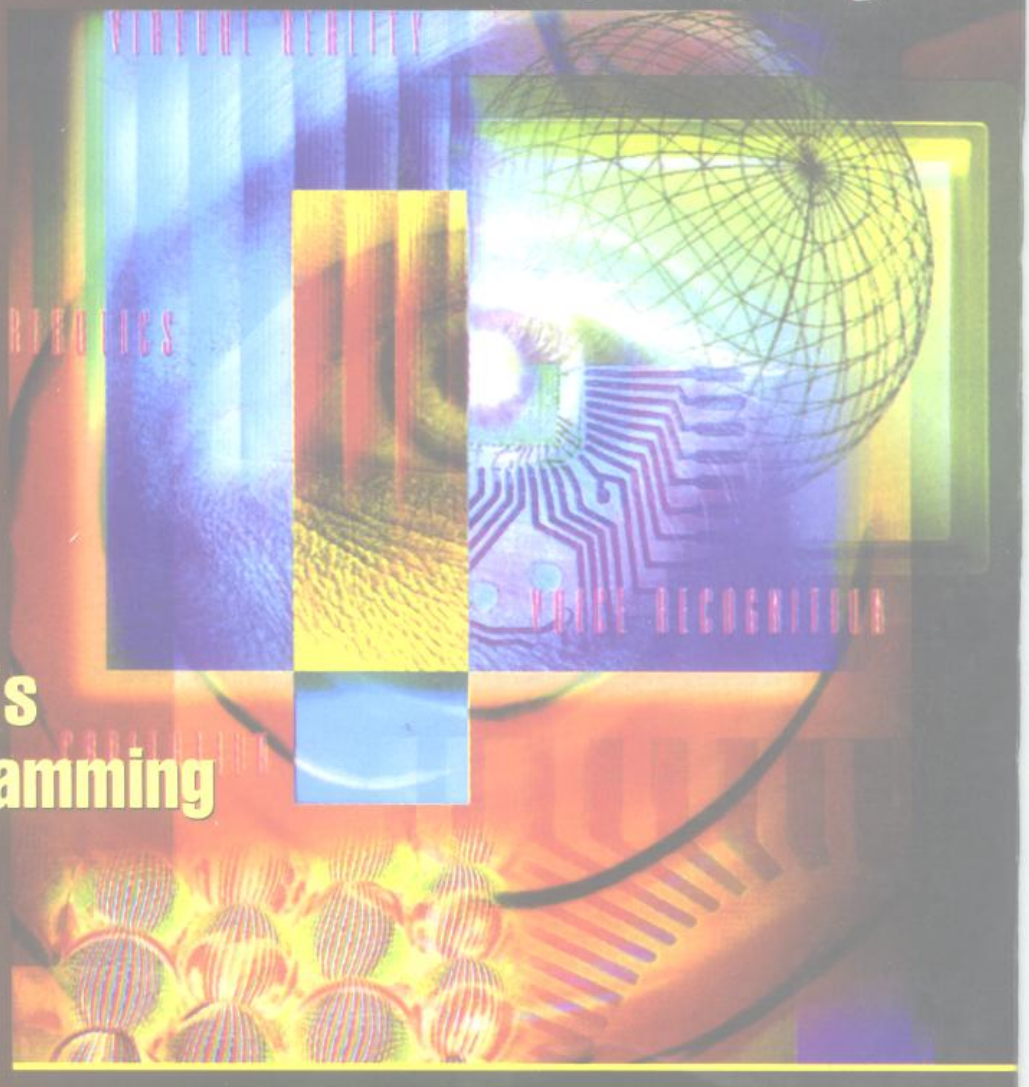
计算机软件开发与程序设计系列丛书

图形程序开发人员指南

（美） Michael Abrash 著
前导工作室 译



附 CD-ROM 赠

Michael Abrash's Graphics Programming Black Book Special Edition

机械工业出版社
CORIOLIS GROUP BOOKS
CMP

计算机软件开发与程序设计系列丛书

图形程序开发人员指南

（美） Michael Abrash 著

前导工作室 译

机械工业出版社

本书由浅入深、由高层到底层，系统、全面地介绍了高性能图形编程的各种知识和技能。作者首先详细地讨论了 x86 系列计算机的硬件及其优化特性、图形编程以及逐步优化的技巧，给出了每一种优化的分析过程、程序清单和性能对比。然后形成了一个基于模式 X 的通用动画图形软件包 X-Sharp。最后介绍了 Quake 的研究和实现技术。

本书分为两大部分，第一部分主要介绍汇编优化；第二部分主要介绍图形硬件(VGA)、高性能图形编程和优化。程序用 C 语言和汇编语言编写。

本书版权登记号：图字：01-98-0737

图书在版编目（CIP）数据

图形程序开发人员指南/（美）阿布拉什（Abrash, M.）著；前导工作室译．-北京：机械工业出版社，1998.8

（计算机软件开发与程序设计系列丛书）

书名原文：Michael Abrash's Graphics Programming Black Book Special Edition

ISBN 7-111-06396-1

Ⅰ.图… Ⅱ.①阿… ②前… Ⅲ.图形软件-程序设计-指南 Ⅳ.TP391.4-62

中国版本图书馆 CIP 数据核字（98）第 15309 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　于　静

北京昌平第二印刷厂印刷 · 新华书店北京发行所发行

1998 年 8 月第 1 版第 1 次印刷

787mm×1092mm 1/16 · 67.5 印张

印数：0001-5000 册

定价：128.00 元（附光盘）

译 者 序

我花了两个月时间读完 Michael Abrash 的这本《图形程序开发人员指南》。可以说，这是我读过的第一本集趣味性、实用性和专业性为一体的图形编程书籍。

作者采用循序渐进、循循善诱的方法，激发读者的创造性思维，培养综级编程（即全面的程序设计和创造性编程）能力。对每种优化，作者本着由浅入深的方法，层层深入地进行讨论。在本书中，性能优化的目标是高速度，而不是最短的代码。这一点也符合当今计算机的发展趋势——存储空间不再是瓶颈，速度才是瓶颈。因此本书的优化方法和程序设计方式很有意义。

本书最后形成了几个有用的工具。一个工具是测试程序（代码段）运行时间的计时器 ZTimer。这个计时工具的精度可以达到 10μs，可以在汇编程序和 C 程序中调用。另一个工具是模式 X，这是分辨率为 320×240 的 256 色图形模式，它具有许多其他模式所不具备的特性。如方形像素、允许页面翻转、允许 VGA 面向位面设计的硬件并行处理像素、每像素一字节等等。最后的一个工具是功能强大的动画图形软件包 X-Sharp，这个软件包基于模式 X，提供了高性能的像素绘制函数、线段绘制函数、多边形填充函数、模式设置函数、页面翻转函数等等。

本书与其他图形编程书籍最大的不同是，它不仅告诉读者怎么做（How），而且告诉读者为什么这样做（Why）。作者本着个人的智慧和劳动成果让每个人分享的精神，毫无保留地介绍了每一种优化的思路、不为人知的图形编程技巧，而且公开了 Quake 的许多技术内幕。

可以毫不夸张地说，掌握了这本书的精髓，就拥有了高性能图形编程的钥匙，就能够编写出 PC 上最优秀、性能最高的 3D 动画软件。

学习本书之前，读者最好对 PC 硬件、PC 汇编语言，以及图形学理论都基本了解。

本书由前导工作室的王无敌组织翻译。前导工作室的所有成员共同完成了本书的翻译、审校和录排工作。

最后祝愿读者在本书的学习中得到最大的收获。

译者

1998 年 3 月

前　言

我在学校读书的时候开始在 Apple Ⅱ计算机上编程，几乎所有早期的工作都基于 Apple 平台。毕业之后，我很快发现 80 年代后期在 Apple II 工作举步艰难，于是我不得不迅速转入 Intel 的 PC 环境。

过去我在 Apple 上花了好些年得到的知识，在 PC 机上不需要几个月就可以学到。

作为一名程序员取得收入，最大的好处就是能够购买所有想要的书和杂志。我买了许多。我处在一个一无所知的领域，于是我阅读在我身边所能找到的所有东西，时事报导、社论甚至广告都有我可以吸取的信息。

早些时候 John Romero 给了我一些 Michael Abrash 的文章，这些都是好文章：图形硬件、代码优化、雄心勃勃的开发者的知识和聪明才智，它们读起来更是轻松愉快。有很长一段时间，我到各个书店去寻找 Michael 的第一本书《Zen of Assembly Language》，但一直没有找到，只好将就利用我所能发掘到的他的文章。

从那些文章里，我学到了 EGA 视频控制不为人知的秘密，并且开发成几个精致的技巧，有几个成为 Commander Keen 系列游戏的基础。正是它们导致了 id 软件公司的产生。

一两年后，就是 Wolfenstein－3D 开发出来以后，我第一次“碰到”Michael，当时我正在 M&T 的 BBS 上在线漫游，这个 BBS 是 Dr. Dobb 出版商在 Internet 火爆起来之前构建的，这时我看见了他发出的邮件。我们交换了 email。有好几个月，我们在图形学论坛扮演着领袖二人组的角色，直到 Doom 游戏占据我的生活。

当 Doom 引起轰动以后，Michael 一个新得到工作的朋友让我们又彼此取得了联系，并且我终于有机会见到 Michael。那天我说得嗓子都哑了，我与 Michael 和他的合作者中感兴趣的人讲述着 Doom 的所有细节。从那以后，每隔几天我就能收到一封发自 Michael 的 email，或是征求我某个观点的细节，或是讨论将来图形学的发展方向。

最后，我征求他的意见——给他一份在 id 软件公司的工作：“只要思考；不要向任何人汇报；有机会整天从一张白纸开始编程。一个编程员做他该作的事情的机会”。但他未答应。但是我一直坚持，一年以后，我终于说服他屈驾来 id 看一看。当时我正在研制 Quake。

从 Doom 到 Quake 是我们迈出的极大的一步。我知道我该在哪里结束，但我一点也不知道如何到达那里。我试过很多很多的方法，即使是失败了也让我学到了许多东西。一定是我的热情感染了他，因为他接受了这项工作。

于是开始史诗般的编程。数十万行的代码编出来，重编，再重编……。

此刻回想，我感到 Quake 的许多方面都有遗憾之处，但那只是一个杰出的人不愿轻易地承认它技术上的成功而已。的确，一年以后，我可能会发现 Quake 中的某些地方有待改进。但现在对我来说，它看起来依然非常好。我们使它引人入胜。

我很高兴 Michael 在他正在发行的文章中描述了许多 Quake 的技术，我们掌握了许多，我希望我们能够教给别人一些。

当一个非编程人员听说我允许发行 Michael 的文章和源代码时，我总见到一幅目瞪口呆

的面孔——“你那样做想干什么?”。他们不会明白的。

编程不是一个零和的游戏，教给你的同行一些东西并不会丢失它们。我与人分享我能做到的一切时觉得很高兴，因为我深深地热爱编程。一辆法拉利小车不过是一种物质享受，但做人最重要的是真诚。

这本书包含了许多最初的文章，正是它们帮助我开始我的编程生涯。对于后面这些文章内容的出版，我希望我所做的努力能为其它人起到同样的铺路石的作用。

id 软件公司

John Carmack

与 John Carmack 一起研制 Quake 像什么？就像被捆在一个即将发射的火箭上，看起来好像全世界都在关注是否 id 软件公司能够超越 Doom，每一份偶然的 email 小报导或是与一个访问者的会谈，不出几小时就会在 Internet 上公布。此时，我们在 Quake 的技术上倾注了我们所知的一切。我经常在早上进入机房时发现 John 仍然在那儿，饶有兴致地研究一个新的设想，不试验成功他是决不会上床睡觉的。在研制快要结束时，我花了大量的时间来提高程序运行速度。我整天着迷地编写优化的汇编代码，蹒跚着走出办公室，步入热闹繁华的德克萨斯州。有时我从 LBJ 高速公路开车回家，车子开得飞快，风嗖嗖地从车身两边掠过，而不用担心超速行驶。在家里，我美美地睡上一觉，第二天又回到眼花缭乱的忙乱生活中，日复一日。事情瞬息万变，承受的压力又如此之大，有时我都想不出我们是如何完成它而没有被摧垮的。

当然，这一过程同时又非常激动人心。John 的设想无穷无尽又才华四溢。Quake 的实现建立了一个 Internet 的新标准，开发了领先的 3D 游戏技术。令人高兴的是，id 软件公司对于信息共享有一种开明的态度，它同意我发表有关 Quake 的技术，包括它如何工作、如何发展。我在 id 软件公司工作有两年了，我在《Dr. Dobb’s Sourcebook》中写了大量关于 Quake 的专栏文章，在 1997 年的计算机游戏开发会议上我又写了一个详尽的综述。读者可以在该书的后半部分找到这些文章，它们披露了鲜为人知的前沿软件开发的发展和内部工作机制。我希望读者能够乐于阅读它们，就像我乐于开发这些技术并撰写它们一样。

这本书的其余部分可以说是我十年来所写的所有关于图形和性能编程方面的程序，它们与现在的编程仍然紧密相关。并且它们覆盖了许多领域。该书包含了《Zen of Graphics Programming》（第二版）的大部分（其它的在 CD 上），以及全部《Zen of Code Optimization》，甚至我 1989 年写的《Zen of Assembly Language》，书中使用了古老的 8080 周期计数，但是有许多有用的观点。这些都在 CD 上。还有近 20000 字节关于 Quake 的资料；你还可以在一个紧凑的压缩文件里找到近十年来我学到的大部分东西。我很高兴能把所有这些材料一起复印，因为在过去的十年中，我总是碰上许多人发现我的文章对他们有帮助，还有更多的人想读一读它们，但是我却找不着。那些已出版的编程材料（特别是选自专栏的那些文章）很难长期保存。我得感谢 The Coriolis Group，特别是我的好朋友 Jeff Duntemann，帮助我找到这些材料（没有 Jeff Duntemann 不仅这本书不可能存在，我整个的写作生涯都不会开始）。

我还得感谢《Dr.Dobb’s》的编辑 Jon Erickson，他不仅给我鼓励，还为我提供一个栏目撰写任何有关实时 3D 编程的文章。即使是在 Quake 开发的时候，我仍然能够每两月写一

个专栏，这一点我自己都觉得很奇怪。如果不是 Jon 将一切安排得轻松舒适，这些都是不可能的。

我还得感谢计算机游戏开发者委员会（CGDC）的 Chris Hecker 和 Jennifer Pahlka，没有他们的鼓励、推荐和善意的指正，我不可能为 CGDC 写出这篇关于 Quake 技术最全面的综合评述。

除了在该书其他地方已经说过的话，我没有什么要说的了。该说的话在前面的介绍中、在多得令人吃惊的章节中都说了。正如读者在阅读中所见到的，这十年是微机程序员的十年，我庆幸我自己不仅能成为其中的一员，而且能够在历史的发展中留下自己的痕迹。

但愿未来的十年依然充满兴奋和喜悦。

Michael Abrash

目　录

第 1 章　大脑是最好的优化器

1.1　代码优化中人的因素

这本书了却了我最热切的心愿：编写好的软件以最大限度地发挥 PC 的性能。运行普通的软件时，PC 机速度慢得惊人。而如果运行精心挑选的软件，PC 可以产生惊人的高速。关键在于，只有在微机上，才可以随意地控制整个机器。没有操作系统或者驱动程序或其他类似的东西阻碍。可以实现任何想实现的功能，甚至可以弄清楚任何事情的来龙去脉。

下面我们立刻就可以看到，这一切的确都可以实现。

在这个拥有便宜的 486 和高速的 Pentium 的时代，性能仍然是关注的热点吗？有多少我们使用的程序，我们觉得它们已经足够快了，再快也没有什么意义呢？我们已经习惯于使用运行缓慢的软件。以致于在低档 PC 上花 2in 才能完成的编译和链接，在 486 上只需 10s 时，我们会欣喜万分。其实我们应该不满足于这些——除非它能做到瞬时响应。

读者认为不可能实现？如果没有精心的设计，包括增强的编译和链接、使用扩充/扩展内容、精心编制的代码，这一切是不可能的。但是，如果读者相信计算机可以做到，并且读者对计算机里里外外都了如指掌，并且愿意探索那些不同于传统的、创新的、更有潜力的方案，PC 可以做读者想象得到的任何事情（稍有例外，比如超级计算机级的数值处理）。我的观点是：PC 能够创造奇迹。让 PC 机做到这一点不是件容易的事情，但毫无疑问值得去努力。在本书中，我们将实现一些那样的奇迹。

1.2　何谓高性能

在编写高性能代码之前，我们必须理解什么是高性能代码。编写高性能软件的目标（不一定都能达到）是使软件能够快速执行指定的任务，实时响应速度达到用户需求的程度。换句话说，高性能代码的速度以如此理想的速度执行，以致对代码的任何改进都是无意义的。必须注意，以上的定义最强调的并不是使软件的速度尽可能快；也没有提到要使用汇编语言，或优化的编译器，甚至都没有谈到编译器；也没有涉及代码是如何设计和编写的。它所说的只是：高性能的代码不能妨碍用户。

这是个极其重要的区别。因为绝大多数程序员往往认为编写高性能代码的关键是汇编语言，或者好的编译器，或者某种特别高级的语言，或者某种设计方法。其实不是这样的，就好象建房子的关键并不是选择某种工具。要建房子，我们确实需要工具，但是那些工具中的任何一种都可以用来建房子。除了工具，我们还需要蓝图，需要了解组成房子的每一件东西；需要有使用这些工具的能力。

与此类似，高性能编程需要有清楚的认识：软件编写的目的、对程序的整体设计、实现特定任务的算法；需要了解计算机的功能和所有相关软件的功能；还需要扎实的编程技能，选择使用一种优化编译器还是汇编语言。而最后的优化过程只是一种辅助手段。

提示　如果没有好的设计、好的算法和对程序操作的全面理解，仔细优化代码其实

毫无用处——产生的只是一个快速运行的低效程序（fast slow program）。

什么是快速运行的低效程序呢？下面这个简洁而真实的故事可以回答这一问题。

欲速则不达

在20世纪70年代初，当第一台手工操作的计算器被推向市场的时候，我认识一个叫Irwin的人。他是一个好学生，他想成为一名工程师。作为一名工程师就意味着要知道怎样使用计算尺，Irwin操作计算尺是最好的。事实上，他是如此优秀，以致于他向一个使用计算器的人挑战并赢了他，因为这事他也成了当地的奇人。

当读者还在认真考虑这件事的时候，Irwin却已被抛在时代的后面。短短的几年中，他费尽心血钻研的计算尺技能将毫无用处，并且这个学科也将基本上趋于消亡。况且，任何稍有头脑的人都可以看出这个将要出现的根本的转变。Irwin浪费了大量的时间和精力用来优化他的转眼就会被摒弃的技能。

这个故事和编程有什么关系呢？有很大关系。如果读者花费时间企图优化设计得很糟糕的代码，或者读者指望一个优化编译器来加快这个代码的执行，就会像Irwin一样，纯粹在浪费优化。特别是在汇编中，如果不先经过精心设计每一部分，以组成高性能的程序设计，那么让一个本身运行起来很慢的程序尽可能快起来其实是在浪费大量的时间和精力——它仍然很慢。而如果读者预先思考一下程序，程序提高的性能就会多得多。我们将会明白，在编程的主要设计中，使用汇编程序和优化编译器并没有读者想象的重要。可以说它们毫无关系，除非应用在好的设计中并且我们对自己要实现的任务和PC机都有彻底的了解。

1.3 编写高性能代码的原则

要编写高性能代码，我们总结有以下原则：

- 明确目的（明白软件的目标）。
- 制定总体规划（头脑中对整体程序设计非常清楚，程序不同部分与数据结构很好地协调）。
- 制定大量的小计划（为整体设计的每一个独立部分设计一种算法）。
- 明确布局（确切地知道计算机怎样执行每一个任务）。
- 知道重要部分（标记程序中重要的部分，不浪费时间去优化其他部分）。
- 考虑多种方法（不死守一种方法，如果读者聪明并富有创造力，总会有更好的方法）。
- 精益求精（把读者认为确实重要的代码最优化）。

制定规划是容易的。难点在于怎样应用到实际编程中。就我的经验，剖析一些有效的代码是获得编程经验的好方法。下面让我们看看性能规则的实际运用。

1.3.1 明确目的

如果要编写高性能代码，首先必须明确代码要实现什么功能。举一个例子来说明。让我们编写这样一个程序，它求一个文件的字节和，产生一个16位的检验和（Checksum）。换句话说，它把某个文件中每一个字节都加起来，形成一个16位的值。当通过用调制解调器传输文件或者当Trojan horse病毒发生时，文件常常遭受破坏。使用这个方法就可以检查出文件是否被破坏。但是，我们唯一要做的是打印出检验和，并且我们感兴趣的是如何尽可能