

# GPU加速的多边形叠加分析

赵斯思,周成虎

(中国科学院地理科学与资源研究所资源与环境信息系统国家重点实验室,北京 100101)

**摘 要:**叠加分析是地理信息系统最重要的分析功能之一,对多边形图层进行叠加分析要花费大量时间。为此,将GPU用于多边形叠加分析过程中的MBR过滤及多边形剪裁两个阶段。对MBR过滤阶段,提出了基于GPU的通过直方图及并行前置和实现的MBR过滤算法。对多边形剪裁阶段,通过改进Weiler-Atherton算法,使用新的焦点插入方法和简化的出入点标记算法,并结合并行前置和算法,提出了基于GPU的多边形剪裁算法。对实现过程中可能出现的负载不均衡情况,给出了基于动态规划的负载均衡方法。通过对这些算法的应用,实现对过滤阶段及精炼阶段的加速。实验结果表明,基于GPU的MBR过滤方法相对CPU实现的加速比为3.8,而基于GPU的多边形剪裁的速度比CPU实现快3.4倍。整体上,与CPU实现相比,GPU加速的多边形叠加提供了3倍以上的加速比。

**关 键 词:**叠加分析;图形处理单元;多边形剪裁;并行计算;空间分析

doi: 10.3724/SP.J.1033.2013.00114

## 1 引言

叠加分析是地理信息系统最重要的空间分析功能之一,常用于提取空间隐含信息。叠加分析是将同一地区的两个或多个图层进行叠加产生一个新要素层的操作,其结果是将原要素分割生成新的要素,新要素综合了原来两层或多层要素所具有的属性(陈述彭等,1999)。

为高效地完成叠加操作,现代的叠加分析算法将整个处理过程分为两个阶段:过滤和精炼。过滤阶段通过引入空间索引、空间对象的近似表示等技术,以较低的计算代价排除叠加结果为空的空间对象,余下的是可能产生非空叠加结果的对象的集合(被称为候选集)。精炼阶段对候选集进行实际的几何运算。例如,对于多边形图层之间的叠加分析中,精炼阶段即是对候选集进行多边形求交、并、差等。本文仅讨论多边形求交(即多边形剪裁)的情形。

由于精炼阶段的几何运算往往需要较大的计算开销,因此,通过提高过滤阶段的效率减小候选集的大小,从而避免或减少精炼阶段的计算量是提高叠加分析效率的重要途径之一。Brinkhoff等(1993)通过使用R树索引进行过滤,获得了较好的效果;周晓芳等(1998)通过使用格网将空间进行分

割后并行处理的方式,实现了高效的并行化过滤;朱欣焰等(2011)基于分布式计算环境,对于区域分割造成的分片,提出了跨(分片)边界的空间拓扑连接优化规则,对跨边界连接和过滤有明显优势。

多边形剪裁算法在过去的几十年中也得到了完善与发展。学者们提出各种剪裁算法以应对多边形剪裁中出现的各类问题。Sutherland等(1974)提出Sutherland-Hodgman算法,解决了任意多边形与凸多边形之间的剪裁问题;Weiler等(1977)提出Weiler-Atherton算法,这种算法对Sutherland-Hodgman算法进行了增强,不再要求剪裁多边形是凸多边形。但是,Weiler-Atherton算法不允许多边形自相交。为解决这类问题,Vatti(1992)提出Vatti剪裁算法,以实现自相交多边形之间的剪裁。

虽然用于精炼阶段的优化方法可以通过减小候选集大小的方式避免一部分不必要的几何运算。但是,精炼阶段的几何运算仍然是叠加分析中最为费时的部分,尤其是最为复杂的多边形剪裁操作。为提高精炼阶段的效率,人们提出了一些优化算法。Greiner等(1998)对Vatti算法进行简化和优化,提出了更加简单和快速的多边形剪裁算法;刘勇奎等(2003)通过应用改进的数据结构及创新的交点计算方法,进一步提高了多边形剪裁操作的效率。但是,这些优化的算法的性能提升较为有限。

收稿日期:2012-06; 修订日期:2012-09.

基金项目:国家自然科学基金项目(40830529);国家"863"计划项目(2011AA120305)。

作者简介:赵斯思(1983-),男,博士,主要研究方向为高性能地学计算、空间分析、地统计等。E-mail: zhaoss@lreis.ac.cn

随着图形处理单元(Graphics Processing Unit, GPU)通用计算能力的提高,利用GPU的大规模并行的特点实现高性能计算的方式已经在地理信息科学中得到了一定的应用,并在网络分析(Garcia et al, 2008; Harish et al, 2007)、高精度曲面建模(闫长清等, 2012)、地统计学(Zhao et al, 2008)、空间过程模拟(李丹等, 2011; Walsh et al, 2009)、空间数据库(Simion et al, 2012)等方面取得了很好的效果。但是,由于GPU在计算模型、编程接口和实现细节等方面与传统的计算单元CPU有较大的不同,许多GIS中的算法与模型需要进行较大幅度的修改在GPU上高效地运行。

本文将GPU用于多边形叠加分析过程,提出基于GPU的多边形剪裁算法及MBR过滤算法。通过应用这两种算法实现对过滤步骤和精炼步骤的加速,以达成高效率的多边形叠加操作的目的。

## 2 叠加算法的GPU实现

叠加分析常采用“过滤—精炼”的方法来提高效率。在实践中,尤其是基于空间索引的叠加分析方法中,过滤阶段又被进一步分为索引过滤阶段和MBR(Minimal Bounding Rectangle)过滤阶段。索引过滤阶段使用索引对多个图层的子空间进行连接得到备选集;MBR过滤阶段滤除MBR不相交的备选项。

对于典型的空间索引R树来说,无论其构建还是读取(查询)均较为复杂,不太适合GPU这种大规模SIMD(Single Instruction Multiple Data)计算单元。因此,在索引过滤阶段中,仅使用CPU进行运算。而在MBR过滤阶段及精炼阶段中,采用GPU对运算进行加速。相对于索引过滤阶段,MBR过滤阶段及精炼阶段才是叠加分析中最耗时的部分。因此,在这两阶段中使用GPU加速的方法是有意义的。

### 2.1 并行的前置和运算

并行的前置和运算在叠加分析中多次被使用,因此,先叙述其定义及并行化实现。对于给定的运算 $\Delta$ ,单位元 $I$ 及一个拥有 $n$ 个元素的数列 $[a_0, a_1, \dots, a_{n-1}]$ ,其前置和(prefix sum)被定义为:

$$[I, a_0, (a_0 \Delta a_1), \dots, (a_0 \Delta a_1 \Delta \dots \Delta a_{n-2})] \quad (1)$$

前置和的运算 $\Delta$ 往往就是加法。前置和的第 $j$ 个元素,都是其前面 $j-1$ 个元素的和。计算前置和

的操作往往被称作Scan。

第一个并行的前置和实现由Hillis等(1986)给出,Horn(1992)给出了它的GPU实现。但是,这种实现并不足够高效,因此,本文采用Harris等(2007)给出的一个GPU上的高效实现。这是一个两阶段的算法:第一阶段被称为向上扫描阶段,也叫规约阶段。规约计算使用高效的全局显存访问模式实现了一个规约加法。第二阶段被称为向上扫描阶段,这一阶段首先将最后一个数字置零,然后二分地执行如下操作:将子数列最后一个数字和子数列中点加总,并赋值子数列最后一个数字,同时将最后一个数字赋值给子数列的中点;然后,将数列一分为二递归执行直到子数列只有2个元素。

### 2.2 MBR过滤

MBR过滤的GPU实现包含2部分:MBR相交测试和测试后过滤。MBR相交测试是一个相对平凡的过程:只需要比较两个多边形的MBR是否相交即可;测试后过滤部分将不符合要求的候选多边形对从结果列表中排除。这一部分在GPU上的实现主要的困难在于GPU线程之间的同步代价很高,为充分发挥GPU的并行特性,需要有一种方法可以进行并行筛选。

利用直方图(histogram)计算和并行的前置和实现GPU上的MBR过滤。假定MBR相交测试后得到一个数组 $A$ , $A$ 的每个成员 $A_i$ 表示第 $i$ 个备选多边形对是否通过多边形MBR相交测试(通过为1,不通过为0)。MBR测试后过滤部分由3阶段构成:

(1) 针对备选多边形对集合,并行地对其中备选多边形对 $\langle P, Q \rangle$ 中的第一个多边形 $P$ 指针(id)制作直方图,每一线程对应一个多边形 $P$ ,统计出每一个 $P$ 对应多少个通过MBR相交测试的备选多边形对。

(2) 对阶段一生成直方图进行并行化前置和操作,计算的结果即是按照 $P$ 分类后,每个类别通过MBR相交测试的备选多边形对的起始地址位置。

(3) 根据阶段二的前置和结果,进行并行化筛选—复制过程。详细过程是,每一个线程以前置和结果数组的相应元素为起始值,根据MBR相交测试结果 $A$ ,将通过测试的备选多边形对复制到输出数据集中。

### 2.3 改进的多边形剪裁算法

根据各种多边形叠加需要,研究者们提出了多种算法。其中,Weiler等(1997)提出Weiler-Atherton算法允许裁剪窗口是非凸多边形。为高效地实现

GPU上的多边形剪裁,对 Weiler-Atherton 算法进行了改进。

将被裁剪多边形记为  $P$ ,它是由一系列顶点  $P_i$  构成 ( $0 \leq i < n, n$  为顶点个数),  $P$  的边被记为  $P_iP_j$  形式,其中,  $0 \leq i < j < n$ 。裁剪窗口多边形记为  $Q$ ,由一系列顶点  $Q_j$  构成 ( $0 \leq j < m, m$  为顶点个数),  $Q$  的边被记为  $Q_jQ_i$  形式,其中,  $0 \leq i < j < m$ 。多边形叠加算法分为3个阶段:

(1) 第一阶段:找到所有的边之间的交点并将交点插入边中

遍历  $A$  的所有边及  $B$  的所有边  $P_iP_j$  及  $Q_jQ_i$ ,判断  $P_iP_j$  与  $Q_jQ_i$  是否相交。若相交则求出交点  $I$ ,将交点  $I$  分别插入  $P_iP_j$  及  $Q_jQ_i$ ,将  $P_iP_j$  划分为两条线段  $P_iI$  与  $IP_j$ ,将  $Q_jQ_i$  也划分为两条线段  $Q_jI$  和  $IQ_i$ 。同时为焦点计算一个  $\delta$  值。在求得交点后,将交点插入在  $P_iP_j$  之间的所有交点中找到第一个比  $\delta$  大的交点即可。 $\delta$  值的计算如公式 1 所示。

$$\delta = \frac{\text{dist}(P_i, I)}{\text{dist}(P_i, P_j)}$$

(2)

(2) 第二阶段:标记交点为出点或入点

根据  $P$  的第一个顶点  $P_1$  是否在  $Q$  之内,然后顺序遍历  $P$  的边以遍历上一步骤中找到的交点,从而交叠地标记交点的出点和入点。例如,若  $P_1$  在  $Q$  之内,则标记第一个交点为出点,第二个交点为入点,第三个交点为出点,以此类推。

(3) 第三阶段:遍历顶点和交点以获得结果

从一个交点  $I$  (入点) 开始遍历  $P$  中的交点之后的顶点,直到遇到一个出点  $I_i$ ,然后跳转到  $Q$  中  $I_i$  所在边继续反向遍历  $Q$  中的顶点,直到遇到一个入点  $I_k$ ,跳转回  $P$  中  $I_k$  所在边继续遍历。重复以上过程直到回到交点  $I$ ,将以上遍历的所有点输出为结果集中的一个多边形,并将  $I$  标记为已访问。重复整个过程直到  $P$  中找不到未被访问过的入点为止。

相对于 Weiler-Atherton 算法,本文算法的改进主要在2个方面:

(1) 在求交点并将交点插入多边形边的时候引入  $d$ 。 $d$  的引入减少了求交点运算:若  $P_iP_j$  首先被交点分割为  $P_iI$  与  $IP_j$ ,下一次求交运算也仅需在  $P_iP_j$  上执行而不需要针对分割后的两条线段  $P_iI$  与  $IP_j$  分别计算。此外,更多的交点分割及求交运算将会影响 GPU 的运行效率——引起频繁的内存变动,而 GPU 对全局存储器的随机访问相对低效。

(2) 在标记出点和入点的时候,仅对多边形使用了一次点在多边形内算法,然后进行交叠标记。

这也减少了标记出点和入点的计算代价。在 GPU 上,这种改进通过前置和操作可以较为高效地实现。

2.4 多边形剪裁算法的 GPU 实现

整个多边形剪裁算法的 GPU 实现流程如图 1 所示。在将多边形剪裁算法进行 GPU 实现时发现,算法的第三阶段由于需要频繁的对全局内存进行搜集(gather)和分散(scatter)操作,且根据条件(是否是交点,是出点还是入点)进行分支并来回在被切割多边形和切割窗口的坐标点串上遍历,所以第三阶段并不适合在 GPU 上实现。考虑到第三阶段所耗费时间占整个过程的比例往往较低,因此,第三阶段的负载均交由 CPU 处理是可以接受的。

第一阶段和第二阶段 GPU 实现方式为:

第一阶段的 GPU 实现与通常的 CPU 串行实现不同——无法在遍历 2 个多边形并求交点时立即将交点插入多边形的边中。为解决这一问题,将其分为两步:求交步骤和交点插入步骤。

(1) 求交步骤

求交步骤并行地遍历两个多边形,求出所有边的交点并计算  $\delta$  值,将这些信息都存入共享内存中 (Shared Memory)。此处使用了 CUDA 的 atomicAdd 操作以在不同线程之间同步交点存储位置的索引值。本步骤的详细算法如算法 1 所示。

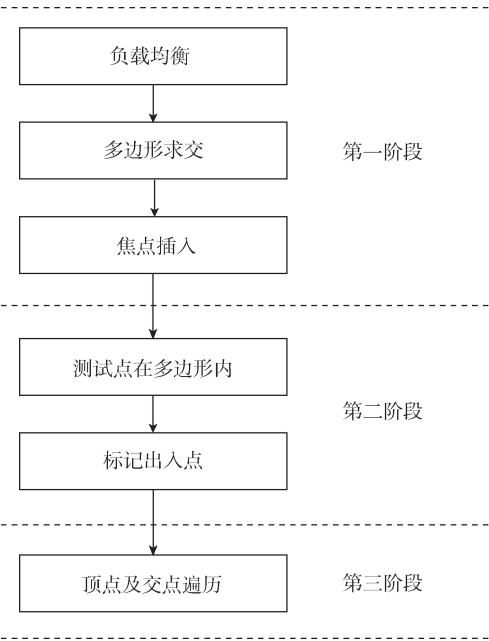


图 1 多边形剪裁算法实现流程图  
Fig. 1 Flow chart of polygon clipping algorithm



```

Procedure edge_intersect(P, Q)
  if threadid=0 then
    count = -1
  end if
  for each e1 in P in parallel do
    for each e2 in Q do
      if intersect_test(e1,e2) then
        pt = intersect(e1,e2)
        index = atomicAdd(&count, 1)
        cache[index]=p
      end if
    next
  next
End

```

## (2) 交点插入步骤

在交点插入步骤,分别对被切割多边形和切割窗口进行交点插入操作。实际上这时的插入操作是将多边形上的顶点和交点复制到新的顶点数组中(存储在全局内存中)。这里仅叙述对被切割多边形的交点插入步骤,对切割窗口的交点插入步骤与此类似,因此不再赘述。

首先,利用并行的前置和来计算交点需要插入到新的顶点数组中的位置;然后,每一个线程对应一条被切割多边形上有交点的边,搜寻并插入交点。在插入交点时,需要查找并比较  $\delta$  值。本步骤的详细算法如算法 2 所示。

```

Procedure insert_point(P, Points, out P2)
  parallel_prefix_sum(P, N)
  for each e in P in parallel do
    index = start_index(P2, e)
    P2[index]= start_point(e)
    for each p2 in Points do
      if p2 on e then
        Insert_by_delta(P2, p2)
      end if
    next
  next
End

```

为提高计算的并行化程度,将第二阶段也分为 2 个步骤:测试点在多边形内步骤和标记出入点步

骤。第一步骤使用射线法进行判定,在此不再赘述。对于标记出入点步骤,采取如下策略:对于包含顶点坐标和交点的输入数组,首先并行地生成一个同样长度的 0~1 标志数组,若输入数组上某位置是顶点坐标,则标志数组对应位置上置为 0,反之置为 1。然后对这个标志数组进行前置和操作,对得到的结果数组进行奇偶判断即可得到交点是出点还是入点了。

GPU 是一种大规模并行处理器,在进行基于 GPU 的批量多边形剪裁时,由于多边形顶点数量变化较大,因此会遇到并行处理的一个典型问题——负载均衡。经过测试,负载均衡问题最为简单和易实现的算法——Round Robin 算法对于多边形叠加算法在 GPU 上的实现来说效率较低。另一方面,考虑到 CUDA 的核调用需要一定成本,因此,GPU 实现采用分批处理的方式:一次向 GPU 发送一大批备选多边形对进行处理。对于这种分批处理的模式,普遍认为效果更好的动态负载均衡调度算法在这里并不适用。

本文借鉴解决背包问题的动态规划算法以实现静态的负载均衡调度算法:首先,估计单一计算的代价和整个计算的总代价,根据它们给出一个批处理的代价指标(背包容量);然后,将每一个计算放到背包中,当背包中总计算代价超过指标则意味着背包满了。此时将接下来的计算放入下一背包,重复这个过程直到所有计算都被放入背包,然后将得到的一系列背包根据 GPU 可以容纳的 block 数量,分批分配给 GPU 的 block 进行处理,从而实现负载均衡的目的。

在多边形剪裁算法中,多边形求交、交点插入、点在多边形内、出入点标记等步骤均存在不同程度的不均衡,可以应用上述算法实现负载均衡。在 GPU 实现中,多边形求交步骤由于其计算复杂度不均衡性更为明显(由被裁剪多边形的顶点数量与裁剪窗口的多边形数量的乘积决定),且多边形求交步骤本身在整个多边形叠加中占用的时间最多。因此,对这一步骤应用负载均衡是非常必要的。

## 3 性能评估

### 3.1 测试环境

本节中所有测试案例均运行于主频为 2.66GHz 的 Intel Core i5 CPU 750 的计算机上, GPU 为

NVIDIA GTX260+。所有的测试都使用 Microsoft C++ compiler 版本 14 和 CUDA 版本 3.0 进行编译。

3.2 MBR 过滤

由于 MBR 过滤部分的计算量相对较小,为了更好地展示基于 GPU 的 MBR 过滤算法的效率,这一部分测试使用了 2 个较大的图层作为测试数据:2009 年浙江省海宁市的土地现状类型图层和规划类型图层。前者有 52805 个对象,后者有 37368 个对象。假设在 MBR 过滤之前并不使用索引,此时需要进行 MBR 相交测试的备选多边形对集合就是 2 个图层的笛卡尔积——共有 1973217240 对多边形需要进行 MBR 相交测试。在这样的负载下,进行 GPU 实现与 CPU 实现的性能比较。

图 2 是 GPU 实现的执行时间与 CPU 实现时间的比较,纵轴是执行时间,横轴中标识 CUDA 核调用中 block 的大小。GPU 实现的执行时间随着 block 数量的增加而逐渐缩短。这是由于较少的 block 数量不足以发挥 GPU 中所有计算单元的性能,并且无法更好的隐藏访问显存的延迟造成的。相对 CPU 实现来说,GPU 实现了最高 3.8 倍的加速比。

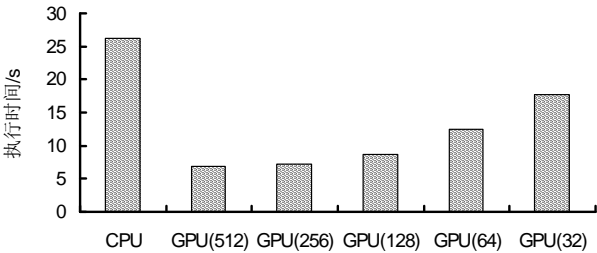


图 2 MBR 过滤 GPU 实现与 CPU 实现执行时间对比  
Fig. 2 Comparison between MBR filtering algorithms implemented on CPU and GPU

3.3 多边形剪裁

采用 Greiner 等(1998)实现的基于 CPU 多边形剪裁算法作为比较对象。此算法略优于经典的 Vatti 算法,并且在多边形顶点较多时能够达到高于 Vatti 算法一倍的速度。

测试数据使用 2010 年土地利用规划中浙江省某乡的土地利用现状图层及土地利用规划图层。

其中,土地利用现状图层有 300 个多边形,多边形的最小顶点数为 5,最大顶点数为 906;而土地利用规划图层有 174 个多边形,多边形最小顶点数为 4,最大顶点数为 1300。

三阶段综合执行时间的对比如图 3 所示,纵轴表示执行时间,横轴表示 CUDA 核调用时 block 的大小。实验的结果表明,第一阶段是最为耗时的操作;第二阶段在 GPU 上执行的效率很高,耗时几乎可以忽略;而对于 CPU 和 GPU 混合实现来说,第三阶段的纯 CPU 实现占用了一定时间,但是比例相对合理。此外,CPU 和 GPU 的混合实现对比纯 CPU 实现,最高达到了 3.4 倍左右的加速比。和 MBR 过滤部分类似,GPU 实现的执行时间随着 block 数量的增加而逐渐缩短。随着 block 数量的增加,GPU 中的计算单元被充分利用,访问显存所造成的延迟也更好地被隐藏。

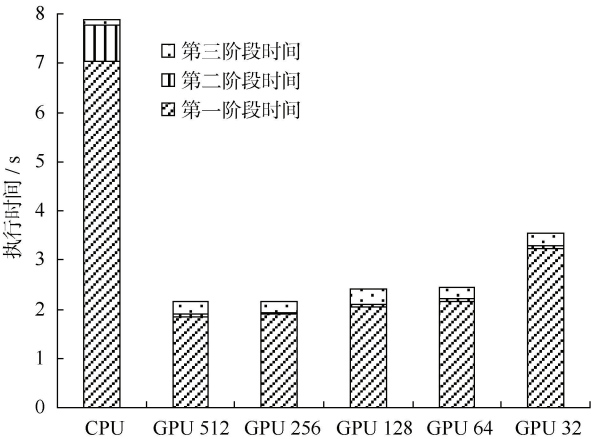


图 3 多边形剪裁的 GPU 实现与 CPU 实现  
执行时间综合对比

Fig.3 Comparison between polygon clipping algorithms implemented on CPU and GPU

3.4 综合性能

最后,将 MBR 过滤和多边形叠加两阶段结合起来,对整体的性能进行综合考察。实验数据仍旧选择多边形剪裁部分的测试数据。根据前面的测试经验,将 block 大小设置为 512,测试结果如表 1 所示。

无论 GPU 实现还是 CPU 实现,相对于多边形剪裁所耗费的时间,用于 MBR 过滤的时间几乎可

表 1 CPU 实现和 GPU 实现的执行时间对比

Table 1 Comparison between polygon overlay algorithms implemented on CPU and GPU

	多边形叠加	MBR 过滤	总时间
CPU	4.77	0.022	4.792
GPU	1.365	0.006	1.371

以忽略不计。GPU 实现的速度大约是 CPU 实现的 3.5 倍。

4 结论

本文基于 GPU 通用计算的观点,提出使用 GPU 对多边形叠加分析操作进行加速。基于 GPU 实现了高效的并行 MBR 过滤算法及多边形剪裁算法。经测试,这两种算法的 GPU 实现相对 GPU 获得均获得了超过 3 倍的加速比。通过将这 2 种算法应用于整个分析操作中,实现了快速高效的多边形叠加分析运算。

本文的研究内容未涉及对空间索引的构建和查询的 GPU 优化。此外,本文仅描述了多边形剪裁的 GPU 实现,而未涉及其他类型的多边形操作(如求并、求差等)。以后将以此为题进行更为深入的探讨。最后,当前的商用 GPU 的双精度浮点运算性能大大低于单精度浮点运算性能。本文实现的 GPU 加速的空间叠加分析方法的双精度版本相对单精度版本也会相应的有较大的性能损失。这个缺陷有待于 GPU 厂商今后对于产品的双精度计算性能的改进。

参考文献(References)

Brinkhoff T, Kriegel H P, Seeger B. 1993: Efficient processing of spatial joins using R-trees//Proceedings of ACM SIGMOD International Conference on Management of Data. Washinton D.C.: ACM Press: 237-246.

Chen S, Lu X, Zhou C. 1999. Introduction to Geographic Information Systems. Beijing, China: Science Press: 125-126. [陈述彭, 鲁学军, 周成虎. 1999. 地理信息系统导论. 北京: 科学出版社: 125-126.]

Garcia V, Debreuve E, Barlaud M. 2008. Fast k-nearest neighbor search using GPU//Proceedings of the Workshop on Computer Vision on GPU. Alaska, USA: 1-6.

Greiner G, Hormann K. 1998. Efficient clipping of arbitrary polygons. Journal of ACM Transactions of Graphics, 17 (2): 71-83.

Harish P, Narayanan P. 2007. Accelerating large graph algorithms on the GPU using CUDA//Proceedings High Performance Computing, 4873: 197-208.

Harris M, Sengupta S, Owens J D. 2007. Parallel prefix sum (scan) with CUDA//Nguyen H. 2007. GPU Gems 3. New Jersey: Addison-Wesley Press: 851-876.

Hillis W D, Steele G L. 1986. Data parallel algorithms. Communications of the ACM, 92(2): 1170-1183.

Horn D. 1992. Stream reduction operations for GPGPU Applications//Pharr M, Fernando R. 2005. GPU Gems 2. New Jersey: Addison-Wesley Press: 573-589.

Li D, Li X, Liu X P, et al. 2011. GPU-CA model and large-scale land-use change simulation. Chinese Science Bulletin, 57(11): 959-969. [李丹, 黎夏, 刘小平, 等 2011. GPU-CA 模型及大尺度土地利用变化模拟. 科学通报, 57(11): 959-969.]

Liu Y Q, Gao Y, Huang Y Q. 2003. An efficient algorithm for polygon clipping. Journal of Software, 14(4): 845-856. [刘勇奎, 高云, 黄有群. 2003. 一个有效的多边形裁剪算法. 软件学报, 14(4): 845-856.]

Longley P A, Goodchild M F, Maguire D J, et al. 2007. Geographic information systems and science. 2nd ed. Beijing, China: China Machine Press: 208-282. [Longley P A, Goodchild M F, Maguire D J, et al. 2007. 地理信息系统与科学. 2 版. 张晶, 刘瑜, 张洁, 等, 译. 北京: 机械工业出版社: 208-282.]

Simion B, Ray S, Brown A D 2012. Speeding up spatial database query execution using GPUs. Procedia Computer Science, 9: 1870-1979.

Sutherland I, Hodgman G W. 1974. Reentrant polygon clipping. Communications of the ACM, 17(1): 32-42.

Vatti B R. 1992. A generic solution to polygon clipping. Communications of the ACM, 35(7): 56-63.

Walsh S D C, Saar M O, Bailey P et al. 2009. Accelerating geoscience and engineering system simulations on graphics hardware. Computers & Geosciences, 35(12): 2353-2364.

Weiler K, Atherton P. 1977. Hidden surface removal using polygon area sorting. Computer Graphics, 11(2): 214-222.

Yan C Q, Yue T X. 2012. A novel method for Dynamic modeling and real-time rendering based on GPU. Geo-information Science, 14(2): 149-157. [闫长青, 岳天祥. 2012. 基

- 于GPU的HASM动态模拟与实时渲染方法. 地球信息科学学报, 14(2): 149-157.]
- Zhao S, Zhou C. 2010. Accelerating spatial clustering detection of epidemic disease with graphics processing unit// Proceedings of Geoinformatics 2010. Beijing, China: 1-6.
- Zhou X, Abel D J, Truffet D. 1998. Data partitioning for parallel spatial join processing. *Geoinformatica*, 2(2): 175-204.
- Zhu X Y, Zhou C H, Wo W, et al. 2011. Distributed spatial data fragmentation and cross-border topological join optimization. *Journal of Software*, 22(2): 269-284. [朱欣焰, 周春晖, 芮维, 等. 2011. 分布式空间数据分片与跨边界拓扑连接优化方法. 软件学报, 22(2): 269-284.]

## Accelerating polygon overlay analysis by GPU

ZHAO Sisi, ZHOU Chenghu

(State Key Laboratory of Resources and Environmental Information System, Institute of Geographical Sciences and Natural Resources Research, CAS, Beijing 100101, China)

**Abstract:** Overlay analysis is one of the most important analysis capabilities of geographic information systems. Overlay analysis with polygon layers is a time-intensive process. To improve time efficiency, modern approaches of overlay analysis are generally divided into two stages, filtering and refinement (also known as polygon clipping). A great deal of effort has been taken to significantly reduce the number of candidates in the first stage (filtering) in order to alleviate the workload of the second stage (refinement). However, the second stage is still the most time-consuming part of the process. In this paper we applied GPGPU (General-purpose Graphics Processing Unit) computing to the two key stages of overlay analysis: MBR filtering (part of the filtering) and polygon clipping, and restricted the search area to polygon intersection analysis. We proposed GPU-based MBR filtering algorithm by combining histogram and parallel prefix-sum algorithms, and introduced GPU-based polygon clipping algorithm by improving Weiler-Atherton algorithm. There are two differences between our algorithm and Weiler-Atherton algorithm: First, it adopts a new method to insert intersecting points which reduces computing workload, making it more suitable to be implemented on GPU. Second, it simplifies the algorithm used to mark entry and exit points. Furthermore, based on dynamic programming, we provided a solution to avoid load imbalance caused by spatial data skew. The improved algorithms and the solution have been applied to filtering stage and refinement stage to achieve better performance. The experimental results show that the speedup ratio between GPU-based MBR filtering implementation and CPU implementation is 3.8. The GPU-based polygon clipping implementation runs 3.4 times faster than the CPU's. Overall, GPU-accelerated polygon intersection implementation achieves performance that is up to three times faster than CPU implementation. Accelerating other types of overlay analyses, such as union, difference, etc., by GPU needs to be studied and implemented in the future.

**Key words:** overlay analysis; GPGPU; polygon clipping; parallel computing; spatial analysis