微机新软件系列丛书

WINDOWS BITMAPPED GRAPHICS

Windows 图像处理实用技术和范例

Steve Rimmer 原著

木 杉 东 岳 翻译

任 天 希 望 审校

学 苑 出 版 社

1994 年·北京

(京)新登字 151 号

内 容 提 要

本书详细介绍了最常遇见的七种位映象图形格式(MacPaint,PCX,GIF,TIFF,WPG,Targa 和 BMP)以及 Windows ICO 格式,同时还提供了在 Windows 环境下读写、打印和操作这些格式文件的 C 函数源代码。本书内容丰富,资料新颖,是引导读者从事图像方面研究和开发的重要参考书。

需要此书的读者可与北京 8721 信箱联系。邮政编码:100080,**电话**:2562329。

版 权 声 明

微机新软件系列丛书

WINDOWS BITMAPPED GRAPHICS

Windows 图像处理实用技术和范例

原　　著:Steve Rimmer

翻　　译:木　杉　东　岳

审　　校:任　天　希　望

责任编辑:甄国宪

出版发行:学苑出版社　　**邮政编码**:100036

社　　址:北京市海淀区万寿路西街 11 号

印　　刷　北京市地矿局印刷厂印刷

开　　本:787×1092　1/16

印　　张:21　**字数**:486 千字

印　　数:1～5000 册

版　　次:1994 年 9 月第 1 版第 1 次

ISBN　7－5077－0976－0/TP・35

本册定价:34.00 元

学苑版图书印、装错误可随时退换

译　序

随着人机界面技术的不断发展，用户对界面的要求愈来愈高。除了要求交互手段简单、易理解、方便、自然外，还要求艺术性、活泼、生动。为此，我们用图象来代替传统的图符、箭头按钮等以实现这一点。目前使用最多的是位映象图形。

采用位映象图形便于编写能和其它软件交流的应用程序。但是，大多数常用图像文件格式(PCX，TIFF，GIF 等等)没有完整的文档说明，即使有也难于得到。为此急需一本能揭开这些图像格式秘密的书。本书就是为了满足用户这一需要而诞生的，它使得用户花很少的功夫即可掌握如何操作这些图像格式。本书多数为常用图像格式提供的示例代码以 C 语言写成。考虑到用户常会在 Windows 和 DOS 两种平台下工作，因此常需在这两种平台间进行代码转换。由于从 Windows 下的代码到 DOS 下的代码转换要容易得多，因此，本书中我们主要讨论 Windows 下的代码，用户可从中抽取到 DOS 下的代码。

本书提供了最常遇见的七种位映象图形格式以及 Windows ICO 格式的完整说明和源代码。这样将学会如何显示、打印和操作图像及 Windows 的适当资源。

本书由木杉和东岳翻译，任天和希望审校，此外参与翻译工作的还有王群山、刘伟、章东灵、吕东胜、章军、孙智伟、林利、晏文、高峰、刘源和李勇。

限于水平和时间仓促，书中若有不妥之处，恳请读者批评指正。

本书译文中多次出现随书磁盘的说明，但英文原版书并未提供随书磁盘，希望读者在阅读本书过程中加以注意。

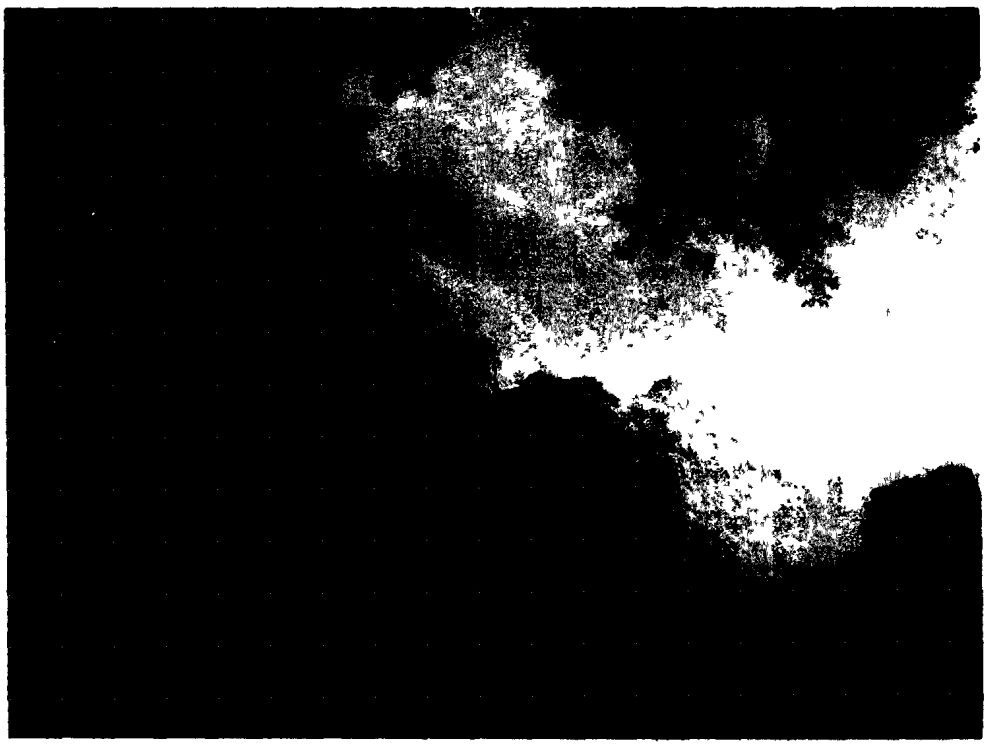
译　者

1994 年 9 月

目 录

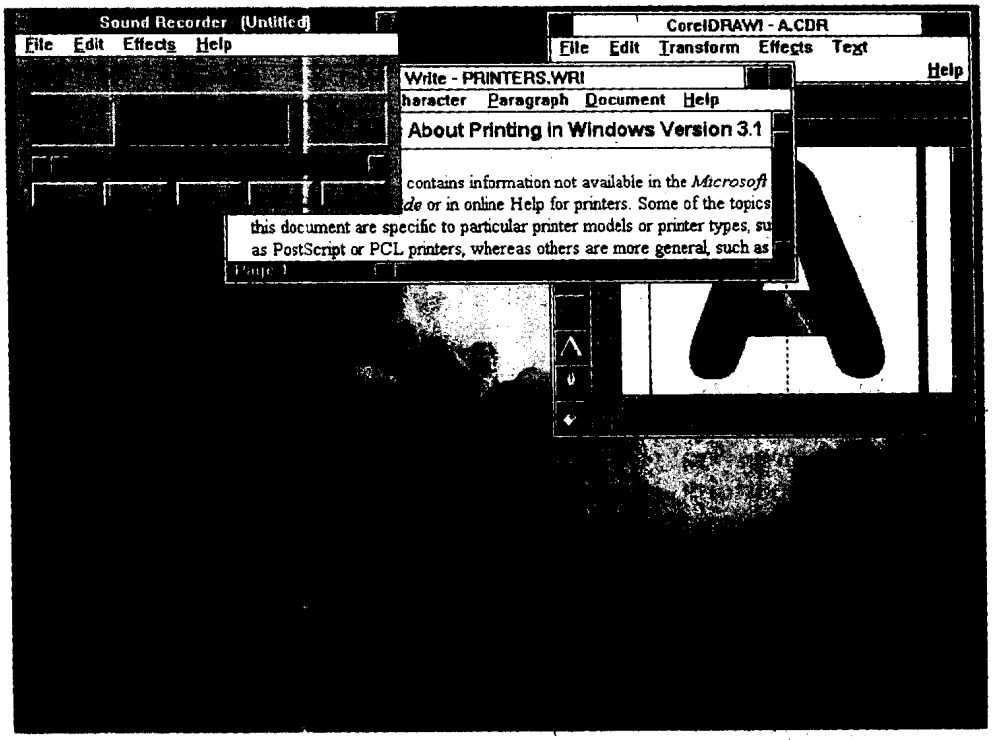
引　言

在我工作的城堡主楼的墙上挂着一些图画，但其中没有一幅能象 Maxfield Parrish 的《拂晓》(Day break) 更能唤起和计算机无关的一切。该画描绘了一个完全没有电话机、监视器和鼠标器的与时间无关的地方。它说明了这样一个世界，其中短语“艺术王国”表示该图画如何趋于完美。我这儿有几件 Parrish 的复制品。《拂晓》如图 I.1 所示，它以黑白形式显示。



I.1　油画《拂晓》，1922 年由 Maxfield Parrish 所做。在这里以屏幕象素表示

设计我的城堡主楼的不幸之一是放置具有大内存以运行 Windows 的 486，以致于使我坐在监视器旁时，却背对着墙上的大多数精美图画。对该局限考虑了一段时间后，我开始疑惑，将某种类似于《拂晓》的图画作为 Windows 墙纸会不会在某种程度上削弱 Windows 桌面的立体空间。结果如图 I.2 所示。



I. 2 拂晓作为 Windows 墙纸

在某种程度上，这个墙纸很奏效。它强烈地驱使我尽可能将所有窗口极小化。当在监视器上显示时只要没有窗口遮挡，该幅图画相当明亮，也很吸引人。

我现在不再把《拂晓》作为墙纸。除了将 486 系统变成一个博物馆外，看来其绘画不会赶上艺术家。于是我将一幅 J. W. Waterhouse 的《Hylas 和美女》挂在计算机旁，这会改善我的工作环境而不会使 Windows 或大家喜爱的艺术家遭受不愉快的事。《Hylas 和美女》没有在本书出现，但是若 Windows 系统需要一点生气，可以找到其复制品。

Windows 和 Windows 应用程序的图形特性会使得在软件中加入图形比用 DOS 软件实现更现实。除了为 Windows 桌面注入很强的艺术外，还可在自己编写的软件中使用复杂位映象图形。在应用程序中使用图像而不是图符或箭头会使其更易于理解，用户界面更友好。软件也许不会要求将一幅 Maxfield Parrish 的图画放入其 About 框中，但是会发现使用与典型 Windows 框和按钮不相象的图像会给用户提供一个驱动应用程序的全新方法。

从更传统的意义上讲，使用商用位映象图形允许编写能和其它软件交换图形的 Windows 应用程序。除了处理隐含于 Windows 中的输入和输出路径外(这些实用程序说明得很清楚，也很好用)，可用的位映象格式能让用户使用产生于其它环境中的图像，例如 DOS 和 Macintosh 产生的图像，同时还可以创建用于输出到这些替换平台的图形。

使用位映象图形的困难在于大多数流行商用图形文件格式（PCX，TIFF，GIF 等等）没有很好的文档说明，即使有文档说明也很难得到。这些文档实际上和 Windows 无关。

尽管从理论上讲，后一个问题不会很麻烦（大多数提供示例源代码的文件格式规范以 C 语言写成，这也是对 Windows 应用程序的更好选择)。但是，实际上，要在 DOS 和 Windows

格式间进行转换是极其复杂的。因为在弄清图形文件格式的时候得进行这种转换，从而通常用 Sanskrit 写成再用随机文本产生器翻译的文档会使将位映象图形结合进 Windows 应用程序相当困难。

本书谈到了在 Windows 软件中使用流行位映象图形文件格式问题。其中提供了最常遇见的七种位映象图形格式以及 Windows ICO 格式的完整说明和源代码。同时也提供了读写这些格式文件的紧凑、模块化 C 函数源代码。将学会如何显示、打印和操作图形图像以及如何使用 Windows 的适当资源。同时还包括了应用程序中应包含的位映象图形部分。

0.1　Graphic Workshop 连接

当有机会详读本书时，你就会意识到，商用位映象图形文件格式并不象其初次出现时那样有很好的文档说明。因为存在创建稍微有些不合法文件的应用程序，从而也就存在格式多样性。通常应用程序并不按其应该的方式读入输入文件。尽管可以通过忽略它们来很便利地处理这种情形，但频繁的错误信息对话框不会增加用户对你的 Windows 应用程序的满意。

本书中的源代码不在这里出现。它来源于 Graphic Workshop for Windows 图形软件包。Graphic Workshop 是最广泛应用的图像文件显示和操作包。其最初 DOS 版本已完成好几年了。它以及本书中的相应源代码能够以已知形式对所有棘手的、令人胆战心惊的、有些偏差的图象文件进行尝试。本书不仅为读者提供了在软件中易于理解和实现的源代码，同时还提供使用现实世界图形的很巧妙的函数。

也许本书提供的最有用的东西是时间。若应用程序在某种程度上和使用位映象图形相关，则会发现本书中的源代码可用极少或不用操作即可被抽出并放入你的应用程序。实际上，在大多数情况下，若你不想，则不必理解其功能。可通过使用编译器的 Edit 菜单中的 Copy 和 Paste 命令来处理应用程序的低层图形函数。并处理使应用程序运行的实际工作。

我希望你欣赏 Windows 位映象图形。无论你对 Windows 编程还很陌生而只是想使用这些确实很有趣的程序，或者在编写商用应用程序而需要把位映象图形引入自己的代码，本书有助于避免许多繁杂的工作。

第一章　面向 Windows 的位映象图形技术

尽管 Windows 在图形方面有明显考虑，但它所提供的支持位映象图形的固有工具却是非常有限的。它能绘画，但实际上不能涂画。在某种程度上这是一个缺憾。位映象图形学是一种涉及虚拟现实的计算机方法。线绘图画则是涉及 Windows 的一种计算机方法。

Windows 和许多运行在 Windows 环境下的应用程序接受大量的排列成份。它保持得如此机械和枯燥以致于地下 12 层都能起火。在许多方面，Windows 名不虚传。但在将来会有一天，对大多数 Windows 用户而言，那些小灰色按钮会开始变得象一个蹩脚的砖头。图 1.1 说明了这样一种可能的观察效果。

图 1.1 是一个由 Windows 应用程序来显示的位映象图形。它表明了关于 Windows 的几个重要方面。最重要的是只要你想方设法则 Windows 在视觉上可以是有趣的甚至是“离墙”的。同时也表明那些灰色小按钮还有其它方面的用处——若确实不很喜欢猫则会记住。

Windows 软件开发工具没有详细阐述如何将 Krazy Kat，身着皮革的自由女神或者 Victorian 画象加进 Windows 应用程序。尽管 Windows 能够让你这样做，但图像文件和软件之间的路径并不很明确。在某种程度上，这也就是本书要讨论的。

大多数 Windows 涉及的图形是矢量（vector）图形——它包括画线和填充对象。位映象图形在一些方面确实存在。许多 Windows 用户收集的 Windows 图符其实就是非常小的位映象图形。相对图像文件格式而言，图符是非常简单的。Windows 支持一种自己专用位映象格式（墙纸所采用的 BMP 格式）。尽管由于本书后面将要讨论，但对于大图像，它却不是一种很好的通用存储方法。

除了这些部分支持位映象图形，Windows 实际上并未提供使用图像的方便方法。特别地，它并未提供涉及位映象图形的资源，这些资源以实际中的应用程序广泛使用的格式存在。本书将帮助你跨过调用与 Windows 和硬盘上成千兆的 GIF、TIFF 和 PCX 文件信息之间的障碍。

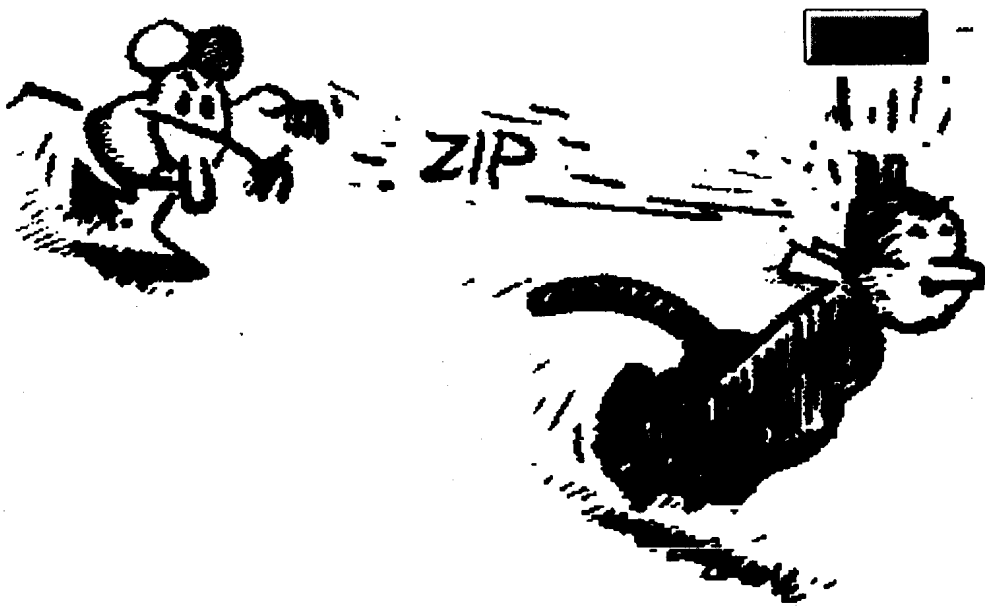


图 1.1 Windows 中的位映象图形，包括 Ignatz、Krazy Kat 和一个绝对按钮控制

特别地，本书详细叙述了如下位映象图形格式，这些格式用 Windows 代码来完成读和写。

- MacPaint
- PC Paintbrush PCX
- Compu Serve GIF
- Tagged image file format (TIFF)
- Word Perfect Graphics (WPG)
- Truevision Targa
- Windows BMP

另外，还讨论 Windows ICO 图符格式。

在 Windows 上位映象图形引起的最后一个问题是在 Windows 环境下处理大图像的复杂性。实际上，Windows 提供了一些合理可选的显示、操作、打印和管理位映象图形的调用，但这些用法很不清楚，而且其局限性并未被 Windows 处理好。这是本书要讲的第三个要点。它为你提供了在 Windows 环境下显示、打印和其它处理位映象图形的代码。图 1.2 给出了一个显示位映象图形的简单 Windows 应用程序——它显示一个 GIF 文件。

若要在观察图 1.2 与观察 Excel 简表之间选择（该简表概括了对巴西东部四分之三机构未来橡胶利润幅度进行的现金流动分析），大多数会宁愿看图 1.2。若不喜欢该应用程序显示的实际图像，当然可以替换成自己的。位映象图形的高明之一在于有许多位映象图形。

和图形学有关的应用以及那些打破图形学而使得对巴西东部橡胶前景的分析减轻一点枯燥的应用是有趣的，而创建这样的应用更有趣。本书中的代码能使你掌握在 Windows 应用程序中实现位映象图形的方法，从而能着手编制自己的软件。

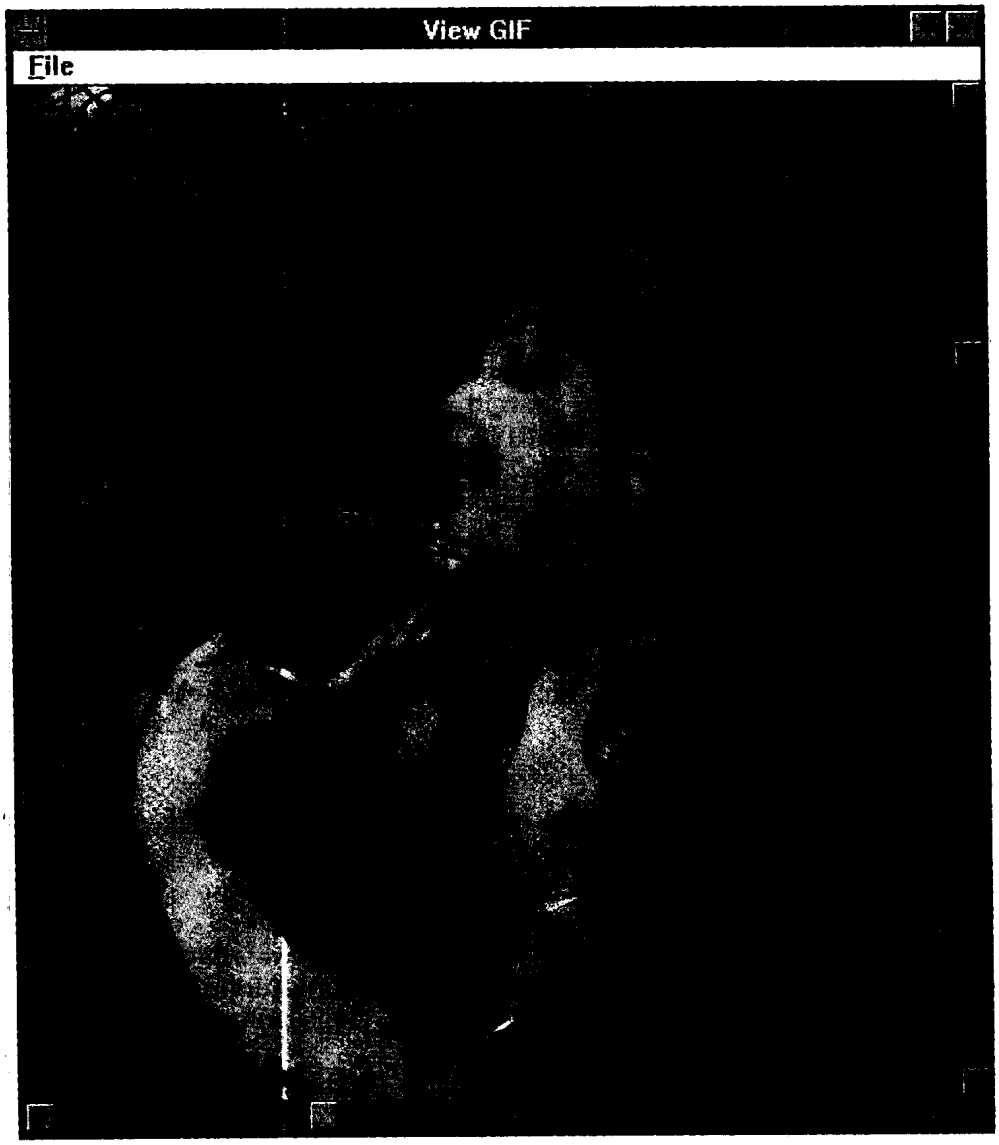


图 1.2　显示 GIF 文件的 Windows 应用程序。这是将在第五章详细讨论的 VIEW-GIF

1.1　技术层次

除非你是一个能以十六进制自然计数的人，否则编制 Windows 应用程序似乎有些不太令人愉快，无论在打开本书前你是否已编过这种程序。Windows 是面向个人计算机的最复杂的操作系统。它所提供的系统调用比编写 DOS 应用程序的系统调用多数十倍。在许多情况下，仅仅清楚地定义你要做什么从而知道从何处开始找一个完成它的系统调用也是很麻烦的。

本书介绍的既不是 C 语言编程也不是编写 Windows 应用程序。关于这些主题当然有一些好书，若在阅读本章时开始感到有些困难，则可能会想将本书搁置几天，在跳过其中的字节

和数据结构前查阅关于 C 或 Window 编程的书。

特别地，若以前从未写过 C 程序，则本书不会给你提供一个友好的开端。若先编写一些基于 DOS 的 C 程序，则会发现该过程最终会使使用本书中的代码更有效。

若融进图形学的思想会吸引你，则也许想学习来自 TAB Windcrest 关于本主题的我写的其它书——位映象图形学，第二版（#4266）；高级位映象图形学（#4012）；或者图形用户界面编程（#3875）。它们用基于 DOS 的程序来讨论图形的不同方面，但以语言复杂性降序排列。

若能用 C 熟练编程但却刚刚开始编 Windows 应用程序，并已熟悉对话框、消息以及设法使一个单任务处理器一次做 12 件事的操作系统的概念，则会发现一些与本书有关的书对你是有用的。Windows 软件开发工具所附带的文献在这方面就不错，若使用 Borland 编译器，则可另外再买 Microsoft Windows 程序员参考库。

由于取决于你打算编写的 Windows 应用程序的类型，用本书中的代码来学习基本技巧也许是更好的选择，而不是用许多对 Windows 编程进行一般介绍的例子。尽管是观点问题，在编写显示图形的软件时我会发现有比编写运算器或人造打字机更大的感召力。

也许值得指出的是，尽管位映象图形学是一个有些深奥的体系，但本书中的代码没有不鲜明的。特别地，没有涉及汇编语言，这样就不会陷入针对 Windows C 的偏差方面。

1.2 开发工具

C 编译器开发者喜欢引证的一件事是 C 所缺乏的明确性足以抵消其便利，因为用 C 语言编写的软件开初时不易理解且、几乎全是由标点组成的。这些 C 编译器开发者接着又创建基于 DOS 的通用 C 编译器，任何两个编译器均不包含相同方式的 8 行代码。

幸运的是，面向 Windows 的 C 编译器并非相互抵触。由一个编译器开发的代码大部分可以毫不费力地由别的编译器编译。当然，选择余地更少了。

本书中的代码是用 Borland Turbo C++ for Windows 写成的。Turbo C++ for Windows 在性能和可靠性方面进行了有趣的权衡。到写本书为至，这是可用于写 Windows 应用程序的唯一且本身也是一个 Windows 应用程序的编译器。它允许你不必费力地在 Windows 和 DOS 之间进行切换而能编辑、编译并运行 Windows 应用程序。Turbo C++ for Windows 还有许多缺限且常会崩溃。具体地说，根据 Windows 版本，在无法预测情况下，会产生无法补救的应用错误或保护模式错误。

Turbo C++ for Windows 是我遇到的第一个在后台闲置时会崩溃的 Windows 应用程序示例。在这方面它实际很少产生什么危害。它仅仅弹出一个对话框，无声死去，并允许你在决定用廉价降落伞飞行前处理正在做的事。

在读完本书后，Turbo C++ 中遇到的问题将可能会被解决。即使解决不了，若要使用本书中的代码，也许想考虑 Turbo C++ for Windows。即便该软件包常会崩溃，它也比 Windows 软件开发工具和 Microsoft C 用起来快。养成经常保存工作的习惯。

Turbo C++ for Windows 产生的代码不会继承其开发者遇到的问题——可以用面向 Turbo C++ for Windows 创建却实可运行的应用程序。图 1.3 表明了存在不可恢复的应用程序错误的 Turbo C++ for Windows。

若不想用 Turbo C++ for Windows，可以通过 Borland C++ 或 Microsoft C 来使用本书中的

代码。这两个软件包均是基于 DOS 的编译器。但须在 DOS 环境中编辑并编译应用程序然后切换到 Windows 来运行它们。

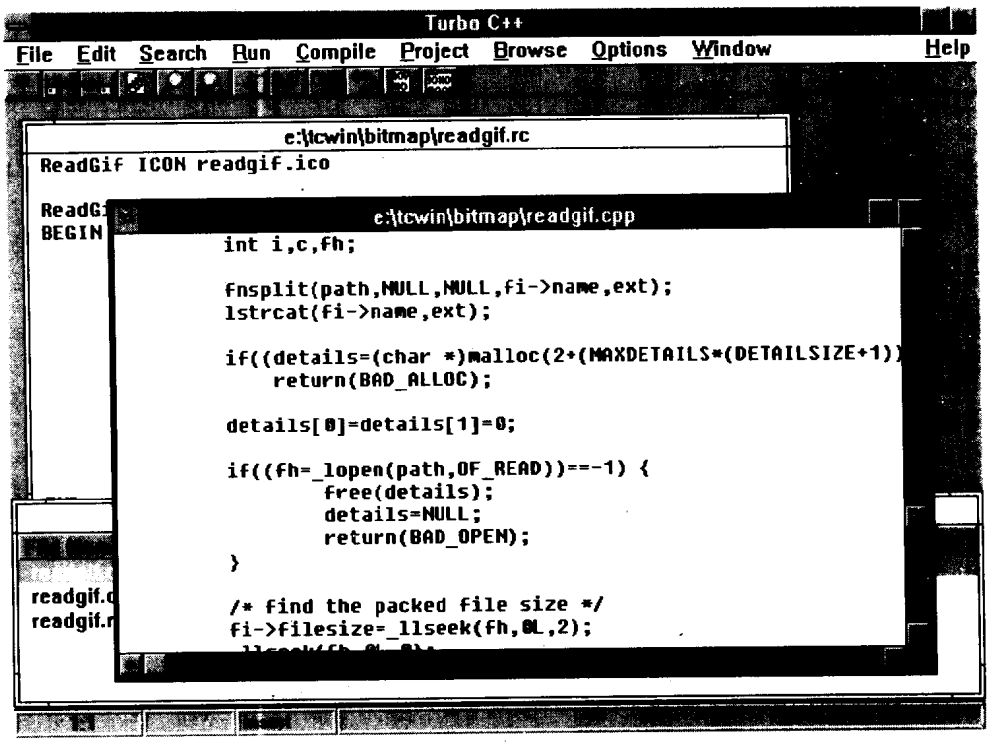


图 1.3　Borland's Turbo C++ for Windows

讨论如何创建这些软件包已超出本书的范围。当从框中产生后，这里所描述的源代码即可用某一编译器或多或少进行适当编译。但仍需记住几件事。

首先，可以用小型内存模式来编译本书中的程序。若要用其中的任何代码来编写更大更复杂的应用程序，将要使用中型内存模式。在 Windows 下用中型模式代替小型模式会涉及一点小小的麻烦。或许想用中型模式从而可以省去以后改变模式的潜在麻烦。

小型内存模式要求所有的 Windows 代码和局部数据容纳在同一个 64K 段内。这并不象看起来那么严格，因为 Windows 可以把部分小型模式的应用程序（象资源）留在磁盘上直到需要它们。对于任何能与其限制并存的应用程序而言这对速度会有一点改善，因为小型模式程序的寻址被作为 16 位指针来处理。

中型内存模式允许多个代码段。任何一个段不得包含超过 64K 的代码，但是可以有尽可能多的代码。在被要求时 Windows 会装入和丢弃段。

使用大型位映象图形的应用程序要占据的内存大而且要求处理器性能高。中型内存模式会对使用它的应用程序在速度和内存方面造成困难。但和展开一个中型 GIF 文件所占的内存和处理器相比，这些几乎是不重要的。

第二个方面是处理回调（Callback）函数，请查阅本书中 C 编译器处理代码的方式。能对编译器进行巧妙的回调操作。回调函数是应用程序中 Windows 调用的那些被告知的位。例如，

若创建一个对话框，还得创建一个 Windows 可调用的函数来处理针对该对话框的消息。该函数就是一个回调。当告诉 Windows 打开一个对话框时，会将适当的回调地址传给它，该回调被 Windows 非正式地称为一个 Procinst。若使用一个不支持回调的编译器，则要确保将这些函数说明为可输出的。

正象本章后边要讨论的，本书中代码的处理方式（编程风格）也许和写 C 程序喜欢用的方式极不相同。因此，在实现这里讨论的函数时，也许得对编译器缺省设置做些改动，以便能接受你的编程风格。

除了 Microsoft 兼容型 Windows C 编译器，还需要一种方法来编译资源。我把 Turbo C++ for Windows 附带的 Resource Workshop 应用程序用到本书示例上。可以使用任何已有的资源管理工具。正象使用产生 Windows 应用程序的 C 语言源代码，资源脚本在不同的 Windows 开发环境间是可移植的。

1.3 语言考虑

本书中的所有代码以标准 ANSI C 写成。这与经典的 Kernighan 和 Ritchie C 以及最近流行的 C++是有区别的。尽管选择这种语言的理由有几个，但也许会发现自己并不赞成。可以较容易地改变这些代码处理编译器的方式。

经典 C，即主要由 Kernighan 和 Ritche 定义的原始贝尔实验室 C 语言，是一个相当不错的程序设计环境。它幸存于 20 几年的技术升级和廉价 PC 系列，因为它达到了一种平衡，即和计算机的工作密切相关，从而能实际完成任务，又和字节及寄存器这样的客观实在充分分离从而避免自己成为一个编译器。

有一些在象 Windows 这样的较大操作环境下关于 C 的被证明是局限的方面。值得注意的是这是一个争论焦点，而你不必被迫赞成。若更喜欢以经典 C 编程，并认为所有随后的源于该语言的 ANSI 扩充仅对那些没有神经而要用廉价降落伞飞行的人有用，你可在编译器中进行适当切换并使用本书中的代码。

经典 C 的主要局限之一是将一切看成是一个整型，除非特别指明为别的类型。尽管背离这种思想是很容易的，但若不如此选择或忘记了则还会导致一个相当严重的缺陷。下面有几个这样的示例。

这是一个有些草率但总体上符合逻辑的 C 语言函数：

```
ThrowBrick (brick)
 {
/* 一些代码在此 */
}
```

在该例中，尽管未对对象 brick 的类型进行说明，但它被认为是一个整型。同样，函数 ThrowBrick 被认为返回一个整型。它也许实际上并不包含一个 return 语句，从而它所返回的任何值均是无意义的。C 和调用 ThrowBrick 的函数均不知道这一点。在经典 C 中，应该记住这一点。若错把 ThrowBrick 作为指针传送给参数串，可能的情况是无论 ThrowBrick 被编写成什么，均不能正确执行。

这儿有一个有这类问题的严重错误的示例：

```
ThrowBrick (brick, trajectory)
    Long brick;
    int (*trajectory) ();
{
    /*一些代码在此*/
}
```

在该版本的 ThrowBrick 函数中，第一个变量是一个长整型，第二个是一个函数指针（用 Windows 术语讲是一个回调）。庆幸的是，在经典 C 下，知道这些参数被如何组织的唯一对象是 ThrowBrick 本身。

大概在执行 ThrowBrick 函数中未指明代码过程中某个地方，由 trajectory 参数指示的函数将被调用。考虑若调用 ThrowBrick 的函数传给它一个整型而不是一个长整型作为第一个参数则会出现什么情况。ThrowBrick 函数并不知道这一点发生了。它从栈中读出前 4 个字节作为长整型参数。前两个字节实际是短整型值而跟随的两字节是 trajectory 指针的头两个字节。它然后从栈中读出 trajectory 或者至少它认为正在如此进行。实际上它从该值得到无意义的数据，该值在此后将会作为一个有效指针调用。

这就是在 Windows 下导致不可补救的应用程序错误的那种情况。你可通过告诉 C 的函数期望何种参数类型以及它们可被期望返回何种值来回避。在前述示例中，将下面一行放在源代码文件首部或者置于头文件的某个地方则会消除给 ThrowBrick 传送了错误参数而造成的恶果。

```
Void ThrowBrick (Long brick, int (*trajectory) ());
```

该行是一个原型，它准确表明了如何调用 ThrowBrick。在本例中，它告诉 C，ThromBrick 期望什么类型的参数。它也告诉 C，ThrowBrick 不会返回有意义的值；那正是 Void 定义所指出的。

有了函数原型，若试图编译一个在其中不正确地调用了原型函数的文件，则 C 会提示错误。这就有捕捉许多缺陷的潜力。

当从框中产生时，Windows C 编译器对允许的类型有极强的检查能力，尽管可能全部或部分关闭它，但最好别这样。在 DOS 下，除了在编译器库中出现的标准 C 函数外，大多数程序员会全部使用自己编写的代码。仅调用你的函数使得传送错误类型的代码几乎不可能。必须为自己编写的每一个函数建造原型是不方便的，在 DOS 编程环境下可能会觉得这样麻烦会更多。

在编写 Windows 应用程序时，须记住在对不是自己编写的代码（Windows 本身）进行大量调用，也许对此很不熟悉。将不适当的参数类型传给 Windows 是比较容易的，但常常导致不良结果。在这种环境中，值得使用 Windows 的原型和扩充类型检查特性。

在所有类型检查激活时，C 对于要求的函数原型和代码匹配是很教条的。在示例 ThrowBrick 中，不仅要求函数有原型定义，希望保证对于 trajectory 参数所指示函数的完整原型，而且要确保设法对该参数传递给 ThrowBrick 的函数地址确实和原型匹配。当然不允许在程序中使用没有原型的函数。

记住原型对大小写是敏感的。原型：

```
Void ThrowBrick (Long brick, int (*trajectory) ());
```