CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **26**:434–446
Published online 26 February 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3005

A parallel algorithm for constructing Voronoi diagrams based on point-set adaptive grouping

Jiechen Wang[1,*,†], Can Cui[2], Yikang Rui[3], Liang Cheng[1], Yingxia Pu[1], Wenzhou Wu[4] and Zhenyu Yuan[1]

[1]*Jiangsu Provincial Key Laboratory of Geographic Information Science and Technology, Nanjing University, Nanjing, 210093, China*
[2]*Urban and Regional Research Centre Utrecht, Faculty of Geosciences, Utrecht University, Utrecht, 3584 CS, The Netherlands*
[3]*Geoinformatics, Royal Institute of Technology, 10044, Stockholm, Sweden*
[4]*State Key Laboratory of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, Beijing, 100101, China*

SUMMARY

This paper presents a parallel algorithm for constructing Voronoi diagrams based on point-set adaptive grouping. The binary tree splitting method is used to adaptively group the point set in the plane and construct sub-Voronoi diagrams for each group. Given that the construction of Voronoi diagrams in each group consumes the majority of time and that construction within one group does not affect that in other groups, the use of a parallel algorithm is suitable. After constructing the sub-Voronoi diagrams, we extracted the boundary points of the four sides of each sub-group and used to construct boundary site Voronoi diagrams. Finally, the sub-Voronoi diagrams containing each boundary point are merged with the corresponding boundary site Voronoi diagrams. This produces the desired Voronoi diagram. Experiments demonstrate the efficiency of this parallel algorithm, and its time complexity is calculated as a function of the size of the point set, the number of processors, the average number of points in each block, and the number of boundary points. Copyright © 2013 John Wiley & Sons, Ltd.

Received 5 January 2012; Revised 6 January 2013; Accepted 6 January 2013

KEY WORDS: Voronoi diagrams; parallel algorithm; adaptive grouping; geographical information system; computational geometry

1. INTRODUCTION

The construction of Voronoi diagrams is a classical problem in computational geometry and has been applied in many fields including geographical information systems, meteorology, ecology, and molecular biology. As a result, the automatic construction of Voronoi diagrams has garnered much research attention [1, 2]. Voronoi diagrams based on multi-dimensional space entities, such as points, lines, and polygons, have been extensively studied. In particular, Voronoi diagrams of a point set in the plane are the most widely researched and used. The associated algorithms can be classified into two groups: raster and vector.

The two main raster approaches are the raster expansion algorithm and the near-raster vested algorithm [3–5]. The key to the neighbor raster expansion algorithm is to rasterize two-dimensional space by assigning each site in the point set to a raster cell. The value of the cell is set as the identifier of the

*Correspondence to: Jiechen Wang, Jiangsu Provincial Key Laboratory of Geographic Information Science and Technology, Nanjing University, Nanjing, 210093, China.
†E-mail: hotmailwang@yahoo.com.cn

Voronoi polygon (typically using the sequence number of the site), and the cell edge is taken as the expansion distance. Each point is regarded as the center of its cell, which then expands to surrounding cells. During expansion, if not previously assigned, each cell will be assigned a value (the Voronoi polygon identifier). Finally, cells with the same value are extracted according to the raster cell attribute, and the edges of the Voronoi diagrams are extracted and structured using the raster boundary tracing algorithm. In contrast to the raster expansion algorithm, the near-raster vested algorithm traverses each raster cell to find that closest to the site, which is then assigned the cell value. Finally, each point in the initial point set possesses a raster cell set, the region of which is the Voronoi polygon of the site. The time efficiency of raster algorithms is heavily dependent on the cell density. Moreover, Voronoi diagrams constructed by raster algorithms lose accuracy, so raster algorithms are only suitable for situations with small amounts of data and low precision requirements, consequently limiting their application.

There are several vector algorithms, such as the incremental algorithm, the sweepline algorithm, the divide-and-conquer algorithm, and the indirect algorithm. The oldest of these techniques is the incremental algorithm, which adds points to the Voronoi polygons one-by-one before modifying the structure of the local Voronoi diagrams to construct a new set of Voronoi diagrams. The underlying idea behind the incremental method is simple, easily implemented, and dynamically updated. However, its time complexity is $O(n^2)$, which makes its practical application difficult with large data sets. As a result, some researchers have optimized incremental methods. For instance, Martin and Stevan [6] designed a randomized incremental insertion algorithm with the improved time complexity of $O(n\log(n))$.

The sweepline (or Fortune's) algorithm was proposed by Fortune [7]. It uses a vertical line to sweep the point set from left to right in the plane, constructing a series of interconnecting parabolas that eventually form the Voronoi diagram. We discuss this algorithm in detail in Section 2.

The basic principle of the divide-and-conquer algorithm is that one point S_i within the point set constructs a perpendicular bisector with only its closest points. This means that the perpendicular bisectors constructed with farther points are invalid Voronoi edges. So, to reduce the determination of invalid points and improve the efficiency of querying and searching for the mid-point, Shamos and Hoey [8] proposed a divide-and-conquer algorithm. The algorithm is implemented as follows: (i) all points are sorted by their x-coordinate and divided into two approximately equal groups by a vertical line; (ii) Voronoi diagrams are constructed for each group; and (iii) the groups are merged at the vertical line. Divide-and-conquer recursively groups the point set in the plane, meaning that the point set of each sub-group is also divided into groups by its inner vertical line. The diagrams on either side of the vertical line are merged using the split chain method. Complete and incomplete edges on each side of the split chain are removed, and the split chain is added to the new Voronoi diagram to complete the merger. With this, Oishi and Sugihara [9] improved the speed and effectiveness of the divide-and-conquer algorithm using the topological direction of the elements in the Voronoi diagram. Divide-and-conquer is relatively efficient, but its recursive calls require considerable amounts of computer memory. Moreover, the frequency of exceptional situations arising from divide-and-conquer algorithms is higher than with other approaches, and it is more difficult to implement completely when merging sub-diagrams by constructing split chains along boundary lines.

Indirect algorithms mainly rely on the fact that Voronoi diagrams and Delaunay triangulated networks have a dual relationship. First, a Delaunay triangulated network is formed on which the subsequent Voronoi diagrams are constructed. The efficiency of this process depends on the time complexity involved in generating the Delaunay triangulated network. A number of researchers have studied the construction and properties of Voronoi diagrams, leading to the proposal of algorithms such as the convex distance function [10], the recursive algorithm [11], and the discrete Voronoi diagrams algorithm [12, 13].

When the data scale for computer processing is increased, it becomes increasingly difficult for existing methods to meet the demands of the applications. Thus, designing a more efficient calculation method for large sets of spatial data has become the focus of research. The development of modern computers with multi-core, distributed technology and other features makes it possible to construct Voronoi diagrams using a parallel computing strategy. To date, there are relatively few approaches that have used parallel computing technology to construct Voronoi diagrams. Cole and Goodrich [14] combined the divide-and-conquer algorithm with parallel computing, and Kühn. (2001) used a convex distance function to implement a randomized parallel algorithm [15]. Yet, constrained by software and hardware limitations, most existing parallel algorithms focus on theoretical analyses. Besides, the scale of test data is not large enough to develop practical algorithms, and some approaches are difficult to apply because of computational constraints. Additionally, the existing parallel algorithms are usually on the basis of a divide-and-conquer algorithm using recursive grouping. It is difficult for a space partition method to control the shape of each group, and it is therefore likely that a long, narrow band will be formed, generating a large number of boundary points when the sub-groups are merged. Moreover, the number of polygons to be merged in each sub-diagram is also quite large, which affects, to a certain extent, the overall efficiency of the algorithm.

On the basis of this analysis of the divide-and-conquer algorithm, we propose splitting the point set into adaptive grids in the plane, so that each group has an approximately equal number of points and the rectangular space of each group has a better form (avoiding the problems posed by the long narrow band). In addition, this paper incorporates the advantages of the sweepline algorithm, in terms of time complexity, performance, and robustness in constructing Voronoi diagrams. As such, our method uses the sweepline algorithm to construct Voronoi diagrams in each sub-group and applies a parallel computing method to complete this process. Using the idea of the beach line in the sweepline algorithm, we extracted the boundary point from between adjacent grids, constructed Voronoi diagrams for the boundary point set, and updated the edge of the polygon corresponding to the original diagram to complete the merger of polygons.

2. VORONOI DIAGRAMS AND FORTUNE'S SWEEPLINE ALGORITHM

The Voronoi diagram of a point set in two-dimensional Euclidean space is defined as follows: given point-set $\{P_i\}$, $i = 1, \ldots, n$, there exists an associated polygon set $\{V_j\}$, $j = 1, \ldots, n$ in which each polygon contains only one point in $\{P_i\}$. The distance from any point within the polygon V_j to P_j is shorter than the distance to any other point in $\{P_i\}$. This can be written as:

$$V_j = \left\{P : \left|P - P_j\right| \leqq \left|P - P_i\right|, P_i \neq P_j\right\}$$

As shown in Figure 1, Voronoi diagrams consist of a group of adjacent polygons whose edges are the perpendicular bisectors of two adjacent points (or *sites*). One important characteristic of Voronoi diagrams is that, if a vertex of any polygon is regarded as the center of a circle whose radius is the distance from that vertex to its site, then the sites of all related polygons lie on the circle.

Fortune [7] proposed a sweepline algorithm for constructing Voronoi diagrams. A vertical sweepline is used to sweep the region of the point set from left to right, building a parabola each time it crosses a site using the point and the sweepline (Figure 2). This is called a *site event*. As the sweepline moves to the right, more points are visited and a series of parabolas are built (Figure 2 (b)). The curve formed by merging the parabolas is called the beach line, and this represents the union of the parabolas closest to the sweepline (Figure 2(c)). The intersection of two parabolas lies on the perpendicular bisector of the two corresponding points. For the current beach line, if it intersects with a straight line that is perpendicular to the sweepline, there will only be one intersection.

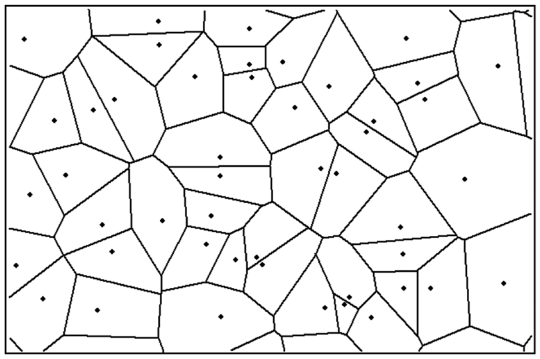

Figure 1. Voronoi diagrams and Voronoi polygons.

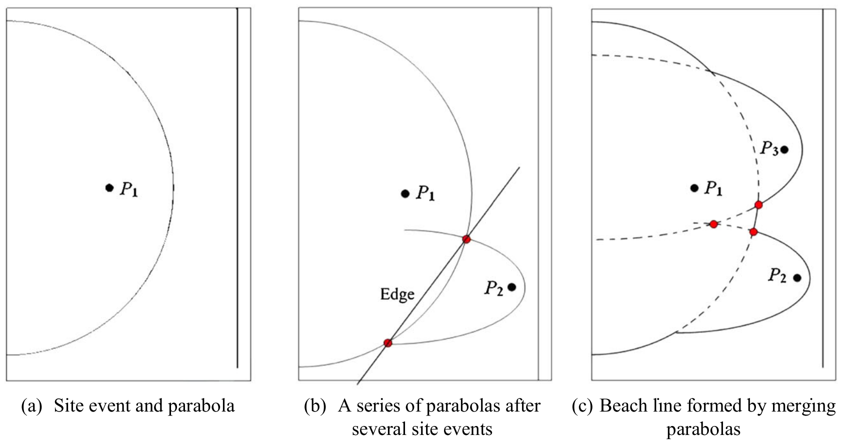
P_1
P_1
Edge
P_2
P_3
P_1
P_2
(a) Site event and parabola
(b) A series of parabolas after several site events
(c) Beach line formed by merging parabolas

Figure 2. Site events, parabolas, and the beach line.

The vertex of a Voronoi diagram is the center of the circle formed by three points (or more than three points) of a point set in the plane. As shown in Figure 3, when the sweepline crosses the region of the circle, the parabola associated with the three points changes dynamically with the moving sweepline. The segment of the parabola in the beach line built by the point furthest from the sweepline will become smaller. When the sweepline is tangential to the circle and the distance from the center of the circle to the sweepline equals the radius of the circle, the parabola built by the furthest point from the sweepline will reduce to one point. This one point is the vertex of the Voronoi diagram. This process, in which one arc of the beach line will be discarded, is called a *circle event*. The keys to this algorithm are as follows: (i) beach line data are maintained; (ii) the addition of a new arc or the discarding of an old arc is performed according to the site event and the circle event; and (iii) the beach line is updated, and a new vertex of the Voronoi diagram is added when an old arc is deleted.

As a classical vector approach, the sweepline algorithm performs better than other methods and has a time complexity of $O(n\log(n))$. In the sweepline algorithm, the Voronoi diagram is dynamically constructed by the continuous search for new vertices according to the site events and

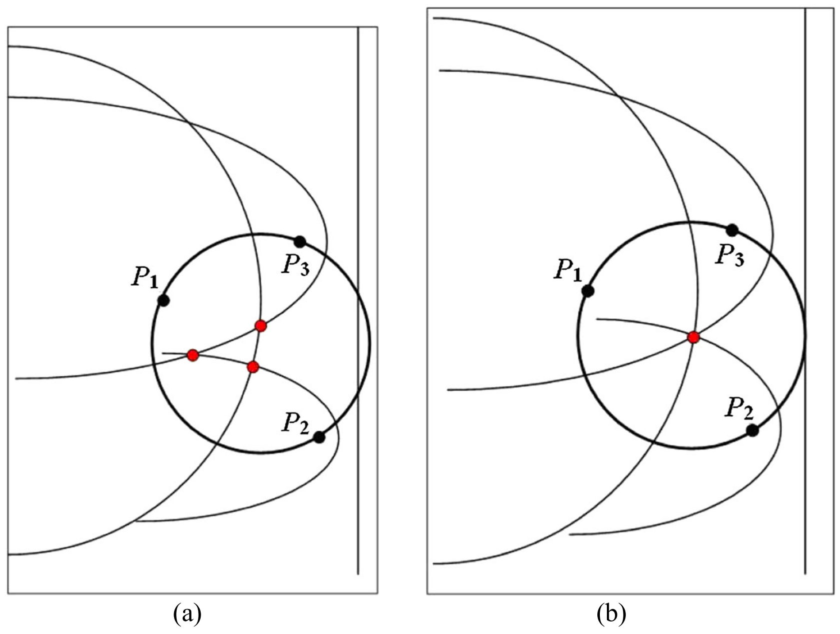

Figure 3. Sweepline and circle event.

circle events. For a point set in the plane, the algorithm is not affected by the direction of sweeping, which can be started from the top, bottom, left, or right. It should be noted that there are rarely computational or topological errors during the construction of a Voronoi diagram. The divide-and-conquer algorithm, which also has a good performance, is less stable than the sweepline algorithm. Divide-and-conquer also involves a large number of recursive operations, requiring substantial computer memory. Additionally, the strategy of splitting a point-set space directly affects the performance of the divide-and-conquer algorithm because, during the merging of sub-diagrams, calculation exceptions and topological errors are common, making the algorithm relatively complex. With the previous factors, this paper uses the sweepline algorithm as the basis for the parallel computation of a Voronoi diagram.

3. ADAPTIVE GROUPING OF A POINT SET IN THE PLANE

In general, the sweepline algorithm has a time complexity of $O(n\log(n))$, and its efficiency is directly related to the total number of points in the point set. As the number of points increases, the algorithm incurs a non-linear increase in time consumption. There is a more effective strategy of constructing a Voronoi diagram for a point set in the plane. To group a point set according to its spatial extent, we can use a sweepline algorithm to construct a Voronoi diagram for each group and then merge the sub-diagrams of each group. This divide-and-conquer strategy has been applied to many spatial analysis algorithms, such as the construction of a minimum convex hull and Delaunay triangulated networks. After the point set is divided, the search for nearby points and other processes is restricted to a small space, rather than the entire region used during the construction of the Voronoi diagram. This can help to improve the computational efficiency of the algorithm.

In existing divide-and-conquer algorithms, the point set is mainly divided layer by layer, horizontally or vertically using a dichotomy method. The overall Voronoi diagram is constructed by a recursive computation: (i) divide the point set into two parts; (ii) compute two sub-diagrams; and (iii) merge the sub-diagrams. It is not difficult to make the number of points in each subset approximately equal, but after dividing layer by layer, the spatial extent of the subset generally results in a narrow band that generates many incomplete Voronoi polygons at its edges. Consequently, a number of Voronoi polygons must be rebuilt during the merging process, which can not only reduce the efficiency of the algorithm but also lead to exceptional situations during the merger.

To avoid this narrow-band problem, as well as other issues with grouping the point set along the direction of a sweepline, this paper considers grouping the space into a gridded structure. As shown in Figure 4, the point set is assigned to a 6×9 grid. If separate Voronoi diagrams are constructed in each group, the final step is to merge all of the sub-Voronoi diagrams. This grouping strategy is similar to the common divide-and-conquer algorithm and can, in theory, improve the efficiency of the original algorithm. However, it should be noted that in practical applications, the spatial

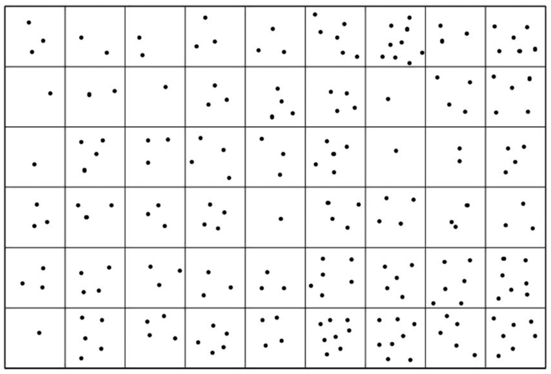
Figure 4. Grid-aided groups of points.

distribution of points is often uneven, which sometimes leads to a very different number of points in each grid cell, thus affecting the overall efficiency. This effect is particularly significant when the algorithm is implemented in parallel, as the different time consumption in each group unbalances the processor loads and reduces the overall computational performance.

Considering the spatial form of the point set, the balance of points, and other factors, this paper utilizes a multi-dimensional binary tree to design the following adaptive grouping scheme. In Figure 5, for example, we assume a minimum of no fewer than 20 points in each grid cell. The first step is to set the root node for the binary tree (Figure 6) to cover the original point set. Secondly, a horizontal line is used to divide the initial grid into two sections with an approximately equal number of points. Next, a vertical line is used to divide this sub-grid until the number of points in each cell is less than twice the threshold. Finally, we obtain the final leaf nodes. Whether the process uses a horizontal line or a vertical line for such divisions depends mainly on the length and width of the grid. Figure 6 shows the binary tree split for Figure 5. The process described previously can easily be implemented via a recursive method.

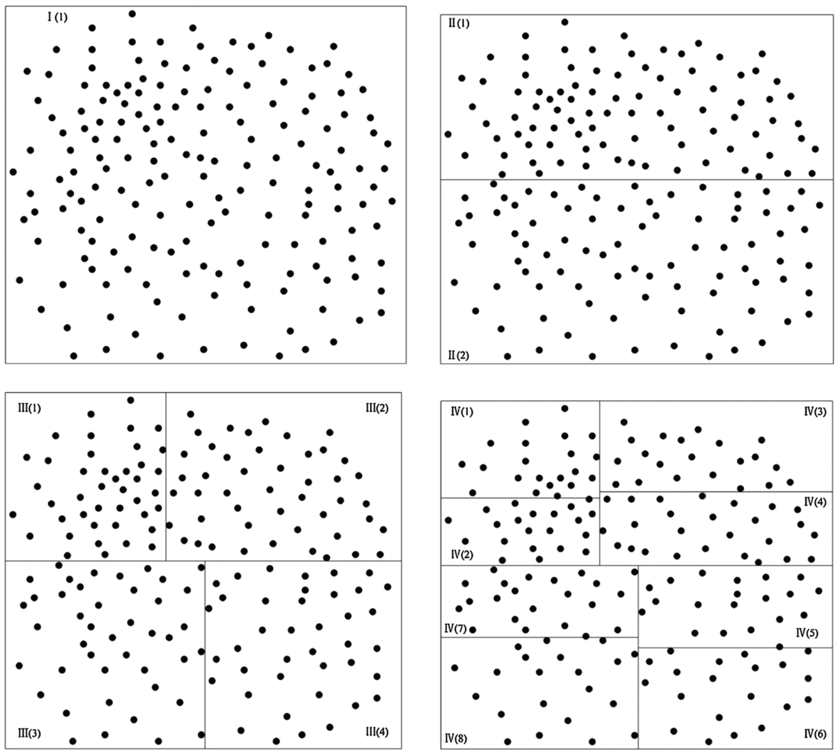

Figure 5. Adaptive group for points.

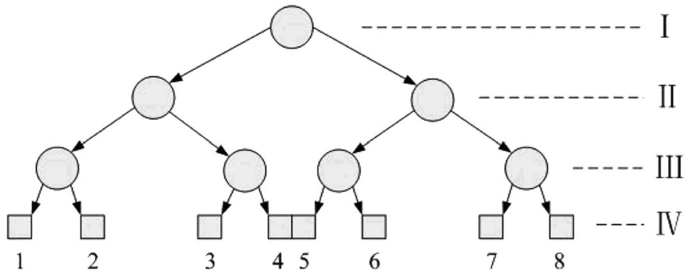

Figure 6. Binary tree and the point groups.

4. DATA STRUCTURE

Generally speaking, the data structure used in Fortune's algorithm can be directly applied to construct a Voronoi diagram in each sub-group. In this paper, the whole data set is dealt with as an 'adaptive grouping for a point set based on a binary tree' that, after being constructed, is merged into a complete Voronoi diagram. To meet the needs of the grouping algorithm and make it easy to process the data, the data structure used in Fortune's algorithm was adjusted and expanded to reflect improvements in computational efficiency. Specifically, the spatial elements are divided into four levels, from small to large, in the following manner: planar point set, Voronoi edge, Voronoi polygon, and binary tree block. Several intermediate structures of Fortune's algorithm were omitted in the interest of limiting the length of this paper.

In the structure mentioned previously, the binary tree block will be assigned a value only at the leaf node, and for non-leaf nodes, the *BlockSite* pointer will be null. This is because Voronoi diagrams are constructed at the leaf node, whereas the root node is only needed for the query and search and does not instantiate the *BlockSite* pointer.

In the Voronoi edge structure, a symbol *blsClip* has been designed to mark the edge intersecting with the block boundary, and this is used to quickly extract sites on the block boundary when merging the sub-diagrams. Figure 7 shows one site and its sub-Voronoi diagrams in a block. The bold lines are Voronoi edges that intersect the block boundary, and the points on either side of the edge are the sites of the block. The Voronoi polygons associated with the boundary sites are not closed and must therefore be merged and rebuilt with the Voronoi polygons in the block on the other side of the boundary (which are not closed either).

5. BOUNDARY SITE EXTRACTION AND MERGING OF SUB-DIAGRAMS

5.1. Extraction of boundary points

After the adaptive group has been formed, the sweepline algorithm is used to construct sub-Voronoi diagrams in each subset. The process of merging the sub-diagrams is the key to the algorithm. When a sub-diagram is being merged, the structure of the associated polygons will change locally; these changes include the update of the edges constructed by the polygons and the closure of the polygons. We call this kind of site a *boundary site*, and its corresponding polygons must be updated when a sub-diagram is merged. Sites corresponding to the incomplete polygons on the boundary of a sub-Voronoi diagram (*boundary Voronoi polygon*) must be boundary sites, and we call these *initial boundary sites*. The polygon closest to the boundary Voronoi polygon may also change when a sub-diagram is merged, and its site is also regarded as a boundary site. However, to distinguish it from the boundary sites defined earlier, we call this an *adjacent boundary site*. When sub-Voronoi

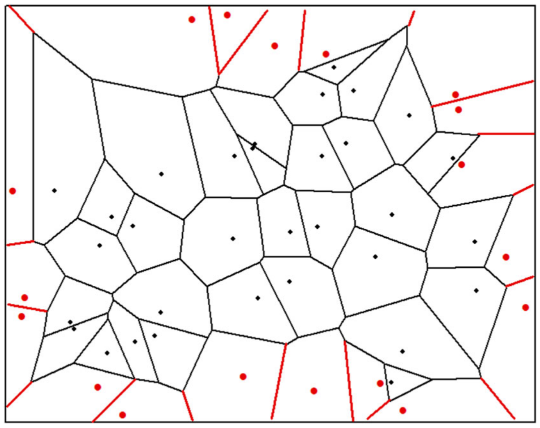

Figure 7. Voronoi edges and sites on boundaries.

diagrams are being constructed, their edges and intersecting boundaries are marked and recorded. The sites associated with the edges are the initial boundary sites, all of which can be easily extracted by traversing the edges of the sub-Voronoi diagram.

An adjacent boundary site can be extracted layer by layer, ‘skinning style’, by initializing the boundary sites with the help of the adjacent relationship of the Voronoi polygons. After the sub-diagrams are merged, only the Voronoi polygons associated with a small number of sites (plus the initial boundary sites) need updating. To improve the merger efficiency, we must ensure that a large amount of sites are not extracted blindly. In fact, the adjacent boundary sites are those that contribute to building the final beach line, that is, that are built when the sweepline arrives at the edge of the block, in the sweepline algorithm. As we explain in the paragraphs that follow, their parabolas must lie on the beach line.

Regardless of the type of scanning mode (from left to right, right to left, top to bottom, or bottom to top), the Voronoi diagram resulting from the sweepline algorithm is unique. If the sweepline is dragged to the block boundary (in this case the block boundary can also be regarded as a sweepline), each block will correspond to four sweeplines moving outside it. As the sweepline moves, the corresponding parabola will update and build a new beach line owing to the addition of new sites. As shown in Figure 8, the vertical line is the common boundary of two neighboring blocks. If the vertical line is regarded as the end of their respective sweeplines (the direction of the sweepline moving in the left block is from left to right and in the right block is from right to left), the final beach line of the left block is constructed by the parabolas generated by points P_1, P_2, and P_3, whereas that in the right block is generated by points Q_1, Q_2, and Q_3. Let us merge the two blocks and assume, on the basis of the beach line theory, that the sweepline of the left block extends to the right. The Voronoi vertices that have been computed on the left of the existing beach line in the left block will not be affected. Similarly, if the right grid extends to left and the sweepline moves left, the Voronoi vertices computed on the right of the beach line in the right block will not be affected. In other words, the polygon vertices, which are built by circle sites behind the final beach line in the two blocks, will not be affected by the merger. Additionally, the Voronoi vertices that must be changed because of the merging of sub-diagrams are only in the area outside the beach line. According to the sweepline algorithm, a Voronoi vertex is generated when a circle site occurs. The circle site is based on the beach line and the point on one side of the beach line, and the point on the other side of the sweepline forms the circle site. In this case, the point must be used to build the beach line. Thus, the point that contributes to building the final beach line is the boundary site that must be extracted when the sub-diagrams are being merged.

To extract all the boundary sites, the beach line formed when the sweepline moves to the boundary of a block must first be built. The extraction of initial boundary sites is relatively easy and is designed in this paper in the following way. First, the initial boundary point is extracted to build an initial beach line according to the principles of beach line theory. However, it is important to note that this beach line may not coincide with the final beach line. Next, we search for the adjoining point of the initial boundary point, that is, the site of the polygon adjacent to a Voronoi polygon associated with the

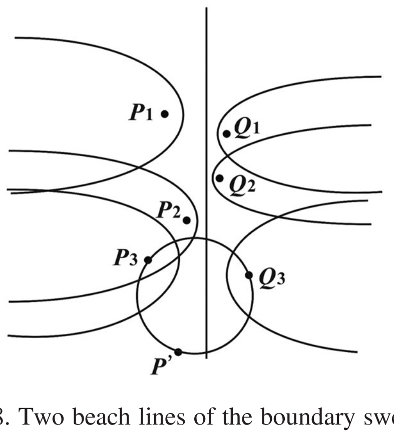

Figure 8. Two beach lines of the boundary sweepline.

initial boundary site. If this kind of site is used to build the parabola that intersects the initial beach line, then we must update the initial beach line. The site becomes the adjacent boundary site and is added to the set of boundary sites. The specific procedure is as follows:

1. Construct a queue and add the initial boundary sites. Mark each point that has been determined.
2. Traverse the boundary sites in the queue and extract adjacent points that have an edge in common with the sub-Voronoi diagram.
3. Traverse the adjacent points. If a point has already been determined, proceed to step 4. Otherwise, construct this point's parabola, compute the intersection between the parabola and the beach line, and, if there is an intersection, update the beach line, add the point to the queue, and mark the point as determined.
4. Stop when all boundary points in the queue have been visited and the point set in the queue is the boundary site corresponding to the beach line. In step 3, when a point adjacent to the boundary point cannot be used to update the beach line, then the search for the second adjacent point is terminated to prevent the extension of the boundary site. Otherwise, the extension will continue layer by layer.

5.2. *Merging of sub-diagrams*

Figure 9 shows the merger of sub-Voronoi diagrams in two adjacent blocks. The vertical line denotes the boundary between the blocks, and the jagged bold line is the new Voronoi edge created by merging the sub-diagrams. These new segments are the bisectors between the two points on either side of the original boundary. Part of the new Voronoi edge intersects the Voronoi polygons in the sub-diagram, which leads to some Voronoi vertices near the block boundary being abandoned. For example, vertex *O* shown in Figure 9 is the center of the circumcircle of sites *A*, *B*, and *C*. When the sub-diagrams are merged, the point on the right side of the sweepline and sites *A*, *B*, and *C* form three new perpendicular bisectors, which split the original bisectors to form a new Voronoi vertex and replace the original vertex *O*.

To merge sub-diagrams and find the new Voronoi edge (as shown in Figure 9), we have designed and utilized a method to update the global Voronoi diagram with the help of a boundary site Voronoi diagram. After the sub-Voronoi diagrams in each block cell are constructed, the boundary points of each block are extracted to form a set of boundary sites. The sweepline algorithm is then used to construct a Voronoi diagram for this set. Thus, for each boundary site, there are two corresponding Voronoi polygons, one from the sub-Voronoi diagram of each block and one from the boundary site Voronoi diagram. Although the two polygons have different shapes, the Voronoi edges within boundary points in each block remain unchanged. In the sub-diagram of each block, the polygon at the block boundary is usually open. In the boundary site Voronoi diagram, the sites are distributed regularly around the boundary. The polygons associated with the boundary sites and

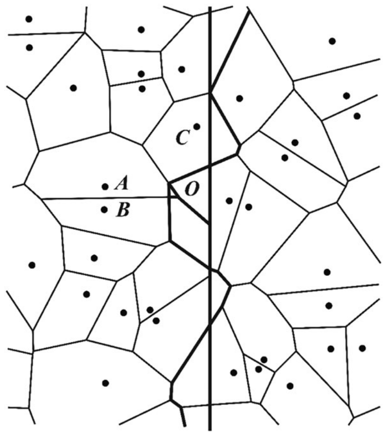

Figure 9. The merging of multi-group Voronoi polygons.

across the boundary are generally intact and closed (except for sites on the edge of the global set region). During the merger of sub-diagrams, the boundary site Voronoi diagram, after being merged, consists of the intersection of the previous two polygon regions. By simply computing the intersections, we found that the two polygons can update the boundary site Voronoi polygon. As shown in Figure 10, the horizontal line is the boundary of two subsets, and the two continuous parabolas denote the beach line corresponding to this boundary. The continuous folder between the two beach lines is necessary for merging the sub-diagrams. In fact, the folder can be built directly during the construction of the boundary site Voronoi diagram. In the sub-diagram containing boundary site *O*, the Voronoi polygon contains *FABC*, where *AF* is radial, *AB* is a segment, and *BC* is radial. In the boundary site Voronoi diagram, the Voronoi polygon contains *BCDEF*, where *CDEF* is the folder between beach lines. The merger of the two polygons is relatively simple, with the main task being to find the intersection of the polygons and construct a new closed polygon. This new polygon, of which the common segments need not be modified, only requires the intersection of its radials to be computed to ensure it is closed.

In our algorithm, we can avoid the slightly complex process used to compute the intersection of polygons by applying a relatively simple merger method to replace the boundary polygon in the sub-diagram directly. The details of this method are as follows. First, to extract the boundary site from the sub-diagram, we extract its second and third neighbor points inside the border to build a point set containing all boundary points and neighbor points. Note that the Voronoi diagram corresponding to the boundary point is the final desired polygon, the shape of which is not affected by any further sites. Next, we use the Voronoi polygon of the boundary point in this diagram to replace the polygon in the sub-diagram.

6. PARALLEL ALGORITHM

In recent decades, parallel computing theories for numerical calculations and non-numerical computing have undergone rapid development. As a result, a variety of parallel computing models, such as PRAM, BSP, and LogP, have been proposed in which the commonly used parallel algorithms fall into two types: single instruction stream multiple data stream (SIMD) and multiple instruction stream multiple data stream. In SIMD, each processor carries out the same instruction simultaneously using different data, resulting in a parallelism in the data process. With multiple instruction stream multiple data stream, each processor carries out different instructions and separately processes different data, which is a parallel processing of the algorithm.

The parallel algorithms used to construct Voronoi diagrams for a point set in two-dimensional space have been researched. For example, Cole and Goodrich [14] provide *n* processors for a recursive method with a point set of *n* points. Given a vertical line, Cole's algorithm divides the point set in the plane into two groups of *n*/2 points, each of which is assigned *n*/2 processors. Next, the 'contour lines' that cross the vertical line on both sides are computed upon each execution of the algorithm. When the recursion ends, the contour lines are merged. Kühn (2001) proposed a stochastic parallel algorithm based on the symmetric convex

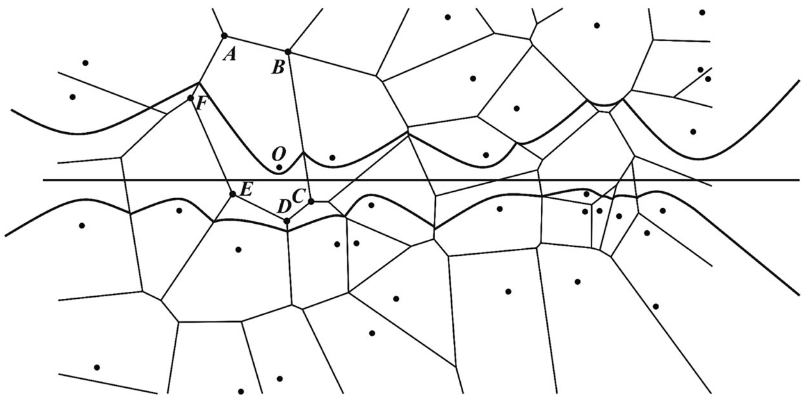

Figure 10. The merging of multi-group Voronoi polygons using a boundary site Voronoi diagram.

distance function, which uses random sampling data to construct Voronoi diagrams with a fixed number of processors [15]. Existing algorithms to construct Voronoi diagrams based on parallel technologies are divided into two categories. The first assigns a processor after determining each group, and the second fixes the number of processors and varies the number of groups. In this paper, we use the latter to complete the parallel computation based on a multi-core processor to group the point set in the plane.

The parallelization of our proposed algorithm is based on an adaptive group in a point set. This is a typical SIMD algorithm in which each processor works with the same content but uses different data in each group. The algorithm refers to grouping a point set, extracting the boundary point to construct a Voronoi diagram, and merging the sub-Voronoi diagrams one-by-one. A host processor, responsible for the adaptive group, distributes group data to slave processors, from which it later receives the results. The merging of sub-Voronoi diagrams can be determined among the processors. The slave processors construct the sub-Voronoi diagrams with the point set assigned by the host processor and extract the boundary points in that group. The detailed process is as follows:

1. The host processor forms an adaptive group of point sets in the plane according to a block parameter and the number of groups N.
2. For M processors, the host assigns a point set to each slave processor one-by-one and monitors the results.
3. When a slave processor completes its computations, it sends the results to the host processor. If there are any remaining unprocessed point sets, the host processor assigns a point set to the slave processor.
4. The previous steps are repeated until the host processor finishes processing all point sets.
5. The host processor constructs sub-Voronoi diagrams for the boundary sites according to those constructed by the point sets in each group and the extracted boundary points.
6. The merger of the previous $N+1$ sub-Voronoi diagrams is then completed.

7. ANALYSIS AND TESTING

The adaptive group parallel algorithm proposed in this paper improves the computational efficiency of the traditional sweepline algorithm from the following two aspects. First, by dividing the point set (with N points) evenly into K groups, we reduce the average size of the point set to $M = N/K$ and the time complexity to $K \times O(M\log(M))$, namely $O(N\log(M))$. Secondly, by adopting a parallelization algorithm, we construct the Voronoi diagrams for each subset simultaneously. If the number of processors is P and $K > P$, the time complexity is reduced to $O((N/P)\log(M))$.

Using a group computing strategy, our proposed algorithm comprises four main steps: (i) grouping the point set, (ii) constructing sub-Voronoi diagrams, (iii) constructing boundary site Voronoi diagrams, and (iv) merging the sub-Voronoi diagrams (Table I). Among these four steps, the adaptive grouping of the point set only requires some simple calculation by traversing the point set in time $O(N)$. The construction of sub-Voronoi diagrams can be completed simultaneously for each group, although it is still the most time consuming step. The time complexity for constructing boundary site Voronoi diagrams is $O(N_B\log(N_B))$, where N_B is the total number of boundary points

Table I. Results of efficiency test.

Number of points (×10 000)	Time taken for four steps (s)				Single processor	Eight processors	Fortune's algorithm
	Adaptive grouping	Constructing sub-Voronoi diagrams (eight groups)	Constructing boundary site Voronoi diagrams	Merging the sub-Voronoi diagrams			
200	0.844	2.12	0.21	0.11	18.124	3.284	23.36
100	0.422	1.10	0.095	0.03	9.347	1.647	10.95
80	0.313	0.84	0.074	0.03	7.137	1.257	8.46
50	0.203	0.51	0.050	0.02	4.353	0.783	4.85
20	0.031	0.17	0.027	0	1.418	0.228	1.75
10	0.022	0.07	0.017	0	0.599	0.109	0.84

determined by the number of groups K. The greater the number of blocks, the longer the total boundary, and therefore the greater the number of boundary points. The boundary points that need to be extracted from each sub-Voronoi diagram represent the outermost three layers of Voronoi points, so the number of boundary points in each block is $3 \times 4 \times M^{1/2}$ and the total number of boundary points is $N_B = 12KM^{1/2} = 12NM^{-1/2}$. Thus, the time complexity of this step is $O(12NM^{-1/2}\log(12NM^{-1/2}))$. To merge sub-Voronoi diagrams, it is essential to use the Voronoi diagrams of the boundary point set to update the boundary site Voronoi diagrams. The computation time of this step is relatively short whether we implement polygon intersection or directly replace the Voronoi polygons in the sub-Voronoi diagrams.

Considering that the construction of the sub-Voronoi diagrams and the boundary site Voronoi diagrams accounts for the majority of the time, the complexity of this algorithm can be expressed as $O((N/P)\log(M)) + O(12NM^{-1/2}\log(12NM^{-1/2}))$. To minimize the computation time, the average number of points in each group M should be approximately equal to the total number of boundary points N_B, when $K = (N/12)^{1/3}$. For instance, if there are 12 million points divided into 100 groups, then the average number of points in each group is 12 0000, and the total number of boundary points is also approximately 12 0000, only 1% of the total. In this case, even without parallel computing ($P = 1$), the construction of sub-Voronoi diagrams and boundary site Voronoi diagrams consumes no more than 1% of the total time using the conventional sweepline algorithm, whose time complexity is $O(n\log(n))$.

8. CONCLUSION

This paper has analyzed the advantages and disadvantages of vector and raster methods in the construction of Voronoi diagrams and proposed an adaptive group method based on a binary tree. Our algorithm is based on Fortune's sweepline algorithm and utilizes block adaptive grouping to improve the efficiency of the algorithm. The adaptive group method groups the point set in the plane and uses parallel computing to further reduce the computational load. As the construction of Voronoi diagrams in each group is independent, it can be easily implemented in a parallel computing environment. For large amounts of data, parallel computing can effectively reduce the memory and disk space limit of the algorithm in a stand-alone environment.

ACKNOWLEDGEMENTS

This work is supported by the project Priority Academic Program Development of Jiangsu, the National Natural Science Foundation of China (Grant No. 41001238), and the project supported by the National Science and Technology Ministry (ID: 2012BAH28B02, 2012BAH28B04-05).

REFERENCES

1. Okabe A, Boots B, Sugihara K, Chiu SN. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley: Chichester, 2000.
2. Ryu J, Park R, Kim DS. Molecular surfaces on proteins via beta shapes. *Computer-Aided Design* 2007; **39**(12): 1042–1057.
3. Li JT, Zhao RL, Chen J. An integrated method of raster and vector for generating Voronoi diagram based on linear quadtree structure. *Remote Sensing of the Environment: 15th National Symposium on Remote Sensing of China* (Proceedings Volume), Proc. SPIE, Vol. **6200**, 2006.
4. Chen J. A raster-based method for computing Voronoi diagrams of spatial objects using dynamic distance transformation. *International Journal of Geographical Information Science* 2001; **13**(3): 209–225.
5. Zhao RL, Li ZL, Chen J, Gold CM, Zhang Y. A hierarchical raster method for computing Voronoi diagrams based on quadtrees. *Lecture Notes in Computer Science* 2002; **2331**: 1004–1013.
6. Martin H, Stevan H. Topology-oriented incremental computation of Voronoi Diagrams of circular arcs and straight line segments. *Computer-Aided Design* 2009; **41**(5): 327–338.
7. Fortune S. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 1987; **2**: 153–174.
8. Shamos MI, Hoey D. Closest-point problems. *Proceedings of the 16th Annual Symposium on Foundations of Computer Science* 1975; 151–162.
9. Oishi Y, Sugihara K. Topology-oriented divide-and-conquer algorithm for Voronoi diagrams. *Graphic Models and Image Proceedings* 1995; **57**(4): 303–314.

10. Chew LP, Dyrsdale RL. Voronoi diagrams based on convex distance functions. *Proceedings of the Symposium on Computational Geometry*, Baltimore, 1985; 235–244.
11. Mark DM. Recursive algorithm for determination of proximal (Thiessen) polygons in any metric space. *Geographical Analysis* 1987; **19**(3): 264–272.
12. Schwarzkopf O. Parallel computation of discrete Voronoi diagrams. *Lecture Notes in Computer Science* 1989; **349**: 193–204.
13. Schueller A. A nearest neighbor sweep circle algorithm for computing discrete Voronoi tessellations. *Journal of Mathematical Anaysis and Applications* 2007; **336**(2): 1018–1025.
14. Cole R, Goodrich MT. A nearly optimal deterministic Paranell Voronoi diagram algorithm. *Algorithmica* 1996; **16**: 569–617.
15. Kühn U. A Randomized Parallel algorithm for Voronoi diagrams based on symmetric convex distance functions. *Discrete Applied Mathematics* 2001; **109**(1–2): 177–196.