图形处理器流水线数据压缩技术研究综述

韩立敏, 田 泽, 张 骏, 郑新建, 任向隆
(中航工业西安航空计算技术研究所, 西安 710068)

摘 要: 提高功耗效率是高端 GPU 的关键设计目标之一。在 3D 图形渲染流水线的多个阶段,使用数据压缩技术能够显著减少 GPU 片外存储器的访问量,从而达到提高图形绘制性能和降低功耗的效果。为了对图形处理器流水线数据压缩技术的应用现状进行总结和分析,立足于 GPU 图形渲染流水线和存储系统的结构特征,归纳了各种缓冲区对象、纹理数据专用压缩算法的关键特性;分析了图形流水线数据压缩技术的研究现状、不足与挑战;并基于应用需求指明 GPU 流水线数据压缩技术进一步的研究内容。

关键词: 图形处理器; 数据压缩; 3D 渲染流水线; 功耗效率
中图分类号: TP303 **文献标志码**: A **文章编号**: 1001-3695(2018)03-0648-06
doi:10.3969/j.issn.1001-3695.2018.03.002

Survey of data compression techniques for GPU pipeline

Han Limin, Tian Ze, Zhang Jun, Zheng Xinjian, Ren Xianglong
(*Xi'an Aeronautics Computing Technique Research Institute of China Aeronautical Industry*, *Xi'an* 710068, *China*)

Abstract: One of the most important design aspects for high-end GPU is improving its power efficiency. Data compression can be used in various stages of the GPU 3D rendering pipeline to reduce off-chip memory traffic, yield high performance and low power consumption. In order to summarize the research progress and application status of data compression techniques for GPU pipeline, this paper outlined the key features of data compression specialized for buffer objects and texture based on the structure characteristics of GPU 3D rendering pipeline and memory system. Moreover, it discussed research status, inefficiencies and challenges of data compression in GPU pipeline. In the end, it indicated the further research content for GPU data compression according to application demands.

Key words: graphics processing unit(GPU); data compression; 3D graphics rendering pipeline; power efficiency

0 引言

人类对图形效果真实性近乎无止境的需求使图形处理器(graphics processing unit,GPU)成为各种嵌入式系统、PC、工作站、个人数字媒体产品(如智能手机、游戏机和平板电脑)不可缺少的核心部件。相比 PC 系统,个人数字媒体产品的可用性受限于电池的续航能力,对 CPU 和 GPU 等核心芯片的面积和功耗要求更为苛刻,GPU 需要以更低的功耗、更有限的存储容量和内存带宽实现高性能、高质量的图形绘制功能。尽管 NVIDIA、AMD 等主流商用 GPU 厂商使用 HBM(high bandwidth memory)或 GDDR(graphics DDR)技术提高了 GPU 的片外存储器带宽,相比持续增长的存储访问需求,存储带宽依然是 GPU 的稀缺资源。

绘制真实感的 3D 游戏和电子地图需要在 GPU 的片外图形存储器(off-chip graphics memory)存放海量的顶点属性、纹理、深度和颜色数据,GPU 3D 图形渲染流水线的多个处理阶段需要并发访问片外图形存储器获取这些绘制相关的数据,存储访问请求的种类多样,并且数目庞大:顶点着色阶段(vertex shading)需要读取每个顶点的多个属性数据;像素着色阶段(pixel shading)需要为每个像素读取多个纹素采样值;光栅化阶段(rasterization)需要设置深度缓冲区的值;光栅操作阶段(raster operation,ROP)需要读取并修改每个像素的深度数据和颜色数据。据统计,GPU 访问片外存储器耗费的功耗接近芯片总功耗的 60% ~70%[1,2]。

为提高 GPU 功耗效率(power efficiency),设计高性能低功耗的存储系统已经成为 3D 图形流水线设计领域的研究热点[3~7]。学术界和商业界已经达成共识:在 GPU 硬件流水线结构中集成压缩/解压缩逻辑、传输和存储压缩形式的图形绘制数据,能显著减少存储器访问请求的个数,降低数据传输量,提高片上图形绘制专用 cache(顶点、纹理、深度和颜色)的利用率。例如,NVIDIA Tegra 4 移动图形处理器核的图形渲染流水线集成了专用存储访问管理系统,实现 16∶1 的无损深度(depth)压缩、4∶1 的无损颜色和 4∶1 的纹理[8]。NVIDIA Tegra X1 的 Maxwell 系列 GPU 核对颜色数据(纹理、颜色缓冲区和帧缓冲区)应用数据压缩技术,相比未使用数据压缩技术的 GPU 系统,平均节省了 60% 左右的存储器带宽[9]。

收稿日期: 2017-02-14; **修回日期**: 2017-03-27

作者简介: 韩立敏(1983-),女,陕西宝鸡人,工程师,博士,主要研究方向为计算机体系结构、GPU 设计、微处理器设计、SoC 设计(hanlm20130808@163.com);田泽(1965-),男,陕西宝鸡人,研究员,博士,主要研究方向为 SoC 设计方法学、航空专用集成电路设计;张骏(1978-),男,陕西西安人,高级工程师,博士,主要研究方向为计算机体系结构、GPU 设计、微处理器设计、SoC 设计;郑新建(1980-),男,山东淄博人,工程师,博士,主要研究方向为计算机体系结构、GPU 设计、微处理器设计、SoC 设计;任向隆(1982-),男,河南洛阳人,高级工程师,博士,主要研究方向为计算机体系结构、GPU 设计、微处理器设计、SoC 设计.

1　3D 图形流水线及其存储系统

图 1 主要描述 3D 图形渲染流水线的逻辑操作流程和存储系统结构,与具体的 GPU 硬件结构是分离可编程着色结构还是统一着色架构无关。GPU 的存储系统包含片外图形存储器和片上缓存系统(on-chip cache system)。GPU 渲染流水线中的片上缓存系统包括存储光照和视窗变换操作之前的顶点属性数据 vertex cache1、存储光照操作和视窗变换操作之后的顶点属性数据 vertex cache2、texture cache、depth cache 以及 color cache。

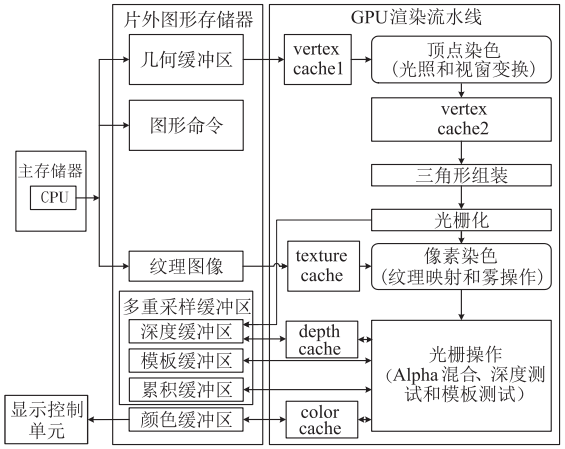

图1　3D渲染流水线及其存储系统

如图 1 所示,3D 图形渲染流水线的顶点着色、纹理映射、光栅操作(rasterization operation,ROP)阶段分别需要访问 GPU 片外的几何缓冲区、纹理存储区域、多重采样缓冲区(模板缓冲区、深度缓冲区和累积缓冲区)、颜色缓冲区。顶点着色阶段对几何(geometry)绘制对象的顶点位置和属性数据执行读操作;像素着色阶段对纹理图像(texture)执行读操作;光栅化阶段对像素执行 Alpha 混合操作,对颜色缓冲区(color buffer)执行写操作和读—修改—写,对深度缓冲区(depth/Z buffer)以及模板缓冲区(stencil buffer)执行读和写操作;在显示输出阶段,显示控制单元(display controller,DC)需要读取颜色缓冲区的数据。除了数据压缩率,为了保证实时绘制性能,GPU 芯片设计人员更加关注压缩操作和解压缩操作的复杂度和实现效率。为了满足实时绘制性能,图形数据的压缩算法一般具备三个特性:a)支持数据的随机存储器访问;b)易于硬件实现,压缩/解压缩操作延迟小,否则会为图形系统引入新的性能瓶颈;c)图形渲染流水线对缓冲区操作包含着对 cache 的访问,压缩算法兼顾 cache 行的尺寸,压缩后的数据块对 cache 硬件设计友好。例如压缩编码刚好与 cache 行的大小相同,引入压缩/解压缩逻辑不改变原有的 cache 结构。

表 1 从压缩/解压缩操作是否要求实时性(real-time/online)、是否为无损(lossless)压缩两个角度总结了各种不同缓冲区压缩算法的特征。

表 1　不同压缩对象压缩算法的比较

数据类型	实时性	无损
几何缓冲区	解压缩	是
深度缓冲区	压缩/解压缩	是
纹理图像	解压缩	否
颜色缓冲区	压缩/解压缩	是

据表 1 所示,GPU 硬件流水线中不同压缩对象的压缩算法具有一定的相似性,但也有不同之处。下文结合各种压缩对象的产生时机、数据特征和读取访问模式分析压缩算法的特点。

a)操作的实时性。深度和颜色缓冲区压缩和解压缩操作是对称的操作,深度和颜色是可读可写的数据,在图形渲染阶段动态产生、被频繁更新,实时压缩和解压缩缓冲区的数据才能满足图形绘制的实时性。相比而言,纹理数据和几何数据的压缩和解压缩算法是非对称的,在主机端以离线方式花费较长的时间对纹理和几何数据进行压缩可以获取较高的压缩率,对压缩操作的性能要求较低,但解压缩算法必须足够简单,以便实现实时解压缩。

b)有损(lossy)与无损(lossless)压缩。纹理数据是只读数据,在整个绘制过程中仅需执行一次压缩操作。一般情况下,压缩纹理图像引入的误差不会对最终的绘图质量产生重大影响,因此纹理压缩通常都是有损压缩。然而对某个缓冲区的数据执行多少次压缩和解压缩操作与在该区域绘制多少个三角形直接相关。对缓冲区数据多次执行压缩和解压缩会将压缩引入错误放大,缓冲区的数据被频繁修改使缓冲区的数据压缩操作对于数据信息损失非常敏感,容易导致绘图错误,因此在深度和颜色缓冲区一般使用无损压缩算法,或者在控制压缩操作所引入的错误上限的情况下使用有损压缩算法。几何缓冲区的数据包含了绘制图形的位置信息,损失精度对于图形的绘制效果不可容忍,一般要求几何缓冲区也实现无损压缩。

在 GPU 出现之初,GPU 流水线的数据压缩技术已经得到了广泛的研究。下文结合缓冲区类型,分类描述几何缓冲区、纹理图像存储区域、深度缓冲区/模板缓冲区和颜色缓冲区压缩技术的特征和研究现状。

2　3D 图形流水线的数据压缩

2.1　几何缓冲区压缩

在顶点着色阶段,渲染三维网格(3D mesh)需要使用大量的三角形顶点。三角形的顶点信息存储在几何缓冲区,顶点(vertex)的数据不仅包含位置信息(position),还包括三角形网格的顶点连接关系(connectivity)信息和属性(attribute)。顶点连接关系指示哪些顶点组成三维网格的面(triangular mesh face)。属性包括顶点和面的颜色、法向量、多组纹理坐标等信息。很多 3D 游戏程序的几何数据仅需要声明一次,被 GPU 流水线多次读取。为了保证图形绘制的实时性,要求几何缓冲区压缩算法必须满足[10]:a)解压缩过程足够简单;b)对顶点位置的解压缩操作能够并行执行,不能相互依赖;c)压缩编码是固定尺寸,与 GPU 硬件数据通路的宽度相互匹配。

十年前,Alliez 等人[11]总结了三维网格压缩算法特征,一些算法压缩率较高,但是压缩操作和解压缩操作相当复杂,面积开销大;另外一些算法无法对压缩数据实现并行解压缩操作,导致这些算法无法应用于嵌入式移动设备。2005 年,Purnomo 等人[12]提出一种几何缓冲区的实时解压缩算法。为了使图像空间的误差最小化,为顶点属性的不同数据(位置、法向量、纹理坐标)分配不同长度的字段,顶点数据被压缩为 96 bit,以压缩形式送给顶点着色器,并在渲染阶段实现实时解压缩操作。2007 年,为了提高 vertex cache1 的空间利用率,Chhugani 等人[13]通过转换几何网格拓扑的表达形式,对网格的拓扑关系进行压缩,硬件实现实时解压缩。2009 年, Meng 等

人[14]使用基于 K-Ring 预测策略的 VQ(矢量量化)算法对三维三角形网格(3D triangle mesh)的顶点数据进行压缩。在此基础上,2010 年, Lee 等人[10]基于局部化的矢量量化策略对网格划分(mesh partition)后的网格顶点位置信息进行压缩,由于多个网格所共享的顶点信息在每个子网格中都存在副本,压缩编码所包含的矢量量化后的顶点位置信息的个数比实际顶点的个数多,使得该压缩算法的压缩率较低。为了提高几何数据的解压缩速率,德国多个研究结构联合 NVIDIA 等企业研究面向实时图形绘制的几何数据解压缩算法[15,16]。

NVIDIA 的 GPU 使用细分曲面等高阶曲面(higher order surface)和替换映射技术(displacement map),对几何数据实现了压缩,显著降低了几何着色阶段存储访问的带宽需求。应用高层次表面技术 GPU 的命令处理器能根据曲面细分(tessellation)技术使用少量控制顶点(controlled point)自动产生成千上万个多边形的几何数据(顶点)。GF100 架构 GPU 在顶点着色阶段使用置换映射技术改变 3D 模型顶点的相对位置,使用小型的替换映射文件表示复杂几何模型[17]。

为了减少顶点阶段原始输入数据量,研究人员分别针对顶点和三角形网格位置信息、法向量、纹理坐标和三角形顶点之间连接关系的一个方面或多个方面的几何缓冲区压缩算法进行研究,研究历史将近 30 年之久,近几年几何压缩算法的研究热点集中在降低解压缩操作的计算复杂度和操作时延方面。然而鉴于该技术目前仍然无法将其有效集成到 GPU 流水线中,现代商用 GPU 中在顶点阶段转而使用(displacement、bump、normal)纹理贴图、曲面细分等渲染技术大幅度减少 3D 模型顶点输入数据到外部存储器中顶点缓冲区的 IO 量,达到对几何数据进行压缩的目的。

2.2 纹理压缩

3D 游戏、电子地图和导航系统依赖 1D、2D、3D、cube(环境映射技术)和 normal(法向量映射技术)等纹理贴图技术实现高质量真实感的视觉效果。应用大量高质量(大分辨率)纹理图像需要占用 GPU 大量存储器空间,纹素采样处于像素着色阶段,每个像素需要获取多个纹素采样值,执行纹理贴图操作成为耗费 GPU 外部存储器容量、存储器带宽和功耗主要因素,对纹理实施压缩成为 GPU 提高性能和降低功耗的必然选择。

常见的纹理压缩算法一般针对 2D 纹理而设置,但是可以用于所有类型的纹理。各种常见纹理压缩格式在商用 GPU 中的应用情况如表 2 所示。其中 BC6H/BC7 用于 PC 机 GPU, S3TC/DXT^n[18]这种轻量级的格式主要用于嵌入式 GPU。目前广泛使用的纹理压缩格式大多数都沿袭 Delp 等人[19]基于块、固定压缩率的压缩模式。基于块的主流纹理压缩格式包括 S3TC,以及爱立信的 ETC1[20]和 ETC2[21]、PVRTC[22,23]。表 2 中的带宽收益指的是应用纹理压缩算法所节约的存储器带宽量,功耗收益指的是应用纹理压缩算法所减少的系统功耗。

表 2 压缩纹理的应用现状

算法名称	GPU 厂商	压缩率	带宽收益/%	功耗收益/%
S3TC/ DXT^n	AMD	8:1 ~ 4:1	61.2	59.4
PVRTC	Imagination	16:1 ~ 4:1	73.5	69.2
ETC1/ETC2	ARM	8:1 ~ 4:1	66.1	64.9
BC6H/BC7	NVIDIA	4:1	47.4	52.2
3Dc +	AMD	4:1 ~ 2:1	44.5	47.9
ASTC	ARM	32:1 ~ 4:1	83.1	60.9

图 2 是 DXT1 的压缩格式。对于每个 4 × 4 的像素区域(pixel tile),首先,通过比较操作选择出两个具有代表性的 16 bit 颜色值;然后,为 4 × 4 的 pixel tile 中的每个 pixel 设置 2 bit 的颜色索引值;最后,形成 64 bit 的压缩编码。

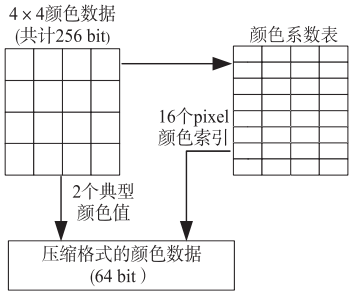

图2 DXT1压缩编码

法线贴图(normal map)捕捉光线的反射光,模拟真实表面的高光效果,使 3D 物体形成复杂的外观效果。NVIDIA 曾经尝试使用 DXT1 对法线纹理实现实时压缩和解压缩操作[24]。实验表明,DXT1 的块失真(block artifact)在一些情况图形绘制效果误差较大,容易出现失真。ATI(AMD)的 3Dc 和 3Dc + 专门用于压缩法线纹理[25],是一种基于块的压缩纹理格式,与 DirectX 10 的 BC5 以及 OpenGL 的 RGTC 格式类似,被应用于 AMD 的多款 GPU[26]。

为了满足因特网上传输高质量纹理的高带宽需求,使 PC 版本的 3D 游戏更为真实、细腻,2015 年,微软计划在新一代 DirectX 中采用 NVIDIA 开发的体纹理压缩格式(volume texture compression format, VTCF)对 3D 纹理数据进行压缩。2016 年, Guthe 等人[27]提出一种可以并行执行的无损压缩算法,为直接体渲染(direct volume rendering)提供压缩形式的体渲染数据,支持随机访问和实时解压缩操作。

由于不同应用环境所需要的纹理图像颜色格式不同、颜色格式的分量个数不同(RGB/RGBA)、动态范围不同(LDR/HDR)、纹理图像的维度(2D/3D)不同、颜色数据的精度范围(2 bit/4 bit/8 bit/16 bit)不同,为了最大限度覆盖纹理压缩算法的应用范围,2012 年,ARM 提出了一种灵活的纹理压缩格式 ASTC(adaptive scalable texture compression)[28],允许对纹理压缩器的配置参数进行设置,以便产生满足指定压缩率和图像质量的压缩纹理格式。

纹理压缩技术已经广泛应用在现代 GPU 中,平均降低了 60% 的存储器带宽使用量和 60% 的系统功耗,效果显著。随着用户对图形视觉质量的要求越来越高,由于经典纹理压缩技术本质上都是有损压缩,解压缩操作不同程度丢失纹理数据细节,对解压缩后的纹理图像质量造成损害,使用有损压缩纹理数据对 3D 对象的表面进行贴图造成最终绘图视觉失真较大,基于二维矢量图形绘制技术和挖掘纹理图像内部细节信息的分布特性构建自适应高视觉质量纹理压缩技术迫在眉睫。

2.3 深度/模板缓冲区压缩

深度缓冲区是解决光栅化流水线中绘制对象可见性的标准技术。为了实现 3D 图形的空间深度层次感,ROP 单元每绘制一帧图像都需要读取片外显存(GPU memory)的深度缓冲区,通过 Z-buffer 测试算法确定图元间的深度关系,对每个像素都需要执行深度测试操作,所以深度缓冲区的存储访问频率非常高。

位于同一个基本图元中的像素点无论在位置信息还是在深度信息上以某种形式存在空间上的一致性关系,这为深度数

据实现压缩提供了可能性。AMD 和 NVIDIA 等商用 GPU 流水线普遍集成了无损 Z-compression(深度压缩)技术[17,26]。Hasselgren[29]分析和评估了专利中所见到的多种深度压缩算法,包括快速深度值清除(fast Z-clears)、DDPCM(differentical differentical pulse code modulation)、锚编码(anchor encoding)、平面编码(plane encoding)、深度偏移(depth-offset),并基于此提出图 3 所示的深度压缩架构,光栅化器(rasterizer)更新非压缩形式的 tile(8×8 的 pixel 块),然后对于 tile 实现无损压缩,将压缩形式的深度数据写入到 depth buffer(深度缓冲区),最后更新 tile table 中的 Z-min(最小深度值)和 Z-max(最大深度值)。

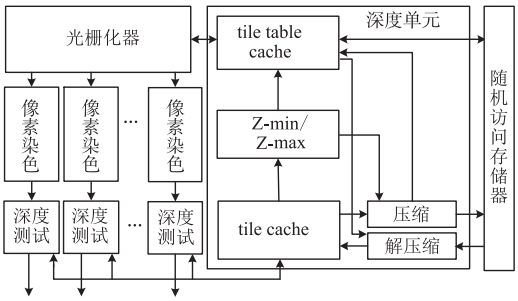

图3　深度缓冲区的压缩框架

DDPCM 基于屏幕空间中深度值的插值操作计算二重深度差分[30]。图 4 描述 DDPCM 计算深度值二重差分流程。其中图 4(a)为 tile(4×4 的像素区域)的原始深度值,(b)为一重列差分,(c)为二重列差分,(d)为二重行差分。首先按列计算 tile 像素间的一重列深度差分;然后按照同样的方式计算 tile 列像素间的二重深度差分;最后对 tile 前两行以相同方式计算行的二重差分。

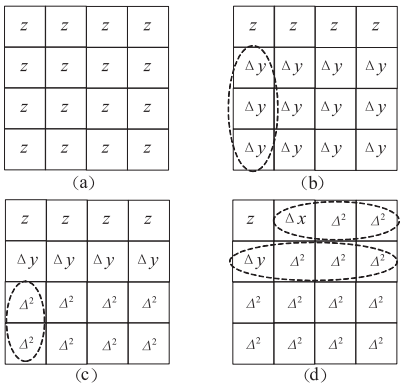

图4　计算深度值的二重差分

Anchor 编码[31]原理与 DDPCM 类似,为每个像素增加额外数据量的同时,并没有提高压缩率。两平面模式(two plane mode)最大的缺点是两个参考点必须来自不同的平面,50% 的情况下这种假设是正确的,其余 50% 的情况假设失效[32]。针对运动模糊问题,2013 年,Intel 的 Andersson 等人[33]简化了深度函数的表达方式,提出一种压缩率更高的通用平面压缩方法(generalized plane encoding,GPE)。

ROP 单元除了需要读写 depth 缓冲区外,还需要读写 stencil 缓冲区[34]。由于 depth 和 stencil 都是向量数据,使用同一套压缩算法对 depth 和 stencil 进行压缩和解压缩有利于简化硬件设计,在 AMD 的多款 GPU 中通常使用同一套压缩解压缩器对 depth 和 stencil 进行压缩和解压缩[26,35]。即便如此,NVIDIA 专门研究了针对 stencil 缓冲区的压缩算法,申请了 stencil 缓冲区压缩算法的相关专利[36]。

受限于存储总线数据宽度和压缩算法本身,深度缓冲区数据的压缩率一般能达到 4∶1 或者 2∶1。在 GPU 流水线中协同设计存储压缩形式的深度值 depth cache 和深度压缩解压缩逻辑,能够大幅度降低深度缓冲区的存储器访问带宽需求。为了进一步提高图形绘制性能,需要研究更高压缩率、适合硬件实现,在确保深度数据精度不损失的同时能够进一步降低压缩和解压缩操作复杂度的深度压缩算法。

2.4　颜色缓冲区压缩

帧缓冲区(frame buffer)包含颜色缓冲区(color buffer)和外视频输入的图像存储区域。帧缓冲区每一帧的数据被需要 GPU 之外的 IP 模块频繁读取,对帧缓冲区实施压缩能够减少 SOC 内不同 IP 之间的数据传输量,达到降低系统级的存储器带宽需求和功耗的目的。

对于颜色缓冲区,ROP 阶段执行多重采样和反走样操作阶段需要读取和修改颜色缓冲区的数据,一个像素的 N 个采样值共享同一个颜色值,使用一个颜色代表 N 个颜色,剔除冗余信息,可以实现对颜色数据的压缩[6]。为了保证绘图的正确性,ROP(光栅操作)单元的 Alpha 混合操作要求颜色缓冲区必须实现无损压缩。Hasselgren[37]提出一种两级无损颜色压缩算法,并且实时无损压缩和解压缩操作。如图 5 所示,首先,基于颜色空间转换操作可逆的计算公式对 8×8 的像素 tile 中的每个像素执行格式转换,使用亮度—色度颜色空间替代 RGBA 颜色空间的数值实现颜色数据的无损压缩;然后,使用预测器对颜色数据 X 进行预处理,得到 X';最后,对处理后的颜色值 E 实施 Golomb-Rice 压缩,形成压缩码流。Kim 等人[32]使用 DDPCM 产生差分(difference),最后使用 Golomb-Rice 算法对差分得到最终的编码。

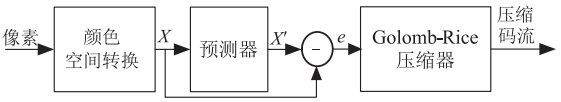

图5　颜色缓冲区压缩算法

当 GPU 将帧缓冲区中的输入视频数据作为 3D 场景的纹理数据,或者帧缓冲区的数据被 DC 读取,执行视频后处理操作和刷新当前的显示内容。在多个 IP 核之间应用压缩技术能够减少多个 IP 核(GPU、CPU 和 DC)之间的数据传输量。Rasmusso 等人[38]提出一种针对帧缓冲区(颜色数据)的无损压缩和解压缩方法。在商用 GPU 中,ARM 的 Mali-T880 使用基于 4×4像素块的无损、实时、支持细粒度随机访问的压缩和解压缩的帧缓冲区压缩算法(ARM frame buffer compression,AFBC)[39]。类似地,AMD RS780E GPU 采用了一种自适应的帧缓冲区压缩技术[40]。

在空间维度,一些帧缓冲区压缩技术利用每一帧内部的局部信息实现数据压缩,能减少平均 50% 的系统存储器带宽。在时间维度,连续帧在视觉上具有较大程度的相似性,相邻帧相同位置的数据大部分是相同的,剔除对同一个区域相同数据的写操作或者利用相似性对下一帧实施压缩,能够达到减少存储带宽的目的。目前,帧缓冲区压缩算法主要针对空间维度的局部性进行压缩,挖掘连续帧的时间领域局部性的研究相对较少。联合使用空间维度和空间维度的帧缓冲区带宽节省技术成为必然趋势。

3 新兴 GPU 流水线数据压缩技术

为了进一步降低压缩和解压缩逻辑的设计复杂度,提高算法的灵活性,新兴出现的 GPU 流水线数据压缩算法主要包括以下两类:

a)为 GPU 浮点数据缓冲区设计统一架构的压缩和解压缩算法。

压缩和解压缩逻辑一般位于存储控制器单元中,设计一种适用于多种图形绘制数据的通用压缩/解压缩硬件逻辑,有助于降低硬件设计的复杂度和提高片上网络及片外总线的功耗效率。针对 16 bit 浮点格式的深度和颜色缓冲区数据,Jacob 等人[41]提出一种统一架构的压缩解压缩算法,同一套压缩解压缩单元逻辑既可用于颜色和纹理数据,也可用于深度数据。如图 6 所示,为了缩短解压缩操作的时间,Chien 等人[42]采用一种基于无损颜色格式转换、空间预测和比特流重分配策略,减少不同采样之间的数据依赖性,使用 DXT5 解压缩硬件逻辑对颜色、深度以及纹理进行解压缩,在该框架还包含颜色、深度以及纹理共用的 cache 和 tile 表。针对浮点数据,包括几何数据(如顶点位置、法向量和纹理坐标)、颜色、深度缓冲区以及通用可变精度的浮点数据,Pool 等人[43]设计了一种能够处理负值的 32 bit 无损压缩和解压缩器。

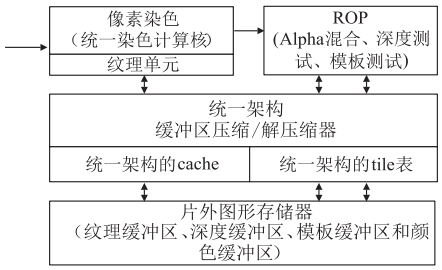

图6 通用缓冲区的压缩框架

纹理、帧缓冲区的数据本质上都是颜色数据(如 RGBA),使用同一种压缩算法对多种颜色数据实现压缩成为一种必然选择。工业界已有类似的尝试,2014 年 AMD 发布的第三代 GCN(graphics core next)架构的 Radeo R9 FURY、2014 年 NVIDIA 发布的 Geforce GTX980[44]和 Tegra X1 Maxwell GPU 核的存储子系统均实现了第三代无损颜色压缩(delta color compression,DCC)[9]。该算法用于纹理、颜色缓冲区以及帧缓冲区的压缩。该算法的核心思想是:首先检查是否能对 4×2 像素区域(压缩率 8:1)进行压缩,如果能压缩,则执行压缩操作;如果不能压缩,则作进一步检查,检查是否能对 2×2 像素区域(压缩率 4:1)进行压缩。如果能压缩,则执行压缩操作,否则不对帧缓冲区的数据进行压缩。对于 4×2 或 2×2 的像素块,压缩编码仅存储一份完整的像素颜色值和 4×2 或 2×2 个像素颜色的差分(difference)。

b)自适应压缩算法。

在主存储器与 GPU 之间,以及在 GPU 内核与 GPU 的本地存储器之间应用数据压缩技术,压缩后的数据降低了存储器的带宽压力,但是位翻转个数(bit toggle count)的增加导致传输功耗增加。为了提高片上网络和片外总线的功耗效率,提高压缩算法的灵活性,压缩算法依据压缩对象的内容自适应地动态选择数据块的大小,并依据数据块的特征计算功耗与时延之积决定是否执行压缩操作[5]。

4 结束语

根据上文的综述,国外在图形流水线数据压缩技术方面的研究取得了一定成果,国内相关研究较少。目前,在 3D 图形渲染流水线的多个处理阶段中联合使用纹理、深度、帧缓冲区压缩等多种带宽节约机制,节省了大量的 GPU 片外存储器的访问带宽,能有效降低系统功耗,提高数据传输和存储效率。相比之下,几何压缩算法串行化执行特征使得解压缩操作难以满足实时性,导致几何缓冲区压缩算法并没有在 GPU 流水线中得到应用。

追求图像真实感的 3D 图形绘图效果使新一代 GPU 流水线的数据压缩技术面临两个方面的挑战:一方面,真实感图形绘制需要使用大量的高分辨率纹理图像,GPU 对纹理数据的访问占片外存储器访问请求总数的 50% 以上,以嵌入式 GPU 为例,读取纹理数据平均花费 62% 的存储器带宽[6],使用固定压缩率的有损压缩技术不仅对最终的图形绘制质量造成损害,而且无法提高纹理图像的压缩率;另一方面,仅使用高级建模技术无法有效减少持续增长的几何数据的原始输入数据量。

基于以上分析,图形流水线数据压缩技术进一步的研究内容主要体现在以下两点:

a)自适应的无损纹理缓冲区压缩算法。依据纹理的类型和纹理图像颜色格式的不同,挖掘压缩对象的数据信息分布特征,自适应地调整纹理图像压缩块尺寸,实现压缩率可变的无损纹理压缩算法。

b)支持实时解压缩的几何缓冲区无损压缩算法。研究顶点位置信息可并行压缩解压缩的几何缓冲区压缩算法,从而提高解压缩阶段的操作并行度,使得 GPU 架构设计人员能够结合 GPU 硬件数据通路的宽度等因素,将几何数据解压缩逻辑高效集成到 GPU 流水线中。

参考文献:

[1] Morein S. ATI Radeon hyperz technology[EB/OL]. (2000-01-01). http://www.graphicshardware.org/previous/www_2000/presentations/ATIHot3D.pdf.

[2] Akenine-Moller T, Strom J. Graphics processing units for handhelds [J]. Proceedings of the IEEE, 2008,96(5):779-789.

[3] Arnau J M, Parcerisa J M, Xekalakis P. Boosting mobile GPU performance with a decoupled access/execute fragment processor[C]// Proc of International Symposium on Computer Architecture. 2012:84-93.

[4] Chu S L, Hsiao C C, Hsieh C C. An energy-efficient unified register file for mobile GPUs[C]//Proc of the 9th IFIP International Conference on Embedded and Ubiquitous Computing. [S.l.]: IEEE Computer Society, 2011:166-173.

[5] Strom J, Akenine-Moller T. Ipackman: high quality, low-complexity texture compression for mobile phones[C]//Proc of SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. New York: ACM Press, 2005:63-70.

[6] Arnau J M, Parcerisa J M, Xekalakis P. Parallel frame rendering: trading responsiveness for energy on a mobile GPU[C]//Proc of the 22nd International Conference on Parallel Architectures Compilation Techniques. [S.l.]: IEEE Computer Society, 2013:7-17.

[7] Gennady P, Evgeny B. A case for toggle-aware compression for GPU sys-

tem[C]// Proc of International Conference on High Performance Computer Architecture. [S. l.]: IEEE Computer Society, 2016:188-200.

[8] NVIDIA. Tegra 4 family GPU architecture [EB/OL]. (2013-02-01). http: //www. NVIDIA. Com/Docs/IO//116757/Tegra _4 _ GPU_Whitepaper_Finalv2. pdf.

[9] NVIDIA. Tegra X1 whitepaper [EB/OL]. (2015-02-01). http://international. download. nvidia. com/pdf/tegra/Tegra-X1-whitepaper-v1.0. pdf.

[10] Lee J, Choe S, Lee S. Mesh geometry compression for mobile graphics[C]//Proc of IEEE Consumer Communication & Networking Conference. [S. l.]: IEEE Press, 2010: 301-305.

[11] Alliez P, Gotsman C. Recent advances in compression of 3D meshes [C]//Advances in Multiresolution for Geometric Modelling. Berlin: Springer, 2005:3-26.

[12] Purnomo B, Bilodeau J, Cohen J D, *et al*. Hardware-compatible vertex compression using quantization and simplification[C]//Proc of ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware. Los Angeles: DBLP, 2005:53-61.

[13] Chhugani J, Kumar S. Geometry engine optimization: cache friendly compressed representation of geometry[C]//Proc of Symposium Interactive 3D Graphics. New York: ACM Press, 2007:9-16.

[14] Meng Shaoliang, Wang Aili, Li Shengming. Compression of 3D triangle meshes based on predictive vector quantization[C]//Proc of the 3rd International Symposium on Systems and Control in Aeronautics and Astronautics. [S. l.]: IEEE Press, 2010:1403-1406.

[15] Christian D, Jens S, Rudiger W. Efficient geometry compression for GPU-based decoding in real-time terrain rendering[J]. Computer Graphics Forum, 2009,28(1): 67-83.

[16] Quirin N M. Real-time geometry decompression on graphics hardware [D]. Erlangen: University of Erlangen Press, 2012.

[17] Wittenbrink C M, Kilgariff E, Prabhu A. Fermi GF100 GPU architecture[J]. IEEE Micro, 2011,31(2):50-59.

[18] Iourcha K, Nayak K S,Hong Zhou. System and method for fixed-rate block-based image compression with inferred pixels values[EB/OL]. (2004-08-10). http://www. freepatentsonline. com/6775417. html.

[19] Delp J, Mitchell R. Image compression using block truncation coding [J]. IEEE Trans on Communications, 1979,27(9):1335-1342.

[20] Jiang Yifei, Huan Dandan. Improved texture compression for S3TC [C]//Proc of Picture Coding Symposium. 2010: 386-389.

[21] Ström J, Pettersson M. ETC2: texture compression using invalid combinations [C]//Proc of SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. New York: ACM Press, 2007: 49-54.

[22] Fenney S. Texture compression using low-frequency signal modulation [C]//Proc of ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. 2003: 84-91.

[23] PowerVR: getting great graphics performance with the PowerVR insider SDK [EB/OL]. (2012 -01-01). http://imgtec. eetrend. com/sites/imgtec. eetrend. com/files/article/201402/1457-2109-powervr-neibusdkgaishu. pdf.

[24] Van Waveren J M P, Castano I. Real-time normal map DXT compression [EB/OL]. (2008-02-07). http://www. nvidia. com/object/real-time-normal-map-dxt-compression. html.

[25] Wilson D. A new compression scheme: 3Dc[EB/OL]. (2004-05-04). http://www. anandtech. com/show/1314/6.

[26] AMD RS880 databook device specification for the RS880[EB/OL]. (2013-01-01). http://support. amd. com/techdocs/46112. pdf.

[27] Guthe S, Goesele M. GPU-based lossless volume data compression [C]//Proc of Conference: the True Vision-Capture, Transmission and Display of 3D Video. 2016:1-4.

[28] Nystad J, Lassen A, Pomianowski A, *et al*. Adaptive scalable texture compression [C]//Proc of International Conference on Computer Graphics and Interactive Techniques. [S. l.]:The Eurographics Association, 2012: 105-114.

[29] Hasselgren J. Efficient depth buffer compression[C]//Proc of International Conference on Computer Graphics and Interactive Technique. [S. l.]: The Eurographics Association, 2006:103-110.

[30] Deroo J, Morein S, Favela B, *et al*. Method and apparatus for compressing parameter values for pixels in a display frame: USA, 6476811[P]. 2002-11-05.

[31] Van Dyke J M, Margeson J E. Method and apparatus for managing and accessing depth data in a computer graphics system: USA, 6961057[P]. 2005-11-01.

[32] Kim H S, Lee J, Kim H, *et al*. A lossless color image compression architecture using a parallel Golomb-Rice hardware CODEC [J]. IEEE Trans on Circuits and Systems for Video Technology, 2011,21(11):1581-1587.

[33] Andersson M, Munkberg J, Akenine-Möller T. Stochastic depth buffer compression using generalized plane encoding[J]. Computer Graphics Forum, 2013,32(2):103-112.

[34] The OpenGL graphics system: a specification version 2.0[EB/OL]. (2004-10-22). https://khronos. org/registry/OpenGL/specs/gl/glspec20. pdf.

[35] AMD M74/M72 databook technical reference manual [EB/OL]. (2013-01-01). http://dev. xdevs. com/attachments/download/276/42013_m74_ds_nda_2. 02. pdf.

[36] Amsinck C, Schneider B O, Bolz J A. Stencil buffer data compression:USA,9390464[P]. 2016-07-12.

[37] Hasselgren J. Performance improvements for the rasterization pipeline [D]. Lund: Lund University Press, 2009.

[38] Rasmusso J, Akenine-Moller T, Hosselgren J, *et al*. Frame buffer compress and decompress method for graphics rendering: USA, 8031973[P]. 2011-10-04.

[39] Ian B. The ARM® Mali™-T880 mobile GPU [C]//Proc of IEEE Hot Chips 27 Symposium. 2016:1-27.

[40] AMD780E databook technical reference manual [EB/OL]. (2009 -01-01). http://support. amd. com/TechDocs/45732. pdf.

[41] Jacob M, Wennersten P, Rasmusson J, *et al*. Floating-point buffer compression in a unified codec architecture[C]//Proc of ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware. [S. l.]:Eurographics Association, 2008:75-84.

[42] Chien S Y, Lok K H, Lu Y C. Low-decoding-latency buffer compression for graphics processing units[J]. IEEE Trans on Multimedia, 2012,14(2): 250-263.

[43] Pool J, Lastra A, Singh M. Lossless compression of variable-precision floating-point buffers on GPUs[C]//Proc of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. New York: ACM Press, 2012:47-54.

[44] Geforce GTX980 whitepaper [EB/OL]. (2014-01-01). http://international. download. nvidia. com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL. PDF.