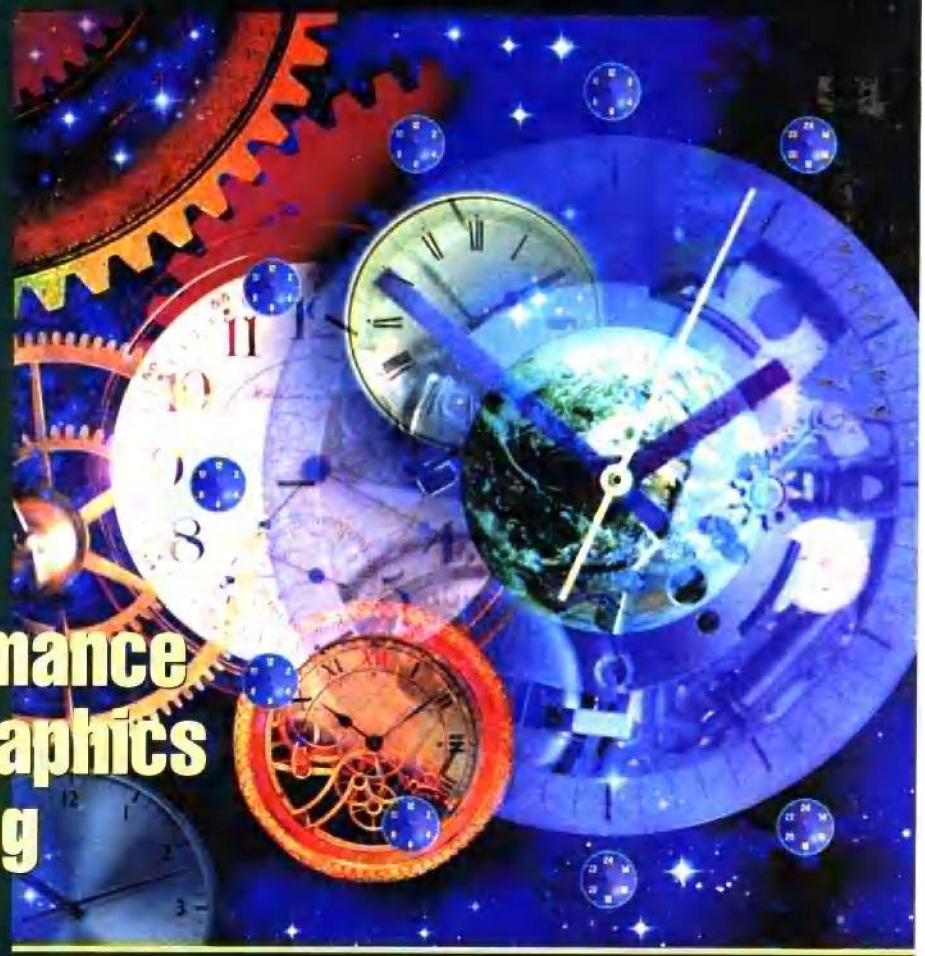
计算机软件开发与程序设计系列丛书

高性能Windows图形程序设计

（美） Stan Trujillo 著
李国岫 王磊 王恩惠 等译

附CD-ROM 1张



High Performance Windows Graphics Programming

机械工业出版社
CORIOLIS GROUP BOOKS
CMP

本书讲述了如何使用 DirectDraw 快速制作图形。主要介绍基本知识、显示模式、刷新率、表面和像素模式，并讨论了光标问题、视频回放等。书中列有大量应用程序。

本书可供从事图形软件开发的人员阅读。

Stan Trujillo : High Performance Windows Graphics Programming.

Authorized translation from the English language edition published by The Coriolis Group, Inc.

本书版权登记号：图字：01-98-0625

图书在版编目（CIP）数据

高性能 Windows 图形程序设计/（美）特鲁吉洛（Trujiuo,S.）著；李国岫等译.－北京：机械工业出版社，1998

（计算机软件开发与程序设计系列丛书）

书名原文：High Performance Windows Graphics Programming

ISBN 7-111-06339-2

Ⅰ.高… Ⅱ.①特… ②李… Ⅲ.图形软件,DirectDraw-程序设计 Ⅳ.TP391.4

中国版本图书馆 CIP 数据核字（98）第 12052 号

出 版 人：马九荣（北京市百万庄大街 22 号 邮政编码 100037）

责任编辑：温莉芳 李 红

北京忠信诚胶印厂·新华书店北京发行所发行

1998 年 6 第 1 版第 1 次印刷

787mm×1092mm 1/16 · 18.5 印张

印数：0001-5000 册

定价：53.00 元 (含光盘)

译 者 序

DirectDraw 是一个功能强大并且灵活好用的 API，它可以用来编写任何类型的 Windows 图形程序。DirectDraw 的主要用途是尽可能快、尽可能可靠、尽可能连续地将图像拷贝到视频显示设备上。也可以把 DirectDraw 作为一个视频内存管理器。

本书主要描述 DirectDraw 更高级的功能和性能。书中首先简要介绍了 DirectDraw 的基本知识，包括一些关键的术语和概念；随后重点论述了一些既有趣又有用的各色俱全的主题，包括 DirectDraw 深层次的内容、显示模式和刷新率、表面和像素格式、DirectInput 的有关内容、光标问题、视频回放，以及处理碰撞检测的有关问题；最后还介绍了开发者应注意的事项。

参加本书翻译工作的人员有李国岫、王磊、王恩惠、董明杰、李连源、王志东、李东升、王常红、戴薇、林雪聪、陈莉华、王宇、吕永中、吴剑峰、王勇等。DirectDraw 的内容比较新，而且国内有关资料极少，给翻译工作带来很多困难，由于时间紧迫、译者水平有限，译文中难免有不准确之处，恳请同行和读者批评指正。

注：英文原书附带 CD-ROM 一张，若读者有兴趣购买，可与华章公司联系。电话(010)68318309。

CD-ROM 上的内容:

为《高性能 Windows 图形程序设计》准备的 CD-ROM 包括下列内容:

- 为书中出现的所有程序提供预编译项目工作空间。
- BmpView,一个支持所有显示模式和滚动大量图像的 DirectDraw BMP 浏览器。
- Aviplay，一个 DirectDraw AVI 播放器。
- Switch，一个允许任何被检测的显示模式激活和测试的示例。
- 用户可以使用在 Visual C++5.0 内部的一个定制 DirectDraw AppWizard。
- DirectX 5.0 运行期安装。

备有一个安装程序，可将上述各项内容拷贝到用户准备的硬盘上。

运行要求:

软件: Windows95 或 Windows NT4.0
Visual C++5
DirectX SDK

硬件: Pentium75 或更高档次

译 者

1998 年 2 月 16 日

前　言

1. 本书的目的

DirectDraw 已经流行了一段时间，这足以从一定程度上证明它的潜力，并且已经有一些相关的书出版了。当然，这些书的质量参差不齐。这些书大体上都是一些比较好的指导性的书，像绝大多数计算机图书一样，由那些在写书的时候还在学习 DirectDraw 的人在 3 个月之内就写出来了。这就导致绝大多数的这类书只能提供一些介绍性的材料。既然 DirectDraw 已经使用一段时间了，并且很有基础（至少要比其他一些 DirectX 的软件出现的时间长一些），那就到了该脱离开基本的东西，去看一看 DirectDraw 的更高级功能和性能的时候了。

本书从其他相关书籍所遗漏下来的内容开始写起。我们将讨论 DirectDraw 的基本知识，但相当简洁。那些有编程经验但却初次面对 DirectDraw 的读者应该能够迅速入门。随后我们将转到那些既有趣又有用的各色俱全的主题上去。

本书的目的是教你如何使用 DirectDraw，而并不只是提供一个框架或定制 API 处理好你所有的工作。示例使用 C++和 MFC 编写，但并不隐藏细节。C++和 MFC 是非常好的工具，因为它们允许以任何方式编写应用程序。本书中用以编写代码的策略是一个经过良好组织的，又易于读懂的项目，向你说明发生了什么和为什么那样。

你会发现许多示例像使用 DirectDraw 一样使用 DirectInput。DirectInput 在 Windows 图形编程中并非是必需的，但是你也一样可以使用它。它要比交替使用快得多，而且它和 DirectX 捆绑在一起，因此你使用时不必需要一个额外的 SDK。

2. 读者须知

本书将教给你想要知道的 DirectDraw 的所有内容。然而本书不能再教你去学 C、C++、MFC 或 Visual's Developer Studio。你最好对上述内容都比较熟悉。当然你不必是专家。例如，你将使用 MFC，但仅仅是扩展到它如何帮助我们编写应用程序。因而在读本书之前也不必成为一个 MFC 专家。

3. 软件要求

你可以在 Windows NT 4.0 或在 Windows95 环境下使用本书。你也需要 Visual C++5.0 或更高版本。

你将需要拷贝 DirectX 3a 或更高的版本（最好是 DirectX 5）。记住，DirectX 包括两部分：运行期部分和 SDK。SDK 可以在下述网址自由下载：www.microsoft.com/msdownload。

4. 硬件要求

你的计算机应该是 Pentium 或者是更高档次的。在 Windows NT 环境下，需要 32MB 的 RAM，在 Windows95 环境下，至少需要 16MB 的 RAM。通常最好还需要一个 CD-ROM 驱动器。

最后，你将需要一个 DirectDraw 支持的视频卡（在这一点上，显示硬件的每一个先进部件 DirectDraw 都支持）。让我们开始吧，我们将以 DirectDraw 的应急教程开始。

目　录

第 1 章　DirectDraw 应急教程

只用一章的篇幅来概括 DirectDraw 的内容是不可能的，毕竟整本书都是根据 DirectDraw 这个题目来写的。DirectDraw 是一个功能强大并且灵活好用的 API，可以用来编写任何类型的 Windows 图形应用程序。其灵活性使得它更难以概括，因此试图在开始只用一个章节来解释每一件事情是愚蠢的。

然而我决定试一试。

让我以告知本章不准备做的事情作为开始吧。无疑，你或许也听说过有关 DirectDraw 的一些知识，你也看见过用 DirectDraw 编写的示例或游戏。因此我将与你共同分享有关 Windows 图形未来的一段长篇演说。因为一个编写得好的 DirectDraw 应用程序本身就是最具有说服力的，所以我就不再多罗嗦了。

我们也将浏览有关 HALs（Hardware Abstraction Layers）、HELs（Hardware Emulation Layers），以及那些你在某些 DirectDraw 书中看到的 SDK 帮助文件中的复杂流程图。你想读这本书是因为你想用 DirectDraw 来编写程序，而不是去编写 DirectDraw 设备驱动程序或研究 Microsoft 的内部结构。

在本章，我们将从一个软件开发者的应用前景的角度来讨论 DirectDraw 的实用术语。首先将给 DirectDraw 一个定义，然后讨论 DirectDraw API，最后将讨论一些确实对 DirectDraw 的开发者有所影响的实际问题。

1.1　什么是 DirectDraw

DirectDraw 的一个有趣的描述来自于 FastGraph。FastGraph 是一个曾经流行过一段时间的图形软件包。

目前，FastGraph 的一个版本以提供 DirectDraw 支持的形式存在，然而却以一个定制 API 隐含在 DirectDraw API 中。FastGraph 的创建者和承办者 Ted 和 Diana Grubers 在他们的 Web 上放置了一个文件，以证明 FastGraph 是一个比 DirectDraw 更好的选择。

Grubers 的一个观点是 DirectDraw“只是一个 bltting 发动机”。这是相当准确的,但却太简化了。更准确地讲，DirectDraw 是一个可以提供软件仿真测试的独立于硬件设备的 bltting 发动机。DirectDraw 的主要用途是尽可能快、尽可能可靠并且尽可能连续地将图形拷贝到视频显示设备上。

另外一个定义 DirectDraw 的方式是把它作为一个视频存储器管理器，同常规的存储器管理器一样，DirectDraw 发放存储器信息包，跟踪每一个信息包的状态。信息包可以随意地创建、复制、修改或破坏，同时这些操作的细节被程序员隐含起来，这样讲是过于简单了。此外，DirectDraw 是能够使用系统 RAM 和视频 RAM 的。存储器管理器也经常被设计成和主要目标一样强健，而不只是追求性能。对于 DirectDraw，性能只是设计目标之一。

从技术角度讲，DirectDraw 是随同设备驱动器集合的便携式 API。DirectDraw 设计成完全

避开传统意义上的 Windows 图形机构（GDI，或称图形设备接口）。GDI 由于性能低而名声不好，所以 DirectDraw 的设备独立性在提供最佳性能方面是至关重要的。

1.2 术语和概念

记住上述这些有关 DirectDraw 的描述，再了解一些作为 DirectDraw 语言一部分的术语和概念。我们将以一些基本的但却是最基础的应用于图形的一般材料开始，然后再转到 DirectDraw 的特定材料上。

1.2.1 显示模式

显示模式是由允许将要显示的图形输出的显示硬件支持的可视配置。最常用的显示模式属性是分辨率。Windows 使用的显示模式的默认值是 640×480 的分辨率。这意味着，水平方向有 640 个像素，垂直方向有 480 个像素。其他一些常见的显示模式分辨率有 800×600，1024×768。一些显示卡支持 Mode X 显示模式。一个典型的 Mode X 显示模式的分辨率为 320×200。

显示模式也随像素深度的变化而变化。像素深度决定着每一个像素所容纳的多少不同的值，因而也就可以显示多少种颜色。例如对于 8 位像素深度的显示模式，每个像素能够再现 256 种颜色的一种。像素深度为 16 位的显示模式支持 65536 种颜色，典型的像素深度为 8、16、24 和 32 位。

显示模式由安装在机器中的显示设备或视频卡支持。显示设备有自己的 RAM，从计算机的 RAM 中分离出来。我们把位于显示设备中的存储器称为显示 RAM，而把常规存储器称为系统 RAM。

支持一个给定的显示模式的 RAM 的容量取决于显示模式的分辨率和像素深度。例如，640×480×8(640×480 像素，深度为 8 位）的显示模式需要 307200 字节。1024×768×16 的显示模式需要 1572864 字节。支持显示模式的存储器必须是显示 RAM。一个给定显示设备所支持的显示模式因此也被可以利用的显示 RAM 的容量所限制。例如，1024×768×16 的显示模式因为需要一兆字节以上的内存，所以就不能被只有一兆字节 RAM 的显示设备所支持。

DirectDraw 的一个主要特征是显示模式切换 。这允许一个 DirectDraw 应用程序检测和激活所安装的显示设备所支持的任何显示模式。我们将在第 4 章中讨论显示模式切换的细节。

1.2.2 硬件加速

DirectDraw 具有最优性能的最重要的原因是，它尽可能地使用硬件加速来进行设计。硬件加速发生在当显示设备能够用建立在显示设备之中的处理功能执行操作时。硬件加速具有两个优点，首先，当硬件加速出现的时候，硬件按指定要求设计成支持图形操作，这提供了执行给定任务的最快的方法；其次，硬件加速使得计算机主处理器从执行操作中解放出来，这使得主处理器可以执行其他任务。

1.2.3 表面

表面是存储器的一个矩形部分的 DirectDraw 术语，通常包括图像数据。该存储器通常用

来表示一个存在于显示 RAM 或系统 RAM 中的表面。驻留在显示 RAM 中的表面享有超高性能，因为绝大多数显示硬件不能对系统 RAM 直接存取。

表面分为几大类，最简单的类型是脱离屏幕表面。脱离屏幕表面可以驻留在显示 RAM 中或系统 RAM 中，但却不能被显示。这类表面一般用于存储子画面和背景。

另一方面，一个主表面是可在屏幕上看到的视频 RAM 的一部分。所有的 DirectDraw 程序（可以提供视频输出）都拥有主表面。主表面必须驻留在显示 RAM 中。

主表面通常很复杂，或是可翻转的。可翻转表面允许页面翻转，这是一项整个表面的内容可以通过一个硬件操作而瞬时可见的技术。页面翻转用于许多基于 DirectDraw 或其他的图形应用程序中，因为它可以产生相当平滑、不闪烁的动画。一个可翻转的主表面实际上是两个表面，一个可见，另一个不可见。不可见的表面称为后备缓冲区。当发生页面翻转时，以前是后备缓冲区的表面就成为可见的，而以前可见的表面则成为后备缓冲区。

离屏表面和主表面都有两类：调色板的和无调色板的。在 DirectDraw 中，只有 8 位表面是调色板表面。调色板表面并不包含色彩数据，但是却引入一个色彩表。该表称为调色板。像素深度为 16、24 或 32 位的表面是无调色板表面。无调色板表面存储实际色彩值，而不引入调色板。

因为在无调色板表面中的每一个像素都存储色彩数据，所以知道表面的像素格式是很重要的。像素格式描述了存储于像素中的红色、绿色和蓝色（RGB）元件的方式。像素格式随像素深度、显示模式和硬件设计的不同而不同，在第 5 章中可以了解所有的像素格式。

1.2.4 Bltting

Bltting 是用于复制的图形语言。典型的 blt 操作是将离屏表面的内容拷贝到一个后备缓冲区中。当 Bltting 通过硬件完成的时候，执行速度相当快。如果无法得到硬件加速，DirectDraw 将使用一个软件操作来仿真 blt。这种仿真操作虽然也能够完成任务，但却比硬件慢得多。一般只有驻留在显示 RAM 中的表面能够通过使用显示硬件来完成 blt。

blt 操作调用一个源表面和一个目标表面，源表面的内容被拷贝到目标表面中。源表面中的内容在操作中不会改变，只有目标表面受 blt 的影响。blt 操作也并不需要使用全部的源表面或目标表面。源表面中的任何矩形区域可以被放置于目标表面中的任何位置。

不规则形状表面的 Bltting（例如典型的子画面）是以透明方式完成的。透明性是通过指定表面中某个不被 blt 操作拷贝的像素而获得的。像素值通过使用色彩键码给以标志。

色彩键码可以附加到源表面或目标表面上。源色彩键码是很普遍的。源色彩键码允许透明性，因为源表面中的像素值并未被拷贝。至于目标色彩，只有目标表面中通过色彩所指定的像素值能够被源表面的内容所覆盖。

DirectDraw 也支持一些特定的操作，包括拉伸、压缩、镜像映射，以及混合等。这些功能的实现往往取决于显示硬件。DirectDraw 能够仿真其中的某些操作，但是跟性能相比，价格往往是昂贵的。

DirectDraw 也有不能仿真的功能（例如目标色彩键码）。使用这些功能是冒险的，除非该功能为所安装的显示硬件支持，否则使用该功能的操作将失败。这给 DirectDraw 的开发者带来两种基本选择：要么放弃使用这些功能；要么往应用程序中增加定制软件。

1.2.5 调色板

使用 8 位显示模式的应用程序需要提供调色板。调色板就是任何时候都可以使用的色彩表。如果 8 位显示模式不需要调色板，应用程序将被迫使用 256 种颜色的固定设置。调色板允许用户定义将要使用的 256 种颜色之一。

当你使用调色板显示模式时，必须保证在应用程序中的图像也使用同一调色板。如果没有做到这一点，所显示的一些或全部图像中将出现错误的颜色。调色板也会带来麻烦，尤其是用一个调色板来显示大量图像的时候。调色板也有一些优势。正如前面提到的，调色板允许在一个有限色彩的场合使用最多的色彩。调色板也允许调色板动画。

调色板动画是动画通过改变调色板项目，而不是改变像素值来执行的技术，这就使得一个屏幕上的很多像素可以瞬时改变颜色。对于一些有限的应用程序，诸如分配的、重复的动画，调色板动画很有用处。

1.2.6 剪裁

理想状态下，一个 blt 操作就是整个表面被 blt 成为另一个表面。通常源表面被 blt 成为目标表面的边，或者目标表面被另一个表面或窗口遮蔽。像这样的情况就需要进行剪裁。剪裁只允许一部分或一个表面的一部分被 blt。

在编写窗口 DirectDraw 应用程序时经常用到剪裁，因为这些应用程序必须遵守 Windows 桌面的规则。我们将在本章后面讨论窗口应用程序。

DirectDraw 提供全矩形剪裁支持。也有这种情况，就是付费提供定制剪裁例程，我们将在第 3 章中研究定制剪裁解决方案。

1.2.7 其他表面

离屏表面和主表面（具有任选的后备缓冲区）是绝大多数 DirectDraw 应用程序的主干。然而一些其他的表面就有些不同，包括重叠表面、alpha 通道表面、Z-缓冲区以及 3D 设备表面等。

重叠表面是硬件单色画面，因而也就仅在支持重叠的显示硬件上获得。和软件单色画面不同，它可以被移动而不需要背景图像被恢复。

Alpha 通道表面用来执行 alpha 调配。Alpha 调配是透明的高级形式。允许表面以透明度或半透明方式来拷贝。alpha 通道表面可用来控制每一像素的透明度设置。Alpha 通道表面的深度有 1、2、4、8 位。1 位深度 alpha 通道表面仅支持两种透明设置，不透明（非透明）或不可见（全透明）。另一方面，8 位 alpha 通道表面允许 256 种不同的透明度设置。Alpha 调配是不被 DirectDraw 仿真的功能的一个例子。为了使用 alpha 调配，因而就需要有支持它的显示硬件或建立在应用程序之中的定制调配方案。

Z-缓冲区和 3D 设备表面用于 3D 应用程序中。这些类型的表面已被特别地加入到 DirectDraw 之中，以支持 Direct3D。Z-缓冲区用于景象绘制时期，以跟踪景象中离浏览者最近的对象，从而该对象可以在其他对象的前面出现。3D 设备表面可以用来作为 Direct3D 绘制目标的表面。本书并不包括 Z-缓冲区或 3D 设备。

1.3 Microsoft 的 COM 规格

DirectDraw 根据 Microsoft 的 COM(Component Object Model)规格得以实现。COM 设计成用来提供完全便携的、安全的、可升级的软件结构，COM 是一个大的项目，但是它并不是本书的一个项目。我们讨论 COM 只是为了方便使用 DirectDraw 进行编程。

COM 使用一个面向对象的模型，这要比像 C++等语言所用的模型更严格。例如，COM 对象经常通过成员函数进行存取，而且并不具备公共数据成员。COM 对继承性的支持与 C++相比也是有限的。

1.3.1 对象和接口的比较

COM 在对象和接口之间具有很大的区别。COM 对象提供实际函数性，而 COM 接口则提供存取函数性的方法。COM 对象不能被直接存取，相反，所有的存取都通过接口来完成。这条规则是如此强有力的得到遵守，以致于我们不能给任何 COM 对象以名称。我们只能给用来存取对象的接口名称。因为我们无法存取 COM 对象，所以我绝大多数时候是根据接口来讲的。

一个 COM 对象能够支持多个接口。这听起来像个特例，但是它经常出现，因为根据 COM 规格，一个 COM 接口一旦定义之后，就不能再被改变或增加。这样做是为保证旧程序在一个 COM 对象升级的时候不会被停止使用。这个初始接口始终存在，一个新的、替换的接口在提供存取对象的新的函数性的时候才被提供。

1.3.2 IUnknown 接口

所有的 COM 接口都是从 IUnknown 接口中衍生出来的。“I”标志经常用于命名 COM 接口（它代表 Interface）。DirectDraw 接口总以“I”开头。但是在文献中经常看不到这个标志。本书在提到接口时也将省略“I”标志。

IUnknown 接口将提供 3 个成员函数，所有 COM 接口因而继承这些函数：

- AddRef()
- Release()
- QueryInterface()

AddRef()和 Release()成员函数为称为生命期封装（lifetime encapsulation）的 COM 功能提供支持。生命期封装是一个将每一个对象根据它自己的结构放置的协议。

生命期封装通过引用值来实现。每一个对象拥有一个可以跟踪对象的指针数，或者引用的内部值。当对象创建之后，该值为 1。如果附加的接口或接口的指针被创建，则该值递增。与此类似，如果接口的指针被破坏，则该值递减。当它的引用数到 0 的时候，该对象自行破坏。

AddRef()函数用来使对象的内部引用值递增。绝大部分时间里，该函数通过 DirectDraw API 被用户调用。例如，当你使用 DirectDraw API 创建一个新的接口时，创建函数就自动调用 AddRef()。

Release()函数用来给对象的内部引用值递减。用户应该在接口指针将要超出范围时或者要通过使用接口指针来结束时使用这个函数。AddRef()和 Release()函数都返回一个值，表示对象

新的引用值。

QueryInterface()函数允许 COM 对象就它们是否支持特定接口进行查询。例如，升级的 COM 对象提供附加的接口，而非现有接口的修改版。QueryInterface()函数可以用来确定旧的接口，以决定新的接口是否被支持。如果被查询的对象不支持有问题的接口，则更替接口的指针就返回。

1.3.3 GUID

为了查询一个对象是否支持使用 QueryInterface()函数的指定接口，就有必要识别有问题的接口。这通过接口的 GUID(Globally Unique IDentifier)来实现。一个 GUID 是一个 128 位的值，也就是说，对于所有意图和目的是唯一的。所有 DirectDraw 接口的 GUIDs 都包含在 DirectX 头文件中。

上述对于 COM 的简单介绍，就是为有效使用 DirectDraw API 所需要的全部内容。以后当我们再讨论 DirectDraw API 时，你会发现这个情况是如何相关的。

1.4 DirectDraw API

衡量 API 的一个方法就是看它的大小。一个庞大复杂的 API 可能就是计划不周的结果。另一方面，一个庞大的 API 有时就意味着每一种情况都有可能出现。一个小的 API 就是一个新的、缺乏功能的软件包的证据。它也意味着，一个 API 只是做它所需要做的，而不能多做一点。

DirectDraw API 是比较小的，因此本章中所讨论的每一个函数不致于使本章看起来像一本参考手册。DirectDraw 提供很少的方便，也很少有限制。

DirectDraw 由 4 个 COM 对象构成，每个对象可以通过一个或多个接口存取。这些接口包括：

- DirectDraw
- DirectDraw2
- DirectDrawSurface
- DirectDrawSurface2
- DirectDrawSurface3
- DirectDrawPalette
- DirectDrawClipper

我们将讨论每一个接口，并随后讨论它们的成员函数。本节并不是一本参考手册。DirectX SDK 提供帮助文件，除去它的限制之外，还可以提供一个合适的参考章节，因此我们不用讨论每个函数的细节。相反，我们将讨论每个函数是干什么的，为什么这样使用，以及你有可能如何去使用它。

1.4.1 Directdraw 和 DirectDraw2 接口

当 DirectX 首次推出的时候（早先它被称作 Games SDK),DirectDraw 核心函数性以 DirectDraw 接口表示。后来，当 DirectX2 推出的时候，DirectDraw 也已经被升级了。DirectDraw 遵守 COM 规格而未被改变。新的函数性可以通过 DirectDraw2 接口存取。

特别需要注意的是，DirectDraw2 接口是 DirectDraw 接口的超级设置。DirectDraw2 接口可提供 DirectDraw 接口的所有函数，另外还增加了一些新的函数。如果你正在使用 DirectX 或更高版本，那么你可以随意选用 DirectDraw 接口或 DirectDraw2 接口。但是，由于 DirectDraw2 接口较 DirectDraw 接口的功能更强，所以没有必要使用 DirectDraw 接口。同样，Microsoft 并不主张使用这些无组织的、风格可变的接口。因此，在本书以后的程序中我们只使用 DirectDraw2 接口。

DirectDraw 和 DirectDraw2 接口提供的成员函数如下（按字母顺序排列）：

- Compact()
- CreateClipper()
- CreatePalette()
- CreateSurface()
- DuplicateSurface()
- EnumDisplayModes()
- EnumSurfaces()
- FlipToGDISurface()
- GetAvailableVidMem()
- GetCaps()
- GetDisplayMode()
- GetFourCCCodes()
- GetGDISurface()
- GetMonitorFrequency()
- GetScanLine()
- GetVerticalBlankStatus()
- RestoreDisplayMode()
- SetCooperativeLevel()
- SetDisplayMode()
- WaitForVerticalBlank()

接下来我们讨论 DirectDraw 接口函数。注意，在本章以后的内容中，DirectDraw 接口既表示 DirectDraw 接口，也表示 DirectDraw2 接口。只有在区分 DirectDraw 接口和 DirectDraw2 接口的函数时，才加以区别。

1. 接口创建函数

DirectDraw 接口表示 DirectDraw 本身。该接口在被用于创建其他 DirectDraw 接口实例时，是一个主接口。DirectDraw 接口提供三个这样的接口实例创建函数：

- CreateClipper()
- CreatePalette()
- CreateSurface()

CreateClipper()函数用于创建 DirectDrawClipper 接口实例。并非所有的 DirectDraw 应用程序都用到剪裁器，所以该函数并不是在所有的程序中都出现。我们将很快讨论 DirectDrawClipper 的细节。

CreatePalette()函数用于创建 DirectDrawPalette 接口实例。同 DirectDrawClipper 一样，并非所有的 DirectDraw 应用程序都用到调色板。比如，应用程序使用 16 位显示模式时，就不用调色板。但是，当应用程序使用 8 位显示模式时，就必须创建至少一个 DirectDrawPalette 实例。

CreateSurface()函数用于创建 DirectDrawSurface 接口实例。任何一个 DirectDraw 应用程序都要用表面来生成图像数据，因此经常要用到这一函数。

DirectDraw 接口自己的实例是由 DirectDrawCreate()函数创建的。DirectDrawCreate()是 DirectDraw 函数中少有的几个常规函数之一，但并不是 COM 接口成员函数。

2. GetCaps()函数

DirectDraw 接口允许准确确定软硬件都支持的特征。GetCaps()函数可以对两个 DDCAP 结构实例进行初始化。一个结构表明哪些特征由显示硬件直接支持，另一个结构表明哪些特征由软件仿真支持。最好是用 GetCaps()函数来决定你将用到的特征是否被支持。

提示：DirectX 浏览器

DirectX SDK 是与 DXVIEW 程序同时推出的。DXVIEW 说明了 DirectX 组件的功能，包括 DirectDraw。大多数系统中，有两个 DirectDraw 项目：主显示驱动器和硬件仿真层。第一项说明了显示硬件的功能。第二项说明了在缺乏硬件支持的情况下，DirectDraw 将要仿真的一些特征。在具有两个以上的 DirectDraw 支持的显示卡的计算机中，DXVIEW 会显示每一个显示卡的功能。

3. SetCooperativeLevel()函数

SetCooperativeLevel()函数用于指定应用程序所要求的对显示硬件的控制程度。比如，一个正常合作度意味着应用程序既改变不了当前显示模式，也不能指定整个系统调色板的内容。而一个专有的合作度允许显示模式切换，并能完全控制调色板。不管你决定使用哪种合作度，都必须调用 SetCooperativeLevel()函数。

4. 显示模式函数

DirectDraw 接口提供 4 种显示模式操作函数。它们是：

- EnumDisplayModes()
- GetDisplayMode()
- RestoreDisplayMode()
- SetDisplayMode()

EnumDisplayModes()函数可用于查询 DirectDraw 使用何种显示模式。通过设置 EnumDisplayModes()函数默认值可以得到所有的显示模式，而且可以通过显示模式描述消除那些不感兴趣的模式。进行显示模式切换的过程中最好使用 EnumDisplayModes()函数。现在市场上有各种各样的显示设备，每种显示设备都有自己的特征和局限。除了默认的 640×480×8 窗口显示模式，最好不要依靠任何给定的显示模式的支持。

GetDisplayMode()函数可以检索到有关当前显示模式的信息，并在 DDSURFACEDESC 结构实例中显示当前显示模式的宽度、高度、像素深度以及像素格式等信息。还有别的途径可以检索到同样的信息（比如检索主表面描述），因此该函数并不出现在所有的程序中。

SetDisplayMode()函数用于激活所支持的显示模式。SetDisplayMode()函数的 DirectDraw2 版本还允许设定显示模式的刷新率。而 DirectDraw 接口版本的 SetDisplayMode()函数只能进行

显示模式宽度、高度和像素深度的设置。任何一个要进行显示模式切换的程序都要用到SetDisplayMode()函数。

RestoreDisplayMode()函数用于存储调用 SetDisplayMode()函数之前的显示模式。SetDisplayMode()和 RestoreDisplayMode()函数都要求优先使用 SetCooperativeLevel()函数得到的专有合作存取。

5. 表面支持函数

除了 CreateSurface()函数之外，DirectDraw 接口还提供了以下几个表面相关函数：

- DuplicateSurface()
- EnumSurfaces()
- FlipToGDISurface()
- GetGDISurface()
- GetAvailableVidMem()
- Compact()

DuplicateSurface()函数用于拷贝当前表面。该函数只复制表面接口，不复制内存。被复制的表面与源表面共享内存，因此改变内存的内容就同时改变了两个表面的图像。

EnumSurfaces()函数可用于迭代所有满足指定标准的表面。如果没有指定标准，那么所有当前表面都被枚举。

FlipToGDISurface()函数的作用是在终止页面翻转应用程序前确保主表面得以正确存储。取消页面翻转时，有两个表面交替显示。这就是说，在终止应用程序之前有可能没有保存最初的可见表面。这种情况下，Windows 通过绘制一个不可见表面来恢复。利用FlipToGDISurface()函数就可以轻而易举地避免发生这种情况。

GetGDISurface()函数可以向只被 GDI 认可的表面返回一个指针。GDI 表面是 Windows 用于输出的表面。在进行屏幕捕捉时，这个函数非常有用，DirectDraw 可以捕捉到 Windows 桌面的任一部分。

GetAvailableVidMem()函数用于检索正在使用中的视频存储器（显示 RAM）的数量。这一函数由 DirectDraw2 接口提供，而不是由 DirectDraw 接口提供。该函数用于确定应用程序利用显示 RAM 可创建表面的数量。

Compact()函数不是通过 DirectX5 实现的，但它可以为视频存储器提供碎片整理技巧。在基于显示 RAM 的表面被不断创建或受到破坏的时候，可以释放大量内存。

6. 监视器刷新函数

DirectDraw 接口提供了 4 种适于计算机显示设备或监视器的函数，但这些函数不适于显示卡，它们是：

- GetMonitorFrequency()
- GetScanLine()
- GetVerticalBlankStatus()
- WaitForVerticalBlank()

这些函数尤其与监视器的刷新机制紧密相连。这在确保生成动画时尽可能不产生闪烁和图像撕裂现象时是至关重要的。但必须注意，并非所有的显示卡/监视器组合都支持这些函数。

GetMonitorFrequency()函数用于检索监视器当前的刷新率。刷新率通常用赫兹表示，缩写

为Hz。例如，60Hz的刷新率表示屏幕每秒更新60次。

GetScanLine()函数用于向监视器返回当前正在被刷新的扫描行（水平像素行）。不是所有的显示设备/监视器组合都支持该函数。如果这一功能得不到支持，该函数将返回DDERR_UNSUPPORTED。

对于高性能图形应用程序来说，通常要求利用垂直刷新同步地更新屏幕。尤其是，当显示器刚完成屏幕刷新时，最好能够更新主表面。否则，屏幕的一部分显示新的图像数据，而另一部分仍显示旧的图像数据，这种现象就是所谓的图像撕裂。DirectDraw 默认利用垂直刷新同步更新屏幕。如果不是这样还可以利用 GetVerticalBlankStatus()和 WaitForVerticalBlank()函数实现同步刷新。

7. GetFourCCCodes()函数

DirectDraw 接口提供的最后一个函数是 GetFourCCCodes()函数。该函数用于返回显示卡所支持的 FourCC 代码。FourCC 代码用于描述非 RGB 或 YUV 表面。我们不在此讨论 YOV 表面，它们已超出本书的范围。

1.4.2 DirectDrawSurface 接口

同 DirectDraw 接口一样，DirectDrawSurface 接口也遵守 COM 规格。最初，表面支持是由 DirectDrawSurface 接口提供的。DirectX2 介绍了 DirectDrawSurface2 接口的新的函数性，DirectX5 介绍了 DirectDrawSurface3 接口。

尽管本书中使用 DirectDraw2 接口，而不用 DirectDraw 接口，但我们仍忠于最初的 DirectDrawSurface 接口，因为 DirectDrawSurface2 和 DirectDrawSurface3 接口新增的函数并不十分重要。在以后的内容里，我们将用 DirectDrawSurface 接口来表示这 3 种接口，除非特别注明。

DirectDrawSurface 是最大的 DirectDraw 接口，它允许表面内容的拷贝、清除以及被调用程序直接存取。DirectDrawSurface 接口总共提供 36 个成员函数，按字母顺序排列如下：

- AddAttachedSurface()
- AddOverlayDirtyRect()
- Blt()
- BltBatch()
- BltFast()
- DeleteAttachedSurface()
- EnumAttachedSurfaces()
- EnumOverlayZOrders()
- Flip()
- GetAttachedSurface()
- GetBltstatus()
- GetCaps()
- GetClipper()
- GetColorKey()
- GetDC()