Algorithmica
DOI 10.1007/s00453-012-9704-9

Computing the Map of Geometric Minimal Cuts

Jinhui Xu · Lei Xu · Evanthia Papadopoulou

Received: 23 March 2010 / Accepted: 18 October 2012

Abstract In this paper we consider the following problem of computing a map of geometric minimal cuts (called MGMC problem): Given a graph $G=(V,E)$ and a planar rectilinear embedding of a subgraph $H=(V_H,E_H)$ of G, compute the map of geometric minimal cuts induced by axis-aligned rectangles in the embedding plane. The MGMC problem is motivated by the *critical area* extraction problem in VLSI designs and finds applications in several other fields. In this paper, we propose a novel approach based on a mix of geometric and graph algorithm techniques for the MGMC problem. Our approach first shows that unlike the classic min-cut problem on graphs, the number of all rectilinear geometric minimal cuts is bounded by a low polynomial, $O(n^3)$. Our algorithm for identifying geometric minimal cuts runs in $O(n^3\log n(\log\log n)^3)$ expected time which can be reduced to $O(n\log n(\log\log n)^3)$ when the maximum size of the cut is bounded by a constant, where $n=|V_H|$. Once geometric minimal cuts are identified we show that the problem can be reduced to computing the L_∞ Hausdorff Voronoi diagram of axis aligned rectangles. We present the first output-sensitive algorithm to compute this diagram which runs in $O((N+K)\log^2 N\log\log N)$ time and $O(N\log^2 N)$ space, where N is the number

The research of J. Xu and L. Xu was supported in part by NSF through a CAREER Award CCF-0546509 and grants IIS-0713489 and IIS-1115220. The research of E. Papadopoulou was supported in part by the Swiss National Science Foundation grant 200021-127137.

J. Xu (✉) · L. Xu
Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260, USA
e-mail: jinhui@buffalo.edu

L. Xu
e-mail: lxu@buffalo.edu

E. Papadopoulou
Faculty of Informatics, Università della Svizzera Italiana, Lugano 6904, Switzerland
e-mail: evanthia.papadopoulou@usi.ch

of rectangles and K is the complexity of the Hausdorff Voronoi diagram. Our approach settles several open problems regarding the MGMC problem.

Keywords Geometric minimal cut · Hausdorff Voronoi diagram · Output sensitive algorithm · Plane sweep algorithm · Computational geometry

1 Introduction

In this paper, we consider the following problem (called Map of Geometric Minimal Cuts or MGMC problem): Given a graph $G = (V, E)$ and a planar embedding of a subgraph $H = (V_H, E_H)$ of G, compute a map[1] $\mathcal{M}$ of the embedding plane P of H so that for every point $p \in P$, the cell in $\mathcal{M}$ containing p is associated with the "closest" *geometric cut* (in G) to p, where distance between a point p and a cut C is defined as the Hausdorff distance between p and the minimum inducing region $R(C)$ of C (exact definitions of $R(C)$ is given in next section). A geometric cut C of G induced by a given geometric shape S is a set of edges and vertices in H that overlaps S in P and whose removal from G disconnects G. In this paper we consider the case where geometric cuts are induced by axis-aligned rectangles and distances are measured in the L_∞ metric. The main objective of the MGMC problem is to compute the map $\mathcal{M}$ of all geometric minimal (or canonical) cuts (the exact definition of geometric minimal cuts will be given in next section) of the planar embedding of H.

The MGMC problem is motivated by the VLSI critical area computation problem as explained in [20, 21]. Critical area is a measure reflecting the sensitivity of a design to random particle defects during the manufacturing process and its extraction is essential in today's VLSI manufacturing. The critical area problem for various types of faults can be reduced to different variants of Voronoi diagrams that lead to accurate critical area extraction [18, 22, 23]. A VLSI net N can be modeled as a graph $G = (V, E)$ with a subgraph embedded on every conducting layer. The subgraph $H = (V_H, E_H)$ on a layer X is vulnerable to random defects associated with this layer. Defects on layer X may create *cuts* on graph G that result in disconnecting the net N. The Voronoi framework for critical area extraction in the case of open faults asks for a subdivision of layer X into regions that reveal for every point p the radius of the smallest disk centered at p inducing a cut of G, which can be modeled as an MGMC problem. This subdivision corresponds to the *Hausdorff Voronoi diagram* of geometric minimal cuts [20, 21]. The MGMC problem also finds applications in other networks (in the L_∞ metric), such as transportation networks, where critical area may need to be extracted for the purpose of flow control and avoiding disaster.

In this paper we present a novel approach to solve the MGMC problem in the L_∞ metric for a rectilinear embedding of H based on a mix of geometric and graph algorithm techniques. We first classify geometric cuts into two classes: 1-D cuts and 2-D cuts, and show that the number of all possible geometric 1-D and 2-D minimal

[1] A map means a partition of the embedding plane (as in trapezoidal map) into cells so that all points in the same cell share the same "closest" geometric minimal cut.

cuts is $O(n^2)$ and $O(n^3)$ respectively. Directly computing the geometric minimal cuts could take $O(n^3(N+M))$ time, where $n = |V_H|$, $N = |V|$, and $M = |E|$. Based on interesting observations and dynamic connectivity data structures, we show that the time complexity can be reduced to $O(n^3 \log n (\log\log n)^3)$ (expected time). We also consider the case in which the inducing rectangle of a cut has a constantly bounded edge length. For this case, we show that the time complexity of our algorithm can be significantly improved to $O(n \log n (\log\log n)^3)$. Once all geometric minimal cuts are identified, the solution to the MGMC problem can be reduced to computing their Hausdorff Voronoi diagram.

To compute the Hausdorff Voronoi diagram of the geometric minimal cuts, we revisit the plane sweep construction of the L_∞ Hausdorff Voronoi diagram of rectangles. The Hausdorff Voronoi diagram of polygonal objects (e.g., point clusters or polygons) has been studied in [1, 3, 5, 18, 19, 24]. Given a set S of rectangles in the plane, the L_∞ Hausdorff Voronoi diagram is a subdivision of the plane into regions that reveal for every point p its "closest" rectangle, where the distance between a point p and a rectangle R is measured by the maximum L_∞ distance between p and R, or equivalently by the L_∞ Hausdorff distance between p and R. For the construction of the L_∞ Hausdorff Voronoi diagram, we present the first output-sensitive algorithm which runs in $O((N+K)\log^2 N \log\log N)$ time and $O(N \log^2 N)$ space, where N is the number of rectangles and K is the complexity (or size) of the Hausdorff Voronoi diagram. Previous results on the construction of the Hausdorff Voronoi diagram are either $O(n^2)$ [5] or depend on some parameters, such as crossing and interacting pairs of rectangles [18, 19], which could be significantly larger than the actual size of the diagram in the worst case.

Our approach settles several open questions regarding the MGMC problem and it can be viewed as a complement of the approach followed in [20, 21] which is based on higher order Voronoi diagrams. The $O(n^3)$ bound on the number of geometric minimal cuts that is shown in this paper makes the enumeration of geometric cuts feasible. In [21] geometric cuts are not pre-computed but they are determined on the fly based on an iterative construction of order-k Voronoi diagrams, which results in the Hausdorff Voronoi diagram of geometric cuts. The graph embedding in [21] reflects arbitrary polygons. The connectivity data structures we employ to efficiently compute the geometric minimal cuts can also be used to speed up connectivity queries in [20, 21].

The rest of this paper is organized as follows. Section 2 introduces the concepts of 1-D and 2-D geometric minimal cuts. Section 3 presents our algorithms for computing the geometric minimal cuts. The algorithm for computing the Hausdorff Voronoi diagram is given in Sect. 4.

2 Geometric Cuts

Let $G = (V, E)$ be the undirected graph in an MGMC problem and let $H = (V_H, E_H)$ be a planar subgraph embedded in the plane P with $|V| = N$, $|E| = M$, $|V_H| = n$, and $|E_H| = m$. Due to the planarity of H, $m = O(n)$. In this paper, unless explicitly mentioned otherwise, we assume that all edges in H are either horizontal

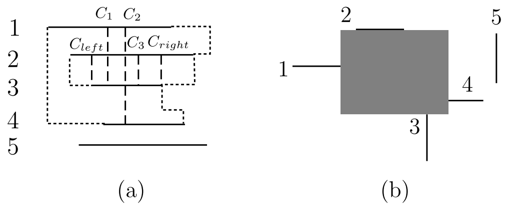

Fig. 1 (**a**) 1-D cuts $C_1 = \{1,2,3\}$, $C_2 = \{1,2,3,4\}$, $C_3 = \{2,3\}$ with C_3 being the minimal cut. (**b**) $R(C)$ is bounded by a 2-D cut with 4 edges $\{1,2,3,4\}$

or vertical straight line segments. A pair of vertices u and v in a graph is connected if there is a path in this graph from u to v. Otherwise, they are disconnected. A graph is connected if every pair of its distinct vertices is connected. Without loss of generality (WLOG), we assume that G is connected. A cut C of G is a subset of edges in G whose removal disconnects G. A cut C is minimal if removing any edge from C no longer forms a cut.

Definition 1 Let R be a connected region in P, and $C = R \cap H$ be the set of edges in H intersected by R. C is called a geometric cut of G induced by R if the removal of C from G disconnects G.

When there is no ambiguity of the region R, we often call the cut induced by R as a geometric cut for simplicity. For a given cut C, its *minimum inducing region* $R(C)$ is the minimum axis-aligned rectangle which intersects every edge of C (i.e., if we shrink the width or length of $R(C)$ by any arbitrarily small value ϵ, some edge in C will no longer be intersected by $R(C)$).

Definition 2 A geometric cut C is a geometric minimal cut if the set of edges intersected by any region shrinking from $R(C)$ no longer forms a geometric cut.

For some geometric cut C, its $R(C)$ could be degenerated into a horizontal or vertical line segment, or even a single point. It is easy to see that if $R(C)$ is not degenerated, it is unique.

Definition 3 A geometric cut C is called a 1-D geometric cut (or a 1-D cut) if $R(C)$ is a segment. If $R(C)$ is an axis-aligned rectangle, then C is called a 2-D geometric cut (or a 2-D cut).

The following lemma easily follows from the above definitions and the problem setting.

Lemma 1 *Let C be any geometric minimal cut. If C is a* 1*-D cut, then each endpoint u of $R(C)$ is incident with either an endpoint of an edge e in H with the same orientation as $R(C)$ and $e \cap R(C) = u$ or an edge in H with different orientation as $R(C)$* (*see Fig.* 1(a)). *If C is a* 2*-D cut, each bounding edge s of $R(C)$ is incident with either an endpoint v of an edge e in H of different orientation with s and $R(C) \cap e = v$ or an edge in H with the same orientation as s* (*see Fig.* 1(b); *Note that s could be incident with multiple edges*).

From the above lemma, it is clear that each 1-D geometric minimal cut is bounded by up to two edges in H and each 2-D geometric minimal cut is bounded by up to 4 edges in H. For a cut C, we let $B(C)$ denote the set of edges bounding $R(C)$. Due to the minimality nature of C, removing any edge in $B(C)$ will lead to a non-cut. This means that any edge in $B(C)$ is necessary for forming the cut. However, this is not necessarily true for edges in $C \setminus B(C)$. Thus, a geometric minimal cut may not be a minimal cut. Note that for a given graph, the number of minimal cuts could be exponential. However, as we will show later, the number of geometric minimal cuts is much less (i.e., bounded by a low degree polynomial).

For a 1-D geometric minimal cut C, it is possible that all edges in C have different orientation with $R(C)$. In this case, there may exist an infinite number of 1-D geometric minimal cuts, all cutting the same set of edges C (in other words, the minimum inducing region for such a 1-D geometric C is not unique). For example, if we slide $R(C)$ along $B(C)$, we will obtain an infinite number of 1-D minimal cuts. Among these cuts, there are two extreme cuts, C_{left} and C_{right} (or C_{top} and C_{bottom} if $R(C)$ is horizontal), incident with the left and right (or up and bottom) endpoints of some edges in C respectively. Since all these geometric cuts cut the same set of edges, we count them as one cut.

3 Identifying Geometric Minimal Cuts

To solve the MGMC problem, our main idea is to first identify all possible geometric minimal cuts and then construct a Hausdorff Voronoi diagram of these cuts as the map $\mathcal{M}$. Thus three major problems need to be solved:

1. How to find all 1-D geometric minimal cuts;
2. How to find all 2-D geometric minimal cuts;
3. How to efficiently construct the Hausdorff Voronoi diagram for the geometric cuts.

In this section, we discuss our main ideas for problems 1 and 2. In next section, we will discuss our approach for problem 3.

3.1 Computing 1-D Geometric Minimal Cuts

As discussed in last section, a 1-D geometric minimal cut can be induced by either horizontal or vertical segments. The following lemma upper bounds the total number of 1-D geometric minimal cuts.

Lemma 2 *There are $O(n^2)$ 1-D geometric minimal cuts in H.*

Proof We consider only the 1-D cuts induced by vertical segments. Cuts induced by horizontal segments can be dealt with similarly. If we place a vertical line through each vertex (or endpoint), then the plane P is partitioned into $O(n)$ vertical slabs, with each slab containing no endpoint in its interior. For a particular slab S containing k edges (an edge need not be fully contained in S), say $e_1, e_2, \ldots, e_k$, we claim that there are at most $O(k)$ 1-D geometric minimal cuts. To see this, we consider all 1-D minimal cuts formed by the k edges. We say that a cut C is owned by an edge e_i if $e_i \in B(C)$. Clearly, each 1-D minimal cut has at least one owner. Now consider each edge e_i, it can only be the owner of up to two 1-D minimal cuts, one bounded by e_i from the bottom and one bounded by e_i from the top. By slab decomposition, the longer cut bounded by e_i from the top contains all edges of a possibly shorter cut of this type. Due to the fact that the cuts are all minimal, it is impossible to have two cuts bounded by e_i from the top (or bottom) simultaneously. Thus each edge in S owns at most two 1-D geometric minimal cuts. Hence, the total number of 1-D geometric minimal cuts in S is $O(k)$. Summing over all slabs, we know that the total number of 1-D geometric minimal cuts is $O(n^2)$. Thus the lemma follows. □

To compute the $O(n^2)$ 1-D geometric minimal cuts, we observe that each cut is a set of edges $C \in H$ which appear consecutively in some slab and whose removal disconnects G. Thus we need to first (a) identify all possible cuts in H and then (b) for each possible cut, determine whether it is indeed a cut (such a test is called a *cut query*). To overcome the two difficulties, a straightforward way is to enumerate all possible cuts, and for each possible cut C, use graph search algorithms (such as depth first search (DFS) or breadth first search (BFS)) to check the connectivity of $G \setminus C$. As shown in the proof of Lemma 2, the embedding plane P can be partitioned into $O(n)$ slabs and for each slab with k edges, there are $O(k^2)$ subsets of consecutive edges. Thus a total of $O(n^3)$ possible subsets need to be checked and the connectivity checking for each subset takes $O(N + M)$ time. Therefore this will lead to a total of $O(n^3(N + M))$ time.

To speed up the computation, we first simplify the graph G. We observe that the cuts involve only the edges in H. The connectivity in $G \setminus E_H$ will not be affected no matter which subset of edges in H is removed. To make use of this invariant, we first compute the connected components of $G \setminus E_H$. For each connected component CC, we contract it into a supernode v_{CC}. For each vertex $u \in H$, if there was an edge connecting u and a vertex in CC, we add an edge (u, v_{CC}). Let the resulting graph be $G' = (V', E')$ (called contracted graph). The following lemma gives some properties of this graph.

Lemma 3 *The number of vertices in G' is $|V'| = O(n)$ and the number of edges in G' is $|E'| = O(n)$. Furthermore, a subset of edges in H is a cut of G if and only if it is a cut of G'.*

Proof Follows from the construction of G'. □

From the above lemma, we know that the size of G' could be much smaller than G. Thus the time needed for answering a cut query is significantly reduced from $O(N + M)$ to $O(n)$.

Converting Cut Queries to Connectivity Queries As discussed previously, to compute all 1-D geometric minimal cuts we have to check $O(n^3)$ possible subsets of edges in H. Many of them are quite similar (i.e., differ only by one or a small number of edges). Thus it would be highly inefficient if we handle them independently and answer each cut query by search the graph G'. To yield a better solution, we have to consider all these cut queries as a whole and try to share the connectivity information among similar cut queries.

To share information among slightly different cut queries, one possible way is to use persistent data structure [4]. However, due to the fact that inserting (or removing) an edge could cause a linear, instead of a constant, number of 1-D geometric minimal cuts to be destroyed or generated, persistent data structures require substantial amount of updating after each insertion or deletion, thus do not lead to highly efficient solution.

To overcome this difficulty, our main idea is to further decompose each cut queries into a set of connectivity queries with each connectivity query involving only one edge. The reduced granularity in query enables us to better share information among related cut queries.

Connectivity Query Before continuing our discussion on connectivity queries, we first briefly review some existing algorithms and data structures for the dynamic connectivity problem. In the fully dynamic connectivity problem, the input is a graph G whose vertex set is fixed and whose edges can be inserted and deleted. The objective is to construct a data structure for G so that the connectivity of any pair of vertices can be efficiently determined. Most fully dynamic connectivity data structures support the following three operations: (1) $\mathit{Insert}(e)$, (2) $\mathit{Delete}(e)$, and (3) $\mathit{Connectivity}(u, v)$, where the $\mathit{Insert}(e)$ operation inserts edge e into G, the $\mathit{Delete}(e)$ operation removes edge e from G, and the $\mathit{Connectivity}(u, v)$ operation determines whether u and v are connected in the current graph G. Extensive research has been done on this problem and a number of results were obtained [6, 9, 10, 14, 15, 26]. In [15], Thorup et al. gave a simple and interesting solution for this problem which answers each connectivity query in $O(\log n)$ time and takes $O(\log^2 n)$ time for each update. Later Thorup gave a near optimal solution for this problem [26] which answers each connectivity query in $O(\log n/\log\log\log n)$ time and completes each insertion or deletion operation in $O(\log n(\log\log n)^3)$ expected amortized time.

In this paper we use the data structure in [26] for our problem. In practice (e.g., critical area computation), the simpler algorithm in [15] may be more practical. When the choice of the connectivity data structure is unclear, we use *MaxQU* to represent the maximum of the connectivity query time and the update operation time.

Enumerating 1-D Geometric Minimal Cuts in a Slab To make use of the connectivity data structure, we first consider the problem of identifying 1-D geometric minimal cuts in a vertical slab S with k edges (see Fig. 2). Clearly, there are $O(k^2)$ possible 1-D geometric cuts and $O(k)$ 1-D geometric minimal cuts (see the proof of Lemma 2). To find out the $O(k)$ minimal cuts from $O(k^2)$ possible locations, we first sort edges based on the y coordinates, and let $e_1 = (u_1, v_1), e_2 = (u_2, v_2), \ldots, e_k = (u_k, v_k)$ be the k edges. We build a fully dynamic connectivity data structure $\mathit{FDC}(G')$ for the contracted graph G', and then run the algorithm for a slab (called 1DSlab). The main steps of 1DSlab are the follows.

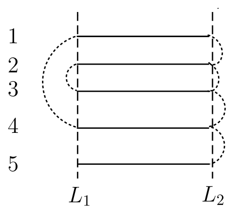

Fig. 2 An example for 1DSlab (only show the portions inside the slab), where the *dotted lines* are the edges in $G' \setminus H$. $\{2,3\}$ and $\{5\}$ are the 1-D geometric minimal cuts

1. Initialize a variable $r = 1$ to represent the index of the next to be deleted edge.
2. Starting from e_r, repeatedly delete edges of S from G' according to their sorted order and store them in a queue Q. For each deleted edge e_i, query $FDC(G')$ the connectivity of u_i and v_i. Stop the deletion when encountering the first edge e_j whose two endpoints u_j and v_j are disconnected or the last edge. In the latter case, insert all deleted edges back and stop.
3. Insert the deleted edges in Q back in the same order as they are deleted and update $FDC(G')$. After inserting each edge e_i, query the connectivity of the two endpoints of e_j, u_j and v_j.
4. If u_j and v_j are disconnected, add a forward pointer from e_i to e_j and insert edges in Q back to G'.
5. If u_j and v_j are connected, add a forward pointer from e_i to e_j, set $r = j + 1$, and repeat Steps 2 to 5 until encountering the last edge e_k. In this case, insert all remaining edges in Q back to G' and $FDC(G')$.
6. Reverse the order of the k edges and repeat the above procedure. In this step the added pointers are backward pointers.
7. For each edge e_j, find the nearest edge e_i which has a forward pointer to e_j and the nearest edge e'_i with a backward pointer to e_j. Output the set of edges between e_i and e_j (including e_i and e_j) as a 1-D geometric minimal cut, and edges between e_j and e'_i (including e_j and e'_i) as another 1-D geometric minimal cut.

Below we first show the correctness of the above algorithm.

Lemma 4 *Assume G' is originally a connected graph. In Step 2 of the 1DSlab algorithm, if u_i and v_i (the endpoints of the last deleted edge e_i) are connected, the set of deleted edges (i.e., e_r to e_i) does not form a cut in G'. On they other hand, if u_i and v_i are disconnected, then the set of deleted edges does form a cut in G.*

Proof Firstly, if u_i and v_i are disconnected, obviously the set of deleted edges forms a cut. Thus we focus only on the case that u_i and v_i are connected, and show that in this case the set of deleted edges does not form a cut.

We prove this by induction on the number i of deleted edges. The base case is $i = 1$. In this case, since G' is originally a connected graph and u_i and v_i are still connected in $G' \setminus \{e_i\}$, obviously there exists no cut. Assume that after deleting $i - 1$ edges, the set of edges $S_{i-1} = \{e_1, e_2, \ldots, e_{i-1}\}$ does not form a cut. Then since $G' \setminus S_{i-1}$ is a connected graph, after deleting edge e_i, the case becomes the same as the base case. Thus the lemma follows. □

Lemma 5 *In the* 1*DSlab algorithm, the disconnectivity of* u_j *and* v_j *in Step* 4 *or the connectivity of* u_j *and* v_j *in Step* 5 *implies that the set* $S_{i,j} = \{e_i, e_{i+1}, \dots, e_j\}$ *forms a cut in* G'.

Proof The first case (i.e., u_j and v_j are disconnected in Step 4) is obvious. For the second case (i.e., u_j and v_j are connected in Step 5), since u_j and v_j are disconnected before the insertion of e_i, it means that $S_{i,j}$ is a cut. □

The above lemmas indicate that the cut query can be answered by a sequence of connectivity queries.

Theorem 1 *The* 1*DSlab algorithm generates all* 1*-D geometric minimal cuts in the slab* S *in* $O(kMaxQU)$ *time, where MaxQU is the maximum of the query time and the updating time of the* $FDC(G')$ *data structure.*

Proof We first show that the 1DSlab algorithm generates all 1-D geometric minimal cuts. By Lemmas 4 and 5, we know that the forward pointers to e_j indicate all cuts whose edge set appears consecutively and ends at edge e_j. Similarly, the backward pointers to e_j indicate all cuts whose edge set appears consecutively and starts from e_j. By finding the nearest edges with forward and/or backward pointers to e_j, the algorithm identifies the (up to) two 1-D geometric minimal cuts starting from and ending at e_j. Since the computation is for every edge in S, it generates all 1-D geometric minimal cuts.

For the running time, it is clear that the sorting takes $O(k \log k)$ time. After sorting, in each of the forward and backward computations, every edge in S is deleted and inserted once. As for the connectivity queries, we notice that the endpoints of each edge can be queried multiple times (one in Step 2 and others in Step 3). To bound the total query time, we charge the query on u_j and v_j in Step 3 to edge e_i. Since e_i is inserted only once, it will be charged only once. Thus each edge in S will be responsible for at most two queries. Therefore the total number of connectivity queries is $O(k)$. Since the insertion, deletion, and query operations take no more than $O(MaxQU)$ time, the theorem follows. □

From the above theorem, we know that even though there are $O(k^2)$ consecutively appearing subsets of edges that could be 1-D geometric minimal cuts, the connectivity data structure can help us to reduce the time to near linear.

Generating 1-D Geometric Minimal Cuts With the algorithm 1DSlab, we now combine it with a plane sweep algorithm to generate all possible 1-D geometric minimal cuts in H.

A straightforward way of using 1DSlab is to partition the plane P into $O(n)$ slabs and apply the 1DSlab algorithm in each slab. This will lead to a $O(n^2 MaxQU)$-time solution. A more output sensitive solution is to use the following plane sweep algorithm.

In the plane sweep algorithm, a vertical sweeping line L is used to sweep through all edges in H to generate those 1-D geometric minimal cuts induced by vertical segments. Similarly we can generate those 1-D geometric minimal cuts induced by horizontal segments by sweeping a horizontal line. For the vertical sweeping line algorithm, the event points are the set V_H of endpoints in H. At each event point, the set of edges intersecting the sweeping line L forms a slab. However, instead of applying the 1DSlab algorithm to the whole set of intersecting edges, we work only on a subset of edges.

Let u be the event point. If u is the left endpoint of an edge e, then e is the new edge just encountered by L. Thus we need only to identify all 1-D geometric minimal cuts which contain e. This means that the 1DSlab algorithm needs only to consider the minimal cut C_1 bounded by e from the bottom, the minimal cut C_2 bounded by e from the top, and all minimal cuts between the top edge e_t of C_1 and the bottom edge e_b of C_2. To deal with this case, we can easily modify the 1DSlab algorithm. Particularly, we can start from e, have a backward computation, and stop at the first edge e_t whose removal disconnects its two endpoints, to generate C_1. Similarly, we can start from e and have a forward computation to generate C_2. In this way, we avoid dealing with possibly many edges beyond e_t and e_b.

If u is the right endpoint of e, we need to check those cuts containing e to see whether they are still geometric minimal cuts. Also we need to check whether new cuts can be generated. Clearly, we need only to consider the edges between e'_t and e'_b, where e'_t is the top edge of the 1-D geometric minimal cut bounded by e from the bottom and e'_b is the bottom edge of the 1-D geometric minimal cut bounded by e from the top. Thus we can first obtain such information from the previous step (i.e., the step before encountering u) and then directly apply the 1DSlab algorithm on the subset of edges between e'_t and e'_b.

Theorem 2 *All* 1*-D geometric minimal cuts of H can be found in $O(n \times MaxC \times MaxQU)$ time*, *where $MaxC$ is the maximum size of a* 1*-D geometric minimal cut.*

Proof Follows from the above discussion. □

3.2 Computing 2-D Geometric Minimal Cuts

In this section, we extend the algorithm in Sect. 3.1 to compute 2-D geometric minimal cuts.

To compute 2-D geometric minimal cuts, we first observe that a 1-D geometric minimal cut is a degenerated version of a 2-D geometric minimal cut. The only difference is that the minimum inducing region of a 1-D cut has two opposite sides degenerated to points. Thus if we conceptually "contract" two opposite sides of the minimum inducing region $R(C)$ of a 2-D cut C into "points", we can apply the plane sweep algorithm for 1-D cuts to generate 2-D cuts.

To implement this idea, we use two parallel sweeping lines L_1 and L_2 (called primary and secondary sweeping lines) to bound the "contracted" sides of $R(C)$. By Lemma 1, we know that each 2-D geometric minimal cut is bounded by the endpoints (or edges) of up to four edges. This suggests that the possible locations of L_1 and L_2 are the endpoints of the input edges. Similar to the plane sweep algorithm for 1-D cuts, we sweep the edges in H twice, one vertically and the other horizontally. Below we focus our discussion on horizontal sweeping (i.e., using vertical sweeping lines).

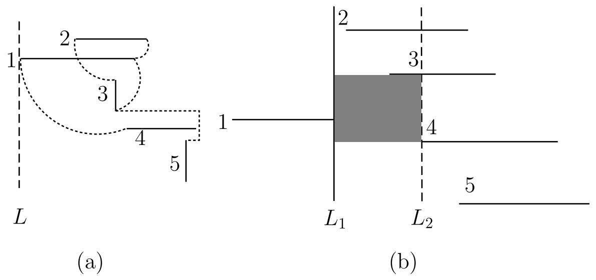

Fig. 3 (**a**) An example for illustrating the plane sweep algorithm for 1-D cuts. (**b**) An example for illustrating the double plane sweep algorithm for 2-D cuts

Our double plane sweep algorithm first sorts all edges in H based on the x coordinates of their left endpoints (for vertical segments, we view their upper endpoints as the left endpoints and lower endpoints as the right endpoints). Let $S_1 = \{e_1, e_2, \ldots, e_n\}$ be the sorted order. The primary sweeping line L_1 stops at the left endpoint of each edge e in the order of S_1. If the right endpoint of e is to the left of the left endpoint of the next edge $e' \in S_1$, we move the sweep line L_1 to the right endpoint of e (see Fig. 3(b)). If e is a cut by itself, we continue to move L_1 to the next edge in S_1. If e is not a cut, we fix L_1 and sweep the secondary sweeping line L_2 from the left endpoint of the next edge e' in S_1 to the last edge in $S1$. When L_2 stops, we use the plane sweep algorithm for 1-D cuts (in Sect. 3.1) to compute all 2-D geometric minimal cuts formed by the set S of edges intersecting the vertical region VR bounded by L_1 and L_2.

For edges in S, we assume that there are two sorted orders S_L and S_U. S_L is sorted based on the y coordinates of the lower endpoints and S_U is based on the y coordinates of the upper endpoints. (For horizontal edges, we view their left endpoints as upper endpoints and right endpoints as the lower endpoints. It is not difficult to see that the two sorted lists can be dynamically maintained in the double plane sweep algorithm.) S_L is used to generate cuts for edges above e and S_U is used to generate cuts for edges below e.

Each 2-D geometric minimal cut C in the vertical region VR is bounded by L_1 from left, L_2 from right, an edge e_u from top, and an edge e_d from bottom. Since all edges in S are in sorted order and $R(C)$ intersects edges in the same order in either S_L or S_U (depending on their relative locations to e, the 2-D geometric minimal cuts in VR can be viewed as 1-D geometric minimal cuts induced by vertical "segments" with a segment width equal to the horizontal distance of L_1 and L_2. Thus they can be generated by using the plane sweep algorithm for 1-D cuts. Furthermore, similar to the plane sweep algorithm for 1-D cuts, in VR we need only to consider those 2-D geometric minimal cuts containing edge e (see Fig. 4).

Once all the cuts in VR are identified, we move L_2 to the next edge in S_1. After sweep L_2, we move L_1 to the next edge in S_1 and repeat the above procedure until L_1 sweeps every edge.

To analyze our double plane sweep algorithm, we first prove the following lemma.

Lemma 6 *There are at most $O(n^3)$ 2-D geometric minimal cuts in H.*

Proof By Lemma 1, we know that the inducing region of each 2-D geometric minimal cut is bounded by an edge each side. There are $O(n^2)$ pairs of edges to bound the

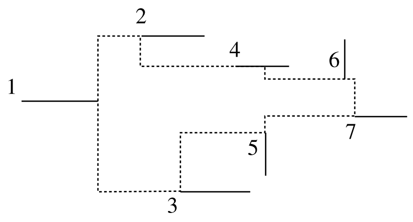

Fig. 4 Example of the double plane sweep algorithm for 2-D cuts. *The region* bounded by the *dotted lines* is the actual region where the algorithm searches for all 2-D geometric minimal cuts bounded by 1 from *the left*. Note different $R(C)$s may overlap

left and right sides of the inducing region. For each pair, the vertical region is similar to a slab (according to the above discussion). By a similar argument in the proof of Lemma 2, we know that in each vertical region, there at most $O(n)$ 2-D geometric minimal cuts. Thus the lemma follows. □

Note that the 2-D geometric minimal cuts in the above lemma could overlap each other.

Now we analyze the running time of the double plane sweep algorithm. The primary sweeping line L_1 stops at $O(n)$ location. For each fixed location of L_1, L_2 sweeps all edges not yet encountered by L_1 which can be $O(n)$ edges. For each vertical region bounded by L_1 and L_2, it takes $O(MaxC \times MaxQU)$ time (by Lemma 2). Thus the total time is $O(n^2 \times MaxC \times MaxQU)$.

Theorem 3 *All 2-D geometric minimal cuts in H can be found in $O(n^2 \times MaxC \times MaxQU)$ time.*

Proof Follows from the above discussion. □

Corollary 1 *All geometric minimal cuts in the MGMC problem can be found in $O(n^3 \log n (\log\log n)^3)$ expected time.*

Proof Follows from Theorems 2 and 3, and article [26]. □

Remarks In variants of critical area computation one may be interested in specific defect sizes. In this case the size of an inducing rectangle could be assumed bounded by a constant. Furthermore, since edges in VLSI design are separated by at least some constant distance, the total number of edges in an inducing rectangle can also be regarded a constant. In this case, the secondary sweeping line L_2 needs only to sweep a constant number of edges. Thus the running time of both plane sweep algorithms is $O(n \times MaxQU)$.

Corollary 2 *If the maximum size of a cut C is bounded by a constant, then all geometric minimal cuts in H can be found in $O(n \log n (\log\log n)^3)$ expected time.*

Proof Follows from Theorems 2 and 3, article [26], and the above remarks. □

4 Generating Map of Geometric Minimal Cuts

In this section, we show that problem 3 can be solved by using Hausdorff Voronoi diagram.

4.1 From Geometric Minimal Cuts to Hausdorff Voronoi Diagram

Given two sets A and B, the directed Hausdorff distance from A to B is

$$h(A, B) = \max_{a \in A} \min_{b \in B} \{d(a, b)\},$$

and the undirected Hausdorff distance between A and B is

$$d_h(A, B) = \max\{h(A, B), h(B, A)\},$$

where $d(a, b)$ is the distance between points a and b. In case set A degenerates to a single point a then the Hausdorff distance $d_h(A, B)$ simplifies to the maximum (farthest) distance between a and B. In this paper we use the L_∞ metric to measure the distance.

Given a set $\mathcal{C}$ of geometric minimal cuts of H, the Hausdorff Voronoi diagram $HVD(\mathcal{C})$ of $\mathcal{C}$ is a partition of the embedding plane P of H into regions (or cells) so that the Hausdorff Voronoi cell of a cut $C \in \mathcal{C}$ is the union of all points whose farthest (equiv. Hausdorff) distance to $R(C)$ is closer than to any other cut in $\mathcal{C}$. This means that for any point $p \in P$, if we grow an L_∞ ball from p (i.e., an axis-aligned square centered at p), the ball entirely contains $R(C(p))$ earlier than the minimum inducing region of any other cut, where $C(p)$ is the cut owning the Hausdorff Voronoi cell containing p.

In our MGMC problem, we have two types of objects, the minimum inducing regions of 1-D geometric minimal cuts and the minimum inducing regions of 2-D geometric minimal cuts. For every 2-D cut C, the rectangle $R(C)$ is fixed. However for a 1-D cut C, the location of $R(C)$ is not fixed, since there may be an infinite number of 1-D cuts cutting the same set of edges. In this case, we use the union of the infinite number of inducing segments $R(C)$ to represent the cut, which is a rectangle (denoted by $UR(C)$) bounded by the two extreme 1-D cuts C_{left} and C_{right} (or C_{top} and C_{bottom}), and the two bounding edges $B(C)$ of C. Thus from thereafter, we assume that our objects of the Hausdorff Voronoi diagram are two types of axis-aligned non-disjoint rectangles, corresponding to the $R(C)$s of 1-D and 2-D cuts respectively. Later, we will show that the two types of rectangle can be handled similarly with a slight difference on the Hausdorff distance computation. Thus our problem reduces to computing the L_∞ Hausdorff Voronoi diagram of a set of rectangles.

The L_∞ Hausdorff Voronoi diagram was introduced in [18] as a solution to the VLSI critical area extraction problem for a defect mechanism called a *via-block*. The Hausdorff Voronoi diagram in its general form first appeared in [5], as the *cluster Voronoi diagram*, where several combinatorial bounds and an $O(n^2)$-time algorithm for its construction were described. A tight combinatorial bound and a plane sweep construction were given in [19]. In [18] the Hausdorff Voronoi diagram in the L_∞ metric was reduced to computing an L_∞ Voronoi diagram of additively weighted axis-parallel line-segments (referred to as HVD-to-VDSA reduction) and it was addressed by plane sweep. The sweeping process in [18, 19] was based on the standard plane sweep paradigm for Voronoi diagrams [2, 8], its adaptation for line-segments in L_∞ [22], and it was generalized with the ability to handle the special features of Hausdorff Voronoi diagrams (such as line segments not entirely enclosed within their Voronoi region and disconnected Voronoi regions). In the L_∞ metric this resulted in an $O((n+K)\log n)$ algorithm, where K, although negligible in the experimental results of [18], is formally $O(n^2)$, which can be significantly larger than the actual size of the diagram. In a recent paper [25] (following the submission of this paper) the 2D plane sweep construction of [18] was improved to near optimal in the worst case and to optimal $O(n \log n)$-time, in the case of non-crossing rectangles. Nevertheless, the time complexity remains not output sensitive and it can be asymptotically larger than the actual size of the diagram. In the case of non-crossing rectangles, the randomized incremental construction for abstract Voronoi diagrams of [13] could also be employed to derive an $O(n \log n)$-expected time. However, abstract Voronoi diagrams, as they currently appear in the literature [11, 12], are not applicable to arbitrary rectangles that appear in the MGMC problem, as Hausdorff bisecting curves may be disconnected (see [24]). Abstract Voronoi diagrams have been used in [1] in the case of disjoint objects. For general configurations of objects no output-sensitive algorithm exists so far for constructing a Hausdorff Voronoi diagram.

In the remainder of this paper, we present the first output sensitive algorithm for the Hausdorff Voronoi diagram of arbitrary rectangles. We adopt a plane sweep approach in 3D space with time as the third dimension. With the help of some 3D data structures, such an approach enables us to directly relate the computation to the vertices and edges of the Voronoi diagram and thus leads to an output sensitive solution.

To present our solution, we first show how the problem is reduced to a 3D problem. Although this could also be obtained easily from the HVD-to-VDSA reduction, we discuss the details here as it could help understanding our approach.

The Hausdorff Voronoi diagram construction can be viewed as propagating a wave from each rectangle with unit speed. To better illustrate the wave propagation, we consider the farthest distance from an arbitrary point p to an axis-aligned rectangle $R(C)$, corresponding to the minimum inducing region of a 2-D cut C. The farthest (resp. Hausdorff) distance of $d_h(p, R(C))$ is achieved at one of the four corner points, $v_1(C)$, $v_2(C)$, $v_3(C)$, and $v_4(C)$, of $R(C)$. Thus $d_h(p, R(C)) = \max\{d_\infty(p, v_1(C)), d_\infty(p, v_2(C)), d_\infty(p, v_3(C)), d_\infty(p, v_4(C))\}$. To propagate a wave $W(C)$ from $R(C)$, we observe that the initial wavefront $\partial W(C)$ is the set of points whose farthest (resp. Hausdorff) distance to $R(C)$ is the minimum. Notice that when $R(C)$ has positive size (i.e., $R(C)$ is not a point), the minimum Hausdorff distance is positive (i.e., $\max\{a, b\}/2$, where a, b are the length and width of $R(C)$), and achieved when $d_\infty(p, v_1(C))$, $d_\infty(p, v_2(C))$, $d_\infty(p, v_3(C))$ and $d_\infty(p, v_4(C))$ have the same distance. The wavefront then expands to points having larger Hausdorff distance to $R(C)$. Since the Hausdorff distance to $R(C)$ is determined by the four corner points, an equivalent view is to propagate 4 separated waves from the 4 corner points of $R(C)$ with each being an L_∞ ball. Let $B_1(C)$, $B_2(C)$, $B_3(C)$ and $B_4(C)$ be the 4 L_∞ balls. The common intersections of the 4 balls are the wave of $R(C)$ (i.e., $W(C) = \bigcap_{i=1}^{4} B_i(C)$). Initially $\partial W(C)$ is empty. Once the size of the 4 balls

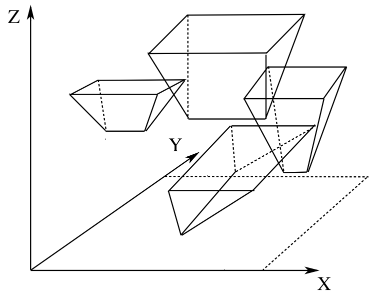

Fig. 5 Wavefronts of geometric minimal cuts

$B_i(C)$ reaches the minimum Hausdorff distance to $R(C)$, their common intersection forms a segment s_C located at the center of $R(C)$ and parallel to the shorter side of $R(C)$. As $B_i(C)$ grows, $\partial W(C)$ becomes a rectangle. The segment s_c is exactly the axis-parallel portion of the L_∞ farthest Voronoi diagram of $R(C)$ (see also [18]).

To visualize the whole growing process, we can lift the waves to 3D with time being the third dimension. Thus the wavefront of $R(C)$ becomes a 4-sided facet cone in the 3D space and apexed at s_C (i.e., the apex is not in the xy plane due to its positive minimum Hausdorff distance; see Fig. 5). Each facet of $\partial W(C)$ forms a 45 degree angle with the xy plane.

For a 1-D cut C, its wavefront is slightly different. Let $UR(C)$ be the rectangle of C. Since $UR(C)$ is the union of an infinite number of inducing segments $R(C)$, the Hausdorff distance to an arbitrary point p is calculated differently. For a fixed inducing segment $R(C) \in UR(C)$, let $u_1(R(C))$ and $u_2(R(C))$ be its two endpoints. The Hausdorff distance from $R(C)$ to p is achieved at one of the two endpoints (i.e., $d_h(p, R(C)) = \max\{d(p, u_1(R(C))), d(p, u_2(R(C)))\}$). Thus the wavefront of $R(C)$ is the common intersection of two L_∞ balls centered at the two endpoints respectively, which is also a facet cone in 3D space.

The Hausdorff distance from p to $UR(C)$ is the minimum distance from p to one of the inducing segments in $UR(C)$, i.e.,

$$d_h\big(p, UR(C)\big) = \min_{R(C)\in UR(C)} d_h\big(p, R(C)\big).$$

Thus the wavefront of $UR(C)$ is the union of an infinite number of wavefronts, which is still a facet cone with similar shape to the wavefront of a 2-D cut. The difference between the wavefront of $UR(C)$ and that of a 2-D cut with exactly same-shaped $R(C)$ is that their respective s_C may orient differently and locate at different heights. The apex segment s_c is exactly the axis parallel portion of the farthest Voronoi diagram of C.

Lemma 7 *Let C be a 1-D or 2-D geometric minimal cut. At any moment, the wavefront of C is either empty or an axis-aligned rectangle. Furthermore, the wavefront in 3D is a facet cone apexed at a segment and with each facet forming a 45 degree angle with the xy plane.*

Proof Follows from the above discussion. $\square$

With the 3D wavefronts of all cuts, we can construct the Hausdorff Voronoi diagram by computing the vertical projection of the lower envelope of the set of 3D wavefronts.

Lemma 8 *The Hausdorff Voronoi diagram can be obtained by projecting the lower envelope of the* 3*D facet cones to the xy plane.*

Proof Follows from the definitions of the 3D wavefronts and Hausdorff Voronoi diagram. □

To connect these observations with previous literature note that the Hausdorff Voronoi diagram can also be interpreted as the (weighted) Voronoi diagram of segments corresponding to the apex segments of the 3D facet cones as projected in 2D. The height of each apex segment in 3D becomes an additive weight in 2D. Each apex segment in 2D (termed core segment in [18]) is the axis-parallel portion of the L_∞ farthest Voronoi diagram of the corresponding cut. Its weight is the radius of the smallest square that intersects all elements of the cut, which corresponds to the distance function of the farthest Voronoi diagram along its axis-parallel edge. The Hausdorff Voronoi diagram of geometric cuts is equivalent to the (weighted) Voronoi diagram of their core segments, where each core segment is the axis parallel edge of the L_∞ farthest Voronoi diagram of the cut [18]. Additively weighted Voronoi diagrams of line segments have not been given attention in the literature and they exhibit different properties than their counterparts of points. For example, line segments may not be entirely contained in their Voronoi regions and Voronoi regions may be disconnected (see e.g. [18]). The existing machinery of abstract Voronoi diagrams, as they currently appear in the literature, is thus not directly applicable to our problem.

4.2 Plane Sweep Approach and Properties of 3D Cones and Hausdorff Voronoi Diagram

To efficiently construct the Hausdorff Voronoi diagram $HVD(\mathcal{C})$, we follow the spirit of Fortune's plane sweep algorithm for points [8], and sweep along the x axis direction a tilted plane Q in 3D which is parallel to the y axis and forms a 45 degree with the xy plane (see Fig. 6(a)). Q intersects the xy plane at a sweep line L parallel to the y axis.

Since every facet of a 3D facet cone forms a 45-degree angle with the xy plane and apexed at either a horizontal or vertical segment, the intersection of Q and a cone $\partial W(C)$ is either a V-shape curve (i.e., consisting of a 45-degree ray and a 135-degree ray on Q) or a U-shape curve (i.e., consisting of a 45-degree ray, a segment parallel to L, and a 135-degree ray; see Fig. 6(a)). When the cone is first encountered, it introduces either a V-shape curve or a U-shape curve to Q. When L (or Q) moves, the curve grows and its shape may change from a V-shape to a U-shape. Accordingly, the lower envelope (or beach line) of all those V or U-shape curves forms a monotone polygonal curve.

With the beach line, one might think of directly applying Fortune's algorithm to sweep the objects. However, due to the special properties of this problem, quite a few

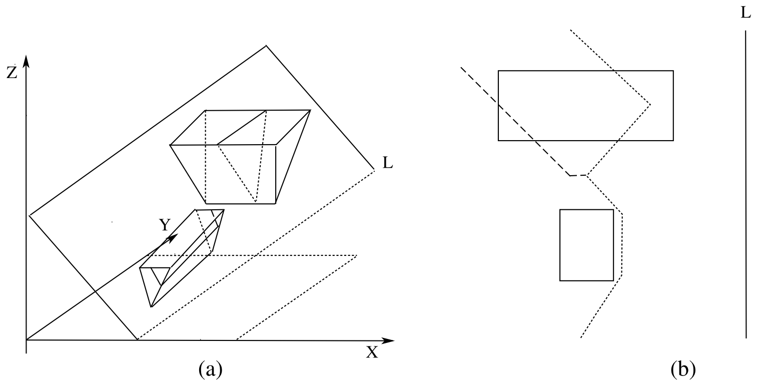

Fig. 6 (**a**) The intersections of *3D plane Q* and *cones*. (**b**) The Voronoi diagram with *sweep line* and *beach line*

significant differences exist between our problem and the ordinary Voronoi diagram problem, which fail the Fortune's algorithm. Below is a list of major differences as they occur in the 3D sweep.

(1) When a cone is first encountered by Q, its corresponding initial V or U-shape curve may not necessarily be part of the beach line.
(2) The initial V or U-shape curve may affect a number of curves in the beach line.
(3) Once a V shape moves away from the beach line, it may re-appear in the beach line at a later time.

The differences indicate that it is not sufficient to maintain only the beach line in order to extract all possible event points and produce the Voronoi diagram. More information of the arrangement of the V and U-shape curves is needed. This seemingly suggests that an algorithm with running time of the order of the complexity of the arrangement might be unavoidable. Below we show that with several interesting observations and ideas, we are able to obtain an output sensitive plane sweep algorithm. We start with presenting some basic properties of the Hausdorff Voronoi diagram of $\mathcal{C}$.

Definition 4 A 3D facet cone $\partial W(C)$ is a U-cone (or V-cone) if its apex segment s_C is parallel to the y (or x) axis.

First we consider U cones. Let $\partial W(C)$ be any U cone with apex segment s_C, and v_1 and v_2 be the two endpoints of s_C. When the sweep plane Q first encounters $\partial W(C)$, it introduces a U-shape curve C_u to Q. Let r_l, r_r, and s_m be the left and right rays and the middle segment of C_u respectively. Initially s_m is the apex segment s_C, and r_l and r_r are the two edges of the facet cone. When Q (or L) moves, C_u grows and always maintains its U-shape.

Lemma 9 *Let $\partial W(C)$, C_u, r_l, r_r and s_m be defined as above. When Q moves in the direction of the x axis, C_u is always a U-shape curve. The two supporting lines of r_l*

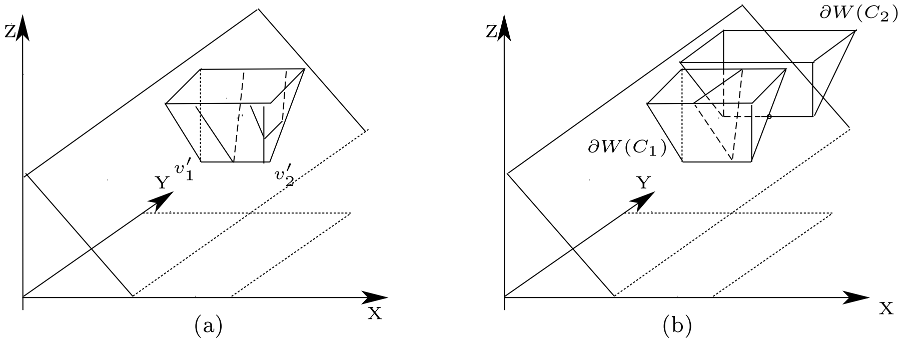

Fig. 7 (**a**) The *V-shape curve* changes to *U-shape curve*. (**b**) The *V-cone* is hidden by another *V-cone*

and r_r remain the same on Q, and the two endpoints of s_m (also the fixed points of r_l and r_r) moves upwards in unit speed along the two supporting lines.

Proof Follows from the shape and orientation of a U cone. □

For an arbitrary V cone $\partial W(C')$, let $s_{C'}$ be its apex segment, and v_1' and v_2' be its two endpoints (or left and right endpoints). When Q first touches $\partial W(C')$ at v_1', it generates a V-shape curve C_v'. C_v' remains a V-shape curve before encountering v_2'. After that, C_v' becomes a U-shape curve (see Fig. 7(a)).

Lemma 10 *Let r_l and r_r be the two rays of C_v', and s_m be the middle segment of the U-shape curve C_v' after Q visiting v_2'. During the whole sweeping process, the supporting lines of r_l and r_r are fixed lines on Q. C_v' remains the same V-shape curve on Q before encountering v_2'. s_m moves upwards in unit speed along the supporting lines of r_l and r_r after Q encounters v_2'.*

Proof Follows from the shape and orientation of a V cone. □

As mentioned earlier, the apex segment of each 3D cone is located at different height (i.e., its minimum Hausdorff distance). The heights and shapes of the 3D cones are the main reasons which cause the three differences in our problem. Indeed, as mentioned before, the height of cones corresponds to additive weights in 2D and the Voronoi diagram of additively weighted line segments has different structural properties than the one of ordinary (unweighted) line segments. For example, due to the existence of height in a 3D cone, the initial curve created by a newly encountered cone may be above the beach line (i.e., Difference (1)). Also due to the different size of the initial curve (not like the ordinary Voronoi diagram in which the initial curve is a vertical ray), it may intersect a number of segments or rays of the beach line (i.e., Difference (2)). More importantly, due to the existence of U and V-shape curves, a V-shape curve which is not part of the beach line could become part of the beach line in a later time (i.e., Difference (3)). To see this, we have the following lemma.

Lemma 11 *Let $\partial W(C_1)$ be either a U or V cone and $\partial W(C_2)$ be a V cone with its left endpoint v_1 of s_{C_2} being inside of $\partial W(C_1)$ and its right endpoint v_2 being outside of $\partial W(C_1)$. If $\partial W(C_2)$ is not entirely contained by the union $\bigcup_{C_i \in \mathcal{C}; C_i \neq C_2} \partial W(C_i)$, the V-shape curve C_2 introduced by $\partial W(C_2)$ will be hidden by the beach line at the beginning and then becomes part of the beach line later. This is the only case in which a hidden U or V-shape curve could appear in the beach line.*

Proof To simplify our proof, we assume that $\partial W(C_1)$ and $\partial W(C_2)$ are the only two cones in $\mathcal{C}$ (see Fig. 7(b)). The multiple cones case can be proved similarly by induction.

By the definition of the two 3D cones, we know that Q first encounters $\partial W(C_1)$ and generates a curve C_1 on Q. C_1 is the only curve in the beach line. When v_1 of $\partial W(C_2)$ is first encountered by Q, it introduces a V-shape curve C_2 on Q. Since v_1 is inside $\partial W(C_1)$ and every facet of the two cones forms a 45-degree angle with the xy plane, the two rays of C_2 will be inside the region defined by the two rays of C_1. This means C_2 will not be part of the beach line (or lower envelope). Since $\partial W(C_2)$ is not entirely inside $\partial W(C_1)$, v_2 must be outside of $\partial W(C_1)$. Let v_i be the intersection point of s_{C_2} and the wall of $\partial W(C_1)$. When Q sweeps through v_i, the apex point of the V-shape curve C_2 will intersect the U-shape curve C_1 (note that at this moment C_1 can only be a U-shape curve even if $\partial W(C_1)$ is a V cone), since C_1 is the intersection of Q and $\partial W(C_1)$ and v_i is the intersection of $\partial W(C_1)$ and s_{C_2}. Thus C_2 becomes part of the beach line.

To show that this is the only case where a hidden curve could appear in the beach line, first we note that the apex segment s_{C_2} of $\partial W(C_2)$ cannot be completely outside of $\partial W(C_1)$, since otherwise the initial curve of C_2 will be part of the beach line. Second, $\partial W(C_1)$ cannot be a U cone. If this is the case, then s_{C_2} is either partially or entirely inside $\partial W(C_1)$. For the first case, the middle segment s_m of the initial U-shape curve C_2 will intersect one of the two rays of C_1, which makes C_2 be part of the beach line. For the second case, the initial U-shape curve C_2 will be hidden by C_1. When Q moves, only the middle segment s_m of C_2 moves upwards in unit speed along the two rays of C_2 (by Lemma 9). If $\partial W(C_1)$ is a U cone, then by Lemma 9, the middle segment of C_1 will also move upwards in unit speed. Thus it will never catch up s_m. Therefore C_2 will never be part of the beach line. Similarly we can prove the same for the case $\partial W(C_1)$ is a V cone. Thus $\partial W(C_2)$ has to be a V cone. In this case, if s_{C_2} is completely inside of $\partial W(C_1)$, then by the above same argument, we can show that C_2 will never be part of the beach line. Thus the lemma follows. □

In the above lemma, the point v_i indicates that when Q sweeps it, the beach line is having a topological structure change. Thus v_i needs to be an event point for the plane sweep algorithm. However, since v_i is the intersection point of an apex segment and a 3D cone, it has to be computed on the fly. This indicates that in our problem there is a new type of event points.

Now we discuss our ideas for constructing the L_∞ Hausdorff Voronoi diagram *HVD*$(\mathcal{C})$. First we consider the bisector of two rectangles (or cuts). Let C_1 and C_2 be two axis aligned rectangles in $\mathcal{C}$. The bisector of C_1 and C_2 is a line or a segment with

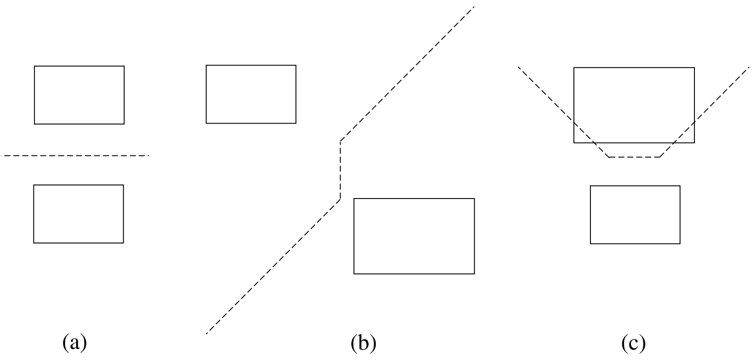

Fig. 8 (**a**) Bisector is a *straight line*. (**b**) and (**c**) Bisector is composed by a segment with two *half-lines*

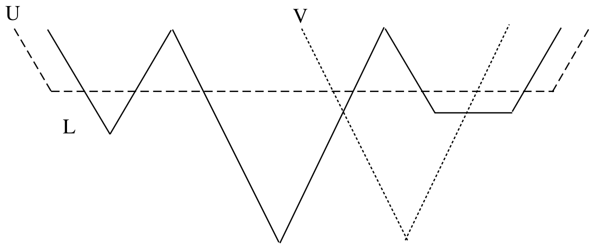

Fig. 9 *The beach line L* intersects by a new *U-cone* (*dashed line*) or *V-cone* (*dotted line*)

two rays as shown in Fig. 8. In the latter case, each bisector contributes two vertices to the Hausdorff Voronoi diagram. Hence the Hausdorff Voronoi diagram consists of two types of vertices: (a) The intersection points of the bisectors and (b) the vertices of the bisectors. For more examples see [18, 25]. For examples of disconnected L_∞ bisectors between crossing rectangles see [25].

Lemma 12 *Let $\mathcal{C}$ be a set of N rectangles. The edges of $HVD(\mathcal{C})$ are either segments or rays, and the vertices of the $HVD(\mathcal{C})$ are either the vertices of bisectors or the intersections of bisectors.*

Proof Follows from the above discussion. □

To obtain a plane sweep algorithm, we need to design data structures to maintain the beach line and the event points. In our problem, the beach line is the lower envelope of the set of V and U-shape curves, and is a y-monotone polygonal curve. For non-disjoint 3D cones, the complexity of the beach line may not be linear in the number of the rectangles in $\mathcal{C}$. Figure 9 shows a newly generated U-shape curve intersecting the beach line a number of times and contributing multiple edges to the beach line. Consequently, the complexity of the $HVD(\mathcal{C})$ is not linear. The following lemma is a straightforward adaptation of Theorem 1 in [19] for the L_∞ metric.

Lemma 13 *The size K of the L_∞ Hausdorff Voronoi diagram of N rectangles is $O(N + M')$, where M' is the number of intersecting pairs of rectangles.*

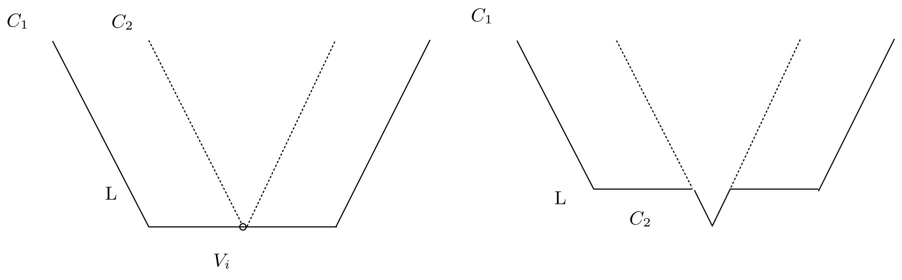

Fig. 10 (**a**) *The bottom segment* of *U-shape curve* C_1 reached V_i. (**b**) *The hidden V-shape curve* C_2 appeared on *the beach line L*

4.3 Events

For event points, we need to detect all events that cause the beach line to have topological structure changes. More specifically, we have to identify all the moments when a U or V-shape curve is inserted to or deleted from the beach line. There are two ways that a curve could appear in the beach line:

(A) A newly generated U or V-shape curve becomes part of the beach line;
(B) A hidden V-shape curve appears in the beach line.

There are also two ways for a curve or a portion of a curve to disappear from the beach line:

(C) A curve (or part of the curve) becomes hidden by a newly generated curve;
(D) A U-shape curve (or part of the U-shape curve) moves out of the beach line.

For (A), we know that this is caused by the sweep plane Q encountering a new 3D cone $\partial W(C)$. Such events can be detected in advance and are called *site events*. A site event, however, does not necessarily lead to a topological structure change to the beach line, since the new curve C can be hidden by the beach line. If $\partial W(C)$ is a U cone, the two endpoints of s_C are encountered by Q at the same time and either of them can be treated as a site event. If the corresponding U-shape curve C is not hidden, it may intersect the beach line multiple times as shown in Fig. 9. In this case, we have to update the beach line for each breakpoint introduced by C. Consequently, the U-shape curve C will be partitioned into multiple pieces. Each piece is either a part of the beach line (called *unhidden portion* of C) or hidden by other U or V shape curves in the beach line (called *hidden portion* of C). If $\partial W(C)$ is a V cone, the left endpoint v_1 of s_C is encountered first and can be viewed as a site event. When Q sweeps the right endpoint v_2 of s_C, C changes from a V-shape curve to a U-shape curve. To distinguish from the site event, we call v_2 as a *U event*. For unhidden V-shape curve C, it intersects the beach line at most twice (see Fig. 9).

(B) occurs when an unhidden portion of the bottom segment s_m of a U-shape curve C_1 moves upwards and encounters the apex point of a hidden V-shape curve C_2. We call this kind of events as *V events* (see Figs. 10 and 7(b)). Note that V-events in this paper correspond to the *active events* of [18] and *mixed-vertex events* of [19]. The V

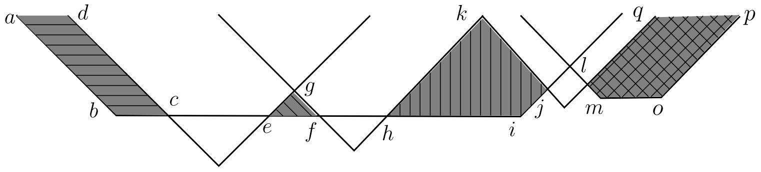

Fig. 11 Dominating regions of unhidden portions of U*-shape curves*

events are not known in advance and need to be computed by using the saved information in our data structures. Note that when s_m moves upwards, its hidden portions may also encounter the apex point of some hidden V-shape curve. In this case, we do not view it as an event, since the beach line has no topological structure change, thus generating no new Voronoi vertex or edge. To distinguish the two cases, we associate with each unhidden portion of a U-shape curve a region, called *dominating region* (see Fig. 11), which is the region swept by the unhidden portion when it moves upwards. The dominating regions could have a few different shapes (see the shaded regions in Fig. 11), with each of them bounded by zero, one, or two 45-degree rays, zero, one, or two 135-degree rays, and an unhidden portion of the bottom segment s_m of some U-shape curve. Clearly, for a hidden V-shape curve to cause a V event, its apex point has to fall in a dominating region of an unhidden portion of a U-shape curve. Thus to capture all V events, we need only to focus on those hidden V-shape curves whose apex points fall in the dominating regions.

(C) is obviously caused by a site event and thus can be detected in advance. For (D), the disappearing U-shape curve C (or its unhidden portion) is caused by the upwards movement of the bottom segment of C. It involves three curves, C and its immediate left and right neighbors C' and C'' in the beach line. Since this event is similar to the circle event in standard Voronoi diagram. We also call it circle event. The circle events cannot be detected in advance and have to be computed on the fly.

Thus we have in total four types of events, site events, circle events, U events, and V events. In the next section we discuss our ideas on how to handle these events.

4.4 Data Structures and Events Handling

To construct $HVD(\mathcal{C})$, we use doubly-connected edge lists to store $HVD(\mathcal{C})$. We also need two data structures for the plane sweep algorithm: an event queue and a sweep plane status structure representing the beach line.

The status structure for the beach line consists of three balanced binary search trees $\mathcal{T}$, $\mathcal{T}_{\pi/4}$, and $\mathcal{T}_{3\pi/4}$. $\mathcal{T}$ stores the y-monotone polygonal curve of the beach line. Each part of the curve corresponds to a V-shape curve or an unhidden portion of a U-shape curve. The leaves of $\mathcal{T}$ correspond to the V-shape curves and the unhidden portions of the U-shape curves on the beach line sorted by their y coordinates. Each leaf also stores location information of the corresponding 3D cone. The internal nodes of $\mathcal{T}$ adjacent to the leaves represent the breakpoints (i.e., the intersection of a pair of U or V curves) on the beach line. A breakpoint is stored at an internal node by an ordered tuple of curves $\langle C_i, C_j\rangle$, where C_i is the left curve of the breakpoint and C_j is the right curve of the breakpoint. $\mathcal{T}_{\pi/4}$ is used to maintain the orders (along the norm direction) of the 45-degree rays of all U or V-shape curves which appear in the beach line. Similarly, $\mathcal{T}_{3\pi/4}$ maintains the orders of the 135-degree rays of U or V-shape curves which appear in the beach line. Each leaf node in $\mathcal{T}_{\pi/4}$ or $\mathcal{T}_{3\pi/4}$ represents a 45-degree or 135-degree ray and each ray corresponds to a U or V-shape curve (representing by leaf node in $\mathcal{T}$) in the beach line. For each leaf node of $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$, we maintain a pointer pointing to the corresponding leaf node in $\mathcal{T}$. We also augment the $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ trees. For each node of the trees, we store the lowest z-coordinate of the bottom (segment) of all rays in the subtree rooted at that node. We call this as z'-coordinate, to distinguish it with the current z-coordinate of a U or V curve (note that the current z-coordinate of a U-shape curve is the z-coordinate of its apex point minus the x-coordinate of its apex point plus the current x-coordinate of the sweep-line L, and the current z-coordinate of a V-shape curve is the z-coordinate of its apex point until it becomes a U-shape curve; clearly the current z-coordinate of a curve can be easily obtained from the information of the cone and the position of the sweep-line L). For a U-shape curve, the z'-coordinate of its bottom segment is the z-coordinate of its apex point minus the x-coordinate. The z'-coordinate of the bottom segment of a V-shape curve is the z-coordinate of its apex point minus the x-coordinate and the length of bottom segment of the 3D V-cone. With the augmentation, the three trees provide all information of the beach line. Thus, by doing binary search on $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ and the pointers between the trees, we are able to locate the positions in the beach line for the apex points of each newly encountered cone at a site event in $O(\log N)$ time, and update the beach line in $O(k \log N)$ time (using current z-coordinate), where k is the number breakpoints destroyed and created after inserting the newly encountered U or V-shape curve into the beach line. For example, for a newly encountered U-shape curve, we first locate the positions of its two endpoints by doing binary search on $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$. Since all the information of the beach line between the two endpoints can be obtained by those three trees, updating the beach line can be done by binary search of the internal and leaf nodes of the three trees in the above time.

The event queue $\mathcal{Q}$ is implemented by a priority queue, where the priority of an event is the x coordinate of L (i.e., the intersection of Q and xy-plane) when the corresponding event point occurs. If two points have the same x-coordinate (of L), the one with larger y coordinate has the higher priority. All the site events and U-events are known in advance and are stored in $\mathcal{Q}$. Our main challenge is to detect the V events.

For a V event, we know that it occurs when the apex point of a hidden V-shape curve C_2 appears in the beach line. To detect such events, we have to store in our data structure the information of all hidden V-shape curves. This could potentially require us to maintain the whole arrangement of all curves on Q, and therefore results in unnecessarily high computational cost. To efficiently detect all possible V events, our main idea is not to maintain the arrangement, but rather to use the properties of V events to convert the problem into a query problem in 3D. To achieve this, we first observe that a V event is always caused by the upwards movement of an unhidden portion of the bottom segment s_m of some U-shape curve C_1 and occurs when s_m coincides with the apex point v of a hidden V-shape curve C_2 (by Lemma 11). Thus, in order to detect all possible V events, we need to overcome the following two

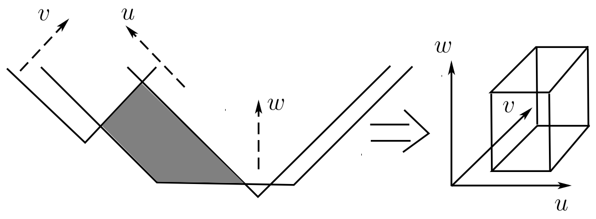

Fig. 12 A dominating region is converted to a 3D box in the orthogonized 3D space for MD

difficulties: (1) Identify the next V event associated with each unhidden portion of a U-shape curve, and (2) for each newly encountered hidden V-shape curve (in a site event), find the unhidden portion of a U-shape curve which will later cause a V event involving this V-shape curve. There is another one (Difficulty (3)) related to the two difficulties: How to find the exact boundary of the dominating region for a given unhidden portion c of a U-shape curve.

We first consider Difficulty (1). For this difficulty , we know that the next V event associated with an unhidden portion c of a U-shape curve C_1 is the hidden V-shape curve C_2 whose apex point v lies inside the dominating region of c and is the closest to the bottom segment s_m of C_1. However, as we have already noticed in last section, the dominating region could have various shapes which seemingly suggests that it is costly to find the next V event even if we have the dominating region. To overcome this difficulty, we first observe that the dominating region is bounded by 45-degree and 135-degree rays and the bottom segment. From Lemmas 9 and 10, we know that the rays of any U or V curve have fixed directions (e.g., forming 45-degree and 135-degree angles with the y axis) and their supporting lines remains the same during the whole sweeping process. This suggests that we could orthogonize the dominating regions in 3D space. The three dimensions of the new space are the orthogonal directions of the 45, 135 degrees lines in the sweep plane Q and the orthogonal direction of the bottom segment (or the y axis). In this way, each dominating region is converted into a (possibly unbounded) box in 3D (see Fig. 12).

To efficiently find the next V event in the orthogonized dominating region, we process the apex points of all V-shape curves into a 3D dynamic range search tree data structure MD [16, 17]. In MD, the apex point of each hidden V-shape curve is mapped into a 3D point. Thus for a particularly dominating region R, we can use its orthogonized box as the query range to find the closest apex point (among all hidden V-shape curves whose apex points fall in R) to the unhidden portion of the bottom segment of a U-shape curve in $O(\log^2 N \log\log N)$ time [16].

To efficiently maintain MD, we can build MD in advance for all possible V-shape curves. In each node p of the MD tree, we store a mark to indicate whether there is any active V-shape curve in the subtree rooted at p. A V-shape curve C is active if its corresponding 3D cone has already been encountered by the sweep plane Q, and C has not yet changed to a U-shape curve due to a U event. In this way, we need only to change the marks when the status of a V-shape curve changes, and therefore avoid complicate updating (such as tree rotation) to the MD.

To make use of MD, we have to identify all scenarios in which we need to either update MD or query MD for detecting potential V events. Firstly, we notice that a site event or a U event could introduce (a) a new U-shape curve C to the beach line and

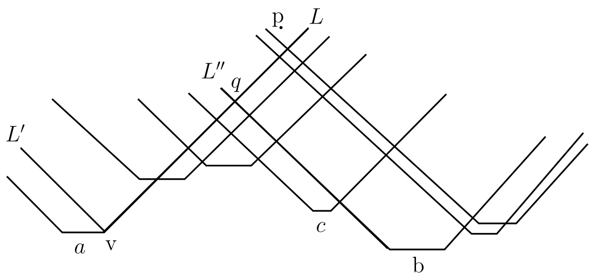

Fig. 13 Finding the dominating region of p

generate a set of unhidden portions of C and (b) a hidden V-shape curve C'. Thus for (a), in each such event we perform a 3D range query in MD for each unhidden portion c_i to find the closest hidden V-shape curve to c_i in its dominating region, and insert a V event into the event queue $\mathcal{Q}$. For (b), we have to find out the exact dominating region R which contains the apex point of the newly encountered hidden V-shape curve C' (i.e., Difficulty (2)) and then insert the hidden V event into MD, since the V event of C' might be the new next V event of the unhidden portion c of R. (We will discuss our idea later on this challenging problem.) Secondly, After a V event, we also have to perform a 3D range query in MD to find the closest hidden V-shape curve to the new U-shape curve converted from the V-shape curve of the V event. Thirdly, if an unhidden portion c of a U-shape curve C disappears from the beach line (e.g., a circle event), we delete its associated V event from $\mathcal{Q}$, since c will never appear in the beach line again by Lemma 11.

Now we discuss our ideas for Difficulty (2) (i.e., finding the unhidden portion of the dominating region containing the apex point of a hidden V-shape curve in a site event). Let p be the apex point of the hidden V-shape curve. As shown in Fig. 13, finding the four rays (two 45-degree rays and two-135 degree rays) bounding p does not necessarily give us the correct dominating region. This is because the rays bounding p could be stopped by the rays bounding the actual dominating region. For example, L is stopped by L'' at point q. Thus a ray inside a dominating region may not be a ray bounding that dominating region.

Definition 5 A ray (or a portion of a ray) is active if it bounds some dominating region and inactive otherwise.

Let l be any ray in the beach line. If we start from the bottom (i.e., the endpoint) of l and walk along it, l is active until it is stopped by some other ray and thus becomes inactive. It is easy to see that once a ray becomes inactive, it will never be active again. In Fig. 13, L is active until stopped by L'' and will no longer be active. A ray has two sides and it may not be active on both sides simultaneously. Also one side of a ray may bound more than one dominating region (L'' bounds the dominating regions generated by U-shape curves with bottom segments a and c). It is easy to see that for any ray, there is always one side bounding at most one dominating region. Otherwise, there will be an intersection between two dominating regions, thus contradicting the definition of dominating regions.

With these observations, to find out the actual dominating region of p, we can first find the four rays bounding it by searching the $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ trees. Further we need to find out whether these rays are active or not on the sides containing p. To determine this, we need to know whether each of the four rays has been stopped by other rays. If we can find out the ray L which stops these rays, then we know that p belongs to the dominating region bounded by L. This means that for a given ray, we need to have a way to efficiently determine which ray stops it. With the augmented information in the $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ trees, we can search for a ray L the ray L'' which stops L in the two trees in $O(\log N)$ time.

To better understand the idea, consider the example in Fig. 13. Let L be a 45-degree ray with its endpoint v. A ray L'' which stops L is a 135-degree ray if it exists. To find L'', we first create another 135-degree ray L' with endpoint v. We can first search in $\mathcal{T}_{3\pi/4}$ the position of L'. Then we only need to find the ray between L' and p (in the direction of L) with (1) lower z'-coordinate than v and (2) the closest z'-coordinate. This can be done by following the searching path of L' in $\mathcal{T}_{3\pi/4}$ upwards until we find a node satisfying the two conditions and then move downwards to locate the ray L''. In Fig. 13, the dominating region with bottom segment b dominates p. Thus we have the following lemma.

Lemma 14 *It takes $O(\log N)$ time to find out the exact dominating region of an unhidden portion c for the apex point p of the hidden V-shape curve C' generated by the site event.*

Proof Follows from the above discussion. □

It is not difficult to see that the augmented information can be maintained during the whole sweep process within the same time bound. With the augmented information, we can also find the exact boundary of a dominating region R for an unhidden portion c of a U-shape curve (i.e., Difficulty (3)) in $O(\log N)$ time following the same idea (i.e., finding the rays stopping the bounding rays of R).

Circle events can be handled in a way similar to the standard Voronoi diagram. More specifically, we check every new triple of consecutive U or V-shape curves that appear in the beach line. If such a new triple has converging breakpoints ,the event is inserted into the event queue $\mathcal{Q}$. Furthermore, for all disappearing triples, we de-queue the corresponding event from $\mathcal{Q}$ if it has been inserted.

4.5 Algorithm and Analysis

Now we are ready to describe our plane sweep algorithm. The main steps of the algorithm are as follows.

Algorithm (HAUSDORFF-VORONOI-DIAGRAM($\mathcal{C}$))
`Input`. A set $\mathcal{C}$ of axis aligned rectangles (or geometric minimal cuts) in the plane.
`Output`. The Hausdorff Voronoi diagram in a doubly connected edge list $\mathcal{D}$.

1. Initialize the event queue $\mathcal{Q}$ with all site events and U events; initialize empty sweep plane status structures $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$, and an empty doubly-connected edge list $\mathcal{D}$; initialize a 3D range search tree MD for all possible V-shape curves with all nodes marked as inactive.

```
2. while $\mathcal{Q}$ is not empty
3.     do Remove an event with the smallest $x$-coordinate from $Q$.
4.     if the event is a site event, then HANDLE-SITE-EVENT
5.     if the event is a circle event, then HANDLE-CIRCLE-EVENT
6.     if the event is a $U$-event, then HANDLE-U- – EVENT
7.     else the event is $V$-event, then HANDLE-V-EVENT
```

HANDLE-SITE-EVENT

1. Let C be the new curve. If status structure is empty, insert C into it and mark the apex point in MD as active if it is a V-cone. Otherwise, continue with steps 2–8.
2. If C is a V-shape curve, mark its apex point in MD as active and continue with steps 3-5.
3. Locate the position of the apex point of C in the beach line by searching $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$.
4. If C is not hidden, insert C to the beach line by updating $\mathcal{T}, \mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$. This includes inserting C into the beach line, computing new breakpoints, inserting possible circle events into $\mathcal{Q}$, and removing all curves hidden by C from the beach line; If an unhidden portion of a U-shape curve is removed, delete its corresponding V event from $\mathcal{Q}$; If an unhidden portion of a U-shape curve is partially blocked by C, search for its closest hidden V-shape curve in the reduced dominating region if necessary.
5. Else if (the apex of) C is inside the dominating region an unhidden portion c of a U-shape curve C', update the associated V event of c if needed.
6. Else if C is a U-shape curve, continue with steps 7–8
7. Locate the position of C in the beach line by searching $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$.
8. If C is not fully hidden, insert C into the beach line by updating $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$. This includes computing possibly multiple breakpoints and the corresponding circle events for all its unhidden portions, removing blocked curves (similar to Step 4), and finding the possible V event for each unhidden portion of C and partially blocked unhidden portion.

HANDLE-CIRCLE-EVENT

1. Update the beach line by updating $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$. Delete unnecessary circle events from Q involving the disappeared part on the beach line. If an unhidden portion of a U-shape curve moves out of the beach line, delete its associated V event.
2. Add the vertex to $\mathcal{D}$. Two new breakpoints of the beach line will be traced out.
3. Check the new triples involving the left or right neighbor of the disappeared part and insert the corresponding circle event into Q if necessary.

HANDLE-U-EVENT

1. If the V-shape curve C introduced by the V cone appeared on the beach line, the corresponding part of the beach line is changed from a V-shape curve to a U-shape curve. Update $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$, and add vertex to $\mathcal{D}$. Also find the possible V event for C by querying MD, and insert it into $\mathcal{Q}$.
2. Mark the node in MD containing C as inactive, and update its ancestors if needed.
3. Detect the circle events and insert them into $\mathcal{Q}$ if necessary.

HANDLE-V-EVENT

1. Update $\mathcal{T}$, $\mathcal{T}_{\pi/4}$ and $\mathcal{T}_{3\pi/4}$ by inserting the V-shape curve C into the beach line. Create new breakpoints, detect the possible circle event, and insert them into $\mathcal{Q}$.
2. For the corresponding unhidden portion c of a U-shape curve C', break it into two unhidden portions, c' and c'', and find possible new V event for c' and c'' respectively by querying MD.

Theorem 4 *The L_∞ Hausdorff Voronoi diagram $HVD(\mathcal{C})$ of a set $\mathcal{C}$ of geometric minimal cuts (or axis aligned rectangles) can be constructed by a plane sweep algorithm in $O((N+K)\log^2 N\log\log N)$ time, where $N=|\mathcal{C}|$ and K is the complexity of the Hausdorff Voronoi diagram.*

Proof The discussion on the algorithm shows that $HVD(C)$ can be constructed by the plane sweep algorithm. Thus we mainly focus on the time complexity.

We first consider site events. Let C be a newly encountered U or V-shape curve. If C is a V-shape curve, it is either part of the wavefront or a hidden V-shape curve. For the former case, the computation cost includes inserting C into the beach line, and updating MD and $\mathcal{D}$. The cost for these is $O(\log^2 N\log\log N + k\log N)$, where k is the total number of breakpoints hidden by C. Since each breakpoint corresponds to an Voronoi edge in $HVD(C)$, we can charge the cost of $O(k\log N)$ to the breakpoints (and their corresponding Voronoi edges). Thus each will be charged a cost of $O(\log N)$. The cost of $(\log^2 N\log\log N)$ can be charged to each V-shape curve. For the latter case, if C is not in the dominating region of any unhidden portion of a U-shape curve, the only cost is $O(\log^2 N\log\log N)$, which can be charged to C. If C is inside the dominating region of some unhidden portion c of a U-shape curve which can be checked in $O(\log N)$ time, we also need to check whether C is the closest V-shape curve to c, and this can be done in $O(1)$ time. Thus in the latter case, C is charged a cost of $O(\log^2 N\log\log N)$. If C is a U-shape curve, it could be fully hidden by the beach line and thus will never appear in the beach line in a later time. In this case, the total cost is $O(\log N)$ which can be charged to C. If C appears in the beach line and contributes some unhidden portions, then we need to update the sweep plane status structure which takes $O(k\log N)$ time, and find the closest V-shape curve for each unhidden portion c. The cost of $O(k\log N)$ can be evenly charged to all hidden and newly created breakpoints. Thus each of them will be charged a cost of $O(\log N)$. The closest V-shape curve to each unhidden portion c can be found by a query to MD, which takes $O(\log^2 N\log\log N)$ time and can be charged to the breakpoint bounding c. In a site event, up to two unhidden portions could be partially hidden by the newly encountered U or V-shape curve C. In this case, each of them may need to find a new closest hidden V-shape curve in its reduced dominating region. For this, we charge the cost of $O(\log^2 N\log\log N)$ to the new breakpoint created by the two rays of C and the two unhidden portions. Thus after all site events, each V-shape curve will be charged a cost of $(\log^2 N\log\log N)$ and each U-shape curve will be charged a cost of $O(\log N)$. Some breakpoints will be charged a cost of $O(\log^2 N\log\log N)$.

For circle events, clearly each of them can be handled in $O(\log N)$ time, and the total number of such events is bounded by $O(K)$. Thus the total cost for all circle events is $O(K\log N)$.

For U events, there are only $O(N)$ such events. Each event, takes $O(\log^2 N \log\log N)$ time to update MD and find the closest hidden V-shape curve to the new U-shape curve. Again, we charge all the cost to the V-shape curve.

For V events, from the algorithm, we know that each of them takes $O(\log^2 N \log\log N)$ time. To bound the total cost of all V events, we need to bound its total number. For this, we notice that at any moment, (1) each unhidden portion of a U-shape curve is associated with only one hidden V-shape curve and (2) the association of a hidden V-shape curve C and an unhidden portion c of a U-shape curve can change for only two reasons: (a) The dominating region of c changes (in a site event); (b) a new hidden V-shape curve C' is activated and C' is closer to c than C. For (a), it means part of c is hidden by other U or V shape curve C'', and the cost of de-association can be charged to the edges or vertices introduced by C''. For (b), the cost of de-association can be charged to C'. Each V-shape will be charged no more than once. This means that computation cost of each V event can be charged to the Voronoi edge or vertex corresponding to the two breakpoints bounding the unhidden portion. Each Voronoi vertex and edge is charged a constant times with a total cost of $O(\log^2 N \log\log N)$. Thus the total cost for all V events is $O(K \log^2 N \log\log N)$.

Putting all together, we have a total cost of $O((N + K)\log^2 N \log\log N)$. Thus the theorem follows. □

Acknowledgements The authors would like to thank three anonymous referees for their thoughtful comments and suggestions which significantly improve the presentation of the paper.

References

1. Abellanas, M., Hernandez, G., Klein, R., Neumann-Lara, V., Urrutia, J.: A combinatorial property of convex sets. Discrete Comput. Geom. **17**, 307–318 (1997)
2. Dehne, F., Klein, R.: The Big Sweep: on the power of the wavefront approach to Voronoi diagrams. Algorithmica **17**, 19–32 (1997)
3. Dehne, F., Maheshwari, A., Taylor, R.: A coarse grained parallel algorithm for Hausdorff Voronoi diagrams. In: Proc. 2006 International Conference on Parallel Processing, pp. 497–504 (2006)
4. Driscoll, J.R., Sarnak, N., Sleator, D., Tarjan, R.: Making data structures persistent. In: Proceedings of the 18th annual ACM symposium on Theory of computing, pp. 109–121 (1986)
5. Edelsbrunner, H., Guibas, L.J., Sharir, M.: The upper envelope of piecewise linear functions: algorithms and applications. Discrete Comput. Geom. **4**, 311–336 (1989)
6. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, A.: Sparsification—a technique for speeding up dynamic graph algorithms. J. ACM **44**(5), 669–696 (1997)
7. Ferris-Prabhu, A.: Defect size variations and their effect on the critical area of VLSI devices. IEEE J. Solid-State Circuits **20**(4), 878–880 (1985)
8. Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174 (1987)
9. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. **14**(4), 781–798 (1985)
10. Henzinger, M.R., King, V.: Randomized dynamic graph algorithms with polylogarithmic time per operation. In: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pp. 519–527 (1995)
11. Klein, R.: Concrete and Abstract Voronoi Diagrams. Lecture Notes in Computer Science, vol. 400. Springer, Berlin (1989)
12. Klein, R., Langetepe, E., Nilforoushan, Z.: Abstract Voronoi diagrams revisited. Comput. Geom. **42**(9), 885–902 (2009)
13. Klein, R., Mehlhorn, K., Meiser, S.: Randomized incremental construction of abstract Voronoi diagram. Comput. Geom. **3**, 157–184 (1993)

14. Henzinger, M.R., Thorup, M.: Sampling to provide or to bound: with applications to fully dynamic graph algorithms. Random Struct. Algorithms **11**, 363–379 (1997)
15. Holm, J., Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48**(4), 723–760 (2001)
16. Mehlhorn, K., Näher, S.: Dynamic fractional cascading. Algorithmica **5**, 215–241 (1990)
17. Nakamura, Y., ABE, S., Ohsawa, Y., Sakauchi, M.: MD-tree: a balanced hierarchical data structure for multi-dimensional data with highly efficient dynamic characteristics. IEEE Trans. Knowl. Data Eng. **5**(4), 682–694 (1993)
18. Papadopoulou, E.: Critical area computation for missing material defects in VLSI circuits. IEEE Trans. Comput.-Aided Des. **20**(5), 583–597 (2001)
19. Papadopoulou, E.: The Hausdorff Voronoi diagram of point clusters in the plane. Algorithmica **40**, 63–82 (2004)
20. Papadopoulou, E.: Higher order Voronoi diagrams of segments for VLSI critical area extraction. In: ISAAC'07. LNCS, vol. 4835, pp. 716–727 (2007)
21. Papadopoulou, E.: Net-Aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **30**(5), 704–717 (2011)
22. Papadopoulou, E., Lee, D.T.: The L_∞ Voronoi diagram of segments and VLSI applications. Int. J. Comput. Geom. Appl. **11**, 503–528 (2001)
23. Papadopoulou, E., Lee, D.T.: Critical area computation via Voronoi diagrams. IEEE Trans. Comput.-Aided Des. **18**(4), 463–474 (1999)
24. Papadopoulou, E., Lee, D.T.: The Hausdorff Voronoi diagram of polygonal objects: a divide and conquer approach. Int. J. Comput. Geom. Appl. **14**(6), 421–452 (2004)
25. Papadopoulou, E., Xu, J.: The L_∞ Hausdorff Voronoi diagram revisited. In: IEEE-CS Proceedings of Int. Symposium on Voronoi Diagrams in Science and Engineering, ISVD 2011 (2011)
26. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: STOC'00, pp. 343–350 (2000)