计算机图形学原理 (虎书)

Fundamentals of Computer Graphics

作者：Steve Marschener & Peter Shirley

组织：515

时间：2022 年 11 月 6 日

版本：0.0

译者：Max Zhang

ElegantLaTeX Program

世界上只有一种英雄主义，那就是看清生活的真相之后依旧热爱生活。——罗曼罗兰

目录

前言

这一版的《计算机图形学基础》包含了对于材料着色、光反射、路径追踪的大量重写，以及从头至尾的更正。这一版书通过名为基于物理的材料和基于物理的渲染更好的介绍了计算机图形学技术，并且二者逐渐在实践中占据主导地位。现在这本材料被更好的整合了，我们认为这本书很好地反映了目前许多教师教授图形课程的组织大纲。

本书的结构与第四版基本相似。多年来我们对本书不断进行修订，我们努力保留了早期版本所特有的非正式、直观的表达方式，同时提高了一致性、精确性和完整性。我们希望在众多计算机图形学教材中，读者会觉得这本书是一个很有吸引力的平台。

关于封面

封面图片来自于 J.W.Baker 的《水中之虎》(www.jwbart.com)

老虎的主题参考了 Alain Fournier (1943-2000) 于 1998 年在康奈尔大学的一次研讨会上发表的精彩演讲。他的演讲是关于老虎运动的，是一个令人回味无穷的描述。他总结了自己的观点：

尽管在过去的 35 年中，计算机图形学的建模和渲染已经得到了极大的改进，但我们仍然无法自动模拟出在河中游泳老虎的所有精彩细节。自动是指不需要艺术家/专家仔细的进行手动调整。

坏消息是我们还有很长的路要走。

好消息是我们还有很长的路要走。

在线资源

本书的网址是http://www.cs.cornell.edu/~srm/fcg5/。我们将继续维护勘误表和那些使用本书的课程链接，以及与本书风格相匹配的教材。本书中的大多数图片都是 Adobe Illustrator 格式，我们很乐意根据要求将指定的图片转换为可移植格式。请随时通过 shirley@cs.utah.edu 或 srm@cs.cornell.edu 与我们联系。

致谢

以下人员提供了有关本书各个版本的有用信息、评论或反馈：Ahmet Oğuz Akyüz, Josh Andersen, Beatriz Trinchão Andrade Zeferino Andrade, Bagossy Attila, Kavita Bala, Mick Beaver, Robert Belleman, Adam Berger, Adeel Bhutta, Solomon Boulos, Stephen Chenney, Michael Coblenz, Greg Coombe, Frederic Cremer, Brian Curtin, Dave Edwards, Jonathon Evans, Karen Feinauer, Claude Fuhrer, Yotam Gingold, Amy Gooch, Eungyoung Han, Chuck Hansen, Andy Hanson, Razen Al Harbi, Dave Hart, John Hart, Yong Huang, John “Spike” Hughes, Helen Hu, Vicki Interrante, Wenzel Jakob, Doug James, Henrik Wann Jensen, Shi Jin, Mark Johnson, Ray Jones, Revant Kapoor, Kristin Kerr, Erum Arif Khan, Mark Kilgard, Fangjun Kuang, Dylan Lacewell, Mathias Lang, Philippe Laval, Joshua Levine, Marc Levoy, Howard Lo, Joann Luu, Mauricio Maurer, Andrew Medlin, Ron Metoyer, Keith Morley, Eric Mortensen, Koji Nakamaru, Micah Neilson, Blake Nelson, Michael Nikelsky, James O’Brien, Hongshu Pan , Steve Parker, Sumanta Pattanaik, Matt Pharr, Ken Phillis Jr, Nicolò Pinciroli, Peter Poulos, Shaun Ramsey, Rich Riesenfeld, Nate Robins, Nan Schaller, Chris Schryvers, Tom Sederberg, Richard Sharp, Sarah Shirley, Peter Pike Sloan, Hannah Story, Tony Tahbaz, Jan Phillip Tiesel, Bruce Walter, Alex Williams, Amy Williams, Chris Wyman, Kate Zebrose, Angela Zhang。

Ching-Kuang Shene 和 David Solomon 允许我们借用他们的例子。Henrik Wann Jensen、Eric Levin、Matt Pharr 和 Jason Waltman 慷慨地提供了图片。Brandon Mansfield 帮助改进了关于光线追踪的分层包围体的讨论。Philip Greenspun (philip.greenspun.com) 好心地允许我们使用他的照片。John ”Spike” Hughes 帮助改进了对抽样理

论的讨论。Wenzel Jakob 的 Mitsuba 渲染器在帮助我们创作照片方面非常宝贵。我们非常感谢 J.W. Baker 帮助创作了 Pete 设想的封面。抛开他才华横溢艺术家的身份，与他个人合作也是一种极大的乐趣。

章节注释中引用了许多有助于编写本书的作品。然而，一些影响内容和表达的关键书籍值得特别关注。其中包括两个经典的计算机图形学书籍，我们从两本书中学习了基础知识：计算机图形术：*Computer Graphics: Principles & Practice(Foley、Van Dam、Feiner 和 Hughes，1990)* 和 *Computer Graphics (Hearn & Baker, 1986)*，Hill 的 *Computer Graphics Using OpenGL (Francis S. Hill, 2000)*，Angel 的 *Interactive Computer Graphics: A Top-Down Approach Using OpenGL (Angel, 2002)*, Hugues Hoppe 的华盛顿大学论文 (Hoppe, 1994)，Rogers 的两篇优秀的图形学文章 (D. F. Rogers, 1985, 1989)。

我们要特别感谢 Alice 和 Klaus Peters，感谢他们鼓励 Pete 编写本书的第一版，并感谢他们在完成一本书方面的高超技巧。他们对作者的耐心，以及他们致力于让自己的书尽可能地做到最好的态度，对这本书的成功起到了至关重要的作用。没有他们非凡的努力，这本书无法完成。

译者续

本书翻译著名的 *Fundamentals of Computer Graphics*(计算机图形学原理，虎书)，水平有限，最大的目的是为了学习。

译者贡献列表 (实时更新)：

- Nancy Dong
- Max Zhang

第 1 章 介绍

API:application program interface: 应用程序接口

Computer graphics 描述的是任何使用计算机去创造和操纵图像的一门学科。这本书介绍了可用于创建各种图像的算法和数学工具—逼真的视觉效果，信息技术插图或者是生动的计算机仿真。Graphics 也可以是二维或三维的，images 可以被完全合成或者由一些 photographs 操纵。这本书是关于基础算法与数学，特别是那些用来产生 three-dimensional objects 和 scenes 的算法。

一般情况做计算机图形需要了解一些特定的硬件、文件格式，还有个别的图形 API（application program interface）。计算机图形学是一个快速发展的领域，所以对于该领域的学习是无止境的。因此，在这本书中，我们尽力避免依赖任何特定的硬件或 API。我们也支持读者用他们的软件和硬件环境的相关文件来补充该文本。幸运的是，计算机图形文化有足够的标准术语和概念，本书的讨论应该很好地应用到大多数环境。

本章定义了计算机图形学的一些基本术语，并提供了一些发展背景，以及与计算机图形有关的资料来源。

1.1 Graphics Areas（图形领域）

在任何行业领域中硬性的分类都是不合理的，但大多数图形学研究者认同以下计算机图形学主要领域。

(1) Modeling（建模）以可存储在计算机上的方式处理 mathematical specification of shape and appearance properties。例如，一个咖啡杯可以被描述为一组有序的 3D 点，以及一些连接点的插值规则和一个描述光线如何与杯子相互作用的反射模型

(2) Rendering（渲染）是从艺术领域借鉴而来的术语，涉及到从三维计算机模型中创建阴影图像

(3) Animation（动画）是一种通过图像序列创造运动图像的技术。动画使用建模（Modeling）和渲染（Rendering），但随时间移动是其关键问题，这在基础建模和渲染中通常是无法解决的。还有许多其他涉及计算机图形的领域，它们是否属于核心图形领域，见仁见智。这些都将在文中至少有所触及。这些相关领域包括以下内容：

(4) User interaction（用户交互）涉及输入设备（如鼠标和平板电脑）、应用程序、图像中对用户的反馈以及其他感官反馈之间的界面。从发展历史上看，这一领域与图形学有关，主要是因为图形学研究人员最早接触到了现在无处不在的输入/输出设备。

(5) Virtual reality（虚拟现实）试图让用户沉浸在一个三维虚拟世界中。这通常要求至少有立体图形和对头部运动的反应。对于真正的虚拟现实，还应该提供声音和力反馈。因为这一领域需要先进的 3D 图形和先进的显示技术，所以它通常与图形密切相关。

(6) Visualization（可视化）试图通过视觉显示让用户深入了解复杂的信息。通常，在一个可视化问题中，有一些图形问题需要解决。

(7) Image processing（图像处理）涉及对二维图像的操作，在图形和视觉领域都有应用。

(8) D scanning（3D 扫描）使用测距技术来创建测量的三维模型。这样的模型对于创造丰富的视觉图像很有帮助，而对这种模型的处理往往需要图形算法。

(9) Computational photography（计算型摄影/拍照滤镜）是使用计算机图形、计算机视觉和图像处理方法，以实现拍摄物体、场景和环境的新方法。

1.2 Major Applications（应用领域）

几乎任何工作都可以在一定程度上使用计算机图形，但计算机图形技术的主要消费者包括以下行业。

(1) Video games（电子游戏）越来越多地使用复杂的 3D 模型和渲染算法。

(2) Cartoons（动画片）通常是直接由 3D 模型渲染的。许多传统的二维动画片使用由三维模型渲染的背景，这使得连续移动的视角不需要大量的 artist time。

(3) Visual effects（视觉特效）几乎使用所有类型的计算机图形技术。几乎每部现代电影都使用数字合成技术，将背景与单独拍摄的前景叠加在一起。许多电影还使用三维建模和动画来创造合成环境、物体，甚至是大多数观众不会怀疑的角色。

(4) Animated films（动画电影）使用了许多与视觉特效相同的技术，但不一定要以看起来真实的图像为目标。

(5) CAD/CAM（计算机辅助设计和计算机辅助制造）这些领域使用计算机技术在计算机上设计零件和产品，然后使用这些虚拟设计来指导制造过程。例如，许多机械零件在 3D 计算机建模包中设计，然后在计算机控制的铣削设备上自动生产。

(6) Simulation（仿真）可以被认为是精确的视频游戏。例如，飞行模拟器使用复杂的 3D 图形来模拟驾驶飞机的体验。这种模拟对于安全关键领域 (如驾驶) 的初始培训以及有经验用户的情景培训 (如成本太高或太危险而无法实际创建的特定消防情况) 非常有用。

(7) Medical imaging（医学成像）创建通过扫描患者产生数据的有价值的图像。例如，计算机断层摄影 (CT) 数据集由大型 3D 矩形密度值阵列组成。计算机图形被用来创建阴影图像，帮助医生从这些数据中提取最突出的信息。

(8) nformation visualization（信息可视化）创建的数据图像不一定具有“自然”的视觉描述。例如，十种不同股票价格的时间趋势没有明显的视觉描述，但巧妙的图形技术可以帮助人类看到这些数据中的模式。

1.3 Graphics APIs（图形学应用程序接口）

使用图形库的一个关键部分是处理图形 API。应用程序接口 (API) 是执行一组相关操作的标准函数集合，图形 API 是执行基本操作的一组函数，例如将图像和 3D 表面绘制到屏幕上的窗口中。

每个图形程序都需要能够使用两个相关的 API: 一个用于可视化输出的图形 API 和一个用于获取用户输入的用户界面 API。目前有两种图形和用户界面 API 的主流范例。第一种是集成方法，以 Java 为例，其中图形和用户界面工具包是集成的、可移植的包，完全标准化并作为语言的一部分得到支持。第二种以 Direct3D 和 OpenGL 为代表，其中绘图命令是与 C++ 等语言相关的软件库的一部分，用户界面软件是一个独立的实体，可能因系统而异。在后一种方法中，编写可移植代码是有问题的，尽管对于简单的程序，使用可移植库层来封装系统特定的用户界面代码是可能的。

无论选择哪种 API，基本的图形调用基本上都是一样的，本书的概念也适用。

1.4 Graphics Pipeline（图形管道）

如今，每台台式电脑都有强大的 3D 图形管道。这是一个特殊的软件/硬件子系统，可以有效地绘制三维图元。通常，这些系统针对处理具有共享顶点的 3D 三角形进行了优化。在 pipeline map 中其基本操作是将 3D 顶点位置映射到 2D 屏幕位置，并对三角形进行着色，以使它们看起来真实，并以 back -to -front 顺序出现。虽然用 back-to-front 的顺序绘制三角形曾经是计算机图形学中最重要的研究问题，但现在大多是使用 z-buffer 来解决，z-buffer 使用一种特殊的内存缓冲区以 brute-force 方式解决问题。

事实证明，在 graphics pipeline 中使用的几何操作几乎完全可以在 4D 坐标空间中完成，该空间由 3D 坐标和有利于（perspective viewing）透视观察的第四个（homogeneous coordinate）齐次坐标组成。这些 4D 坐标使用 4 × 4 矩阵和 4 个向量来操作。因此，graphics pipeline 包含许多用于高效处理和合成这种矩阵和向量的 machinery。这个 4D 坐标系是计算机科学中使用的最微妙和最美丽的构造之一，它当然是学习计算机图形学时要跳过的最大的智力障碍。每本图形学书第一部分的很多内容都是关于 4D 坐标的。

生成图像的速度很大程度上取决于所画三角形的数量。因为在许多应用程序中，交互性比视觉效果更重要，所以尽量减少用于表示模型的三角形数量是值得的。此外，如果从远处观察模型，比从近处观察模型需要更少的三角形。这表明用不同的 level of detail(LOD) 来表示模型是有用的。

1.5 Numerical Issues(数值问题)

许多图形程序实际上只是 3D 数值代码。在这样的程序中，数值问题通常是至关重要的。在“过去”，以健壮和可移植的方式处理这样的问题是非常困难的，因为机器对数有不同的内部表示，处理异常的方式不同，且不兼容。幸运的是，几乎所有的现代计算机都符合 IEEE 浮点标准 (IEEE Standards Association，1985)。这使得程序员可以对如何处理某些数字条件做出许多方便的假设。

虽然 IEEE 浮点有很多特性，在编码数字算法时很有价值，但在图形学中遇到的大多数情况下，只有少数几个特性是必须知道的。首先，也是最重要的一点，就是要了解 IEEE 浮点中的实数有三个” 特殊” 值。

1. 无穷大（Infinity ）这是一个有效的数字，比其他所有有效的数字都大。

2. 负无穷大（Minus infinity ）这是一个有效的数字，比其他所有的有效数字都小。

3. 非数值形式（NaN）这是一个无效的数字，产生于一个后果未定的操作，如零除以零。

IEEE 浮点的设计者做出了一些决定，对程序员来说非常方便。其中很多都与上述三个处理异常的特殊值有关，比如除以零。在这些情况下，会记录异常，但在很多情况下，程序员可以忽略这一点。具体来说，对于任何正的实数 a，以下涉及除以无限值的规则 IEEE 浮点有两种零的表示方法，一种是被视为正数，一种是被视为负数。虽然-0 和 +0 之间的区别只是偶尔会出现，但在出现时还是值得记住的。

$$\begin{aligned} +a/(+\infty) &= +0 \\ -a/(+\infty) &= -0 \\ +a/(-\infty) &= -0 \\ -a/(-\infty) &= +0 \end{aligned} \tag{1.1}$$

其他涉及无限值的操作的行为与人们预期的一样。同样对于正 a，其操作如下。

$$\begin{aligned} \infty + \infty &= +\infty \\ \infty - \infty &= NaN \\ \infty * \infty &= \infty \\ \infty/\infty &= NaN \\ \infty/a &= NaN \\ \infty/0 &= NaN \\ 0/0 &= NaN \end{aligned} \tag{1.2}$$

涉及无限值的布尔表达式中的规则与预期一致。

1. 所有有限的有效数字都小于正无穷
2. 所有有限的有效数字都大于付无穷
3. 负无穷 is less than 正无穷

涉及有 NaN 值的表达式的规则很简单：

1. 任何包含 NaN 的算术表达式的结果都是 NaN
2. 任何涉及 NaN 的布尔表达式都是假的

也许 IEEE 浮点数最有用的方面是如何处理除以零的问题；对于任何正实数 a，以下涉及除以零值的规则是成立的。

$$\begin{aligned} +a/+0 &= +\infty \\ -a/+0 &= -\infty \end{aligned} \tag{1.3}$$

如果程序员利用 IEEE 规则，有许多数字计算会变得简单得多。例如，考虑表达式。

$$a = \frac{1}{\frac{1}{b} + \frac{1}{c}} \tag{1.4}$$

这样的表达方式出现在电阻和透镜上。如果除以零会导致程序崩溃（在 IEEE 浮点运算之前的许多系统中都是如此），那么就需要两个 if 语句来检查 b 或 c 的小值或零值。相反，使用 IEEE 浮点技术，如果 b 或 c 为零，我们就会如愿得到 a 的零值。另一种避免特殊检查的常用技术是利用 NaN 的布尔特性。考虑一下下面的代码段

$$\begin{aligned} a &= f(x) \\ if(a > 0)then & \\ do\ something & \end{aligned} \tag{1.5}$$

在这里，函数 f 可能会返回”ugly ” 的值，如 ∞ 或 NaN，但 if 条件仍然是定义明确的：当 a=NaN 或 a=-∞ 时为假，a=+∞ 时为真。在决定返回哪些值时要小心，通常 if 可以做出正确的选择，而不需要特别的检查。这使得程序更智能，更健壮，更有效率。

1.6 Efficiency（有效性）要使代码更有效率没有捷径。效率是通过仔细的权衡来实现的，而这些权衡对于不同的架构是不同的。然而，在可预见的未来, 程序员应该更加关注 (memory access patterns) 内存访问模式而不是 (operation counts) 操作数。这与 20 年前的最佳启发式方法相反。出现这种转换是因为内存的速度没有跟上处理

器的速度。由于这一趋势仍在继续，有限和连贯的内存访问对优化的重要性应该只会增加。一个合理的使代码快速化的方法是按以下顺序进行:

1. 尽可能以最直接的方式编写代码, 根据需要及时计算中间结果，而不是存储它们。（最重要的方法）

2. 在优化模式下进行编译。

3. Use whatever profiling tools exist to find critical bottlenecks（关键瓶颈）。

4. 检查数据结构，寻找提高定位性的方法。如果可能的话，使数据单元大小与目标架构上的缓存/页面大小相匹配。

5. 如果剖析揭示了 numeric computations 的瓶颈，检查由编译器生成的汇编代码是否有遗漏的效率。重写源代码以解决你发现的任何问题。大多数的" 优化" 使代码更难读，但却没有加快速度。此外，前期花在优化代码上的时间通常是用来纠正错误或增加功能的。另外，要注意旧文本中的建议；一些经典的技巧，如使用整数而不是实数，可能不再产生速度，因为现代的 CPU 通常可以执行浮点运算，就像执行整数运算一样快。

在所有情况下，都需要分析以确保任何优化对特定机器和编译器的好处。

1.7 Designing and Coding Graphics Programs（设计和编码图形程序）

有些常见的策略在计算机图形学编程中是很有用的，下面章节便是为此给出我们对于计算机图形学的一些建议。

1.7.1 Class Design

任何图形程序的关键部分都是为几何实体 (如向量和矩阵) 以及图形实体 (如 RGB 颜色和图像) 提供良好的类或例程。这些例行程序应该尽可能地简洁和高效。一个通用的设计问题是位置和位移是否应该是单独的类，因为它们有不同的操作，例如，位置乘以 1 / 2 没有几何意义，而位移乘以 1 / 2 有几何意义 (Goldman, 1985;DeRose, 1989)。在这个问题上几乎没有达成一致意见，这可能会在图形从业者之间引发数小时的激烈辩论，但为了举例，我们假设我们不进行区分。这意味着要编写的一些基本类包括:

1. vector2 一个二维向量类，它存储一个 x 和 y 分量。它应该将这些组件存储在长度为 2 的数组中，以便能够很好地支持索引操作符。还应该包括向量加法、向量减法、点乘、叉乘、标量乘法和标量除法的运算。

2. Vector3 一个类似于 vector2 的 3D 向量类。

3. Hvector 具有四个分量的齐次向量 (见第 7 章)

4. Rgb 一种 RGB 颜色，存储三个组件。你还应该包括 RGB 加法，RGB 减法，RGB 乘法，标量乘法和标量除法的操作。

5. Transform 一个用于变换的 4 × 4 矩阵。你应该包含矩阵乘法和成员函数，以应用于位置、方向和表面法向量。如第 6 章所示，这些都是不同的。

6. Image 带有输出操作的 RGB 像素的 2D 数组。

1.7.2 Float vs. Double

现代架构表明，降低内存使用量和保持一致的内存访问是提高效率的关键。这就建议使用单精度数据。然而，为了避免数值问题，建议使用双精度算法。权衡取决于程序，但是最好在类定义中有一个默认值。

1.7.3 Debugging Graphics Programs

如果您四处询问，您可能会发现随着程序员越来越有经验，他们使用传统调试器的次数越来越少。这样做的一个原因是，在复杂的程序中使用这样的调试器比在简单的程序中使用更尴尬。另一个原因是，最困难的错误是概念上的错误，其中实现了错误的东西，很容易浪费大量的时间在逐级检查变量值而没有检测到这种情况。我们发现了几种调试策略在图形中特别有用。

The Scientific Method

在图形程序中，有一种替代传统调试的方法，通常非常有用。它的缺点是，这非常类似于计算机程序员在职业生涯早期被教导不要做的事情，所以如果你这样做，你可能会觉得“顽皮”: 我们创建一个图像，然后观察它有什么问题。然后，我们提出一个关于问题原因的假设，并对其进行验证。例如，在光线追踪程序中，我们可能会有许多看起来有点随机的深色像素。这是大多数人在编写射线跟踪程序时会遇到的典型“shadow acne”问题。传统的调试在这里没有帮助; 相反，我们必须意识到阴影射线击中表面被遮蔽。我们可能会注意到黑点的颜色是环境色，所以缺少的是直接照明。直接照明可以在阴影中关闭，所以您可能会假设这些点被错误地标记为阴影

中，而实际上它们并不是。为了验证这个假设，我们可以关闭跟踪检查并重新编译。这将表明这些是假的阴影测试，我们可以继续我们的假设工作。这种方法有时是很好的实践的关键原因是，我们从来不必发现错误的值或真正确定我们的概念错误。

相反，我们只是缩小了实验上的概念错误。通常只需要进行几次试验就可以跟踪问题，而且这种类型的调试是令人愉快的。

Images as Coded Debugging Output 在许多情况下，从图形程序中获取调试信息的最简单的通道就是输出图像本身。如果你想知道某个变量的值，在每个像素运行的计算中，你可以临时修改你的程序，直接将该值复制到输出图像中，并跳过通常需要完成的其他计算。例如，如果您怀疑曲面法线的问题导致了阴影的问题，您可以直接将法向量复制到图像中 (x 变为红色，y 变为绿色，z 变为蓝色)，从而生成计算中实际使用的向量的彩色插图。或者，如果您怀疑某个特定值有时超出了它的有效范围，那么让您的程序在发生这种情况的地方编写鲜红色像素。

其他常见的技巧包括用明显的颜色绘制表面的背面 (当它们不应该可见时)，根据物体的 ID 号为图像着色，或根据计算所需的工作量为像素着色。

Using a Debugger

仍然有一些情况，特别是当科学方法似乎导致矛盾的时候，当没有任何替代方法可以精确观察正在发生的事情时。问题是，图形程序经常涉及到对相同代码的多次执行 (例如，每个像素执行一次，或每个三角形执行一次)，这使得从一开始就在调试器中逐级执行完全不切实际。最困难的 bug 通常只发生在复杂的输入中。

一个有用的方法是为 bug“设置一个陷阱”。首先，确保您的程序是确定性的——在单个线程中运行它，并确保所有随机数都是从固定的种子计算出来的。然后，找出哪个像素或三角形显示了错误，并在您怀疑是错误的代码之前添加一条语句，该语句只会在可疑的情况下执行。例如，如果你发现像素 (126,247) 显示了错误，然后添加:

$$fx = 126\ and\ y = 247\ then$$
$$print"bl\ arg!" \tag{1.6}$$

如果在 print 语句上设置了断点，那么就可以在计算感兴趣的像素之前进入调试器。一些调试器具有“条件断点”特性，可以在不修改代码的情况下实现相同的功能。在程序崩溃的情况下，传统的调试器对于确定崩溃的位置非常有用。然后你应该在程序中开始回溯，使用断言和重新编译，以找到程序出错的地方。这些断言应该保留在程序中，以应对将来可能添加的错误。这再次意味着避免了传统的分步过程，因为这样就不会向程序中添加有价值的断言。

Data Visualization for Debugging（用于调试的数据可视化）

通常情况下是很难理解你的程序在做什么，因为它在最终出错之前会计算很多中间结果。情况类似于测量大量数据的科学实验，其解决方案是: 为自己制作好的图表和插图，以理解数据的含义。

例如，在光线跟踪器中，你可以编写代码来可视化光线树，以便你可以看到哪些路径对像素有贡献，或者在图像重采样过程中，你可以绘制显示从输入中获取样本的所有点的图。花费时间编写代码来可视化程序内部状态，在优化它的时候，也可以更好地理解它的行为得到反馈。(将调试输出语句格式化，生成 matlab 或 Gnuplot 脚本可以绘制一些有用的图)

Notes（备注）

我们可以在网络上找到关于计算机图形学有关的年度会议，如 ACM SIGGRAPH 和 SIGGRAPH Asia、Graphics Interface、Game Developers Conference (GDC)、Eurographics、Pacific Graphics、High Performance Graphics、Eurographics Symposium on Rendering 以及 IEEE VisWeek。

第 2 章 数学杂项

第 3 章 光栅图像

第 4 章 光线追踪

第 5 章 表面着色

第 6 章 线性代数

第 7 章 变换矩阵

第 8 章 视图

第 9 章 图形管线

第 10 章 信号处理

第 11 章 纹理映射

第 12 章 图形学的数据结构

第 13 章 采样

第 14 章 基于物理的渲染

第 15 章　曲线

第 16 章 电脑动画

第 17 章 使用图形硬件

第 18 章 色彩

第 19 章 视觉感知

第 20 章 色调重现

第 21 章 隐式建模

第 22 章 游戏中的计算机图形学

第 23 章 可视化