DOI:10.16652/j.issn.1004-373x.2015.04.032

Voronoi图栅格生成算法GPU并行实现

屠文森，汪佳佳
（南京理工大学 计算机科学与工程学院，江苏 南京 210094）

摘 要：针对矢量法生成Voronoi图计算与存储复杂的缺点，重点分析研究了Voronoi图的栅格生成方法。对不同的栅格生成算法的复杂性和效率进行了比较分析，并针对以往方法速度较慢的问题，提出一种CUDA平台下GPU并行栅格扫描的方法。该方法利用GPU的多线程特性，将各个栅格的计算分散到不同的线程中并行处理。相比其他栅格生成方法，该方法不需要考虑栅格的规模，能够以几乎线性的时间完成Voronoi图的生成，极大地提高了生成速度。

关键词：Voronoi图；栅格法；GPU；CUDA
中图分类号：TN919-34 **文献标识码**：A **文章编号**：1004-373X(2015)04-0066-03

Raster-based method for Voronoi diagram using GPU parallel technology

TU Wen-sen, WANG Jia-jia
(School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract: Aimed at the complexity of calculation and storage in Vector-based method for Voronoi diagram, the raster-based method is researched emphatically. different methods' complexity and efficiency of generating the Voronoi diagram are analyzed. A raster-based method for Voronoi diagram generating with GPU parallel technology is raised to resolve the problem of low speed. Compared with other methods, grid size was not took into account in this method. It improves the generation speed obviously

Keywords: Voronoi diagram; Raster-based method; GPU; CUDA

Voronoi图是一种空间分割算法。其是对空间中的n个离散点而言的，它将平面分割为n个区域，每个区域包括一个点，此区域是到该点距离最近的点的集合。由于Voronoi图具有最邻近性，邻接性等众多性质和完善的理论体系，其被广泛的应用在地理学、气象学、结晶学、航天、机器人等领域。

Voronoi图的生成主要有矢量方法和栅格方法[1-3]。矢量法中，典型的方法有增量法、分治法和间接法[4-6]。分治法是一种递归方法，算法思路简单，但是很难在应用过程中实现动态更新。间接法则是根据其对偶图Delaunay三角网来构造Voronoi图，因此其性能的高低由所采用的 Delaunay三角网的构造算法所决定。增量法通过不断向已生成的Voronoi图中增加点来动态构建Voronoi图。相对于前两种方法，增量法构造简单并且容易实现动态化，所以被广泛应用[7]。矢量方法的优势是生成Voronoi图精度高，但是存在存储复杂，生长元只能是点和线，以及难以向三维及高维空间扩展等问题[8]。因此本文重点研究了Voronoi图的栅格生成方法，首先比较了常见的栅格方法生成Voronoi图的优缺点，然后结合CUDA的出现，提出一种基于GPU的Voronoi图并行栅格生成算法。

收稿日期：2014-08-15
基金项目：国家重大科学仪器设备开发专项（2012YQ05025004）

1 栅格法简介

栅格方法生成Voronoi图主要是将二值图像转化为栅格图像，然后确定各个空白栅格归属。主要方法有两类，一类以空白栅格为中心，计算每个空白栅格到生长目标的距离，以确定其归属，常见的方法有代数距离变换法，逐个空白栅格确定法等；另一类以生长目标为中心，不断扩张生长目标的距离半径，填充其中的空白栅格，直到将整个图像填充完成，主要有圆扩张法，数学形态学距离变换法等。代数距离变换法对距离图像进行上行扫描（从上到下，从左到右）和下行扫描（从下向上，从右到左）两次扫描，计算出每个空白栅格最邻近的生长目标，以此生长目标作为其归属。此方法中栅格距离的定义直接影响了空白栅格的归属和Voronoi图的生成精度，通常使用的栅格距离定义有街区距离、八角形距离、棋盘距离等[8]。距离变换的栅格生成方法精度

低、耗时长[2,7]，所需要花费的时间和栅格的数量成正比，当栅格为$n\times n$大小时，其时间复杂度为$O(n\times n)$。圆检测法[9]以生长目标为圆心，以一定的步长为初始半径，所有生长目标同时对其构成的圆内的空白栅格进行覆盖。通过不断扩大生长目标的半径，将会有越来越多的空白栅格被各个圆所覆盖，直到最终覆盖完整个图像。数学形态学距离变换法与圆检测法类似，其思想来源于数学形态学中膨胀操作，膨胀操作起到了扩大图像的效果，通过不断的对生长目标进行膨胀操作，最终扩张到所有的空白栅格。这两种方法有个共同的缺点，在每次扩张后，都需要判断整个栅格图像是否已完成扩张，而这需要遍历栅格图像，十分耗时。

2 GPU下的栅格生成方法

2.1 CUDA编程模型与GPU

CUDA是一个并行编程模型和一个软件编程环境，其采用了C语言作为编程语言，提供了大量的高性能计算指令开发能力，使开发者能够在GPU的强大计算能力上建立起一种更加高效的密集数据计算解决方案[10]。

CUDA将CPU作为主机端，GPU作为设备端，一个主机端可以有多个设备端。其采用CPU和GPU协同工作的方式，CPU主要负责程序中的串行计算的部分，GPU主要负责程序中的并行计算的部分。GPU上运行的代码被称为内核函数，其能够被GPU上内置的多个线程并行执行。一个完整的任务处理程序由CPU端串行处理代码和GPU端并行内核函数共同构成。当CPU中执行到GPU代码时，其首先将相关数据复制到GPU中，然后调用GPU的内核函数，GPU中多个线程并行执行此内核函数，当完成计算后，GPU端再把计算的结果返回给CPU，程序继续执行。通过将程序中耗时的且便于并行处理的计算转移到GPU中使用GPU并行处理，以提高整个程序的运行速度。CUDA是以线程网格(Grid)，线程块(Block)，线程(Thread)为三层的组织架构[11]，每一个网格由多个线程块构成，而一个线程块又由多个线程构成，如图1所示。在GPU中，线程是并行运行的最小单元，由此可见，当存在大量的线程时，程序的并行程度将会十分高。目前的GPU上一个网格最多包含65 535×65 535个线程块，而一个线程块通常有512个或1 024个线程，所以理论上可以对65 535×65 535×512个栅格同时进行计算。

2.2 并行Voronoi图栅格生成算法

传统的栅格生成算法中，不论是采用以空白栅格为中心确定其归属的方法，还是以生长目标为中心通过不断增长生长目标半径对空白栅格进行覆盖的方法，他们在计算每个空白栅格距离时，只能通过遍历栅格，逐一处理。而栅格处理过程中的一个重要特点是，各个栅格的计算并不依赖于其他栅格的计算结果。即各个栅格的计算是相互独立的，而由于CPU的串行性，导致了各个栅格只能顺序处理，降低了处理速度。

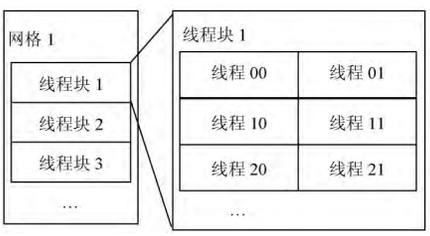

图1 GPU组织架构

由于GPU下的多个线程都是硬件实现的，各个线程的处理都是并行的，因此将栅格距离的计算分散到GPU端各个线程，必然能够提高其生成速度。为了并行处理栅格化图像，可以采用如下的想法，将每一个栅格点对应于一个线程，此线程计算此栅格到所有的生长目标的距离，取最小距离的生长目标作为其归属。即采用一个线程用来确定一个空白栅格归属的方法。

确定方法后，就需要对GPU端内核函数进行设计，由于内核函数是并行处理的执行单元，其设计方式直接决定了GPU端的程序运行效率。因此如何设计良好的内核函数是提高并行速度的关键。本文采用如下方式进行内核函数的设计，假设共分配了K个并行处理线程，栅格规模为$M\times N$，设$A[i]$为第i个线程处理的栅格编号。当$K<M\times N$时，只需要将栅格与线程按序依次分配，即第i个线程处理第i个栅格(式(1))即可。当线程个数少于栅格个数时，一个线程必须负责处理多个栅格。首先可以计算出每个线程平均需要处理的栅格个数$C=M\times N/K$，然后对栅格采用具体的分配方式，可以有两种分配方式，一种将连续的C个栅格分配给一个线程处理，即第i个线程处理第$i\times C,i\times C+1,i\times C+2,\cdots,i\times C+C-1$个栅格(式(2))。另一种方式为将栅格按$C$大小分块，将每块的第$i$个栅格分配给第$i$个线程，即第$i$个线程处理第$i,C+i,2\times C+i,3\times C+i,\cdots,(C-1)\times C+i$个栅格(式(3))。

$$A[i]=\{i\} \tag{1}$$

$$A[i]=\{i,i+1,i+2,\cdots,i+C-1\} \tag{2}$$

$$A[i]=\{i,C+i,2\times C+i,\cdots,(C-1)\times C+i\} \tag{3}$$

由于显卡上的内存是动态随机存储(DRAM)，因此最有效率的存取方式，是以连续的方式存取。当采用第一种方式时，看似是一种连续的存取方式，实际上恰好是非连续的，当第i个线程处理第i个栅格时，由于处理需要一定的时间，此时GPU自动将下个一线程$i+1$需要的内存数据取出给其使用，此时下一个线程的内存数据

却是在$i+C$处，内存变成了间断存取。而在使用第二种方式进行处理时，恰好是一种连续的存取方式，由于第i个线程正在处理第i个栅格数据，此时GPU为第$i+1$个线程准备数据，而此时的数据正好为第$i+1$内存处。满足了内存的连续存取特性。因此本文采用第二种方式，内核函数的设计伪代码如下：

```
__global__ static void kenelDo(M,N )
                                    //M为栅格行数,N为栅格列数
{
const int tid = threadIdx.x;                          //线程编号
      for(k = tid; k < DATA_SIZE; k += THREAD_NUM)
{       //DATA_SIZE: 栅格总数,THREAD_NUM:线程总数
            i= k / N;                    //获取待处理栅格行号i
      j = k-i * N;                      //获取待处理栅格列号j
      DoRaser(i,j);                      //具体的栅格处理计算
}
}
```

具体步骤如下：(这里假设栅格的规模为$M\times N$)：

Step1：根据栅格图像的规模，确定GPU端线程块和线程的分配方式和分配数量，初始化GPU端的参数。

Step2：程序调用GPU端内核函数，同时将待处理栅格图像数据传入GPU中。数据主要是图像的栅格距离，一般是二维数组，0表示空白栅格，其他各生长目标可由1，2等不同的数字定义。

Step3：GPU分配$M\times N$个thread对栅格进行处理，当$M\times N$大于所有的thread的总数时，可以将$M\times N$个栅格分块处理，即将其分成A行×B列×C块，其中$A\times B$小于thread的总数。对于分成了C块的栅格来说，每个线程只需要处理C个栅格。

Step4：当生长目标数目不多时，每一个线程计算其对应的栅格到所有的生长目标点的距离，取距离最小的生长目标，为此线程对应的空白栅格的归属，转Step6。当生长目标过多时，则转Step5。

Step5：当生长目标较多时，为了减少遍历生长目标的时间，通过借鉴王新生[13]的算法，不计算栅格点到每一个生长目标的距离，通过对空白栅格不断的进行邻域扩张，直到遇到目标生长点的方法确定此栅格的归属。

Step6 将生成后的数据返回CPU端，CPU端完成栅格图像的显示与后处理。

3 实验与总结

在CPU参数为Intel ® Xeon ® CPU E5-2609，2.4 GHz，2处理器8核心，GPU参数为TeslaC2075，448CUDA核心，显存5.25 GB的试验平台下，做了不同方法在不同栅格规模下生成Voronoi图的对比试验，试验中生长目标的个数定义为100个。由于不同的方法都采用了相同的距离定义，因此各种方法的Voronoi图生成结果都是相同的，即他们之间的生成精度是相同的，所以这里重点比较了不同方法的生成耗时。表1列出了不同方法生成Voronoi图的用时，图2为表1的折线图，从图2中可以明显看出，当栅格数量较少时，GPU并行技术的使用并不能提升生成速度，但是当栅格点数量增加时，逐点法和距离变换法用时明显增加，但GPU并行算法用时几乎不变。

表1 不同方法生成Voronoi图用时表 ms

规模	方法		
	方法逐点法	距离变换法	GPU并行
1 000×800	126.4	67.9	109.1
1 500×1 000	230.3	117.7	131.5
2 500×2 000	746.8	383.5	147.7
4 000×3 000	1 718.6	907.6	177.1
5 000×4 000	2 777.8	1494.2	219

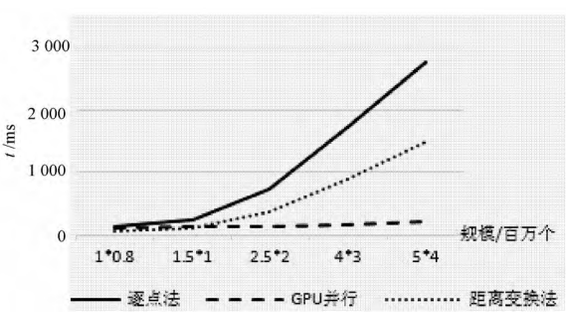

图2 Voronoi图生成时间对比图

4 结 语

通过实验结果可以看出，采用GPU对Voronoi图的生成进行并行加速，能够很好的提高生成速度。其生成Voronoi图所需时间与只与生长目标的数量有关，而与栅格规模没有关系，当生长目标数量为n时，其时间复杂度近似于$O(n)$，为线性的生成时间。相对于前面的几种CPU下串行算法，尤其是在栅格规模过大的情况下，能够很好的提高Voronoi图的生成速度。

参 考 文 献

[1] LI Cheng-ming, CHEN Jun. Raster methods of the generation of Voronoi diagrams for spatial entities [J]. International Journal of Geographical Information Science, 1999, 13: 209-225.

[2] 陈军.Voronoi动态空间数据模型[M].北京：测绘出版社，2002.

[3] KOKICHI Sugihara. VORONOI2: a fortran program for eonstructing the Voronoi Diagram [J]. Geographic Systems, 1994 (1): 347-349.

(下转第72页)

应用请求空间分配的大小。子池之间负载不均衡。有些子池空间耗尽,有些子池空闲空间较多。

4.1 共享池空闲空间碎片过多

4.1.1 测试方法

执行下面的查询检测空闲空间的碎片情况:

```
select 'sga heap('||KSMCHIDX||',0)' sga_heap,
ksmchcom ChunkComment,
decode(round(ksmchsiz/1000),0,'0-1K', 1,'1-2K', 2,'2-3K',
3,'3-4K',4,'4-5K',5,'5-6k',6,'6-7k',7,'7-8k',8,'8-9k',9,'9-10k','> 10K') Size,
count(*),
ksmchcls Status,
sum(ksmchsiz) Bytes
from x$ksmsp
where KSMCHCOM = 'free memory'
group by 'sga heap('||KSMCHIDX||',0)', ksmchcom, ksmchcls,
decode(round(ksmchsiz/1000),0,'0-1K', 1,'1-2K', 2,'2-3K',
3,'3-4K',4,'4-5K',5,'5-6k',6,'6-7k',7,'7-8k',8,'8-9k',9,'9-10k','> 10K');
```

如果1~4 KB的空闲空间数量较多,5 KB以上的空闲空间数量较少,说明共享池空闲空间碎片较多。共享池中会话之间游标不能共享,子游标较多,是造成此现象的主要原因。

4.1.2 解决方法

(1) 修改初始化参数SHARED_POOL_RESERVED_SIZE,此参数默认为5%,可以适当增大;

(2) 将初始化参数CURSOR_SHARING设置为EXACT,不能设置为SIMILAR或FORCE;

(3) 修改应用系统中的查询语句,尽量多的使用绑定变量。

4.2 子池负载不均衡

共享池中有多个子池,Oracle将进程分配到某个子池中,由于此子池中没有多于空闲空间,分配空间操作失败。虽然其他子池有很多空闲空间,也不会响应该进程的空间分配请求。当确定共享池空闲空间较多,并且碎片较少时,可以确定子池负载不均衡。每个进程都会存储大量的动态初始化参数设定。在数据库启动时,将进程分配到每个子池中。当PROCESSES参数值较大(允许连接数据库的进程较多),而实际的并发量不大,实际连接到数据库的进程分配的子池不均衡,会出现此现象。

可以通过下面的方法解决:

(1) 将初始化参数PROCESSES改小;

(2) 修改隐含参数_kghdsidx_count,适当增加子池数量。

参 考 文 献

[1] CYRAN Michele. Oracle database concepts 10 g release 2 (10.2) [EB/OL]. [2014-07-09]. http://docs.Oracle.com/cd/E11882_01/server.112/e25789/memory.htm#CNCPT1226.

[2] BANSAL Amit. Simplified approach to resolve ORA-4031[EB/OL]. [2008-07-21]. http://askdba.org/weblog/2008/04/application-design-and-ora-4031.

[3] LEWIS Jonathan. Oracle core essential internals for DBA and developers [M]. [S.l.]: Springer Science+Business Media, 2011.

[4] SHAMSUDEEN Riyaj. A stroll through shared pool heaps [EB/OL]. [2009-01-15]. http://orainternals.wordpress.com/2009/01/15/a-stroll-through-shared-pool-heaps.

[5] GREEN Russell. Understanding shared pool memory structures [EB/OL]. [2005-09-22]. http://docs.Oracle.com.

[6] PODER Tanel. ORA-04031 errors and monitoring shared pool subpool memory utilization with Sgastatx SQL [EB/OL]. [2009-06-04]. http://www.blog.tanelpoder.com.

作者简介:胡晨光(1982—),吉林白山人,高级工程师,硕士。研究方向为高校信息化、系统集成、数据库管理。

(上接第68页)

[4] OKABE A, BOOTS B, SUGIHARA K, et al. Spatial tessellations: concepts and applications of Voronoi diagrams [M]. 2nd ed. New York: John Wiley and Sons, 2000.

[5] HAMMER Auren. Voronoi diagrams: a survey of a fundamental geometric data Strueture [J]. ACM Computing Surveys, 1991 (23): 345-405.

[6] 李成名,陈军.Voronoi图生成的栅格算法[J].武汉测绘科技大学学报,1998,23(3):208-210.

[7] 李成名.基于Voronoi图的空间关系描述、表达与推断[D].武汉:武汉测绘科技大学,1998.

[8] 陈军,赵仁亮.Voronoi动态空间数据模型[M].北京:测绘出版社,2002.

[9] 罗以宁,王开升.散平面Voronoi图的光栅图形算法[J].四川大学学报,2003(3):596-599.

[10] 李波,赵华成,张敏芳.CUDA高性能计算并行编程[J].微型电脑应用,2009,25(9):55-57.

[11] 吴焰斌.CUDA编程模型[J].科技风,2009(3):63-64.

[12] 王新生,刘纪远,庄大方,等.一种新的构建Voronoi图的栅格方法[J].中国矿业大学学报,2003(3):293-296.

作者简介:屠文森(1989—),男,硕士研究生。研究方向为计算机可视化。
汪佳佳(1990—),女,硕士研究生。研究方向为计算机可视化。