doi：10.11830/ISSN.1000-5013.201605020

采用 GPU 的提升纹理缓存命中光线投射方法

杜松江[1]，张思超[2]

（1. 长江大学工程技术学院 信息工程学院，湖北 荆州 434020；
2. 中国矿业大学 机电工程学院，江苏 徐州 221116）

摘要：提出一种改善纹理缓存命中率的方法.首先，分析图形处理器(GPU)中三维纹理组织的布局特性；进而提出根据视点的变化动态选择线程配置的策略，目的在于最小化 warp 级的投射光线纹理访存跨距；最后，算法用 CUDA(compute unified device architecture)实现并验证.实验结果表明：当视点分别围绕 x，y，z 坐标轴旋转时，改进后算法的帧速率分别为改进前的 1.08，1.14，0.98 倍.

关键词：三维纹理；光线投射；图形处理单元；纹理缓存

中图分类号：TP 391　　**文献标志码**：A　　**文章编号**：1000-5013(2016)05-0627-06

Improving Texture Cache-Hit Rate of GPU-Based Ray Casting

DU Songjiang[1], ZHANG Sichao[2]

(1. College of Information Engineering, Yangtze University College of Engineering Technology, Jingzhou 434020, China;
2. School of Mechanical and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: This paper presents a method of improving the texture cache hitrate for GPU-based volume rendering. Firstly, we analyze the data layout of 3D texture in GPU. Based on it, a dynamic strategy of selecting the thread block shape according to the viewpoint is proposed. The strategy can minimize the access stride for the warp-level threads. Finally, we realize the method in CUDA (compute unified device architecture) and testify the effectiveness. The experimental results show that when the viewpoint rotates around the x-, y-, z- axis, the frame rates are 1.08, 1.14 and 0.98 time faster than that of static thread block shape configuration, respectively.

Keywords: 3D texture; ray casting; graphics processing unit; texture cache

光线投射算法作为体绘制技术中的一种，在医学、天文、地学等领域有着广泛的应用.由于算法需要对屏幕上的每个像素做运算，因此，光线投射算法的计算量很大，很难满足实时交互方面的应用.另一方面，投射光线之间是相互独立的，该算法适合并行化实现.Kruger 等[1]通过图形应用程序编程接口(application programming interface，API)的功能调用，将体数据作为三维纹理保存在图形处理器(graphics processing unit，GPU)中，利用图形流水线的可编程着色器进行算法的实现.之后，大量的研究工作都是针对图形流水线模式下的算法进行改进和优化[2-4].2007 年以来，Nvidia 公司推出支持计算统一设备

收稿日期：2016-03-15

通信作者：张思超(1973-)，男，教授，博士，主要从事数据库应用、软件工程的研究.E-mail：dusongjiang2014@163.com.

基金项目：国家自然科学基金资助项目(51204186)

架构 CUDA (compute unified device architecture)的GPU,该技术使GPU从图形领域的应用进一步扩展到了更多的领域.CUDA编程模型可以使更多的通用算法在GPU上得到了实现,并取得可观的加速效果[5].在支持CUDA的GPU上进行光线投射算法加速的研究工作中,Marsalekl等[6]首先实现了算法的移植,加速效果优于基于Shader的实现.在不降低绘制速度的前提下,Zhang等[7]利用3次B样条改善CUDA的光线投射算法的视觉效果.然而,通过观察发现,当体数据规模较大时,绘制帧数率的变化受视点变换的影响严.本文从GPU的硬件体系结构和访存模型出发,最小化warp级投射光线访问相邻体数据时的跨距,从而提高绘制性能.

1 CUDA编程模型

CUDA编程模型将CPU作为主机,GPU作为协处理器.CPU负责进行逻辑性强的事务处理和串行计算,GPU则专注于执行高度线程化的并行处理任务[8-10].CUDA计算流程通常包括CPU到GPU数据传递、Kernel函数执行、GPU到CPU数据传递3个步骤.

CUDA采用单指令多线程(single instruction multiple thread,SIMT)执行模式,即GPU上的所有线程并行执行内核函数Kernel[11].另外,CUDA将线程组织成块网格、线程块、线程3个不同的层次[12-13],并采用多层次的存储器结构.存储器包括只对单个线程可见的寄存器和本地存储器、对块内线程可见的共享存储器、对所有线程可见的全局存储器等.其中,全局内存可以被绑定为纹理内存,主要用在图形图像等应用中.

2 纹理访存分析及线程配置策略

不同于全局内存的2级缓存,纹理内存提供的缓存主要是缓存空间上相邻的数据[14].空间相邻的投射光线对体数据采样时,同样会访问空间上相邻的体素.因此,使用纹理内存保存体数据是合适的选择.然而在交互过程中,视点并非静止的.在不同位置访问体数据所表现出的缓存效果也是不同的.

2.1 GPU纹理内存的访存分析

将体数据保存为三维纹理时,数据在纹理内存中的布局,如图1所示.

令体数据的长宽高都为 N,且 $N=2^l$.图1中:三维纹理可以看成是二维纹理切片沿 z 方向的集合.在全局内存中,数据以一维线性的方式保存;而在纹理内存中,每个二维纹理切片在纹理存储器中以Morton编码的方式组织(箭头所指的)[15-16].Morton编码具有递归的特性,因此,第 l 层的纹元编码由第 l-1层的编码决定,层次之间的结构关系,如图2所示.

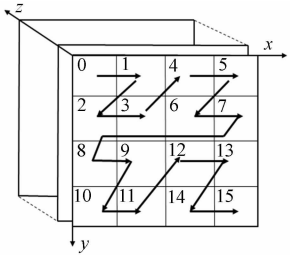


图1 三维纹理布局

Fig. 1 3D texture layout

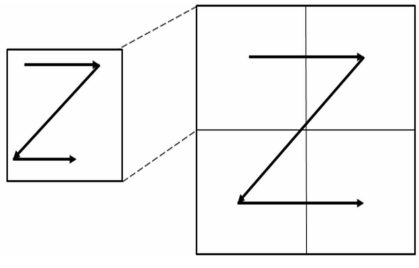


图2 莫顿编码的层次结构

Fig. 2 Hierarchical structure of Morton code

该编码方式虽然优化了二维访问的空间局部性,但是造成相邻数据的间隔距离不均匀,如图1中的纹元1和4,纹元2和3.将相邻纹元之间的跨度大小分为以下两种情况讨论.

1) 相邻纹元有不同的 z 坐标.这种情况下,相邻纹元出现在两个相邻的二维切片上.因为有相同的 x 和 y 坐标,相邻纹元之间的跨距都是 N^2.

2) 相邻的纹元有不同的 x 或 y.在该情况下,相邻纹元出现在同一切片中.相邻纹元之间的跨度与坐标轴平行有关.当相邻体素与 x 轴平行时,相邻体素的跨距范围为1~3.然而,第 l 层的最大跨距出现在内部 l-1层的相邻块之间(图2).例如,沿着水平坐标轴的纹元为 a 和 b;沿着垂直坐标轴的纹元为

c 和 d；纹元 b 和 d 的物理索引分别为 $4l$-1 和 $2\times(4l-1)$；纹元 a 和 c 的物理索引分别为 $\sum_{k=0}^{l-2}4^k$ 和 $2\times\sum_{k=0}^{l-2}4^k$. 因而，沿着 x 坐标轴的最大跨距为

$$4^{l-1}-\sum_{k=0}^{l-2}4^k=\frac{2\times4^{l-1}+1}{3}. \tag{1}$$

沿着 y 坐标轴的最大跨距为

$$2\cdot4^{l-1}-2\cdot\sum_{k=0}^{l-2}4^k=2\cdot\frac{2\times4^{l-1}+1}{3}. \tag{2}$$

因为 $N=2^l$，故沿着 x，y 坐标轴的纹元最大访问跨距分别为 $(N2+2)/6$ 和 $(N2+2)/3$. 由以上分析可知，沿着 x，y，或 z 坐标轴访问相邻纹元时，访问跨距近似为 1∶2∶6. 由于纹理缓存的大小仅为几十 KB，当体数据切片过大时，对纹理内存的大跨距访问会造成频繁的缓存命中失效，使得算法性能的下降.

不同视点访问同一三维纹理切片的示意图，如图 3 所示. 由图 3 可知：c 视点位置的访存效率最高，因为相邻的投射光线访问的是沿 x 轴平行的相邻体素；反之，b 视点访存性能最差，因为相邻的投射光线访问的是不同切片上的体数据. 为提高相邻投射光线在访问纹理内存时的纹理缓存命中率，访问纹理内存数据时应尽量沿着 x 方向进行.

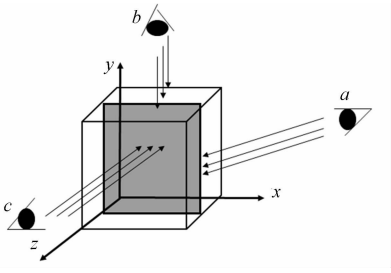


图 3　不同视点访问同一三维纹理切片的示意图

Fig. 3　Sketch of accessing same slice at different viewpoints

2.2　线程块及 warp 的几何形状

CUDA 将所有并行线程等分为多个线程块. 一个线程块中的线程由 1 个或多个 warp 组成，而一个 warp 是连续的 32 个线程. 线程块之间和线程块内部可以组织成一维或者二维的形状. 当将线程块设置为二维形状时，也间接决定了 warp 的形状. 线程块和 warp 的形状，如表 1 所示.

表 1　线程块和 warp 的形状

Tab. 1　Geometry shape of thread block warp

线程块形状	warp 形状	高宽比	线程块形状	warp 形状	高宽比	线程块形状	warp 形状	高宽比
1×256	1×32	1∶32	8×32	8×4	2∶1	64×4	32×1	32∶1
2×128	2×16	1∶8	16×16	16×2	8∶1	128×2	32×1	32∶1
4×64	4×8	1∶2	32×8	32×1	32∶1	256×1	32×1	32∶1

由表 1 可知：线程块的几何形状表现为从垂直到水平的过程；对应的 warp 形状也做相应的变化.

2.3　基于视点的线程块形状动态分配

假定视点位于坐标原点 O，体坐标轴中的两个轴代表的平面和成像屏幕平行时的 6 种典型情况，如图 4 所示.

灰色平面为当体坐标轴中的两个轴和成像屏幕平行时，体数据的二维切片和成像屏幕之间的 6 种状态. 前文描述中，沿 x 轴访问相邻体数据时跨距最小，因此，令体数据的 x 轴为主轴. 基于视点的动态线程形状配置主要通过以下 3 个步骤确定.

步骤 1　平行平面检测. 在视点变换的过程中，从 xy-，xz-及 yz-平面中选择和屏幕最为平行的平面，如图 4(a)和

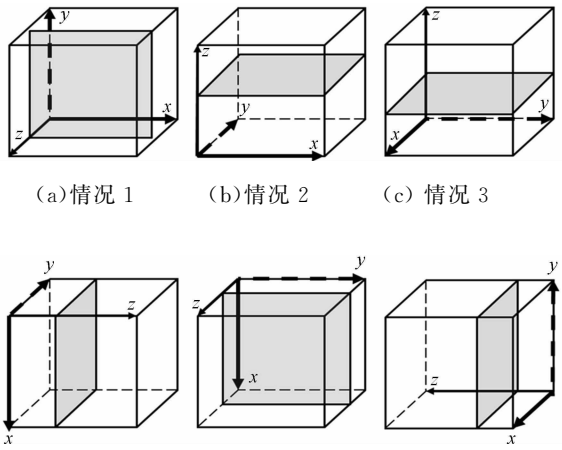


图 4　体数据和成像平面之间的 6 种典型情况

Fig. 4　6 kinds of typical situation between volume axes and screen

图4(e)中的xy-平面.

步骤2 确定主轴.从平行平面中选择访存跨距最小的轴为主轴.例如,图4(a),图4(e)中的xy-平面中,由于沿x轴访问比沿y轴的访问跨距小,因此,选择x轴为主轴.

步骤3 确定线程块的形状.根据主轴被绘制在屏幕上的方向选择线程块的形状.如果主轴在屏幕上用垂直线绘制,选择垂直warp的线程块,如图4(d),图4(e),线程块形状为1×256;如果主轴在屏幕上用水平线绘制,选择水平warp的线程块,如图4(a),图4(b),线程块形状为256×1.对于其他处于中间过度状态的情况,选择垂直和水平混合的warp形状.为此,将0°~90°的旋转区域再次细分成6组过渡区域,即每15°为一个过渡区域.

在交互过程中,视点的任意旋转变换可以看做是分别绕3个坐标轴的旋转变换组合而成.为更进一步说明过渡区域的线程块选择策略分别讨论绕x轴旋转、绕y轴旋转、绕z轴旋转的选择策略.旋转角度分别为Θ_x,Θ_y及Θ_z,细分后的过渡区域的选择,如表2所示.

表2 线程块几何形状的动态选择

Tab.2 Dynamic choices of thread block geometrical shapes

旋转轴	旋转区域					
	0°~15°	15°~30°	30°~45°	45°~60°	60°~75°	75°~90°
(Θ_x,0,0)	32×1	32×1	32×1	32×1	32×1	32×1
(0,Θ_y,0)	32×1	16×2	8×4	4×8	2×16	1×32
(0,0,Θ_z)	32×1	16×2	8×4	4×8	2×16	1×32

由表2可知:绕x轴旋转时,主轴x-始终平行于屏幕.warp中的相邻投射光线沿着x轴访问体素时跨距最小,缓存命中率也就越大.因此,选择水平状warp的线程块形状,即256×1;绕y轴旋转时,平行平面由xy平面逐步过渡到yz平面,主轴也由x轴变为y轴.为了尽量减少访存跨距增大引起的命中下降,warp的形状也由水平状逐渐过渡到垂直状;同样地,绕z轴旋转时,虽然平行平面始终为xy平面,但主轴由水平状变为垂直状,warp的形状也跟着相应的变化,与绕y轴旋转不同的是,在这个过程中主轴没有发生改变.

旋转角度为90°~360°时,warp形状及线程块的形状配置利用几何的对称关系得到.

3 算法框架

假设成像屏幕的高和宽分为W和H,整个屏幕成像所需要的线程数量为$W\times H$.一般情况下,GPU中一个线程block中的线程数量远远低于绘制整个屏幕需要的线程数量.通过将屏幕分块,采用屏幕块对应线程块的做法可以解决这一问题.线程块采用二维布局,维度大小为$w\times h$.同样地,线程grid也采用二维结构,总共需要的线程块个数为ceil(W/w) ×ceil(H/h)个.线程grid中线程坐标和屏幕上每个像素坐标的对应关系为

$$u = \text{blockId}x.x \times \text{BLOCK\_SIZE} + \text{threadId}x.x,$$

$$v = \text{blockId}x.y \times \text{BLOCK\_SIZE} + \text{threadId}x.y.$$

除了将体数据作为三维纹理保存外,充分利用GPU中各种存储器的特性,即传递函数主要用于将投射光线击中的体素值转换为颜色值和不透明度,将其绑定为类型为float4的一维纹理,只需保存少量颜色值,其余的值可以利用硬件支持的插值算法生成.体数据显示到屏幕上是一个三维对象变换为二维图像的过程,并且每个投射光线所代表的线程都会用到该变换,利用该存储器的广播功能,将变换矩阵保存在常量内存中.投射光线在对体数据进行采样时,使用寄存器变量保存临时累加值.最后,将每个投射光线的计算结果写入全局内存.

4 实验和分析

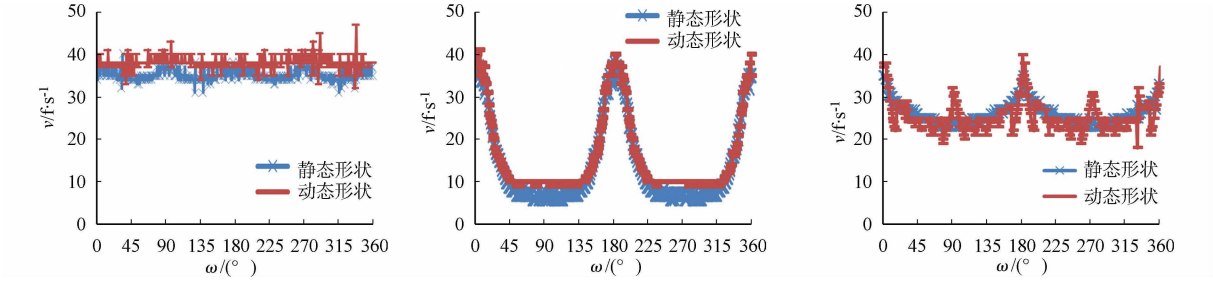
为验证基于warp级纹理访存优化的线程块动态配置方法,实验部分主要通过绘制帧速率的提高说明方法的有效性.算法所用的计算平台为Nvidia开普勒GK110架构的Geforce GT740M型GPU;CUDA SDK为5.5.GPU的硬件规格的计算能力为3.5;CUDA核心数量为384;处理器频率为1.03

GHz；SM 数量为 2；全局内存为 2 GB；共享内存为 48 KB.

所用数据为 1 024×1 024×1 024 的 Hydrogen Atoms 体数据，每个体素大小为 8 bit. 成像屏幕的大小为 1 024×1 024.

将视点分别绕 x 轴、y 轴、z 轴旋转 360°，分别测量不采用动态线程配置时的帧速率和采用动态线程配置后的帧速率. 每个线程块的大小设定为 256，不采用动态配置的线程块形状为 16×16，采用动态线程块配置的形状根据前述方法进行变化.

体数据分别绕 x，y，z 轴旋转 360°的帧速率结果进行对比，如图 5 所示. 图 5 中：ω 为旋转角度；v 为帧速率.



(a) x 轴　　(b) y 轴　　(c) z 轴

图 5　静态形状和动态形状的帧速率对比图

Fig. 5　Comparison chart of frame rate between static shape and dynamic shape

由图 5(a)可知：虽然在旋转过程中平行平面一直在变化，但是由于主轴 x 轴一直是水平无变化的，因此，整个旋转范围内的绘制帧速率平均高于绕其他两个轴的旋转. 在采用动态配置优化后，更能适应 warp 中相邻投射光线访问纹理内存的特点，绘制性能有了进一步的提升.

由图 5(b)可知：帧速率表现出了很大的差异. 在旋转范围为 0～90°时，因为主轴由 x 轴逐渐变化成为 y 轴，该变化过程导致 warp 级投射光线访问纹理缓存时的命中率降低，绘制性能随旋转角度的增加而降低. 当旋转角度由 90°～180°改变时，主轴又逐渐变回为 x 轴，绘制帧速率也得到回升. 由于体数据本身具有对称性，当旋转角度为 180°～360°时，绘制帧速率同样也表现出了对称性. 采用动态配置方法也起到了改善绘制性能的作用.

由图 5(c)可知：当绕 z 轴旋转时，虽然平行平面没有发生改变，但是主轴 x-的方向却一直在变化，算法的绘制性造成一些影响，总体情况比绕 y 轴时要好. 值得注意的是，当绕 z-轴旋转时，在理论上动态线程配置是能够适应主轴的旋转改变，并提升性能，但实际效果相反，绘制速率表现出了震荡效应. 造成该现象的原因是线程块的几何形状在旋转角度为 0°～15°时，选择的是 256×1 的水平状线程块，warp 形状高宽比为 32∶1. 该高宽比下，线程块的形状有 4 个备选方案.

线程块形状改为 32×8，0°～90°旋转区间的运行效果，如图 6 所示. 由图 6 可知：修改线程块形状后，0°～15°这一区间的帧速率变化得到了改善.

最后，体数据分别绕 x，y，z 坐标轴旋转 360°时，提出方法的总体改进效果，如表 3 所示.

表 3　提出方法的总体改进效果

Tab. 3　Overall improvement of proposed method　　f·s^{-1}

坐标轴	v(静态)	v(动态)	加速比
x	35.18	38.01	1.08
y	14.07	15.97	1.14
z	26.01	25.56	0.98

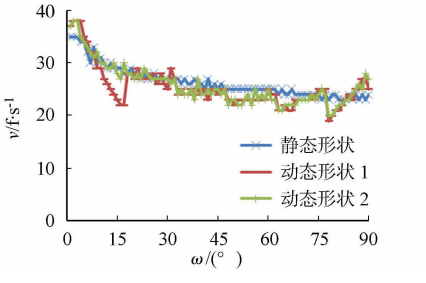


图 6　改进前和改进后的帧速率对比图

Fig. 6　Comparison chart of frame rate Before and after improvement

5　结束语

根据视点动态选择线程块，改善基于 GPU 的光线投射算法在访问纹理内存时的性能. 分析体数据保存为三维纹理后的数据布局及线程块形状和 warp 形状的关系. 在给定视点下，在旋转变换中确定与成像屏幕平行的平行平

面,进而确定主轴的方法指导线程块的几何形状选择,可以改善 warp 级投射光线在访问纹理缓存时的命中率失效的问题.实验结果表明:该方法能够改善光线投射算法的性能.下一步的工作将继续优化体数据绕 z-坐标轴旋转时的纹理缓存,继续考虑关于线程块 block 级的优化.

参考文献:

[1] KRUGER J,WESTERMANN R. Acceleration techniques for GPU-based volume rendering[C]// Proceedings of the 14th IEEE Visualization. Washington D C:IEEE Computer Society,2003:287-292.

[2] SALAMA C R,KELLER M,KOHLMANN P. High-level user interfaces for transfer function design with semantics [J]. Visualization and Computer Graphics,2006,12(5):1021-1028.

[3] GOBBETTI E,MARTON F,GUITIÁN J A I. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets[J]. Visual Computer,2008,24(7/8/9):797-806.

[4] 李国和,段忠祥,吴卫江,等.针对全空子数据体的 GPU 体绘制[J].中国图象图形学报,2014.19(4):577-582.

[5] OWENS J D,HOUSTON M,LUEBKE D,et al. GPU computing[J]. Proceedings of the IEEE,2008,96(5):879-899.

[6] MARŠÁLEK L,HAUBER A,SLUSALLEK P. High-speed volume ray casting with CUDA[C]// IEEE Symposium on Interactive Ray Tracing. Los Angeles:CA Press,2008:185.

[7] ZHANG Changgong,XI Ping,ZHANG Chaoxin. CUDA-based volume ray-casting using cubic B-spline[C]// International Conference on Virtual Reality and Visualization. Beijing:IEEE Press,2011:84-88.

[8] 甘新标,沈立,王志英.基于 CUDA 的并行全搜索运动估计算法[J].计算机辅助设计与图形学学报,2010,22(3):457-460.

[9] 赵丽丽,张盛兵,张萌,等.基于 CUDA 的高速 FFT 计算[J].计算机应用研究,2011,28(4):155-159.

[10] 肖江,胡柯良,邓元勇.基于 CUDA 的矩阵乘法和 FFT 性能测试[J].计算机工程,2009,35(10):7-10.

[11] 王蓓蕾,朱志良,孟琭.基于 CUDA 加速的 SIFT 特征提取[J].东北大学学报(自然科学版),2013,34(2):200-204.

[12] JENKINS J,ARKAKAR I,OWENS J D,et al. Lessons learned from exploring the backtracking paradigm on the GPU[J]. Lecture Notes in Computer Science,2011,6853(2):425-437.

[13] 周洪,樊晓桠,赵丽丽.基于 CUDA 的稀疏矩阵与矢量乘法的优化[J].计算机测量与控制,2010,18(8):1906-1908.

[14] SANDERS J,KANDROT E. CUDA by example:an introduction to general-purpose GPU programming[M]. Boston:Addison-Wesley Professional,2010:116-117.

[15] MONTRYM J,MORETON H. The geforce 6800[J]. IEEE Micro,2005(2):41-51.

[16] MORTON G M. A computer oriented geodetic data base and a new technique in file sequencing[M]. New York:International Business Machines Company,1966:56-60.

(**责任编辑:** 陈志贤 **英文审校:** 吴逢铁)