VR三维技术系列

GLSL

渲染编程基础与实例

（C#版本）

赵　辉　楚含进　王晓玲　编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

VR 三维技术系列

GLSL 渲染编程基础与实例（C#版本）

赵　辉　楚含进　王晓玲　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书介绍了用 GLSL 语言进行三维渲染的方法，以及用大量的实例来展示如何进行 GLSL 编程。本书详细讲述了 GLSL 渲染流程；GLSL 着色器编程；顶点光照；像素光照；卡通渲染、影线渲染、分形渲染、Gooch 渲染等非真实感渲染的实现；三维噪声的生成，以及噪声在云彩、木头纹理、大理石等渲染特效中的应用；棋盘、砖墙、Toyball 等基于过程的渲染特效的实现；各种特殊光照效果渲染实现；通过 GLSL 进行图像处理的算法及实现。本书的特点是以各种渲染实例为核心，通过本书的学习，可以快速掌握 GLSL 语言的编程。

本书不仅可以作为数字媒体技术专业的专业基础课教材，还可以作为计算机学科和软件工程学科“数据结构和算法”、“计算机图形学”等课程的教材和参考书。

图书在版编目(CIP)数据

GLSL 渲染编程基础与实例：C#版本/赵辉，楚含进，王晓玲编著. —北京：电子工业出版社，2017.7
(VR 三维技术系列)
ISBN 978-7-121-31683-8

Ⅰ. ①G… Ⅱ. ①赵… ②楚… ③王… Ⅲ. ①三维动画软件－程序设计 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2017）第 120543 号

策划编辑：张 迪
责任编辑：底 波
印 刷：中国电影出版社印刷厂
装 订：三河市良远印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：11.5 字数：294 千字
版 次：2017 年 7 月第 1 版
印 次：2017 年 7 月第 1 次印刷
定 价：59.00 元

所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：(010)88254469，zhangdi@phei.com.cn。

序

2015 年以来，虚拟现实技术的应用在国际国内发展很快。教育、医疗、娱乐、影视、游戏、安全、交通等各行各业都对虚拟现实技术进行了大量应用。虚拟现实技术的基础和核心是三维计算机图形学，分为四大模块：建模、渲染、动画、交互。目前国内大量的虚拟现实应用都局限于在西方开发的虚拟现实引擎的技术上进行开发的上层应用。我们这套丛书着重底层核心技术的讲解，三维计算机图形学在知识结构上来说需要数学、物理、工程、计算机编程、艺术五个方面。设计建模、渲染等算法需要微分几何、线性代数、概率统计等数学知识的理解和掌握；动画模拟需要流体、刚体等物理知识的理解和掌握；把这些数学、物理理论变为程序需要极强的编码能力，也就是从理论到实践的工程能力；三维图形学的最终表现形式是视觉上可看得到的，因此也需要良好的艺术修养和审美。虚拟现实和它所依赖的三维计算机图形学特别适合锻炼并能够融会贯通学生的数学、物理、工程、编程和艺术能力。三维计算机图形学是一个跨学科的领域，三维图形学处理的是三维模型数据，学生在这个领域中学到的数学建模、工程等能力，也可以用到其他行业，如人工智能等，对其他行业的大数据进行分析和处理。

2008 年以来，全国各个高等院校纷纷在各自软件工程学科专业的基础上开设了数字媒体技术专业。数字媒体技术专业和计算机科学专业的区别是，前者主要是着重学习二维图像和三维图形相关的算法和应用开发，而后者还需要学习其他计算机科学相关的知识。由于开设和建立时间短，各学校的数字媒体技术专业的教学工作都还处在摸索阶段，也没有形成统一、成熟的教材体系。根据在数字媒体技术专业多年的教学实践经验，我们总结出本专业要以计算机三维图形学的理论和算法为基础，以三维应用开发为导向进行建设。

根据多年一线教学经验与反馈，以及当前的三维图形学研究成果，我们编写了本套丛书。本套丛书涵盖了三维图形学算法的三个方面：建模、动画和渲染。内容根据数字媒体技术专业的教学特点分散到 5 本 VR 三维技术系列图书中。通过本系列专业图书，再加上已有的成熟的计算机基础编程教材，以及三维软件使用的教材，就可以完整地覆盖数字媒体技术专业的所有课程。

书里的代码采用 C#编程语言。C#编程语言是一种结合了 C ++ 和 Java 优点的编程语言。C#语言相对于其他编程语言来说比较容易学习和掌握，但是本套丛书里讲述的原理和算法不仅限于 C#语言。读者可以通过示例中的代码，采用自己熟悉的编程语言来进行编程。本套丛书包含了很多计算机图形学会议 Siggraph 论文里最新的、核心的、关键突破和进展的图形学算法讲解、实现和分析。

前　言

虚拟现实应用离不开逼真的渲染。渲染分为实时渲染和非实时渲染两大类。在影视特效中，常用到的是非实时渲染。而在游戏等应用中，需要能够实时显示的渲染技术。非实时的渲染通过对光照进行物理模拟，从而达到和相机拍摄无法区分的效果。但由于要进行光线追踪等计算，这种算法耗时很长。而实时渲染由于对物理光照进行了大量的简化，从而可以很快速地进行计算。

在 GPU 上进行实时渲染，是目前成熟的解决方案。需要用特定的编程语言来对 GPU 进行编程，从而使 GPU 能执行设计好的光照公式。对于不同的光照程序，可以得到不同的渲染效果。如果只用 OpenGL 进行渲染，那么就受限于 OpenGL 内置的渲染效果。GPU 渲染是三维游戏、虚拟现实场景等应用中一个最重要的核心模块。本书提供了 GPU 编程的基础、代码和实例。

本书介绍了用 GLSL 语言进行三维渲染的方法，以及用大量的实例来展示如何进行 GLSL 编程。本书详细讲述了 GLSL 渲染流程；GLSL 着色器编程；顶点光照；像素光照；卡通渲染、影线渲染、分形渲染、Gooch 渲染等非真实感渲染的实现；三维噪声的生成，以及噪声在云彩、木头纹理、大理石等渲染特效中的应用；棋盘、砖墙、Toyball 等基于过程的渲染特效的实现；各种特殊光照效果渲染的实现；通过 GLSL 进行图像处理的算法及实现。本书的特点是以各种渲染实例为核心，通过学习本书的内容，可以快速掌握 GLSL 语言的编程。

本书不仅可以作为数字媒体技术专业的专业基础课，还可以作为计算机学科和软件工程学科“数据结构和算法”、“计算机图形学”等课程的教材和参考书。需要书中部分代码的读者，可发邮件向作者索取，邮箱地址：graphicsresearch@ qq. com。

赵　辉

2017 年 5 月于美国哈佛大学

作者简介

赵辉，虚拟现实专家、清华大学丘成桐数学科学中心访问学者、哈佛大学访问学者。主要研究计算微分几何、拓扑、三维模型处理算法（三维模型简化、细分、分割、变形、光滑、参数化、向量场、四边形化等）、三维动画算法（骨骼动画、蒙皮算法）、渲染算法（非真实感渲染、实时渲染、基于物理渲染），以及三维技术在3D打印、虚拟现实、增强现实、三维游戏、手机游戏、影视特效等的应用。

楚含进，现任 AMD 中国区 VR 与计算平台总监，负责图形处理器（GPU）技术在虚拟现实（VR）中的应用，以及游戏设计、计算机图形与仿真等技术领域的应用和合作，是国内 VR 产业早期从业者，为国内 VR 媒体撰写各类有关文章。同时，在异构计算领域，推动将 GPU 异构计算用于机器学习、计算机视觉领域。曾带领团队先后将 GPU 异构计算贡献于 OpenCV，以及 Caffee、MLP 等开源项目，主导引进并支持多个有关 OpenCL 计算的书籍。

王晓玲，北京科技大学教授，美国西北大学、哈佛大学访问学者，有限元模拟、机械仿真，物体相变、生物材料分析、三维打印材料专家。

目　录

第 1 章 GPU 与图形应用编程介绍

1.1 GPU 发展史与 Shader

提到 Shader 编程就必须讲述 GPU 的历史，尤其伴随着当代虚拟现实（VR）技术的兴起，GPU 作为图形处理器更是 VR 系统的核心，对 GPU 的了解是编写 VR 程序的必备之路。谈及 GPU 的发展史，每个读者都有自己的坐标，网上也有 GPU 的历史和各厂家的产品介绍等相关信息。今天我们从 GPU 与 Shader 可编程的角度来了解 GPU 的发展。

让我们把目光放在 GPU 产生之后的日子里。早期的显卡只是为显示输出而设计的专用加速器，而 GPU 的概念最早在 1999 年由 NVidia 提出，随着显卡芯片的技术变革和飞速发展，其用途和概念已经超出了功能单一的图形硬件加速器，渐渐向使用灵活功能强大的通用计算处理器进化。这一举动标志着 GPU 在 PC 主机里地位的不断上升。

OpenGL 是出现于 20 世纪 90 年代初的专业图像 API，并很快成为了个人计算机领域图像发展的主导力量和硬件发展的动力。虽然 OpenGL 的影响带起了广泛的硬件支持，但在当时用软件实现的 OpenGL 仍然很普遍。20 世纪 90 年代末，DirectX 开始受到 Windows 游戏开发商的欢迎。不同于 OpenGL，微软坚持提供严格的一对一硬件支持。这种做法使得 DirectX 身为单一的图形 API 方案并不得人心，因为许多的 GPU 也提供自己独特的功能，而当时的 OpenGL 应用程序已经能满足它们，导致 DirectX 往往落后于 OpenGL 一代。随着时间的推移，微软开始与硬件开发商展开更紧密的合作，并开始重视 DirectX 的发布与图形硬件的支持。

Direct3D 5.0 是第一个增长迅速的 API 版本，而且在游戏市场中迅速获得普及，并直接与一些专有图形库竞争，而 OpenGL 仍保持重要的地位。Direct3D 7.0 支持硬件加速坐标转换和光源（T&L）。此时，3D 加速器由原本只是简单的栅格器发展到另一个重要的阶段，并加入 3D 渲染流水线。硬件坐标转换和光源（两者已经是 OpenGL 拥有的功能）于 20 世纪 90 年代在硬件中出现，为往后更为灵活和可编程的像素着色引擎和顶点着色引擎设置了先例。2000 年后，随着 OpenGL API 和 DirectX 类似功能的出现，GPU 增加了可编程着色的功能。现在，每个像素都可以经由独立的小程序处理，当中可以包含额外的图像纹理输入，而每个几何顶点同样可以在投影到屏幕上之前被独立的小程序处理。如果说 GPU 名词的出现体现出业界对显卡发展的方向和态度，那么 ATI（AMD Radeon 显卡所在公司的前身）在 2002 年发布的首款支持具有划时代意义的 DirectX 9 图形 API 的 R300 系列 GPU（即大名鼎鼎的 Radeon 9700 和 9500 系列）从根本上将 GPU 推上了前进的轨道。它是世界上首个 Direct3D 9.0 加速器，像素和顶点着色引擎可以运行循环和长时间的浮点运算，就如 CPU 般灵活，并达到了更快的图像数组运算。像素着色通常被用于凹凸纹理映射，使对象通过增加

纹理呈现更加丰富的明亮、阴暗、粗糙，或是偏圆及被挤压效果。虽然 DirectX 8 率先提出了 GPU 可编程概念，从而引出了最早的 Vertex Shader 与 Pixel Shader，但真正使两者发扬光大并大规模应用的是 DirectX 9。也是从这时候开始，微软开始逆袭居统治地位长达 10 年之久的 OpenGL，由此可见，可编程 GPU 在 GPU 发展史上的重要意义，自此 3D 图形应用发展到了一个崭新的时代，而 DirectX 9 的统治时间更是长达 7 年之久，ATI 之后发布的 R400 和 R500 系列一直将优势保持到了 2006 年。在这一年，微软随着颇受争议的 Windows Vista 发布了 DirectX 10，由于这个系统推出得过于仓促及不合时宜，Windows Vista 和 DirectX 10 都没有得到广泛的应用。但 ATI 还是率先发布了支持 DirectX 10 的 R600 系列，这个版本主要为开发者带来了强大的 Geometry Shader（几何着色）及一些先进新特性，为 3D 图形编程又带来了新的元素。但随后 NVidia 同样发布了著名的支持 DirectX 10 的 GTX 8800 系列，暂时夺回了 GPU 性能宝座。正如前面提到的，DirectX 10 并没能取得成功，只是一个过渡产品。

不久，随着微软在 2009 年发布了 Windows 7，与之捆绑的 DirectX 11 走入了人们的视线，它也成为 DirectX 9 的终结者。DirectX 11 为 GPU 编程带来了众多新特性，Tessellation Shader、Compute Shader 及多线程支持。此时的 ATI 已被 AMD 收购，同年 AMD 又一次率先发布了支持 DirectX 11 的 Radeon HD 5000 系列，AMD 再次利用技术上的创新和优势引领 GPU 的发展，在 Radeon HD 5000 的整个生命周期里都没有遇到任何挑战。2010 年 AMD 又发布了 Radeon HD 6000 系列，继续巩固着胜果。就在此时，一个小插曲开始出现，故事源于 DirectX 11 里一个由 AMD 提出的全新技术 Tessellation。从广义上讲，Tessellation Shader 的设计与 DirectX 10 引入的 Geometry Shader 有相似之处，它们的本质都是在已有模型的基础上生成更多的图元或三角形，从而达到特定的效果。这两者为当代 GPU 编程流水线增色添彩，为开发者提供了更多的选择。但由于要实现这一功能开销很大，需要在 GPU 的设计中加入大量的硬件处理单元来生成三角形。而 Tessellation 的用途却比较有限，一般应用在使用 Displacement mapping 来生成地形等应用中。从前面的介绍可以看出，Tessellation 是一项功能单一却开销很大的技术，这也是为何要配备大量硬件处理单元进行计算的原因（道理等同于硬件编解码）。这一技术后来被 AMD 的对手 NVidia 利用，着力加强对 Tessellation 的利用，在 GPU 设计中加入大量的三角形生成与处理单元，这样就可以在处理图形渲染过程中最基本的元素三角形时获得巨大优势，这一点在游戏巫师 3 和古墓 10 中体现尤其明显，两者都使用了大量的 Tessellation。这一点与 AMD 显卡的技术路线完全不同，此时 AMD 已经开始研究为下一代 DirectX 12 带来事实标准的 Mantle。无奈 NVidia 的市场和跟随策略非常成功，而 AMD 内部动荡，这一时期一直是 NVidia 占据着市场的主导位置。尽管如此，AMD 随后推出的 Mantle 标准技术也非常成功，促使了 DirectX 12，OpenGL 的下一代 Vulkan 甚至苹果公司的 Metal 图形接口标准的出现。依旧坚定地推动 GPU 的发展，为 GPU 不断注入新鲜的活力。伴随着 Mantle，2012 年，AMD 推出了新一代架构 GCN，旨在提升 GPU 的通用计算性能，这一理念与 DirectX 11 设计一脉相承，引入了 ACE（Asynchronous Compute Engine），要知道这就是后来带动了 DirectX 和 OpenGL 变革的核心技术，也是 DirectX 12 和 Vulkan 为开发者带来的最大甜品。无独有偶，2014 年 Oculus 被 Facebook 收购，虚拟现实一夜之间成为下一代互联网计算平台，但是虚拟现实所需要的图形处理能力需要指数级的增长，AMD GPU 的 ACE 架构将 VR 对图形处理能力的需求在不损失质量的情况下降低了一个级别，这也就是为什么 Oculus 的 VR 头显对 AMD 的显卡支持明显要求低的原因。

Compute Shader 是 DirectX 11 的最重量级特性，它的灵活性和强大功能远超 GPU 渲染流水线中的其他成员，它的引入使得以往不可能的特效变为了现实，通过物理计算来模拟现实世界中的各种效果，比如毛发渲染、布料渲染、草地渲染、光线追踪、物理碰撞等。从技术上来说，Compute Shader、异步处理单元等都代表着未来的图形技术发展方向，未来 GPU 将越来越强调通用计算性能，GPU 越来越像 CPU。无奈过去几年 AMD 在软件、工具等方面并没有提供足够的用户体验给开发者，以至于流失很多开发者转向 NVidia 的显卡，但不能否认 AMD 的显卡架构师对未来图形处理的远见卓识。经过几年调整，AMD 在 CPU 上刚刚发布 AMD 锐龙 Ryzen 系列，10 年后回归 CPU 主场。同时，在 2017 年，AMD 还将发布全新 Vega 架构系列 GPU，事隔 5 年后，AMD 再一次大刀阔斧地对 GPU 架构进行变革，引入 NCU（Next – generation Compute Unit），并对流水线进行大幅改动。未来 GPU 发展趋势如何，让我们拭目以待。

1.2　GLSL Shader 编程在图形设计中的作用

首先了解 GPU 进入可编程时代的发展史。

Shader 其实就是一段执行在 GPU 上的程序，此程序使用 OpenGL ES SL 语言来编写。最早的 GPU 仅仅由几种不同的硬件处理单元组成，进行最基本的 3D 图形加速功能。从 2003 年发布的 DirectX 8.0 和 OpenGL 1.5 开始引入了可编程特性。最早被提出的 GPU 渲染中最核心的两个组件是 Vertex Shader 和 Fragment Shader。2004 年发布的 OpenGL 2.0 则伴随着全新的 GLSL 标准为 GPU 编程带来了划时代的意义。GLSL 由当时的 GPU 巨头 3DLabs 提出并完善。当时的 GPU 为开发者提供了两种不同类型的可编程处理器：Vertex Processor 和 Fragment Processor，基于这两种处理器，开发者有能力开发出比 Fix – Function 流水线更加丰富的特效。

此后，OpenGL 陷入了长达 6 年的停滞期。在此期间微软发布了 DirectX 9 和 DirectX 10，引入 HLSL 和一系列高级特性，将 OpenGL 甩在身后。大量游戏转投微软阵营，一时间基于 OpenGL 的游戏除了 Doom 和 Quake 系列几乎绝迹。而 DirectX 10 更是将一直分离的 Vertex Processor 和 Fragment Processor 进行统一，ATI 于 2006 年提出了 Unified Shader Model 概念，不同的 Shader 编程全部由 Streaming Processor 来进行计算，同时又提出了一个全新的 Geometry Shader，这也是 GPU 通用计算的雏形。随后的 OpenGL 3.0 只是在前者上修修补补，没有太多新意。

直至 2010 年 OpenGL 4.x 的发布，OpenGL 才在特性上追上了微软的脚步，同时微软提出了大名鼎鼎的 DirectX 11，直到今天它还是被应用最广泛的桌面图形 API，它的统治时间更是长达 8 年之久。随着 DirectX 11 的出现，GPU 架构的硬件特性又达到了一个新高度，由 AMD 提出的 Tessellation Shader 和 Compute Shader 以及 CPU 多线程渲染被引入。OpenGL 虽然也在 4.0 版和 4.3 版分别加入了 Tessellation Shader 和 Compute Shader 的支持，但至今仍不支持 CPU 多线程渲染特性。

这一情境在全新图形 API Vulkan 面世后被终结，微软和 Khronos 组织分别于 2015 年和 2016 年发布了全新的 DirectX 12 和 Vulkan，这两种 API 都秉承了 AMD Mantle 的设计理念，大幅降低了驱动开销，支持多线程渲染，并支持由 AMD 提出的 Asynchronous Compute Engine 异步计算引擎。目前 GLSL 的最新版本为 4.5，Vulkan 版本的 GLSL 的标准也正在制定中，相信不久的将来便会有完整的文档出现（目前开发者可参考 OpenGL 版的 GLSL 来进行 Sha-

der 编程，它们的绝大多数用法是一致的）。

接下来，我们来看看 Shader 编程在图形设计中的作用。

通过前面的介绍我们可以看出，Shader 编程在图形设计中的重要性越来越高，它决定了图形效果的丰富程度和性能表现。

简单来说，图形程序的设计理念是让 GPU 承担尽可能多的计算任务，从而将 CPU 解放出来，之所以这样做有以下几个原因。首先 GPU 的架构设计与图形计算中的任务高度并行相吻合，而从硬件设计角度讲，扩充 GPU 的宽度（流处理器个数）要远比扩充 GPU 的高度（流处理器频率）容易，道理与 CPU 向着多核方向发展相同，都是在无法大幅提升主频的情况下通过增加宽度来提高处理器的性能。其次 GPU 在 PC 中的地位除了进行图形运算外还负责图像输出，也就是我们从显示器上看到的一切都是经由 GPU 处理完成并呈现的。我们知道 CPU 与 GPU 享有着不同的存储空间，CPU 的存储空间就是我们常见的内存，而 GPU 的存储空间则是其自身配备的调整显存。在计算机的硬件体系中，两者间的数据交互需要通过 PCIE 总线，虽然这个高级的总线在 PC 中的传输速度出类拔萃，但相比内存与显存来说，这显然是瓶颈。因此，在程序的设计中要尽可能避免或减少内存与显存之间的数据传输。如今的图形计算流程是由 CPU 准备数据的，这些数据可能存储在速度相对较慢的外部存储器中，如固态硬盘或磁盘，之后再将数据从内存传递到显存中，由 GPU 进行计算处理最终输出到显示终端。通过分析这个处理流程我们可以看出，在整个过程中涉及一次数据从内存到显存的传输，这是我们能够做到的最优方案，虽然这个时间也会比较长，但我们可以通过游戏启动时的加载过程或通过多线程预加载来完成，也就是说这个时间不会影响到游戏运行时的处理速度。这也就解释了为什么我们希望 GPU 来尽可能多地完成计算处理，这样就不需要在游戏中将 GPU 的数据从显存传回内存，待 CPU 处理完成后再返回到显存的漫长过程了。

看到这里我们应该明白 Shader 编程在图形设计中的作用了，我们希望能够使 GPU 形成一个单独的自治区域，所有的任务都高度自治，GPU 的重要性不言而喻。AMD 的图形技术发展也在逐步印证这一点，从最初的 Vertex Shader 和 Fragment Shader（Pixel Shader），到后来逐步完善的 Geometry Shader、Tessellation Control Shader（Hull Shader）和 Tessellation Evaluation Shader（Domain Shader），以及现在的 Compute Shader。这一系列特性的加入都是围绕着使 GPU 成为独立计算单元的目标进行的。

下面让我们来一一介绍这些 Shader 的作用及地位。

Vertex Shader 是最常见的历史最悠久的 Shader，它主要用于将 3D 模型的顶点坐标从 3D 虚拟空间转变成 2D 屏幕坐标和深度缓冲。Vertex Shader 是图形流水线的第一站，也是每个流水线必须包含的阶段。Vertex Shader 是必然存在的。Vertex shader 主要完成以下工作：基于点操作的矩阵乘法位置变换；根据光照公式计算每点的 color 值；生成或者转换纹理坐标。

Tessellation Shaders 是在 OpenGL 4.0 和 DirectX 11 中引入的，它们包含了两个阶段，分别是 Tessellation Control Shader 及 Tessellation Evaluation Shader。它们可以在运行时生成大量的细分三角形来使一个粗粒度的三角形模型变成更细粒度的三角形模型。一般的使用场景包括根据距离远近使用 LoD（Level of Details）优化性能、通过 Displacement Mapping 和 Height Map 生成地形等。这两个 Shader 的位置处在 Vertex Shader 之后和 Geometry Shader 之前（如果图形流水线配备了 Geometry Shader，否则会被直接送入 Rasterizatoin 阶段，随后进入 Fragment Shader。Tessellation Shader 是可选的。

Geometry Shader 是在 OpenGL 3.2 和 DirectX 10 中引入的，在这个阶段，开发者可以控制在 Vertex Shader 输入的 3D 模型基础上生成一些新的图形元素，包括点、线段、三角形等。在某些简单的图元生成上作用与 Tessellation Shaders 类似。同时 Geometry Shader 还提供了顶点流处理的能力，可以通过这个阶段为顶点缓冲中的顶点进行索引并输出到一个特定的缓冲区。在 Geometry Shader 里，我们处理的单元是 Primative。虽然根本上都是顶点的处理，但进入 Vertex Shader 的是一次一个的顶点，而进入 Geometry Shader 的是一次一批的顶点，Geometry Shader 掌握着这些顶点所组成的图元信息。Geometry Shader 的处理阶段处于流水线的栅格化之前，也在视锥体裁剪和裁剪空间坐标归一化之前。虽说裁剪过程会剔除部分图元也会分割某些图元，但就目前来说，不会有其他流水线的可编程阶段会在 Geometry Shader 之后提供影响图元的性质（形式和数量）——这是 Geometry Shader 鉴于其位置的特殊性而拥有的一个重要特点。Geometry Shader 程序在 Vertex Shader 程序执行之后执行。

Fragment Shader 是除 Vertex Shader 之外另一个最常见的 Shader，它与 Vertex Shader 一同出现，处于 Rasterizatoin 阶段之后，是整个图形流水线中唯一的 2D Shader。Pixel Shader（像素着色器）就是众所周知的 Fragment Shader（片元着色器），它计算每个像素的颜色和其他属性。通过应用光照值、凹凸贴图，阴影，镜面高光，半透明等处理来计算像素的颜色并输出。它也可改变像素的深度（z - buffering）或在多个渲染目标被激活的状态下输出多种颜色。一个 Pixel Shader 不能产生复杂的效果，因为它只在一个像素上进行操作，而不知道场景的几何形状。它的处理对象是在屏幕的坐标上输出颜色值，结果将被呈现在显示器上。Fragment Shader 在整个图形流水线的 Shader 中是限制最少的，也是功能最强大的，一般较为复杂的计算将在这一阶段进行，来达到丰富输出效果的目的。

整个图形流水线的 Shader 就如同工厂流水线上的工人一样，每个不同的工种都被赋予了特定的权限和功能，它们在各自的环境下施展着不同的能力，因此它们的能力都是受限的。接下来我们来了解 Shader 中最具革命性的成员 Compute Shader。

Compute Shader 是在 OpenGL 4.3 和 DirectX 11 中引入的。它独立于图形流水线，可以存在于图形队列和计算队列的不同位置。这也是 GPU Shader 编程诞生以来出现的限制最少、功能最强大的 Shader。它的能力几乎等同于 GPGPU，也就是常说的 OpenCL。在 Compute Shader 中，开发者被赋予最大的权限来访问 GPU 中的各类资源，使用 Compute Shader 可以编写出灵活性极高、功能极为强大的算法，来辅助图形编程中的效果、物理计算、模拟等高级特性。而 AMD 的 TressFX（毛发渲染技术）就是一个非常成功的案例，TressFX 的算法设计就是通过 Compute Shader 对毛发运动效果的模拟来达到实时毛发渲染的目的。而物理模拟计算也将成为未来提升图形渲染水平的重要趋势，如布料、草、树叶及光线追踪等效果。在 GPU 飞速发展的今天，将会有越来越多的复杂物理效果被应用于实时图形应用中。AMD 引入 Computer Shader 的计算单元还有一个目的是最大限度保证 GPU 的性能发挥。现在市面上有很多硬件独立单元来做诸如物理开发，这在一段时期内是必然的，随着 GPU 的发展必然会过渡到 Computer Shader。

笔者对 AMD 的显卡较为熟悉，所以下面以 AMD 显卡为例为大家说明一些问题。通过前面的介绍，我们了解了 GPU 图形渲染的每个重要环节，以及 Compute Shader 所扮演的重要角色。而这一切也反映在 AMD GPU 设计的理念中。目前已经发布的 GPU 中，从第一代 GCN 开始，代号为 Tahiti、Hawaii、Fiji 的旗舰产品在 GPGPU 通用计算中的性能都处在绝对

的领先地位，而将在2017年发布的Vega 10架构会将通用计算的性能再推向下一个巅峰。

接下来我们就看看GCN架构为通用计算带来了哪些特性。

如图1-1所示为GPU架构进化的过程，从最早的Fixed Function设计到Radeon 9700时代的简单Shader模型，再到Radeon HD 5000 6000系列的VLIW5和VLIW4。最新的GCN架构相对VLIW4架构进行了彻底的颠覆革新。从图中我们可以看出VLIW4是一个超长指令架构，4个红色矩形代表了在每次ALU计算中包含4条不相关指令。这样的设计非常适合早期的纯图形Shader计算，Fragment Shader中的每个像素都完全独立，没有依赖关系，这样能够维持极高的VLIW4指令利用率，然而通用计算中由于分支很多，依赖关系复杂，这样的架构设计将导致大规模的资源浪费。

图1-1　GPU架构进化的过程

如图1-2所示很好地说明了这个问题，在VLIW4中，每个CU配备了16个ALU、64个流处理器及1个SIMD指令，每个ALU在每个周期可以对4个通道进行4次不同指令的计算，也正好对应着图形计算中的RGBA通道，因此非常适合图形计算。同时这样的架构非常依赖编译器来优化遍历每个周期所执行的指令尽可能达到4个。

图1-2　VLIW4和GCN架构

而 GCN 架构则完全推翻了之前的设计来为通用计算保驾护航。可以看出，每个 CU 配备了 4 个 ALU，可分别处理 4 个不同的 SIMD 指令，同样拥有 64 个流处理器。每个 SIMD 应用于 16 个流处理器，并且不再需要打包 4 个指令后再启动 ALU，这就给指令的发射带来了很大的灵活性，同时也不需要编译器做特殊的优化，全部是比较规整的细粒度操作。

GCN 架构还使每个 CU 里的 64 个流处理器同步执行，从而在使用 LDS（Local Data Share）时不需要进行任何同步操作，去除了同步操作的开销。

另外，如图 1-3 所示，GCN 架构的 GPU 还在每个 CU 里专门配备了 Integer Atomic Units 来高效地实现 LDS 原子操作的性能，Reduction 操作在通用计算中非常常见，而在很多场合中 Reduction 操作都需要原子操作来实现。

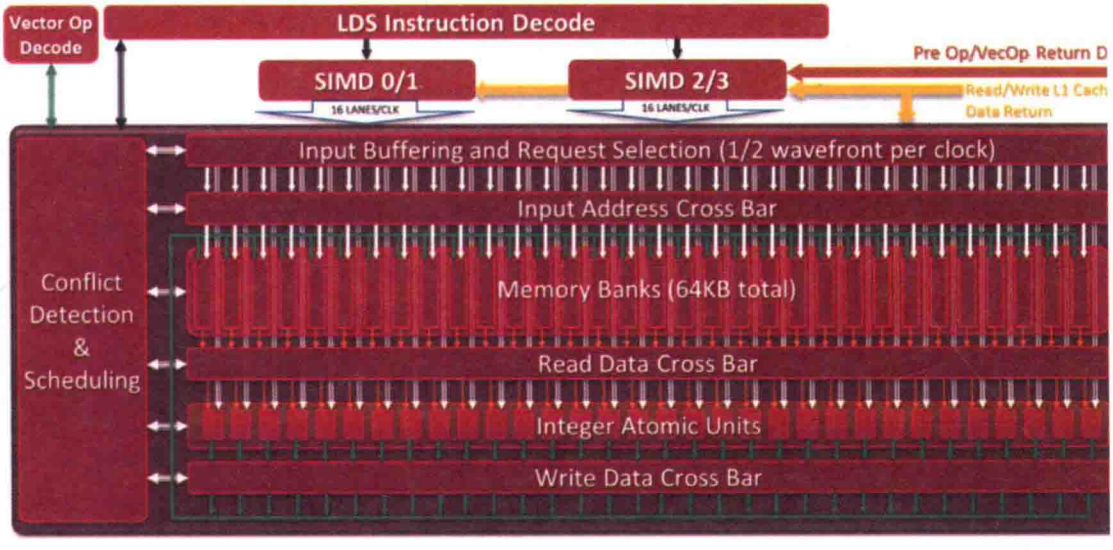

图 1-3　Integer Atomic Units

除此之外，GCN 架构的 GPU 还专门为 CPU 开辟了一块最大 256MB 的显存空间，CPU 可以高效地对这片存储空间进行写操作，这样做的好处是，CPU 可以高效地直接更新显存上的数据，同时保证 GPU 能以最高效的内存访问方式（即访问显存）来读取数据，从而节省一次内存到显存的复制。GPU 访问内存的效率通常不高，远远低于访问显存的性能，而这往往成为 GPU 计算的最大瓶颈（GPU 在进行通用计算中的瓶颈往往都来源于对显存和内存的访问）。

以上这些架构的重大变革被证实在通用计算中展示出巨大的实力，一经推出便所向披靡，横扫竞争对手，几年前兴起的挖矿热潮便是源于 GPU 的通用计算。直至今日，GCN 架构的通用计算能力仍无法被超越。

最后，我们来说一说 GCN 架构为 GPU 图形和计算带来的最重磅特性，ACE（Asynchronous Compute Engine）异步计算引擎。这项由 AMD 设计并实现的硬件引擎成为了 DirectX 12 和 Vulkan 引入的最大特性，并且在它出现 5 年之后，仍没有竞争对手能够实现。ACE 的设计思路是为了在图形渲染中能够更充分地利用流处理器等通用计算单元，由于流处理器只有在 Shader 执行的过程中才会被使用，而图形渲染过程中有大量的计算时间消耗在 GPU 的 Fixed Function 硬件处理单元上，如 Vertex Assembler、Input Assembler、Tessellator

和 Rasterization 等。2016 年，已经有众多顶级游戏大作支持了 DirectX 12 和 Vulkan，而 DirectX 12 的被接纳速度也成为有史以来之最。AMD GPU 在众多 DirectX 12 和 Vulkan 游戏中表现抢眼，在最新发布的狙击精英 4 中，AMD GPU 在 DirectX 12 模式下借助 ACE 特性取得了 20% ~30% 的性能提升。

Compute Shader 不仅拥有对资源的最大访问权限，如 LDS、原子操作、AMD 专用的 ShaderInstrinsic Function 以及 Work Item、Work Group、Local ID、Global ID 等索引信息，而且还提供了极大的资源访问灵活性，开发者可以通过参数设置来决定每个流处理器与计算任务的映射，控制 Workgroup 的大小和维度，从而控制 LDS 的使用量、Wavefront 的数量。这里每一个细节都对通用计算中 GPU 资源的利用率起到关键作用，这也给性能优化带来最大的可能性。

综上所述，GPU 编程发展的未来属于通用计算（即 Compute Shader），而 AMD 的 GPU 设计也一直在迎合这一趋势，将于 2017 年发布的 Vega 10 GPU 架构不仅对相对薄弱的 Geometry Pipeline、Deferred Shading 等进行增强，还引入了为增强通用计算性能而提出的次世代计算单元 NCU（Next - generation Compute Unit），能够以双倍于单精度浮点性能来执行半精度浮点计算。AMD 将一如既往地为推动 GPU 技术的进化与发展不懈努力。

1.3 游戏引擎的发展

游戏引擎是为视频游戏的开发而设计的软件框架。开发人员使用它们来开发主机、移动设备和电脑游戏。游戏引擎提供的核心功能通常包括用于 2D 或 3D 图形的渲染引擎（“渲染器”）、物理引擎或碰撞检测（和碰撞响应）、声音、脚本、动画、人工智能、网络组件、资源和内存管理、多线程、场景编辑器。在很大程度上，通过使用相同的游戏引擎来创建不同的游戏，或者将游戏移植到多个平台，游戏引擎可以帮助开发者省去大量的开发流程和时间。

在许多情况下，除了可重用的软件组件之外，游戏引擎还提供了一套可视化开发工具。这些工具通常以数据驱动的方式提供简单、快速的游戏开发方式。引擎开发者通过开发几乎所有游戏构建需要的元素来组成强大的软件套件来帮助真正的游戏开发者摆脱这种重复的、高投入的、底层的工作，从而可以直接去创造游戏内容。

大多数游戏引擎套件提供了便于开发的组件，如图形、声音、物理和 AI 功能。这些游戏引擎有时被称为“中间件”，因为与中间件的商业意义一样，它们提供灵活和可重复使用的软件平台。其开箱即用的所有核心功能，成为开发游戏，同时降低成本和复杂性的关键因素。Unity、Unreal、Frostbite、Cocos2D/3D、CryEngine 便是这种被广泛使用的游戏引擎。

与其他中间件解决方案一样，游戏引擎通常提供跨平台功能，允许在各种平台上运行相同的游戏，包括游戏机和 PC，同时不用对游戏源代码做过多更改。通常，游戏引擎被设计为基于组件的架构系统，允许使用更专用（并且通常更昂贵）的其他中间件来替换或扩展引擎中的特定系统，如用于物理模拟的 Havok，用于声音的 Miles Sound System 或 Bink 视频，用于全局光照组件 Enlighten，用于植被模拟的 Speedtree。一些游戏引擎，如 RenderWare 甚至被设计为一系列松散连接的游戏中间件组件，可以选择性地组合以创建定制化引擎。

尽管名称中有游戏两个字，似乎看起来功能比较单一，但实际上游戏引擎通常也能用于有实时图形需求的其他类型的交互式应用，如营销演示、建筑可视化、训练模拟等。

一些游戏引擎仅提供实时 3D 渲染能力，而没有游戏所需的全部其他的功能。这些引擎需要游戏开发者来实现其他功能或使用其他游戏中间件来组合成完整的引擎。这些类型的引擎通常被称为“图形引擎”、“渲染引擎”或“3D 引擎”，而不是包含更多组件的术语所谓“游戏引擎”。图形引擎的一些示例是：Crystal Space、Genesis3D、Irrlicht、OGRE、RealmForge、Truevision3D 和 Vision Engine。

现代游戏引擎是一整套复杂的应用程序，通常有几十个子系统交互，以确保精确控制用户体验。游戏引擎在持续演变，在渲染、脚本、美术和关卡设计之间形成了更大的独立性。例如，在如今的一个典型的游戏开发团队中，美术人员的数量是程序人员数量的许多倍。

随着移动平台的爆发，更多的游戏引擎出现，其中最重要的便是 Unity，它也是第一个推出跨平台的商用引擎：写一遍代码，引擎通过自己的翻译器，生成 CIL（通用中间语言），运行时从 CIL 到本地设备即时编译运行。这种方式大大省去了游戏开发者在移动平台两大系统（安卓和 iOS）间的支持工作。随后，Unreal Engine（虚幻引擎）也慢慢开始跨平台，除了老本行 PC 和游戏机，最近几年也开始支持移动平台。

游戏引擎也随着时间进化更新着，对于新技术的出现也会很快吸收并转化为引擎自身的部分。随着新的图形 API 的推出，主流商业引擎也纷纷开始投入人力进行 DirectX 12 和 Vulkan 的支持，并且对于新兴的 VR 产业，也不断进行优化来提供更好的性能以帮助 VR 开发者，Unreal Engine 最近就优化了它的前向渲染，使得开发者可以获得更多的性能。

随着游戏引擎技术的成熟和变得更加用户友好，游戏引擎的应用范围已经扩大。以 CryEngine 为例，它们现在被用于更加严肃的领域：可视化、培训、医疗和军事模拟应用程序。同时为了促进这种引擎使用范围的扩张，包括移动电话（如 Android 手机、iPhone）和网络浏览器等硬件、软件在内的新平台，都变成了引擎的目标平台（如 WebGL、Shockwave、Flash、Trinigy 的 WebVision、Silverlight、Unity Web Player、O3D 和纯 DHTML）。

此外，更多的游戏引擎正建立在诸如 Java 和 C#/. NET（如 TorqueX 和 Visual3D. NET）、Python（Panda3D）或 Lua Script（Leadwerks）等高级语言之上。由于大多数 3D 游戏现在主要是 GPU 限制的（即受到显卡能力的限制），所以更高级语言的翻译开销导致的潜在性能降低变得可以忽略，而由这些语言提供的生产力增益则是巨大的。最近的这些趋势被微软等公司用来推动独立游戏开发。微软开发的 XNA 便是作为在 Xbox 和相关产品上发布的所有游戏的首选 SDK，这里包括 Xbox Live Indie 游戏频道，其是专为那些没有足够资源在零售货架上包装销售游戏的小型开发商而设计的。

游戏引擎、D3D/OpenGL、Shader、编程、显卡是游戏开发环节上的整体生态，互相关联、互相影响。显卡的革新和技术往往会影响其他 4 个形态，然后最终影响游戏和图形应用开发者的内容质量。一个好的开发者应该对这几个模块都有所理解。

AMD Radeon RX480 显卡是一款非常适合读者学习开发的中端显卡。后面的章节展示了一些在 Unity 引擎上的 Shader 编程，所有程序都由 AMD 员工在该显卡上一一验证，以方便读者学习。