doi:10.11887/j.cn.201604011 http://journal.nudt.edu.cn

单硬件限制下的电磁环境加速绘制*

冯晓萌[1],吴玲达[1,2],于荣欢[1],杨 超[1]
(1. 装备学院 复杂电子系统仿真实验室,北京 101416;
2. 国防科技大学 信息系统工程重点实验室,湖南 长沙 410073)

摘 要:单硬件实现的高效电磁环境绘制适用范围更广;但是,并行光线投射绘制电磁环境时,其效率受硬件性能制约。在研究硬件限制并行光线投射效率的基础上,提出一种面向硬件制约的像素插值方法。当硬件限制并行光线投射绘制不能实时完成时,减少并行投射的光线数量,即部分图像像素由光线投射生成,其余像素插值生成。像素插值以图像质量换取执行效率,当图像更新停顿时重新使用光线投射生成插值获得的像素,以恢复图像内容。实验结果表明,低硬件配置条件下,像素插值能够大幅度提高绘制图像的生成效率。同时,对比多个体数据的绘制效果和误差统计得出:电磁环境数据场最适合使用像素插值方法。

关键词:电磁环境;光线投射;硬件限制;像素插值;统一计算设备架构
中图分类号:TP391.9 **文献标志码**:A **文章编号**:1001-2486(2016)04-069-07

Accelerated rendering for electromagnetic environment under single device restriction

FENG Xiaomeng[1], *WU Lingda*[1,2], *YU Ronghuan*[1], *YANG Chao*[1]
(1. Science and Technology on Complex Electronic System Simulation Laboratory, Equipment Academy, Beijing 101416, China;
2. Key Laboratory of Information System Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: Electromagnetic environment with a high efficiency based on single device supports has wide range of applications. But the efficiency of parallel ray-casting for rendering electromagnetic environment is restricted by the device. Based on researching the restriction of device for parallel ray-casting, a pixel interpolation method focusing on the restriction was presented. The number of rays was reduced when the parallel ray-casting rendering under device restriction couldn't be completed immediately, namely, a part of pixels in the rendering image were generated by ray-casting and the rest pixels through interpolation. Pixel interpolation got rendering efficiency at the cost of image quality, so when image update paused, the interpolated pixels were regenerated to recover image quality. The experiments show that pixel interpolation obviously improves rendering efficiency when implemented on a low device. Compared with the rendering images of some volume data and the errors in these images, the electromagnetic environment data has the best rendering result, which proves that pixels interpolation is useable especially for rendering electromagnetic environment on a low device.

Key words: electromagnetic environment; ray-casting; device restriction; pixel interpolation; compute unified device architecture

电磁环境已经成为现代信息化战场中决定胜负的关键因素,对其进行可视化是虚拟战场环境的重要研究内容。在现有技术中,效果最好的电磁环境可视化方法是体绘制方法[1]。

体绘制方法将三维数据场直接映射成二维图像[2],以展示三维数据场中的相关信息,方便用户观察和理解数据场。体绘制方法生成的绘制结果展示了复杂电磁环境的内部细节[3],为用户观察和理解电磁环境态势提供了支持。光线投射算法[4]是绘制效果最好的体绘制算法。但是,其计算量大,达到实时绘制通常需要基于图形硬件GPU的并行实现[5]。统一计算设备架构(Compute Unified Device Architecture, CUDA)[6]出现后,GPU进行通用计算更加方便,使用其加速体绘制的算法[7]越来越普遍。

文献[5]总结了基于GPU的体绘制研究现状,体绘制算法在GPU中并行实现后加速效果明显,能够交互绘制大规模体数据。文献[3]中使用GPU对电磁环境体绘制进行加速,提高了绘制效率。文献[8]基于CUDA架构加速绘制动态电

* **收稿日期**:2015-04-28
基金项目:国家自然科学基金资助项目(61202129)
作者简介:冯晓萌(1986—),男,河北石家庄人,博士研究生,E-mail:130123feng@163.com;
吴玲达(通信作者),女,研究员,博士,博士生导师,E-mail:wld@nudt.edu.cn

磁环境,绘制效率相比文献[3]有所提升。

但是,CUDA 加速依赖于硬件的自身性能,当单个硬件限制体绘制执行效率时,使用多个硬件可以提高效率[9]。然而,单硬件环境比多硬件环境更容易获得,使用单硬件完成电磁环境绘制的应用范围将更为广泛,特别是针对指挥员的使用,低硬件配置更容易满足需求。因此,研究单硬件条件下的电磁环境加速绘制问题具有很强的实用意义。

1 光线投射的硬件限制

光线投射算法中光线相互独立,易于并行实现,从而提高其执行效率。但是,同一并行光线投射算法在不同型号的显卡上执行时间不同,如图1所示。

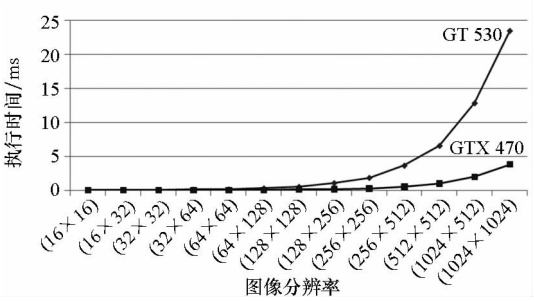


图1 并行光线投射效率

Fig. 1 Efficiency of parallel ray-casting

图1中,不仅硬件不同对应的执行时间不同,当图像分辨率超过一定规模后,两个硬件的执行时间均与分辨率成正比。而并行光线投射算法中,每个像素对应一条光线,说明并行投射光线越多,执行时间越长。因此,在硬件保持不变且无法修改光线投射算法时,若需要提高执行效率,可适当减少投射光线的数量。

2 像素插值

减少投射光线数量即减少光线投射生成的像素,此时能够提高算法执行效率,但是应该同时保持图像分辨率才有意义。因此,使用光线投射算法生成图像中的部分像素,其余像素使用比光线投射计算量小的方法生成。

2.1 像素插值可行性

光线投射算法是将三维数据场映射为二维图像效果最好的方法,因此,在提高算法执行效率的同时应该保证光线投射对图像生成的作用。图像中部分像素使用光线投射算法生成,然后利用这些像素插值生成其余的像素,通过插值间接获得光线投射的绘制效果。

在光线投射算法中,体数据为三维离散数据场,光线穿越数据场时,对数据的采样点并非全部命中数据点,通常需要使用采样点附近的多个数据点进行插值获取采样点值。说明,插值过程在光线投射算法中已有使用,插值得到的数据值转换为像素值的过程由传递函数[10]决定,可以看作是相邻的可插值数据经过传递函数变成了相邻的像素。因此,相邻像素间插值避开了传递函数,节省光线投射操作时间的同时引入了插值与传递函数不同所致的误差。

电磁环境的可视化技术中[3,11],其数据场由连续的电磁信号值离散得到,数据点与数据点之间相关且无突变,较适合进行插值。因此,电磁环境数据场相对一般体数据而言,其数据场内部较适宜插值,从而经过传递函数后得到的像素也会相对比较适宜插值。这一推论将在后面的实验结果中得到验证。

2.2 保持图像分辨率

为便于后续讨论,定义下述4个用语。

1)源像素:光线投射生成的像素。

2)插值像素:通过插值生成的像素。

3)行插值:某像素利用其同一行左右相邻的两个像素进行插值生成。

4)列插值:某像素利用其同一列上下相邻的两个像素进行插值生成。

对某像素,插值生成的值与光线投射生成的值之间存在差异。因此,像素插值影响生成图像的质量。为尽量保证图像质量,插值像素数量应该尽量少,以减少像素值的误差。

图2为源像素与插值像素数量比例为1∶1时在生成图像中的分布示意图,其中显示了两种分布情况:图2(a)至图2(d)为第一种;图2(e)和图2(f)为第二种。这两种分布中,插值像素与源像素的相对位置有规律性,利于并行程序的

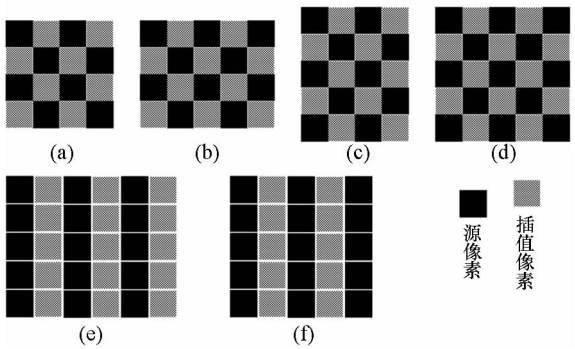


图2 源像素与插值像素分布

Fig. 2 Distribution of source and interpolation pixels

编写。

观察图 2 中的 6 幅子图:当插值像素位于图像内部时可以统一进行行插值;当插值像素位于图像边缘时,需要特殊考虑。据此可知,图 2 的第一种分布情况中,需要特殊考虑左右两个边缘的像素,并区分奇偶行;第二种分布情况则只需特殊考虑右侧边缘的像素,且不区分奇偶行。因此,图 2 中的第二种分布情况插值类型少,更易于实现并行编程。

2.3　提高插值像素比例

图 2 中,源像素与插值像素数量比为 1 ∶ 1,由图 1 可知,减少一半的投射光线数量可以节省不到一半的执行时间。当需要节省更多的执行时间时,可以继续减少投射光线的数量,即提高插值像素的比例。在图 2 的基础上继续减少一半的源像素后,像素分布如图 3 所示,源像素与插值像素的比例约为 1 ∶ 3。此时,相对图 2 所示的分布,在执行过程中光线投射时间减少、插值操作时间增加,总时间将会减少,从而进一步提高生成图像的效率。

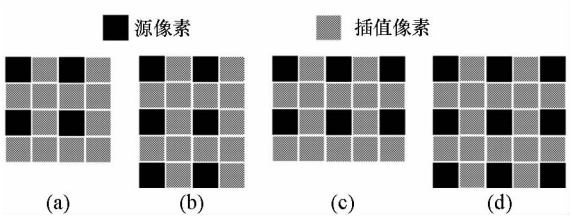


图 3　源像素与插值像素近似比例 1 ∶ 3

Fig. 3　Ratio of source pixel to interpolation pixel is 1 ∶ 3

图 3 中的插值像素可以分两步生成,首先生成图中源像素下方相邻的插值像素,将此时插值生成的像素看作源像素即与图 2 中的第二种分布相同,然后再利用图 2 中第二种分布的插值方法生成其余像素。在上述两个插值步骤中,除边缘像素需特殊考虑外,其他插值像素在第一步中进行列插值,在第二步中进行行插值,操作统一利于并行处理。

绘制电磁环境数据场时,图像边缘通常对应数据场的边缘,而数据场边缘的数据信息量很小,可以不精确显示。因此,绘制电磁环境数据场时,将位于图像边缘的插值像素设置为与其相邻的源像素值,以尽量节省执行时间。

图 3 所示的分布中,源像素所占的比例已经比较小,若再降低,将使误差继续扩大,严重影响生成图像的质量。因此,非特殊需要将不再继续提高插值像素比例。

3　恢复插值像素

像素插值在牺牲图像质量的前提下提高图像生成速度,以支持图像的高更新频率。但是,当图像更新停顿时,应使用光线投射重新生成插值像素,以恢复图像内容。

3.1　插值像素的恢复流程

从像素插值到恢复插值像素的流程如图 4 所示。像素插值是一个循环过程,保证实时更新图像,通过判断“是否空闲”决定继续循环还是跳出循环。跳出循环后,对插值生成的像素使用光线投射算法进行恢复。

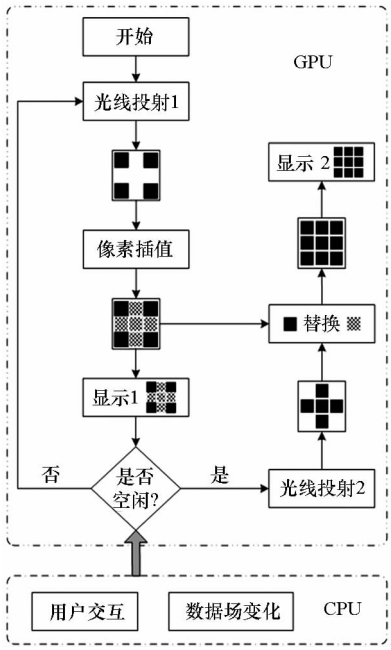


图 4　像素插值及恢复流程图

Fig. 4　Flow of pixel interpolation and recovery

图 4 中,上下两个虚线框分别表示设备端(GPU)和主机端(CPU)中的操作,由主机端确定“是否空闲”,“是”表示暂时不需要更新图像,“否”表示需要立即更新图像。当用户通过主机端对体绘制进行交互控制或者电磁环境数据场发生变化时,均需要重新生成并显示图像,此时采用像素插值的方法提高图像生成效率。空闲时,设备端恢复图像并显示。此时,若主机端需要继续更新图像,则从“开始”操作重新执行设备端的处理流程。图 4 中表示 GPU 操作的虚线框中明确标识了“开始”未标识“结束”,这是因为是否结束由 CPU 控制,可以在任何操作时结束。

3.2　是否空闲的判断

图 4 中,是否空闲的含义为是否需要更新图像,由主机端决定,其因素主要有用户交互、数据

场变化等。用户交互调整传递函数或者观察视角时,图像需要根据用户的调整及时更新,以反馈交互效果。当硬件限制而不能实时更新图像时,可采用像素插值的方法提高图像生成效率。然而,用户的交互操作会有停顿即“空闲”,此时进行像素恢复。同理,数据场变化有停顿时也进行像素恢复。使用时间阈值 ε 判断“空闲”状态:在 ε 时间内既没有用户交互操作,数据场又未变化时,认定此时为“空闲”。

“空闲”状态下恢复图像是为了消除像素插值造成的生成图像误差,恢复图像质量,便于用户更好地观察电磁环境的绘制结果。

4 具体实现

依据图4所示的流程进行编码实现,程序主要分为两个部分:主机端代码和设备端代码。主机端除负责将绘制电磁环境需要的数据和控制参数传递到设备端外,最主要的任务就是实时监控“空闲”状态,并将结果反馈给设备端。设备端使用CUDA架构并行实现,光线投射、像素插值、像素替换等操作均并行实现。

算法1中显示了主机端控制像素插值流程的关键代码,其中插值像素的分布如图3所示。其中,第3行获取图像分辨率;第4和第5行取图像分辨率的1/4;第6至8行设置开启的线程数,每个源像素对应1个线程;第9行进行光线投射生成源像素;第10行进行列插值;第11至13行更新开启的线程数,为图3中的行插值做准备;第14行进行行插值。

算法1 像素插值流程关键代码

Alg. 1 Key code of pixel interpolation pipeline

```
1.  int imageSize[2]; //图像大小
2.  dim3 blockSize, gridSize;//启动线程数量
3.  GetImageSize(imageSize);
4.  imageSize[0] = (imageSize[0] + 1) * 0.5;
5.  imageSize[1] = (imageSize[1] + 1) * 0.5;
6.  blockSize.x = blockSize.y = 16;
7.  gridSize.x = (imageSize[0] - 1) / blockSize.x + 1;
8.  gridSize.y = (imageSize[1] - 1) / blockSize.y + 1;
9.  g_CastRay <<< gridSize, blockSize >>> ();
10. g_InsertPix1 <<< gridSize, blockSize >>> ();
11. GetImageSize(imageSize);
12. imageSize[0] = (imageSize[0] + 1) * 0.5;
13. gridSize.y = (imageSize[1] - 1) / blockSize.y + 1;
14. g_InsertPix2 < < < gridSize, blockSize > > > ()
```

由于图3中行插值像素个数大约为列插值像素个数的2倍,因此在11至13行对线程个数进行了重置,表1中行插值函数 *g_InsertPix*2( )比列插值函数 *g_InsertPix*1( )开启的线程数多1倍。光线投射函数 *g_CastRay*( )的具体执行过程与文献[8]中相同。函数 *g_InsertPix*1( )的伪代码见算法2。

算法2 像素插值内核线程

Alg. 2 Kernel thread of pixel interpolation

```
1.  unsigned x = blockIdx.x * blockDim.x +
    threadIdx.x; //获取线程的横坐标
2.  unsigned y = blockIdx.y * blockDim.y +
    threadIdx.y; //获取线程的纵坐标
3.  x = x * 2; //转换为插值像素横坐标
4.  y = y * 2 + 1; //转换为插值像素纵坐标
5.  if Pixel(x,y) is in the image then
6.    if (y + 1) = = imageSize[1] then //图像边缘
7.      Pixel(x,y) = Pixel(x,y - 1);
8.    else
9.      Pixel(x,y) = [Pixel(x,y - 1) + Pixel(x,y +
      1)] * 0.5;
10.   end if
11. end if
```

算法2中,前四行根据线程的ID号确定其对应插值像素的坐标,当插值像素位于图像边缘时直接将相邻源像素的值赋予它(第7行),否则进行线性列插值(第9行)。函数 *g_InsertPix*2( )的执行代码与算法2中相似,插值时使用行插值。

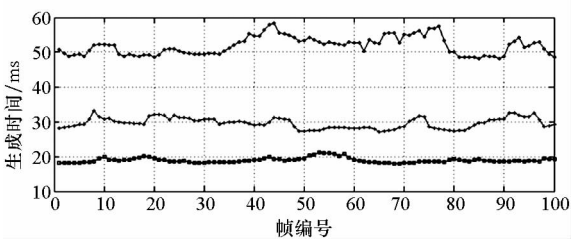
5 实验结果与分析

基于CUDA架构实现本算法,编程开发环境为集成了NVIDIA GPU Computing SDK 4.2的Microsoft Visual Studio 2005。硬件支持环境为:Intel(R) Core(TM) i5 - 2400 CPU 3.10 GHz、4 GB内存、NVIDIA GeForce GT 530显卡。由图1可知,此显卡并行计算能力相对较低,对并行光线投射算法的硬件限制较大,当其并行能力不能支持实时光线投射时,可以使用像素插值的方法进行加速。

5.1 执行效率

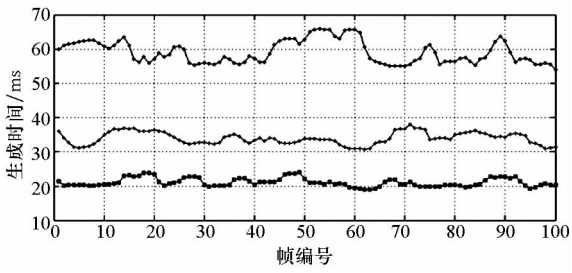
使用体绘制中的DVR(直接体绘制)算法和MIDA算法[12]进行了两组实验,生成图像分辨率为1200×900,连续生成100帧图像的时间统计如图5所示。每组实验分别统计了三种情况下的生成时间:①全部像素使用光线投射生成;②源像素与插值像素数量比例为1∶1;③源像素与插值像素数量比例为1∶3。图5的两幅图中,上中下

三条曲线分别对应第 1、第 2、第 3 种情况。



(a) DVR 生成图像时间

(a) Implementation time of DVR



(b) MIDA 生成图像时间

(b) Implementation time of MIDA

图 5　算法执行时间

Fig. 5　Time for algorithm implementation

通过图 5 两幅子图中三条曲线的具体时间可以看出,像素插值能够将图像的生成时间从40 ms 以上降到 40 ms 以下,即达到实时生成图像。说明,在由于硬件能力限制而不能实时生成图像的情况下,像素插值能够加速图像生成,使其能够实时完成。同时,将各子图中的三条曲线进行对比可以得出如下结论:生成图像的分辨率不变时,插值像素越多图像的生成时间越短。因此,适当提高插值像素在图像中所占的比例能够获得更好的加速效果。

图 5(a)中三条曲线的平均每帧时间为 51.84 ms,29.68 ms,18.96 ms,中下两条曲线的平均时间分别是最上面曲线的 57.25%,36.57%;上述数据值对应到图 5(b)中分别是 59.06 ms,33.79 ms,21.08 ms,57.21%,35.69%。对比可知,虽然 MIDA 算法比 DVR 算法用时要长,但是像素插值对两个算法的加速效率相差不多,甚至 MIDA 算法的加速效率高些。说明,像素插值对光线投射生成图像的加速效率比较稳定,可适用算法范围较广。

5.2　绘制效果与误差统计

记电磁环境数据场为 EME,使用文献[8]中的方法计算得到。实验中,使用 4 个体数据场(EME:200 × 200 × 100, Sphere:64 × 64 × 64, Engine:256 × 256 × 110, stagbeetle:277 × 277 × 164)生成分辨率为 1200 × 900 的图像时,4 个体数据对应的图像中人眼均无法辨别出使用了像素插值的图像。而将生成图像分辨率调整为与体数据场规模的最大两维相等后,生成图像中部分区域如图 6 所示。图 6 中,每行对应一个体数据,从上至下依次为 EME, Sphere, Engine, stagbeetle;3 列从左到右依次为全部光线投射生成、图 2 中第二种分布的像素插值生成、图 3 所示分布的像素插值生成。

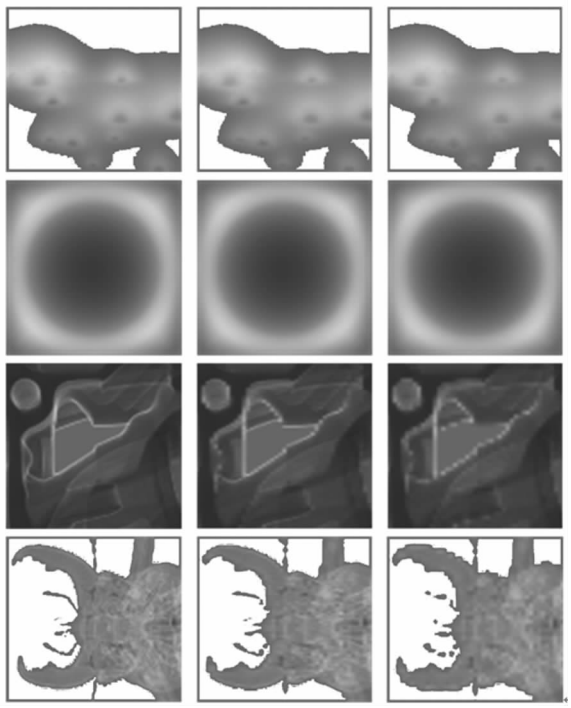


图 6　绘制效果对比

Fig. 6　Contrast of rendering results

仔细观察对比图 6 中每个体数据的不同绘制结果可知,EME 和 Sphere 的绘制结果几乎看不出差别,而 Engine 和 stagbeetle 的绘制结果则插值像素越多越模糊、效果越差。而将生成图像分辨率改为 1200 × 900 后,Engine 和 stagbeetle 的绘制结果也几乎看不出差别,说明小规模的数据场生成大分辨率的图像时,插值像素对绘制结果的影响减小。而当数据场规模与图像分辨率相当时,EME 和 Sphere 这种计算获得的数据场要比 CT 扫描获得的数据场(Engine 和 stagbeetle)更适合使用像素插值方法进行加速绘制。

从数据的角度对像素插值生成的图像中的误差进行统计分析:保持数据场规模不变,生成图像分辨率为 1200 × 900 时的统计结果见表 1;生成图像分辨率与体数据场规模的最大两个维度相等时的统计结果见表 2。其中,所有数据均使用相同的传递函数绘制生成。表 1 和表 2 中数据均为

10 帧图像数据的统计均值,“最大误差”列中数据只保留了整数。两个表的数据中“B”通道统计数值均为 0 是因为传递函数设置中未使用蓝色通道。

表 1 插值像素误差统计 1

Tab. 1 Statistic of pixel interpolation error 1

体数据名称	最大误差 R-G-B-A	平均误差 R-G-B-A	误差像素比/%
EME	5-3-0-6	0.50-0.61-0-0.62	4.8
	5-3-0-5	0.52-0.62-0-0.63	7.7
Sphere	3-3-0-3	0.67-0.80-0-0.67	4.8
	3-3-0-3	0.72-0.85-0-0.75	12.2
Engine	16-23-0-23	0.58-0.90-0-0.35	13.0
	20-24-0-24	0.71-1.04-0-0.43	23.4
stagbeetle	12-30-0-11	1.02-1.30-0-0.96	27.8
	28-41-0-26	1.53-1.87-0-1.44	45.7

表 2 插值像素误差统计 2

Tab. 2 Statistic of pixel interpolation error 2

体数据名称	最大误差 R-G-B-A	平均误差 R-G-B-A	误差像素比/%
EME	5-4-0-5	0.59-0.71-0-0.70	10.4
	5-6-0-5	0.60-0.73-0-0.70	17.1
Sphere	5-4-0-4	0.65-1.07-0-0.51	15.0
	5-4-0-4	0.75-1.16-0-0.55	25.1
Engine	66-68-0-70	2.61-3.29-0-1.93	31.6
	106-68-0-102	2.86-3.32-0-2.18	48.5
stagbeetle	51-53-0-56	6.05-5.72-0-6.03	34.1
	61-53-0-65	6.77-6.26-0-6.78	53.6

对于某个像素点,称其由插值生成的值与光线投射生成的值之间的差为误差,最大误差指图像中误差最大值。平均误差计算方法为所有误差的和除以存在误差的像素总数。误差像素比是存在误差的像素总数除以图像中的非零(4 个通道不全为零)像素总数。每个体数据均对应两行数据,上面一行为插值像素与源像素比例为 1∶1 的情况,下面一行为 3∶1 的情况。

分别对比表 1 和表 2 中每个数据对应的两行数据可知,插值像素个数较少时,误差情况相对较小。对比所有数据的“最大误差”列,前两个体数据明显优于后两个,而 EME 和 Sphere 均为数值计算得到的数据,后两者则是实物扫描数据,说明数值计算得到的数据场较适于使用像素插值的方法进行加速。“平均误差”与“误差像素比”两列的数据同样是 EME 的最好,说明电磁环境数据场更适合使用像素插值的方法进行绘制时的加速。

对比两个表中数据,印证了图像分辨率对像素插值结果的影响:数据场规模一定时,生成的图像分辨率越大,像素插值的误差越小。同时,表 2 与图 6 相对应,其中 EME 对应的数据说明当使用更大规模电磁环境数据场生成高分辨率的图像时,仍然可以使用像素插值的方法加速体绘制。而其他扫描得到的实体数据则将不适用像素插值方法。

5.3 像素恢复

表 1、表 2 显示绘制电磁环境时使用像素插值方法存在误差,为保证图像质量恢复插值像素是有必要的。实验中,根据插值像素分布设置空闲判断阈值 ε:源像素与插值像素为 1∶1 时设置为使用像素插值时生成第一帧图像的执行时间;1∶3 时设置为生成第一帧图像执行时间的 1.5 倍。如此,基本保证阈值 ε 大约为不进行像素插值的一半,当已空闲 1 个 ε 时使用 ε 或 1.33ε 的时间进行像素恢复。如此设置阈值 ε 在实验中未影响用户的交互操作,是可行的。

上述实验和分析说明,像素插值方法适用于绘制电磁环境数据场,不仅绘制误差小,同时还能够在绘制效率受硬件性能制约时进行加速绘制,从而得到更高的绘制效率。因此,本算法能够应用于文献[3]和文献[8]中,进一步加速其中的电磁环境绘制效率。同时,若接受一定的绘制效果损失,可以将本算法应用于绘制其他体数据,继续加速文献[5]中提到的基于 GPU 的体绘制算法。

6 结论

提出一种适用于电磁环境数据场绘制的像素插值算法,其可以在硬件性能限制绘制效率时对绘制进行加速,获得更高的绘制效率为其实时交互和应用提供支持。实验结果表明:像素插值算法的加速效果明显,且生成图像的误差明显优于其他体数据。算法的不足是,牺牲生成图像的质量换取绘制效率的提升,对其应用具有一定的限制。提高插值像素的精度可以改善这一限制,是下一步的研究工作。

参考文献(References)

[1] 吴迎年, 张霖, 张利芳, 等. 电磁环境仿真与可视化研究综述[J]. 系统仿真学报, 2009, 21(20): 6332-6338.

WU Yingnian, ZHANG Lin, ZHANG Lifang, et al. Survey on electromagnetic environment simulation and visualization [J]. Journal of System Simulation, 2009, 21(20): 6332 - 6338. (in Chinese)

[2] Arie K, Klaus M. Overview of volume rendering [M]// Hansen C D, Johnson C R. The Visualization Handbook, USA: Academic Press, 2005.

[3] 杨超, 徐江斌, 吴玲达. 硬件加速的虚拟电磁环境体可视化[J]. 北京邮电大学学报, 2011, 34(1): 55 - 59.
YANG Chao, XU Jiangbin, WU Lingda. Hardware accelerated volume visualization in virtual electromagnetic environment [J]. Journal of Beijing University of Posts and Telecommunications, 2011, 34(1): 55 - 59. (in Chinese)

[4] 马千里, 李思昆, 白晓征, 等. CFD 非结构化网格格心格式数据高质量体绘制方法[J]. 计算机学报, 2011, 34(3): 508 - 516.
MA Qianli, LI Sikun, BAI Xiaozheng, et al. High-quality volume rendering of unstructured-grid cell-centered data in CFD[J]. Chinese Journal of Computers, 2011, 34(3): 508 - 516. (in Chinese)

[5] Beyer J, Hadwiger M, Pfister H. A survey of GPU-based large-scale volume visualization [C]// Proceedings of the Eurographics Conference on Visualization 2014, Swansea Wales, UK, 2014: 105 - 123.

[6] Rosen P. A visual approach to investigating shared and global memory behavior of CUDA kernels[C]// Proceedings of the Eurographics Conference on Visualization, Leipzig, Germany, 2013: 161 - 170.

[7] Zhang Y B, Dong Z, Ma K L. Real-time volume rendering in dynamic lighting environments using precomputed photon mapping [J]. IEEE Transactions on Visualization and Computer Graphics, 2013, 19(8): 1317 - 1330.

[8] 冯晓萌, 吴玲达, 董士伟. CUDA 加速的动态电磁环境数据场实时绘制[J]. 系统仿真学报, 2014, 26(9): 2044 - 2049.
FENG Xiaomeng, WU Lingda, DONG Shiwei. CUDA accelerated real-time rendering for dynamic electromagnetic environment volume data [J]. Journal of System Simulation, 2014, 26(9): 2044 - 2049. (in Chinese)

[9] Stuart J A, Chen C K, Ma K L, et al. Multi-GPU volume rendering using MapReduce [C]//Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 2010: 841 - 848.

[10] Arens S, Domik G. A survey of transfer functions suitable for volume rendering [C]//Proceedings of the 8th IEEE/EG International Conference on Volume Graphics, 2010: 77 - 83.

[11] 陈鹏, 杨超, 吴玲达. 硬件加速的三维雷达作用范围表现[J]. 国防科技大学学报, 2007, 29(6): 49 - 53.
CHEN Peng, YANG Chao, WU Lingda. Hardware accelerated 3D radar detection range visualization[J]. Journal of National University of Defense Technology, 2007, 29(6): 49 - 53. (in Chinese)

[12] Stefan B, Eduard M G. Instant volume visualization using maximum intensity difference accumulation [J]. Computer Graphics Forum, 2009, 28(3): 775 - 782.