以调用 **QueryInterface** 函数分别检查 **ICash**、**ICash2** 和 **IPlastic** 接口，然后根据检查的结果，屏蔽掉用户界面上的某些功能。

继续上面的例子。假如现在你把软件包卖给那些期待已久的用户了，而且一切正常。然后几个月之后，“**Cash Register**”的开发者又发布了它的一

个新版本，在这个版本中新增了 **ICheck** 接口。你的客户有的购买了新版的“**Cash Register**”，有的没有，这样，一切还都是正常的。也许有的客户会发现你的软件并不能支持新的 **ICheck** 接口——但这怎能怪你呢？你在编写软件包时根本就不知道存在这个功能。由于“**Cash Register**”的开发者没有改变它原有的接口设置，所以你原来开发的软件包在新版“**Cash Register**”下仍能够正常地工作。等到你发行了新一版的软件包之后，在程序代码中会加入查询 **ICheck** 的功能，这时，购买你新版软件的用户将可以从“**Cash Register**”的新增功能中受益。所以，不管怎样配置你的软件包和“**Cash Register**”部件，它们总是能够稳定地工作的。

我们也用同样的方法来升级 **DirectX**——新的功能用新的接口。由于 **DirectX** 可以免费下载，所以大部分用户都可以马上拥有它的最新版本。当然，对于那些没有最新版本的用户，只要应用程序是经过仔细设计的，也可以正常工作。在前面已经提到过，**DirectX** 的版本总是与以前的版本兼容的，因此旧版本 **DirectX** 中的接口是不会被改变的，这样，即使另一个应用程序安装了一个更新版的 **DirectX**，原来的应用程序也不会受到影响。在后面的章节里，我们将给出详细的例子演示怎么使用 **COM** 和 **DirectX**。

2.3 编程经验

当你浏览 **SDK** 例子程序的代码时，会发现代码的风格很不一致，这是

因为这些代码是不同的开发者编写的。在本书的例子中，我们尽量使代码风格一致。下面我们列举了某些在编程中经常遇到的规则问题，列举如下：

2.3.1 调用方法

C++和 **COM** 都使用 **Vtable** 结构来访问方法。**Vtable** 结构包含指向每个方法的代码的指针。所有的方法调用都是通过 **Vtable** 结构进行的。例如，在 **C** 语言中，调用 **DirectDraw** 对象的 **Release** 方法，格式如下：

```
lpDD->lpVtbl->Release(lpDD);
```

当用 **C** 来调用 **DirectX** 方法时，总是必须把指向当前对象的指针作为第一个参数。这个参数也被称为 **this** 指针。同时，方法的调用必须通过 **Vtable** 结构。当使用 **C++**来调用 **DirectX** 方法时，**C++**可以自动处理很多细节，所以调用形式可以简化如下：

```
lpDD->Release();
```

在 **SDK** 的例子中，你会发现同时存在上述两种调用格式。但在本书中，将一致采用 **C++**风格来表示。事实上，这两种调用方法的执行过程是一样的。

2.3.2 测试返回值

在每次发行 **DirectX** 时，我们都要提供一些代码以演示它的新功能。遗

憾的是，编写 **SDK** 例子的开发人员总是优先考虑怎样让 **DirectX** 运行起来，而且他们在编写代码时，总是要编写一些新的程序代码，而不是设法修饰原来已有的代码。这样做的结果是这些代码运行起来并不象原来设想的“安全 **DirectX**”那么可靠。其中的一个例子是测试返回值。在这里，我们建议读者要尽量避免有些 **SDK** 例子中所使用的方法：

if(lpDDSBack->GetDC(&hdc) = = DD_OK)

而用以下代码代替：

if(FAILED(lpDDSBack->GetDC(&hdc)))

其中 **FAILED** 宏和 **SUCCEEDED** 宏都是在 **Windows** 的头文件中定义的。当发生错误时，**DirectX** 总是返回一个负值。对于目前的版本，**DirectX** 成功时返回数值 **0(??_OK)**。在将来，**DirectX** 也许会用返回正数来表示函数的正确执行，在这种情况下，如果用判断**_OK** 的方法来检测返回值反而会导致错误。当然，在实际开发中如果函数的返回值变了，将会重新定义一个接口，这样使得用老接口的应用程序也不会受到影响。但在这里，我们还是强调：使用宏是一个好习惯。在本书中的所有例子中，都将采用宏来检测返回值。

2.4 调试 DirectX

DirectX 应用程序，特别是那些用到 **DirectDraw** 的应用程序，调试起

来是相当困难的。用 **Visual C++**所提供的代码级调试工具来得到 **DirectDraw** 的屏幕输出结果是不容易的，同时它所提供的错误信息也是很不明确的。当然，我们还是可以采取一定的措施，以减少调试代码时所遇到的挫折。

2.4.1 核心调试器(Kernel Debuggers)

核心调试器可以把自己嵌入到操作系统的底层，它的功能十分强大。一般的集成开发环境都提供了核心调试器或远程调试工具，它们可以把调试结果输出到另一台计算机或终端上。如果你想在调试 **DirectX** 时少遇到些挫折，那么最好用另一台计算机作为监视器。能够用于做“哑终端”的机器都可以用作调试监视器。

2.4.2 Debug DLLs(调试动态库)

用户经常抱怨 **DirectX** 的错误代码晦涩难懂。虽然我们在每一次发行 **DirectX** 的时候都会提供更为详细的错误信息，但这看起来还远远不够。其中最糟糕的是 **DDERR_GENERIC** 错误代码，它使用户感到迷惑不堪。另外，即使像 **DDERR_INVALIDPARMS** 这样较为详尽的错误代码，在用户要同时传递十个参数时也不见得能提供充足的帮助。事实上，上述的问题可以用如下方法解决。

当你在安装 **DirectX** 的时候，会遇到如图 **2-2** 所示的选择。如果你选取

Debug

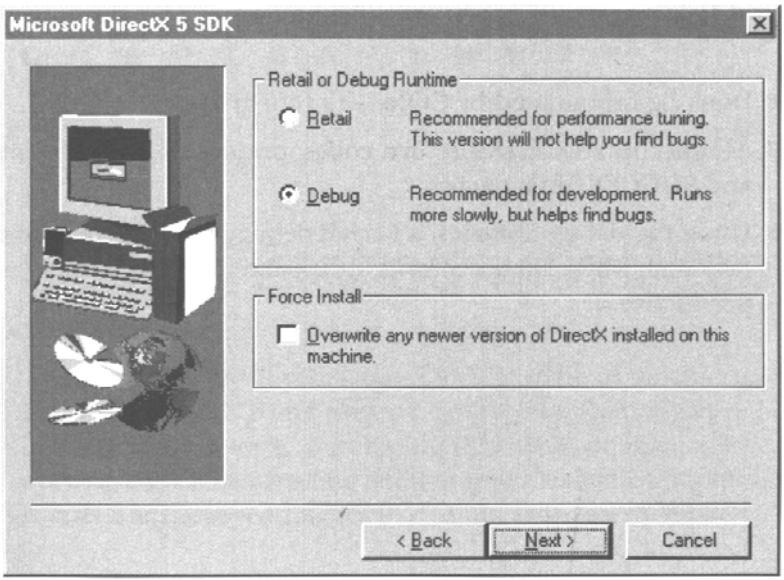

图 **2-2**　安装 **DirectX** 调试库

选项，**DirectX** 将安装一组专门的库，用于在你的调试窗口（或远程调试监视器）上生成各种注释。

例如，假设你在调用 **DirectDraw** 的 **CreateSurface** 方法时，得到了 **DDERR_UNSUPPORTED** 错误代码，或者在另一次调用时又得到了

DDERR_INVALIDCAPS 错误代码。这时你就要花费很多时间来确定问题到底出在什么地方，这时就可以使用调试动态库所给出的信息。在第一种情况下，调试信息输出如下：

Creating flippable mip-map chains with a single call is not yet implemented

而在第二种情况下，调试信息如下：

Can't specify both SYSTEMMEMORY and VIDEOMEMORY

这些调试信息再加上调用 **CreateSurface** 方法的具体过程，你将很容易找到问题所在，从而节省调试时间。不过请注意，使用调试库的应用程序要比使用产品库的程序运行速度慢。虽然组合使用远程调试监视器和调试库是你最佳选择，但你也不一定非得这么做。

2.5 总结

- 不要被 **COM** 所吓倒——它是你的朋友。
- 要检查 **DirectX** 的返回值，而且使用 **FAILED** 和 **SUCCEEDED** 宏来检查。
- 虽然在开发过程中，不一定要把调试库、核心调试器和远程调试终端都用上，但如果能很好地利用这些工具，开发效率可以有很大的提高。

第三章　开始使用 DirectX

在本章中，将讲解某些安装 **DirectX 5** 的细节。即使在购买本书以前，你已经安装了 **DitrectX** 并且工作很正常，花费一些时间仔细阅读本章还是值得的，在这里，你将会发现一些很有价值的信息。

3.1　安装

DirectX 5 的完全安装将包括库文件、头文件、例子程序源代码、多媒体游戏程序以及文档。它大约需要 **80M** 硬盘空间。由于安装过程是由 **AutoPlay** 启动的，所以基本上没有什么需要解释的——只要把 **CD** 放入光驱，安装过程就自动启动了。如果没有自动启动，则可按以下步骤操作：

1. 从 **Start** 菜单中选取 **Run** 命令。

2. 单击 **Browse** 按钮，找到光驱所在的位置。

3. 选择 **SETUP.EXE,**并单击 **OK** 按钮。

一系列向导屏幕会引导你完成整个安装过程。在安装过程中我们建议选择 **Complete Installation** 选项，并且还要选取 **Debug** 选项，以节省以后的

程序调试时间。

在安装完成并且重新启动计算机之后，可以在硬盘的**\DXSDK** 目录下找到 **DirectX SDK**。

3.2 文档

DirectX 的开发技术文档的编者必须竭尽全力才能赶得上开发组的进度，因为开发组的进度很快。你可以在**\DXSDK\DOCS** 目录中找到他们的劳动成果。

3.2.1 Windows 帮助文件

标准的 **Windows** 帮助文件将会被安装到**\DXSDK\DOCS\WINHELP** 目录。这些帮助文件包含有主目录、搜索索引以及所有 **Windows** 帮助系统所应具备的东西。如果你是用 **Microsoft Visual Studio** 来开发 **DirectX** 的，那么只能从 **Windows** 的开始菜单中访问这些帮助文件，这是因为这些帮助文件与 **Visual Studio** 的文档系统不兼容。

3.2.2 Word 文档

在**\DXSDK\DOCS\WORDDOC** 目录下保存了与在线帮助系统内容完全一样的 **Word** 文档，以便于用户打印和阅读。如果你喜欢阅读打印出来的文档，可以使用这个目录下的内容。不过请注意准备很多纸，因为所有的这些文档打印出来会超过一千页的。

3.2.3 HTML

在**\DXSDK\DOCS\WEB** 目录下包含了分类的 **HTML** 文件，可以用网页浏览器来查看它们。在开始菜单中，已经加入了指向这些 **HTML** 文件的快捷方式。这些文档不仅包括 **DirectX** 开发人员写的文档，而且还包括连到某些 **Internet Web** 网址的超链接。其中可以找到很多 **Microsoft** 技术人员写的关于 **DirectX** 的文章，这些文章都是值得一看的。

3.3 例子程序源代码

在 **DXSDK\SDK\SAMPLES** 目录下，包含了 **60** 多个从简单到复杂的例子。其中某些例子的可执行文件存放于**\DXSDK\SDK\BIN** 目录下，这样不

用编译任何东西就能执行某些例子，从中可以体会到 **DirectX** 的效果。

FoxBear 和 **Rockem** 是经常用于测试 **DirectX** 的两个例子。它们的可执行文件分别放置在**\DXSDK\FOXBEAR** 和**\DXSDK\ROCKEM** 目录下，而它们的源代码则跟其它的例子放在一起。**FoxBear** 的例子如图 **3-1** 所示。对于狐狸，它的主要目标是逃得越快越好，以免被熊吃掉。请别担心，狐狸可以轻而易举地逃离，然后自由自在地奔跑。**FoxBear** 展示了用 **DirectX** 实现平滑侧滚动画的效果。

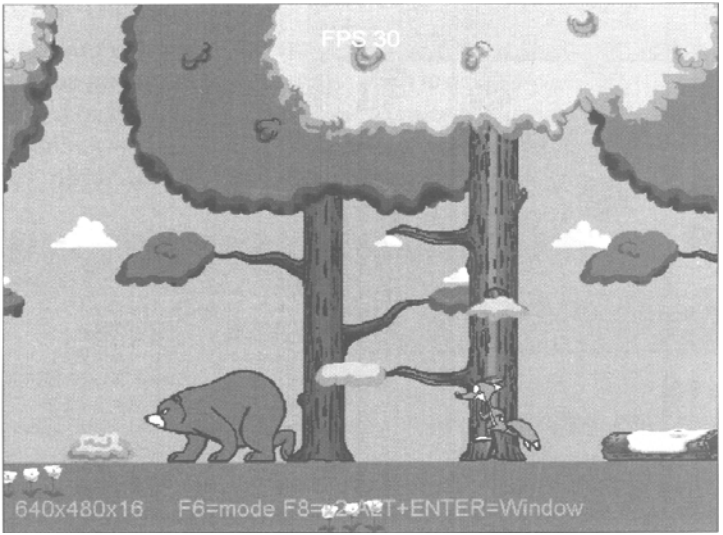

图 **3-1**　狐狸和熊

在前一段时间，我们用 **FoxBear** 来演示在 **Windows** 底下实现的良好动画效果。**FoxBear** 的功能键可以让用户切换显示模式、切换到 **Windows** 的一个窗口或查看动画的帧。**FoxBear** 是一个用于展示 **DirectX** 特征的优秀例子。

3.4 其他有用的信息

在本单元中，将介绍一些有用的工具。用于监视、调试和定制 **DirectX**。

3.4.1 控制面板

在系统的控制面板**(Control Panel)**中，可以看到 **DirectX** 的图标，如图 **3-2** 所示。双击图标会出现 **DirectX Properties** 对话框。

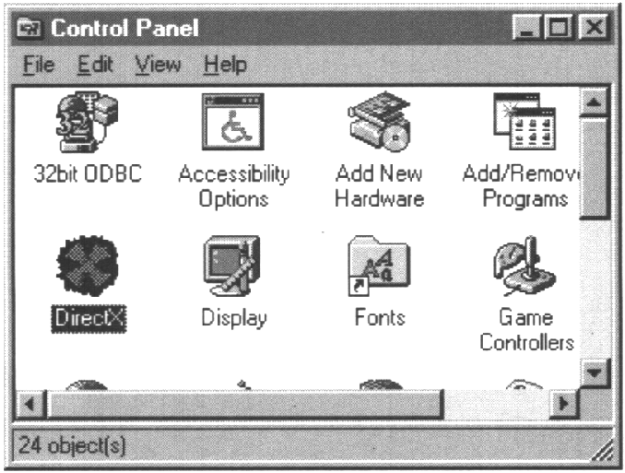

图 **3-2**　控制面板中的 **DirectX** 图标

这个标签对话框用于提供 **DirectX** 的配置信息。你也可以通过它来修改 **DirectX** 的注册信息以控制 **DirectX** 的行为。如图 **3-3** 所示的标签页面提供了当前 **DirectX** 的版本信息。如系统由于安装了 **DirectX** 而不能很好地工作，可以用 **Restore Drives** 按钮来恢复以前的驱动程序。其他的标签页面则用于提供每个 **DirectX** 组件的信息，这将在后面的章节中讨论。

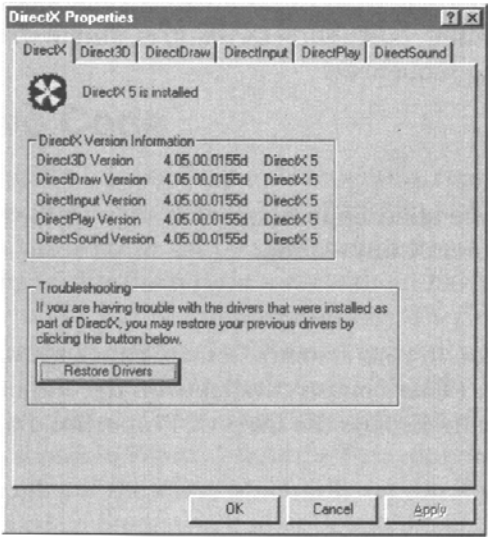

图 **3-3　DirectX Properties** 对话框

3.4.2　DirectX 设备查看器

在安装完 **DirectX** 之后，可以用 **DirectX** 设备查看器来查看它的布局。这是一个很得力的工具，如图 **3-4** 所示。通过它可以查看 **DirectX** 的每一个角落。每个 **DirectX** 组件都有自己的一个文件夹，这样 **DirectX** 的信息就被

划为易于管理的几组。

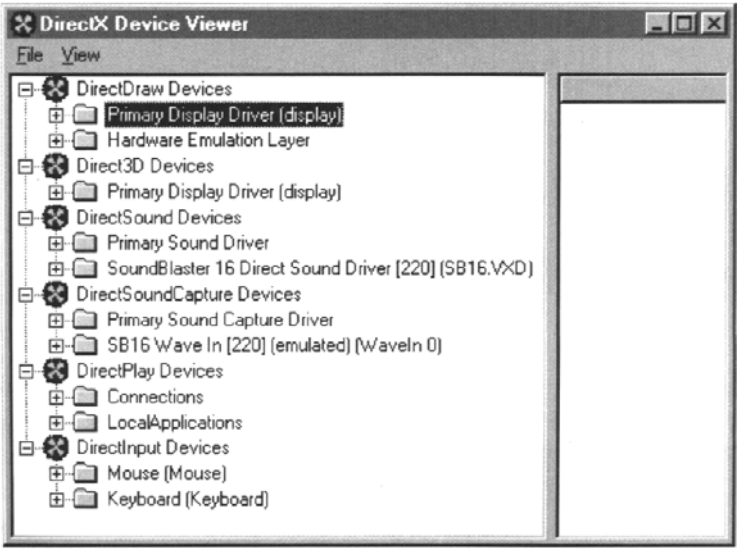

图 **3-4** **DirectX** 设备查看器

每次为新的硬件安装完 **DirectX** 后，几乎总是调用 **DirectX** 设备查看器来判断安装是否成功。图 **3-5** 列出了 **DirectDraw** 的几级文件夹，其中列出了显示卡的能力（见右边的面板）。为了使得你的界面与书上的一致，请从 **View** 菜单选择 **All** 命令。

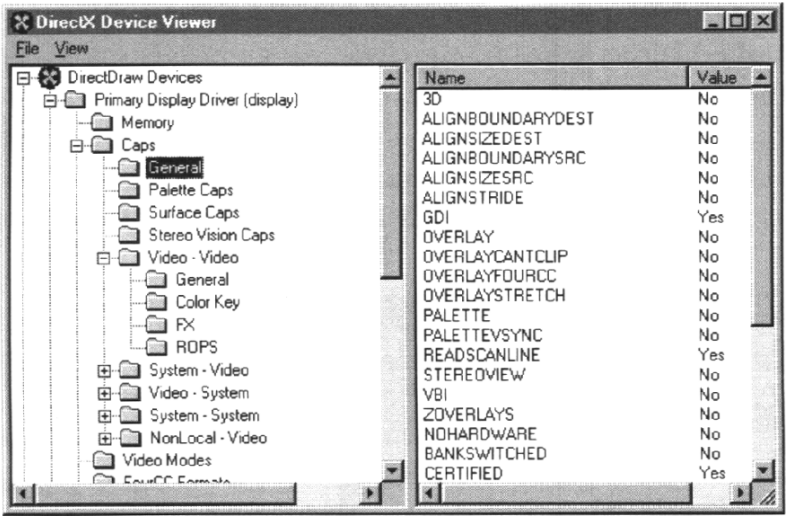

图 **3-5** 用 **DirectX** 设备查看器来查看显示能力

3.4.3 DXBug

这个小工具如 **3-6** 所示。它主要用于生成 **DirectX** 的错误报表，然后用户可以把它发往 **Microsoft**。**DXBug** 将收集你计算机的信息和 **DirectX** 各个组件的版本信息。计算机的信息主要包括错误被发现时的硬件、软件配置

信息。另外，你可以提供额外的信息以使你的报告更加具体，这样将有助于我们重新模拟错误的产生过程。

3.4.4 测试应用程序

你可以使用 **SDK** 例子中所提供的几个测试程序来测试 **DirectX(**比如说 **DirectDraw Test** 等**)**。我们将会在后面的章节中使用这些测试程序，到时侯再讲解其中的具体细节。图 **3-7** 展示了其中的一个例子——**DirectDraw** 测试应用程序，在这里你可以直接使用 **DirectDraw**。

3.5 使 DirectX 开始工作

在使用某些例子验证 **DirectX 5** 可用之后，你可能想要用自己的开发环境来编写一些例子。编程中所需要的 **DirectX** 库文件在**\DXSDK\SDK\LIB** 目录下，而头文件则放在**\DXSDK\SDK\INC** 目录下。所有的这些路径都必须加到开发环境的搜索路径中。如果你正好在使用 **VC++5.0**，那么可以从 **Tools** 菜单选择 **Option** 命令来添加这些路径，如图 **3-8** 所示。

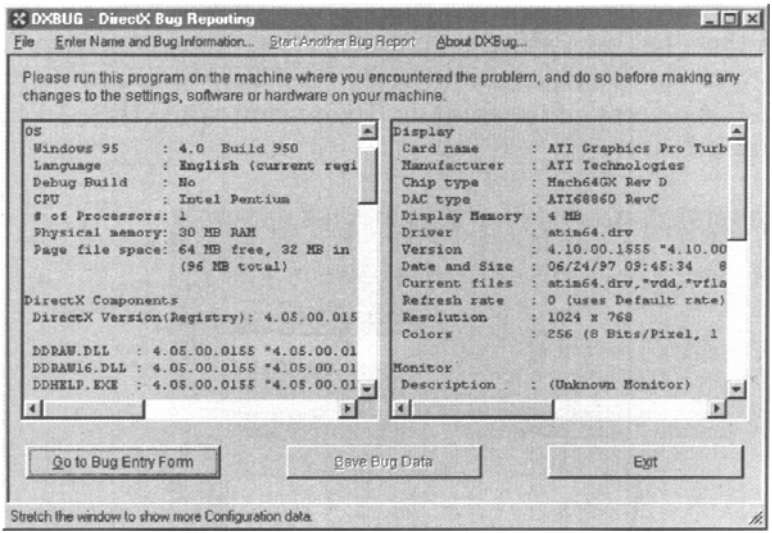

图 **3-6　DXBug** 报告工具

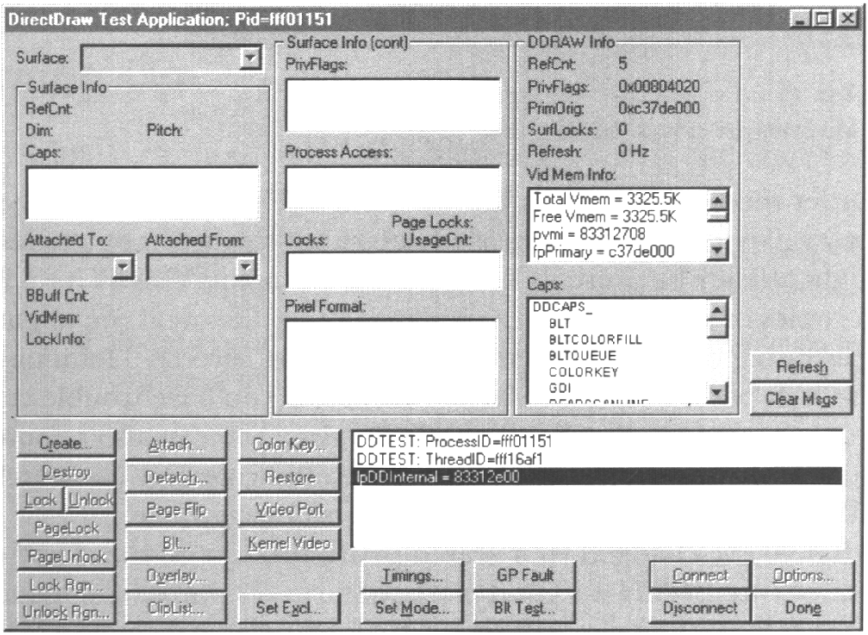

图 **3-7　DirectDraw** 测试应用程序

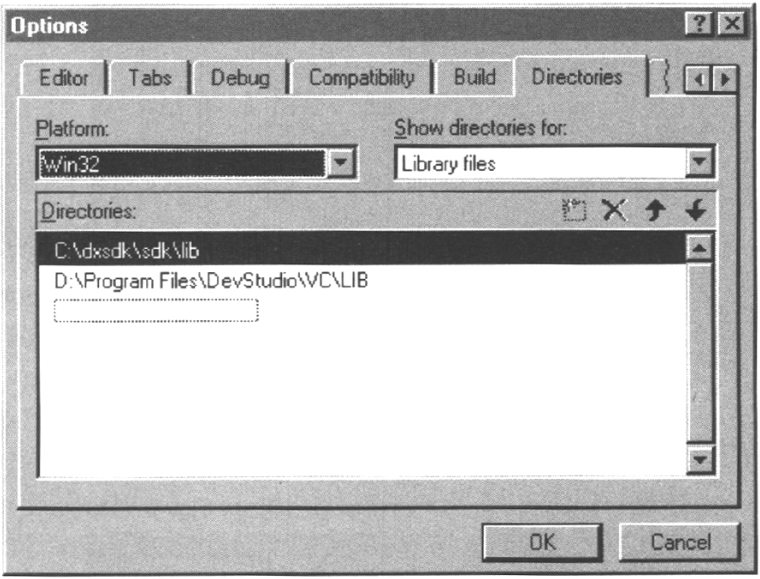

图 **3-8** 设置编译搜索路径

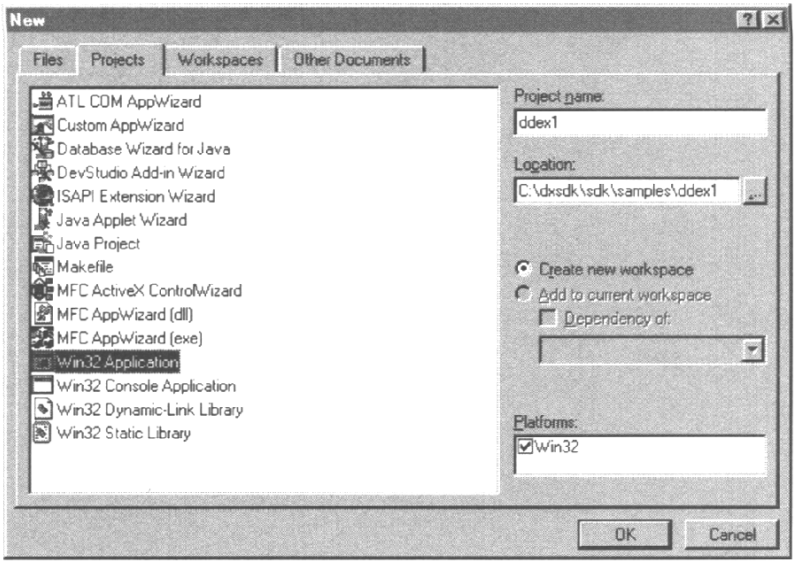

图 **3-9**　创建一个新的工程文件

最好能够象图 **3-8** 所示的那样把库文件和头文件排在前面，以免引起不必要的错误。举个例子来说，假如 **Microsoft Visual C++ 4.2** 安装了 **DirectX 3** 的头文件和库文件，这时如果在 **VC++5.0** 的搜索路径中正好包含 **DirectX 3** 的路径，那么将会出现编译与连接错误。如果你在编译 **SDK** 的例子时遇到错误，请首先检查这一选项。当然，要避免上述问题的出现，可以把引起混乱的头文件和库文件删除，或者可以把它们压缩到某个文件中。

在改变搜索路径之后，可以试着编写一两个例子。**DDEX1** 是一个很好的起点——它很简单，而且仅用到了 **DirectX** 中的 **DDRAW.LIB** 库。每个例子都包含了一个 **Makefile**。由于我们更喜欢用 **Microsoft Visual Studio** 来开发程序，所以不用例子中提供的 **Makefile**，而自己创建一个新的工程文件**(**见图 **3-9**)，然后再添加必要的 **DirectX** 源代码和库文件。

在 **DDEX1** 例子中，先创建一个 **Win32** 的工程文件，然后把 **DDEX1.CPP** 和 **DDEX1.RC** 文件加入到工程文件，并把 **DDRAW.LIB** 加入工程文件的库列表中，这时就可以准备编译执行 **DDEX1** 文件了。

3.6 总结

- 在安装 **DirectX** 5 时，选择完全安装方式，并且不要忘记选取 **Debug** 选项。
- 可以用 **FoxBear** 例子和 **DirectX** 设备查看器来检查 **DirectX** 是否已正确安装。
- 如果在编译例子时出错，请检查一下是否只装了部分 **DirectX** 库文件。