# 第 1 部分　背景与概览

这一部分讲述 GPGPU 和开源项目的背景。

# 第 1 章　行业背景和发展

## 1.1　GPGPU 的历史与发展

GPGPU（General-Purpose computation on Graphics Processing Units）是指利用图形处理器（GPU）执行图形渲染之外的通用计算任务的设备。自 20 世纪中期计算机图形学诞生以来，图形处理器逐渐从专用于加速图像生成的硬件，演变为能够执行通用计算任务的重要计算单元。这一转变不仅提高了图形渲染的性能，同时也推动着高性能计算、科学模拟、人工智能等多个领域的发展。

GPGPU 的起源可以追溯到早期计算机图形的发展阶段。在图形硬件加速需求增加的背景下，图形处理器逐步从辅助计算的角色转变为核心的计算部件。早期的计算机图形系统最初只是为了实现简单的显示任务，然而随着计算机辅助设计（CAD）、飞行模拟等应用的需求增加，图形硬件逐渐向更加专用化、高效化的方向发展。到了 20 世纪 90 年代，随着 3D 图形渲染的复杂性和计算需求的增加，出现了越来越多专用的图形加速器。1999 年，NVIDIA 推出 GeForce 256，首次引入了“GPU”这个术语，标志着现代图形处理器时代的开端。最初，GPU 被设计为固定功能硬件，专门用于图形渲染，但随着硬件技术的进步，GPU 逐渐引入了可编程着色器，使开发者可以编写自定义的程序，灵活地处理图形数据。由此，GPU 的计算能力不再局限于图形渲染，逐渐扩展到其他领域。到了 21 世纪初，随着 CUDA、OpenCL 等编程框架的出现，GPU 正式进入通用计算时代，也就是所谓的 GPGPU 时代。

如今，GPGPU 已成为高性能计算领域的一个重要组成部分。凭借其强大的并行计算能力，GPGPU 在人工智能、深度学习、科学计算等领域得到广泛应用，成为推动这些领域技术进步的关键力量。GPGPU 的快速发展体现了计算硬件从专用设备向通用化、智能化方向的演进，展示了计算机图形学与通用计算领域的深度融合。

### 1.1.1　早期计算机图形的发展

计算机图形技术从最初的简单显示需求，逐步演化为现代复杂的 3D 渲染和虚拟现实技术。在这一发展过程中，计算机图形的硬件和软件技术不断提升，奠定了后来 GPGPU 技术的基础。

**1. 最初的计算机图形系统的诞生**

图形显示技术的起源可以追溯到 1907 年，俄国科学家 Boris Rosing 利用阴极射线管（CRT）将简单的几何图像传输到电视屏幕上。CRT 显示器通过发射电子束来激发涂有荧光材料的显示屏幕，是矢量和光栅显示的硬件基础。CRT 技术的出现大幅提升了显示器的分辨率和图像生成能力，使得早期的矢量和光栅显示器都能够以较低成本实现复杂的

图像渲染。

早期计算机图形系统的诞生可以追溯到 20 世纪 40 年代末到 50 年代初，尽管当时的硬件技术还非常原始，但这些早期的尝试为后来图形处理器的发展奠定了基础。如图 1.1 所示的 EDSAC（Electronic Delay Storage Automatic Calculator）是 20 世纪最早的电子存储程序计算机之一，1949 年在英国剑桥大学由莫里斯·威尔克斯（Maurice Wilkes）和他的团队设计建造。虽然其设计初衷是用于科学计算，但其设计理念为后来的计算机显示技术奠定了基础。

图 1.1　英国剑桥大学设计建造的 EDSAC 计算机

1950 年，美国麻省理工学院的旋风一号（Whirlwind Ⅰ）计算机配备了世界上第一台图形显示器——阴极射线管（CRT），如图 1.2 所示，这标志着计算机开始具备图像显示功能。然而，当时的图形显示还处于“被动式”阶段，即计算机能够输出图形，但无法进行交互操作。麻省理工学院利用该计算机实现了首个可编程飞行模拟器，为飞行员的训练提供了全新的工具。这是世界上第一个可以编程的飞行模拟器，主要用于军事和航空工业中的飞行仿真。该系统要求实时显示飞行器的状态和环境变化，因此需要计算机具备一定的图形处理能力。

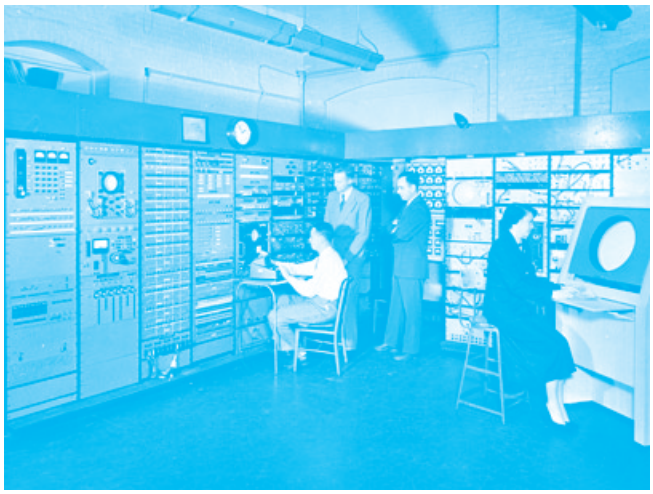

图 1.2　美国麻省理工学院的旋风一号计算机

1960 年，贝尔实验室的威廉·费特（William Fetter）及其团队成功生成了第一张 3D 透视图像。这项技术通过数学方法计算三维物体在二维平面上的投影，展示了计算机生成三维图像的潜力。这张图像最早被应用于飞机驾驶舱设计的视角分析，为工业设计领域提供了一个全新的工具。这一时期的图形技术虽然相对简单，但展示了图形学在现实工程中的实用性。

1963 年，Sketchpad 项目在麻省理工学院林肯实验室诞生，由伊万·萨瑟兰（Ivan Sutherland）领导开发。图 1.3 展示的 Sketchpad 是首个实现交互式图形处理的计算机系统，它允许用户通过光笔直接在屏幕上绘制和编辑几何图形，这种交互方式开创了现代图形用户界面的雏形。Sketchpad 还首次引入了对象的约束机制，能够根据用户设定的条件自动调整图形的结构和属性，例如，保持线段平行或使点位于特定路径上。这一系统的设计理念直接影响了后来的计算机辅助设计（CAD）、图形用户界面以及面向对象编程的发展。

图 1.3　首个实现交互式图形处理的计算机系统 Sketchpad

随着计算机图形技术在 20 世纪 60 年代初步发展，几家重要的技术公司迅速抓住了这一领域的潜力，并通过硬件创新推动了图形系统的实际应用。Evans & Sutherland、IBM 和 Intergraph 等公司，在早期计算机图形硬件的开发过程中起到了至关重要的作用。

Evans & Sutherland 是计算机图形领域最早的商业公司之一，由图形学先驱伊万·萨瑟兰（Ivan Sutherland）和大卫·埃文斯（David Evans）于 1968 年创办。E & S 开发了世界上第一批图形硬件，特别是用于实时仿真和飞行模拟的三维图形系统。它们的早期产品，如图 1.4 中的 LDS-1（Line Drawing System-1），是早期图形显示的杰出例子，能够生成简单的二维图形，该模型被称为第一个具有图形处理器的图形设备。

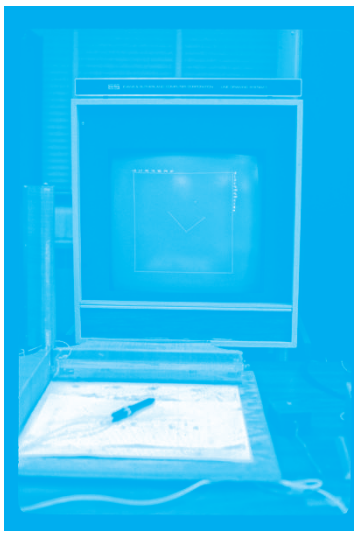

图 1.4　第一个具有图形处理器的图形设备 LDS-1

IBM 作为计算机硬件的领导者，也在推动早期图形系统发展中发挥了关键作用。图 1.5 中展示的是 1964 年发布的 IBM 2250 图形终端，是第一款商用图形显示终端，允许用户通过光笔与显示屏交互。这一设备开创了计算机与人交互的图形操作方式，并用于工业设计、数据可视化等领域。IBM 2250 不仅推动了早期的交互式图形技术，还在当时为工程、科学计算和设计提供了基础。

图 1.5　第一款商用图形显示终端 IBM 2250

这些公司的早期硬件产品和系统极大地提升了计算机图形的实际应用效率，并为后来的 GPU 发展提供了基础技术。这些早期图形处理技术推动了计算机从简单的文本界面向复杂的图形界面的过渡，是图形处理器演变过程中不可忽视的关键环节。

**2. 图形显示技术的演进**

随着计算机图形技术的不断发展，图形显示设备也经历了从简单的矢量显示到复杂

的光栅显示的演进过程。这一演进不仅提升了图像的精度与表现力，还为现代 GPU 和显示技术奠定了基础。矢量显示器和光栅显示器的技术路线在早期计算机图形领域中分别占据了一定的地位，并推动了像素和图像处理的核心概念逐渐成型。

20 世纪 60 年代，如图 1.6 所示的矢量显示器（Vector Display）成为图形显示的主要技术。这种显示器通过电子束在屏幕上绘制几何图形的线条，而不是用像素网格来显示图像。矢量显示器的一个典型的应用是战斗机中的平视显示器，这是因为通过减慢电子束在荧光粉上的移动速度，可以实现更亮的显示，使显示器在直射阳光下对飞行员清晰可见。这种技术尽管在显示几何图形方面非常高效，但由于无法显示复杂的阴影、颜色和纹理，其应用受到了很大的限制。

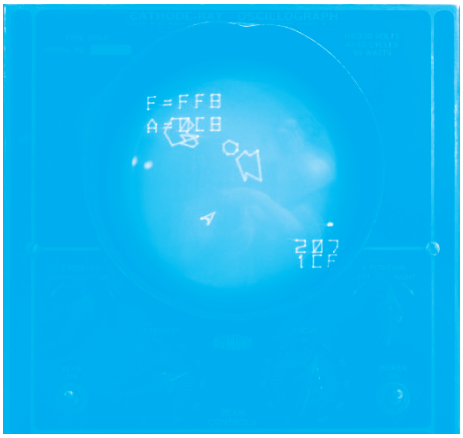

图 1.6　矢量显示器

然而，随着图形应用的扩展，特别是随着计算机辅助设计（CAD）和其他需要显示细节的应用程序的需求增加，光栅显示器（Raster Display）于 20 世纪 60 年代末到 70 年代初逐渐兴起。与矢量显示不同，光栅显示器通过将屏幕划分为一个个小的像素来生成图像。这种图像表示方式被称为光栅图形（Raster Graphics），也常被称为位图图形（Bitmapped Graphics）。每个像素都能独立控制颜色和亮度，允许显示设备能够绘制复杂的色彩和图像细节。这一技术的出现解决了矢量显示无法呈现细节丰富的图像和复杂色彩的问题。其中，“像素”这一概念首次被引用是在 1965 年，由弗雷德里克·比林斯利（Frederic Billingsley）用来描述数字图像的最小显示单元。在这一时期，光栅显示器逐渐在科学计算和设计领域中得到广泛应用，能够生成更高分辨率的图像和复杂的纹理。

光栅显示器的分辨率通过像素密度来衡量，单位是每英寸像素数（Pixels Per Inch，PPI）（1 英寸 =2.54 厘米）。图 1.7 展示了 10 PPI 和 20 PPI 的对比，可以看到更高的像素密度（PPI）意味着每单位面积内有更多的像素，从而能够显示出更细腻、更清晰的图像。例如，现代高分辨率显示器至少具有 1920×1080 px 的分辨率，总计约 207 万像素（或 200 万像素），这被称为“全高清”分辨率。

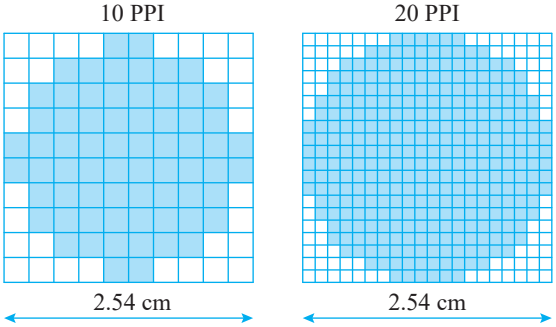


图 1.7　不同的像素密度

### 1.1.2 专用图形处理器的诞生

随着计算机图形技术的逐渐成熟，在 20 世纪 70—80 年代，图形处理的复杂性和计算需求不断增长，单靠 CPU 已经难以满足越来越高的图形渲染要求。因此，专用的图形处理硬件应运而生。这些硬件提升了图形处理的效率，为现代图形处理器（GPU）的发展奠定了基础。图形处理器的诞生标志着图形处理从依赖通用计算设备，向更加专业化、并行化硬件的转变。

**1. 图形控制器的出现**

早期计算机图形处理的发展离不开图形控制器（Graphics Controller）的出现。图形控制器是一种专用的硬件设备，用于加速计算机中的图形显示任务。它的主要功能是减轻中央处理器（CPU）在处理图形时的负担，专门处理点阵图像、线条绘制以及 2D 图形渲染。

20 世纪 80 年代，图形控制器在大型计算机和微型计算机中开始广泛应用。这一时期的图形控制器还非常初级，主要用于绘制简单的线条和点阵图形，帮助在显示器上显示 2D 的几何图像。它们的功能非常有限，主要是加速 CPU 已经处理好的图形数据的显示过程。例如，1982 年，日本电气公司（NEC）推出了世界上首个采用大规模集成电路技术的图形控制器——NEC 7220（如图 1.8 所示）。这一图形控制器广泛应用于 CAD 终端、个人计算机和客户机 / 服务器显示系统。在 NEC 7220 问世之前，图形控制器通常依赖大量分立逻辑和存储芯片来处理图形任务，体积庞大且效率低下。NEC 7220 通过集成电路技术大大简化了电路设计，使图形加速器更加紧凑且高效。此外，ATI（现为 AMD）等公司推出的早期图形加速卡也是此类技术的代表。这些硬件能够加速窗口的移动、图像的刷新以及基本的 2D 图形操作。

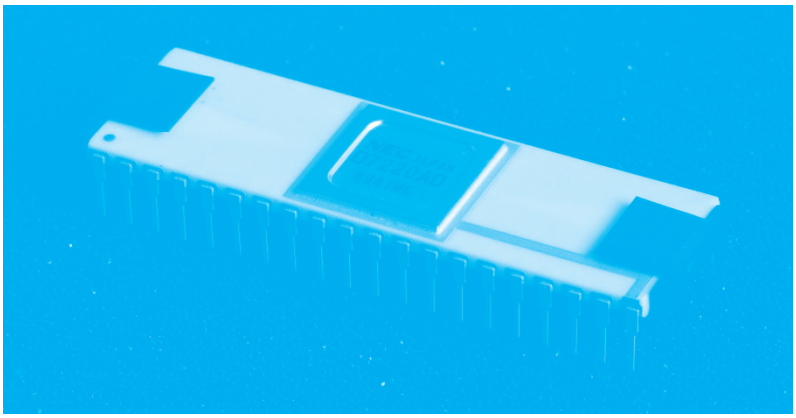

图 1.8　首个采用大规模集成电路技术的图形控制器 NEC 7220

到了 20 世纪 90 年代，3D 图形的需求迅速增加。随着个人计算机和视频游戏产业的崛起，3D 图形处理的性能需求达到了前所未有的高度。此时，专用的 3D 图形加速器开始出现。以 3Dlabs、NVIDIA 和 SGI 为代表的公司，推出了一系列支持 3D 图形的图形加速卡，专门用于处理 3D 几何变换、纹理映射和光照计算等任务。

1982 年，SGI 推出了一款专门的 3D 图形加速器——几何引擎（Geometry Engine）。这款几何处理器的核心功能是专门为计算机图形中的几何计算任务而设计的，主要执行矩阵变换、裁剪和坐标映射等基本操作。与之前依赖 CPU 的图形处理不同，几何引擎通

过超大规模集成电路技术，极大地提高了几何运算的效率。它可以处理 2D 和 3D 的浮点运算，使图形数据在显示设备上得以实时渲染。最显著的优势之一是，它将复杂的几何变换，如旋转、缩放和透视投影，整合在一个系统中，从而大大加快了 3D 图形渲染的速度。此外，3Dlabs 的 GLint Gamma 和其他几何处理器也在 20 世纪 90 年代初期广泛应用于专业图形工作站中。

尽管这些早期的图形控制器和加速器在性能上有很大的提升，但它们的功能仍然比较有限，无法实现超出硬件预设的功能，通常只能够处理专门的图形任务，缺乏灵活性和可编程性，限制了其在复杂场景中的应用。

### 2. 固定功能图形流水线

在这些早期的图形控制器和加速器基础上，固定功能图形流水线（Fixed-Function Graphics Pipeline）成为当时图形硬件的主流设计。所谓固定功能流水线，是指图形处理的每一个阶段都是由硬件预定义的，开发者只能通过图形 API（如 OpenGL 或 DirectX）来调用固定的渲染功能，而不能对硬件进行直接编程。这样的架构虽然加速了图形处理任务，但也限制了图形渲染的灵活性。

渲染流水线（Rendering Pipeline）是固定功能图形流水线的一个具体实现，它是用于将 3D 场景转换为 2D 图像的基本处理流程。在 *Real-time Rendering* 一书中，渲染流水线被划分为 4 个阶段，分别是应用（Application）阶段、几何处理（Geometry Processing）阶段、光栅化（Rasterization）阶段和像素处理（Pixel Processing）阶段，如图 1.9 所示。

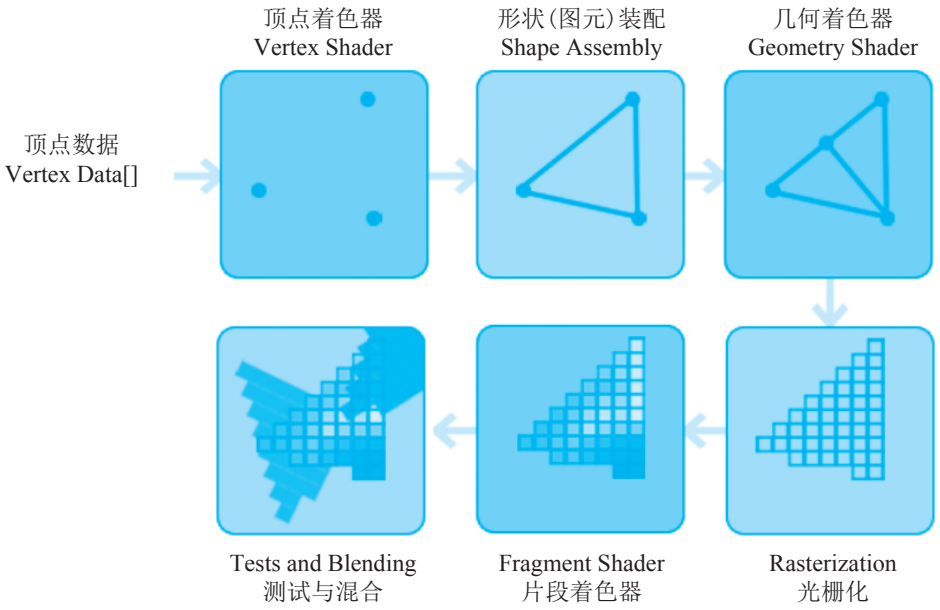


图 1.9　图像渲染的不同阶段

在应用阶段，CPU 执行一系列操作，如碰撞检测、全局加速算法、动画和物理模拟。应用程序定义当前要构建的世界模型图元（Primitives），如点、线和三角形，并将这些数据和参数传递给 GPU。在这个阶段，CPU 还会对 GPU 中的各个着色器（Shaders）进行编程和参数设定。

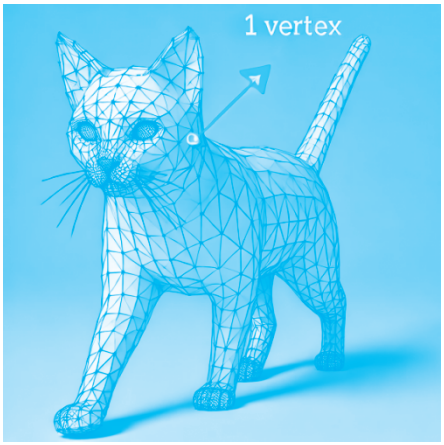

图 1.10　图形中的顶点

进入几何处理阶段，GPU 对输入的图元进行调整、修改和转换，处理对象始终是图元。首先是顶点着色（Vertex Shading），顶点着色器（Vertex Shader）对每个顶点（如图 1.10 所示）进行处理，包括坐标变换（将顶点从模型空间转换到视图空间）、计算光照影响、应用动画变形以及生成纹理坐标（Texture Coordinates）。接下来是细分（Tessellation），在原有图元内部增加更多的顶点和面，使模型更加精细和平滑。这涉及外壳着色器（Hull Shader）、细分器（Tessellator）和域着色器（Domain Shader）。举例来说，将低多边形的球体细分为高多边形的球体，以呈现更平滑的表面。然后是几何着色，几何着色器可以在图元外部添加或删除顶点，生成新的图元。在投影（Projection）阶段，将 3D 场景投影到 2D 视平面，通常使用透视投影或正交投影。透视投影模拟人眼的视觉效果，远处的物体看起来更小。接着是裁剪（Clipping）阶段，移除视锥体之外的图元，减少不必要的计算。最后是屏幕映射，将标准化的坐标转换为屏幕坐标系中的像素位置，准备进行光栅化。

在光栅化阶段，将处理后的图元转换为片段或像素（Pixels），确定哪些像素被图元覆盖。首先是三角形设置，计算图元的边界和属性，为遍历做好准备。例如，确定三角形的边方程，计算用于光栅化的参数。然后是三角形遍历，遍历像素网格，确定哪些像素位于图元内部，生成需要着色的像素列表。

最后，在像素处理阶段，计算每个像素的最终颜色，包括纹理映射、光照计算和颜色混合。首先是像素着色（Pixel Shading），对每个像素进行颜色计算，应用纹理、光照和其他效果。例如，根据像素的纹理坐标，从纹理图像中采样颜色，并结合光照模型计算最终颜色。然后是合并（Merging），又称为输出合并（Output Merging），在此阶段进行深度测试、模板测试和颜色混合，决定像素是否显示以及如何与已有像素融合。例如，在渲染半透明物体时，需要混合其颜色与背景颜色，计算叠加效果。

固定功能图形流水线技术的一个重要里程碑是 1999 年发布的 NVIDIA GeForce 256，见图 1.11，这款显卡首次将变换与光照（Transformation and Lighting，T&L）引擎集成到单一芯片中。这种创新使得 3D 图形渲染的效率大幅提高，T&L 引擎能够处理顶点的几何变换、光照计算，并将 3D 物体转换为屏幕上的 2D 图像。GeForce 256 的发布标志着真正意义上 GPU 的诞生，它使得图形渲染的多个步骤可以通过专用硬件自动完成，彻底解放了 CPU 资源。

在固定功能流水线的框架下，图形 API 的发展也起到了关键作用。DirectX 7 于 1999 年发布，成为 Windows 平台上图形开发的重要工具。它引入了对 T&L 硬件加速的支持，使开发者能够充分利用固定功能流水线的能力来渲染 3D 场景。同时，OpenGL 作为跨平台的图形 API，也在科学计算和工业设计领域得到了广泛应用。

早期的图形硬件主要服务于特定的行业需求，如 CAD、模拟和娱乐业等。在这些领域，图形处理器被用于渲染 3D 模型、生成图形特效和加速交互式应用。尽管固定功能

流水线具有一定的局限性，但它在推动计算机图形硬件发展上起到了重要的作用。

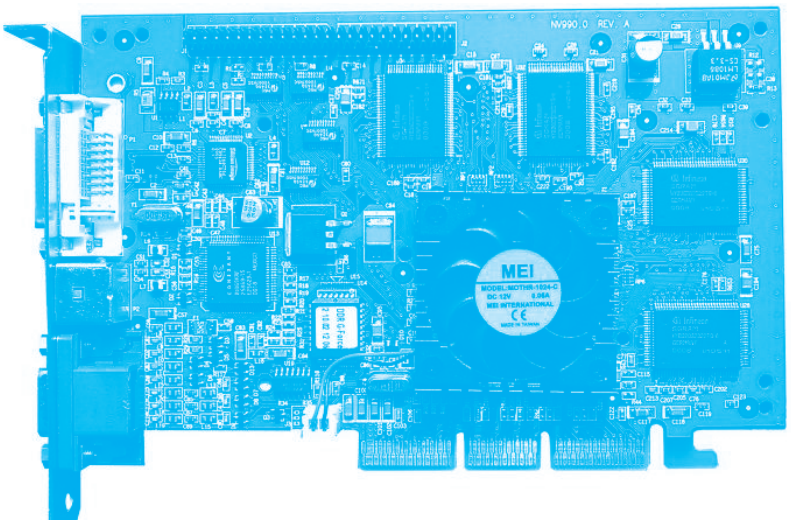


图 1.11　NVIDIA GeForce 256

### 3. GPU 概念的提出与定义

GPU 的概念在 20 世纪末得到了明确的定义和广泛传播，标志性的事件是 1999 年 NVIDIA 发布的 GeForce 256 显卡，见图 1.12。这款显卡是 NVIDIA“GeForce”产品线的首个成员，NVIDIA 当时将其称为“全球首款 GPU”，并赋予了 GPU 明确的定义：一个单芯片处理器，集成了几何变换（Transformation）、光照计算（Lighting）、三角形设置与裁剪（Triangle Setup/Clipping）、渲染等多个功能。这一集成使 GeForce 256 能够每秒处理至少 1000 万个多边形，成为图形处理硬件领域的重大技术突破。

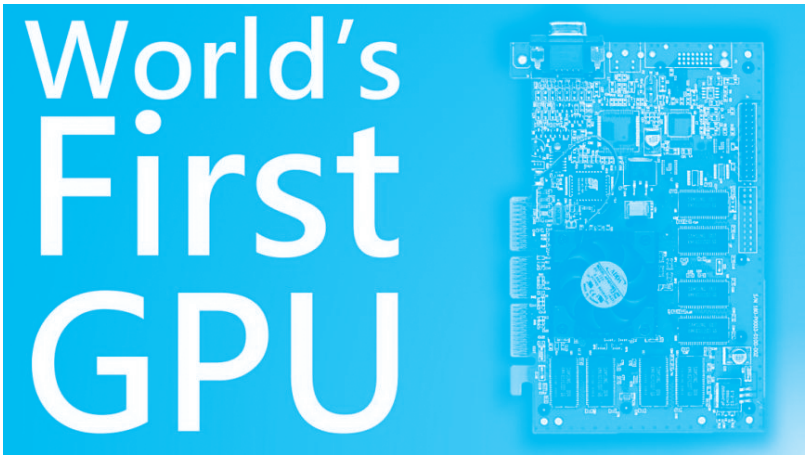


图 1.12　第一台 GPU GeForce 256

在此之前，图形处理器通常被称为“3D 加速器”，其功能主要集中在渲染和像素处理，无法承担几何计算等复杂的任务。GeForce 256 通过整合 T&L 引擎，改变了这一局面。在过去，几何变换和光照计算都是由 CPU 完成的，称为“软件 T&L”。这种计算方式增加了 CPU 的负担，导致在处理复杂的 3D 场景时，性能往往受限。GeForce 256 的硬件 T&L 引擎专门处理几何计算和光照渲染，显著减轻了 CPU 的压力。这种创新为游戏

行业带来了显著的性能提升，极大地推动了消费级图形硬件的普及。

GeForce 256 的技术特点进一步巩固了 GPU 的概念和地位。它采用了 4 条 64 位像素流水线（QuadPipe Rendering Engine），支持在单一时钟周期内处理多个像素，这意味着其在渲染效率上有了质的飞跃。这种硬件设计使得 GeForce 256 在单纹理游戏场景中，每个时钟周期可以输出 4 个像素，在双纹理场景中则可以输出两个多纹理像素。通过这一架构，GeForce 256 在 3D 游戏中的性能提升超过了以往的 3D 加速器，尤其是在需要更高帧率的场景下，它比前代产品如 RIVA TNT2 和 3dfx 的 Voodoo3 系列显卡有了明显的优势。

GeForce 256 还扩展了其他图形处理功能。它为 MPEG-2 视频解码提供了硬件运动补偿，支持更高质量的视频回放。这一功能对于当时的多媒体处理也有重要意义，因为它提升了个人计算机在处理视频内容时的性能，减少了对 CPU 的依赖。

尽管在最初发布时，GeForce 256 的硬件 T&L 并未被广泛的软件所支持，导致其优势在一些场景中未能完全发挥，但随着 DirectX 7 的普及和更多游戏对硬件 T&L 的支持，GeForce 256 的潜力逐渐显现。相比于当时的其他显卡，尤其是依赖强大 CPU 来弥补图形性能不足的竞争对手（如 3dfx 的 Voodoo3 系列），GeForce 256 在低预算 CPU 系统中显示出了更强的性能优势。这一变化促使图形硬件的发展逐渐从依赖 CPU 向依赖 GPU 过渡，开启了图形计算的新时代。

## 1.2 GPU 的可编程化与 GPGPU 的萌芽

随着图形处理需求的不断增长，早期的固定功能 GPU 在灵活性上逐渐暴露出局限性。在固定功能流水线中，每个阶段的任务（如顶点变换、光照、像素着色等）都是由硬件预定义的，开发者只能调用这些固定的功能，而无法通过编程来改变处理的细节。例如，光照的计算方式、材质的处理方法等，都是由硬件内部决定的。这种限制导致开发者在想要实现更复杂的光照、纹理或特效时，往往受到硬件的束缚，无法实现更加灵活和定制化的渲染效果。这种模式虽然适合简单的图形任务，但面对日益复杂的 3D 场景和渲染效果，无法满足日渐提高的视觉要求。

随着图形处理需求的增加和技术的发展，GPU 逐渐具备了处理更复杂计算任务的能力，而这为后来的通用计算打下了基础。

### 1.2.1 可编程着色器时代（2001—2006 年）

GPU 进入可编程时代的关键标志是 2000 年微软发布的 DirectX 8.0，如图 1.13 所示，其引入了可编程的顶点着色器（Vertex Shader）和像素着色器（Pixel Shader）。在此之前，GPU 依赖于固定功能流水线，开发者只能通过调用预定义的硬件功能来处理图形渲染任务，灵活性受限。而 DirectX 8.0 中的可编程着色器允许开发者编写自定义的程序，从而控制顶点和像素的处理方式。这标志着 GPU 从固定功能阶段向可编程阶段的转变，极大地提高了图形处理的灵活性和计算能力。


图 1.13　微软推出的着色器模型 DirectX


图 1.14　支持可编程着色的显卡 GeForce 3

图 1.14 是 NVIDIA 在 2001 年发布的 GeForce 3，是第一批支持可编程着色器的显卡之一。这款显卡支持开发者通过编写自定义着色器来灵活处理图形渲染的每个步骤，解决了固定功能流水线的局限性。通过顶点和像素着色器，开发者可以控制物体的几何变换、光照方式以及材质处理。GeForce 3 让开发者能够实现更复杂的光照、纹理特效等视觉效果，如法线贴图、环境映射等技术，这些技术极大地提高了图形的逼真度和细节表现。

ATI 的 Radeon 8500 同样是可编程着色器时代的重要代表之一。Radeon 8500 于 2001 年 10 月发布，基于 ATI 的 R200 架构，成为与 NVIDIA GeForce 3 抗衡的产品。Radeon 8500 提供了与 GeForce 3 类似的顶点和像素着色器功能，允许开发者编写自定义的着色器程序。

可编程着色器的引入，提高图形渲染的灵活性和计算能力。与固定功能流水线相比，可编程着色器让图形渲染更加灵活，为开发者提供了更多的创作自由，开发者能够精确地控制每个像素的处理流程，从而实现复杂的光照效果、纹理处理以及动态反射等高级图形渲染技术。

可编程着色器带来了对硬件支持的需求，这促使了着色器模型（Shader Model）的演进。着色器模型定义了 GPU 硬件能够支持的着色器功能和复杂度。DirectX 8.0 引入的 Shader Model 1.0 是最早的可编程着色器模型，它支持了基本的顶点和像素操作，允许开发者通过编程控制几何变换和光照计算。随着图形处理需求的增加，着色器模型逐步升级。DirectX 9 带来的 Shader Model 3.0 提供了更多功能，支持更复杂的着色器指令、更长的着色器程序和更大的渲染寄存器集。这些升级使开发者可以实现更细腻的图形效果。

为了简化开发者编写着色器程序的过程，各种着色器编程语言相继出现。这些语言为开发者提供了高层次的编程接口，使编写复杂的着色器程序变得更加便捷。微软推出了 HLSL（High-Level Shader Language），这是 DirectX 平台下的着色器编程语言，允许开发者直接在代码中编写顶点和像素着色器。与之类似，GLSL（OpenGL Shading Language）是 OpenGL 平台的高层次着色器语言，NVIDIA 开发的 Cg 语言则为其 GPU 硬件提供了高效的编程工具。

### 1.2.2　通用计算需求的增长

因为图形处理器在可编程性上的进步，GPU 的潜力逐渐超越了图形渲染本身。在图形处理之外，其他领域对大规模并行计算的需求也在不断增加，推动了 GPU 的进一步发展。在科学计算、数据分析等需要高性能计算的领域，传统 CPU 无法高效处理这些任务，而 GPU 的并行处理能力使其成为潜在的解决方案。

### 1. 非图形计算需求的增长

在早期，GPU 的设计专注于图形渲染任务。然而，随着数据量的增加以及计算复杂度的提升，非图形领域的计算需求也迅速增长。在科学计算、物理仿真、气候建模、数据挖掘和金融分析等需要处理大量并行数据的应用中，任务通常包含复杂的数学计算，且需要处理大规模的数据集。传统的 CPU 在这类任务中往往显得力不从心，因为 CPU 的设计更适合于串行计算，难以高效地处理大规模并行任务。

与此同时，GPU 的并行计算架构展现出了其独特的优势。GPU 擅长处理高度并行的任务，能够同时处理数千个小任务，这使得它在一些非图形的计算任务中也具有较大的潜力。例如，科学计算中常见的矩阵运算和线性代数计算本质上就是高度并行的任务，可以在 GPU 上高效完成。

### 2. 用户利用 GPU 进行通用计算的尝试

在可编程着色器逐步被引入后，开发者开始意识到，GPU 不仅可以用于图形渲染，它的并行处理能力也可以用于非图形领域的计算任务。随着对并行计算需求的增加，早期的研究者和开发者开始利用 GPU 来执行复杂的数学计算，开创了基于 GPU 的通用计算的先河。

在这些初期的尝试中，研究人员主要利用当时的图形 API 来编写着色器程序，间接执行非图形计算。例如，早期的矩阵乘法运算就是一种典型的通用计算应用，通过将矩阵数据映射为图像纹理，再利用像素着色器执行矩阵乘法操作，研究人员成功地利用 GPU 加速了这一计算任务。2003 年，马克·哈里斯（Mark Harris）发表了一篇关于 GPGPU 的开创性论文“General-Purpose Computation on Graphics Hardware”，详细介绍了如何通过图形 API 来进行非图形的计算任务。这个研究展示了 GPU 在矩阵计算、向量处理等并行任务中的显著性能提升，并激发了其他领域对 GPU 通用计算潜力的兴趣。

在数值模拟任务方面，GPU 也被用于对稀疏矩阵和多重网格求解。具体来说，研究人员实现了两个基本的计算 Kernel：稀疏矩阵的共轭梯度求解器和规则网格的多重网格求解器。这些求解器被应用于几何流和流体模拟中，并取得了显著的性能提升。这些实现的关键在于将传统的 CPU 数值运算方法映射到 GPU 的并行计算架构上。通过将数据组织成适合 GPU 并行处理的数据结构，并使用纹理内存来存储矩阵和向量，可以利用 GPU 的可编程片段着色器完成矩阵 - 向量乘法和内积等核心计算任务。

同时，生物医学领域也进行了早期的 GPU 通用计算尝试。蛋白质折叠是生物化学中的一个关键问题，涉及大量复杂的分子间相互作用，计算量非常繁重。通过将蛋白质折叠问题的计算映射到 GPU，研究人员能够显著提高模拟效率，使得原本需要几天甚至几周的计算任务在数小时内完成。这一尝试表明，GPU 的并行计算能力可以用于更多领域的复杂计算，推动了相关领域的研究进展。

### 3. 开发新型编程模型的必要性

尽管这些早期的尝试展示了 GPU 在通用计算中的巨大潜力，但随着应用的深入，现有的编程模式很快暴露出诸多问题和限制。早期的 GPGPU 编程主要依赖于图形 API，这些 API 的设计初衷是为了加速图形渲染，而不是为通用计算任务提供支持。由于 API 本身的局限性，开发者不得不采用一种“曲线救国”的方式，将非图形计算任务映射成图形处理问题，再利用 GPU 的并行处理能力完成计算。这种间接的方式大大增加了编程

的复杂性，迫使开发者在不熟悉的图形框架中进行非图形领域的计算，实现效率也因此受到影响。

开发者逐渐意识到，依赖图形 API 进行非图形计算并非长久之计。科学计算、数据分析和机器学习等领域的快速发展，对大规模并行计算能力提出了更高的要求，而现有的工具链显然无法满足这些需求。这一局面催生了对新型、更高效编程模型的迫切需求，能够直接为通用计算任务服务，充分发挥 GPU 的并行计算潜力。

一个有效的新编程模型需要从根本上解决图形 API 带来的限制。首先，新模型应直接面向通用计算任务设计，开发者可以绕过图形渲染的框架，直接访问和利用 GPU 的强大并行处理能力。这种模型应当简化编程过程，帮助开发者专注于解决具体的计算问题，而不必通过复杂的转换将计算问题“包装”成图形任务。通过这一改变，开发者可以更高效地利用 GPU 的硬件资源，从而提升整体的计算性能。其次，新的编程模型应当大幅提高编程效率，降低开发门槛，允许更多领域的开发者轻松使用 GPU 进行并行计算。

## 1.3 GPGPU 通用计算时代的兴起（2006 年起）

### 1.3.1 统一渲染架构的引入

在 2006 年之前，GPU 的设计主要采用分离架构，顶点、几何和像素着色器分别由专门的硬件单元完成特定任务。这种固定功能流水线的设计在特定场景中效率较高，但随着图形处理需求的多样化和复杂化，其资源利用率的局限性逐渐显现。统一架构的提出，正是为了解决这一问题。微软在 DirectX 10 中引入了统一着色器模型（Unified Shader Model），提出将各类着色器整合为统一的可编程处理单元，以动态适配任务需求。

NVIDIA 于 2006 年年底发布了 Tesla G80 架构，这是首个支持 DirectX 10 并实现统一架构的 GPU。G80 架构中的处理单元采用了流多处理器（Streaming Multiprocessor，SM）的设计理念，将顶点、几何和像素着色器集成到统一的流水线中。这一架构显著提高了硬件资源的利用率，同时为更加灵活的编程模型奠定了基础。此外，几何着色器（Geometry Shader）的引入为复杂的几何处理提供了强大支持，进一步扩展了 GPU 的应用场景。统一架构的成功推动了行业的发展，AMD 也在随后推出了自己的统一架构 R600，以适应 DirectX 10 和 OpenGL 3.2 的规范，然而，R600 在性能和功耗效率方面未能赶上 NVIDIA 的 G80，导致 AMD 在高端 GPU 市场的竞争中处于劣势。

### 1.3.2 CUDA 的推出与影响

2007 年，NVIDIA 正式发布了 CUDA（Compute Unified Device Architecture），开启了 GPU 通用计算的新时代。CUDA 构建在统一架构之上，通过为开发者提供一个基于 C 语言扩展的编程接口，使 GPU 从图形专用硬件转变为通用计算平台。开发者可以通过 CUDA 直接访问 GPU 硬件资源，编写并行程序以处理科学计算、机器学习等领域中的高强度任务。

CUDA 的核心理念在于提供一个易于理解的并行编程模型，包括线程、线程块和网格的分层结构，以及对共享内存和全局内存的高效管理。这一模型的核心在于将程序

员编写的描述标量线程行为的代码，映射到大规模并行的底层硬件上。在实际应用中，CUDA 迅速被用于高性能计算领域，涵盖从物理模拟到深度学习的多种场景。

CUDA 的推出标志着 GPU 角色的重大转变：从专注图形渲染的硬件加速器发展为高性能计算领域的核心工具。随着 CUDA 平台的普及，研究人员也开始探索优化策略，以进一步挖掘 GPU 的计算潜力。这些优化包括提升内存访问效率、改进线程调度策略以及优化指令吞吐量。通过这些努力，GPU 的计算能力得到了持续提升，推动了深度学习、大数据处理等领域的飞速发展。CUDA 不但奠定了 GPU 通用计算的基础，更成为 GPGPU 编程模型中的经典范例。

### 1.3.3 OpenCL 的出现与标准化

#### 1. OpenCL

OpenCL（Open Computing Language）是一种开放标准的并行计算框架，于 2008 年由 Apple 公司牵头，与 AMD、IBM 等企业合作开发，并由行业协会 Khronos Group 于 2009 年正式发布。OpenCL 旨在为异构计算提供统一的编程接口，使开发者能够高效地在 CPU、GPU、FPGA 等多种硬件平台上运行并行计算任务。这一标准的出现，为通用 GPU 计算（GPGPU）提供了跨平台的解决方案，成为推动异构计算发展的重要里程碑。

OpenCL 的核心优势在于通用性和可移植性。开发者通过 OpenCL 编写一次代码，即可在不同硬件设备上运行，借助其硬件抽象层屏蔽底层实现细节，从而实现跨平台的高效计算。在这一框架中，OpenCL 为存储器模型、计算任务划分等提供了统一的抽象，简化了开发者的工作，同时支持对内存层次结构的优化，例如，全局内存和共享内存的高效利用。这些特性使 OpenCL 特别适用于处理需要大量数据并行的任务，例如，矩阵计算、信号处理和复杂仿真等。

OpenCL 提供了数据并行和任务并行两种编程机制。数据并行适合处理结构化的大规模任务，如图像处理和线性代数运算；而任务并行则支持对工作流进行细粒度划分，特别适合异构任务的协调执行。OpenCL 采用了一种基于 C99 的扩展语言——OpenCL C，允许开发者编写高效的内核程序（Kernel），充分挖掘底层硬件的并行计算能力。通过共享虚拟内存（Shared Virtual Memory，SVM）的支持，OpenCL 还显著优化了主机和计算设备之间的数据交互效率。

#### 2. 行业对 OpenCL 的支持和影响

自推出以来，OpenCL 受到了包括 AMD、Intel 和 Apple 在内的众多硬件与软件厂商的广泛支持。这些厂商不仅将 OpenCL 集成到其硬件和驱动中，还积极推动该标准的改进和推广。在 Khronos Group 的持续维护下，OpenCL 的功能得到了不断增强，其最新版本 3.0 在优化性能的同时，进一步提高了对现代计算需求的适应性，引入了一系列新功能，如共享虚拟内存（SVM）、基于 C++ 的类接口以及可移植队列等。

OpenCL 的行业支持推动了 GPGPU 生态系统的扩展，同时增强了其在异构计算中的影响力。尽管 CUDA 在专有 GPU 平台中占据主导地位，但 OpenCL 凭借其开放性和跨平台特性，成为异构计算领域的重要力量。通过为不同硬件设备提供统一的接口，OpenCL 不仅加速了计算密集型任务的解决，还促进了异构计算的标准化进程。

### 1.3.4 计算着色器的引入与 GPGPU 的成熟

从 2009 年开始，计算着色器（Compute Shader）的引入标志着 GPU 技术的一次重大转型。微软在 DirectX 11 中首次将计算着色器作为图形 API 的正式组成部分，允许开发者直接通过图形管线调用 GPU 的通用计算能力。这种设计打破了传统着色器仅服务于特定渲染阶段的限制，为执行非图形任务（如物理模拟、粒子系统、后处理效果）提供了一种灵活高效的方式。计算着色器通过直接整合到现有的图形 API（如 DirectX 和 OpenGL），让图形应用开发者无须额外学习 CUDA 或 OpenCL，即可利用 GPU 的并行计算能力。

这一时期，硬件架构也为计算着色器的普及提供了坚实基础。2010 年，NVIDIA 的 Fermi 架构大幅提升了对计算任务的支持，优化了线程调度和共享内存性能，为计算着色器提供了更高的执行效率。AMD 在 2011 年推出的 TeraScale 2 和 VLIW4 架构，也通过支持 DirectX 11 和多线程模型进一步增强了 GPU 的并行计算能力。这些技术进步使得计算着色器能够高效处理复杂任务，如动态光影、体积渲染和后期特效。

2012 年，Khronos Group 在 OpenGL 4.3 中支持了计算着色器，进一步推动了这一技术的跨平台普及。开发者可以使用相同的编程接口，在不同厂商的硬件上运行计算任务。这种整合性使得计算着色器迅速成为实时渲染和通用计算的桥梁，从而推动了游戏引擎（如虚幻引擎和 Unity）对复杂视觉效果的支持，也使科学计算、数据处理等领域受益。

随着计算着色器的广泛应用，GPU 进入了一个新的发展阶段。计算着色器扩展了图形 API 的功能，与 CUDA、OpenCL 等专用计算平台形成了互补关系。它的灵活性和易用性让 GPU 成为通用计算的主力设备，也为现代 API（如 Vulkan 和 DirectX 12）的低开销设计以及异构计算的兴起奠定了基础。

## 1.4 光线追踪与 AI 时代

光线追踪和人工智能（Artificial Intelligence，AI）技术的崛起为 GPU 带来了全新的发展方向。这两项技术在性能与视觉效果上的革命性进步，使 GPU 从传统图形渲染硬件逐步演变为多领域计算的核心平台。光线追踪的实时化和 AI 计算的快速普及，也驱动了 GPU 架构设计的深刻变革。

### 1.4.1 光线追踪的崛起：从离线到实时

光线追踪（Ray Tracing）是一种通过模拟光线传播与物体交互的物理过程生成图像的渲染技术，能够实现高度逼真的光影效果。传统的光栅化方法通过将 3D 场景中的几何体投影到 2D 屏幕上，利用着色器逐像素填充颜色，虽然计算效率极高，但在复杂光照场景（如全局光照和真实反射）中效果有限。而光线追踪直接从光的传播路径入手，可以自然实现精确的反射、折射和阴影效果，因此长期被视为图形学中的“理想目标”。

然而，由于光线追踪的计算复杂度极高，在很长一段时间内，它只能用于电影特效和动画等离线渲染领域。

2018 年，NVIDIA 在其 Turing 架构中引入专门的 RT Cores（Ray Tracing Cores），这

是实时光线追踪得以实现的关键突破。这些硬件单元显著加速了光线与场景几何体相交的运算，并优化了 BVH（Bounding Volume Hierarchy）树的构建和遍历过程。同时，微软通过 DirectX 12 引入 DXR（DirectX Raytracing）标准化了光线追踪的编程接口，使开发者能够更容易地在游戏和其他实时应用中集成这一技术。结合传统光栅化的混合渲染方式，实时光线追踪逐渐成为次世代游戏引擎和设计工具的核心功能。

这一技术的应用拓展不仅提升了游戏画面的真实性，也为影视和建筑设计等领域带来了更快的工作流。例如，虚幻引擎的实时光线追踪功能缩短了设计师和艺术家的迭代周期，让创意更加高效地转化为视觉内容。

### 1.4.2 AI 驱动的 GPU 变革

AI 技术、深度学习的快速发展，对计算性能提出了极高的需求。从 AlexNet 的成功开始，GPU 因其大规模并行处理能力，成为训练深度学习模型的主力硬件。这一趋势推动了 GPU 设计向 AI 任务方向的进一步优化。

2017 年，NVIDIA 推出了 Volta 架构，并引入 Tensor Core，这是专门为深度学习中的张量运算设计的硬件单元。通过让同一个线程束（Warp）内的线程在一定程度上共享数据，打破了原有 SIMT 编程模型的边界，成功地将 DSA（Domain Specific Architecture）与 SIMT 编程模型结合起来。Tensor Core 支持混合精度计算（如 FP16 和 FP32），能够显著加速矩阵乘法等关键操作。这一设计极大地提升了神经网络的训练和推理效率，也为大规模 AI 模型的快速发展提供了硬件基础。

与此同时，软件生态的完善进一步降低了 AI 开发的门槛。NVIDIA 通过 CUDA 优化了深度学习框架（如 TensorFlow 和 PyTorch）的性能，推出了 cuDNN、cutlass 等专用库，大幅提升了神经网络的运行效率。而 AMD 和 Intel 也开始在各自的 GPU 架构中引入 AI 优化单元，推动 AI 加速的多元化发展。

AI 技术不仅在大模型、科学计算、自动驾驶和医疗等领域大放异彩，还与图形渲染紧密结合。以 NVIDIA 的 DLSS（深度学习超采样技术）为例，它通过 AI 模型对低分辨率图像进行高质量重建，显著提升了游戏帧率与画面质量。实时光线追踪为了保持性能，往往需要减少光线采样，而 AI 模型可以通过训练生成高质量的光影结果，从而弥补采样不足的问题。这种技术极大地提高了实时光线追踪的效率和效果。

## 1.5 GPU 的未来：架构演化与技术创新

现代 GPU 架构正在向更高性能、更低能耗和更强灵活性的方向演化。

### 1.5.1 硬件架构的持续演化

Ampere 架构（2020 年）：作为 NVIDIA 在 RTX 系列中的重要升级，Ampere 架构在光线追踪和 AI 计算领域均有显著改进。其 RT Cores（光线追踪核心）和 Tensor Cores（张量计算核心）的性能相比上一代提升了一倍以上，同时支持 TF32 等混合精度计算，显著提升了 AI 训练效率和光线追踪场景中的实时性能。

Hopper 架构（2022 年）：Hopper 架构引入了 Transformer Engine 软件库，专为大规模

AI 模型优化，能够加速 Transformer 类模型的训练和推理。其专用硬件单元针对矩阵计算和深度学习中的注意力机制进行优化，成为生成式 AI（如 ChatGPT）的重要计算平台。

Blackwell 架构（2024 年）：NVIDIA Blackwell 架构是其最新一代面向高性能计算（HPC）和人工智能（AI）任务的 GPU 架构，专注于提升计算效率和系统可扩展性。该架构引入了第二代 Transformer 引擎，通过支持 FP4 精度和微张量缩放（Microscaling）技术，显著优化了大型语言模型（LLM）和专家混合模型（MoE）的推理与训练性能。此代架构引入 Tensor Memory，进一步将 DSA 的思想应用于 GPU 中。

AMD RDNA/CDNA 架构：AMD 的 RDNA 系列架构在光线追踪和 AI 计算方面持续改进。RDNA 3 架构的产品采用 Chiplet 设计，将多个计算单元模块化封装，从而提升了能效比。此外，RDNA 3 在硬件上优化了光线追踪性能，同时为 AI 推理任务提供了更高效的计算单元。与 RDNA 架构不同，CDNA 完全剥离了光栅化、光线追踪等图形功能模块，优化了计算单元（Compute Units，CU）和矩阵运算单元（Matrix Cores），专注于通用计算任务。

Intel Arc GPU：Intel 进入独立显卡市场后推出了 Arc 系列，主打光线追踪和 AI 加速。这些 GPU 为入门级市场竞争注入了新的活力。

### 1.5.2 网格着色器

网格着色器（Mesh Shader）是一种全新的几何处理模型，它替代了传统图形管线中的几何着色器（Geometry Shader）、细分着色器（Tessellation Shader）和部分顶点着色器（Vertex Shader）的功能。通过引入两个关键阶段——任务着色器（Task Shader）和网格着色器（Mesh Shader），这一模型在几何处理上具备更高的灵活性和并行效率。任务着色器负责生成任务并分配给网格着色器。开发者可以在此阶段执行裁剪（Culling）等优化操作，减少不必要的几何处理任务。网格着色器以工作组（Work Group）的形式并行生成几何数据，并将其直接传递给光栅化阶段。

相比几何着色器，网格着色器能批量处理几何基元，提高了吞吐量。它支持复杂的 LOD（Level Of Detail）技术，以及动态生成和实例化几何。通过整合多个阶段，网格着色器降低了传统几何处理管线的复杂性，同时增强了开发者的控制能力。网格着色器在大规模场景渲染、动态粒子系统和自定义几何生成中展现出了卓越的性能。例如，在开放世界游戏中，网格着色器可以动态生成低细节远景模型，并在需要时逐渐增加细节，从而优化性能。

### 1.5.3 封装和互连技术的演进

GPU 性能的持续增长不仅依赖于芯片内部计算能力的提升，还需要在封装与互连技术上不断创新，以解决多芯片协作、高速通信和能效比的挑战。现代封装技术通过模块化设计整合不同类型的芯片，而互连技术则在 GPU 之间以及 GPU 与其他硬件之间搭建了高带宽、低延迟的数据传输通道。

**1. 封装技术的发展**

早期 GPU 通常采用单芯片（Monolithic Die）设计，将所有功能集成在一颗芯片内。然而，随着工艺节点的缩小和芯片面积的增大，单芯片设计面临诸多挑战：大面积芯片

更易受到制造缺陷的影响，导致良率显著降低；单一芯片设计限制了不同功能模块之间的带宽扩展，增加了数据传输延迟；随着芯片复杂度的提升，热设计和散热管理变得愈发困难，带来了热设计功耗（TDP）问题。

1）2.5D 封装与 CoWoS 技术

为克服单芯片设计的局限性，2.5D 封装技术应运而生，其中，CoWoS（Chip on Wafer on Substrate）是一个典型的代表。CoWoS 通过硅中介层（Silicon Interposer）将多个芯片模块（如 GPU 逻辑芯片和 HBM 高带宽内存）集成在一个封装内。

CoWoS 相比单芯片拥有高带宽低延迟的特性，硅中介层内的数千条互连通道显著提高了芯片之间的数据带宽，并降低了延迟。在一个芯片系统中，逻辑芯片和内存芯片可以分别采用不同的工艺节点制造，从而优化成本和性能。一些现代 GPU 产品，如 NVIDIA A100、H100，以及 AMD MI200、MI300 等高性能加速器均采用 CoWoS 封装技术。

2）3D 堆叠与 HBM 内存

进一步提高集成度和性能的方式是 3D 堆叠技术，如图 1.15 所示。通过 TSV（Through-Silicon Via，硅通孔），多个存储层可以垂直堆叠在逻辑芯片之上，形成紧凑的 3D 封装。其典型应用是 HBM（High Bandwidth Memory，高带宽内存），将多个内存层垂直堆叠并通过 TSV 与逻辑芯片连接，逻辑芯片充当内存控制器的角色，负责与外部处理器（如 GPU 或 CPU）之间的通信，同时集成了电源管理功能。HBM 也广泛应用于现代高性能 GPU 中，由于 HBM 和逻辑芯片仍然在平面板卡上集成，通常将 HBM+GPU 称为 2.5D 架构。

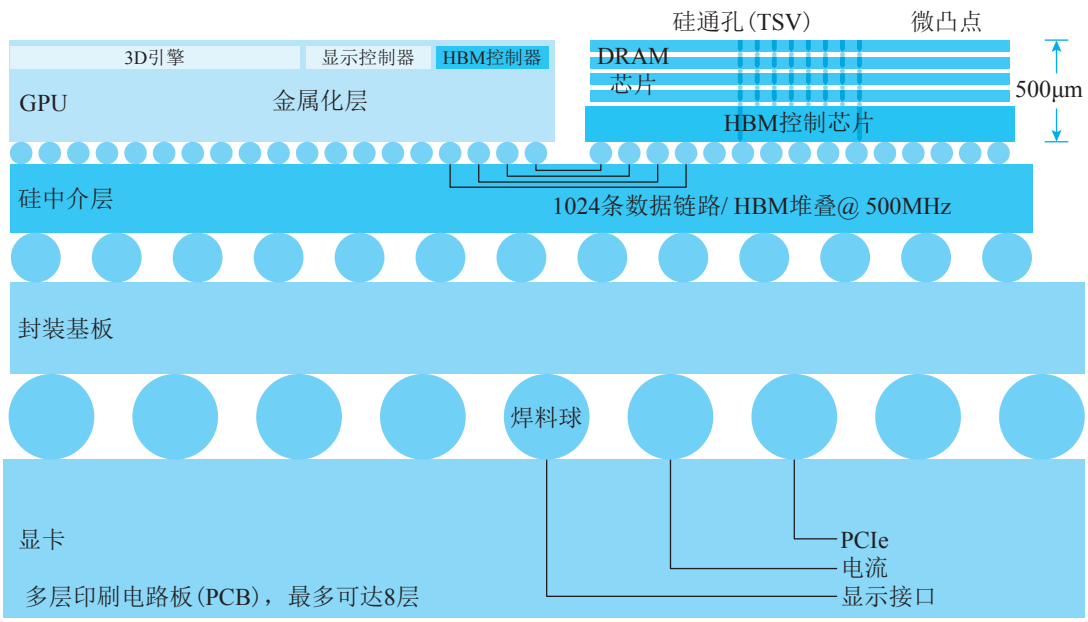


图 1.15　3D 堆叠 HBM 与 2.5D GPU 封装

此外，在 3D 堆叠的应用上，近年来一些新兴设计方案提出将 GPU 的计算逻辑直接集成到 HBM 的逻辑芯片上，利用引线键合或 TSV 连接，实现垂直架构。这一设计结合了存储和计算的功能，旨在解决传统 GPU 架构中的存储带宽和延迟瓶颈。逻辑芯片和存

储芯片直接通过 TSV 连接，能够获得比 HBM 更高的带宽，有效应对大模型推理过程的内存墙。

3）Chiplet 与 MCM 设计

近年来，多芯片模块（MCM）和芯粒（Chiplet）设计成为封装技术的趋势。通过将不同功能模块拆分为多个小 Chiplet，并在封装中重新集成，MCM 设计解决了单芯片的良率和成本问题。模块化设计降低了生产成本，同时允许不同模块独立升级。例如，AMD MI300X 采用 MCM 设计，将多个计算芯片（基于 CDNA 3 架构）与 HBM3 高带宽内存集成在一个封装中。NVIDIA 在 Hopper GPU 中仍然采用传统的单芯片设计（Monolithic Die），省去了芯粒之间的互连延迟，这体现了其追求极致性能和最低通信延迟的策略；不过，在 Blackwell GPU 中，NVIDIA 采用了两个 Die 互连的结构。

**2. 互连技术的演进**

PCIe（Peripheral Component Interconnect eXpress）长期以来是 GPU 与主板、其他设备通信的主要接口，但其带宽和延迟逐渐成为高性能计算的瓶颈。尽管 PCIe Gen4 和 Gen5 提供了高达 64GB/s 和 128GB/s 的双向带宽，但相比专用互连技术，其带宽仍偏低，不支持多 GPU 高效协作。

为突破 PCIe 的限制，NVIDIA 推出了 NVLink 互连技术，用于多 GPU 间的高速通信。最新一代 NVLink（NVLink 5）带宽达到 1800GB/s，比 PCIe 快数倍。通过 NVSwitch，NVLink 可以支持多达 72 个 GPU 组成的全互连网络，实现高达 130TB/s 的总带宽。NVLink 和 NVSwitch 广泛应用于 NVIDIA DGX 系统，为训练大型 AI 模型提供高效通信支持。

CXL（Compute eXpress Link）是一种基于 PCIe 物理层的新型高速互连协议，旨在解决异构计算系统中 CPU、GPU、内存和专用加速器之间的通信瓶颈，由 CXL 联盟主导开发，最初由 Intel 提出，现已成为广泛采用的行业标准。CXL 扩展了 PCIe 的功能，通过支持缓存一致性（CXL.cache）和共享内存访问（CXL.memory），多个设备能够高效协作，并共享统一的内存空间。与 PCIe 相比，CXL 的延迟更低，并显著优化了设备间的数据传输效率和资源利用率。其灵活的架构适用于多设备、多节点的分布式计算环境，是构建下一代数据中心和 AI 系统的关键技术之一。

UCIe（Universal Chiplet Interconnect eXpress）是一种新兴的标准化互连技术，旨在促进 Chiplet 设计的开放生态。它为芯粒间的物理互连和协议层提供了统一的标准，支持不同供应商的芯粒在同一封装中互操作。UCIe 旨在解决多芯粒设计中的互连兼容性问题，通过高效的短距离通信和低功耗传输，推动芯粒集成技术的发展。UCIe 的主要特点包括对物理接口和高层协议的统一定义，特别是在高带宽、高效率和低延迟的数据传输上提供了优化。它支持现有的传输协议，如 PCIe 和 CXL，从而确保了兼容性，并为广泛应用场景提供灵活的解决方案。此外，UCIe 的开放性设计使得不同厂商的芯粒可以轻松集成进同一平台，从而降低了多厂商合作和系统设计的成本。

在高速互连中，SerDes（Serializer/Deserializer，串并转换器）技术是通信的核心。它是一种将数据从并行形式转换为串行形式（发送端），或将串行数据转换回并行形式（接收端）的技术，用于芯片间的高速数据通信。在芯片之间或芯片内通信时，数据传输

的并行总线方式曾经占主导地位。然而，随着频率提高，传统并行总线的信号的串扰和噪声问题显著增加；并行总线需要大量信号线，占用更多空间并增加复杂性。SerDes 上的串行数据通过单根信号线传输，以更高频率实现高带宽，同时避免了并行传输的同步问题。它的高效运行依赖于自适应均衡技术和信号恢复算法，以应对高速传输中的信号损失。

**3. GPU 互连系统**

现代 GPU 在面对大规模 AI 模型时，尤其是在训练和推理过程中，需要处理的计算负载变得越来越复杂。随着模型规模的不断扩大，单个 GPU 的计算和内存资源往往不足以满足需求，这就催生了多 GPU 和多节点系统的应用场景。

首先，大型 AI 模型通常会被拆分成多个子任务，分布在多个计算节点（例如，多个流式多处理器（SM）组成的芯片组或多颗 GPU 的集群）中进行并行计算。这些子单元之间需要高效的通信机制来传递数据和同步计算结果。因此，GPU 的互连系统在数据传输和协同计算方面起着关键作用。

此外，在 GPU 集群中，节点间的通信还涉及网络结构和路由算法的选择。在大规模系统中，通信延迟和带宽限制可能会成为性能瓶颈。例如，当一个计算任务需要跨多个 GPU 进行数据交换时，数据如何通过网络传输，以及如何避免网络拥堵，直接影响着整体性能。因此，高效的路由算法和网络拓扑设计变得尤为重要。常见的拓扑如全互连网络（全连接的 NVLink 网络）和分层网络（如通过 NVSwitch 进行拓展的 NVLink）都可以优化节点间的通信效率。随着系统规模的扩大，出现了采用分布式网络架构，利用动态路由和自适应调度机制来应对网络负载的变化。

内存访问方面，大型 AI 模型对内存带宽和访问延迟有着极高的要求。现代 GPU 系统通常采用统一内存架构（UMA）来进行内存管理，但物理内存的访问速度和延迟也受到限制，因此需要通过高效的内存调度、缓存优化和分布式内存管理策略来减少数据传输的瓶颈。而在一些 NUMA（非统一内存访问）系统中，访问内存的延迟是非对称的，这意味着某些节点访问本地内存的速度比访问远程内存要快得多，需要更加复杂的软硬件协同手段来优化数据流。在多 GPU 系统中，如果模型和数据分布不均，可能会出现内存访问的热点，从而引发性能瓶颈。

## 1.5.4 GPGPU 技术的挑战

GPGPU 在诸多领域展现了强大的并行计算能力，在开拓应用中遇到了许多瓶颈和难点，需要业界加强自身核心竞争力的同时注重合作共赢。

（1）编程模型的复杂性：并行编程的难度仍然不可忽视，需要开发者具备深厚的专业知识，同时为提升 GPGPU 编程效率，新的工具和抽象层如高级语言和自动优化编译器也不可或缺。近年来，为了配合 AI 技术的演进，出现了如 triton、cuTile 等 tile-level 编程框架，虽然进一步简化了编程难度，但是面临着诸多 DSA、NPU 相同的问题：适用范围受限、性能调优困难。

（2）能耗和散热管理的问题：随着 GPGPU 性能不断提升，能耗和散热管理逐渐成为制约芯片高性能运行的一大难题。

（3）新型计算架构的出现：随着神经网络处理器（NPU）等新型计算架构的快速发

展，高性能计算领域的竞争愈发激烈，GPGPU 需要在现有 SIMT 编程框架和并行计算优势基础上持续创新。

（4）AI 系统部署与推理框架：随着 AI 应用的广泛推广，GPGPU 在 AI 系统部署中发挥了重要作用。与硬件更紧密耦合的训练和推理框架，如 Megatron 和 vLLM 等，能够在大规模分布式计算和并行处理任务中实现更加高效的推理过程。然而，如何优化这些框架以充分发挥 GPGPU 的并行计算能力，减少延迟并提高吞吐量，仍然是一个亟待解决的挑战。需要针对不同的硬件架构和应用需求，设计定制化的推理框架和接口，以提升 AI 系统的整体性能。

（5）安全性和可靠性：在关键任务与安全领域中，数据泄露和恶意攻击对运行模型的安全性、完整性构成威胁。为此，需要进一步加强 GPGPU 在数据保护、容错性以及访问控制等方面的机制，确保其在高可靠性场景下的适用性。

（6）行业标准和兼容性：GPGPU 技术需要实现跨平台的支持和优化，以确保在不同软硬件平台上都能发挥最佳性能。供应商之间的竞争与合作并存，注重行业标准制定和兼容性，加强合作共同推动 GPGPU 技术的发展。

（7）人才培养与社区建设：为推动 GPGPU 技术的普及和创新，人才培养是关键环节。通过支持开源项目、建设技术社区、提供资源共享与经验交流平台，可降低技术门槛，并促进统一技术标准的形成，进一步推动 GPGPU 技术的广泛应用。

## 1.6 本章小结

本章从早期计算机图形的发展历程出发，介绍了图形硬件从最初的固定功能图形控制器到现代可编程 GPU 的演进过程。随着可编程着色器的引入和编程模型的不断完善，GPU 不再局限于图形渲染，而逐步成为通用计算平台。在统一渲染架构、CUDA、OpenCL 以及计算着色器的推动下，GPU 的并行计算能力在科学计算、AI 等领域得到广泛应用。近年来，实时光线追踪与 AI 技术的崛起，进一步推动了 GPU 在硬件架构与生态系统层面的深度变革，从 RT Cores、Tensor Cores 到新型封装与互连技术，都体现出高性能计算与低能耗并行的发展趋势。面对快速演进的行业需求，GPGPU 技术在编程模型、能效、生态标准等方面仍面临诸多挑战，但其在大规模并行计算中的核心地位已不可撼动，并将在未来持续引领计算架构的创新与变革。