



图形程序

开发人员 指南

(美) Michael Abrash 著
前导工作室 译



**Michael Abrash's
Graphics Programming
Black Book
Special Edition**



机械工业出版社

COROLIS
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

JS/KC/04

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。

Michael Abrash: Michael Abrash's Graphics Programming Black Book Special Edition.

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

Copyright 1997 by The Coriolis Group Inc.

All rights reserved.

本书中文简体字版由机械工业出版社出版，未经出版者书面许可，本书的任何部分不得以任何方式复制或抄袭。

版权所有，翻印必究

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)

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

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 $10\mu\text{s}$ ，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，这时我看见了发他的邮件。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。一个程序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

译者序	
前言	
第 1 章 大脑是最好的优化器	1
1.1 代码优化中人的因素	1
1.2 何谓高性能	1
1.3 编写高性能代码的原则	2
1.3.1 明确目的	2
1.3.2 制定总体规划	3
1.3.3 制定大量的小规划	3
1.3.4 明确布局	7
1.3.5 知道重要部分	8
1.3.6 考虑多种方法	9
1.3.7 知道应付意外	11
1.4 小结	14
第 2 章 汇编优化	15
2.1 汇编语言最优化的独特性	15
2.2 指令：个别与整体	15
2.3 汇编的特征	16
2.3.1 低效转化	16
2.3.2 独立性	17
2.3.3 学无止境	18
2.4 开动脑筋	18
第 3 章 综计时器	20
3.1 掌握和使用综计时	20
3.2 粗心的代价	20
3.3 综计时器	21
3.3.1 综计时器是一种手段不是目标	30
3.3.2 综计时器的开始	30
3.4 计时与 PC 机	31
3.5 停止综计时器	33
3.6 报告计时结果	33
3.7 关于综计时器的几点说明	34
3.8 综计时器使用举例	35
3.9 长周期综计时器	39
3.10 长周期综计时器的使用举例	54
3.11 在 C 中使用综计时器	58
3.12 仔细优化汇编程序	60
3.13 综计时器是一种好的工具	61
第 4 章 时钟周期耗费的本质	62
4.1 PC 硬件如何影响代码性能	62
4.2 时钟周期耗费的因素	62
4.3 时钟周期耗费的特点	63
4.4 8 位数据总线时钟周期耗费	64
4.4.1 8 位数据总线的限制	65
4.4.2 如何克服 8 位总线时钟周期耗费	66
4.5 指令预取队列时钟周期耗费	68
4.5.1 不要相信技术手册上的执行时间	69
4.5.2 指令执行时间是不定的	70
4.5.3 估计全面执行时间	74
4.5.4 如何克服指令队列的时钟周期 耗费	74
4.5.5 继续讨论 8088	75
4.6 DRAM 刷新	75
4.6.1 PC 中 DRAM 刷新的方式	76
4.6.2 DRAM 刷新的影响	77
4.6.3 如何克服 DRAM 刷新的时钟周期 耗费	78
4.7 等待状态	79
4.8 显示适配器时钟周期耗费	80
4.8.1 显示适配器时钟周期耗费的影响	82
4.8.2 如何克服显示适配器时钟周期 耗费	85
4.8.3 时钟周期耗费总结	86
4.8.4 结束语	86
第 5 章 使用可重用块进行优化	87
5.1 使用可重用块搜索文件	87
5.2 避开字符串	88
5.3 采取强制技术	88
5.4 使用 memchr ()	89
5.5 明确重点	94
5.6 跟踪执行	95
第 6 章 透过表面价值	96
6.1 机器指令无所不能	96
6.2 通过内存寻址进行数学运算	98

6.3 使用 LEA 指令倍乘, 无需使用 2 的幂次运算.....	100	11.4.1 系统等待状态	159
第 7 章 局部优化.....	101	11.4.2 数据对齐	161
7.1 介于算法一级和时钟周期计算一级的优化.....	101	11.5 代码对齐	163
7.2 不需要 LOOP	102	11.5.1 386 上的对齐方式	165
7.3 LOOP 指令与 JCXZ 指令的教训	102	11.5.2 对齐方式和栈	165
7.4 局部优化.....	103	11.5.3 DRAM 刷新的时钟周期耗费——不可改变	166
7.5 展开循环.....	105	11.5.4 显示适配器的时钟周期耗费	166
7.5.1 使用表进行循环移位和偏移.....	109	11.6 286 的新指令和新特性	167
7.5.2 关于 NOT 指令	110	11.7 386 的新指令和新特性	168
7.5.3 带/不带进位标志增加	110	11.7.1 286 和 386 的优化规则讨论之一	169
第 8 章 使用汇编语言加速 C 程序	112	11.7.2 286 和 386 的优化规则讨论之二	169
8.1 如果有助于加快速度, 不妨换一种语言.....	112	11.8 286 的 POPF 指令	171
8.2 尽量不在汇编中使用函数调用.....	113	第 12 章 486 的研究之一	176
8.3 栈帧 (Stack frame) 速度太慢	113	12.1 486 不仅仅是个快速的 386	176
8.4 多段处理问题.....	113	12.2 优化的规则	177
8.5 使速度到达极致.....	115	12.2.1 变址寻址的弊端	177
第 9 章 读者使我成功	128	12.2.2 提前计算指针	178
9.1 从另一个角度看 LEA 指令	128	12.3 给程序员的告诫	180
9.2 Kennedy 的工作	129	12.3.1 栈寻址以及地址流水线	181
9.3 加速乘法操作.....	131	12.3.2 使用字节寄存器的问题之一	182
9.4 不断优化搜索代码.....	131	12.3.3 使用字节寄存器的问题之二	183
9.5 快速排序.....	138	12.3.4 计算 486 程序运行的时间	184
9.6 全 32 位除法	139	12.4 补充说明	185
9.7 重新讨论最佳位置 (Sweet Spot)	142	第 13 章 486 的研究之二	186
9.8 核心时钟周期计算.....	143	13.1 流水线及损耗	186
9.9 硬件相关的远程跳转.....	144	13.2 BSWAP 比读者想象的更实用	188
9.10 设置 32 位寄存器: 时间因素和空间因素的权衡	145	13.3 存储器数据的压栈和弹栈	190
第 10 章 耐心编码, 使速度更快	146	13.4 最佳的 1 位平移和循环移位	191
10.1 草率编码导致性能低下	146	13.5 32 位寻址模式	192
10.2 笨拙的直接求解方法	147	第 14 章 Boyer-Moore 串搜索方法	194
10.3 递归	152	14.1 一个最佳搜索算法的实现	194
第 11 章 286 和 386 的研究	157	14.2 串搜索回顾	194
11.1 新的寄存器、新的指令、新的速度、新的复杂度	157	14.3 Boyer-Moore 算法	195
11.2 我们熟悉的 Intel 处理器.....	157	14.4 Boyer-Moore 算法的优缺点	198
11.3 286 和 386 的保护模式	158	14.5 Boyer-Moore 算法的进一步优化	206
11.4 时钟周期耗费的本质——讨论之二	158	第 15 章 链表	211
		15.1 链表前言	211
		15.2 链表	212
		15.3 哑元和标记	214

15.4	循环链表	216	20.2	流水线并行	298
15.5	24 个字节编程	221	20.3	V 流水线能执行的指令	300
第 16 章	代码的优化永无止境	223	20.4	执行的一致性	303
16.1	结果比手段重要	223	20.5	超标量的注意事项	307
16.2	快速统计单词数	223	第 21 章	充分利用 V 流水线	309
16.3	挑战和冒险	232	21.1	充分利用两条流水线	309
16.3.1	选择最好的算法	232	21.2	地址相关 (AGI)	309
16.3.2	不要想当然	233	21.3	寄存器冲突	312
16.4	巧妙优化的奇迹	233	21.4	确定指令运行所在的流水线	313
16.5	优化的级别	238	21.5	在运行中优化	314
16.6	二级优化: 新视角	241	第 22 章	综级优化	321
16.6.1	三级优化: 突破	242	22.1	简要回顾	321
16.6.2	小结	245	22.2	综级优化	321
第 17 章	游戏 Life 的优化	246	第 23 章	VGA 的硬件结构和特性	327
17.1	算法优化	246	23.1	标准 PC 图形的重点	327
17.2	Conway 的游戏	246	23.2	VGA	327
17.3	运行时间分布	252	23.3	介绍 VGA 编程	328
17.4	抽象的优缺点	253	23.4	VGA 内核	328
17.5	C++ 优化的重要性	259	23.4.1	线性位面和真 VGA 模式	330
17.6	创造性思维	262	23.4.2	平滑绘制	343
17.6.1	再次分析任务	262	23.4.3	颜色位面操作	345
17.6.2	相邻单元记录方法的实现	263	23.4.4	页面翻转	346
17.6.3	一生的挑战	270	23.5	VGA 的不兼容性	348
第 18 章	奇妙的 Life	271	23.6	一切还刚刚开始	348
18.1	优化的限制条件	271	23.7	关于宏汇编	348
18.2	打破限制条件	271	第 24 章	使用 VGA 进行并行处理	349
18.3	神奇的驱动表	272	24.1	一次从图形存储器中取出 4 个字节	349
18.4	使用变化表跟踪变化	287	24.2	VGA 的编程: 运用 ALU 和锁存器	349
18.5	外行对 QLIFE 的看法	289	24.3	ALU/锁存器演示程序注意事项	356
第 19 章	Pentium	291	第 25 章	VGA 中的数据控制部件	359
19.1	优化的历史	291	25.1	桶形移位器、位屏蔽和设置/复位 部件	359
19.2	优化作为艺术的回归	291	25.2	VGA 的数据循环移位	359
19.3	Pentium 综述	292	25.3	位屏蔽部件	360
19.3.1	跨越 cache 块取指	292	25.4	VGA 的设置/复位电路	367
19.3.2	cache 的组织方式	293	25.4.1	设置所有位面以形成一种颜色	370
19.4	更快地寻址	294	25.4.2	各位面独立操作	373
19.5	分枝预测	295	25.5	设置/复位部件注意事项	375
19.6	Pentium 的各种特点	296	25.6	双字节 OUT 指令注意事项	376
19.6.1	比较 486 与 Pentium 的优化	296	第 26 章	VGA 的写模式 3	377
19.6.2	超标量	297	26.1	关于写模式 3	377
第 20 章	Pentium 优化规则	298	26.2	陌生的写模式 3	377
20.1	在超标量体系结构中的优化	298			

26.3 保存寄存器位的注意事项	393	31.5 其他的思考	492
第 27 章 VGA 的写模式 2	394	第 32 章 360×480 256 色模式	493
27.1 写模式 2, 大位图块以及文本—图形 共存	394	32.1 使 256 色模式和标准 VGA 模式 一样快	493
27.2 写模式 2 和设置/复位部件	394	32.2 扩展的 256 色模式	493
27.2.1 写模式 2 处理举例	394	32.3 360×480 256 色模式	494
27.2.2 写模式 2 拷贝大位图块	396	32.4 360×480 256 色模式工作原理	504
27.2.3 写模式 2 绘制基于颜色模板的 线条	402	32.4.1 每屏 480 个扫描行	504
27.3 写模式 2 和设置/复位部件的权衡	409	32.4.2 每个扫描行 360 个像素	504
27.4 VGA 模式 13H 简介	409	32.4.3 在 360×480 256 色模式中访问 显存	505
27.5 文本模式/图形模式切换	410	第 33 章 调色板 RAM 及 DAC 寄 存器	507
第 28 章 读 VGA 显存	416	33.1 VGA 颜色产生的基础	507
28.1 读模式简介	416	33.2 VGA 颜色产生原理	507
28.2 读模式 0	416	33.2.1 调色板 RAM	508
28.3 读模式 1	422	33.2.2 DAC	509
28.4 读模式 1 特殊情况	426	33.2.3 颜色选择寄存器的颜色页面 调度	509
第 29 章 屏幕保存和其他 VGA 内幕 ..	431	33.2.4 256 色模式	510
29.1 VGA 操作综述	431	33.2.5 设置调色板 RAM	510
29.2 保存和恢复	431	33.2.6 DAC 设置	511
29.3 从 64 种颜色中选取 16 种颜色	439	33.3 不能调用 BIOS 时的处理	511
29.4 越界颜色	446	33.4 举例说明 DAC 的设置	512
29.5 一个好的清屏器	447	第 34 章 不通过写像素改变颜色	518
29.6 修改 VGA 寄存器	449	34.1 实时操作 DAC 颜色的特殊效果	518
第 30 章 拆分屏幕	451	34.2 颜色循环	518
30.1 拆分屏幕简介	451	34.3 核心问题	519
30.2 拆分屏幕的工作原理	451	34.3.1 通过 BIOS 装载 DAC	519
30.2.1 拆分屏幕的实现	452	34.3.2 直接装载 DAC	520
30.2.2 不要混淆两种拆分屏幕操作	461	34.4 颜色循环的测试程序	521
30.3 设置相关寄存器	461	34.5 有效的颜色循环方法	528
30.4 EGA 拆分屏幕的问题	462	34.6 补充说明	529
30.5 拆分屏幕和绘制	462	34.6.1 DAC 屏蔽	529
30.6 设置/读出寄存器注意事项	472	34.6.2 读 DAC	529
30.7 其他模式的拆分屏幕	474	34.6.3 结束语	530
30.8 拆分屏幕的安全性	474	第 35 章 Bresenham 线段绘制算法	531
第 31 章 VGA 更高的 256 色分辨率	475	35.1 Bresenham 线段绘制算法的实现与 优化	531
31.1 到底是 320×200 还是 320×400? ..	475	35.2 关于线段绘制	531
31.2 为什么是 320×200?	475	35.3 Bresenham 的线段绘制算法	532
31.3 320×400 的 256 色模式	476	35.4 使用 C 语言实现	535
31.3.1 320×400 模式中显存结构	476		
31.3.2 像素读、写	478		
31.4 两个 256 色页面	486		

35.4.1 分析 EVGALine 函数	541	绘制	636
35.4.2 绘制每一条线段	543	42.2 Wu 的反走样算法	636
35.4.3 绘制每一个像素	543	42.3 绘制以及亮度和为 1	638
35.5 C 语言实现程序的讨论	544	42.4 Wu 反走样算法的样本	641
35.6 汇编语言实现的 Bresenham 算法	545	第 43 章 位面动画	655
第 36 章 Bresenham 步距长度片算法	554	43.1 有限颜色的简单和快速动画方法	655
36.1 采用步距长度片方法的快速 Bresen-		43.2 基本概念——位面	656
ham 算法	554	43.3 位面动画的实现	659
36.2 步距长度片原理	555	43.4 位面动画的局限	671
36.3 步距长度片的实现	557	43.5 剪切和页面翻转	673
36.4 步距长度片的详细讨论	558	43.6 关于 PC 上的动画	673
第 37 章 步距长度片绘制算法的优化	567	第 44 章 使用拆分屏幕的页面翻转	675
37.1 对步距长度片绘制算法的大力优化	567	44.1 页面翻转的局限	675
37.2 快速步距长度片方法	567	44.2 计算机动画的几个问题	675
37.2.1 步距长度片方法的速度	575	44.3 一个页面翻转动画演示	675
37.2.2 进一步的优化	576	44.3.1 写模式 3	689
第 38 章 多边形原操作	577	44.3.2 文本绘制	691
38.1 高效快速地绘制多边形	577	44.3.3 页面翻转	691
38.2 填充的多边形	577	44.4 引入拆分屏幕	693
38.3 如何拼接多边形	579	第 45 章 “脏矩形”动画	695
38.4 填充不重叠的凸多边形	580	45.1 小狗 Sam 的故事	695
38.5 其它	589	45.2 狗的启迪	695
第 39 章 快速的凸多边形填充算法	590	45.3 VGA 访问	696
39.1 多边形的快速填充	590	45.4 “脏矩形”动画	697
39.2 快速凸多边形填充	591	45.5 “脏矩形”动画的实现	699
39.2.1 快速绘制	592	45.6 高分辨率的 VGA 页面翻转	705
39.2.2 快速边扫描	594	45.7 页面翻转的另一改进	709
39.3 最后的方法：汇编语言	597	第 46 章 使用屏蔽图像的“脏矩形”	
39.4 更快的边扫描方法	600	动画	711
第 40 章 复杂多边形的简化	605	46.1 优化“脏矩形”动画	711
40.1 处理不规则的多边形区域	605	46.2 “脏矩形”动画探讨之二	711
40.2 填充任意的多边形	605	46.3 屏蔽图像	723
40.3 复杂多边形填充的实现	614	46.4 内部的动画	723
40.3.1 关于活跃边的再讨论	618	46.5 绘制顺序和显示质量	724
40.3.2 关于性能的讨论	618	第 47 章 模式 X：神奇的 256 色	
40.4 非凸多边形	620	VGA 模式	725
第 41 章 多边形命名的重要性	622	47.1 VGA 未公布的“动画优化”模式	
41.1 把一个数据结构概念化时名称很		介绍	725
重要	622	47.2 模式 X 的特性	725
41.2 术语的使用	622	47.3 选择 320×240 256 色模式	726
第 42 章 Wu 的反走样算法	636	47.4 从模式 X 的角度进行设计	733
42.1 使用 Wu 的算法的反走样线快速		47.5 硬件加速	738

第 48 章 模式 X 对锁存器的使用	743	第 54 章 3D 浓淡处理	842
48.1 动画的最好视频显示模式	743	54.1 使 3D 动画对象具有真实表面	842
48.2 模式 X 中的显存分配	749	54.2 对以前处理器的支持	842
48.3 在显存中拷贝像素块	751	54.3 浓淡	861
48.4 小结	758	54.3.1 环境浓淡	862
第 49 章 模式 X 的 256 色动画	759	54.3.2 漫反射浓淡	862
49.1 如何充分发挥 VGA 的性能	759	54.4 浓淡的实现	865
49.2 屏蔽拷贝	759	第 55 章 256 色模式中的颜色表示	868
49.2.1 快速屏蔽拷贝	762	55.1 用 RGB 模式表示 X-Sharp 中的	
49.2.2 注意事项	769	颜色	868
49.3 动画	769	55.2 颜色模型	868
49.4 模式 X 动画的实现	769	55.3 从 Bit Man 那里得到收益	873
49.5 模式 X 印象	776	第 56 章 纹理映射	880
第 50 章 三维动画	778	56.1 使用快速纹理映射	880
50.1 使用模式 X 的 3D 动画	778	56.2 快速的脏纹理映射原理	881
50.2 3D 绘制的参考文献	779	56.2.1 使纹理映射更容易	881
50.3 3D 绘制流水线	779	56.2.2 DDA 纹理映射注意事项	883
50.3.1 投影	781	56.3 快速纹理映射方法的实现	884
50.3.2 平移	781	第 57 章 多边形纹理映射的优化	892
50.3.3 旋转	781	57.1 实现快速、光滑纹理映射的经验	
50.4 3D 编程举例	783	规则	892
50.5 小结	794	57.2 视觉效果	892
第 51 章 背面删除方法	795	57.3 定点算法的缺陷	893
51.1 使用背面删除方法消去不可见面	795	57.4 纹理映射：与方向无关	894
51.2 有一个可见面的多边形	795	57.5 把纹理映射到多边形	896
51.3 递增变换	805	第 58 章 纹理映射的优化	905
51.4 对象表示	808	58.1 使用最佳方法优化纹理映射	905
第 52 章 快速 3D 动画：使用 X-Sharp	810	58.2 纹理映射的缺陷	906
52.1 一个通用的 3D 动画软件包	810	58.2.1 固定思维的优化	906
52.2 本章的演示程序	810	58.2.2 完全不同的角度	908
52.3 一个新的动画框架：X-Sharp	826	58.3 最终的优化	910
52.4 实时动画的三个关键之处	826	58.4 关于纹理映射的几点说明	916
52.4.1 缺点	827	第 59 章 BSP 树	918
52.4.2 运行时间分配	828	59.1 BSP 树的概念	918
第 53 章 最高速度及其它	829	59.2 BSP 树	919
53.1 提高 3D 动画速度的真理	829	59.2.1 可见性判别	919
53.2 最高速度第一部分：汇编语言	829	59.2.2 BSP 树的局限	920
53.3 最高速度第二部分：查找表	837	59.3 建立 BSP 树	921
53.3.1 不可见面	838	59.4 BSP 树的顺序遍历	925
53.3.2 舍入	841	59.4.1 完全了解	927
53.4 球的 3D 动画	841	59.4.2 测量与学习	929
		59.5 BSP 树参考资料	931

第 60 章 编译 BSP 树	933	64.5 重复绘制	983
60.1 BSP 树的实现	933	64.6 beam 树	984
60.2 编译 BSP 树	934	64.7 三维引擎	985
60.2.1 参数线段	934	64.7.1 光线投射子划分	985
60.2.2 参数线段裁剪	936	64.7.2 不需要顶点的表面	986
60.2.3 BSP 编译器	936	64.7.3 绘制缓存	986
60.3 优化 BSP 树	943	64.7.4 基于段的绘制	986
60.4 有待研究的 BSP 优化	944	64.7.5 孔洞	986
第 61 章 参照物	945	64.8 突破	986
61.1 3D 图形的数学基础	945	64.9 简化并不断尝试新事物	987
61.1.1 3D 数学	945	64.10 学以致用	988
61.1.2 基本定义	946	64.11 参考书	988
61.2 点积	946	第 65 章 3D 裁剪	989
61.3 叉积和多边形法向量	948	65.1 信息交流	989
61.4 利用点积的符号	950	65.2 3D 裁剪的基础	990
61.5 点积用于投影	951	65.3 多边形裁剪	992
第 62 章 BSP 绘制算法	954	65.3.1 裁剪到视体内	995
62.1 优化被编译的 BSP 树	954	65.3.2 程序 65-3 的技巧	1001
62.2 基于 BSP 的绘制	955	65.4 视空间裁剪的优点	1002
62.3 绘制流水线	965	65.5 进一步学习	1003
62.3.1 观察者移动	965	第 66 章 Quake 中面的消隐	1004
62.3.2 变换到视空间	966	66.1 不可见面的 Z 序消隐	1004
62.3.3 裁剪	966	66.2 创造性的变化和不可见表面	1004
62.3.4 投影到屏幕空间	967	66.2.1 绘制运动对象	1004
62.3.5 遍历、背面删除及绘制	968	66.2.2 性能影响	1005
62.4 关于 BSP 绘制程序的几点说明	969	66.2.3 改进与均衡性能	1005
第 63 章 实时 3D 处理的浮点操作	970	66.3 跨度排序	1006
63.1 紧急事件的处理	970	66.4 边排序关键值	1010
63.2 浮点操作的新概念	971	66.4.1 1/z 方程的推导	1011
63.3 Pentium 的 FP 优化	971	66.4.2 Quake 和 Z 排序	1011
63.4 FXCH 指令	973	66.5 排序方法的取舍	1012
63.5 点积优化	974	第 67 章 实现跨度排列	1013
63.6 叉积优化	975	67.1 实现独立的跨度排序	1013
63.7 变换优化	976	67.2 Quake 和跨度排序	1013
63.8 投影优化	978	67.3 按 1/z 进行跨度排序的类型	1015
63.9 舍入控制	978	67.3.1 相交跨度排序	1015
63.10 不再使用 3D 定点	979	67.3.2 邻接跨度排序	1015
第 64 章 Quake 的可视表面判别	980	67.3.3 独立跨度排序	1016
64.1 可见性判别	980	67.4 1/z 跨度排序的实现	1017
64.2 VSD: 最难的三维问题	981	第 68 章 Quake 的光照模型	1029
64.3 Quake 级的结构	981	68.1 做感兴趣的事	1029
64.4 背面删除和可视表面判别	982	68.2 光照难题	1029

68.3 Gouraud 浓淡方法	1030	第 70 章 Quake: 回顾和展望	1047
68.3.1 Gouraud 浓淡方法的问题	1030	70.1 预处理基本空间	1048
68.3.2 透视校正	1031	70.2 可能可见集 (PVS)	1049
68.4 寻求其他光照方法	1032	70.3 几乎发生的改变	1050
68.4.1 分离光照和光栅化	1033	70.4 绘制基本空间	1050
68.4.2 尺寸和速度	1034	70.5 光栅化	1052
68.5 表面 cache	1035	70.5.1 光照	1052
68.5.1 均值映射 (Mipmapping)	1035	70.5.2 动态光照	1052
68.5.2 关于表面 cache 的最后两点 说明	1036	70.6 实体	1053
第 69 章 表面 cache 和 Quake 的三角形 模型	1037	70.6.1 BSP 模型	1053
69.1 保持理智	1037	70.6.2 多边形模型和 Z-buffer	1054
69.2 与硬件相关的表面 cache	1038	70.6.3 再分割光栅化程序	1055
69.2.1 使用图形卡构造纹理	1038	70.6.4 特殊效果	1055
69.2.2 作为 Alpha 纹理的光照图	1039	70.6.5 粒子	1056
69.3 绘制三角形模型	1039	70.7 Quake 推入市场后我们的计划	1056
69.3.1 快速绘制三角形模型	1039	70.7.1 Verite Quake	1056
69.3.2 子像素的精度换取速度	1041	70.7.2 GLQuake	1057
69.3.3 一种无效的方法	1041	70.7.3 WinQuake	1058
69.3.4 一种有效的方法	1042	70.7.4 QuakeWorld	1058
69.3.5 其他可能有效的方法	1046	70.7.5 Quake 2	1060
		70.8 展望	1061
		著后语	1063

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，再快也没有什么意思呢？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