

在多态阵列处理器上实现统一渲染架构

郭 丹, 韩俊刚

(西安邮电大学 计算机学院, 陕西 西安 710121)

摘 要:多态阵列处理器是一种将成千上万的单处理核集成于一块芯片的新型并行处理器,其具有多种并行计算模式。针对传统的分离式图形渲染管线产生负载不均衡的问题,以业界提出的统一渲染思想为指导,充分利用多态阵列处理器的多种并行计算模式,设计并实现了一种基于多态阵列处理器的统一渲染架构。该设计将多态阵列处理器的单处理核作为统一渲染架构中的流处理器,同时设计缓冲区对象、缓存机制、分配器、重定序等相关机制配合流处理器工作,以此达到对图形顶点数据和像素数据统一渲染的目的。最后在实现的统一渲染架构中对基本的 3D 模型数据进行了仿真。结果表明,设计的统一渲染架构很好地实现了顶点着色和像素着色的负载平衡,并且在数据处理方面实现了较高的数据并行性。

关键词:统一渲染;多态阵列处理器;着色器;分配器

中图分类号:TP302

文献标识码:A

文章编号:1673-629X(2016)08-0039-04

doi:10.3969/j.issn.1673-629X.2016.08.008

Implementation of an Unified Rendering Architecture on Polymorphism Array Processor

GUO Dan, HAN Jun-gang

(School of Computer and Technology, Xi'an University of Posts and Telecommunications,
Xi'an 710121, China)

Abstract: Polymorphism array processor is a kind of new parallel processor which integrates hundreds or thousands of single nuclear on one chip. It has many parallel computing model. For the problem of uneven load produced by the traditional separate graphics rendering pipeline, guided by the ideology of unified rendering by industry, making full use of varieties of parallel computing model of polymorphism array processor, an unified rendering architecture based on the polymorphism array processor is designed and implemented. It takes single nuclear of polymorphism array processor as a stream processor of unified rendering architecture, and designs the buffer object, cache mechanisms, dispatcher, reorder mechanism and so on to cooperate the works of stream processor, achieving the purpose of rendering the vertex data and pixel data in unified manner. Finally on this unified rendering architecture, the data of the basic 3D model is tested. The experimental results show that the unified rendering architecture designed has realized the load balancing between vertex shader and pixel shader, and achieved the higher parallelism in the aspect of data processing.

Key words: unified rendering; polymorphism array processor; shaders; dispatcher

0 引 言

进入可编程时代的 GPU 可以对顶点处理单元和像素处理单元两个部分进行编程。在顶点处理单元和像素处理单元上运行的程序分别称为顶点着色器(vertex shader)和像素着色器(pixel shader)^[1]。微软 DirectX10 发布之前, GPU 采用将顶点处理单元和像素处理单元分开的分离式渲染架构^[2], 当图形流水线采用分离式渲染架构时, 会产生负载不均衡的现象^[3]。

在 DirectX10 时代图形界引入了统一渲染架构^[4], 即在统一渲染架构中, 不再有顶点处理单元和像素处理单元之分, 取而代之的是支持通用计算^[5]的流处理器阵列。流处理器不区分所处理的数据, 对于顶点和像素的处理也只是当作普通的数据处理, 因此, 统一渲染架构下的 GPU 具备了进行通用计算的基础。

文中研究了统一渲染架构在多态同构阵列处理器 (Polymorphic Array Architecture for Graphics,

收稿日期: 2015-11-06

修回日期: 2016-03-04

网络出版时间: 2016-06-22

基金项目: 国家自然科学基金重大项目(61136002); 西安邮电大学研究生创新基金(CXL2014-28)

作者简介: 郭 丹(1991-), 女, 硕士研究生, 研究方向为并行计算与图形学; 韩俊刚, 教授, 研究方向为软件和硬件的形式化验证、图形处理器和新型计算机体系结构研究。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20160622.0845.050.html>

PAAG)^[6]上的实现方法。

1 多态同构阵列处理器

PAAG 由若干个基本处理簇和特定功能单元组成,16 个简单处理器单元 (Processing Element, PE) 按 4×4 的二维阵列组成一个簇。每个簇中包含一个簇控制器、一个全局共享存储、四个行控制器和四个列控制器。簇控制器可以向列控制器和行控制器发送指令或者信号,然后这些指令或信号被行控制器或列控制器通过路由器发送至 PE。相反的路程可以是 PE 的执行状态向簇控制器的反馈。最小的处理器单元 PE 包含一个算术逻辑单元、一个路由器、一个 PE 控制器、一个本地数据存储、一个本地指令存储器和四个与相邻处理器共享的共享存储器。

PE 可单独执行程序,相互之间互不影响,也可以是相邻的几个 PE 或者一组指定的 PE 协作执行一段程序。同属于一个簇的 PE 都可以从全局共享存储器存取数据。相邻 PE 之间通过本地共享存储器进行数据通信,不相邻的 PE 可以通过路由器进行数据通信。

PAAG 中的行 (列) 控制器可同时向属于同一行 (列) 的 PE 发送同一条指令来实现单指令多数据操作,而多指令多数据操作在 PAAG 中则体现为行 (列) 控制器向所属行 (列) 的 PE 发送不同的指令。在 PE 内部,同一时刻可包含八个线程,并且这八个线程或独立或协作工作。通过结合 PAAG 的数据级并行、操作级并行以及线程级并行等多种计算模式来充分挖掘程序中的并行性^[7]。

2 统一渲染架构下的图形流水线

结合开放图形库 (Open Graphics Library, OpenGL) 图形流水线^[8]和统一渲染架构思想,同时加入 OpenGL 缓冲区对象^[9]机制,文中提出了统一渲染架构下的图形渲染流水线,见图 1。

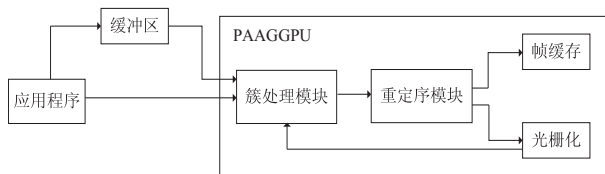


图 1 PAAG 上实现统一渲染架构图形渲染流水线

数据缓冲区 (Buffer Object, BO) 作为 GPU 顶点数据的来源,PAAG 的一个簇用作流处理器阵列,每一个 PE 对应一个流处理器,同时为实现统一渲染的目标,在这个簇的内部设计了顶点缓存 (Vertex Cache, VC)、像素缓存 (Pixel Cache, PC) 和分配器 (Dispatcher)。经过 PE 处理后的数据写入重定序模块 (Reorder Buffer, RB),由 RB 根据数据类别决定数据的去向。

3 统一渲染架构在 PAAG 上的具体实现

3.1 Buffer Object 的设计

BO 是一个独立于 CPU 和 GPU 的存储区域^[9],由于它向 GPU 传输数据的方式为直接内存访问 (Direct Memory Access, DMA),所以它的出现代替了 CPU 向 GPU 传输数据这种费时的过程。文中对来自 CPU 端应用程序的顶点信息进行提取和整合,然后将其存储在 BO 中,每一个顶点数据包含有顶点的四类属性共 16 个标量 (坐标、颜色、纹理、法向量)。对每个需要生成的图形,其顶点数据存入 BO, CPU 端只保留其顶点索引数组。

3.2 Vertex Cache 和 Pixel Cache 的设计

在 PAAG 簇的全局共享存储器里设置 VC 和 PC,分别存储来自 BO 的顶点数据和来自光栅化模块的像素数据。这样设置的原因有两个:

(1) 现代图形处理器中通常都设有 VC^[10],以减轻顶点读取时的带宽要求。

(2) PC 作为 PE 像素数据的来源,不仅可以存储光栅化模块产生的大量像素数据,而且和 VC 一起配合后面所提到的 Dispatcher 对 PE 进行数据的分发。在分发的过程中 Dispatcher 对 PC 和 VC 一视同仁,使得每个 PE 不仅有可能得到顶点数据,也可能得到像素数据,以此达到统一渲染的目的。

受硬件影响,VC 的容量不可能无限大,故文中在进行顶点传输时进行了优化,以此来减少顶点复用^[11]导致的重复传输浪费。如图 2 所示,经过传输,在 VC 中的顶点数据顺序不同于在 BO 中,重新排列的顶点数据意味着 VC 的命中率提高,进而提高了渲染速度。

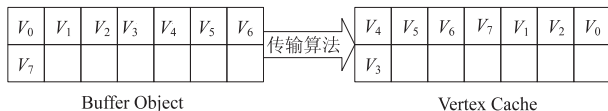


图 2 Buffer Object 到 Vertex Cache 的数据流通示意图

顶点传输优化算法的伪代码如下:

```
char Buffer[ MAX ]; /* 缓冲区 */
char Vertex_cache[ size ]; /* 顶点缓存 */
/* int Indices[ len ]; 顶点索引 */
/* int Flag[ num ]; 标记此顶点是否已经存在于缓存中 */
void Read_from_buffer( int Indices[ ], int Flag[ ] )
{
    int j=0;
    int current_indices=0;
    for( int i=0; i<len; i++ )
    {
        current_indices=Indices[ i ];
        /* 如果此顶点已经存在于缓存中,那么循环继续 */
        if( Flag[ current_indices ] != 0 )
            continue;
```

```
/* 如果此顶点不存在于缓存中,那么就将此顶点数据从缓冲区复制到缓存中 */
```

```
else
Copy_data(Buffer[current_indices], Cache[j++]);
Flag[current_indices] += 1;
}
```

3.3 分配器的设计

3.3.1 地址存储

Dispatcher 包含四个存储地址的队列,分别存储当前 PC 和 VC 里所有数据的地址以及一个标记此数据属于哪个 Cache 的标志位(下面统称为地址),存储序列如图 3 所示(图中 Address Queue 中的 p_0 表示地址而不是数据)。若 PC 为空,PE 会优先存储顶点地址;若 PC 不为空,PE 会优先存储像素地址,在存储完像素地址后转而存储顶点地址。

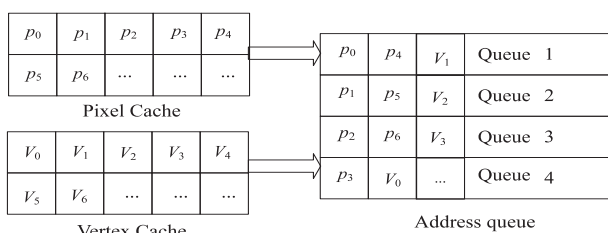


图 3 地址存储序列示意图

3.3.2 分发地址

存储地址完成后,Dispatcher 要分发地址到各组 PE(文中对 PE 按列分组)。四个地址队列分别对应四组 PE。由于四个队列发放过程是相互独立的,因此提高了数据并行性。

3.3.3 收集信号

分发地址之前,Dispatcher 通过检测列控制器中的信号 signal 来判断本组 PE 是否可以分发。当 signal 为 0 时,本组 PE 可以分发;当 signal 大于 0 时,本组 PE 不可以分发。每分发一次地址,Dispatcher 将 signal 的值加 1,所以当本组 PE 全部获得地址,对应的列控制器中的 signal 应变为 4。

3.4 流处理器阵列中 PE 的设计

PE 作为流处理器的原型,其内部程序可由程序员写入。依据分配器分发的地址,PE 从 Cache 中获得数据并进行处理。PE 不仅可以进行通用计算,而且 PE 的内部会根据附着在地址中的标志位识别出此数据来自哪一个 Cache,所以不同的数据在 PE 中会得到相应的处理,处理后的结果连同标志位写入 RB,同时将所在组的 signal 减 1。所以,当每组 PE 把结果都写入到 RB 后,对应组的 signal 应该变为 0,这也意味着本组 PE 可以进行下一次的地址分发了。

3.5 重定序模块设计

RB 包含一块存储区域,放置在 PAAG 簇全局共享

存储中,可以存放 16 个顶点(编号 0 ~ 15,对应 16 个 PE),同时包含状态变量 State 来指示此区域是否已经写满数据。因为 PE 产生结果的顺序有先后之分和类别之分,所以此区域可以用来对 PE 产生的结果进行同步和分类^[12]。当通过 State 判断出 RB 满时就会触发对该存储区域的刷新,如果是顶点数据,进入下一级图元装配模块(Primitive Assemble),如果是像素数据,写入帧缓冲区(Framebuffer),刷新完毕后重置 State 变量。考虑到 PE 内部进行通用计算所用的时钟数不会相差太多,故这里等候 RB 存满的时间不会对整体性能造成很大影响。

3.6 统一渲染整体架构

统一渲染的整体架构图如图 4 所示。

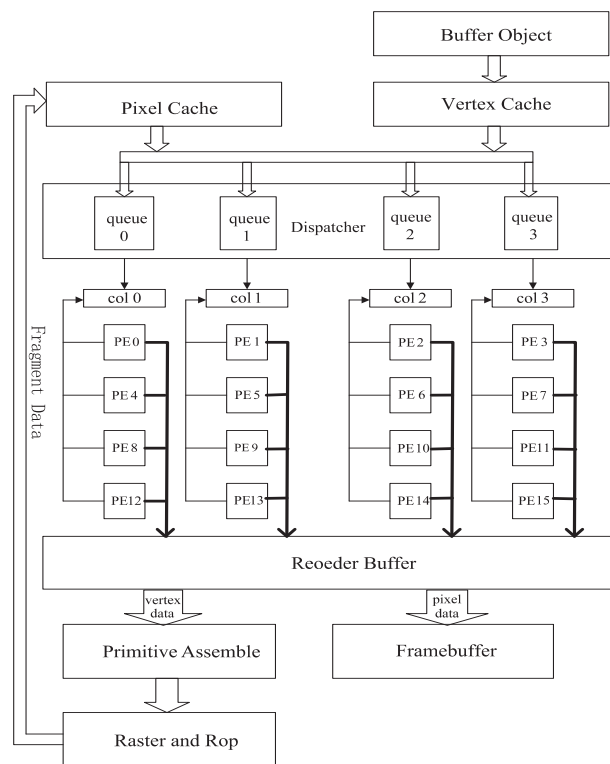


图 4 统一渲染架构在 PAAG 上实现的整体架构图

4 实验结果验证及分析

(1) 针对基本 3D 模型的顶点数据,计算了 VC 的平均缓存失效率(Average Cache Miss Ratio, AC-MR)^[13-14],见表 1。ACMR 越低,表明 Cache 的命中率越高。实验所设置的 VC 的大小为可以存放 16 个顶点。

(2) 针对基本 3D 模型,通过统计程序在运行时顶点处理所需时钟数(Clock_V)、像素处理所需时钟数(Clock_P)以及整体渲染所需时钟数(Clock_T)进行性能分析。用 Clock_O 表示其他模块所需要的时钟数,于是有:

$$\text{Clock}_V + \text{Clock}_P + \text{Clock}_O = \text{Clock}_T$$

当等式成立,表明 PE 实现满载,负载达到基本平衡。在这里忽略了 Clock_O,所以只要 Clock_V 和 Clock_P 之和接近于 Clock_T 即可,见表 2。

表 1 经过顶点传输算法优化后 VC 的 ACMR 值

3D model	需要渲染的 三角形个数	实际传输 顶点次数	ACMR
八面体	8	5	0
立方体	12	8	0
鼓型	24	14	0
半球	448	526	0.643
环	576	705	0.689
球体	960	1 115	0.668
茶壶	1 024	991	0.536
兔子	69 451	156 596	1.268

表 2 顶点处理和像素处理时钟个数统计表

3D model	Clock_V	Clock_P	Clock_T
八面体	20	76 800	76 864
立方体	32	23 504	23 552
鼓型	56	99 524	99 584
半球	2 104	91 568	93 696
环(缩小 20 倍)	2 820	59 760	62 592
球体(缩小 20 倍)	4 460	135 188	139 648
茶壶(缩小 20 倍)	4 064	59 760	63 744
兔子	626 384	10 200	636 608

5 结束语

结合 PAAG 丰富灵活的并行计算模式和统一渲染思想,文中设计了一种对顶点和像素能统一高效着色的架构。实验数据说明了该架构对渲染性能有一定提升,并且能解决 GPU 负载不均衡的问题。如何在该架构上进行优化和扩展,将在未来工作中继续研究。

参考文献:

[1] 田绪红,陈茂资,田金梅. DirectX 发展及相关 GPU 通用计算技术综述[J]. 计算机工程与设计,2009,30(23):5432-5436.

[2] 韩俊刚,刘有耀,张 晓. 图形处理器的历史现状和发展趋势[J]. 西安邮电学院学报,2011,16(3):61-64.

[3] 林一松,唐玉华,唐 滔. GPGPU 技术研究与发展[J]. 计算机工程与科学,2011,33(10):85-92.

[4] VISA. 大一统的时代——统一渲染架构引领未来[J]. 电脑迷,2007(11):32-33.

[5] 戴长江,张尤赛. 基于图形处理器的通用计算技术的研究[J]. 现代电子技术,2013,36(4):157-161.

[6] 黄虎才. 多态阵列处理器的并行计算研究[D]. 西安:西安邮电大学,2014.

[7] 杨 柳,刘铁英. GPU 架构下的并行计算[J]. 吉林大学学报:信息科学版,2012,30(6):629-632.

[8] 王兰美,赵继成,秦华东. OpenGL 及其在 VC++ 下的开发应用[J]. 武汉大学学报:工学版,2006,39(4):62-65.

[9] Kilgard M J. Modern OpenGL usage: using vertex buffer objects well[C]//Proc of ACM SIGGRAPH Asia Courses. [s. l.]:[s. n.],2008.

[10] Yu C H, Chung K, Kim D, et al. An energy-efficient mobile vertex processor with multithread expanded VLIW architecture and vertex caches[J]. IEEE Journal of Solid-State Circuits, 2007,42(10):2257-2269.

[11] Barczak J D, Nehab D F, Sander P V. Fast triangle reordering for vertex locality and reduced overdraw: US, US8379019B2 [P]. 2013.

[12] 韩俊刚,姚 静,李 涛,等. 多态并行机上的 3D 图形渲染[J]. 西安邮电大学学报,2015,20(2):1-6.

[13] 陈思远,史广顺,王庆人. 通过三角形 Strip 衍生实现三维模型数据的渲染优化[J]. 计算机辅助设计与图形学学报, 2009,21(8):1155-1163.

[14] Lin G, Yu P Y. An improved vertex caching scheme for 3D mesh rendering[J]. IEEE Transactions on Visualization & Computer Graphics,2006,12(4):640-648.

(上接第 38 页)

[10] 崔平远,郑黎方,裴福俊,等. 基于卡尔曼/粒子组合滤波器的组合导航方法研究[J]. 系统仿真学报,2009,21(1):220-223.

[11] Kalman R E. A new approach to linear filtering and prediction problems[J]. Transaction of the ASME-Journal of Basic Engineering,1960,82:35-45.

[12] Saber R O, Shamma J S. Consensus filters for sensor networks and distributed sensor fusion[C]//Proc of IEEE conference

on decision and control. Seville: IEEE Press, 2005: 6698-6703.

[13] Ren W, Beard R W, Kingston D B. Multi-agent Kalman consensus with relative uncertainty[C]//Proc of American control conference. Oregon: IEEE Press, 2005: 1865-1870.

[14] Marco C, Giorgio F, Riccardo G, et al. Real-time data fusion and MEMS sensors fault detection in an aircraft emergency attitude unit based on Kalman filtering[J]. IEEE Sensors Journal, 2012, 12(10): 2984-2992.