

# Chapter 1 Introduction 简介

实时渲染是指在计算机上快速生成图像，它是计算机图形学中互动性最强的领域。屏幕上会显示一张图像，观察者在看到图像之后会做出一些反应和操作，这些反馈紧接着又会对下一张图像的生成产生影响。这个包含反应和渲染的循环会以一个很高的速度发生，以至于观察者根本意识不到自己正在观察一系列相互独立的图像，而是沉浸在这样一个动态的过程中。

通常会使用每秒显示的帧数（frames per second, FPS）或者赫兹（Hertz, HZ）来衡量图像显示的速率。如果图像以每秒一帧的速率进行显示，那么就几乎没有交互感可言，用户会意识到每一张新图像的到来，这是一个痛苦的过程。当图像显示速率达到 6 FPS 左右时，会逐渐开始增加交互性。电子游戏的目标是 30, 60, 72 或者更高的 FPS，在这样的图像显示速率下，用户可以将注意力集中在自身的行动和反应上。

电影放映机（movie projector）会以 24 FPS 的速率来进行显示，但是它会使用一个快门系统（shutter system）来将每帧重复显示 2–4 次，从而避免画面出现闪烁。这被称为刷新率（refresh rate），其单位是赫兹（Hz），它和上文中所说的显示速率（display rate）是两个概念。以刚才的电影放映机为例，一个可以每帧照亮三次的快门，其刷新率为 72 Hz。同样的，LCD 显示器的刷新率和显示速率也是两个不同的概念。

在显示器上观看以 24 FPS 出现的图片也许是可以接受的，但是更高的帧率可以有效降低最小反应时间。当显示延迟大于 15 毫秒的时候，就会对交互的流畅感产生干扰[1849]。举个例子，头戴的 VR 显示设备一般需要 90 FPS 来最小化延迟。

实时渲染也不仅仅只包含交互性，如果渲染速度是唯一的衡量标准的话，那么任何能够快速响应用户指令，并在屏幕上绘制图像的应用程序都符合这个条件。实时渲染通常指的是将三维场景渲染成二维图像。

交互性和三维场景是实时渲染的充分条件，但是第三个元素也已经逐渐成为了其定义的一部分，即图形加速硬件（graphics acceleration hardware）。许多人都认为，在 1996 年上市的 3Dfx Voodoo 1 图形加速卡是消费级显卡的开端[408]。随着这个市场的快速发展，如今每个电脑，平板和手机中都内置了相应的图形处理器。图 1.1 和图 1.2 展示了一些优秀的实时渲染案例，它们都使用了硬件加速来进行渲染：



图 1.1 《极限竞速 7》的游戏内画面。



图 2.2 《巫师 3》中的鲍克兰港。

图形硬件的进步推动了交互式计算机图形学领域的爆炸式发展。本文将重点关注用于提高渲染速度和图像质量的相关方法，同时也会介绍一些加速算法和图形 API 的特性，以及它们的局限性。我们不可能对每一个话题都进行深入，因此我们的目标是介绍关键性的概念和术语，介绍该领域中最健壮和最实用的算法，同时提供一些深入学习的方向指引。我们尝试提供一些工具，来帮助您更好地理解这个领域，同时希望这份尝试能够对得起您阅读本书所付出的时间和努力。

## 1.1 内容概述

下面是对各个章节的简要概述。

**第 2 章-图形渲染管线 (The Graphics Rendering Pipeline)**。实时渲染的核心是一组操作步骤，它将场景描述作为输入，并将其转换为我们能够看得见的图像。

**第 3 章-图形处理单元 (The Graphics Processing Unit)**。现代 GPU 使用固定功能单元和可编程单元的组合，来实现渲染管线的各个阶段。

**第 4 章-变换 (Transforms)**。变换是用于控制物体位置、朝向、尺寸、形状以及相机位置、相机视角的基本工具。

**第 5 章-着色基础 (Shading Basics)**。我们首先会讨论材质和光源的定义，以及它们如何用于实现所需的表面外观，无论这个表面是写实的还是风格化的。该章节还会介绍其他与外观表现相关的话题，例如使用抗锯齿、透明度和伽马校正来获得更高的图像质量。

**第 6 章-纹理 (Texturing)**。实时渲染中最强大的工具之一，就是能够快速访问图像，并将其显示在表面上。这个过程叫做纹理化，有各种各样来实现它的方法。

**第 7 章-阴影 (Shadows)**。在场景中添加阴影效果可以增强画面的真实感和表现力。我们会在这个章节中介绍几种目前比较流行的、能够快速计算阴影的方法。

**第 8 章-光和颜色 (Light and Color)**。在我们实现基于物理的渲染之前，我们首先需要了解如何对光和颜色进行量化。并且在我们执行基于物理的渲染之后，我们还需要将最终结果转换为可以显示的数值，并考虑屏幕属性和观察环境对它的影响。本章节将会介绍以上两个主题。

**第 9 章-基于物理的着色 (Physically Based Shading)**。我们将从头开始建立起对基于物理的着色模型的理解。在本章节中，我们首先会从潜在的物理现象开始，介绍一系列包含各种渲染材质的着色模型，最后介绍这些材质的混合方法和过滤方法，从而避免材质相互重叠并保持表面外观。

**第 10 章-局部光照 (Local Illumination)**。本章节研究了刻画复杂光源的算法。我们在进行表面着色的时候，需要考虑到光线是从某些物理对象中发射出来的，这些物体具有独特的形状。

**第 11 章-全局光照 (Global Illumination)**。本章节研究了模拟光线和场景多次相交的算法，这可以大大增强图像的真实感。我们还会研究环境光遮蔽 (ambient occlusion) 和定向遮蔽 (directional occlusion)，介绍在漫反射表面和镜面表面上渲染全局光照效果的方法，以及一些很有前景的统一方法。

**第 12 章-图像空间特效 (Image-Space Effects)**。图形硬件擅长进行高速的图像处理。在本章节中，我们首先会讨论图像过滤技术和重投影 (reprojection) 技术，然后我们将介绍几种常见的后处理特效：镜头光晕 (lens flare)，动态模糊 (motion blur) 和景深 (depth of field)。

**第 13 章–超越多边形 (Beyond Polygons)**。对于描述物体形状而言，三角形并不总是速度最快或者效果最逼真的方法，一些诸如基于图像，点云 (point clouds)，体素 (voxels) 或者其他样本集合的替代方法，都有着各自的独特优势。

**第 14 章–体积和半透明渲染 (Volumetric and Translucency Rendering)**。本章节的重点是体积材质的表示，以及体积材质与光线相互作用的理论与实践。它可以模拟的现象有很多，大到大范围的大气效果，小到毛发纤维的光线散射等。

**第 15 章–非真实感渲染 (Non-Photorealistic Rendering)**。尝试将一个场景渲染得更加逼真，只是众多渲染方式中的一种，除此之外还有很多其他风格，比如卡通渲染和水彩效果等。同时我们还会对直线和文本生成技术进行讨论。

**第 16 章–多边形技术 (Polygonal Techniques)**。几何数据的来源有很多，有时候我们需要对其进行修正，才能更好更快的进行渲染。本章节讨论了有关多边形数据表示和多边形数据压缩的相关内容。

**第 17 章–曲线和曲面 (Curves and Curved Surfaces)**。使用一些更加复杂的表面表达方式可以提供很多优势，例如可以在渲染质量和渲染速度之间进行权衡，可以具有更加紧凑的表示以及可以生成更加平滑的表面。

**第 18 章–管线优化 (Pipeline Optimization)**。对于一个已经使用了高效算法，并且正在运行的应用程序，我们还可以使用各种优化技术来进一步提高它的运行效率。本章节讨论了如何找应用程序的性能瓶颈 (bottleneck) 并对其进行处理，以及多线程优化等问题。

**第 19 章–加速算法 (Acceleration Algorithms)**。当我们让一个程序成功运行起来之后，下一步就是让它运行得更快。本章节讨论了各种各样的剔除技术 (culling)，以及层次细节 (LOD, level of detail) 等技术。

**第 20 章–高效着色 (Efficient Shading)**。场景中的大量光源会严重降低性能表现；在无法确定一个片元最终是否可见之前对其进行着色计算，也是重要的性能开销来源 (过度绘制)。本章节中我们会介绍很多方法，来解决着色过程中可能会出现低效率问题。

**第 21 章–虚拟现实和增强现实 (Virtual and Augmented Reality)**。这些领域有着特殊的挑战和技术，例如以一个较高且稳定的帧率，来高效的生成逼真的图像。

**第 22 章–相交测试技术 (Intersection Test Methods)**。相交测试对于渲染，用户交互和碰撞检测而言十分重要。在本章节中，我们介绍了许多用于几何相交测试的高效算法，并对其进行了深入讨论。

**第 23 章-图像硬件 (Graphics Hardware)**。本章节对一些硬件组件进行了重点关注，例如颜色深度，帧缓冲以及其他基本结构类型等。同时提供了一个具有代表性的 GPU 案例学习。

**第 24 章-展望未来 (The Future)**。预测未来的技术发展趋势，以及对读者的建议。

我们还完成了一章有关碰撞检测 (Collision Detection) 的内容，以及一章有关实时光线追踪的内容 (Real-Time Ray Tracing)，这里限于篇幅，我们将其放在了配套网站 [realtimerendering.com](http://realtimerendering.com) 上，你可以在这里下载到相关内容，同时网站上还有关于线性代数以及三角学的附录内容。

## 1.2 符号和定义

首先我们需要解释一下本书中所用到的数学符号。如果你想对本小节或者本书中的术语做更加深入的了解，你可以在 [realtimerendering.com](http://realtimerendering.com) 上获得我们的线性代数附录。

### 1.2.1 数学符号

表 1.1 中总结了大部分我们将用到的数学符号，这里我们将对其中的一些概念进行详细描述。

请注意表格中的规则也有一些例外，这主要是因为着色方程中所使用的符号，在相关文献中已经非常完善且统一了，例如  $L$  代表辐射度 (radiance)， $E$  代表辐照度 (irradiance)， $\sigma_s$  代表散射系数 (scattering coefficient) 等。

角度和标量都是取自于  $\mathbb{R}$  (实数集)，即它们都是实数。向量和点使用粗体的小写字母进行表示，其各个分量的表示如下：

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

该向量以列向量的形式给出，这种表达形式在计算机图形学中被广泛使用。在本书中的某些地方我们会使用行向量  $(v_x, v_y, v_z)$  来表示向量或者点，之所以不使用形式更加正确的  $(v_x, v_y, v_z)^T$ ，只是因为前者阅读起来更加容易。

类型	数学标记	例子
角度 (angle)	小写希腊字母	$\alpha_i, \phi, \rho, \eta, \gamma_{242}, \theta$
标量 (scalar)	小写斜体	$a, b, t, u_k, v, w_{ij}$
向量, 点 (vector, point)	小写粗体	$\mathbf{a}, \mathbf{u}, \mathbf{v}_s, \mathbf{h}(\rho), \mathbf{h}_z,$
矩阵 (matrix)	大写粗体	$\mathbf{T}(\mathbf{t}), \mathbf{X}, \mathbf{R}_x(\rho),$
平面 (plane)	$\pi$ : 一个向量和一个标量	$\pi: \mathbf{n} \cdot \mathbf{x} + d = 0,$ $\pi_1: \mathbf{n}_1 \cdot \mathbf{x} + d_1 = 0$
三角形 (triangle)	$\triangle$ + 三个顶点	$\triangle \mathbf{v}_0 \mathbf{v}_1 \mathbf{v}_2, \triangle \mathbf{cba}$
线段 (line segment)	两个顶点	$\mathbf{uv}, \mathbf{a}_i \mathbf{b}_j$
几何实体 (geometry entity)	大写斜体	$A_{OBB}, T, B_{AABB}$

表 1.1 本书中所用的公式总结

使用齐次 (homogeneous) 坐标表示法, 一个坐标可以使用四个值来进行表示, 即  $\mathbf{v} = (v_x \ v_y \ v_z \ v_w)^T$ , 其中  $\mathbf{v} = (v_x \ v_y \ v_z \ 0)^T$  代表一个向量,  $\mathbf{v} = (v_x \ v_y \ v_z \ 1)^T$  代表一个点。有时我们会使用只包含三个分量的向量或者点, 我们会尽量避免关于使用何种表示类型的歧义。对于矩阵运算而言, 使用相同符号形式的点和向量是十分有用的, 更多内容详见第 4 章中有关变换的部分。在某些算法中, 使用数字索引来代替  $x, y, z$  下标会很方便, 例如  $\mathbf{v} = (v_0 \ v_1 \ v_2)^T$ 。所有这些有关向量和点的符号规则, 同样也适用于只包含两个分量的向量, 在二维向量的情况中, 我们会直接跳过向量的第三个分量。

矩阵值得我们多进行一些解释。常用的矩阵尺寸包括  $2 \times 2$ ,  $3 \times 3$ ,  $4 \times 4$ , 这里我们将以  $3 \times 3$  矩阵  $\mathbf{M}$  为例, 来回顾矩阵的访问方式, 其他尺寸矩阵的操作也类似。矩阵  $\mathbf{M}$  的 (标量) 元素记为  $m_{ij}, 0 \leq (i, j) \leq 2$ , 其中的  $i$  代表该元素所在的行,  $j$  代表该元素所在的列, 如方程 1.1 所示:

$$\mathbf{M} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{02} \end{pmatrix} \quad (1.1)$$

方程 1.2 中的符号也代表一个  $3 \times 3$  矩阵，这种表达形式用于从矩阵  $\mathbf{M}$  中分离向量： $\mathbf{m}_{,j}$  代表第  $j$  个列向量； $\mathbf{m}_i$  代表第  $i$  个行向量（以列向量形式进行表示）。与向量与点一样，如果使用起来更加方便的话，列向量也可以使用  $x, y, z, w$  来进行索引：

$$\mathbf{M} = \begin{pmatrix} \mathbf{m}_{,0} & \mathbf{m}_{,1} & \mathbf{m}_{,2} \end{pmatrix} = \begin{pmatrix} \mathbf{m}_x & \mathbf{m}_y & \mathbf{m}_z \end{pmatrix} = \begin{pmatrix} \mathbf{m}_{0,}^T \\ \mathbf{m}_{1,}^T \\ \mathbf{m}_{2,}^T \end{pmatrix} \quad (1.2)$$

我们使用  $\pi : \mathbf{n} \cdot \mathbf{x} + d = 0$  来表示一个平面，它包含了定义平面所需的数学公式，即平面的法线  $\mathbf{n}$  以及标量  $d$ 。其中平面法线是一个描述平面朝向的向量，对于更一般的表面（例如曲面），法线描述了表面上某个特定点的朝向；而对于平面而言，平面上所有点都具有相同的法线。 $\pi$  通常被用作代表平面的数学符号，平面  $\pi$  会将空间一分为二，其中位于正半空间中的点满足  $\mathbf{n} \cdot \mathbf{x} + d > 0$ ；位于负半空间中的点满足  $\mathbf{n} \cdot \mathbf{x} + d < 0$ 。剩下所有的点都位于平面  $\pi$  上。

一个三角形可以使用三个顶点  $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$  来进行定义，记为  $\triangle \mathbf{v}_0 \mathbf{v}_1 \mathbf{v}_2$ 。

表 1.2 展示了其他的一些数学运算符及其符号表示，你可以在配套网站 [realtimerendering.com](http://realtimerendering.com) 上找到线性代数附录，其中包含了点乘、叉乘、行列式以及模长操作符的相关解释。转置操作符可以将一个列向量转换为一个行向量，反之亦然，这样我们就可以将一个列向量写在一行中，例如  $\mathbf{v} = (v_x \ v_y \ v_z)^T$ 。表中的第四个操作符在《*Graphics Gems IV*》[735]中有详细介绍，这是一个作用于二维向量的一元操作符，它作用于向量  $\mathbf{v} = (v_x \ v_y)^T$  上，并会生成一个与其垂直的向量，例如  $\mathbf{v}^\perp = (-v_y \ v_x)^T$ 。

序号	数学标记	说明
1	$\cdot$	点乘
2	$\times$	叉乘
3	$\mathbf{v}^T$	向量 $\mathbf{v}$ 的转置
4	$\perp$	一元操作符，垂直点乘操作符
5	$ \cdot $	矩阵的行列式
6	$ \cdot $	标量的绝对值

7	$\ \cdot\ $	范数（长度和模长）
8	$x^+$	将 $x$ 的最小值限制在 0
9	$x^\mp$	将 $x$ 限制在 0 到 1 之间
10	$n!$	阶乘
11	$\binom{n}{k}$	二项式系数

表 1.2：一些数学操作符的标记

我们使用  $|a|$  来表示标量  $a$  的绝对值，使用  $|\mathbf{A}|$  来表示矩阵  $\mathbf{A}$  的行列式。有时我们还会使用  $|\mathbf{A}| = |\mathbf{a} \ \mathbf{b} \ \mathbf{c}| = \det(\mathbf{a}, \mathbf{b}, \mathbf{c})$  这种表示方式，其中  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  分别是矩阵  $\mathbf{A}$  的列向量。

第 8 和第 9 个操作符是限制操作符（clamp），它在着色计算中经常使用。操作符 8 会将输入值的负数部分限制到 0：

$$x^+ = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise,} \end{cases} \quad (1.3)$$

操作符 9 则会将输入值限制在 0 到 1 之间：

$$x^\pm = \begin{cases} 1, & \text{if } x \geq 1, \\ x, & 0 < x < 1, \\ 0, & \text{otherwise,} \end{cases} \quad (1.4)$$

操作符 10 是阶乘（factorial）操作符，其定义如下所示，请注意  $0! = 1$ ：

$$n! = n(n-1)(n-2) \cdots 3 \cdot 2 \cdot 1 \quad (1.5)$$

操作符 11 是组合数，也叫做二项式系数，其定义[方程 1.6](#)：

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad (1.6)$$

除此之外，我们一般将  $x = 0$ ， $y = 0$ ， $z = 0$  这三个平面叫做坐标平面（coordinate planes）或者轴对齐平面（axis-aligned planes）。将  $\mathbf{e}_x =$

$(1 \ 0 \ 0)^T, \mathbf{e}_y = (0 \ 1 \ 0)^T, \mathbf{e}_z = (0 \ 0 \ 1)^T$  叫做主轴 (main axes) 或者主方向 (main direction)；或者分别叫做  $x$  轴,  $y$  轴和  $z$  轴。这组向量通常也会被称为标准基 (standard basis)。除了特殊说明之外, 我们将会使用标准正交基 (即由相互垂直的单位向量所组成的基底)。

我们将同时包含  $a, b$ , 以及两者之间所有数字的范围区间记为  $[a, b]$ 。如果我们只想要  $a, b$  之间的数字, 而不想要  $a, b$  本身的话, 那么我们可以将其记为  $(a, b)$ 。我们也可以将开闭区间进行组合使用, 例如:  $[a, b)$  代表包括  $a$  在内, 但是不包括  $b$  在内的,  $a, b$  之间的所有数字。

序号	函数	描述
1	$\text{atan2}(y, x)$	二元反正切函数
2	$\log(n)$	$n$ 的自然对数

表 1.3: 一些特殊的数学函数的表示方法

$\text{atan2}(y, x)$  是一个 C 语言中的数学函数, 它在本文中经常使用, 因此值得我们去关注一下。它是数学函数  $\arctan(x)$  的一个拓展, 它俩的主要区别在于  $-\frac{\pi}{2} < \arctan(x) < \frac{\pi}{2}$ , 而  $-\pi \leq \text{atan2}(y, x) \leq \pi$ ; 并且后者包含一个额外的参数输入。这个函数的常见应用是用来计算  $\arctan(y/x)$ , 当  $x = 0$  时, 分母就为 0 了。而拥有两个参数的  $\text{atan2}(y, x)$  则可以避免这一点。

在本书中,  $\log(n)$  始终代表了自然对数, 即  $\log_e(n)$ , 而不是以 10 为底的对数  $\log_{10}(n)$ 。

颜色使用一个三维向量来进行表示, 例如  $(red, green, blue)$ , 其中每个分量的范围都是  $[0, 1]$ 。

## 1.2.2 几何定义

几乎所有图形硬件使用的渲染图元 (primitive, 也叫做 drawing primitives) 都是点、线和三角形。

我们所知道的唯二例外就是 Pixel-Planes[502], 它可以绘制球体; 以及 NVIDIA NV1 芯片, 它可以绘制椭球体。

在本书中, 我们会将一个几何实体 (geometric entities) 的集合称为模型 (model) 或者物体 (object)。场景 (scene) 是指环境中所有待渲染模型的集

合，同时场景中还包含了材质信息，灯光信息，以及观察信息等。

这里的物体可以是一辆车，一栋建筑甚至是一条直线。在实际中，一个物体中包含了一系列的渲染图元，但是也有例外，物体也可以是其他更加高级的几何表现形式，例如 Bezier 曲线 (Bezier curves)、Bezier 曲面或者是细分曲面 (subdivision surface)。同时，一个物体也可以同时包含其他的物体，例如一辆车包含了四个车门以及四个轮子等。

### 1.2.3 着色

按照约定俗成的计算机图形学惯例，本书中的“着色 (shading)”和“着色器 (shader)”以及相关的派生词，常常被用来指向两个相关但是完全不同的概念：一个是计算机生成的视觉外观，例如：“着色模型 (shading model)”，“着色方程 (shading equation)”，“卡通渲染 (toon shading)”等；另一个是渲染系统中的可编程组件，例如：“顶点着色器 (vertex shader)”，“着色器语言 (shading language)”等。在这两种不同的情况下，你可以通过上下文来推断出它具体指向的含义。

## 深入阅读和资源

我们能够给你提供的、最重要的资源，就是本书的配套网站：

[realtimerendering.com](http://realtimerendering.com)，其中包含了最新信息的链接以及每章相应的网站。实时渲染的研究领域也是实时变化的，在本书中，我们试图关注那些最基本的概念，以及那些不太可能过失的技术。在这个网站上，我们可以展示与当今软件开发有关的信息，并且我们有能力将其进行不断更新。