ELSEVIER

Computer-Aided Design 38 (2006) 260–272

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

A sweepline algorithm for Euclidean Voronoi diagram of circles

Li Jin [a], Donguk Kim [b], Lisen Mu [a], Deok-Soo Kim [c], Shi-Min Hu [a,*]

[a] *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, People's Republic of China*
[b] *Voronoi Diagram Research Center, Hanyang University, Seoul 133-791, South Korea*
[c] *Department of Industrial Engineering, Hanyang University, Seoul 133-791, South Korea*

Received 3 April 2005; received in revised form 2 October 2005; accepted 13 November 2005

Abstract

Presented in this paper is a sweepline algorithm to compute the Voronoi diagram of a set of circles in a two-dimensional Euclidean space. The radii of the circles are non-negative and not necessarily equal. It is allowed that circles intersect each other, and a circle contains others.

The proposed algorithm constructs the correct Voronoi diagram as a sweepline moves on the plane from top to bottom. While moving on the plane, the sweepline stops only at certain event points where the topology changes occur for the Voronoi diagram being constructed.

The worst-case time complexity of the proposed algorithm is $O((n+m)\log n)$, where n is the number of input circles, and m is the number of intersection points among circles. As m can be $O(n^2)$, the presented algorithm is optimal with $O(n^2 \log n)$ worst-case time complexity.

Keywords: Euclidean Voronoi diagram; Circle; Sweepline; Event; Beach line.

1. Introduction

Suppose that a set of circles with non-negative different radii is given in a two-dimensional Euclidean space. Circles are allowed to be placed in arbitrary positions so that they may intersect each other and a circle may even contain others. Given a circle set, we assign every location in the plane to the closest member in the circle set. Then, the set of locations assigned to each circle forms a region called a Voronoi region, and the set of these regions forms a tessellation of the plane called the Voronoi diagram of the circle set. The distance measure employed in this paper is the ordinary Euclidean distance from the point on the circumference of a circle.

The Voronoi diagram of circles in the plane has various applications for important geometric problems. The computation of minimum cable packing many electricity wires, which is an NP-complete problem, is important for the design and simulation of automobiles, especially at the early phase of the design. The Voronoi diagram of circles has been used for the design of a very efficient heuristic algorithm [18]. Finding the geodesic, or the shortest path, among obstacles has been also an important problem and a recent study extends the usual facetted obstacles to circular obstacles by taking advantage of the proximity information provided by the Voronoi diagram of disk obstacles [7]. The Voronoi diagram has been even used for the simulation and mathematical modelling of the conductivity of a composite material consisting of copper and tungsten [11].

We also want to mention here that the algorithm for the Voronoi diagram of circles is the basis for one to compute the Voronoi diagram of spheres in higher dimensions [4,8]. In 3D, for example, the problem has various important applications in Nano-Technology and Bio-Technology. For example, efficiently analyzing the structural characteristics of proteins and identifying docking sites for the design of new drugs require an efficient representation of topology among atoms in the protein and the Voronoi diagram has been accepted as the best representation for these purposes [9,15].

There have been a few researches on this or related problems. Lee and Drysdale [10] considered the problem of the Voronoi diagram for a set of non-intersecting circles and proposed an algorithm running in $O(n \log^2 n)$ worst-case time, where n is the number of circles. Sharir [16] reported an algorithm with $O(n \log^2 n)$ worst-case time complexity for the Voronoi diagram of a circle set, where circles are allowed to intersect. Fotrune [2] reported a sweepline algorithm for a more generalized additively weighted Voronoi diagram in $O(n \log n)$ worst-case time. Yap [19] reported another $O(n \log^2 n)$ algorithm which is based on the divide-and-conquer scheme. Note that the Voronoi diagram of circles in a plane is a special case of an additively weighted Voronoi diagram where the

* Corresponding author. Tel.: +86 10 62782052; fax: +86 10 62771138.
E-mail address: shimin@tsinghua.edu.cn (S.-M. Hu).

0010-4485//$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2005.11.001

weights of points correspond to the radii of circles. Sugihara [17] reported an approximation algorithm and its implementation by approximating each circle with a set of points on the circle. After computing the Voronoi diagram of all points, the superfluous Voronoi edges and vertices are removed to produce an approximation for the desired Voronoi diagram. Recently, Kim et al. [5,6] reported an edge-flipping algorithm, which transforms the topological structure of point set Voronoi diagram for the centers of circles to the correct Voronoi diagram of circles. Even though the edge-flipping algorithm runs in $O(n^2)$ in the worst-case, it is simple, yet powerful, to implement for the correct Voronoi diagram.

In the previous researches, however, there is a one-to-one correspondence between a complete circle and a Voronoi region. In other words, a circle has a uniquely defined corresponding Voronoi region. When two circles intersect, therefore, the Voronoi edges are only defined exterior to both circles or interior to both circles. The locations exterior to one circle while interior to another are not considered in the previous works.

In this paper, we discuss the problem in a more general setting. We want to construct the correct Voronoi diagram regardless how the circles intersect each other so that every point in the plane is assigned to the closest point on all circles. Hence, the relation between a circle and the related Voronoi regions is one-to-many.

The basic idea used in the proposed algorithm in this paper is the sweepline approach suggested by Fortune [2]. The original sweepline algorithm for the point set Voronoi diagram defines two types of events (a *site event* and a *circle event*) and two types of curves in the plane (a *sweepline* and a *beach line*). Events facilitate modifying the topology of Voronoi diagram appropriately when a sweepline visits the event locations. In our algorithm for the circle set, we consider two more types of events: a *cross event* and a *merge event*. Similar to the original algorithm by Fortune for a point set, our algorithm modifies the topology of the Voronoi diagram of circles when a sweepline visits each event locations. After all the events are processed according to the specified rules to be discussed later, the desired Voronoi diagram is correctly and efficiently obtained in the plane.

The resulting Voronoi diagram in this paper forms a tessellation of the plane into a set of exclusive regions where each region corresponds to a circular arc or a circle. A region contains an arc when the arc is obtained by dividing a circle at intersecting points of the circle with others. Since n circles may result into $O(n^2)$ arc segments in the worst-case, the problem size of our algorithm is $O(n^2)$. It turns out that our sweepline algorithm takes $O(n^2 \log n)$ time in the worst-case to compute the correct Voronoi diagram for all arcs. Considering the quadratic problem size, our algorithm is optimal for n circles in the plane.

This paper is organized as follows. In Section 2, the basic concept of Fortune's sweepline algorithm for the Voronoi diagram of points in the plane is reviewed for the convenience of presentation for our algorithm for circles. After providing some terminologies in Section 3, we introduce four types of events, which are the most fundamental building blocks in Section 4. Then, we present the details of the main algorithm in Section 5. Section 6 presents the analysis of the worst-case time complexity and then the paper concludes.

2. Brief on Fortune's algorithm for Voronoi diagram of points

Fortune presented a sweepline algorithm, with $O(n)$ space and $O(n \log n)$ time complexity in the worst-case, for computing the Voronoi diagram of points in the plane [1,2]. In his algorithm, the *sweepline*, the *beach line*, and *events* are the most fundamental concepts. The sweepline is a horizontal straight line moving from top to bottom of the plane. The beach line, which is a set of consecutive parabola segments, splits the plane into two regions: a *safe region* and an *unsafe region*. A safe region contains a partial Voronoi diagram with a correctly computed topological structure, and an unsafe region corresponds to an undetermined part of the Voronoi diagram that has to be computed later. At a certain point in the plane, there is a well-defined set of actions that causes changes in the topological structure of the Voronoi diagram. Note that the point corresponds to a unique event and is called an *event location* in this paper. Hence, a topology change occurs when the sweepline visits an event location. Note that the beach line is monotone with respect to the sweepline.

In Fortune's algorithm, there are two types of events: a *site event* which corresponds to a point site, and a *circle event* which corresponds to the bottom-most point of the circle passing through three point sites. When the sweepline visits the location of a site event, the algorithm splits a parabola segment, which corresponds to the site event, in the beach line into two parts. Then, the algorithm inserts a new parabola segment between the two splitted segments, and adds a new Voronoi edge in the Voronoi diagram being constructed. When the sweepline visits the location of a circle event, the algorithm removes the corresponding parabola segment in the beach line, adds a Voronoi vertex, and creates a new Voronoi edge in the Voronoi diagram. At the location of each event, the algorithm calculates new potential circle events and inserts them into an event queue. After the sweepline visits the locations of all the events in the event queue, the construction of the Voronoi diagram is completed.

In Fortune's paper, he also discussed the sweepline algorithm to compute the Voronoi diagram of line segments and weighted points. We find that the sweepline idea can be applied to the construction of Voronoi diagram for circles in the plane, because a beach line can be constructed and correctly maintained while the sweepline moves in the plane.

3. Preliminaries

Let $G=\{g_1, g_2, \ldots, g_n\}$ be a set of generator circles $g_i=(c_i, r_i)$ for the Voronoi diagram where $c_i=(x_i, y_i)$ and r_i are the center and the radius of circle g_i, respectively. It is allowed that two circles intersect each other, and a circle contains others.

Suppose that there are l non-intersecting circles among n generator circles.

Let m be the number of intersection points among $n-l$ circles. If $n-l$ intersecting circles in G are divided at m intersection points, $2m$ circular arcs are obtained. For the simplicity of discussion, we assume that no three-generator circles intersect at the same point, and no four generator circles have a common tangent circle. Note that $m=2\binom{n}{2}$ in the worst-case of $l=0$.

Let $S=\{s_1, s_2, \ldots, s_{l+2m}\}$ be the set of all *sites* defined from G, where a site s_j refers to either a complete circle which does not intersect any others or an arc on a circle divided at intersection points. Therefore, there are $l+2m$ elements in the set S. For a site s and an arbitrary point z in the plane, let $d_s(z)$ be the shortest Euclidean distance between z and a point $p\in s$. Let $d_s(z)$ be the shortest Euclidean distance from z to the nearest site $s\in S$. Then, we assign a point z to the Voronoi region of a site s if, and only if, $d_s(z)=d(z)$. In other words, a site s defines an exclusive Voronoi region $VD(s)$. Then, the Voronoi diagram of S, $VD(S)$, is the collection of all such Voronoi regions. In other words, $VD(S)=\{VR(s_1), VR(s_2), \ldots, VR(s_{l+2m})\}$. In this paper, we say that a point q *majorizes* a point p when $y_p<y_q$ or $y_p=y_q$ and $x_p<x_q$, and denoted as $p<q$.

The topology of a Voronoi diagram in our problem can be represented in a standard data structure such as a doubly connected edge list [14], or equivalently in a winged-edge data structure [12], as a graph of Voronoi vertices, edges, and regions. While the equation of a Voronoi edge in the Voronoi diagram for point set is linear, its counterpart in the Voronoi diagram for circles is conic [6].

Shown in Fig. 1 are the entities (vertices, edges and regions in the Voronoi diagram) involved in the proposed algorithm. Let e be a Voronoi edge separating two Voronoi regions $VR(s_1)$, and $VR(s_2)$, where $\{s_1, s_2\}\subset S$ and $s_i\subseteq g_j\in G$. Note that s_i is an arc on a circle $g_j=(c_j, r_j)$. Then a point p on e satisfies one of the following equations:

$$d(p,c_1)-r_1 = d(p,c_2)-r_2 > 0, \tag{1}$$

$$r_1-d(p,c_1) = r_2-d(p,c_2) > 0, \tag{2}$$

$$r_1-d(p,c_1) = d(p,c_2)-r_2 > 0, \tag{3}$$

$$d(p,c_1)-r_1 = r_2-d(p,c_2) > 0, \tag{4}$$

In Fig. 1, Eq. (1) applies to e_1 and e_3, and Eq. (2) applies to e_2. The Voronoi edges, or equivalently the bisectors, e_4 and e_5 are from Eqs. (3) and (4), respectively. Rearranging the above system produces the follows: Eqs. (1) and (2) merge into an equation as

$$d(p,c_1)-d(p,c_2) = r_1-r_2 \tag{5}$$

which corresponds to the case that the edge e is either interior to both g_1 and g_2, or exterior to both circles. In this case, Eq. (5) defines a hyperbola. Similarly, Eqs. (3) and (4) produce

$$d(p,c_1)+d(p,c_2) = r_1+r_2 \tag{6}$$

when e is exterior to one circle while interior to the other. Note that Eq. (6) defines an ellipse.

Note that the definition of the distance between points $p\in\mathbf{R}^2$ and $q\in g(c, r)$ in this paper differ from that in the previous works such as Fortune [2] and Kim et al. [5]. While the distance is defined as $d(p, c)-r$ in the previous works, the distance in this paper is defined as $|d(p, c)-r|$. The difference is well-described in [13] in detail. Due to this difference in definitions, they have different Voronoi regions and therefore result in different Voronoi diagrams.

As suggested in Fortune's sweepline algorithm for the Voronoi diagram of points, we also define a beach line to separate the safe region and unsafe region. The safe region corresponds to the subset of plane that the sweepline has passed. Note that the partially computed Voronoi diagram in this region is correct. The other part of Voronoi diagram to be computed is in the unsafe region. The monotonicity of the beach line with respect to the sweepline is also preserved in our problem for circles.

In Fig. 2, the horizontal line S denotes a sweepline, and the beach line B is shown as a piecewise conic segments β's right above the sweepline. Note that B separates a safe region from an unsafe region. The thick solid curve connected to B is a correctly constructed Voronoi edge at the current instance of time. Then, a point $p\in B$, which lies above the sweepline S,

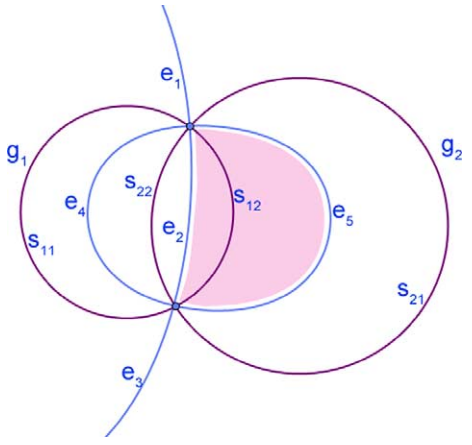

Fig. 1. Voronoi site, cell, vertex, and edge. g_1 and g_2 are circles, s_{11}–s_{22} are arc sites, and e_1–e_5 are Voronoi edges.

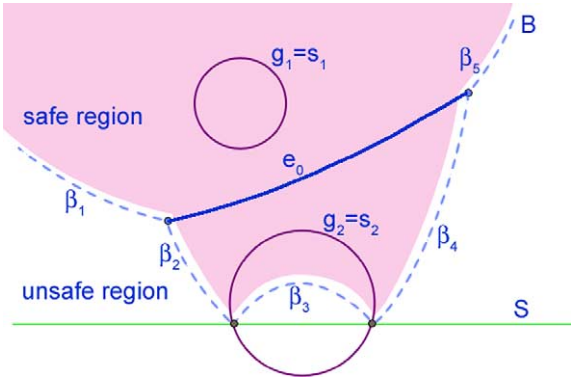

Fig. 2. Sweepline and beach line.

should satisfy the following constraint.

$$d(p,S) = \min(\mathrm{d}_s(p)) \tag{7}$$

where s is an arbitrary site. Since $s \subseteq g$, the above constraint can be divided into the following equations:

$$d(p,S) = d(p,c) - r, \quad \text{if} \quad p \quad \text{is} \quad \text{exterior to } g, \tag{8}$$

$$d(p,S) = r - d(p,c), \quad \text{if} \quad p \quad \text{is} \quad \text{interior to } g. \tag{9}$$

The above equations, therefore, denote that p is on a parabola by a directrix D and a focus c. Note that the directrix D for Eq. (8) is determined by translating the sweepline S downward by the distance r, while D for Eq. (9) is determined by translating the sweepline upward by the distance r. Hence, the beach line B consists of a set of parabola segments β's. Note that the intersection point p_0 of two neighboring parabola segments on B lies on a Voronoi edge since p_0 is equi-distant from two nearby circles. We also want to mention that a beach line edge β is associated with a unique corresponding generator g.

4. Events for line sweeping in Voronoi diagram of circles

Events are the most important elements in a sweepline algorithm. An event is associated with a type and a location. While Fortune's algorithm has two types of events, there are four types of events in the presented sweepline algorithm of the Voronoi diagram for circles.

4.1. Types of events

In the proposed algorithm, there are four types of events: a *site event*, a *circle event*, a *cross event*, and a *merge event*.

- *Site event*: This event occurs when the sweepline S first visits a generator circle. Hence, the top-most extreme point of a circle is the location of a site event since the sweepline always moves from top to bottom in the plane.
- *Cross event*: This event occurs when the sweepline S visits an intersection point between two generator circles. Hence, the location of this event is the position of an intersection point. Note that intersection points define the boundaries of sites.
- *Merge event*: This event occurs when the sweepline S leaves a generator circle. Hence, the bottom-most extreme point of a circle is the location of this event.
- *Circle event*: This event occurs when the sweepline S leaves a common tangent circle of three generator circles. Hence, the location of this event is the bottom-most extreme point of the common tangent circle.

It can be shown that the above four types of events fully cover all events in the construction of Voronoi diagram of circles. In Fortune's algorithm, there are two events: site events and circle events. A site event in Fortune is further refined into a site event and a merge event in the proposed algorithm since a point generator is now generalized to a circle. Since two circles may intersect, unlike point generators in Fortune's algorithm, the case for the intersection between two circle generators should be also considered for the correct maintenance of topology. The circle event remains in both Fortune's algorithm and the proposed algorithm even though the way to compute the Apollonius circle may be more complicated in this paper. Hence, four types of events suffice the enumeration of all events in the Voronoi diagram of circles in a plane.

4.2. Circle events

While the former three events (the site, the cross, and the merge events) are relatively straightforward to compute, it is not the case with the circle events.

To compute a circle event appropriately, it is necessary to compute circles tangent to three generator circles. To calculate the tangent circles, called Apollonius circles, we have adapted and extended the method proposed by Kim et al. [6], which is based on Möbius transformation and a point location problem.

While Fortune [2] and Kim et al. [5] considered the Apollonius circles tangent to three circles from outside only, we consider in this paper more configurations of possible tangent circles for three generator circles. Hence, we compute the Apollonius circles tangent to three circles not only from outside but also inside of some or all of the input circles. Since there are three generator circles, there can be eight different cases of Apollonius circles.

Shown in Fig. 3 are the cases that an Apollonius circle defined from outside generators (Fig. 3(a)), defined from outside two generators but inside one generator (Fig. 3(b)), and so on. Hence, an appropriate Apollonius circle can be computed by simply looking at its relationship with respect to the generators.

Given a set of three generator circles, there are at most eight Apollonius circles. However, it is not necessary to compute all eight Apollonius circles to choose a correct one since the orientations of beach line edges provide the information for the configuration of a desired Apollonius circle.

If the parabola for a beach line edge β is open to positive y-direction, the Apollonius circle to be computed is located outside the generator circle g corresponding to β. If the parabola for β is open to negative y-direction, the Apollonius circle is located inside the circle g corresponding to β. Hence, the desired configuration from which the Apollonius circle has to be computed can be found by simply looking at the geometries of three consecutive beach line segments [3].

However, it should be noted that there could be zero, one, or two Apollonius circles for a given configuration of three generator circles depending on the conditions [3]. When there are two instances of tangent circles for a given configuration (for example, a small generator between two large generators), the situation can be complicated and has to be handled carefully. In such a case, the correct solution can be identified easily if we consider the relative orientations of the tangent points between generator circles and the Apollonius circle.

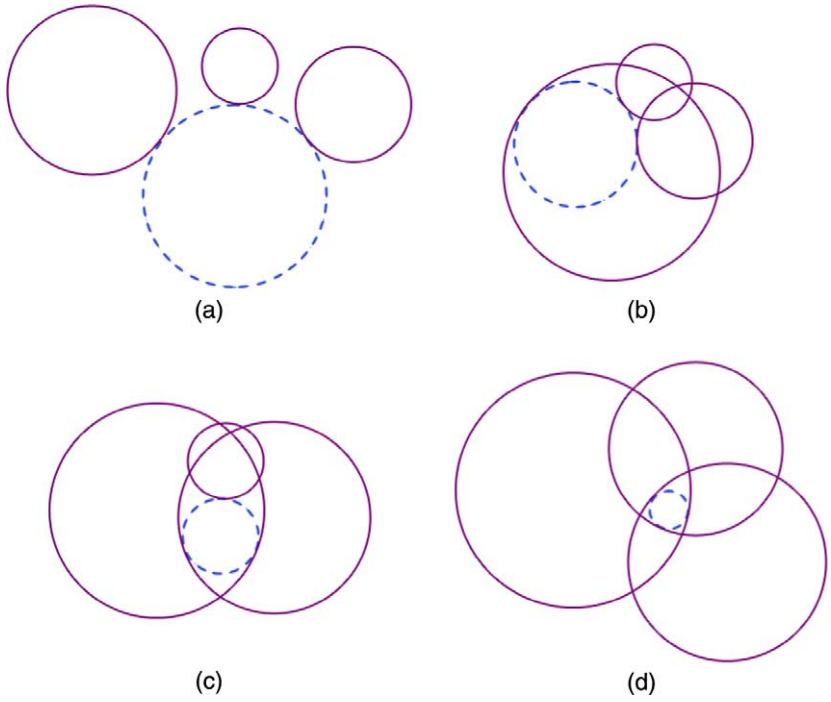

Fig. 3. Four different configurations of an Apollonius circle, where the circle is tangent to three generators. (a) All from outside, (b) one generator from inside while two generators from outside, (c) two from inside while one from inside, and (d) all from inside.

5. Line sweeping for Voronoi diagram of circles

The proposed sweepline algorithm runs event-by-event starting from top to bottom in the plane. The algorithm consists of three major steps: initialization, event processing, and post processing.

5.1. Initialization

In the initialization step, we first find both the top and bottom extreme points for all input circle generators and enqueue them into an event queue Q. We also compute all intersection points between the generators and enqueue them into Q as well. Then, we sort all the events in Q in a descending order with respect to the location of points. In other words, place higher priority to p than q if $p > q$. Finally, we create an initial sweepline S and a beach line B.

5.2. Event processing

Once the event queue Q is ready, the sweepline processes all events by passing through event locations from the first to the last in Q. This is done by iterating the actions discussed in the below after dequeuing an event at a time from Q. Each event is associated with its own unique actions to be done which consists of three major elements as follows:

- modifying the topology of the beach line B by splitting, removing, or adding some beach line edges appropriately
- modifying the current event queue Q by removing meaningless circle events and adding new circle events appropriately
- modifying the topology of the Voronoi diagram in the safe area by adding or merging some Voronoi edges appropriately

5.2.1. Site event

Fig. 4 shows the topology structures of the Voronoi diagram and the beach line before and after the processing of a site event. In this figure, the generator circle g_4 has a corresponding site event E.

As shown in Fig. 4(a), the position of site event E is at the top extreme point of g_4. We first locate a beach line edge β_2, which corresponds to the location of event E. Then, β_1 and β_3 which are immediately left and right to β_2 on the beach line B can be easily located if B is properly maintained. Furthermore, the generator circles g_1, g_2, and g_3, which correspond to β_1, β_2, and β_3 are also immediately identified.

When we process the site event E, we modify the beach line B, the Voronoi diagram, and the event queue Q. For the beach line B, we split β_2 into two beach line edges β_2' and β_2'' as shown in Fig. 4(b), and insert three new beach line edges β_4, β_5, and β_6. Note that these new beach line edges correspond to g_4 in-between β_2' and β_2''. Note also that β_4 and β_6 are convex downward since they are at the outside of g_4, and β_5 is convex upward since it is inside g_4. While the foci of β_2' and β_2'' remain identical to the focus of β_2, the foci of β_4, β_5 and β_6' are the center of the circle g_4. The directrixes of all the five beach line edges are the sweepline after appropriate translations.

The Voronoi diagram itself can be correctly updated by creating a new Voronoi edge e_3 defined by generator circles g_2 and g_4 and connecting with B appropriately. For the proper maintenance of the event queue Q, we use (β_1, β_2', β_4) to

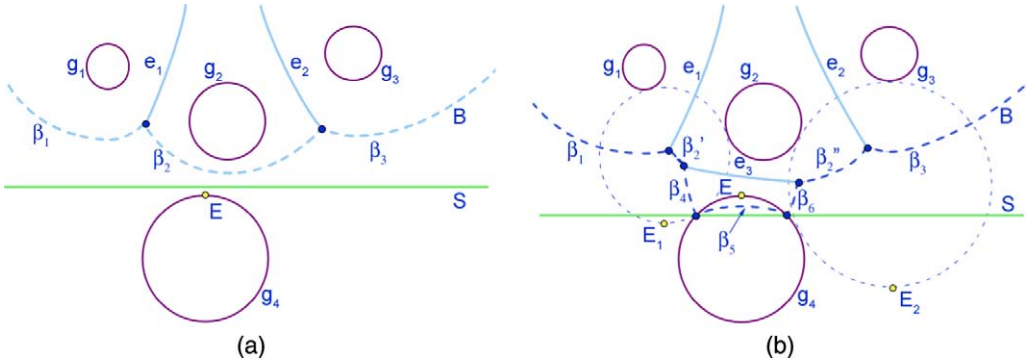

Fig. 4. Processing a site event when the sweepline S touches a site event E. (a) before the touch, (b) after the touch.

compute an Apollonius circle, and therefore a corresponding circle event E_1 if the Apollonius circle exists. Then, we also compute the second circle event E_2 from (β_6, β_2, β_3) in a similar way of E_1. Both E_1 and E_2 are enqueued into Q in the sorted order.

At any time instance, a valid circle event is defined by a triplet of three neighboring beach line edges. Whenever the local topology among these beach line edges is changed by removal or insertion of one or more edges, the corresponding circle event becomes invalid. Then, such an invalid circle event should be removed from the event queue Q.

Since the processing of a site event E changes the topology connections among β_1, β_2 and β_3 due to the split of β_2, the circle event corresponding to (β_1, β_2 and β_3) should be removed from the event queue Q. Hence, processing a site event deletes one circle event while creating two new circle events.

5.2.2. Merge event

Let E be a merge event corresponding to the bottom point of a circle g_2 in Fig. 5. A merge event falls into two cases: a case with one and only one beach line edge in the circle g_2 (Case 1), and a case with more than one beach line edges in g_2 (Case 2). Fig. 5(a) and (b) illustrate Case 1, and Fig. 5(c) and (d) illustrate Case 2. The left and the right columns of Fig. 5 depict the topological structure before and after the merge event is processed, respectively.

When a merge event occurs, beach line edges interior to the corresponding circle generator are removed. In Case 1, as shown in Fig. 5(a) and (b), the beach line edge β_3 is removed. In Case 2, as shown in Fig. 5(c) and (d), on the other hand, the beach line edges β_3, β_3, and β_3 are removed. After removing the beach line edges, two immediately neighboring beach line edges β_2 and β_4, which are incident to the removed edges and exterior to the circle generator g_2, are merged together to form one single beach line edge, β_{24} in the figure. The focus of β_{24} is the center of g_2 and the directrix is the sweepline after an appropriate translation.

Since the removed beach line edges modify some local topological structure among beach line edges, there can be some circle events that are not valid any more. Hence, the circle events related to the removed beach line edges have to be removed from the event queue Q regardless the type of event being processed except a merge event. In the case of merge event, however, there is no circle event to be removed from Q since both consecutive beach lines β_2 and β_3, for both Case 1 and Case 2, in Fig. 5 are associated with the same generator circle g_2.

In the case of a merge event, new circle event corresponding to the merged new beach line edge, β_{24} in Fig. 5, should be computed and enqueued into Q. Note that there can be one possible new circle event to be created.

Then, it is necessary to appropriately update the topology structure of the Voronoi diagram. In Case 1, it is not necessary to do any action for the topology of the Voronoi diagram since there is no removed or created Voronoi vertex or edge. Whether the center of the circle g_2 corresponding to the event E is handled as an isolated Voronoi structure or not is immaterial for the discussion and we simply ignore it from the complete Voronoi structure.

In Case 2, on the other hand, the existing Voronoi vertices interior to the circle g_2 are connected to the removed beach line edges at the corresponding Voronoi vertices. Since the beach line edges are removed, the incident Voronoi edges, e_2 and e_4 shown in Fig. 5(c) has to be merged to form one single new Voronoi edge e_{34}. This is why this event is called a merge event. We want to mention here that the seemingly special case that the circle g_4 is completely contained inside the circle g_2 can be also handled in the same treatment. In such a case, the edges e_3 and e_4 are from an identical edge and therefore a completely isolated loop, without even a vertex, results when they are merged. Note that the centers of both g_2 and g_4 in the figure play the role of foci for the new Voronoi edge of an ellipse.

In both cases, there are at most three beach line edges inside the generator circle g_2 just before processing the merge event. However, in a special case as shown in Fig. 5(e), a circle event can coincide a merge event. Even though there are four beach line edges inside g_2 in such a case, it can be handled by processing the circle event before the merge event. After the proper processing of the circle event, the situation is simplified to the case of either Fig. 5(a) or (c).

5.2.3. Cross event

Suppose that a cross event E is defined by two circle generators $g_2=(c_2, r_2)$ and $g_3=(c_3, r_3)$, as shown in Fig. 6(a). Immediately before the cross event E is processed in the figure,

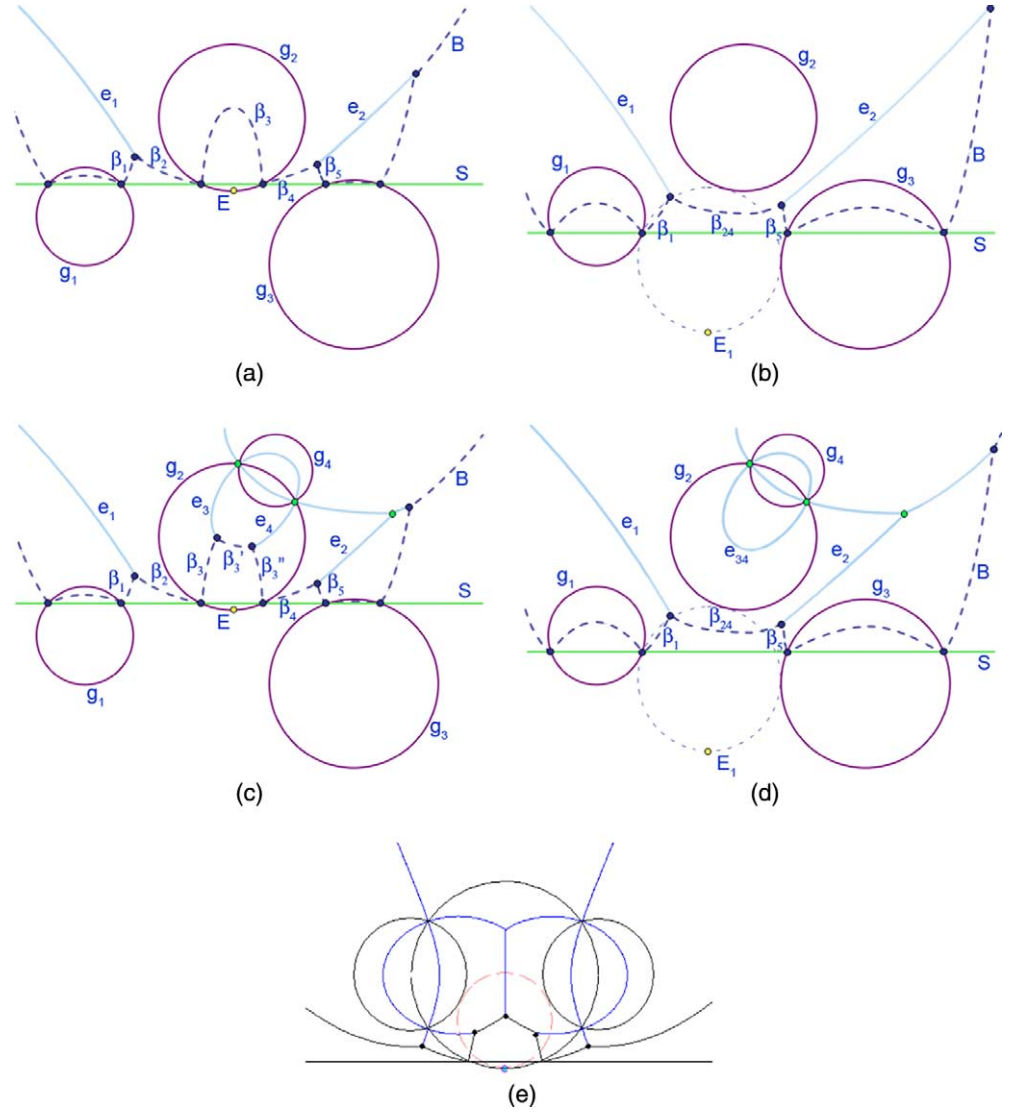

Fig. 5. Processing of a merge event. (a) Before the processing for the case of a single generator, (b) after the processing, (c) before the processing for the case of more than one generator involved in the processing, (d) after the processing, (e) special case: circle events and a merge event are at the same position.

there are two beach line edges, β_3 and β_4, approaching to the location of the event E.

As shown in Fig. 6(b), the beach line edges β_3 and β_4 are removed and four new beach line edges β_7, β_8, β_9, and β_{10} are created between the beach line edges β_2 and β_5. Note that the intersection point corresponding to the cross event E now becomes a new Voronoi vertex for the existing Voronoi edge e_1. The foci for β_7, β_8, β_9 and β_{10} are c_3, c_3, c_2, and c_2, respectively. Similarly, the directrices are also defined as the sweepline translated accordingly by the amount of r_3 for β_7 and β_8, and by the amount of r_2 for β_9 and β_{10}.

In the case of a cross event, there is no circle event corresponding to the removed beach line edges. In other words, two possible triplet of consecutive beach line edges $(\beta_1, \beta_2, \beta_3)$, $(\beta_2, \beta_3, \beta_4)$, $(\beta_3, \beta_4, \beta_5)$, and $(\beta_4, \beta_5, \beta_6)$ do not define any circle event. However, there can be two possible new circle events defined by two triplets $(\beta_1, \beta_2, \beta_7)$ and $(\beta_{10}, \beta_5, \beta_6)$. They should be computed and inserted in Q.

Since the safe region is increased, the corresponding topology structure of the Voronoi diagram should also be updated. The intersection point which define the cross event E now becomes a new Voronoi vertex v and becomes the end vertex of the existing Voronoi edge e_1. Then, three new Voronoi edges, e_2, e_3 and e_4 are created with v as the starting vertices while their end vertices are not defined yet. Note that the degree of the Voronoi vertex v is four, not three. Therefore, a special treatment is necessary to cope with this vertex to handle the spatial queries regarding on the spatial structure correctly.

Even though the cross event is discussed for the case that the safe region expands into the intersection region, the other case of leaving an intersection region, as shown in Fig. 6(c) and (d), can be easily described similarly.

5.2.4. Circle event

Let A be the common tangent Apollonius circle corresponding to a circle event E. Then, there are three generator circles

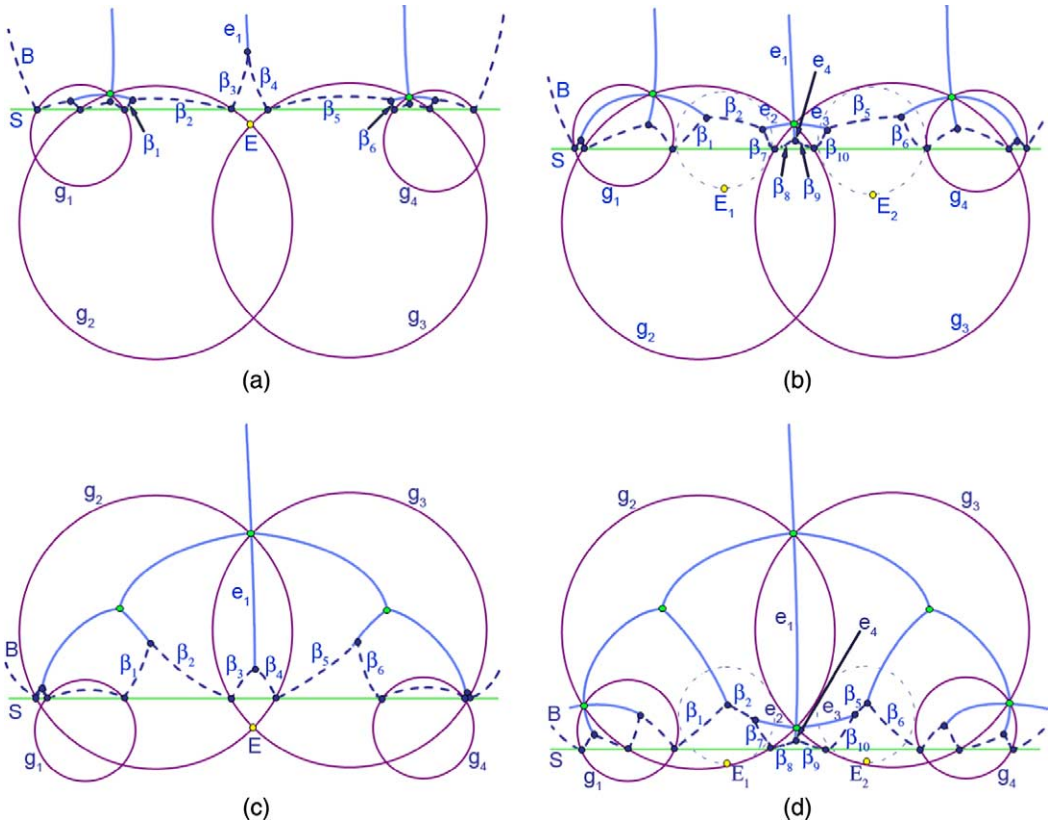

Fig. 6. Processing a cross event. (a) Before the processing for the case of entering an intersection region, (b) after the processing, (c) before the processing for the case of leaving the intersection region, and (d) after the processing.

corresponding to the event E and A is simultaneously tangent to these three circles.

Shown in Fig. 7(a) is the simplest case of a circle event. Let A be the Apollonius circle simultaneously tangent to three-generator circles g_2, g_3, and g_4 in the figure. Among the three beach line edges β_2, β_3, and β_4 corresponding to g_2, g_3, and g_4, respectively, the middle beach line edge β_3 has to be removed. Then, a new Voronoi vertex v is created at the center of the Apollonius circle A and is connected to the two existing Voronoi edges e_1 and e_2 as their end vertices. Then, a new Voronoi edge e_3 is also created having v as its starting vertex. Note that e_3 is connected with two beach line edges β_2 and β_4 and its equation is defined by two corresponding circle generators g_2 and g_4.

While a circle event removes one circle event from the event queue Q, it creates two more new circle events and enqueues them into Q. New circle events are defined as follows. Let another beach line edge left to β_2 be β_1 and the other edge right to β_4 be β_5 as shown in Fig. 7(b). Let E_1 be the circle event corresponding to a triplet of beach line edges (β_1, β_2, β_4). Another event E_2 can be also defined similarly for a triplet of (β_2, β_4, β_5). Then, these new circle events are enqueued into Q. On the other hand, two circle events, (β_1, β_2, β_3) and (β_3, β_4, β_5), associated with β_3 has to be removed since β_3 has been removed. Note that, in the circle event, no new beach line edge is created.

Shown in Fig. 8 are different cases of circle events. In Fig. 8(a), the Apollonius circle A is interior to one generator while exterior to two other generators. The other cases can be similarly interpreted. In all of these cases, the above-explained algorithm can be identically applied without any problem.

5.3. Postprocessing

The last event in the whole process is either a merge event or a circle event. After processing the last event in the event queue Q, the Voronoi diagram is about to be completed for the whole plane. However, in the half-space below the sweepline at the last event, an unsafe region is still remained in the plane. The region above the still-existing beach line is only safe.

The beach line after the last event still consists of some beach line edges, where each beach line edge is defined by a corresponding generator circle and the sweepline. Each pair of two consecutive beach line edges is connected with an already constructed Voronoi edge in the safe region where the end Voronoi vertex of the Voronoi edge is not yet determined. Note that the Voronoi region for the generator circle contributing the boundary of the convex hull of the whole input circles is unbounded. This means that two edges on the boundary of such a Voronoi region are also unbounded. Since the beach line at this moment plays the role of infinity, the Voronoi edges intersecting, and therefore connected to, the beach line should emanate to infinity. Hence, the not yet determined Voronoi vertices can now be declared to be the infinite vertices. Then the correct Voronoi diagram for the whole input data can be completed by removing the beach line and the sweepline.

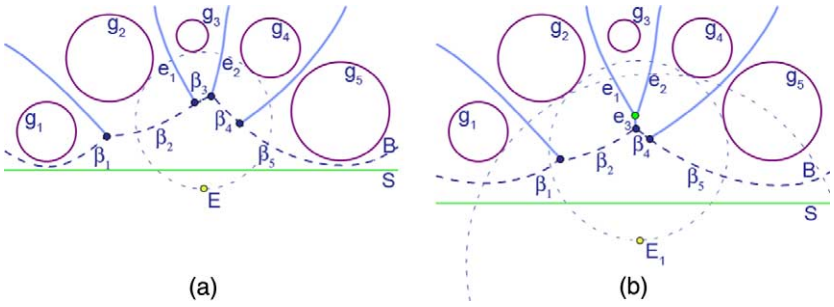

Fig. 7. Processing of a circle event for the case when a Apollonius circle is tangent to generators from outside. (a) before the processing, and (b) after the processing.

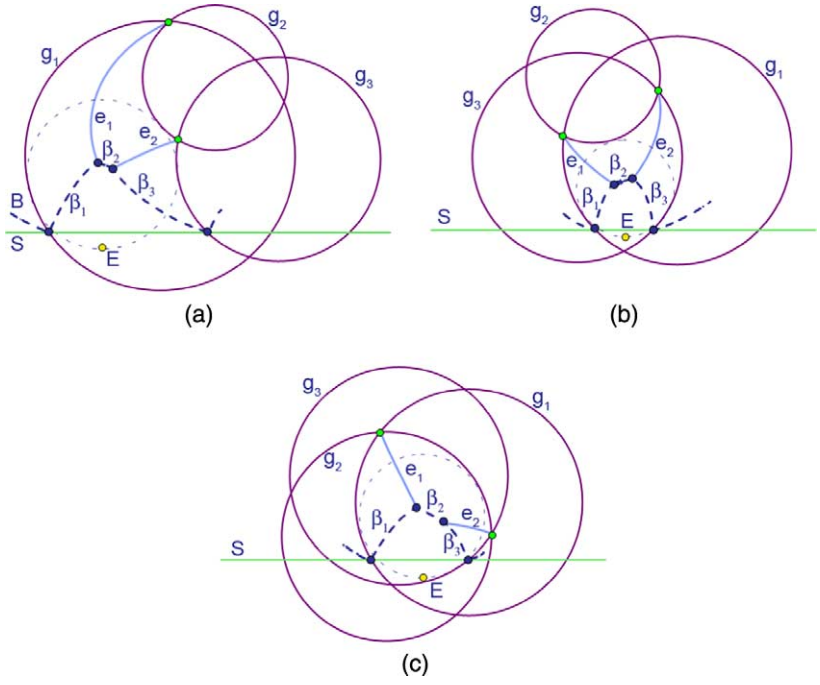

Fig. 8. Processing of a circle event, where the circle is tangent to three generators (a) one generator from inside while two generators from outside, (b) two from inside while one from inside, and (c) all from inside.

5.4. Algorithm

The above described can be summarized as an algorithm in a pseudocode as follows:

Algorithm (Sweepline for the Voronoi Diagram of Circles)
Input: A set of circles G
Output: Voronoi diagram for G
1. Initialization:
(1) create an initial beach line edge corresponding to the circle at infinity and a sweepline S.
(2) insert site, merge, and cross events into a event queue Q.
2. Event processing:
(1) pop an event E from Q
(2) switch(E)
- CASE: Site Event
  (a) find corresponding beach line edge β
  (b) split β into two beach line edges β_l and β_r, and insert three new beach line edges between β_l and β_r.
  (c) create a Voronoi edge e
  (d) connect e with beach line B
  (e) create two circle events and push them into Q
- CASE: Merge Event
  (a) find corresponding beach line edge β and two beach line edges β_l and β_r connected to β
  (b) check if β_l and β_r are exterior to the corresponding circle
    (i) if yes:
      (A) remove β
      (B) merge β_l and β_r to make one beach line edge β_{lr}
    (ii) if no:
      (A) merge two Voronoi edges to create one Voronoi edge
      (B) merge left(β_l) and right(β_r) to make one beach line edge β_{lr} (NOTE: left(β) and right(β) denotes left and right adjacent beach line edge to β, respectively.)

(C) remove three beach line edges β_l, β, and β_r
(c) create one circle event defined by a triplet (left(β_{lr}), β_{lr}, right(β_{lr})) and push it into Q
- CASE: Cross Event
  (a) find corresponding two beach line edges β_l and β_r
  (b) create four beach line edges and connect them appropriately
  (c) create three Voronoi edges and connect them appropriately.
  (d) create two circle events and enqueue into Q
  (e) remove the two beach line edges β_l and β_r
- CASE: Circle Event
  (a) retrieve corresponding beach line triplet (β_l, β, β_r)
  (b) check if the current beach line triplet is valid (i.e., if the circle event corresponding to this beach line triplet is realized) or not
    (i) if valid:
      (A) find two Voronoi edges e_1 and e_2 connected to β
      (B) create a Voronoi vertex v
      (C) connect e_1 and e_2 at v
      (D) create new Voronoi edge having v as an end vertex
      (E) connect two beach line edges β_l and β_r neighboring β
      (F) create two circle events defined by two triplets (left(β_l), β_l, β_r) and (β_l, β_r, right(β_r)) and push them into Q
      (G) remove β
    (ii) if not valid, do nothing

3. Postprocessing:
   (1) handle Voronoi vertices connected to B as infinity
   (2) delete the beach line B

6. Analyses for time and space complexity

In this section, we will provide the analyses of the time and space complexity of the presented algorithm in the worst-case. Let n be the number of generator circles provided as input, l be the number of non-intersecting generator circles, and m be the number of intersections between generator circles. Hence, the total number of sites, and therefore the total number of the corresponding Voronoi regions as well, to be considered in the computation of the Voronoi diagram are $l+2m$. Note that $m = 2\binom{n}{2} = n(n-1) = O(n^2)$ in the worst-case since a pair of circles can intersect at two points. Therefore, $n \le l+2m \le n(n-1)$.

6.1. Initialization and postprocessing steps

In the initialization step, there are n events for each of site events and merge events since there are one site event and one merge event for each generator circle. Since there are m intersections, there is the same number of cross events. Hence, these three events can be stored in $O(2n+m)$ space. Since there is a sorting operation, the initialization step takes $O((2n+m)\log(2n+m)) = O((2n+m)\log n) = O(n^2 \log n)$ time in the worst-case.

In the postprocessing step, there are some Voronoi edges left to be declared to go infinity and the existing beach line edges should be destroyed. In the worst-case, there can be $l+2m-1$ Voronoi edges connected to the beach line B. Note that the number of beach line edges in B after the final event is processed can be $2(l+2m-1)+1 = 2l+4m-1$ in the worst-case. This number is indeed the upper bound of beach line edges in general, and can be interpreted as follows: a site can only create one more new beach line edge in the middle of an existing beach line edge. Hence, both the time and space complexity for the post-processing are $O(2l+4m-1) = O(l+2m)$ in the worst-case.

6.2. Event processing step

During the processing of an event, the beach line changes its topology. Note that each pair of two consecutive beach line edges is connected to a Voronoi edge. Therefore, the space complexity of the beach line is linear to the complexity of the number of Voronoi edges. Therefore, the beach line can be maintained in $O(l+2m)$ space in the worst-case. The time complexity to handle each event is as follows.

6.2.1. Site event

Since the beach line can consist of $2(l+2m)-1$ parabolic arcs, locating an appropriate beach line edge from the beach line requires $O(\log(2l+4m-1)) = O(\log n)$ time in the worst-case if the topology of the beach line B is stored in an efficient data structure such as a balanced binary search tree. Recall that B is monotone w.r.t. the sweepline.

Once an appropriate beach line edge is located, two new circle events are created and inserted into the event queue Q at an appropriate place. Hence, this insertion takes $O(\log|Q|)$ time in the worst-case where $|Q|$ denotes maximum number of events in Q. Since the other actions take only $O(1)$ time and $|Q|$ can be $O(n)$, which will be explained later, handling the whole site event takes $O(n \log n)$ time in the worst-case since there are n generator circles where each circle creates a site event.

6.2.2. Merge event

Since there are n generator circles, so are n merge events. Processing of a merge event requires to remove some beach line edges from the binary tree and create one circle event to insert into the event queue Q. Hence, these operations altogether take $O(\log n)$ time in the worst-case. The other operations take only $O(1)$ time. Therefore, processing all merge events takes $O(n \log n)$ time in the worst-case.

6.2.3. Cross event

The maximum possible number of cross events is $m = n(n-1) = O(n^2)$. Processing a cross event consists of removing two

beach line edges and creating four beach line edges, and creating three new Voronoi edges. Note that each of removing and inserting a beach line edge from and to the binary tree takes $O(\log\ n)$ time. However, creating a Voronoi edge and connecting it appropriately take only $O(1)$ time. In addition, two new circle events are generated and inserted into the event queue Q, which takes also $O(\log n)$ time. Therefore, processing all cross events takes $O(m\ \log\ n)=O(n^2\ \log\ n)$ time in the worst-case.

6.2.4. Circle event

For each circle event, one beach line edge is removed from B, two circle events are removed from Q, and two new circle events are created to be inserted into Q. Hence, this process takes $O(\log |Q|)$ time where $|Q|$ denotes the maximum size of the event queue Q.

Although there are many circle events generated in the processing, the total number of possible circle events to be processed is under a certain level, which is shown below.

Note that the number of non-intersecting circles is l and the number of intersections from the other $(n-l)$ circles is m. Then, the number of sites, and therefore the number of Voronoi regions, is $(l+2m)$. Suppose that no circle is completely contained by another circle. Then, the number of vertices of the Voronoi diagram will not be larger than $2(l+2m)-5$, including m intersection points, since the Voronoi diagram is a planar graph. (It is known that a Voronoi diagram of N points has at most $2N-5$ vertices and $3N-6$ edges. Equality holds when the vertex degree is exactly 3.) If circles are allowed to be contained by others, the cardinality becomes even smaller. In addition, the Voronoi vertices at the intersection points have degree four and this fact may reduce the cardinality even smaller.

A vertex is created only from the center of an Apollonius circle corresponding to a circle event, except the vertices corresponding to m intersection points. Hence, the number of vertex creations, and therefore the related topology updates, never exceeds $2(l+2m)-5-m$. Hence, both the space and time complexity of circle event processing are $O(l+m)$.

Note that a site event creates two circle events. Since there are n circles, $2n$ circle events are created from site events. One merge event creates one circle event. Therefore, n circle events are created from merge events. Similarly, $2m$ circle events are created from m cross events. Note that some circle events will be removed in the middle of the process. When a circle event is processed, two new circle events are also created. In addition, the number of the process of the circle events is the same as the number of vertices, which is bounded by $2(l+2m)-5-m$ as explained earlier. Hence, the total number of circle events created in the plane sweepping can be obtained by summing up the number of circle events enqueued in the process circle events and the one enqueued by the other events, and is $2(2(l+2m)-5-m)+(2n+n+2m)=3n+4l+8m-10<7n+8m$.

Therefore, the total space and time complexities can be obtained by summing up all the time complexities in the above to result in $O(n+m)$ and $O((n+m)\log\ n)$ where $0\leq m\leq n(n-1)$.

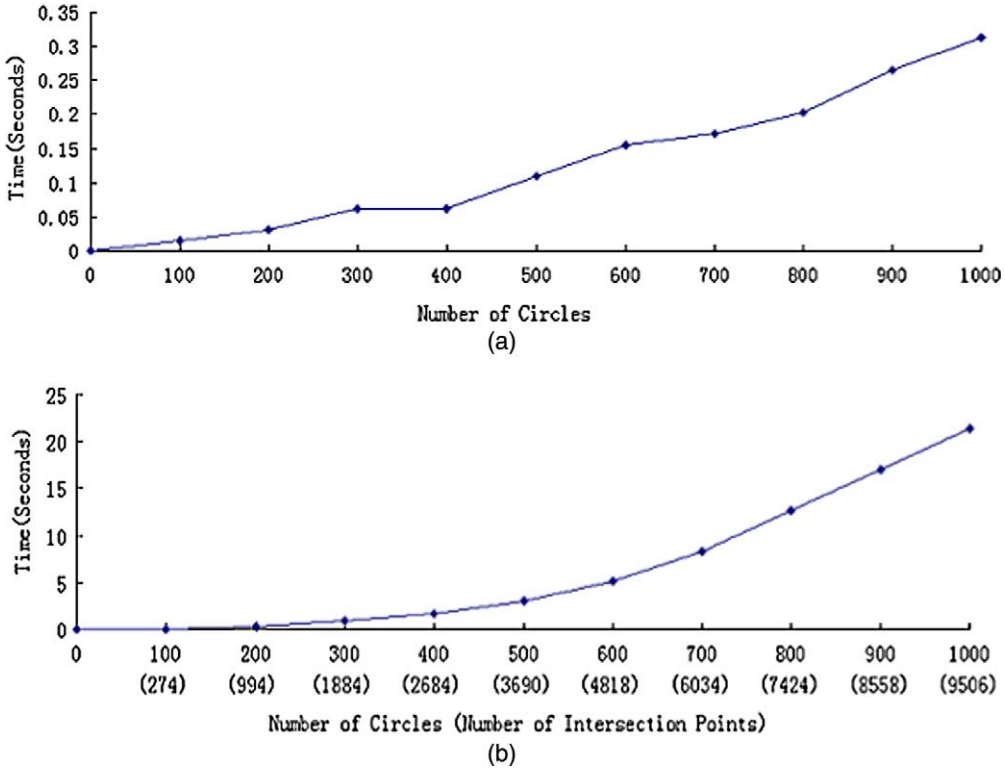

Fig. 9. Statistics of computation time for various sets of randomly generated circles. The numbers in the horizontal axis denote number of circles where the numbers in the parenthesis are the number of intersections among the circles. The vertical axis denotes the computation time. (a) No intersection is allowed between circles. (b) Circles are allowed to intersect each other.

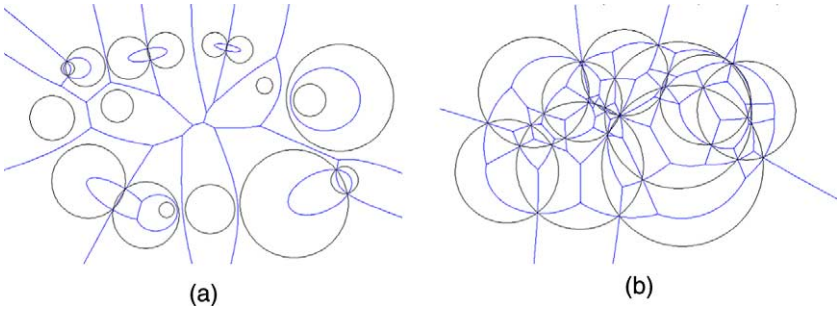

Fig. 10. Examples of Voronoi diagram of circles computed by the proposed sweepline algorithm.

7. Discussions and conclusions

We have presented a sweepline algorithm for computing the Voronoi diagram for circle in the plane. Circles are allowed to be placed in arbitrary locations, and therefore they may intersect each other and a circle may even contain some others. The radius of a circle is allowed to be non-negative meaning that it can degenerate to a point.

In this paper, we discussed the construction of the Voronoi diagram for circles in a more general setting. We want to construct the correct Voronoi diagram regardless how the circles intersect each other. In the previous researches, the Voronoi edges are only defined exterior to both circles or interior to both circles. The locations exterior to one circle while interior to another are not considered in their works. In this paper, every point in the plane is assigned to the closest point on all circles. Hence, the relation between a circle and the corresponding Voronoi regions is one-to-many.

The time complexity of the proposed algorithm is $O((n+m)\log n)$, which is input sensitive but will not exceed $O(n^2 \log n)$ in the worst-case. The algorithm uses a sweepline to sweep through the plane and makes the Voronoi diagram grow. The sweepline only stops at the event points, where the topology of the growing Voronoi diagram will be changed.

The proposed algorithm has been fully implemented in Microsoft C++ environment and tested against various data sets. Fig. 9 shows some statistics of the run-time behavior from our experiments. We want to note that our current implementation is not fully optimized yet. Since the beach line, in our current implementation, is arranged not as a binary tree but as a linked list, the time behavior shows $O((n+m)^2 \log n)$ instead of $O((n+m)\log n)$. Fig. 10 shows two examples of the computed Voronoi diagram from the implementation.

We expect that the approach in this article can be applied to more complex cases of Voronoi diagram such as the Voronoi diagram for polygons consisting of line segments as well as circular arcs.

Acknowledgements

This work was supported by the Natural Science Foundation of China (Project number 60225016, 60321002). D. Kim and D.-S. Kim were supported by Creative Research Initiatives from the Ministry of Science and Technology in Korea. We thank Chang Huang from Department of Computer Science and Technology, Tsinghua University, for the discussions and the development of the user interface during the implementation.

References

[1] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational geometry: algorithms and applications. 2nd ed. Berlin: Springer; 2000.
[2] Fortune S. A sweepline algorithm for Voronoi diagrams. Algorithmica 1987;2:153–74.
[3] Kim D, Kim D-S, Sugihara K. Apollonius tenth problem via raidus adjustment and Möbius transformations. Comput Aided Des; 2006;38(1): 14–21.
[4] Kim D, Kim D-S. Region-expansion for the Voronoi diagram of 3D spheres, Comput Aided Des; in press, DOI: 10.1016/.jcad.2005.11.007.
[5] Kim D-S, Kim D, Sugihara K. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. Comput Aided Geom Des 2001;18(6):541–62.
[6] Kim D-S, Kim D, Sugihara K. Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. Comput Aided Geom Des 2001;18(6):563–85.
[7] Kim D-S, Yu K, Cho Y, Kim D, Yap C. Shortest paths for disc obstacles. Lect Notes Comput Sci 2004;3045:62–70.
[8] Kim D-S, Cho Y, Kim D. Euclidean Voronoi diagram of 3D balls and its computation via tracing edges. Comput Aided Des 2005;37:1412–24.
[9] Kim D-S, Cho C-H, Cho Y, Kim D. Pocket recognition on a protein using Euclidean Voronoi diagram of atoms, J Mol Graph Model. Submitted for publication.
[10] Lee DT, Drysdale RL. Generalization of Voronoi diagrams in the plane. SIAM J Comput 1981;10(1):73–87.
[11] Lee YJ, Kim D-G, Kim G-S, Kim D-S, Kim YD. Effect of the W–W contiguity on conductivity of W-Cu composite using the Voronoi diagram. Z fuer Metallkund 2005;3:255–8.
[12] Mäntylä M. An introduction to solid modeling. Rockville, MD: Computer Science Press; 1988.
[13] Okabe A, Boots B, Sugihara K, Chiu SN. Spatial tessellations concepts and applications of voronoi diagrams. 2nd ed. London: Wiley; 1999.
[14] Preparata FP, Shamos MI. Computational geometry: an introduction. New York: Springer; 1985.
[15] Ryu J, Park R, Kim D-S. Molecular surfaces on proteins via beta shapes, Comput Aided Des. Submitted for publication.
[16] Sharir M. Intersection and closest-pair problems for a set of planar discs. SIAM J Comput 1985;14(2):448–68.
[17] Sugihara K. Approximation of generalized Voronoi diagrams by ordinary Voronoi diagrams. Graph Models Image Process 1993;55(6): 522–31.
[18] Sugihara K, Sawai M, Sano H, Kim D-S, Kim D. Disk packing for the estimation of the size of a wire bundle. Jpn J Ind Appl Math 2004;21(3): 259–78.
[19] Yap CK. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. Discrete Comput Geom 1987;2:365–93.

Li Jin is a PhD student in Department of Computer Science and Technology at Tsinghua University. He got bachelor's degree in Computer Science at Tsinghua University in 2003. His research interests are Computer Graphics, CAGD and Computational Geometry.

Donguk Kim is a senior researcher in Voronoi Diagram Research Center at Hanyang University, Korea. He received his B.S., M.S. and Ph.D. degrees from Hanyang University in 1999, 2001 and 2004, respectively. His research interests include computational geometry, geometric modeling and their applications in the molecular biology.

Lisen MU is a PhD student in Department of Computer Science and Technology at Tsinghua University. He got bachelor's degree in Computer Science at Tsinghua University in 2003. His research interests are Grid Computing and Computational Geometry.

Deok-Soo Kim is a professor in Department of Industrial Engineering, Hanyang University, Korea. Before he joined the university in 1995, he worked at Applicon, USA, and Samsung Advanced Institute of Technology, Korea. He received a B.S. from Hanyang University, Korea, an M.S. from the New Jersey Institute of Technology, USA, and a Ph.D. from the University of Michigan, USA, in 1982, 1985 and 1990, respectively. His current research interests mainly lie in the theory and applications of Voronoi diagram while he has been interested in various geometric problems. He is current the director of Voronoi Diagram Research Center supported by the Ministry of Science and Technology, Korea.

Shi-min Hu is currently a professor of computer science at Tsinghua University. His research interests include digital geometry processing, video-based rendering, rendering, computer animation, and computer-aided geometric design. He obtained his Ph.D. in 1996 from Zhejiang University. He is on the editorial boards of Computer Aided Design.