3.3 可编程着色器阶段

现代着色器程序使用统一的着色器设计。这意味着顶点、像素、几何和曲面细分相关的着色器共享一个通用的编程模型。在内部，它们具有相同的指令集架构(ISA)。实现此模型的处理器在DirectX中称为通用着色器内核，具有此类内核的 GPU被称为具有统一着色器架构。这种架构背后的想法是着色器处理器可用于各种角色，GPU可以根据需要分配这些角色。例如，与由两个三角形组成的大正方形相比，具有小三角形的一组网格需要更多的顶点着色器处理。具有单独的顶点和像素着色器核心池的GPU意味着保持所有核心忙碌的理想工作分配是严格预先确定的。使用统一的着色器核心，GPU可以决定如何平衡此负载。

描述整个着色器编程模型远远超出了本书的范围，并且有许多文档、书籍和网站已经这样做了。着色器使用类似C的着色语言进行编程，例如DirectX的高级着色语言(HLSL)和OpenGL着色语言 (GLSL)。DirectX的HLSL可以编译为虚拟机字节码，也称为中间语言（IL或DXIL），以提供硬件独立性。"中间"表示还可以允许离线编译和存储着色器程序。该中间语言由驱动程序转换为特定GPU的ISA。控制台编程通常会避免中间语言步骤，因为系统只有一个ISA。

基本数据类型是32位单精度浮点标量和向量，尽管向量只是着色器代码的一部分，并且如上所述不受硬件支持。在现代 GPU上，本机也支持32位整数和64位浮点数。浮点向量通常包含位置(xyzw)、法线、矩阵行、颜色(rgba)或纹理坐标 (uvwq)等数据。整数最常用于表示计数器、索引或位掩码。还支持聚合数据类型，例如结构体、数组和矩阵。

绘制命令调用图形API来绘制一组图元，从而使得图形管线执行并运行其着色器。每个可编程着色器阶段都有两种类型的输入：统一(uniform)输入，其值在整个绘制调用期间保持不变（但可以在绘制调用之间更改），以及变化(varying)的输入，来自三角形顶点或光栅化的数据。例如，像素着色器可以将光源的颜色作为统一(uniform)值提供，并且三角形表面的位置每个像素都会发生变化，因此也会发生变化。纹理是一种特殊的统一(uniform)输入，曾经是应用于表面的彩色图像，但现在可以将其视为任何大型数据数组。

底层虚拟机为不同类型的输入和输出提供特殊寄存器。用于uniform的可用常量寄存器的数量远大于可用于varying输入或输出的那些寄存器。发生这种情况是因为需要为每个顶点或像素单独存储不同的输入和输出，因此需要多少个自然是有限制的。uniform输入存储一次，并在绘制调用中的所有顶点或像素中重复使用。虚拟机还具有通用临时寄存器，用于暂存空间。所有类型的寄存器都可以使用临时寄存器中的整数值进行数组索引。着色器虚拟机的输入和输出如图3.3所示。

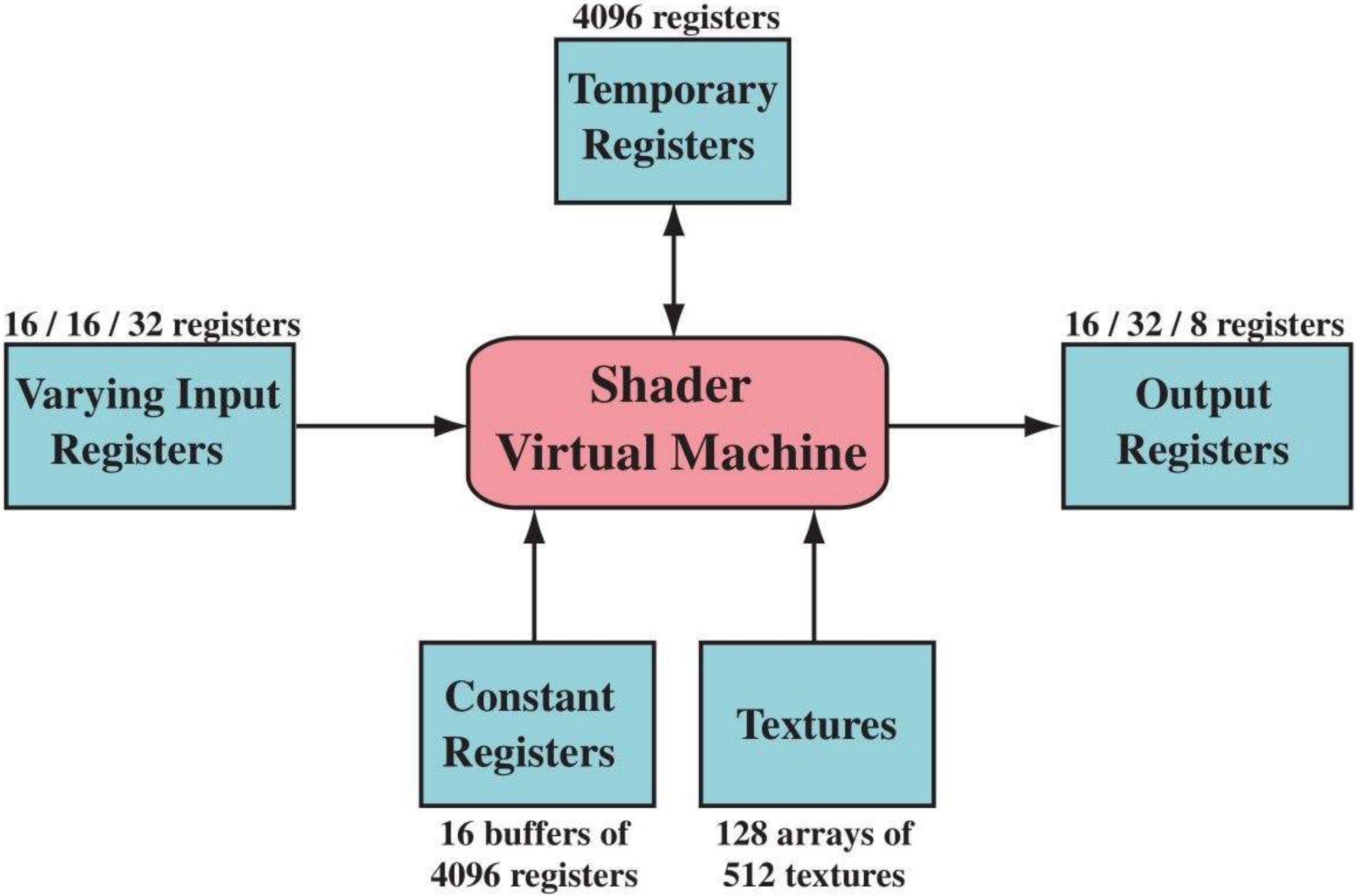
4096 registers
Temporary Registers
16 / 16 / 32 registers
Varying Input Registers
Shader Virtual Machine
16 / 32 / 8 registers
Output Registers
Constant Registers
Textures
16 buffers of 4096 registers
128 arrays of 512 textures

图3.3. 统一虚拟机架构和寄存器布局，Shader Model 4.0下。最大可用数量显示在每个资源旁边。由斜杠分隔的三个数字表示顶点、几何和像素着色器的限制（从左到右）。

图形计算中常见的操作可以在现代GPU上高效执行。着色语言通过诸如$*$和$+$之类的运算符支持了这些操作中最常见的操作（例如加法和乘法）。还有其他的内部函数接口，例如$atan()$、$sqrt()$、$log()$和许多其他为GPU优化的函数。还存在用于更复杂操作的函数，例如向量归一化和反射、叉积以及矩阵转置和行列式计算。

“流控制”这个术语是指使用分支指令来改变代码执行的流程。与流控制相关的指令用于实现高级语言结构，例如“if”和“case”语句，以及各种类型的循环。着色器支持两种类型的流控制。静态流控制分支基于统一输入的值。这意味着代码流在绘制调用中是恒定的。静态流控制的主要好处是允许在各种不同情况下使用相同的着色器（例如，不同数量的灯光）。没有线程发散，因为所有调用都采用相同的代码路径。动态流控制基于不同输入的值，这意味着每个片元可以不同地执行代码。这比静态流控制强大得多，但会降低性能，特别是如果代码流在着色器调用之间发生不规律变化。