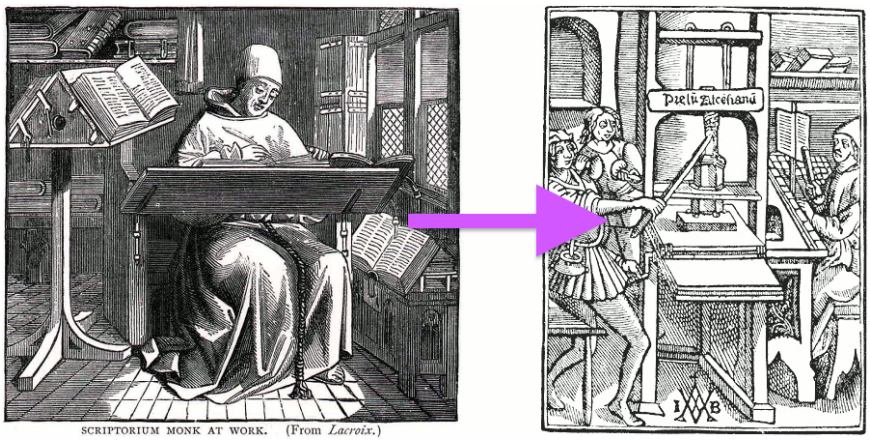
The Book of Shaders by *Patricio Gonzalez Vivo & Jen Lowe*

开始

什么是 Fragment Shader(片段着色器)?

在之前的章节我们把 shaders 和古腾堡印刷术相提并论。为什么这样类比呢？更重要的是，什么是 shader?



From Letter-by-Letter, Right: William Blades (1891). To Page-by-page, Left: Rolt-Wheeler (1920).

如果你曾经有用计算机绘图的经验，你就知道在这个过程中你需要画一个圆，然后一个长方形，一条线，一些三角形……直到画出你想要的图像。这个过程很像用手写一封信或一本书 —— 都是一系列的指令，需要你一件一件完成。

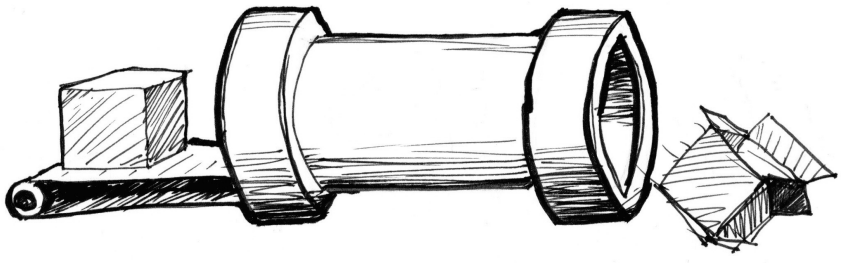
Shaders 也是一系列的指令，但是这些指令会对屏幕上的每个像素同时下达。也就是说，你的代码必须根据像素在屏幕上的不同位置执行不同的操作。就像活字印刷，你的程序就像一个 function（函数），输入位置信息，输出颜色信息，当它编译完之后会以相当快的速度运行。

Chinese movable type

为什么 shaders 运行特别快？

为了回答这个问题，不得不给大家介绍并行处理（parallel processing）的神奇之处。

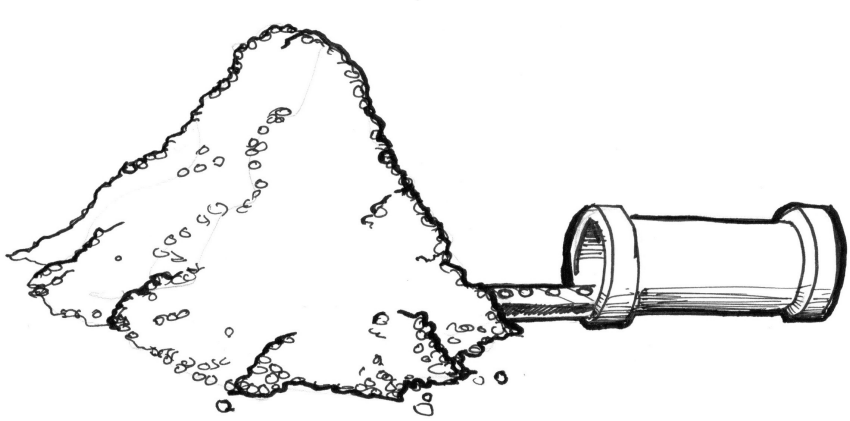
想象你的 CPU 是一个大的工业管道，然后每一个任务都是通过这个管道的某些东西 —— 就像一个生产流水线那样。有些任务要比别的大，也就是说要花费更多时间和精力去处理。我们就称它要求更强的处理能力。由于计算机自身的架构，这些任务需要串行；即一次一个地依序完成。现代计算机通常有一组四个处理器，就像这个管道一样运行，一个接一个地处理这些任务，从而使计算机流畅运行。每个管道通常被称为线程。



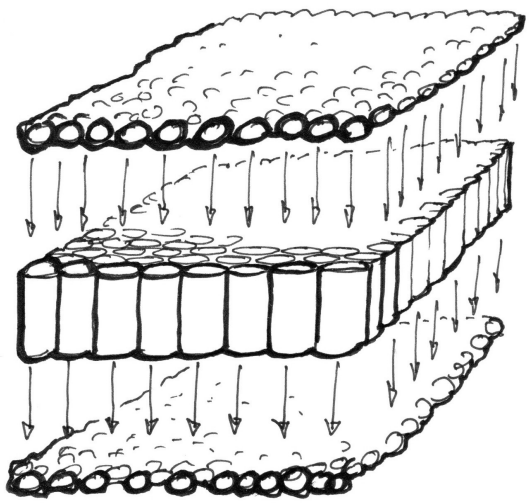
CPU

视频游戏和其他图形应用比起别的程序来说，需要高得多的处理能力。因为它们的图形内容需要操作无数像素。想想看，屏幕上的每一个像素都需要计算，而在 3D 游戏中几何和透视也都需要计算。

让我们回到开始那个关于管道和任务的比喻。屏幕上的每个像素都代表一个最简单的任务。单独来看完成任何一个像素的任务对 CPU 来说都很容易，那么问题来了，屏幕上的每一个像素都需要解决这样的小任务！也就是说，哪怕是对于一个老式的屏幕（分辨率 800x600）来说，都需要每帧处理480000个像素，即每秒进行14400000次计算！是的，这对于微处理器就是大问题了！而对于一个现代的 2800x1800 视网膜屏，每秒运行60帧，就需要每秒进行311040000次计算。图形工程师是如何解决这个问题的？



这个时候，并行处理就是最好的解决方案。比起用三五个强大的微处理器（或者说“管道”）来处理这些信息，用一大堆小的微处理器来并行计算，就要好得多。这就是图形处理器（GPU : Graphic Processor Unit)的来由。



GPU

设想一堆小型微处理器排成一个平面的画面，假设每个像素的数据是乒乓球。14400000个乒乓球可以在一秒内阻塞几乎任何管道。但是一面800x600的管道墙，

每秒接收30波480000个像素的信息就可以流畅完成。这在更高的分辨率下也是成立的 —— 并行的处理器越多，可以处理的数据流就越大。

另一个 GPU 的魔法是特殊数学函数可通过硬件加速。非常复杂的数学操作可以直接被微芯片解决，而无须通过软件。这就表示可以有更快的三角和矩阵运算 —— 和电流一样快。

GLSL是什么？

GLSL 代表 openGL Shading Language，openGL 着色语言，这是你在接下来章节看到的程序所遵循的具体标准。根据硬件和操作系统的不同，还有其他的着色器（shaders)。这里我们将依照Khronos Group的规则来执行。了解 OpenGL 的历史将有助于你理解大多数奇怪的约定，所以建议不妨阅读openglbook.com/chapter-0-preface-what-is-opengl.html。

为什么 Shaders 有名地不好学？

就像蜘蛛侠里的那句名言，能力越大责任越大，并行计算也是如此；GPU 的强大的架构设计也有其限制与不足。

为了能使许多管线并行运行，每一个线程必须与其他的相独立。我们称这些线程对于其他线程在进行的运算是“盲视”的。这个限制就会使得所有数据必须以相同的方向流动。所以就不可能检查其他线程的输出结果，修改输入的数据，或者把一个线程的输出结果输入给另一个线程。允许数据在线程之间线程流动会使数据的整体性面临威胁。

并且 GPU 会让所有并行的微处理器（管道们）一直处在忙碌状态；只要它们一有空闲就会接到新的信息。一个线程不可能知道它前一刻在做什么。它可能是在画操作系统界面上的一个按钮，然后渲染了游戏中的一部分天空，然后显示了一封 email 中的一些文字。每个线程不仅是“盲视”的，而且还是“无记忆”的。同时，它要

求编写一个通用的规则，依据像素的不同位置依次输出不同的结果。这种抽象性，和盲视、无记忆的限制使得 shaders 在程序员新手中不是很受欢迎。

但是不要担心！在接下来的章节中，我们会一步一步地，由浅入深地学习着色语言。如果你是在用一个靠谱的浏览器阅读这个教程，你会喜欢边读边玩书中的示例的。好了，不要再浪费时间了，赶快去玩起来吧！ 点击 **Next >>** 开启 shader 之旅！

< < Previous *Home* *Next > >*