软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2011,22(8):1927−1933 [doi: 10.3724/SP.J.1001.2011.03932]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# 基于 CUDA 渲染器的顺序独立透明现象的单遍高效绘制*

黄梦成[1,4], 刘 芳[1,3+], 刘学慧[1], 吴恩华[1,2]

[1](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[2](澳门大学 科学技术学院 电脑与资讯科学系,澳门)

[3](中国科学院 计算机网络信息中心 超级计算中心,北京 100190)

[4](中国科学院 研究生院,北京 100049)

## Efficient Rendering of Single-Pass Order-Independent Transparency via CUDA Renderer

HUANG Meng-Cheng[1,4], LIU Fang[1,3+], LIU Xue-Hui[1], WU En-Hua[1,2]

[1](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macao, China)

[3](Supercomputing Center, Computer Network Information Center, The Chinese Academy of Sciences, Beijing 100190, China)

[4](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: liufang@sccas.cn, http://www.sccas.cn

**Huang MC, Liu F, Liu XH, Wu EH. Efficient rendering of single-pass order-independent transparency via CUDA renderer. *Journal of Software*, 2011,22(8):1927−1933.** http://www.jos.org.cn/1000-9825/3932.htm

**Abstract**: This paper presents a highly efficient algorithm for efficient order-independent transparency via compute unified device architecture (CUDA) in a single geometry pass. The study designs a CUDA renderer system to rasterize the scene by the scan-line algorithm, generating multiple fragments for each pixel. Meanwhile, a fixed size array is allocated per pixel in a GPU (graphics processing unit) global memory for storage. Next, this paper describes two schemes to capture and sorts the fragments per pixel via the atomic operations in CUDA. The first scheme stores the depth values of the fragments into an array of the corresponding pixel and sorts them on the fly using the atomicMin operation in CUDA. A following CUDA kernel will blend the fragments per pixel in depth order. The second scheme captures the fragments in rasterization order using the atomicInc operation in CUDA. During post-processing, the fragments per pixel array will be sorted in depth order before blending. Experimental result shows that this algorithm shows a significant improvement in classical depth peeling, producing faithful results.

**Key words**: graphics processing unit; compute unified device architecture; order-independent transparency; depth peeling; atomic operation

**摘 要**: 提出一种顺序独立透明现象的单遍高效绘制算法.首先设计了一个基于计算统一设备架构(compute

* 基金项目: 国家自然科学基金(60833007, 60773030); 国家重点基础研究发展计划(973)(2009CB320802); 国家高技术研究发展计划(863)(2008AA01Z301); 中国科学院知识创新工程重大项目(KGCX1-YW-13); 中国科学院信息化专项(INFO-115-B01); 澳门大学研究基金

收稿时间: 2010-01-05; 定稿时间: 2010-08-13

unified device architecture,简称 CUDA)的可编程渲染器.该系统采用扫描线算法光栅化场景,为每个像素生成多个对应的片元,同时,在 GPU(graphics processing unit)的全局内存上为每个像素分配一个数组,以存储其相应的片元.基于这个框架,提出了两种并发的片元收集及排序策略,以单遍高效地绘制顺序独立的透明现象.第 1 种策略利用 CUDA 的原子操作符 atomicMin 收集各个像素上对应的所有片元并按深度动态排序,在后处理中片元即可按序逐一融合;第 2 种策略采用 CUDA 的原子操作符 atomicInc 按光栅化顺序收集所有片元,然后在后处理中按深度排序后再逐一融合.实验结果表明,与基于传统图形管线的经典深度剥离方法相比,该方法可以更高效地绘制顺序独立的透明现象,同时生成正确的绘制效果.

**关键词:** 图形处理器;计算统一设备架构;顺序独立的透明现象;深度剥离;原子操作

**中图法分类号:** TP391 **文献标识码:** A

在现代计算机应用中,图形处理变得越来越重要.由于其通常需要巨大的计算量,因而现代计算机体系需要一个高效的可以并行处理图形应用的核心处理器.在此基础上,图形处理器(graphics processing unit,简称 GPU)应运而生并快速发展.近年来, NVIDIA 计算统一设备架构(compute unified device architecture,简称 CUDA)的诞生使 GPU 可以进一步推广到高计算密度的应用上[1].基于图形处理器的通用计算已成为当前的研究热点及未来的发展趋势[2].

多片元效果是基于 GPU 的图形应用程序中常见的一种现象,包括顺序独立的透明现象(以下简称透明现象)、半透明现象、折射、体绘制、冲突检测、全局光照、阴影映射,等等.其中,尤以透明现象的绘制最为常见,其绘制通常需要按深度顺序根据透明度值融合该像素位置对应的多个片元的颜色信息.现代 GPU 采用深度缓存 Z-Buffer[3]来收集光栅化后每个像素位置上最近或最远的片元,以绘制不透明现象.而经典的透明现象绘制算法,即深度剥离[4,5]则利用该特性重复光栅化整个场景多遍,每遍收集当前离视点最近的一层,即可正确地按物体到视点的距离即深度顺序融合各片元.当场景规模较小时算法性能较高,但对于大型复杂场景来说,模型的多次顶点变换将成为绘制瓶颈,导致算法的效率下降,因而难以在交互式应用中广泛使用.

针对这个问题,本文提出一种基于 CUDA 渲染器的透明现象的高效绘制算法.算法采用 CUDA 语言实现了一个图形流水管线系统[6],每个像素对应多个生成的片元.采用 GPU 的全局内存为每个像素分配一个数组作为存储目标,利用 CUDA 支持的原子操作符收集片元并排序,从而在单遍场景绘制中实现了透明现象的高效绘制,对于大型场景,加速效果尤为显著.

## 1 相关工作

针对 Z-Buffer[3]的局限性,计算机图形学界产生了许多理论上的改进算法.其中,A-Buffer[7]将每个像素位置上的所有片元都存储在一个动态变化的序列中,并在后处理中按照片元的深度进行排序.其他理论上的扩展还包括 R-Buffer[8],F-Buffer[9]以及 $Z^3$ 算法[10]等.这些算法均较为复杂,尚未在硬件上实现或广泛使用.

Mammen[4]最早提出利用 Z-Buffer[3]的特性将场景按深度逐层剥离,之后,Everitt[5]在硬件上实现了该算法.由于其简单性和健壮性,得以广泛地应用于各种图形处理中,尤其是透明现象的绘制.但是由于算法每次只能剥离一层,对于场景深度复杂度为 $N$ 的场景,算法需要 $N$+1 遍绘制,因此对于大型复杂场景来说,顶点的多次变换会成为计算的主要瓶颈,其效率通常较低.K-Buffer[11,12]利用 GPU 上的多重渲染目标缓存作为存储介质,可为每个像素建立一个大小为 $K$ 的数组.在 GPU 的像素程序中,对新进入的片元按深度进行并发插入排序,则可在一个场景遍历中为每个像素收集最多 $K$ 个片元,理论上比传统的深度剥离方法有最多 $K$ 倍的效率提升.由于算法允许多个片元同时对一个存储位置进行并发读写操作,而在当前 GPU 中这种行为是未定义的,故无法保证正确的读写顺序.因此,在绘制深度层次较为复杂的场景时会出现较明显的瑕疵.Liu[12]采用多遍绘制的方式来解决这个问题,但对于大型复杂场景,其效率下降较快.Myers[13]利用 DirectX 10 中的多重采样抗锯齿缓存(multi-sample anti-aliasing,简称 MSAA)来模拟 A-Buffer.该算法可以完全避免读写冲突,但其性能受限于 MSAA 样本数和模板缓存位数.最近,Bavoil[14]提出一种双面深度剥离方法,利用 NVIDIA Geforce 8 硬件新支持的最大/最小融合特性

实现了一个最大/最小深度缓存,可以在一遍场景绘制中同时剥离最近和最远的两层片元并正确融合.但是,由于其每次仅能剥离 2 层,算法的加速理论上限仅为深度剥离的 2 倍.Liu[15]同样利用该特性实现了一种基于桶排序的高效深度剥离算法,但当发生桶内片元冲突时,需要采用多遍绘制或自适应的方法来提高绘制的准确性,因而需要大量额外的内存开销;并且由于绘制遍数的增加,算法的效率也会相应下降.

## 2 基于 CUDA 渲染器的单遍深度剥离

针对以往相关算法的不足,本文提出一种基于 CUDA 渲染器的透明现象的高效单遍绘制算法.首先,我们设计了一个基于计算统一架构(CUDA)的可编程渲染器来模拟传统流水线的功能.算法将模型中三角形的各个顶点的属性和索引值分别打包成两张纹理,作为参数传入 CUDA 的内核函数中.运行时,线程按块组织并发执行,每个线程处理一个三角形.同时,内核中还需要将用于将模型从模型空间变换到屏幕空间的投影矩阵 MVP 作为参数传入 GPU 的常量内存中.采用 MVP 矩阵将模型中三角形的 3 个顶点都变换到屏幕空间内做视锥裁剪之后,我们采用扫描线算法光栅化投影后的三角形.屏幕上每个被投影三角形的包围盒所覆盖的像素都会被逐一测试,如果该像素点位于三角形内,则会生成一个对应的片元,其深度、法向、颜色等属性均可由该三角形 3 个顶点的对应属性采用双线性插值法近似计算.对于具有遮挡关系的场景,同一个像素位置会对应产生多个不同的片元,我们可以采用 CUDA 特有的原子操作来并发地收集所有的片元并按深度排序.

CUDA 的原子操作可以对位于全局或者共享内存上的数据进行原子读写操作.该操作仅被计算能力为 1.1 及以上的设备支持.如果同一个线程组内的多个线程并发的读写同一个内存地址,这些操作将被串行化并全部执行以消除读写冲突,但其执行的顺序由运行时的线程调度顺序决定,无法预先确定.利用 CUDA 的原子操作,本文设计并实现了两种在单遍场景绘制中高效地收集单个像素上所有片元并按深度排序的算法.在第 3 节和第 4 节中,本文将分别详细介绍这两种策略.

## 3 多级深度测试策略

### 3.1 算法流程

多级深度策略采用全局内存作为存储介质,为每一个像素分配一个固定大小的内存.对于复杂度较低的场景,其大小可以设置为场景的最大深度复杂度,即从任意视角观察该场景得到的最大场景层数.然后,将各像素对应的数组的所有项均采用一个简单的 CUDA 内核函数初始化为最大的整型值(0x7FFFFFFF).绘制场景的过程中,算法利用 CUDA 的原子操作符 atomicMin 将每个像素对应的所有的片元按深度增序动态排序并存储到该数组中.atomicMin 操作符可以在一项原子事务中读取全局或者共享内存中的一个 32 字节的整形源数据,将其与当前的整形数据进行比较,将较小的值存储到该地址中,并返回该地址内的原值.由于片元深度值已被归一化为 0~1 之间的正浮点值,故其可以强制类型转化为整形值而保证其深度大小关系不变.

对于每一个生成的片元,算法都会采用 atomicMin 比较操作符循环测试其对应的像素数组,以将其深度值存入第 1 个非空的数组项中.对于当前数组项,如果其非空且其值大于当前待插入的深度值,则 atomicMin 操作符可以保证将当前值存入该数组项中,并返回原较大的值用于继续测试下一数组项.该策略可以确保在并发更新数组时能够完全避免读写冲突造成的错误,从而获得按深度正确排序的结果.对于某一个像素来说,假设有 $N$ 个线程对应生成了 $N$ 个片元,均映射到了该像素上.这些线程将并发执行,以将其对应的片元深度值并发存储入对应的像素数组中(简记为数组 $A$).每个线程将首先开始第 1 次循环,试图将其值存入该数组的第 1 个数组项 $A[0]$中.atomicMin 操作将保证这些位于同一个全局内存地址上读写操作均被序列化并顺序执行.由于 $A[0]$已被初始化为最大的整型值,故第 1 个在该项上进行原子比较的线程必将成功地写入其深度值,返回原最大深度值并退出,剩余线程数减少为 $N$–1 个.其余的线程将先后按调度顺序分别测试 $A[0]$.atomicMin 操作可以保证最终所有深度中的最小值可以存储入该项,其余的 $N$–1 个线程将拥有其余的 $N$–1 个深度值,并先后开始测试下一个位置 $A[1]$.以此类推,当所有的线程都退出循环时,各深度值将按深度增序动态存入对应的像素数组中.最终,绘

制结果可以存入像素缓存对象中,并采用纹理映射直接绘制到屏幕上.

### 3.2 片元深度和颜色的同时收集

对于需要多个片元属性的应用,如透明现象,由于 CUDA 目前尚不支持 64 位的原子操作,故无法简单地将深度和颜色信息合并打包成一个 64 位的长整形数,同时收集和排序.同时,CUDA 也尚未提供任何内置的关键区策略来保证对一块连续内存区域的原子操作,因而难以在一个原子事务中同时更新深度缓存和颜色缓存.比较直观的一种解决方法就是采用增加一遍绘制方式来收集与最近一层片元深度对应的颜色信息,但是相应的算法效率会下降一半以上.在本文算法中,可以采用 atomicExch 原子操作符近似模拟关键区来减少读写冲突[16].

我们将每个像素对应的深度缓存和颜色缓存视为一个关键区,并为其设置一个初始化为 0 的无符号整型的信号量.由于 atomicExch 原子操作符在一个原子事务内将一个数值与全局或者共享内存上的数值进行交换,因此对于每一个映射到该像素位置的片元,采用 atomicExch 操作符用数值 1 将该信号量置 1,并返回其内的原值.如果返回的是 0,则表明该关键区内没有其他线程在访问,因而该线程可以独占该关键区,并进行相应的读写操作而不会引发冲突.当访问结束后,同理可以采用 atomicExch 将该信号量置 0,表明该线程已退出;如果返回的值是 1,则表明有线程正在访问该关键区.由于 CUDA 线程调度器的局限性,尚不支持挂起该线程并在其他线程退出该关键区时唤醒该线程,故我们采用循环测试的方法来模拟这一过程,直到返回的信号量的值为 0 时,该线程才可以进入关键区.但是,当同一个线程块内的不同线程需要同时访问同一个关键区时,CUDA 调度器的局限性会导致线程死锁.此外,由于 CUDA 内核中的寄存器数目有限,如果采用多次循环测试,则可能会导致程序性能大幅度下降甚至崩溃,故我们只能采用较少的测试次数,如果其信号量值仍为 1,则强行访问关键区.实验结果表明,采用 32 次循环测试可以消除大部分读写冲突,但是此时程序性能下降较大.

虽然该模拟方式可以减少大部分读写冲突,但是仍有少量片元冲突无法完全避免,因而会产生少量绘制错误.通过增加循环次数来减少冲突则会影响算法的稳定性,同时会降低算法效率.因而我们设计了一种新的替代方式,以在单遍绘制中较好地解决这个问题.算法基本流程如下:首先,将规范化后的深度值映射到 0.5~1 之间,以保证所有的深度值都具有相同的符号位和指数位(共 9 比特);然后,我们将其强制类型转换为整型,左移 9 个比特位,并将后 12 个比特位置 0,这样转换后的深度值与原深度值的大小关系可以保持一致.对于透明度值固定的片元,我们将其颜色属性的 RGB 三通道值分别量化到 0~255 之间,并打包拼接为一个 24 比特的整型值.其高 12 位和低 12 位可以分别拼接到上述转换过的深度值的两个副本的后 12 比特空位上.合成的两个新值可以采用两个连续的原子操作来分别更新深度缓存和颜色缓存.由于此时这两个合成值的高 20 位均相同,且与原深度值的大小关系一致,因而可以保证深度缓存和颜色缓存的对应项中存储的是同一个片元的信息.当然,当两个片元距离非常近时,由于深度精度的丢失可能会产生二者不一致的现象,但由于此时片元的颜色通常很相近,因而通常不会产生明显瑕疵.在后处理中,片元的颜色属性可以按上述方法的逆处理恢复,且其已按深度有序.

### 3.3 扩展的多层深度剥离

对于深度复杂度较高的场景,由于 GPU 内存大小的局限性,按照场景的最大深度复杂度来分配片元存储数组可能会导致 GPU 内存耗尽,因而无法正常运行.针对这个问题,我们可以类似地采用扩展的多层深度剥离算法,灵活地控制内存的需求量:每遍只剥离场景中的前 $M$ 层,之后按序融合所剥离的片元;在下一遍中,如果片元深度小于等于之前所收集到的最大片元深度,则表明该片元已在之前的绘制中收集过,可以直接抛弃,其余片元则可以继续按照多级深度测试策略收集并排序,直到收集完场景中的所有层.当 $M$ 取 1 时,算法即退化成传统的深度剥离算法.对于复杂度为 $N(N\geq M)$的场景,该扩展算法需要的内存量降为原算法的 $M/N$.同时,由于其需要将绘制遍数增加至$[N/M]$遍,导致算法性能的相应下降,故而是一种算法性能和内存需求量的折中方案.

## 4 固定数组缓存策略

类似于 A-Buffer[6]的思想,我们在全局内存上为每一个像素分配一个固定大小的结构数组以收集片元信息.透明现象的绘制需要按深度有序收集各片元的颜色,我们将其 RGBA 四通道浮点型的颜色值量化为 4 个

0~255 的无符号字符型值,并打包拼接为一个 32 位的整形值,以减少所需的片元存储空间.由于该整型值可以强制类型转换为浮点数,故我们可以为每个像素分配一个 float2 类型的结构数组,以同时存储深度和颜色信息.同时,我们为每个像素分配一个整形计数器并初始化为 0.在场景绘制过程中,每个光栅化生成的片元都会将相应像素位置对应的计数器采用 atomicInc 操作符增加 1.采用这种方式可以按光栅化顺序为每个片元分配一个唯一的序号,并将其存入该序号对应的数组项中.在用于后处理的 CUDA 内核函数中,每个线程可以先把对应像素位置的片元数组全部载入寄存器中,然后采用插入排序或者 bitonic 排序法排序.排序所得结果可以直接用于绘制透明现象,以避免在全局寄存器上的额外的读写操作.

该策略可以解决多级深度策略无法同时正确收集多个片元信息的不足之处;同时,多级深度策略是在全局内存上动态排序的,而固定数组缓存策略则是在寄存器级进行排序的,因而具有较低的访问延时,效率更高.但其不足之处在于,对于只需要前 $N$ 层片元信息的应用来说,多级深度策略只需要分配 $N$ 层的存储空间,而固定数组缓存策略则要求所有的片元完全收集后才可以排序选取前 $N$ 层,因而内存利用效率稍低.

## 5 实验结果与分析

图 1 是采用是本文的多级深度测试策略以及固定数组缓存策略对于 Lucy 模型(约 200 万个面片)的透明现象的绘制结果,以及其与深度剥离算法(DP)的绘制结果的对比图.系统配置为 NVIDIA Geforce 280 显卡,Intel Duo CPU 2.4GHz,3G 内存,采用的 CUDA 驱动版本号为 2.1.容易看出,本文的两种绘制策略都可以生成与深度剥离绘制非常相似的绘制结果,更多的绘制结果如图 2 所示(其中,前两幅图为多级深度测试策略对 Armadillo 和 Lion 模型的绘制结果,后两幅图为固定数组缓存策略对 Dragon 和 Tank 模型的绘制结果).其他多片元效果的绘制,如半透明现象、深度域、折射、体绘制等,由于这些应用都需要同时收集每个像素对应的多层深度信息,并在后处理中按深度顺序进行融合处理,因而都可以类似地采用本文算法来绘制,并从中得到相应的性能提高.

(a) Result of multi-depth test scheme (b) Result of fixed A-buffer scheme (c) Result of depth peeling

(a) 多级深度测试策略的绘制结果 (b) 固定数组缓存策略的绘制结果 (c) 深度剥离算法绘制结果

Fig.1 Comparison of order-independent transparency on the Lucy model

图 1 Lucy 模型顺序独立透明现象绘制对比图

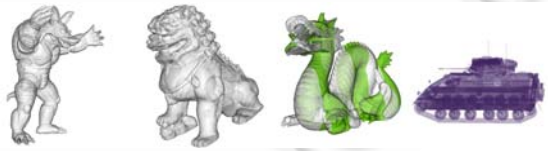

Fig.2 More results of order-independent transparency on various models

图 2 不同模型顺序独立透明现象的更多绘制结果

## 6 算法分析

### 6.1 性能分析

对于本文提出的两种片元收集策略,假设对于深度复杂度为 $N$ 的场景,绘制一遍场景顶点变换所需的时间为 $T_V$,片元收集和排序所需的时间是 $T_f$;而传统深度剥离中顶点变换的时间开销与本文算法相同,但其片元处理时间几乎可以完全忽略,故本文算法对于传统深度剥离理论上的加速比是

$$K=\frac{N\times T_v}{T_v+T_f} \tag{1}$$

当场景较大时,本文算法中的片元处理时间可以被读取顶点所需的内存访问延时所掩盖.故顶点变换的时间将成为绘制的主要开销,即有 $T_V>>T_f$,故此时可以基本忽略 $T_f$,算法加速比约为 $N$ 倍.同时,对于一些面片数较少的场景,如 Armadillo 模型,由于此时像素程序执行时间 $T_f$ 与相比绘制场景时间 $T_V$ 相比变得不可忽略,算法加速比下降下降,故本文算法更适用于大型场景的绘制.

表 1 采用绘制同一组模型的透明现象的方式来比较本文提出的两种片元收集算法与其他算法的效率.实验数据表明,本文提出的固定数组缓存策略(fixed A-buffer scheme,简称 FABS)与基于传统流水管线的深度剥离算法相比约有 $N$ 倍的加速,与双层深度剥离算法(dual depth peeling,简称 DDP)相比约有 $N$/2 倍的加速.采用颜色和深度打包收集的多级深度测试策略(multi-depth test scheme,简称 MDTS)的效率约为固定数组缓存策略的 2/3,这是因为多级深度测试策略需要双倍的原子操作在全局内存上对所有的片元动态排序.K-Buffer[11]和基于桶排序的深度剥离算法(bucket depth peeling,简称 BDP)[15]虽然与本文算法的加速比相似,但是前者受读写冲突的影响,后者受桶内片元冲突的影响,导致这两种算法都会有较明显的瑕疵,难以生成正确的绘制结果.模板引导的 A-Buffer 算法(stencil-routed A-buffer,简称 SRAB)[13]、基于桶排序的深度剥离的多遍扩展算法(BDP2)[15]及自适应划分扩展算法(adaptive depth peeling,简称 ADP)[15]都需要两遍场景绘制,故与之相比,本文算法的加速比均约为 2 倍.同时,K-Buffer 的扩展算法(Liu)[12]需要多遍绘制来保证绘制结果的正确性,而本文算法仅需单遍,故其性能也超过了 K-Buffer 的扩展算法.综上所述,本文算法的性能与基于传统流水管线的绘制方法相比均有很大的提高,同时可以保证绘制结果正确.

**Table 1** Comparison of frame rates (fps) and geometry passes (g) between our algorithm and others

**表 1** 本文算法与其他算法对于不同模型的绘制帧率(fps)及场景绘制遍数(g)的比较

| Model | Triangle number | MDTS | FABS | K-Buffer | BDP | BDP2 | ADP | SRAB | Liu | DDP | DP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Armadillo | 349K | 256fps | 434fps | 294fps | 258fps | 128fps | 106fps | 168fps | 46fps | 41fps | 20fps |
| | | | | | | 2g | 2g | 2g | 6g | 8g | 13g |
| Dragon | 871K | 297fps | 363fps | 304fps | 317fps | 142fps | 116fps | 106fps | 60fps | 40fps | 29fps |
| | | | | | | 2g | 2g | 2g | 6g | 8g | 13g |
| Buddha | 1.0M | 162fps | 220fps | 248fps | 156fps | 116fps | 98fps | 89fps | 42fps | 34fps | 26fps |
| | | | | | | 2g | 2g | 2g | 9g | 8g | 13g |
| Lion | 1.3M | 138fps | 213fps | 163fps | 153fps | 76fps | 70 fps | 42fps | 15fps | 21fps | 13fps |
| | | | | | | 2g | 2g | 2g | 9g | 8g | 13g |
| Statue | 10.0M | 39fps | 61fps | 29fps | 29fps | 14fps | 13fps | 8fps | 2fps | 3fps | 2fps |
| | | | | | | 2g | 2g | 2g | 14g | 9g | 15g |

### 6.2 算法局限性

本文两种算法的局限性主要有以下几点:首先,由于 CUDA 尚不支持 64 位的原子比较操作符,故对于需要多个片元属性的应用来说,多级深度测试策略需要损失一定的深度精度来同时收集多个片元属性,故无法完全保证结果的正确性.由于该功能已于硬件上实现但尚未对 CUDA 开放,我们相信,未来版本的 CUDA 会支持该功能以完全解决这个问题;其次,对于深度复杂度很高的场景,由于本文两种算法排序的复杂度均非线性,故其性能下降较大,在未来的工作中,可以采用桶排序方式来进一步降低排序的算法复杂度.此外,如上一节中所分析的,算法所需的内存量也较大.本文提出的扩展的多层深度剥离方法可以降低内存需求量,但是由于绘制遍数的增加,算法的性能也会相应地下降.

## 7 结束语

本文基于 CUDA 渲染器提出并实现了两种单遍高效绘制透明现象的算法.实验结果表明,这两种算法比基于传统流水管线的相关算法均有很大的加速.目前,本文的 CUDA 渲染器仅为一个原型系统,未来的主要工作方向在于,进一步利用 CUDA 渲染器的可编程性解决遮挡剔除、反走样等现有图形管线上尚难以高效解决的问题.相信在不久的将来,将会有更加成熟、完善的基于 CUDA 的可编程流水管线出现.

**References**:

[1] Wu EH, Liu YQ. General purpose computation on GPU. Journal of Computer-Aided Design & Computer Graphics, 2004,16(5): 601−612 (in Chinese with English abstract).

[2] Wu EH. State of the art and future challenge on general purpose computation by graphics processing unit. Journal of Software, 2004,15(10):1493−1504 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1493.htm

[3] Catmull BE. A subdivision algorithm for computer display of curved surfaces [Ph.D. Thesis]. Salt Lake City: The University of Utah, 1974.

[4] Mammen A. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Computer Graphics and Applications, 1989,9(4):43−55. [doi: 10.1109/38.31463]

[5] Everitt C. Interactive order-independent transparency. Technical Report, Santa Clara: NVIDIA Corporation, 2001.

[6] Liu F, Huang MC, Liu XH, Wu EH. CUDA renderer: A programmable graphics pipeline. In: Proc. of the ACM SIGGRAPH Asia 2009. 2009. [doi: 10.1145/1667146.1667189]

[7] Carpenter L. The A-buffer, an antialiased hidden surface method. ACM SIGGRAPH Computer Graphics, 1984,18(3):103−108. [doi: 10.1145/800031.808585]

[8] Wittenbrink CM. R-Buffer: A pointerless A-buffer hardware architecture. In: Knittel G, ed. Proc. of the ACM SIGGRAPH/ EUROGRAPHICS Workshop on Graphics Hardware. New York: ACM Press, 2001. 73−80. [doi: 10.1145/383507.383529]

[9] Mark WR, Proudfoot K. The F-buffer: A rasterization-order FIFO buffer for multi-pass rendering. In: Knittel G, ed. Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware. New York: ACM Press, 2001. 57−64. [doi: 10.1145/ 383507.383527]

[10] Jouppi NP, Chang CF. $Z^3$: An economical hardware technique for high-quality antialiasing and transparency. In: Knittel G, ed. Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware. New York: ACM Press, 1999. 85−93. [doi: 10.1145/311534.311582]

[11] Bavoil L, Callahan SP, Lefohn A, Comba JLD, Silva CT. Multi-Fragment effects on the GPU using the K-buffer. In: Sloan P, Gooch B, eds. Proc. of the 2007 Symp. on Interactive 3D Graphics and Games. New York: ACM Press, 2007. 97−104. [doi: 10.1145/1230100.1230117]

[12] Liu BQ, Wei LY, Xu YQ, Wu EH. Multi-Layer depth peeling via fragment sort. In: Pan YH, Thalmann D, Peng QS, eds. Proc. of the IEEE Int'l Conf. on CAD/Graphics. 2009. 452−456.

[13] Myers K, Bavoil L. Stencil routed A-buffer. In: Proc. of the ACM SIGGRAPH 2007 Technical Sketch Program. New York: ACM Press, 2007. 21. [doi: 10.1145/1278780.1278806]

[14] Bavoil L, Myers K. Order independent transparency with dual depth peeling. Technical Report, Santa Clara: NVIDIA Cooperation, 2008.

[15] Liu F, Huang MC, Liu XH, Wu EH. Efficient depth peeling via bucket sort. In: Luebke D, Slusallek P, eds. Proc. of the 1st High Performance Graphics Conf. New York: ACM Press, 2009. 51−57. [doi: 10.1145/1572769.1572779]

[16] Zhou K, Hou QM, Ren Z, Gong MM, Sun X, Guo BN. Renderants: Interactive Reyes rendering on GPUs. ACM Trans. on Graphics, 2009,28(5):1−11. [doi: 10.1145/1661412.1618501]

**附中文参考文献:**

[1] 吴恩华,柳有权.基于图形处理器(GPU)的通用计算.计算机辅助设计与图形学学报,2004,16(5):601−612.

[2] 吴恩华.图形处理器用于通用计算的技术、现状及其挑战.软件学报,2004,15(10):1493−1504. http://www.jos.org.cn/1000-9825/15/1493.htm

**黄梦成**(1984－),男,江西南昌人,博士,主要研究领域为 GPU 绘制,图像处理,搜索技术.

**刘学慧**(1968－),女,博士,副研究员,CCF 高级会员,主要研究领域为计算机图形学,虚拟现实.

**刘芳**(1982－),女,博士,助理研究员,主要研究领域为 GPU 绘制,GPGPU.

**吴恩华**(1947－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为计算机图形学,科学计算可视化,虚拟现实.