中图法分类号: TP391 **文献标志码**: A **文章编号**: 1006-8961(2011)07-1269-07
论文索引信息:袁斌. 改进的均匀数据场 GPU 光线投射 [J]. 中国图象图形学报,2011,16(7):1269-1275

改进的均匀数据场 GPU 光线投射

袁斌
(北京应用物理与计算数学研究所,北京 100088)

摘 要: 针对均匀数据场 GPU 光线投射技术中梯度、代理面和预积分计算存在的一些问题和多块绘制中可能存在缝隙的问题,设计了改进的 GPU 光线投射方法。主要改进有按需在物理空间实时计算梯度,估计梯度上界;单趟 GPU 预积分;设计体裁剪方法、缝隙消除方法。实验结果表明,改进的 GPU 光线投射技术便于采用梯度量调制,以突出感兴趣的物质; 可以交互修改转换函数,快速发现感兴趣的特征; 可以避免绘制错误;还可以逐步剥离前面的物体,看到后面的物体;可以对数据进行漫游; 绘制多块数据时没有缝隙。

关键词: GPU; 光线投射算法; 单趟 GPU 预积分; 缝隙

Improved GPU ray-casting for uniform grid

Yuan Bin
(*Institute of Applied Physics and Computational Mathematics*, *Beijing* 100088 *China*)

Abstract: Some problems exist in computing gradient, proxy polygon and pre-integration in GPU ray-casting for uniform grid and cracks may be easily produced in multi-block GPU ray-casting. To address these issues, improved GPU ray-casting for uniform grid is designed and implemented in this paper. Main improvements include an improved gradient computing method where that gradient is computed on need and on physical space and upper limit of gradient magnitude is evaluated, single pass GPU pre-integration, volume clipping method that is utilized to compute proxy polygon correctly and crack-free method for multi-block GPU ray-casting. Improved GPU ray-casting in this paper facilitates gradient magnitude modulation so that the interesting substances can be highlighted. It can compute pre-integration rapidly so that one can modify transfer function interactively and interesting features can be found quickly. Volume can be rendered correctly with it and one can peel the front thing and see the internal things. Crack is free when it renders multi-block dataset.

Keywords: GPU; ray-casting; single pass GPU pre-integration; cracks

0 引 言

在医学、计算物理等领域会产生大规模的体数据集,需要合适的可视化技术。体绘制技术是重要的可视化技术之一。对于均匀数据场有光线投射方法[1]、shear-warp 方法[2]。

可编程的图形处理器 GPU 是特殊的 SIMD 硬件,提供了顶点和片元编程技术[3]。GPU 光线投射方法可以看做光栅化和光线投射结合的方法。在 GPU 中, 首先要绘制三角面片,才能执行相应的片元处理。基于 GPU 片元编程的算法是一种 SIMD 算法。在一般的图形卡上,采用多条绘制流水线,可以对顶点(vertex)计算和片元(fragment)计算进行并行处理。采用 GPU 可以大大加速体绘制,如基于切片的体绘制方法[4-6]和 GPU 光线投射方法[7-10]。在早期的图形卡中,GPU 的光线投射方法要比基于切片的体绘制方法慢[7]。但是,光线投射把对体的计算转换成线的计算,比基于切片的计算更简捷,通过光线早终止等加速技术,光线投射比基于切片的技

收稿日期:2010-04-06;**修回日期**:2010-08-10
第一作者简介:袁斌(1966—),男,高级工程师,硕士。主要研究方向为可视化算法与软件。E-mail:yuanbin@iapcm.ac.cn。

术计算量更少。在 Nvidia Quadro FX 5600 图形卡上,GPU 光线投射比基于切片的体绘制方法快。

在装有单个 GPU 的机器上进行可视化是最普遍和最廉价的方式,在这样的机器上对大规模数据场进行可视化,由于 GPU 内存的限制,只能将数据分块动态装入 GPU,进行绘制,问题是速度较慢。在装有多个 GPU 的机器上进行绘制需要将数据分块,分别装入多个 GPU 进行绘制,然后把绘制结果进行合成。对于 GPU 机群,也需要将数据分块,进行分布绘制。对多块数据进行光线投射会产生缝隙。目前尚没有文献针对 GPU 光线投射,给出具体的消除缝隙的方法。本文设计实现消除多块 GPU 光线投射缝隙的方法。

梯度计算会花费很大的计算时间,存储梯度量会花费很大的内存,采用实时计算梯度的方法[11],大大节省纹理内存,适合于大规模数据场,是很好的想法,但该方法没有注意到对于实验测量数据和科学计算数据,各方向的网格间隔往往不同。计算梯度和光照也要花费一定的计算时间,在光线投射过程中,当采样点的不透明度为 0 时,计算梯度和光照是无意义的。本文实现按需实时计算梯度和光照,给出在线性插值下,更明确的在物理空间通过 6 邻点近似计算梯度的算法,并给出不同邻点距离的绘制结果。梯度量调制可以突出多种物质的分界面。由于查找梯度量所对应的调制因子通过纹理映射实现,需要把梯度量转换成[0,1]中的数,估计梯度量的上界很重要。本文估计了梯度量的上界。

采用预积分可以提高光线投射的精度。在分析数据场的结构特征时,需要交互修改转换函数来突出某些特征。在每次修改转换函数后,都要重新计算预积分,如果用 CPU 的方法计算转换函数,在计算预积分时需要用到指数函数,会花费大量的时间,不便于交互绘制。采用 GPU 计算预积分会大大节省计算时间。目前,尚没有人采用单趟 GPU 方法计算预积分。本文实现了这一方法。

在 GPU 光线投射中,需要绘制体的表面(代理面)来激活光线。但当视点位于体内时,就无法正确绘制代理面,从而无法正确进行 GPU 光线投射。本文给出用平行于视平面的近裁剪面裁剪体数据的方法,并利用该方法正确计算代理面。

Stegmaier 等人给出单趟 GPU 光线投射方法[7]。该方法预先计算梯度并保存起来,这样必然占用大量的内存。Klein 等人给出结合预积分的 GPU 光线投射方法[8],通过绘制体的前表面得到光线的起点,起点减去相机的位置(在纹理空间)得到光线方向。Scharsach 等人[10]根据体包围盒把顶点的坐标变换成颜色,并设置为顶点颜色。绘制前面和后面到不同的缓冲区,后面的图像减前面的图像,得到光线方向。根据这个方向进行光线积分。由于光线方向由后面的图像减前面的图像得到,不具备整体一致性,不便于实现多块数据光线投射的光线拼接。

郑杰等人在纹理切片体绘制方法中,采用实时计算梯度的方法[11],大大节省纹理内存,适合于大规模数据场。Engel 等人[5]对均匀网格在基于切片的体绘制中采用预积分技术提高绘制精度。Roettger 等人在无结构网格体绘制中采用 GPU 计算预积分[12],需要绘制多个长方形(多趟绘制)。

在数据并行的光线投射算法实现中,合成图像中很容易产生缝隙。这是由于光线没有很好拼接。Ma 等人在文献[13]中,采用 k-d 树分割数据,并采用二分交换法合成图像,虽然指出拼接处的走样现象,并采用 1 层虚网格解决这个问题,但采用 1 层虚网格无法保证共享采样点上梯度的一致性。在文献[14]中,作者给出较为详细的 CPU 光线投射消除缝隙的方法。

1 改进技术

用 ARB fp1 汇编语言[3]实现 GPU 光线投射算法,为了采用分支命令和循环命令,在程序开头加上 OPTION NV_fragment_program2。采取光线早终止技术,当累积不透明度接近于 1 时,停止光线投射。针对现有 GPU 光线投射方法存在的若干问题,提出如下的改进技术。

1.1 体裁剪算法

在 GPU 光线投射中,需要绘制体的表面(代理面)来激活光线。但当视点位于体内,采用 GPU 光线投射就无法正确绘制。用平行于视平面的裁剪面(可称为近裁剪面)裁剪掉一部分,使余下部分都位于视平面前面。如图 1 所示,把近裁剪面变换到模型空间;先用近裁剪面裁剪数据体表面的每个面,并计算所得各顶点的纹理坐标,记录裁剪后的面的顶点和新增边的信息。新增边为近裁剪面与每个相交侧面的交线段,它们围成一个多边形。这样可以得到裁剪后的体的各个表面,就是 GPU 光线投射所使用的代理面。经过裁剪的体的每个可见表面称为前

表面,每个不可见的表面称为后表面。

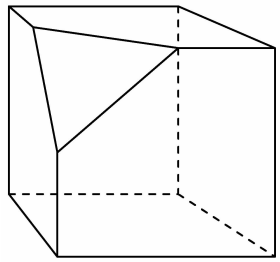


图 1　用近裁剪面裁剪体数据

Fig. 1　Clipping volume with near clipping plane

交互调整近裁剪面可以揭示数据场的内部信息。利用体裁剪技术可以对体数据进行漫游。近裁剪面相对于相机的位置是固定的。当推进相机时,近裁剪面随之改变;当相机位于数据体内时,近裁剪面与数据体相交。

1.2　改进的 GPU 实时梯度计算

在 GPU 光线投射算法中,需要把数据作为纹理数据装入 GPU 内存。模型空间网格通过线性变换 $L = N_1SN$ 变换到纹理空间:S 为各向异性的比例变换;比例因子分别为 s_x, s_y, s_z;N_1, N 为平移变换。在光线积分过程中需要用 S 对采样步长进行修正。

梯度一般应该在物理空间(模型空间)计算,这样才能正确反映物理现实。在物理空间,沿每个坐标轴方向,在采样点前后各取一邻点,使它们到采样点的距离相等,共取 6 点,通过插值方法计算这 6 点的标量值,进而计算中心差分,可称为以采样点为中心的 6 邻点方法。邻点到采样点的距离称为邻点距离,一般为相应的网格间隔乘以一个比率 r。这个比率 r 称为相对邻点距离。纹理映射指令直接使用的坐标是基本纹理坐标,它所在空间称为基本纹理空间。

设在物理空间,邻点距离分别为 dx, dy, dz,一般取为网格间隔。在纹理空间,要通过各向异性的比例变换 S 对 dx, dy, dz 进行修正。则在纹理空间,对于每个采样点 $P(x,y,z)$,设其标量值为 f;将 P 变换到物理空间得到 P_1,计算 P_1 的邻点,并把它们变换回基本纹理空间,则得到采样点 P 的邻点 $(x - s_x dx, y, z)$, $(x + s_x dx, y, z)$, $(x, y - s_y dy, z)$, $(x, y + s_y dy, z)$, $(x, y, z - s_z dz)$, $(x, y, z + s_z dz)$。通过纹理映射依次得到邻点的标量值 $f_{00}, f_{01}, f_{10}, f_{11}, f_{20}, f_{21}$。由此可以在物理空间计算梯度

$$\nabla f \approx \left(\frac{f_{01} - f_{00}}{2dx}, \frac{f_{11} - f_{10}}{2dy}, \frac{f_{21} - f_{20}}{2dz}\right)^{\mathrm{T}}$$

在 GPU 中,颜色数据的每个分量均用[0,1]中的数表示,纹理空间中每个方向的坐标也是用[0,1]中的数表示。当 f_{01}, f_{11}, f_{21} 为 1,f_{00}, f_{10}, f_{20} 为 0 时,梯度的模为 $l = \frac{1}{2}\sqrt{\frac{1}{dx^2} + \frac{1}{dy^2} + \frac{1}{dz^2}}$,显然这是最大模。

通过梯度量调制可以突出感兴趣的物质。通过交互修改梯度量到调制因子的转换函数,可以突出梯度量大的区域,忽略梯度量小的区域。

在 GPU 光线投射中通过 1 维纹理映射实现梯度量调制,1 维纹理空间为[0,1]。如果网格间隔很小,梯度量将是很大的数,很可能大部分梯度量都大于 1,则无论怎样调整转换函数都不能突出感兴趣的物质;如果间隔很大,则梯度量很小,大部分数据都分布在很小的范围内,这样很难通过调整转换函数来突出感兴趣的物质;这两种情况都会严重影响梯度量调制的效果。梯度量应该是[0,1]间的小数,梯度的每个分量应该除以梯度最大模 l。梯度修正为 $\left(\frac{f_{01} - f_{00}}{2l \times dx}, \frac{f_{11} - f_{10}}{2l \times dy}, \frac{f_{21} - f_{20}}{2l \times dz}\right)^{\mathrm{T}}$。

较大的邻点距离可以起到平滑图像的作用。较小的邻点距离可以突出梯度的变化特征,使得我们很容易发现梯度方向变化较大的区域。

为了便于比对,还实现了预计算梯度的方法,在网格节点上计算中心差分,计算时全部采用节点上的标量值。在光线投射过程中采样点的梯度通过线性插值获得(纹理硬件可以实现线性插值)。这样可以获得连续的梯度,从而光照效果也比较光顺。

在光线投射过程中,当采样点的不透明度非零时,才计算梯度和光照。光线积分在纹理空间进行。

1.3　单趟 GPU 预积分

在不采用预积分技术时,由于光线积分精度不够,后面的不该暴露的颜色暴露出来,本文称为颜色暴露现象。这是一种走样现象。采用预积分技术,可以提高绘制精度,消除颜色暴露现象。

在每次修改转换函数后,都要重新计算预积分。在计算预积分时需要用到指数函数,如果用 CPU 的方法计算,会花费大量的时间。利用 GPU 计算预积分表可以大大加快计算速度,在 GPU 中有计算指数函数的指令,可以快速计算指数函数。通过这种方法实现转换函数的交互调整。

由于采样点是均匀的,只要计算固定长度的光线段上的预积分。预积分表是一个 2D 图像,用 FBO(帧缓存对象)实现。

可以在 FBO 中绘制恰好覆盖 256×256 像素的正方形,片元的 x 坐标对应于光线段起点的标量值,y 坐标对应于光线段终点的标量值,在片元程序中,根据转换函数(用 1D 纹理表示)计算光线段的积分。在预积分计算中,采用自适应的数值积分方法,如果 $|x-y|>1$,把光线段分割成 $|x-y|$ 步;否则,不分割。设转换函数预设的采样距离为 d_1,光线投射的实际采样距离为 d_2,$factor = d_1/d_2$。在计算预积分时,由于步长改变,需要用 $\alpha' = 1-(1-\alpha)^{\frac{factor}{|x-y|}}$ 修正不透明度,其中 α 根据当前标量值由转换函数决定。不透明度的修正用 GPU 的计算指数函数的指令实现,这样可以快速计算预积分。

完成预积分表的计算后,把它连接到 2D 纹理,在光线积分过程中,通过纹理映射查预积分表。

1.4 数据分块与缝隙消除

由于 GPU 内存的限制,不可能把大规模数据完全装入,需要将数据分块。先尽可能把全部数据块保存到内存,然后分块装入 GPU 内存进行绘制,并进行图像合成。如果不能全部装入内存,则一边从硬盘读入数据块一边绘制。对多块数据,如果简单地用单块光线投射绘制每块数据,并进行图像合成,会产生缝隙。文献[14]给出并行 CPU 光线投射体绘制消除缝隙的方法。给出多块 GPU 光线投射体绘制的缝隙消除方法。本文采用 6 邻点方法计算采样点的梯度,相对邻点距离一般取为 1,如果采样点落在某一个单元内,则 6 个邻点分别落在该单元的 6 个相邻单元内,为了实现拼缝,至少需要两层虚网格。

对于 Nvidia Quadro FX5600 图形卡,3 维纹理的长、宽、高可以不是 2 的幂,因此,可以采用 k-d 树对数据进行分块,使每块的单元数尽可能相等。经过反复实验发现,s 方向纹理的尺寸必须是 4 的倍数。因此必须调整分割后每个数据块的尺寸,调整后的虚网格数可能大于 2,靠近实际空间的两层虚网格有用,更外层的虚网格没有意义。含有虚网格的体数据称为含虚体数据,原来的不含虚网格的体数据称为实际体数据。

GPU 光线投射采用固定步长的积分方法,步长选取为最小网格间隔的一半。要实现拼缝,需要每条光线在不同块中的共享采样点具有相同的全局纹理坐标。光线落在局部实际体数据内的线段,称为局部光线段。d 为全局纹理空间的采样步长,设 v 为光线的方向(注:为了保证各块数据在重合区域光线方向的一致性,计算光线方向时,用整体数据体前面与光线的交点,减去相机的位置,并规范化为单位向量),P_0 为光线与整体数据体前表面的交点,P_f,P_b 分别为光线与当前局部实际体数据的前表面、后表面的交点

$n_f = \left\lfloor \frac{v\cdot(P_f - P_0)}{d} \right\rfloor$,$n_b = \left\lfloor \frac{v\cdot(P_b - P_0)}{d} \right\rfloor$,则积分步数 $n = n_f - n_b$,初始采样点为 $P_0 + n_f v$。

对于多块数据在统一的全局纹理空间进行光线积分。这里把网格节点的索引 (i,j,k) 称为节点的结构坐标。全局纹理坐标为顶点的整体结构坐标加 0.5。整体光线段的起始点在整体数据体的前表面上。

用片元程序实现多块无缝 GPU 光线投射的基本步骤是:

```
装入必要的片元程序;
用近裁剪面裁剪整体数据体;
连接写纹理坐标的片元程序,
绘制裁剪后的整体数据体的前表面
(把相应的纹理坐标保存到帧缓存对象
FBO1 中,这些坐标就是在全局纹理空
间中前表面与每条光线的交点坐标);
基于 k-d 树对数据块进行排序;
(下面按照排好的顺序处理每块数据)
对每块数据{
    装入含虚体数据;
    用近裁剪面裁剪实际体数据;
    连接写纹理坐标的片元程序,
    绘制局部实际体数据的后表面
    (把每个片元的纹理坐标保存到 FBO2 中,
    每个坐标即局部光线段的终点坐标);
    连接 GPU 光线投射片元程序;
    设置相机在全局纹理坐标系中的位置、
    局部原点、比例因子和其他一些参数;
    绘制局部实数据体的前表面
    (每个片元的纹理坐标即局部光线段
    的起始坐标);
}
```

在光线积分片元程序中,计算光线方向 v,计算在全局纹理空间的采样步长 d,n_f,n_b,积分步数 n,初始采样点。对每个采样点,把整体纹理坐标变换到局部的基本纹理坐标(减去一个局部原点,再做

比例变换),然后去查相应的标量值和邻点标量值,进一步完成梯度和光照计算、梯度量调制和颜色累积。这里,梯度和光照计算采用按需计算的方法。

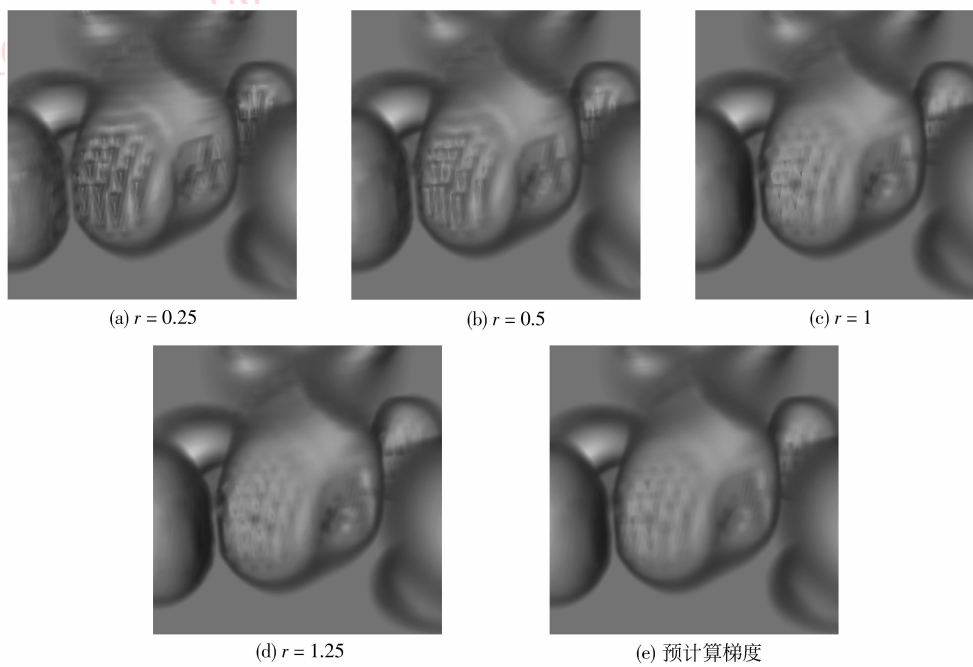
如果动态绘制大规模时变数据场,需要一边读入数据一边绘制,可以用上述方法分割数据并保存在磁盘中。首先基于 k-d 树将数据排序,在绘制时动态导入,但这样绘制速度较慢。

2　实　验

实现了改进的 GPU 光线投射方法,其中含有体裁剪方法、改进的 GPU 梯度计算方法、单趟 GPU 预积分技术及数据分割和拼缝技术。为了便于测试,通过各种技术的组合形成多个算法程序,主要有单块 GPU 光线投射和多块 GPU 光线投射(包含拼缝技术)。

在含有两个 4 核 CPU(Intel Xeon x5355 266 GHz),16 GB 内存的计算机(装有 Nvidia Quadro FX 5600 图形卡,显存 1.5 GB)上测试本文算法。在测试中,为了保证绘制质量,采样步长均为最小网格间隔的一半。图像尺寸均为 512 × 512。

图 2 是本文方法对 ironProt 数据 68 × 68 × 68 的绘制结果。这里给出邻点距离对绘制结果的影响,r 为相对邻点距离。从图中可以看到,较大的邻点距离起到光滑作用。但较小的邻点距离,可以突出梯度方向的变化,在可视化中有一定的意义。在交互操作时,当视点离数据较近,进行旋转和放大操作,用预计算梯度方法图像抖动不大,而以采样点为中心的 6 邻点方法,当 r 较小时会产生抖动,例如,r = 1 就有较大抖动,而 r = 1.25 时抖动不大。对于较密集的数据场,r = 1 时按需计算梯度是可以接受的。



(a) r = 0.25　(b) r = 0.5　(c) r = 1

(d) r = 1.25　(e) 预计算梯度

图 2　梯度计算

Fig. 2　Gradient computation

以下测试,采用 6 邻点方法计算梯度,选取邻点距离为一个网格间隔(相对邻点距离为 1)。

图 3 是本文方法对 head 数据 256 × 256 × 94 的绘制结果。通过交互调整梯度量到调制因子的转换函数,可以得到如图 3 所示结果。图 3 同时显示出骨骼和皮肤的特征。

图 4 是对 XMas_Tree 数据(512 × 512 × 999)的 GPU 体绘制结果。数据读入时间 0.875 s,纹理装入时间 0.328 s。其中,(a)采用 GPU 光线投射,按需计算梯度和光照,未采用预积分,绘制时间 0.93 s,有颜色暴露现象。(b)采用 GPU 光线投射,并采用预积分技术,消除了颜色暴露现象。采用 CPU 计算预积分,计算时间为 0.171 s;采用 GPU 计算预积分,计算时间为 0.0 s,这样可以交互修改转换函数,一边修改一边观察绘制结果,可以发现更多感兴趣的信息(在交互修改转换函数时,GPU 预积分计算

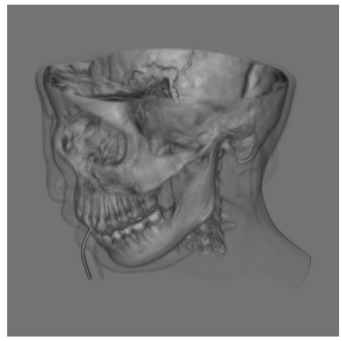
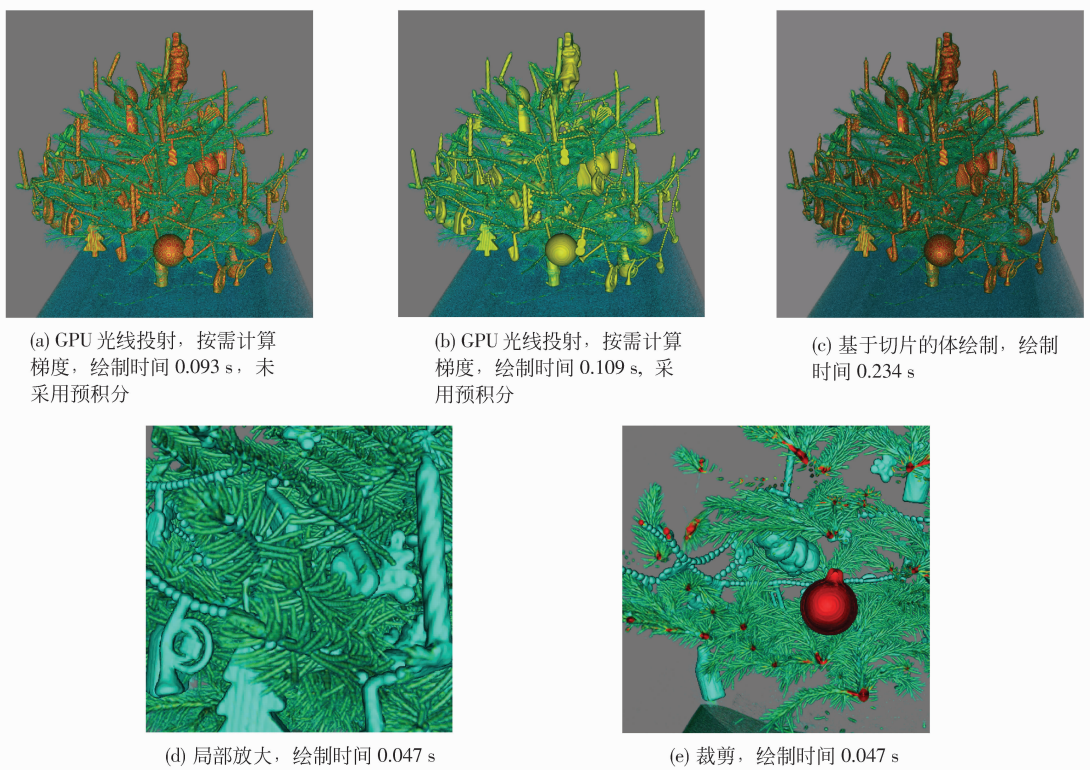


图 3 采用梯度量调制突出骨骼和皮肤

Fig. 3 Showing skelecton and skin with gradient magnitude modulation

的时间有时为 0 s,有时为 0.016 s 左右的值)。调整转换函数后,数据空间有较多空白区域,在光线投射中按需计算梯度与光照,提高了绘制速度,与实时计算梯度方法相比,加速比为 1.86。(c)是基于切片的体绘制技术,绘制时间 0.234 s,比本文实现的 GPU 光线投射速度慢。说明本文实现的 GPU 光线投射适合于较大数据集的绘制,因为它考虑了光线早终止技术和按需绘制技术。(d)是局部放大图。(e)是裁剪图,通过近裁剪面裁剪,剥离前面的物体,露出后面的小狗。

所用图形卡内存比较大,可以把 head 数据全部装入。为了测试本文的数据分割和拼缝方法,把 head 数据分成 6 块。图 5 是对 head 数据的体绘制结果。其中:(a)对每一块采用单块光线投射方法,逐块绘制并合成图像,图中含有缝隙;(b)采用本文的拼缝算法消除缝隙。扣除数据装入时间,两种方法的绘制时间基本相当。但多块 GPU 光线投射,数据采用两重或两重以上的虚网格,数据量增大,数据装入时间稍长一些。



(a) GPU 光线投射,按需计算梯度,绘制时间 0.093 s,未采用预积分

(b) GPU 光线投射,按需计算梯度,绘制时间 0.109 s,采用预积分

(c) 基于切片的体绘制,绘制时间 0.234 s

(d) 局部放大,绘制时间 0.047 s

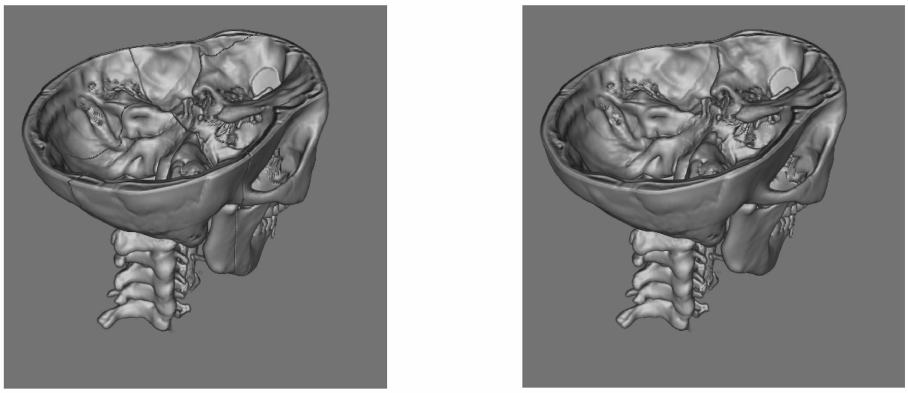
(e) 裁剪,绘制时间 0.047 s

图 4 对 XMas_Tree(512×512×999)数据的 GPU 光线投射体绘制

Fig. 4 GPU raycasting for XMas_tree(512×512×999)

3 结 论

多块数据 GPU 光线投射可以在单个 PC 上处理较大规模的体数据,很好地消除了多块数据之间的缝隙和走样现象。但由于每次绘制都要动态装入纹理,需要有一定的时间开销。拼缝方法可以应用于多 GPU 光线投射和可视化集群的并行 GPU 光线投射,这样可以更好地处理大规模数据场。梯度计算方法,可以更正确地在 GPU 上计算梯度:估计梯度



(a) 采用单块 GPU 光线投射，数据装入时间 0.032 s绘制时间 0.109 s

(b) 多块 GPU 光线投射，数据装入时间 0.047 s，绘制时间 0.125 s

图 5　多块数据 GPU 光线投射(head 数据 256 ×256 ×94)

Fig. 5　Multi-block GPU ray-casting(head 256 ×256 ×94)

量的上界便于采用梯度量调制,以突出感兴趣的物质;按需计算梯度和光照,跳过透明采样点,提高了光线投射的速度。采用单趟 GPU 预积分可以提高积分精度,并能交互修改转换函数。而采用平行于视平面的裁剪面对体数据进行裁剪,可以避免当视点位于数据体内时的绘制错误,交互地剥离前面的物体,看到后面的物体;采用此体体裁剪技术可以对体数据进行漫游。

参考文献(References)

[1] Marc Levoy. Efficient ray tracing for volume data[J]. ACM Transactions on Graphics, 1990,9(3): 245-261.

[2] Philippe Lacroute, Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing trans-formation[C]// Proceedings of the 21st Annual Con-Ference on Computer Graphics and Interactive Techniques. New York, USA: ACM Press, 1994:451-458.

[3] NVIDIA. NVIDIA OpenGL Extension Specifications[EB/OL]. [2007-06-20]. http://developer. nvidia. com/object/nvidia _ opengl_specs. html.

[4] Brian Cabral, Nancy Cam, Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware[C]//Proceedings of the 1994 Symposium on Volume visualization. New York, NY,USA: ACM Press, 1994:91-98.

[5] Klaus Engel, Martin Kraus, Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading [C]//Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Workshop on Graphics Hardware. NewYork, NY, USA: ACM Press,2001: 9-16.

[6] Tong Xin , Tang Zesheng. 3D texture hardware assisted volume rendering with space leaping [J]. Chinese Journal of Computers, 1998, 21 9: 807-812.[童欣,唐泽圣. 基于空间跳跃的三维纹理硬件体绘制算法[J]. 计算机学报,1998,21(9):807-812.]

[7] Stegmaier S, Strengert M, Klein T,et al. A simple and flexible volume rendering framework for graphics-hardware-based raycasting[C]//Proceedings of the International Workshop on Volume Graphics'05. Los Alamitos, CA, USA: IEEE CS Press, 2005:187-195.

[8] Klein T, Strengert M, Stegmaier S, et al. Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware[C]//Proceedings of IEEE Visualization '05. Los Alamitos, CA, USA: IEEE CS Press, 2005:223-230.

[9] Chu Jingjun, Yang Xin, Gao Yan. Ray-casting based volume rendering algorithm using GPU programming [J]. Journal of Computer-Aided Design & Computer Graphics 2007, 19(2): 257-262.[储璟骏,杨新,高艳. 使用 GPU 编程的光线投射体绘制算法[J]. 计算机辅助设计与图形学学报,2007,19(2): 257-262.]

[10] Scharsach H. Advanced GPU raycasting[EB/OL]. [2005-05-20]. http://www. cescg. org/ proceedings. html #cescg.

[11] Zheng Jie, Ji Hongbing. Interactive PC texture-based volume rendering for large datasets[J]. Journal of Image and Graphics, 2008, 13(2): 316-321.[郑杰,姬红兵. 基于动态纹理载入的大规模数据场体绘制[J]. 中国图象图形学报, 2008,13(2): 316-321.]

[12] Roettger S, Ertl Th. A two-step approach for interactive pre-integrated volume rendering of unstructured grids[C]// IEEE Symposium on Volume Visualization'02. Piscataway, NJ, USA: IEEE Press,2002:23-28.

[13] Ma K L, Painter J S, Hansen C D, et al. Parallel volume rendering using binary-swap compositing[J]. IEEE Computer Graphics and Applications, 1994,14(4):59-68.

[14] Yuan Bin. Design and implementation of the algorithm for parallel raycasting without gap[J]. Journal of Computer Research and Development, 2008,45(supl2):157-162. [袁斌. 无缝并行光线投射算法的设计实现[J]. 计算机研究与发展,2008,45(增刊 2):157-162.]