1 Introduction

The term computer graphics describes any use of computers to create and manipulate images. This book introduces the algorithmic and mathematical tools that can be used to create all kinds of images—realistic visual effects, informative technical illustrations, or beautiful computer animations. Graphics can be two- or three-dimensional; images can be completely synthetic or can be produced by manipulating photographs. This book is about the fundamental algorithms and mathematics, especially those used to produce synthetic images of three-dimensional objects and scenes.

计算机图形一词是指使用计算机创建和处理图像。本书介绍了可用于创建各种图像—真实视觉效果、内容丰富的技术插图或精美的计算机动画--的算法和数学工具。图形可以是二维的，也可以是三维的；图像可以是完全合成的，也可以通过处理照片来制作。本书主要介绍基本算法和数学，尤其是用于制作三维物体和场景合成图像的算法和数学。

Actually doing computer graphics inevitably requires knowing about specific hardware, file formats, and usually a graphics API (see Section 1.3) or two. Computer graphics is a rapidly evolving field, so the specifics of that knowledge are a moving target. Therefore, in this book we do our best to avoid depending on any specific hardware or API. Readers are encouraged to supplement the text with relevant documentation for their software and hardware environment. Fortunately, the culture of computer graphics has enough standard terminology and concepts that the discussion in this book should map nicely to most environments.

在实际操作计算机图形学时，不可避免地需要了解特定的硬件、文件格式，通常还需要了解一两个图形应用程序接口（见第 1.3 节）。计算机图形学是一个快速发展的领域，因此这些知识的具体内容是一个不断变化的目标。因此，在本书中，我们尽力避免依赖任何特定的硬件或 API。我们鼓励读者为自己的软件和硬件环境补充相关文档。幸运的是，计算机图形文化有足够多的标准术语和概念，本书的讨论应该可以很好地映射到大多数环境中。

This chapter defines some basic terminology and provides some historical background, as well as information sources related to computer graphics.

本章定义了一些基本术语，介绍了一些历史背景以及与计算机图形学有关的信息来源。

1.1 Graphics Areas

Imposing categories on any field is dangerous, but most graphics practitioners would agree on the following major areas of computer graphics:

• Modeling deals with the mathematical specification of shape and appearance properties in a way that can be stored on the computer. For example, a coffee mug might be described as a set of ordered 3D points along with some interpolation rule to connect the points and a reflection model that describes how light interacts with the mug.

• Rendering is a term inherited from art and deals with the creation of shaded images from 3D computer models.

• Animation is a technique to create an illusion of motion through sequences of images. Animation uses modeling and rendering but adds the key issue of movement over time, which is not usually dealt with in basic modeling and rendering.

对任何领域进行分类都是危险的，但大多数图形从业人员都会同意计算机图形学的以下主要领域：

- 建模（Modeling）：建模是用数学的方式来规范形状和外观属性，并将其存储在计算机中。例如，一个咖啡杯可以描述为一组有序的三维点，以及连接这些点的插值规则和描述光线如何与咖啡杯相互作用的反射模型。

- 渲染（Rendering）是从艺术中继承下来的一个术语，用于根据三维计算机模型创建阴影图像。

- 动画是一种通过图像序列创造运动幻觉的技术。动画使用建模和渲染，但增加了随时间运动的关键问题，而基本建模和渲染通常不会处理这个问题。

There are many other areas that involve computer graphics, and whether they are core graphics areas is a matter of opinion. These will all be at least touched on in the text. Such related areas include the following:

涉及计算机图形学的领域还有很多，至于它们是否属于核心图形学领域，则见仁见智。文中至少会涉及到这些领域。这些相关领域包括:

• User interaction deals with the interface between input devices such as mice and tablets, the application, feedback to the user in imagery, and other sensory feedback. Historically, this area is associated with graphics largely because graphics researchers had some of the earliest access to the input/output devices that are now ubiquitous.

用户交互涉及诸如鼠标和平板电脑等输入设备、应用程序、以图像形式向用户提供的反馈以及其他感官反馈之间的接口。从历史上看，这一领域在很大程度上与图形学相关，主要是因为图形学研究人员最早接触到了如今已无处不在的输入 / 输出设备

• Virtual reality attempts to immerse the user into a 3D virtual world. This typically requires at least stereo graphics and response to head motion. For true virtual reality, sound and force feedback should be provided as well. Because this area requires advanced 3D graphics and advanced display technology, it is often closely associated with graphics.

虚拟现实试图让用户沉浸在三维虚拟世界中。这通常至少需要立体图形和对头部运动的响应。对于真正的虚拟现实，还应提供声音和力反馈。由于这一领域需要先进的三维图形和先进的显示技术，因此往往与图形技术密切相关。

1.3 Graphics APIs

A key part of using graphics libraries is dealing with a graphics API. An application program interface (API) is a standard collection of functions to perform a set of related operations, and a graphics API is a set of functions that perform basic operations such as drawing images and 3D surfaces into windows on the screen.

使用图形库的一个关键部分是处理图形应用程序接口。应用程序接口（API）是执行一系列相关操作的标准函数集，而图形 API 是执行基本操作（如在屏幕上的窗口中绘制图像和 3D 曲面）的函数集。

Every graphics program needs to be able to use two related APIs: a graphics API for visual output and a user-interface API to get input from the user. There are currently two dominant paradigms for graphics and user-interface APIs. The first is the integrated approach, exemplified by Java, where the graphics and user interface toolkits are integrated and portable packages that are fully standardized and supported as part of the language. The second is represented by Direct3D and OpenGL, where the drawing commands are part of a software library tied to a language such as C++, and the user-interface software is an independent entity that might vary from system to system. In this latter approach, it is problematic to write portable code, although for simple programs, it may be possible to use a portable library layer to encapsulate the system specific user-interface code.

每个图形程序都需要能够使用两个相关的应用程序接口：一个是用于视觉输出的图形应用程序接口，另一个是用于获取用户输入的用户界面应用程序接口。目前，图形和用户界面 API 有两种主流模式。第一种是以 Java 为代表的集成方法，其中图形和用户界面工具包是集成的、可移植的软件包，作为语言的一部分得到完全标准化和支持。第二种方法以 Direct3D 和 OpenGL 为代表，绘图命令是与 C++ 等语言相关联的软件库的一部分，而用户界面软件则是独立的实体，可能因系统而异。在后一种方法中，编写可移植代码是个问题，尽管对于简单的程序来说，可以使用可移植库层来封装系统特定的用户界面代码。

Whatever your choice of API, the basic graphics calls will be largely the same, and the concepts of this book will apply.

无论您选择何种应用程序接口，基本的图形调用大体相同，本书的概念也适用。

1.4 Graphics Pipeline

Every desktop computer today has a powerful 3D graphics pipeline. This is a special software/hardware subsystem that efficiently draws 3D primitives in perspective. Usually, these systems are optimized for processing 3D triangles with shared vertices. The basic operations in the pipeline map the 3D vertex locations to 2D screen positions and shade the triangles so that they both look realistic and appear in proper back-to-front order.

如今，每台台式电脑都有一个功能强大的 3D 图形管线。这是一个特殊的软件/硬件子系统，可以高效地绘制透视三维基元。通常，这些系统经过优化，可处理具有共享顶点的三维三角形。管线中的基本操作是将三维顶点位置映射到二维屏幕位置，并对三角形进行阴影处理，使它们看起来既逼真又有适当的前后顺序。

Although drawing the triangles in valid back-to-front order was once the most important research issue in computer graphics, it is now almost always solved using the z-buffer, which uses a special memory buffer to solve the problem in a brute-force manner.

虽然按有效的前后顺序绘制三角形曾经是计算机图形学中最重要的研究课题，但现在几乎都是使用 z 缓冲区来解决这个问题，它使用一个特殊的内存缓冲区，以暴力方式解决问题。

It turns out that the geometric manipulation used in the graphics pipeline can be accomplished almost entirely in a 4D coordinate space composed of three traditional geometric coordinates and a fourth homogeneous coordinate that helps with perspective viewing. These 4D coordinates are manipulated using 4 × 4 matrices and 4-vectors. The graphics pipeline, therefore, contains much machinery for efficiently processing and composing such matrices and vectors. This 4D coordinate system is one of the most subtle and beautiful constructs used in computer science, and it is certainly the biggest intellectual hurdle to jump when learning computer graphics. A big chunk of the first part of every graphics book deals with these coordinates.

事实证明，图形处理流程中使用的几何操作几乎完全可以在 4D 坐标空间中完成，该坐标空间由三个传统几何坐标和有助于透视观察的第四个同质坐标组成。这些 4D 坐标使用 4 × 4 矩阵和 4 向量进行处理。因此，图形管线包含了大量用于高效处理和合成此类矩阵和矢量的机制。4D 坐标系是计算机科学中最精妙、最漂亮的结构之一，也是学习计算机图形学时需要跨越的最大智力障碍。每本图形学书籍的第一部分都有很大篇幅涉及这些坐标。

The speed at which images can be generated depends strongly on the number of triangles being drawn. Because interactivity is more important in many applications than visual quality, it is worthwhile to minimize the number of triangles used to represent a model. In addition, if the model is viewed in the distance, fewer triangles are needed than when the model is viewed from a closer distance. This suggests that it is useful to represent a model with a varying level of detail(LOD).

生成图像的速度在很大程度上取决于绘制三角形的数量。由于在许多应用中，交互性比视觉质量更为重要，因此值得尽量减少用于表示模型的三角形数量。此外，如果从远处观察模型，所需的三角形数量要少于从近处观察模型。这表明，用不同的详细程度（LOD）来表示模型是有用的。

1.5 Numerical Issues

Many graphics programs are really just 3D numerical codes. Numerical issues are often crucial in such programs. In the “old days,” it was very difficult to handle such issues in a robust and portable manner because machines had different internal representations for numbers, and even worse, handled exceptions in different and incompatible ways. Fortunately, almost all modern computers conform to the IEEE floating-point standard (IEEE Standards Association, 1985). This allows the programmer to make many convenient assumptions about how certain numeric conditions will be handled.

许多图形程序实际上只是三维数字代码。在这些程序中，数字问题往往至关重要。在 “过去”，由于计算机内部对数字的表示方法各不相同，更有甚者，处理异常的方法也各不相同，互不兼容，因此很难以稳健、可移植的方式处理此类问题。幸运的是，几乎所有现代计算机都符合 IEEE 浮点标准（IEEE 标准协会，1985 年）。这使得程序员可以就如何处理某些数字条件做出许多方便的假设。

Although IEEE floating-point has many features that are valuable when coding numeric algorithms, there are only a few that are crucial to know for most situations encountered in graphics. First, and most important, is to understand that there are three “special” values for real numbers in IEEE floating-point:

虽然 IEEE 浮点运算有很多特性，对数值算法编码很有价值，但对于图形学中遇到的大多数情况来说，只有少数几个特性是必须了解的。首先，也是最重要的一点是，要了解 IEEE 浮点中的实数有三个 “特殊 ”值：

1. Infinity (∞). This is a valid number that is larger than all other valid numbers.

2. Minus infinity (−∞). This is a valid number that is smaller than all other valid numbers.

3. Not a number (NaN). This is an invalid number that arises from an operation with undefined consequences, such as zero divided by zero.

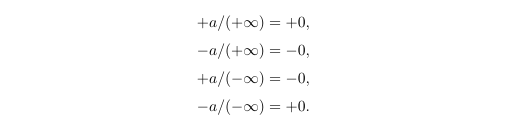
1. 无穷大（∞）。这是一个比所有其他有效数字都大的有效数字。

2. 无穷小 (-∞)。这是一个比所有其他有效数字都小的有效数字。

3. 不是一个数（NaN）。这是一个无效数，由一个具有未定义后果的运算产生，例如零除以零。

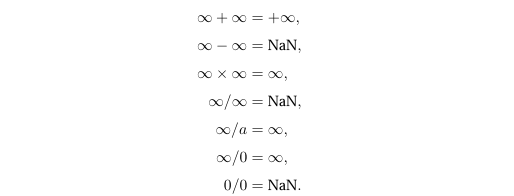
The designers of IEEE floating-point made some decisions that are extremely convenient for programmers. Many of these relate to the three special values above in handling exceptions such as division by zero. In these cases, an exception is logged, but in many cases, the programmer can ignore that. Specifically, for any positive real number a, the following rules involving division by infinite values hold

IEEE 浮点运算的设计者做出了一些对程序员来说非常方便的决定。其中许多都与上述处理异常（如除以零）的三个特殊值有关。在这些情况下，异常会被记录下来，但在很多情况下，程序员可以忽略不计。具体来说，对于任何正实数 a，以下涉及无限值除法的规则都是成立的



Other operations involving infinite values behave the way one would expect. Again for positive a, the behavior is as follows:

其他涉及无穷值的运算与我们预期的情况相同。同样，对于正 a，行为如下：



The rules in a Boolean expression involving infinite values are as expected:

1. All finite valid numbers are less than +∞.

2. All finite valid numbers are greater than −∞.

3. −∞ is less than +∞.

The rules involving expressions that have NaN values are simple:

1. Any arithmetic expression that includes NaN results in NaN.

2. Any Boolean expression involving NaN is false

布尔表达式中涉及无限值的规则与预期的一样：

1. 所有有限有效数字都小于 +∞。

2. 所有有限有效数字都大于 -∞。

3. -∞小于 +∞。

涉及具有 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. 任何包含 NaN 的布尔表达式都是false。

Perhaps the most useful aspect of IEEE floating-point is how divide-by-zero is handled. For any positive real number a, the following rules involving division by zero values hold:

IEEE 浮点运算最有用的方面可能是如何处理除零值。对于任何正实数 a，以下涉及零值除法的规则都是成立的



There are many numeric computations that become much simpler if the programmer takes advantage of the IEEE rules. For example, consider the expression:

如果程序员利用 IEEE 规则，许多数字计算都会变得简单得多。例如，请看表达式



Such expressions arise with resistors and lenses. If divide-by-zero resulted in a program crash (as was true in many systems before IEEE floating-point), then two if statements would be required to check for small or zero values of b or c. Instead, with IEEE floating-point, if b or c is zero, we will get a zero value for a as desired. Another common technique to avoid special checks is to take advantage of the Boolean properties of NaN. Consider the following code segment:

电阻和透镜就会出现这种表达式，（此处翻译可能有问题）。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 的值是否很小或为零。另一种避免特殊检查的常用方法是利用 NaN 的布尔属性。请看下面的代码段：



Here, the function f may return “ugly” values such as ∞ or NaN, but the if condition is still well-defined: it is false for a = NaN or a = −∞ and true for a =+∞. With care in deciding which values are returned, often the if can make the right choice, with no special checks needed. This makes programs smaller, more robust, and more efficient.

在这里，函数 f 可能返回 “丑陋 ”的值，如 ∞ 或 NaN，但 if 条件仍然定义明确：当 a = NaN 或 a = -∞ 时为假，当 a =+∞ 时为真。只要小心决定返回哪些值，if 通常就能做出正确的选择，而不需要特别的检查。这使得程序更小、更健壮、更高效。