软件开发技术丛书

Visual C++ MFC 扩展编程实例

VC++ MFC Extensions by Example

（美）John E. Swanke 著　前导工作室 译

机械工业出版社
China Machine Press

软件开发技术丛书

Visual C++ MFC 扩展编程实例

（美）John E. Swanke 著

前导工作室 译

机械工业出版社
China Machine Press

本书主要介绍了运用Visual C++ 5.0或6.0的高级编程技巧，内容涉及MFC程序设计的最新概念，全书提供了大量VC的编程实例，旨在帮助读者较为全面地掌握VC编程知识、技巧和方法。

全书分为三个部分和附录。第一部分介绍Windows编程的基础知识，第二部分讲解用户界面编程技巧，最后一部分涉及Windows内部进程的一些实例。

本书思路清晰，实用性强，是计算机应用人员及大专院校师生不可多得的参考书。

本书版权登记号：图字：01-1999-2866

图书在版编目(CIP)数据

Visual C++ MFC扩展编程实例/(美) 斯文克(Swanke, J. E.)著；前导工作室译. -北京：机械工业出版社，2000. 1

(软件开发技术丛书)

书名原文：VC++ MFC Extensions by Example

ISBN 7-111-07714-8

Ⅰ.V… Ⅱ.①斯… ②前… Ⅲ.C语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（1999）第54643号

机械工业出版社 (北京市西城区百万庄大街22号 邮政编码 100037)

责任编辑：翟静华

北京忠信诚胶印厂印刷 · 新华书店北京发行所发行

2000年1月第1版 · 2000年3月第2次印刷

787mm × 1092mm 1/16 · 25.25印张

印数：6 001 - 9 000册

定价：53.00元（附光盘）

译 者 序

Windows 95风靡当今世界，是广大微机爱好者的首选操作系统，它功能强大、市场广阔。而对广大软件开发者来说，Microsoft公司推出的Visual C++又是对Windows 98进行应用程序开发的最好工具。能够得心应手地运用Visual C++进行开发是广大程序员的愿望，本书的目的就是要满足他们的愿望，使他们能够在较短的时间内较好地掌握运用MFC进行应用开发。本书作者通过自己在开发软件过程中所得到的经验与教训，总结出了通过实例进行学习是一种高效的方法。通过实例，可以使我们更加清楚地感受到一种语言或一类开发者们最初的意图，他们是怎样让用户使用他们的软件的。用户只有真正了解了一种语言的意图所在，使用它们时才不至于感到迷茫，才能真正达到随心所欲的地步。

Visual C++是编写Windows应用程序最富于挑战性和创造性的强大工具，但是大多数人在使用这一编程利器的时候不免在其庞大的体系构架和内容面前却步，这时如果有别人的实践指路，无疑会大大缩短掌握Visual C++的时间。利用MFC编写程序，再结合本书中大量的具体实例，可以使读者很快地掌握Windows程序的精妙所在。本书中的许多代码，可以直接引用到读者自己的应用程序中，它不啻是软件开发过程中的优秀指南。

本书提供了大量使用Microsoft的Visual C++和MFC的应用实例。每一个想开发基于Windows平台软件的开发人员都可以从这些实例中得到他们想要的精华部分。总之，它是一本计算机软件开发者和Visual C++爱好者必备的技术参考书。

译者忠心地感谢本书的作者John E. Swanke先生为我们提供了这么好的一本书，也为自己能够翻译这么好的一本书感到荣幸。

本书由彦文军、郭祥雷、龚纪元组织，前导工作室的全体同仁共同完成了翻译、录排等工作。

由于时间仓促，且译者的经验和水平有限，译文难免有不妥之处，恳请读者批评指正!

前导工作室

1999年7月

前　言

解决编程问题的最快捷方式莫过于参考实例，实例不仅应该具备可以运用于任何应用程序的一般性，而且要保证其结构便于扩充或者易于修改。此外，良好的示例代码不能与程序关联得过分紧密以至于开发者不知道实例从哪里开始，到哪里结束。本书将尽可能地选择一些良好的编程实例，并以适当的形式向有潜力成为MFC程序员的初学者提供实例代码以节约编程时间。

本书中出现的主题和编程实例实际上是以前一本名叫《Visual C++ MFC编程实例》[1]的书籍的延续。这本书本身是独立的，也同样很有用处。

本书的组织

本书的实例组织成多个章节，覆盖MFC应用程序的几个不同方面，内容涉及从界面编程到应用控制。各章节的组成部分如下所示:

基础

虽然本书是针对实例编写的，但是，在阅读实例之前了解一下相关的基础知识将更有利于理解本书内容。后续章节中往往会引用这一部分的内容，有基础的读者可以略过这一部分内容。

用户界面实例

该部分的实例涉及应用的用户界面。因为MFC特别强调用户界面，所以书中绝大多数的实例都出现在这一部分，包括菜单、工具栏、状态栏、视、对话框、各种控制条和控件窗口，还有一些创建窗口以及窗口绘图的基本实例等。这一部分的最后将介绍一些应用程序实例，其中包括文本编辑器和Wizard。

内部机制实例

该部分的实例都是多数MFC应用中正在使用的代码，他们代表了MFC中不可见的、非用户界面的部分。其中包括消息机制、通信和计时、发声和二进制字串等。

光盘的使用

本书附带的光盘中存有全部实例的Visual C++5.0和Visual C++6.0的工程文件。如果读者想找到对应本书特定实例的工程文件，仅需在光盘上的目录名中定位对应号码即可。

除非特别指出，光盘中的多数实例都是作为一个MDI应用而创建的，这个应用使用AppWizard的全部缺省值，其工程名为Wzd。

---

[1] 本书已由机械工业出版社引进并翻译出版——编者注。

关于SampleWizard

在光盘上还有一个SampleWizard工具，这个工具帮助读者将本书中的实例代码添加到读者自己的应用程序中去。它可以引导读者选取所需要的一系列实例，同时详细介绍这些实例以及有必要加入读者自己的工程的那些代码。SampleWizard还可以让读者用自己的工程名取代实例的工程名。

SampleWizard可以在光盘上的\SWD目录下找到。它利用了光盘上每个实例下的子目录\Wizard并包含了对应实例的特定细节。

仅需执行SW.EXE即可，其余的都很直观，读者具体应用就会明白。

原书名：《VC++ MFC Extensions by Example》
原书号：ISBN 0-87930-588-6
与原作者联系方式：jeswanke @ yahoo.com

目　录

第二部分 用户界面实例

第三部分 内部处理实例

第一部分 基 础

无论读者是否已经读过本系列的书籍，或者已经具备了多年的编程经验，我们仍将在这一部分回顾一下所需要的基本知识，其目的就在于能够使读者更好地理解本书的实例。编写程序常常是一种需要尝试不同方法以达到最终目的的工作。通常情况下，了解用MFC来做什么涉及到对4个基本概念的理解：Windows API怎样创建窗口；MFC如何封装并改进Windows API；MFC如何与窗口通信以及MFC是怎样控制绘图任务的。除了这些概念以外，本部分还将讨论一下工具栏和状态栏。最后我们将讨论一下MFC如何同非Windows构件进行通信，如串行口和Internet站点。

本部分包括的章节介绍如下。

第1章 概述

本章概述MFC如何封装并改进Windows API。如果读者已经阅读过本系列书籍的前一本，将会发现该章是对那些版本基础部分的一些回顾。本书包含这一章是为了保持本书对高层次读者的独立性。

第2章 控件条

本章将讨论MFC支持的控件条。标准的控件条包括工具栏、状态栏和伸缩条(Rebar)等。MFC增加了对话条和停靠栏。该章还要探讨MFC如何避免控件条之间以及它们和视之间互相覆盖的技术内幕。

第3章 通信

本章讨论应用程序与外部世界的不同通信方式。其中最基础的窗口消息将在第一章中讨论。本章还涉及其他一些通信途径，包括局域网、Internet通信、串行和并行端口、DDE、Windows挂钩和管道等。

第1章 概 述

本章将回顾Windows应用程序的基本知识，包括应用程序如何创建窗口、窗口之间如何进行对话以及如何在窗口内绘图。然后将讨论微软基础类库(MFC)以及Developer Studio怎样使创建窗口应用程序的工作变得容易起来。

1.1 Windows基础

当Windows操作系统启动应用程序时，它首先创建一个程序线程，该线程一般只是一个

可执行内存的管理模块，而这些内存则与系统中其他应用程序分享执行时间。如果这个应用程序要通过显示屏幕与用户交互，那么这个程序线程便需要负责创建显示在屏幕上的窗口。

程序线程通过调用操作系统的应用程序编程接口(API)来创建这些窗口。实际调用的函数是::CreateWindowEx()，这个函数需要下列参数：屏幕位置、窗口大小以及即将创建的窗口的风格。

1.1.1 窗口类结构

线程创建的多数窗口具有类似的风格(例如按钮)，这些类似的风格已经被集成为一个名为窗口类(Windows Class)的结构。注意这是一个结构，而不是一个C++类。在创建窗口时必须设定窗口类。也可以使用其他的窗口风格，并且分别设定各自的窗口类结构。

1.1.2 消息

如果用户单击了一个窗口，操作系统就会向这个窗口发送一个消息来把这一事件通知给它。每个窗口用自己的窗口处理过程来处理窗口消息，举个例子，一个按钮的窗口处理过程可能向它的主窗口发送一个消息告诉它需要做什么事情。

每个窗口的处理过程还负责在屏幕上绘制属于自己的窗口。操作系统在绘制窗口时会向目标窗口发送WM_PAINT消息。

所有类似的窗口具备同样的窗口处理过程，例如，所有的按钮控件使用同样的窗口处理过程，因此所有的按钮看起来外表都很类似，其行为也类似。这种情况下的窗口处理过程位于操作系统内。它的地址在窗口的窗口类结构内指定。所有的按钮控件都由同样的窗口类创建，这个窗口类结构的名字叫做BUTTON。

1.1.3 客户区和非客户区

窗口处理过程在屏幕上绘制一个窗口时实际上绘制了两个部分：客户区和非客户区。为了绘制非客户区(nonclient area)，窗口处理过程总是调用所有窗口过程都需要调用的相同的操作系统处理过程。该过程接下来还需要绘制框架、菜单条以及标题栏等等多数窗口共同具有的内容。过程所绘制的东西取决于窗口的风格。例如，由于按钮的风格被指定为不用绘制其非客户区，所以按钮窗口上就不会看见框架和菜单。

窗口的客户区(client area)总是由窗口自己的窗口处理过程绘制，也可能由操作系统来完成这件事，例如所有的按钮都由同样的处理过程来绘制，或者由创建者自己来绘制图像或列表。

1.1.4 重叠窗口、弹出窗口和子窗口

除了窗口类以外，还有成百上千种窗口风格供用户指定窗口的绘制及其行为。其中有3种最重要的风格创建了对应3种最基本的窗口类型：重叠窗口、弹出窗口和子窗口。

- 重叠窗口(overlapped window)，具有应用程序主窗口的全部特点。它的非客户区包括一个可伸缩的框架、菜单条、标题栏和最小化、最大化按钮。
- 弹出窗口(popup window)，具有消息框或者对话框的全部特点。它的非客户区包括一个固定大小的框架和一个标题栏。

■ 子窗口(child window)，具有类似按钮控件的全部特点。它没有非客户区，窗口的处理过程负责绘制窗口的每个部分。

这些窗口在其行为上表现不同，这将在以后讨论。

1.1.5 父窗口和宿主窗口

由于用户界面可能会由好多个窗口组成，所以由程序线程控制它们将是很困难的，例如，如果用户将一个应用程序最小化，那么程序线程应该对那些组成用户界面的所有最小化窗口负责吗？实际上并没有采取直接控制的方式，应用程序创建的每个窗口都通过调用::CreateWindowEx()分配了一个控制窗口。如果这个控制窗口被最小化，那么所有被其控制的窗口都会由操作系统自动地最小化。如果控制窗口被销毁，那么每个被控制窗口也随之被销毁。

每个被控窗口也可以作为其他窗口的控制窗口，结果最小化或者销毁某些窗口都只影响用户界面的一部分。无论是什么窗口或者位于何处，程序员都可以在它内部创建另外的窗口。

子窗口的控制窗口叫做父窗口(parent window)。父窗口剪切其子窗口，也就是子窗口不能在其父窗口以外绘制。当用户与类似按钮的子窗口交互的时候，这些子窗口将自动地生成消息并发送到其父窗口。这就使得控件可以在父窗口的窗口处理过程中被集中处理。

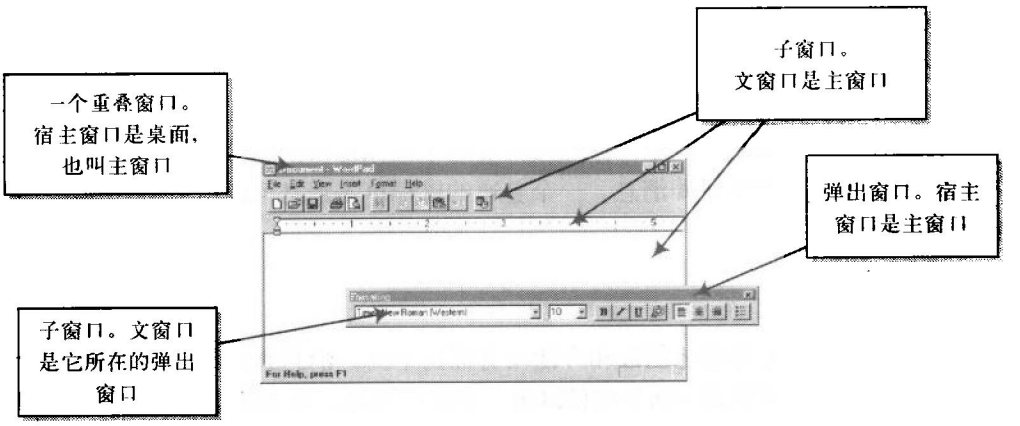

图1-1 构成一个Windows应用程序界面的窗口

弹出窗口和重叠窗口的控制窗口叫做宿主窗口(owner window)。与父窗口不同，宿主窗口并不限制属于它的窗口。然而，当最小化宿主窗口的时候，它的所属窗口也将被最小化。但是，当宿主窗口隐藏的时候，它的所属窗口却仍然显示。请参见图1-1以了解组成Windows应用程序的各种窗口。

1.2 Windows消息

上面提到，每个窗口由其自己的窗口处理过程响应来自操作系统或者其他窗口的消息。例如，用户用鼠标单击了某个窗口，那么操作系统可能就会向这个窗口发送一个消息。如果这是一个标识为Load File的按钮窗口，那么它的窗口处理过程就可能向应用的主窗口发送一个消息通知加载文件。主窗口处理过程将加载文件并在其客户区显示文件内容来作为响应。

接下来讨论消息是如何传送的。

1.2.1 发送或寄送消息

传送消息到窗口有两种方式：发送(send)或者寄送(post)。这两种方式之间的主要差别在于被寄送的消息不必立即处理。被寄送的消息放置于一个先入先出的队列里等待应用程序空闲的时候处理，而被发送的消息需要立即处理。实际上，发送消息到窗口处理过程和直接调用窗口处理过程两者之间几乎没有任何不同。只是，你可以要求操作系统截获所有为达到某个目的而在应用程序中被发送的消息，但不能截获对窗口处理过程的直接调用。

与用户输入相对应的消息(如鼠标单击和按下一个按键)通常都是以寄送的方式传送，以便于这些用户输入可以由运行较缓慢的系统进行缓冲处理。而其余的所有消息都是被发送的。在以上的例子中，系统寄送了鼠标单击消息，而按钮窗口则向其主窗口发送了Load File消息。

1.2.2 消息类型

有3种类型的消息：窗口消息、命令消息和控件通知消息：

■ 窗口消息(Window message)是由操作系统和控制其他窗口的窗口所使用的消息。这一类的消息有Create、Destroy和Move等等。上例中的鼠标单击消息也是一种窗口消息。

■ 命令消息(Command message)是一种特殊的窗口消息，它从一个窗口发送到另一个窗口以处理来自用户的请求。在以上例子中，从按钮窗口发送到主窗口的消息就是命令消息。

■ 控件通知消息(control notification) 是最后一种消息类型。它类似命令消息，当用户与控件窗口交互时，这一类消息就从控件窗口发送到其主窗口。但是，这种消息的目的并不在于处理用户命令，而是为了让主窗口能够改变控件，如加载并显示更多的数据。以上所述的例子中并没有控件通知消息，但是，假如按钮发送了鼠标单击消息给它的主窗口，那么这个消息也可以看作是一个控件通知消息。一个普通的鼠标单击消息可以由主窗口直接处理，然后由控件窗口处理。

1.2.3 接收消息

窗口处理过程看起来与其他函数和方法没有任何不同。消息到来后，按照消息类型排序进行处理。其中的参数则由调用函数提供以进一步区分消息。命令消息由wParam中的命令ID分类。DefWindowProc()函数则发送任何程序员都不会去处理的消息给操作系统。所有传送到窗口的消息都将通过这个函数，甚至包括绘制窗口非客户区的消息——尽管最终它们都将绕过DefWindowProc()函数。

一个主窗口的处理过程实例如下：

```
MainWndProc( HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam )
{
   switch( message )
   {
   case WM_CREATE:
      : : :
         break;
   case WM_PAINT:
      : : :
         break;
   case WM_COMMAND:
```

```
        switch (id)
        {
            case IDC_LOAD_FILE:
                : : :
                break;
        }
            break;
    default:
        return( DefWindowProc( hWnd, message, wParam, lParam ) );
    }
    return( NULL );
}
```

1.2.4 窗口处理函数的子类化

上面提到过，在窗口类中定义了窗口处理过程的地址。由窗口类所创建的窗口将把它们的消息传递给窗口处理过程。如果程序员使用的是一个由系统提供的窗口类，并要增加自己对窗口的特殊处理，就需要使用子类化(swbolassing)。

将窗口指向自己的窗口处理过程，便对窗口进行了子类化，这样所有的消息都可以由程序员自己处理了。如果只想处理一个或者两个消息，只需简单地将剩余消息传递给初始的窗口处理过程即可。我们注意到，采用子类化并没有修改原先的窗口类，而是直接对窗口对象进行了修改，这个对象保存了一份窗口处理过程地址的拷贝。

与此相反，超类化(superclassing)则修改了原始的窗口类，然后将其应用于创建窗口，但由于可以更方便、安全地使用MFC来获得由超类化带来的好处，所以这种方法就很少采用了。

请参见图1-2了解子类化概念。

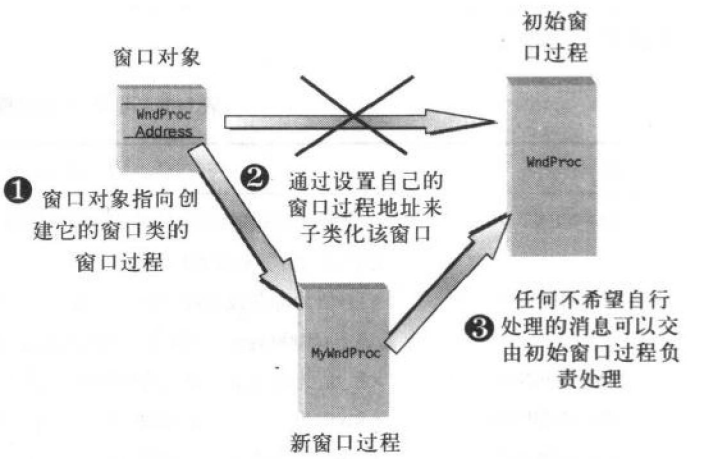

图1-2 窗口处理函数的子类化

1.3 窗口绘图

Windows API为窗口绘图提供了好几种调用函数。它们是点、弧、图形和圆绘图函数以及图形填充和位图绘制等函数。正是因为采用了绘图API，所以程序员必须负责传递坐标数值、颜色、宽度和绘图位置，而API函数则负责剩余的工作。为了简化对Windows API的图像函数调用，一些参数被固化到了一个名为设备环境的可重用对象中。

1.3.1 设备环境

图像设备环境(Device Context)是一个简单的对象，它包含了对绘图而言比较共同的属性，

例如绘图位置、线宽、填充模式的颜色等等。这个对象可以一次设置然后多次重复使用。实际上并不需要自己创建设备环境，只要调用几个可能的API函数，系统就可以返回已经被预先准备好的设备环境值。例如，系统为屏幕创建的设备环境包含屏幕上将用于绘图的颜色以及绘图工具的当前位置，程序员则可以在该位置进行绘图并设置颜色。

1.3.2 绘图工具

设备环境并没有包括绘图所需的全部特征。有几个特征存在于设备环境可以引用的附加图像对象中。这些对象都各自代表了某一特定绘图工具的特征(如：画笔的色彩和宽度、画刷采用的模式等)。这些工具包括绘制直线的画笔；用某种模式填充封闭区域的画刷；确定文本绘制效果的字体以及确定使用何种颜色的调色板等等。另外的两种绘图工具：位图和区域则显得更抽象一些。位图工具看起来像画刷，除了它只能用位图模式填充一个区域。而区域工具则类似于画笔工具，但它起的是剪切作用，例如可以使用区域工具从一个位图中将“STOP”单词分割出来。

1.3.3 映射模式

设备环境保持跟踪程序员采用的映射模式。设置映射模式可以指定调用参数中以英寸或者厘米作为度量单位的坐标x和y，每个绘图函数则可以自动确定绘制多少像素。可用的映射模式如下表所示。

表1-1　可用的映射模式

模　式	使 用 说 明
MM_TEXT	这是缺省的映射模式，坐标值x和y精确地等于一个屏幕像素或者一个打印机的打印点，y的正方向沿屏幕或者打印页向下
MM_HIENGLISH	x和y值为屏幕或者打印页上一英寸的1/1000。窗口决定当前屏幕设备应该有多少个像素等于1/1000英寸。Y的方向则为沿屏幕或者打印页向上
MM_LOENGLISH	x和y值为设备上一英寸的1/1000，y向上为正
MM_HIMETRIC	x和y值为设备上一毫米的1/100，y向上为正
MM_LOMETRIC	x和y值为设备上一毫米的1/10，y向上为正
MM_TWIPS	x和y值为设备上一英寸的1/1440，y向上为正。这个模式通常应用于绘制文本，一twip等于一个字体点的1/20
MM_ANISOTROPIC	程序员通过设置接下来将讨论的窗口和视口以决定x和y代表多少像素
MM_ISOTROPIC	同上，但x和y代表的像素数必须相等

1.3.4 窗口视和视口视

MM_ANISOTROPIC和MM_ISOTROPIC映射模式允许程序员自定义将坐标x和y换算为像素数的转换比率。这个工作由定义两种叫做视的矩形来完成。首先应该定义一个代表将要绘制的整个区域(例如：0，0，1000，1000)的矩形，然后定义一个代表那些最终出现绘图结果的屏幕或打印机上的对等坐标(例如：0，0，500，500)的矩形。第一个矩形叫做窗口视

(Window View)，第二个矩形叫做视口视(Viewport View)。如果在设备环境中定义了这两个矩形，并使用上述两种映射模式之一，那么使用窗口视坐标的图形将自动地被转换为使用视口视的坐标。除了视口视的坐标之外，程序员可以缩小、放大以及颠倒自己的图形，而不需要改变其他任何东西。

1.3.5 逻辑单位和设备单位

使用除MM_TEXT以外的绘图模式绘图的时候，传递给绘图函数的坐标采用逻辑单位(Logical Unit)。逻辑单位可以是英寸、厘米或者像素。绘图函数本身则用设备单位绘图。举一个例子，直线绘图函数可能使用254个像素代表一个1英寸的逻辑单位，这里的数字1就是逻辑单位数值，而数字254则是设备单位(Device Unit)数值。这并不会成为一个问题，除非打算让用户能够使用鼠标绘图。鼠标传回给应用程序的任何坐标都是以设备单位计量的数值，因此必须使用一些Windows API将这些坐标值转换为逻辑单位数值。

1.3.6 绘图函数

Windows API具有多个绘图函数，举例如下：

- 画点函数：如SetPixel()。
- 画线函数：如LineTo()、Arc()、Polyline()。
- 画图形函数：如Rectangle()、Polygon()、Ellipse()。
- 填充和图形反转函数：如FillRect()、InvertRect()、FillRgn()。
- 滚屏函数：ScrollDC()。
- 文本绘制函数：如TextOut()、DrawText()。
- 位图和图标绘制函数：如DrawIcon()、BitBlt()。

1.3.7 抖动和非抖动颜色

所有可用的绘图函数，包括从画线到画图形和文本，都需要使用颜色。但是，除非系统具备足够的显示内存，否则颜色将必须利用抖动(Dithered)方式。所谓抖动颜色实际上是一些主要颜色的集合，在显示的时候通过几个颜色相互混合以获得某种理想的色彩。

对大多数情况，抖动颜色已经足够了。然而，由于抖动颜色显得比较模糊，对图像应用程序而言通常不足以达到要求。图像应用程序不希望线条与其包围的图形颜色相互渗透混合。对图像应用程序可以有以下几种选择方案：

- 增加更多的显示内存。需要颜色抖动的理由在于，虽然屏幕上的每个像素都具有自己的RGB颜色。但一般的系统都没有足够的显示内存来为每个像素存储其RGB值。例如，设每个像素为32位，那么一个800 × 600像素的屏幕就需要2M字节的显示内存以容纳所有的颜色值。
- 只用标准颜色绘图。Windows使用的显示卡保证了至少能定义20种标准颜色，因为它需要使用这些颜色来创建抖动效果。
- 配置自己的颜色。除了标准颜色以外，显示卡还有一些空间可定义200多种颜色，可以在设备环境调色板内定义这些颜色并只用这些颜色绘图。多数图像应用都采用了这种方法，此方法将在实例31中得到详细描述。