信息科学与技术丛书 程序设计系列

刘纯根 著

浮点计算编程原理、实现与应用

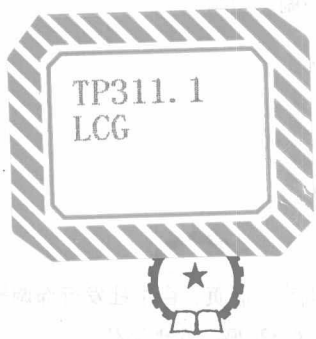
- C++ 与通用编程
- 大整数运算
- 建立超高精度的定点和浮点仿真库
- x87 FPU 编程
- 反汇编 VC6 浮点库
- 常见浮点编程技巧

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书
程序设计系列

浮点计算编程原理、实现与应用

刘纯根　著



机械工业出版社

本书介绍了基本计算算法的实现和代码分析，主要内容有：C++与通用编程、大整数运算、超高精度的定点和浮点仿真库、x87FPU 编程、反汇编 VC6 浮点库和常见浮点编程技巧。其中，超高精度的定点和浮点仿真库是作者为进行科学计算而开发的，具有较高的参考价值。

图书在版编目（CIP）数据

浮点计算编程原理、实现与应用/刘纯根著.—北京：机械工业出版社，2008.6
（信息科学与技术丛书 · 程序设计系列）
ISBN 978-7-111-24383-0

Ⅰ.浮… Ⅱ.刘… Ⅲ.程序设计 Ⅳ.TP311

中国版本图书馆 CIP 数据核字（2008）第 094340 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
责任编辑：车 忱
责任印制：洪汉军

北京振兴源印务有限公司印刷厂印刷

2008 年 8 月·第 1 版第 1 次印刷

184mm × 260mm · 19.75 印张 · 456 千字

0001—5000 册

标准书号：ISBN 978-7-111-24383-0

定价：37.00 元

凡购本图书，如有缺页、倒页、脱页，由本社发行部调换
销售服务热线电话：（010）68326294 68993821
购书热线电话：（010）88379639 88379641 88379643
编辑热线电话：（010）88379753 88379739
封面无防伪标均为盗版

出版说明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术、新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书，来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术受到不断挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着因特网延伸到人类世界的层层面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们对了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络、工程应用等内容，注重理论与实践相结合，内容实用，层次分明，语言流畅，是信息科学与技术领域专业人员不可或缺的图书。

现今，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前　言

本书内容

本书主要介绍基本计算算法的实现和代码分析，涉及任意长的整数计算、任意范围和精度的实数计算（包括定点运算和浮点运算）、常见数学函数的实现和 Visual C++ 6.0（简称 VC6）浮点库 C 形式的反汇编代码分析。

全书分两个部分：

第一部分讲述计算编程的基本原理与算法。自整型运算开始，步步推进直至建立通用的基本数学函数库。建立了支持整型运算的 CGUINT 和 CGINT、支持定点运算的 CFixedPoint、支持浮点运算的 CFloatPoint 和 CSuperFloat。其中 CFloatPoint 和 CSuperFloat 是在 IEEE 浮点运算标准的基础上建立的，支持 IEEE 浮点运算标准的浮点格式和一般约定。

这几个 C++模板几乎实现了常见的各种计算。而且，由于在设计和编码时采用了通用编程技术，使得 CGUINT 和 CGINT 可以支持任意长度的整型运算，CFixedPoint 和 CSuperFloat 可以支持任意精度的实数运算。

第二部分讲述 Intel 公司的 x87 FPU 与 Microsoft 公司的 VC6 浮点库，涉及 x87 FPU 编程模型、VC6 浮点库反汇编代码，深入分析了 VC6 浮点库的一些实现细节。

最后简要介绍了 x87 指令集和本书的源码。

写作缘起

我原是一名工科学生，毕业后成为一名工程技术人员，在专业领域工作了七八年，一直没能离开浮点运算。早年，我在 x86/DOS 上用 FORTRAN 计算弹道和轨道（那时还需要数学协处理器或浮点仿真库），后来转移到 x86/Windows 上使用 C/C++。由于这些语言的编译器对浮点运算的支持相当完善，我几乎没有觉察到浮点运算的特别之处。唯一不快的是，我有时喜欢用汇编编写一些代码，但由于当初学习微机原理时没学 x87 指令，始终没能用汇编编写一行浮点代码。

我从事的工作需要编写计算程序。由于计算错误可能导致严重后果，我常常担心数学函数超出定义域、出现被零除、误差太大之类的问题。自然，代码总是有问题的，即使没有出现问题的代码看上去也很可疑。调试过程中，VC6 的在线反汇编功能非常不错，常常显示下列代码：

```
FLD QWORD PTR [EBP+8]
CALL __fload_withFB
...
FCOMP QWORD PTR [__REAL@8@00000000000000000000]
FNSTSW AX
TEST AH 40h
```

这些代码在那时的我看来犹如天书，有时气得连把计算机砸掉的心都有了，有时又觉得那是对我的嘲弄。我小心地分析算法，检验各种参数，以避免一切可能的错误。例如，我曾

写过类似下列可笑的代码：

```
cos_theta = x/sqrt( x*x + y*y );
if( cos_theta > 1.0 )
    cos_theta = 1.0;
else if( cos_theta < -1.0 )
    cos_theta = -1.0;
return acos( cos_theta );
```

之所以如此，是因为我担心浮点运算的误差以及浮点数表示的不精确会导致数学函数在边界上出现问题。例如，如果 cos_theta 的值是 1.0，但由于浮点数可能带有随机误差，那么它的值可能是 1.00000000000000000034 之类，而这会导致 acos()失败，从而导致整个计算失败。我就是用这些可笑的技巧（有时没有这么可笑）来保证代码稳定的。显然，这种做法即使成功了，也没有什么可骄傲的。

后来，由于工作变动，我慢慢不再编写这类程序。我庆幸自己终于从尴尬的处境（我无法相信自己编写的代码是正确的，但必须相信它的结果）中脱身了，有段时间我看起了编译原理（这是整型的天下）和 Java（这里没有 x87 指令），浮点运算和汇编就像原始丛林一样被生活在现代都市中的我忘却了。

可是有一天，不知哪儿出了问题。首先是一个同事问我 NaN 是什么意思，他像我当年一样遇到了浮点运算中最可怕的标记——NaN。编了超过十年计算代码的我告诉他，那是 Not a Number 的意思。的确没错，但这对他毫无用处。然后在一个网站论坛上，一个可怜的家伙问如何实现浮点除法。我告诉他，现在的通用 CPU 都支持浮点运算，没必要关心浮点除法。可他说他必须实现一个浮点除法，我猜想他是一个偷懒的学生，便不再理他。当天晚上，一个同事走进我的宿舍，说我当年编写的一个程序怎么结果都不对。过去的恐惧一下子又浮上心头。我打开源码，从出错的地方硬着头皮反向查看，没有发现任何可疑之处。过了半个多小时，我突然意识到可能是他输入了错误的数据。打开数据文件，我发现他将高低角和方位角颠倒了。他走了以后，我又翻看了一下代码。在编写一般的程序时，通常我会写大量的检测代码。一旦程序出错，通常在出错的地方就会有告警提示（例如提示某个 Win32 API 返回的句柄无效），但在编写浮点代码时，由于对浮点运算的底层机制不明白，我总是有些缩手缩脚，不知怎样编写合适的检测代码（上面那些可笑的代码毫无用处）。只有当错误的运算结果出来时才发觉计算出错了，而错误的源头却不知道在哪儿。往往一个函数出了差错，却要在几个积分循环以后在另一个函数那儿导致计算无法进行（出现恐怖的 NaN、INF 或定义域错误）。对这类错误，除了搜山似地翻看源码并作种种假设猜想之外，毫无他法。

那段时间我正空闲。于是，我对自己说，那就看看 x87 指令吧，一劳永逸地清除这虚幻的恐惧。既然已经知道了上百条 x86 指令，那么再多几十条 x87 指令也不会太难。这是 2004 年底的事情。

我想，看些别人写好的文章或者书也许有助于理解。在网上搜寻了许久，除了几篇简单的 x87 指令说明（简单到只有指令本身和指令名字的翻译）之外，几乎任何有价值的东西都没找到。在林林总总无所不有的编程世界，浮点运算似乎是一块被人遗忘的角落。最后，只好从 Intel 公司的网站上下载了编程手册（文献[2]），从一个英文网站下载了一篇关于 IEEE 754 标准的演讲稿（文献[8]），从另一个英文网站下载了 IEEE 854 标准（文献[9]），后来又买了一

本密码编程方面的书（文献[1]，里面涉及大整数运算）。这就是我开始学习计算编程的所有资料。在学习过程中，我多次想到，要是当年编写浮点代码时知道这些东西，那该多好啊！可是现实就是这么嘲弄人，在我需要的时候，我不懂，在我懂得的时候，已不需要。

由于资料大都是英文的，看得多了，有时也翻译一些，在翻译的同时还加上自己的理解发挥一下。有时遇到不好理解的段落，就编上一段代码测试一下。这些东西多了以后，便显得杂乱无章，我自然就整理了一下。整理的时候发现，无论怎样，它们也无法连接成一个整体，需要在中间添加一些东西。于是，出现了两三篇文章。同时，为了了解实际中如何进行x87 FPU 编程，我开始反汇编 VC6 浮点库。经过一段时间以后，我想我也可以像 VC6 一样编写自己的浮点库了，可重复别人的代码毫无意义，我就把注意力集中在如何实现浮点运算的基本算法上。开始是基本的数学函数，后来连加减乘除也实现了。自然，这些东西已不是几篇文章所能组织起来的了，又不想这样凌乱地堆放着，写书的想法开始出现。这是 2005 年夏天的事情。

2005 年国庆，利用七天假期，我写下了完整的一章（第 7 章的原稿）。随后，我一边写，一边修饰代码，一边调整章节。在这个过程中，我慢慢感觉到，仅仅关注浮点运算是不行的。一方面是因为浮点运算需要大整数运算的支持，而大整数运算超出了常规整型（例如 C/C++ 中的 int 和 long）的运算范围，是一个不能忽视的问题；另一方面是因为常规的浮点运算存在精度限制问题。为什么不一劳永逸地实现任意类型、任意精度的计算呢？例如，计算 π 到小数点后 1000 位怎么样？于是，任意长度的整型运算和任意精度的实数运算进入了我的视野。开始，我以为这是一个不可实现的目标，但慢慢地还是有所进展，最终一个小小的计算系统出现了。这是 2005 年底的事情。

最初的热情过后，剩下的就是时间和毅力的问题了。

关于读者

由于本书是从学习、理解计算编程的角度编写的，因此只要你对计算有兴趣或有需要，那么你就是本书希望的读者。

算法的数学部分只涉及微分函数，因此只要受过本科教育均不会有困难。演示代码是用C++语言编写的，但只涉及基本的模板、函数重载、操作符重载。有一个地方（CGUINT 优化部分）有嵌入式汇编，但内容不多，而且不是基本的东西，可以不看。不过，需要有些 x86 汇编基础以理解 x87 浮点指令部分和 VC6 浮点库反汇编部分，但要求不高。

算法实现部分有些代码需要一些编程经验才能很好理解（例如数据的内存结构），有些部分使用了 VC6 编译器的特性（如嵌入式汇编），但难度均不高，只要有兴趣，即使是 C++语言初学者也可以通过努力读懂。一些细节可能有点晦涩（如弱规范数运算），需要一些时间才会慢慢明白。

关于代码

写编程方面的书或文章有一个实际问题，就是如何组织显示代码。如果将代码完整地印刷出来，有两方面的问题：一方面代码常常很长，与关心的问题无关的代码尤其长得出奇，这当然浪费了纸张（也浪费读者的金钱和时间）；另一方面，在纸面上、在没有代码管理工具（如集成环境的源码编辑器）的情况下，读者怎么读这些代码而不感到厌烦或头晕呢？可是如果在书中不出现代码，只讲数学形式的代码算法，将编程当作算法分析处理，读者又怎能熟悉那些并不能在算法中表现出来的细节呢？

可见，代码不可能不要，但也不能太多。但是，多少是合理的？对于某些喜欢编码的人而言，没有大量代码的书简直就是垃圾。他们甚至不愿翻这种书，而是直接阅读代码代替看书。对于另一类人，他们看见代码就头晕，喜欢数学形式或伪码形式的算法描述，认为这才是算法的精华，代码中琐碎的细节处理只是实现算法必须忍受的折磨。

这些思考决定了本书的大致特征，即在代码的基础上向算法分析靠拢。读者可以简单地认为本书是为代码编写的、含有算法解说的大型注释。它可以脱离计算机，不用在屏幕前阅读。当然，这不是说不用看代码，而是说核心代码均可以在书中看到。

本书的代码是演示性质的，但实现时考虑到了一些效率问题，因此一些代码可以直接使用，或者经过简单的修改就可使用。不过，出于可以理解的原因，这些代码只进行了简单的测试，在一些特性上也没有严格遵循 IEEE 标准，因此作者不能保证这些代码的可靠性，使用这些代码导致的后果应由使用者自己承担（使用者应该读懂代码，了解代码产生的结果）。

本书代码可在 http://www.cmpbook.com 下载。

读者购买本书就自动获得了本书代码的使用权，可以在任何场合使用。如果读者使用了本书代码或对代码作了修改，请在显眼处说明。

本书所有代码都是在 VC6 中编译通过的，没有在别的环境中作过测试。如果在别的开发环境下编译运行本书的代码，可能需要一些调整。

联系作者

由于工作关系，我很难有上网的机会。读者如果有需要，可以通过出版社与我联系。

作　者

目　录

第1章 引 论

1.1 计算有什么用?

在现代社会中，一个东西或者一种技艺要么有用，要么好玩，否则无人问津。说计算好玩，我想绝大部分人很难相信，那么证明计算有存在价值或者说值得关注的就只有说明它有用了。

无论是对于个人还是对于种族甚至人类，计算能力都是至关重要的。一些人对玛雅文明的惊奇就在于它拥有非凡的天文历法，而之所以说它的天文历法非凡的一个重要原因就是它里面充满了巨大的数字。在这里，计算能力被当做文明程度的标志。在个人生活中，一个幼儿数到 10、100 之类的数字往往是年轻的父母印象深刻的事件。在这里，计算能力被当做智力发育程度的标志。至于现代社会，计算能力已成为一个国家强大的基石。无论是大厦、桥梁的设计，还是飞机、导弹的研制，甚至一片小小的药片的合成都需要计算。下面列出需要计算的几个领域以说明计算的重要和它在现代社会的普及程度。

1.1.1 基础科学

在许多人的印象中，基础科学更多的是由一些抽象、复杂的符号组成的方程以及这些方程背后难以理解的概念。但其实这只是硬币的一面，硬币的另一面就是计算。数论中有一个著名的素数分布问题。大约 2300 年前，欧几里德（Euclid）就已经证明不存在最大的素数，但希尔伯特（Hilbert）二十三个问题之一的素数分布问题至今没有解决。由于素数是 RSA 密码系统的基石，因此研究这个问题不仅是单纯的理论需要，而且有重大的实用价值。时至今日，尽管拥有运算能力万亿次每秒的高速计算机，得到的成果也是有限的。已经发现的几个大素数见表 1-1。

表 1-1 大素数举例

素数	位数	发现者	年代
$2^{20\,996\,011}-1$	6 320 430	Michael Shafer	2003
$2^{6\,972\,593}-1$	2 098 960	Hajratwala, Woltman, Kurowski	1999
$2^{3\,021\,377}-1$	909 526	Clarkson, Woltman, Kurowski	1998
$2^{2\,976\,221}-1$	895 932	Spence, Woltman	1997

上面的例子也许过于抽象，下面再举一个具体一点的例子。

有个简单的迭代公式：

$$Z_{n+1}=Z_n^2+C, \qquad Z_0=0$$

如果一个复数 C 经过上面这个迭代公式无限次变换后得到的 $Z_n(n\to+\infty)$依然是有限值，

那么所有这些 C 就构成了一个集合，即著名的 Mandelbrot 集。这个集合是什么样子的呢？下面是它的大致模样：

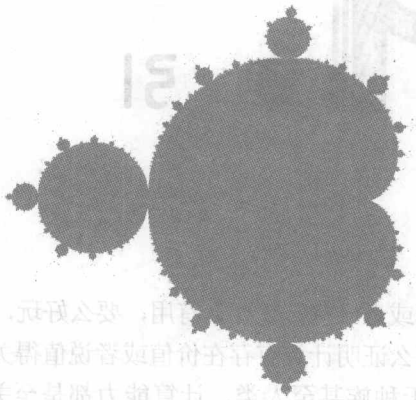


图 1-1 Mandelbrot 集概略图

你能想象出如此简单的迭代公式会产生如此复杂的图形吗？这还只是概略图。由于不可能进行无限次迭代，也不可能以无限小的步长进行搜索，因此无法给出完整的 Mandelbrot 集。粗一看，这似乎没有什么，但如果将图形的边缘部分放大以后，就会发现这种想法大错特错。边缘部分的图形似乎无限地重复，具有无限的细节，而且与迭代次数和搜索步长紧密相关，其形状在计算完成之前是无法预知的。

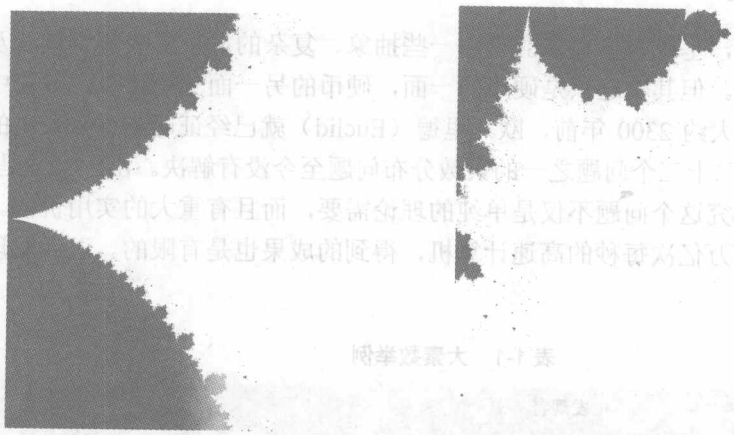


图 1-2 Mandelbrot 集局部图

如果读者对分形图形感兴趣，可以查阅相关资料（互联网上就有）。在本书附带的源码中有一个 Fractal 工程（在 Others 目录下），里面有一些分形图形的生成函数，是我平时的游戏之作，不过质量不高（数学上不够严谨，也没有进行颜色渲染）。

1.1.2 应用科学

应用科学的一般工作过程遵循着分析→建模→分析的循环模式。例如，为了确定一个高

能粒子的轨迹或者预测全球天气在未来几天内的变化，首先要依据普遍的物理定律和经验公式建立一组方程，然后分析、修正这个方程组及其模型参数，使其预测值接近实际观测值。这个过程循环往复，达到一定程度以后，就可使用这个方程组进行预测（下一秒时是什么情形？）和相关分析（某个因素发生了变化，对结果有什么影响？）。在这个过程中，模型参数的估计与修正、模型预测与观测数据的对比分析等所有行为都建立在计算的基础上。下面给出的方程是著名的 Lorenz 方程组，来自对天气的模拟：

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y-x) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = (r-z)x - y \\ \dfrac{\mathrm{d}z}{\mathrm{d}t} = xy - bz \end{cases}$$

这个方程有 3 个模型参数（r,σ,b）和 3 个变元（x,y,z），如果给定一组值：

（r,σ,b）=（ 28.0, 10.0, 2.6 ），（x,y,z）=（ 0.1, 0.2, 0.5 ）

那么可以得到如图 1-3 所示的轨迹：

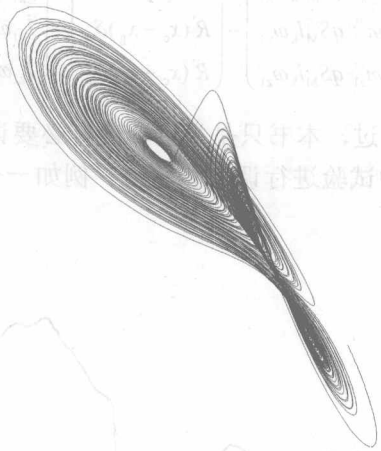


图 1-3　Lorenz 吸引子的轨迹

这条轨迹在三维空间始终绕着两个点转来转去，这就是著名的 Lorenz 吸引子。许多非线性系统（如天气系统）具有类似的特征，Lorenz 吸引子是研究非线性系统的经典案例。

1.1.3　工程项目

假设要进行一次导弹飞行试验。由于一些测量设备需要预先知道导弹的飞行轨迹、分析数据时也需要理论上的飞行轨迹，因此需要计算弹道方程组。它的矢量形式大致如下：

$$\begin{cases} m\dfrac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}t^2} = \boldsymbol{R} + m\boldsymbol{g} + \boldsymbol{P} + \boldsymbol{F}_c + \boldsymbol{F}_k \\ \boldsymbol{I}\dfrac{\mathrm{d}\boldsymbol{\omega}_T}{\mathrm{d}t} + \boldsymbol{\omega} \times (\boldsymbol{I} \cdot \boldsymbol{\omega}_T) = M_{st} + M_d + M'_{rel} + M'_k \end{cases}$$

第一组方程描述质心运动，来自牛顿第二定律；第二组方程描述姿态运动，来自动量矩

定理。它展开后大致是：

$$m\begin{pmatrix}\frac{\mathrm{d}v_x}{\mathrm{d}t}\\ \frac{\mathrm{d}v_y}{\mathrm{d}t}\\ \frac{\mathrm{d}v_z}{\mathrm{d}t}\end{pmatrix}=G_B\begin{pmatrix}P_e\\ Y_{1c}+2\dot{m}\omega_{Tz1}x_{1e}\\ Z_{1c}-2\dot{m}\omega_{Ty1}x_{1e}\end{pmatrix}+G_V\begin{pmatrix}-C_x qS_M\\ C_y^{\alpha}qS_M\alpha\\ -C_y^{\alpha}qS_M\beta\end{pmatrix}+m\frac{g_r}{r}\begin{pmatrix}x+R_{0x}\\ y+R_{0y}\\ z+R_{0z}\end{pmatrix}+m\frac{g_{we}}{\omega_e}\begin{pmatrix}\omega_{ex}\\ \omega_{ey}\\ \omega_{ez}\end{pmatrix}$$

$$-m\begin{pmatrix}a_{11}&a_{12}&a_{13}\\ a_{21}&a_{22}&a_{23}\\ a_{31}&a_{32}&a_{33}\end{pmatrix}\begin{pmatrix}x+R_{0x}\\ y+R_{0y}\\ z+R_{0z}\end{pmatrix}-m\begin{pmatrix}b_{11}&b_{12}&b_{13}\\ b_{21}&b_{22}&b_{23}\\ b_{31}&b_{32}&b_{33}\end{pmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{pmatrix}$$

$$\begin{pmatrix}I_{x1}&0&0\\ 0&I_{y1}&0\\ 0&0&I_{z1}\end{pmatrix}\begin{pmatrix}\frac{\mathrm{d}\omega_{Tx1}}{\mathrm{d}t}\\ \frac{\mathrm{d}\omega_{Ty1}}{\mathrm{d}t}\\ \frac{\mathrm{d}\omega_{Tz1}}{\mathrm{d}t}\end{pmatrix}+\begin{pmatrix}(I_{z1}-I_{y1})\omega_{Tz1}\omega_{Ty1}\\ (I_{x1}-I_{z1})\omega_{Tx1}\omega_{Tz1}\\ (I_{y1}-I_{x1})\omega_{Ty1}\omega_{Tx1}\end{pmatrix}$$

$$=\begin{pmatrix}0\\ m_{y1}^{\beta}qS_M l_k\beta\\ m_{z1}^{\alpha}qS_M l_k\alpha\end{pmatrix}+\begin{pmatrix}m_{x1}^{\omega_{x1}}qS_M l_k\omega_{x1}\\ m_{y1}^{\omega_{y1}}qS_M l_k\omega_{y1}\\ m_{z1}^{\omega_{z1}}qS_M l_k\omega_{z1}\end{pmatrix}-\begin{pmatrix}2R'r_c\delta_r\\ R'(x_c-x_g)\delta_{\psi}\\ R'(x_c-x_g)\delta_{\varphi}\end{pmatrix}-\begin{pmatrix}\dot{I}_{x1}\omega_{Tx1}\\ \dot{I}_{y1}\omega_{Ty1}\\ \dot{I}_{z1}\omega_{Tz1}\end{pmatrix}-\dot{m}\begin{pmatrix}0\\ x_{1e}^2\omega_{Ty1}\\ x_{1e}^2\omega_{Tz1}\end{pmatrix}$$

方程组中有大量参数（不过，本书只是举例，没有必要详细解说这些参数）都需要在设计时充分考虑，并通过各种试验进行调整和验证。例如一个气动力的模型参数如图 1-4 所示。

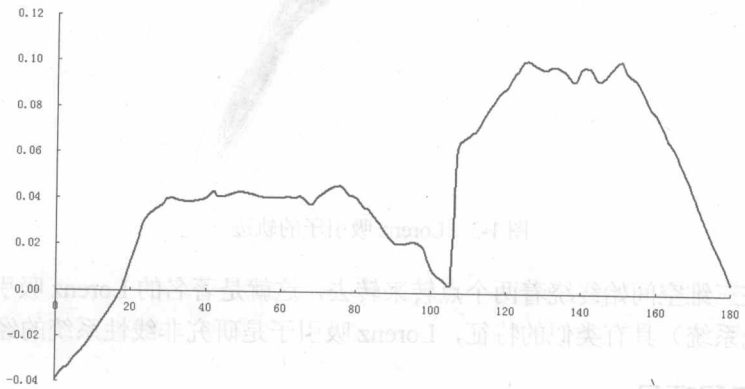


图 1-4　某气动力模型参数

显然，这个过程如果没有大量计算是不可能完成的。也正是因为如此，历史上研制导弹成了计算机技术发展的巨大动力。

1.1.4　日常生活

现代社会是信息社会，无处不在的信息自然伴随着无处不在的计算。一张 JPEG 图片的显示（余弦变换、Huffman 编码）、一个游戏场景的生成（三维模型、消隐处理、光

照模型、纹理处理）、打一个电话（语音采样、压缩、编码）、从银行或 ATM 机上取一笔钱（密码的加密与解密）等似乎极简单的日常行为无不激活、调动着巨大的、看不见的计算系统。

JPEG 文件格式如今已成为最常见的图片文件格式，整个互联网上 JPEG 文件无处不在。它最大的优势就是在保证显示质量的同时有极高的压缩率，而这个优点正是通过相当繁复的计算实现的。生成 JPEG 文件的大致流程如图 1-5 所示。

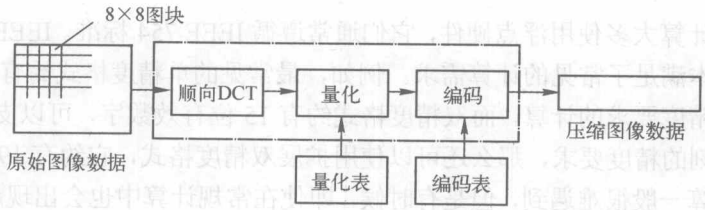


图 1-5 JPEG 压缩流程

显示 JPEG 文件的大致流程如图 1-6 所示。

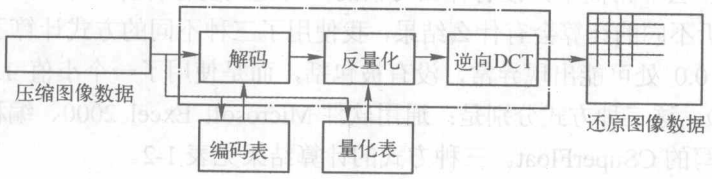


图 1-6 JPEG 解压缩流程

这里只是给出了流程的框图，实际实现时的计算量相当可观。图 1-7 是一幅图片与它做过 JPEG 压缩后的图片的比较。为了突出两者之间的差别，我特意大幅度降低了图片质量。

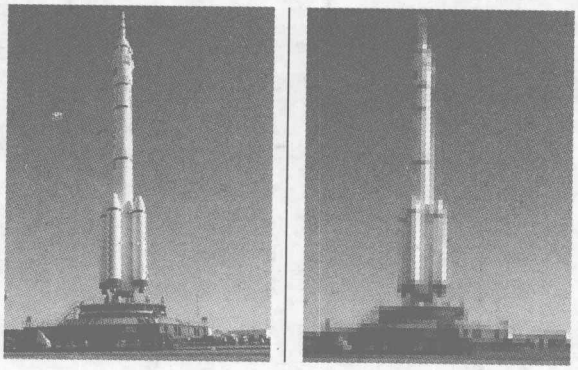


图 1-7 JPEG 解压效果示意

1.2 超高精度计算有什么用?

对计算而言，最基本的要求是计算正确。在数学上，一个数就是一个数，不能用别的数