Parallel computing 2D Voronoi diagrams using untransformed sweepcircles

Shi-Qing Xin\textsuperscript{a}, Xiaoning Wang\textsuperscript{b}, Jiazhi Xia\textsuperscript{c,e,*}, Wolfgang Mueller-Wittig\textsuperscript{b,d}, Guo-Jin Wang\textsuperscript{f}, Ying He\textsuperscript{b}

\textsuperscript{a} Ningbo University, China
\textsuperscript{b} School of Computer Engineering, Nanyang Technological University, Singapore
\textsuperscript{c} BeingThere Center, Institute for Media Innovation, Nanyang Technological University, Singapore
\textsuperscript{d} Fraunhofer IDM@NTU, Nanyang Technological University, Singapore
\textsuperscript{e} School of Information Science and Engineering, Central South University, China
\textsuperscript{f} State Key Lab of CAD&CG, Zhejiang University, China

\textbf{a r t i c l e i n f o}

Keywords:
Voronoi diagram
GPU
Sweep line
Sweep circle
Parallel algorithm

\textbf{a b s t r a c t}

Voronoi diagrams are among the most important data structures in geometric modeling. Among many efficient algorithms for computing 2D Voronoi diagrams, Fortune’s sweepline algorithm (Fortune, 1986 [5]) is popular due to its elegance and simplicity. Dehne and Klein (1987) [8] extended sweepline to sweepcircle and suggested computing a type of transformed Voronoi diagram, which is parallel in nature. However, there is no practical implementation of the sweepcircle algorithm due to the difficulty in representing the transformed edges. This paper presents a new algorithm, called \textit{untransformed sweepcircle}, for constructing Voronoi diagram in $\mathbb{R}^2$. Starting with a degenerate circle (of zero radius) centered at an arbitrary location, as the name suggests, our algorithm sweeps the circle by increasing its radius across the plane. At any time during the sweeping process, each site inside the sweep circle defines an ellipse composing of points equidistant from that point and from the sweep circle. The union of all ellipses forms the beach curve—a star shape inside the sweep circle which divides the portion of the plane within which the Voronoi diagram can be completely determined, regardless of what other points might be outside of the sweep circle. As the sweep circle progresses, the intersection of expanding ellipses defines the Voronoi edges. We show that the sweep line algorithm is the degenerate form of the proposed sweep circle algorithm when the circle center is at infinity, and our algorithm has the same time and space complexity as the sweep line algorithm. Our untransformed sweepcircle algorithm is flexible in allowing multiple circles at arbitrary locations to sweep the domain simultaneously. The parallelized implementation is pretty easy without complicated numerical computation; the most complicated case is nothing but an arc-cosine operation. Furthermore, our algorithm supports the additively weighted Voronoi diagrams of which the Voronoi edges are hyperbolic and straight line segments. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

A Voronoi diagram is a special kind of decomposition of a space, determined by distances to a given set of objects in the space. As a fundamental geometric data structure, Voronoi diagrams have been widely applied in various computer graphics and visualization applications, including collision detection [1], meshing [2], vector field visualization [3], artistic effect, image segmentation [4], just to name a few.

Among many efficient algorithms for the construction of a Voronoi diagram, Fortune’s sweep line algorithm [5] is popular due to its elegance and simplicity. Given a set of 2D points (called sites), the sweep line algorithm maintains a vertical sweep line moving from left to right across the plane as the algorithm progresses. At any time during the algorithm, the input sites on the left side of the sweep line have already been incorporated into the Voronoi diagram, while the sites right of the sweep line have not been considered yet. Each site to the left of the sweep line generates a parabola of points equidistant from that point and from the sweep line. The wavefronts of all parabolas form a curved beach line, which follows the sweep line moving from left to right. As the sweep line progresses, the vertices of the beach line, at which two parabolas cross, trace out the edges of the Voronoi diagram. The sweep line algorithm is proven to be optimal with $O(n \log n)$ time complexity and $O(n)$ space complexity.

In spite of its simplicity and popularity, there is no effective technique to parallelize the sweep line algorithm. Thurston [6] pointed out that if the input vertex set is very large, then...
the sweepcircle technique allows one to compute the Voronoi diagram locally [7]. Dehne and Klein [8] proposed a sweepcircle technique to compute the transformed Voronoi diagram, where the transformation is made in polar coordinates:

\[ x \triangleq (p, \theta) \rightarrow x' \triangleq (p + |x - x_i|, \theta), \]  

(1)

where \( x_i \) is the nearest site to \( x \). The purpose of this transformation is to guarantee that the site \( s \) is touched before the Voronoi cell of \( s \) is swept by a sweeping circle. They proved that the transformed diagram has the same combinatorial structure as the original Voronoi diagram [8]. This transformed sweepcircle algorithm is parallel in nature and can be also applied to compute the Voronoi diagram on cones. However, it is difficult to represent the transformed Voronoi edges from the implementation perspective, since they are not straight line segments or conic curves anymore.

To the best of our knowledge, there is no practical implementation of the sweepcircle algorithm.

This paper aims to tackle this challenge by proposing a new algorithm, called untransformed sweepcircle, for computing a 2D Voronoi diagram. Starting with a degenerate circle (of zero radius) centered at an arbitrary point (not necessarily a site), as the name suggests, our algorithm sweeps the circle by increasing its radius across the plane. At any time during the sweeping process, each site inside the sweep circle defines an ellipse composing of points equidistant from that point and from the sweep circle. The union of all ellipses forms the beach curve, a star shape inside the sweep circle which divides the portion of the plane within which the Voronoi diagram can be completely determined, regardless of what other points might be outside of the sweep circle. As the sweep circle progresses, the intersection of expanding ellipses traces out the Voronoi edges.

We show that the sweep line algorithm is the degenerate form of the proposed sweep circle when the circle center is at infinity, and our algorithm also has the same time and space complexity as its line counterpart. However, our algorithm fundamentally distinguishes itself to the sweep line algorithm in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously. As a result, this leads to a very natural parallel implementation, while the sweep line algorithm does not have such a feature. Compared with the traditional sweepcircle technique [8], our algorithm is pretty easy to implement, where the most complicated numerical operation is nothing but an arc-cosine calculation. Furthermore, our algorithm supports the computation of additively weighted Voronoi diagrams where the Voronoi edges are hyperbolic and straight line segments.

The specific contributions of our paper include:

- We propose the untransformed sweepcircle algorithm for computing 2D Voronoi diagram. We prove the correctness of our algorithm and show its optimal space and time complexity.
- We show that the sweep circle algorithm is more flexible and general than sweep line algorithm due to its parallel nature. We demonstrate the effectiveness of our algorithm on a GPU implementation.

2. Related work

Voronoi diagrams are a fundamental geometric structure, which can be traced back to René Descartes in 1644. They are named after the Russian mathematician Georgy Fedoseevich Voronoi who defined and studied the general \( n \)-dimensional case in 1908. Since then, Voronoi diagrams have been extensively studied and widely applied to many science and engineering fields, including computer graphics, image processing, robot navigation, computational chemistry, materials science, climatology, etc. [9, 10], due to their nice geometric properties.

As the dual of Voronoi diagrams, Delaunay triangulations also possess a number of useful properties. For example, among all triangulations of a planar point set the Delaunay triangulation maximizes the minimum angle, which is valuable for quality mesh generation [11, 12]. Also, in all dimensions, the Euclidean minimum spanning tree is a subgraph of the Delaunay triangulation [13]. There is a large body of literatures on the Delaunay triangulation problem. For example, Devillers [14] proposed a hierarchical data structure to compute the Delaunay triangulation on a 2D plane. Both Voronoi diagram and Delaunay triangulation can be computed in the optimal \( O(n \log n) \) time for a set of \( n \) sites in 2D [7]. Among these algorithms, some are in a divide-and-conquer scheme [15–17], while others are incremental [18, 19]. The same optimal worst-case bound is also obtained by Fortune's elegant sweep line algorithm [5] and the traditional sweepcircle algorithm [8].

For a Voronoi diagram of moving points, Albers et al. [20] showed the upper bound of the number of topological events and proposed a numerically stable algorithm for the update of the topological structure of Voronoi diagram, using only \( O(\log n) \) time per event. Zhou et al. [21] presented the bi-cell filtering technique to utilize the connectivity coherence of the Delaunay triangulation of moving points, which leads to an efficient algorithm to compute the dynamic Delaunay triangulation.

There are several techniques, such as randomization [22] and divide-and-conquer [23], for parallel computation of Voronoi diagrams. However, these parallel algorithms are rather complicated and highly non-trivial to implement. To improve the performance of computing the Voronoi diagram, many researchers focus on computing approximate Voronoi diagrams using the help of the modern graphics hardware. Hoff III et al. [24] developed a highly efficient algorithm for computing generalized Voronoi diagrams in 2D and 3D using rasterization hardware. The algorithm first divides the space into regular cells; then for each cell it computes the closest primitive and the distance to that primitive using polygon scan-conversion and Z-buffer depth comparison. Sud et al. [1] computed the 2nd order discrete Voronoi diagram using a GPU and applied it to perform N-body culling. Very recently, Rong et al. [25] presented a GPU-assisted Voronoi diagram algorithm for accelerating the computation of Centroidal Voronoi Tessellation (CVT). These graphics hardware accelerated algorithms are efficient, but produce only a discrete approximation of the Voronoi diagram. Our algorithm, in contrast, computes the exact 2D Voronoi diagram in parallel. Besides, Cao et al. [26] proposed a parallel banding algorithm on the GPU to compute the distance map for a binary image.

3. Preliminaries

This section briefly reviews the theoretical background of the Voronoi diagram and Fortune’s sweep line algorithm.

Let \( S = \{s_1, s_2, \ldots, s_n\} \) be a set of points (called sites) in \( \mathbb{R}^d \). The Voronoi cell \( \text{Vor}(s_i) \) of a site \( s_i \) is

\[ \text{Vor}(s_i) = \{x \in \mathbb{R}^d | ||x - s_i|| \leq ||x - s_j|| \forall j \neq i\}, \]

where \( ||p - q|| \) denotes the Euclidean distance between points \( p \) and \( q \). The Voronoi cell \( \text{Vor}(s_i) \) is the set of all the points \( x \) that are at least as close to \( s_i \) as to any other site \( s_j \) in \( S \). All the Voronoi cells form a partition of the given space. It is well known that each Voronoi cell is convex and contains exactly one site. A Voronoi cell is unbounded if and only if the corresponding site lies on the convex hull of \( S \). The number of Voronoi edges and vertices is \( O(n) \). The Delaunay triangulation is the dual structure of the Voronoi diagram. By dual, we mean two Voronoi sites are connected by a line segment if they share a Voronoi edge. Fig. 1(a) shows...
an example of 2D Voronoi diagram and its dual Delaunay triangulation.

Fortune’s sweep line algorithm [5] maintains a sweep line and a beach line as the algorithm progresses. Without loss of generality, assume the sweep line is a vertical straight line moving from left to right across the plane. Each site left of the sweep line defines a parabola of points equidistant from that point and from the sweep line. The beach line is the boundary of the union of these parabolas, which splits the plane into two regions: the left region contains a partial Voronoi diagram with a correctly computed topological structure, and the right region corresponds to the to-be-determined part of the Voronoi diagram. For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.

Suppose $x_i$ is an active site, then the parabolic segment determined by $x_i$ is totally inside $x_i$’s Voronoi cell. In the sweep line algorithm, a balanced binary search tree is used to maintain the combinatorial structure of the beach line, and a priority queue listing potential pending events that possibly change the beach line structure. It can be shown that there are $O(n)$ events to process and $O(\log n)$ time required to process each event, and hence the total time is $O(n \log n)$.

Although the sweep line algorithm is conceptually elegant and widely adopted, it is non-trivial to parallelize and implement it on modern GPUs. As shown in Fig. 2, to compute the Voronoi diagram for the square domain in parallel, we partition it to a set of square-shaped blocks with side length $r$, and then run the sweep line algorithm for each block independently. Without loss of generality, we assume that at least one site is contained in the green block. It can be shown that only the sites in the red round-cornered rectangle, $(4\sqrt{2} + 2\pi + 1)r^2$ in size, affect the Voronoi diagram of the green block. Therefore, we need to place a sweep line at the left boundary of the red round-cornered rectangle (see Fig. 2(a)) and consider the sites located inside this area in the subsequent sweeping process. In this example, as sites $A$ and $B$ are inside the red region, the sweep line algorithm results in a Voronoi edge bisecting $A$ and $B$ (see the yellow line segment in Fig. 2(b)). However, the correct Voronoi diagram (dotted blue lines in Fig. 2(a)) consists of only part of the bisecting line of $A$ and $B$ due to the existence of site $C$, which is outside of the red region. In fact, it can be shown that only the Voronoi diagram inside the green region is correct and contributes to the final result. Therefore, we have to trim the computed Voronoi diagram (of the entire red region) with the centered green region to get the final result. In other words, each sweep line algorithm computes the Voronoi diagram for a region of size $(4\sqrt{2} + 2\pi + 1)r^2$, but only the result inside the centered region of size $r^2$ is used. Roughly speaking, $1 - \frac{1}{4\sqrt{2} + 2\pi + 1} = 92.3\%$ computation is totally wasted. Due to these drawbacks, it is difficult to parallelize the sweep line algorithm in an efficient and effective manner.

The proposed untransformed sweepcircle algorithm overcomes the drawbacks of the sweep line algorithm, and can be naturally extended to the parallel setting. In the following, we first introduce the key concepts and properties of the untransformed sweepcircle algorithm in Section 4 and then present the parallel implementation on modern GPUs in Section 5. We provide the detailed experimental results in Section 6 and discuss the fundamental differences between the untransformed sweepcircle algorithm and Dehne & Klein’s sweepcircle method [8] in Section 7. Finally in Section 8, we conclude the paper and suggest several future research directions.

4. Untransformed sweepcircle algorithm

4.1. Algorithm description

In contrast to the sweep line algorithm which propagates the sweep/beach line from left to right, our algorithm starts with a degenerate circle at an arbitrary location and then sweeps the circle by increasing its radius to cover the entire domain. Our untransformed sweepcircle algorithm contains four fundamental elements:

- The sweep circle, centered at the user-specified or a random location $c$, sweeps the plane by increasing its radius. The initial radius is usually set to zero.
- The beach curve is a set of consecutive elliptic segments, each of which bisects a site and the sweep circle. The beach curve splits the plane into two regions: the inside region contains a partial Voronoi diagram with a correctly computed
Fig. 3. Illustration of sweep circles for two sites. The beach curve consists of elliptic segments, each of which corresponds to a site. The elliptic segment is the bisector of the corresponding site and the sweep circle. The vertices of the beach curve trace out the bisector between two sites. The sweep circle, the beach curve and the bisector are drawn in green, blue and red, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The topological structure, and the outside region corresponds to an undetermined part of the Voronoi diagram that will be computed later.

- A touching event occurs when the sweep circle touches a new site. As a result, a new elliptic segment is inserted into the current beach curve.
- A vanishing event occurs when an elliptic segment disappears. Consequently, a new Voronoi vertex is generated.

Fig. 3 illustrates the basic idea of sweep circle for a two-site example. Initially, we place a degenerate circle (i.e., a point) at the user-specified or at random location $c$. Assume the circle center does not coincide with any sites. The circle grows until it touches the closest site, say $s_1$, which triggers the first touching event (see (b)). As a result, we construct a degenerate ellipse with focus points $s_1$ and $c$, satisfying $||x - s_1|| + ||x - c|| = R$, where $R = ||c - s_1||$ is the radius of the current sweep circle. Clearly, the initial beach curve is a line segment (i.e., degenerate ellipse). Throughout the sweeping process, the ellipse always satisfies the constraint $||x - s_1|| + ||x - c|| = R$ for the increasing radius $R$. It can be shown that the ellipse is the bisector of $s_1$ and the sweep circle $⊙(c, R)$.

Furthermore, the beach curve is always inside the sweep circle (see (c)). The next event (again, a touching event) occurs when the sweep circle reaches the second site $s_2$, which creates an ellipse $||x - s_2|| + ||x - c|| = R$ (now $R = ||c - s_2||$). This newly generated ellipse is inserted into the current beach curve. Thus, the updated beach curve contains two elliptic segments, each of which is the bisector of the sweep circle and the corresponding site (see (d)). As the sweep circle progresses, the vertices of the beach curve, at which two ellipses cross, trace out the bisector (in red) of $s_1$ and $s_2$ (see (e)). Note that the continuous animation is used to illustrate the basic concept of sweep circle. In fact, our algorithm sweeps the circle and processes the events in a discrete manner.

Fig. 4 shows the circle sweeping on a domain with five sites. The key idea of our untransformed sweep circle algorithm is to maintain the (circular) ordered list of elliptic segments and trace out the vertices of the beach curve. The two neighboring (in circular order) elliptic segments correspond to an active Voronoi edge and the two neighboring active edges (in circular order) are joined by an elliptic segment. When an elliptic segment vanishes (i.e., a
vanishing event occurs), the two neighboring active edges meet at a point, which defines a Voronoi vertex (see the insets in Fig. 4(d), (f), (g)). When the sweep circle touches a new site (i.e., a touching event occurs), a new Voronoi edge is generated. The discrete events are maintained in a priority queue sorted by the distances of the corresponding sites to the center c (from near to far). Throughout the algorithm, the Voronoi diagram inside the beach curve has been determined. The algorithm stops when the priority queue is empty and at this moment the complete Voronoi diagram is known. The essential data structures in the untransformed sweep circle algorithm include the Voronoi diagram, the beach curve and the event manager, shown as follows:

//2D point, vector or interval
typedef pair<double,double> double2;

//Lists of sites and Voronoi vertices
vector<double2> SiteList;
vector<double2> VoronoiVertexList;

struct Voronoiedge {
  //two endpoints of the edge
  int vert1, vert2;
  //two sites contributing to the bisector
  int site1, site2;
};
vector<Voronoiedge> VoronoiedgeList;

//An active edge, shared by two consecutive elliptic
//segments, contains one fixed endpoint and an
//extending direction
struct ActiveEdge {
  int indexFixedEndpoint;
  double2 extendingDirection;
  EllipticSegment* leftSegment;
  EllipticSegment* rightSegment;
};
struct EllipticSegment {
  //One focus point is the center of the sweep circle
  double2 circleCenter;
  //The other focus point is a site
  int indexActiveSite;
  //Sandwiched by two neighboring active edges.
  ActiveEdge* leftActiveEdge;
  ActiveEdge* rightActiveEdge;
};
struct beachCurve {
  //The beach curve contains the elliptic segments in
  //circular order
  BalancedBinaryTree<EllipticSegment> beachCurve;

  //Discrete events are handled from near to far
  enum EventType { TOUCH, VANISH };
  struct Event {
    EventType type;
    double whenOccur;
    int indexSiteToTouch;
    EllipticSegment* segmentToDisappear;
  }
  priority_queue<EventType> EventQueue;

  Since two neighboring elliptic segments trace out an active
  Voronoi edge and two neighboring active Voronoi edges sandwich
  an elliptic segment, we use pointers to maintain the relationship
  between ActiveEdge and EllipticSegment. We also use a balanced
  binary tree to dynamically maintain the beach curve that contains
  the elliptic segments in a circular order. The complete structure of
  the Voronoi diagram, such as the incident edges of a Voronoi vertex
  and the bounding edges of a Voronoi cell, can be induced from
  the above data structures. The pseudo code of our untransformed
  sweep circle algorithm is shown in Algorithm 1.

  Algorithm 1 Untransformed sweep circle algorithm for computing
  2D Voronoi diagram

  Input: A set of 2D sites S = {s1, . . . , sn} and the center of the
  sweep circle c;
  Output: The Voronoi diagram of S
  //T: a binary tree to maintain the beach curve;
  //V: the Voronoivertex and edge structure;
  //Q: the priority queue of discrete events
  Push n touching events into Q sorted by ||c−si||, 1 ≤ i ≤ n.
  while Q.isEmpty() == false do
    Pop up the top event q from Q;
    if q is a touching event then
      Insert a new ellipse E into T;
      //Suppose E splits the existing elliptic segment
      //E′ into E1 and E2, and let p be the intersection
      //point. Build two radial edges rooted at p, one shared by E1 and E2,
      //the other shared by E and E′;
      Compute the time when E, E1, and E2 disappear; Push
      the vanishing event into Q;
    else if q is not out of data then
      //q is a vanishing event.
      //Let E denote the vanishing
      //elliptic segment, E1 and E2 the two neighboring
      //elliptic segments. If one of them is out of data,
      //so is the event. See Figure 5.
      Create a Voronoi vertex v;
      Terminate the two active edges, joined by E, at v;
      Update the Voronoivertex structure v′;
      Create an active Voronoiedge e;
      Push a vanishing event according to e and its neighboring
      active edges into the queue Q;
    end if
  end while

4.2. Correctness

This subsection proves the correctness of our untransformed
sweep circle algorithm. Consider a set of 2D sites S = {s1, s2, . . . ,
sn}. Let us denote ∪(c, R) by the sweep circle centered at c with
radius R.

Lemma 1. Each elliptic segment of the beach curve bisects the sweep
circle ∪(c, R) and the corresponding site si. When the center c
coincides with a site si, the elliptic segment degenerates into a circular arc.

Proof. Let ei be the ellipse corresponding to site si, i.e., ei = {x| ||x−
c|| + ||x − si|| = R}. For any point x ∈ ei, the shortest distance
between x and the sweep circle is d(x, ∪(c, R)) = R − ||x − c|| =
||x − si||. Thus, the ellipse is the bisector of the sweep circle and si.
When c = si, we obtain ||x−c|| = 1, which is a circle.

Lemma 2. For any two sites satisfying ||c−si|| < R and ||c−sj|| < R, the
eEllipses
ei = {x| ||x−c|| + ||x − si|| = R}
and
ej = {x| ||x−c|| + ||x − sj|| = R}
intersect at exactly two points, which are on the bisector of si and sj.

Proof. The number of intersection points of two ellipses is from 0
to 4. First, the number of intersection points of ei and ej is at least 2
The two ellipses share the focus point c, which is completely inside the ellipses $e_i$ and $e_j$.

- Let $p_i \in e_i$ (resp. $p_j \in e_j$) be the closest point on $e_i$ (resp. $e_j$) to the sweep circle $\odot(c, R)$. Then $p_i$ is outside of $e_i$ and $p_j$ is outside of $e_j$ (see the above figure).

On the other hand, the intersection points between $e_i$ and $e_j$ satisfy $\|x - s_i\| = \|x - s_j\|$ and determine a straight line $L$. Any intersection point in $e_i \cap e_j$ must be a subset of $L \cap e_i$ (or $L \cap e_j$). This implies that the number of intersections cannot exceed 2. Therefore, the two ellipses $e_i$ and $e_j$ meet at exactly two points, which determine the bisector of $s_i$ and $s_j$. □

Lemma 3. The beach curve is a closed star shape with center $c$, and is always inside the sweep circle $\odot(c, R)$.

Proof. Let $e_i$ be the ellipse corresponding to the site $s_i$, i.e., $\|x - c\| + \|x - x_i\| = R$. Clearly, the ellipse $e_i$ is inside the sweep circle $\odot(c, R)$, since $\|x - c\| \leq R$. Let $\Omega = \bigcup_{e_i \odot(c, R)} \{x : \|x - c\| + \|x - s_i\| \leq R\}$ denote the union of the ellipses which are inside the sweep circle. The beach curve is the boundary of $\Omega$. Due to the fact $\partial \cap \partial = 0$, the beach curve is closed.

Then we show $\partial \Omega$ is star shaped. Observe that each ellipse $\|x - s_i\| + \|x - c\| = R$ is a star shape with respect to the center $c$. Assume the beach curve is not a star shape w.r.t. $c$, there must exist two points $p_i \in e_i$, $p_j \in e_j$ on the beach curve such that $p_i$, $p_j$, $c$ are collinear and $p_i$ is between $p_j$ and $c$, respectively. Therefore, $p_i$ is totally inside $e_i$, since $e_i$ itself forms a star shape w.r.t. $c$. This contradicts the assumption that $p_i \in e_i$ is on the beach curve, which is the boundary of $\bigcup e_i$. □

Lemma 4. The shortest distance between the center $c$ and the beach curve is at least $\frac{R - d}{2}$, where $d$ is the distance between $c$ and the nearest site.

Proof. Suppose the $s^*$ is the nearest site to the center $c$. Then the ellipse $\|x - s^*\| + \|x - c\| = R$ must be completely inside the beach curve. Observe that the shortest distance between the ellipse and the center $c$ is $\frac{R - d}{2}$. So the shortest distance between the beach curve and $c$ is at least $\frac{R - d}{2}$. □

Lemma 5. The elliptic segment corresponding to site $s_i$ is inside the Voronoi cell of $s_i$.

Proof. We distinguish the input site set $S$ into the outside group $S_0 = \{s_i | s_i \notin \odot(c, R)\}$ and the inside group $S_I = \{s_i | s_i \in \odot(c, R)\}$.

We first examine the inside group $S_I$. Let $s_i \in S_I$ be a site inside the sweep circle and $e_i$ be the corresponding ellipse $\|x - s_i\| + \|x - c\| = R$. For any point $x \in e_i$, we have $d(x, s_i) < d(x, s_j)$, $\forall s_j \in S_0$, since the ellipse $e_i$ is the bisector of the sweep circle and the site $s_i$ (by Lemma 1).

Then, we prove that $\forall x \in e_i, d(x, s_i) \leq d(x, s_j)$ for any $s_j \in S_I$ by contradiction. Assume there is a point $x \in e_i$ and a site $s_j \in S_I$ such that $d(x, s_j) > d(x, s_i)$. Let $e_j$ be the ellipse of site $s_j$, i.e., $e_j = \{x : \|x - s_j\| + \|x - c\| = R\}$. Then the point $\hat{x}$ must be completely inside the ellipse $e_j$ since $\|\hat{x} - s_i\| + \|\hat{x} - c\| < R$. Observing that the beach curve is exactly the boundary of the union of ellipses inside the sweep circle, $\hat{x}$ is not on the beach curve, which leads to a contradiction!

Putting it together, the elliptic segment $e_i$ is closer to $s_i$ than any other sites in $S$. Thus, $e_i$ is inside the Voronoi cell of $s_i$. □

Lemma 6. The intersection of consecutive elliptic segments of the beach curve lies on the Voronoi edges.

Proof. Suppose $x$ is the intersection of two consecutive elliptic segments $e_1$ and $e_2$, corresponding to sites $s_1$ and $s_2$ respectively. By Lemma 5, $e_1$ (resp. $e_2$) is inside the Voronoi cells of $s_1$ (resp. $s_2$). Thus, the intersection of $e_1$ and $e_2$ is of equal distance to $s_1$ and $s_2$ (and closer to $s_1$ and $s_2$ than any other sites), i.e., it lies on the Voronoi edge. □

Lemma 7. The Voronoi diagram inside the beach curve has been completely determined.

Proof. For every point $p$ inside the beach curve, the site which is closest to $p$ can be determined by the elliptic segment passing through $p$. Thus, the Voronoi diagram inside the beach curve can be completely determined regardless of the outside sites of the sweep circle. □

Lemma 8. The current beach curve contains an elliptic segment corresponding to the site $s_i$ if and only if $s_i \in \odot(c, R)$ and its Voronoi cell is still open.

Proof. $\Leftarrow$: Assume the current beach curve still contains one of its elliptic segments, say $e$. This implies that part of this Voronoi cell lies outside the current beach curve, which cannot be determined by our algorithm (see Lemma 7). Thus, the Voronoi cell is not yet complete, which contradicts the assumption that the Voronoi cell of $s_i$ is closed.

$\Rightarrow$: If the Voronoi cell of $s_i$ is still open, it must intersect the current beach curve at two or more points. Without loss of generality, consider two intersection points $p_1$ and $p_2$ joined by one or more elliptic segments that are inside the Voronoi cell of $s_i$. So every point $x$ on this curved segment is closer to $s_i$ than any other sites, which implies that $x$ satisfies the equation $\|x - s_i\| + \|x - c\| = R$. Thus, the current beach curve must contain the elliptic segment of site $s_i$. □
popped from the queue. Therefore, the event when the sweep circle touches the site \( s \), which is simply ignored by the sweep circle algorithm.

Fig. 5. Out of date events. The red elliptic segment is sandwiched by two Voronoi edges, say \( l_1 \) and \( l_2 \), and will vanish when the sweep circle touches the point \( w_1 \). As a result, a vanishing event \( q \) has been pushed into the priority queue \( Q \). However, when the sweep circle touches the site \( s_q \), the edge \( l_1 \) is intercepted by a newly-created Voronoi edge (see the inset). Therefore, the event \( q \) is out of date when it is popped from the queue \( Q \), which is simply ignored by the sweep circle algorithm.

![Fig. 5](image)

Theorem 1. The Fortune's sweep line algorithm [5] is a degenerate form of our sweepcircle algorithm if the center \( c \) is at an infinite distance to the sites.

Proof. The site \( s_i \) contributes to the beach curve after the sweep circle hits \( s_i \) and before the corresponding Voronoi cell becomes a closed polygon. During this period, the elliptic segment of \( s_i \) is inside the Voronoi cell (by Lemma 5) and it is closer to the focus point \( s_i \) than the other focus point \( c \). The center of the sweep circle since \( c \) is assumed to be at infinite distance to the sites. The distance between \( s_i \) and its elliptic segment is very limited, and cannot exceed the size of \( s_i \)'s Voronoi cell.

Under this condition, if the center \( c \) is at infinite distance to \( s_i \), the eccentricity of the ellipse approaches 1. As a result, the sweep circle becomes a straight line and the elliptic segment becomes a parabola segment. At the same time, the beach curve still serves as the bisector of the sweep line and the active sites, exactly the same situation with that achieved in Fortune's sweep line algorithm. Therefore, we can conclude that Fortune's sweep line algorithm is the degenerate form of our untransformed sweepcircle algorithm. □

4.3. Complexity

Theorem 2. The untransformed sweepcircle algorithm computes the Voronoi diagram in \( O(n \log n) \) time and requires \( O(n) \) space.

Proof. First, sorting the sites according to the distances to the sweep center takes \( O(n \log n) \) time.

Second, we show the number of elliptic segments, as well as that of the events, is \( O(n) \). There are two kinds of events, namely, the touching event and the vanishing event. The number of touching events is equal to the number of sites \( n \). When the sweep circle touches a new site, the algorithm splits an existing elliptic segment of the current beach curve into two and then add a new elliptic segment corresponding to the new site. Therefore, at most \( 2n \) elliptic segments are generated, and at most \( 3n \) vanishing events need to be considered throughout the algorithm. Note that some vanishing events may be out of date and are discarded when the sweep circle progresses (see Fig. 5).

Third, we show handling each event takes \( O(\log n) \) time. For the touching event, a new elliptic segment is inserted into the beach curve, maintained by a balanced binary tree \( T \). The insertion takes \( O(\log n) \) time. For the vanishing event, the priority queue \( Q \) is updated, which also takes \( O(\log n) \) time.

Fourth, the Voronoi diagram, the balanced binary tree and the priority queue require \( O(n) \) space to store the vertex-edge structure, the \( O(n) \) elliptic segments and \( O(n) \) events respectively. Putting it altogether, the sweep circle algorithm takes \( O(n \log n) \) time and requires \( O(n) \) space. □

5. Parallel sweep circles

To compute the Voronoi diagram in a parallel fashion, we need to partition the domain and then solve each sub-domain independently. To use the sweep line algorithm, one needs to place the initial sweep line at one side of the to-be-computed region, and then moves it in a certain direction across such region. As shown in Fig. 2, the sweep line algorithm needs to sweep the region much larger than the desired region. Furthermore, as only the centered
Both the sequential sweep circle and sweep line algorithms have a very similar number of discrete events. With the parallel sweep circle algorithm, the average number of events for each GPU thread is significantly reduced. The horizontal axis and the vertical axis show the number of sites and the average number of discrete events respectively. The number in the bracket represents the number of sweep circles.

Compared to the classical sweep line algorithm, the proposed untransformed sweepcircle algorithm has two unique advantages, which makes it preferable for parallel implementation. First, the sweep circle can be placed anywhere in the domain. Second, the beach curve of our sweepcircle algorithm is closed and the Voronoi diagram inside the beach curve can be completely determined regardless of the sites outside the sweep circle. Thus, each sweep circle thread can work independently. Even though the sweep circles may have overlap, we do not need any postprocessing to trim the Voronoi diagram.

Given a 2D domain \( \Omega \) containing sites \( S \), we can partition the domain \( \Omega \) into disjoint sub-regions, \( \Omega = \bigcup_{i=1}^{m} \Omega_i \), \( \Omega_i \cap \Omega_j = \emptyset \), \( \forall i \neq j \). Then each sub-region \( \Omega_i \) is assigned a sweep circle thread, which places a sweep circle at arbitrary location inside \( \Omega_i \). In practice, one can set the center \( c_i \) the barycenter of \( \Omega_i \). Each thread stops when its event queue is empty. If one sub-region contains no sites at all, we can merge that region with neighboring region such that each sub-region contains at least one site. The pseudo code of the parallel sweep circle algorithm is shown in Algorithm 2.

Algorithm 2 Parallel sweep circle algorithm

**Input:** A 2D region \( \Omega \) containing \( n \) sites \( S = \{s_1, \ldots, s_n\} \) and a partition of \( \Omega \) into \( m \) disjoint sub-regions \( \Omega_i \), \( \Omega_i \cap \Omega_j = \emptyset \), \( \forall i \neq j \).

**Output:** The Voronoi diagram of \( S \)

parallel for each sub-region \( \Omega_i \) do
  Compute the barycenter \( c_i \) of \( \Omega_i \)
  Run the sweep circle algorithm centered at \( c_i \)
parallel end for

In fact, since each sweep circle thread cares for only the Voronoi structure inside \( \Omega_i \), we can modify the sweep circle algorithm such that it can terminate early, i.e., without processing all the vanishing events in the event queue. By Lemma 7, the Voronoi diagram inside the beach curve is complete. Thus, we can test whether...
Fig. 10. Computing the additively weighted Voronoi diagram, where the conventional Euclidean distance is modified by the weights assigned to the generator sites. The weights are illustrated by the size of the site. The resulting Voronoi diagram contains both hyperbolic segments and line segments. The animation from (a) to (e) illustrates the sweeping process.

the beach curve contains the entire sub-region $\Omega_i$. If so, we can stop the sweep circle algorithm for $\Omega_i$. In practical application, the sub-region $\Omega_i$ is usually modeled by a polygon. So this test can be implemented easily. Let $R_s$ denote the radius of the sweep circle when the algorithm stops. The following theorem provides an upper bound of the stopping radius $R_s$:

**Theorem 3.** Given the sub-region $\Omega_i$, let $\odot(c_i, r_i) \supset \Omega_i$ be the smallest covering disc. The site $s \in \Omega_i$ is the nearest one to the center $c_i$. Define $d = \|cs\|$. The sweep circle algorithm for the sub-region $\Omega_i$ can stop when the radius $R_s \geq 2r_i + d$ or the event queue is empty, or whichever occurs first.

**Proof.** By Lemma 3, the ellipse $\|x-s\| + \|x-c_i\| = R$ must be inside the beach curve, since $s$ is the nearest one to the center $c_i$. By Lemma 4, the beach curve can contain a disc $\odot(c_i, \frac{R-d}{2})$ completely. In order to ensure that the Voronoi diagram inside $\Omega_i$ has been fully determined, we require the terminating radius $R_s$ satisfying $\frac{R_s-d}{2} \geq r_i$, which completes the proof. □

We would like to point out the above upper bound of the stopping radius $R_s = 2r_i + d$ is very conservative. Based on our experiments, we observe that the beach curve usually follows the sweep circle very closely. As a result, the stopping radius is usually slight larger than the size of the sub-region $r_i$. In practice, we can partition the domain into equally sized regular hexagon, which is very effective to reduce the sweep circle. The theoretical lower bound of $R_s$ is $R_s = r_i$, which occurs for the uniformly distributed sites with such hexagonal partition (Fig. 6).

Furthermore, as the Voronoi diagram inside the beach curve is guaranteed to be correct, the overlap among the sweep circles does not affect the final result. This feature significantly distinguishes our algorithm from the sweep line algorithm, which has to trim the computed Voronoi diagrams.

6. Experimental results

We implemented our untransformed sweatcircle algorithm in C++ and tested it on a PC with an Intel Xeon 2.50 GHz CPU and 8 GB memory. The graphics card is an NVIDIA GTX 580 with 512 cores and 1.5 GB memory. Our program is compiled using CUDA 4.0 RC2.

We have shown that the sweep circle has the same time and space complexity as the sweep line algorithm. To measure and compare the practical performance, we count the number of discrete events for the sweep circle algorithm and the sweep line algorithm. Although based on different sweeping strategies, both approaches have a very similar number of events, as shown in Fig. 8. Due to the trigonometric functions used for computing the joint point between consecutive elliptic segments, its computational cost for processing each event is slightly higher than that of the sweep line algorithm. As a result, the single-core sweep circle algorithm is slower than the sweep line algorithm. However, as mentioned before, the proposed sweep circle algorithm is superior than the sweep line algorithm due to its parallel nature. With the parallel untransformed sweatcircle algorithm, the average number of events can be significantly reduced by increasing the number of sweep circles.

Fig. 9 shows an example of 250 sites within an irregular domain, which is partitioned into $m^2$ ($m = 2, 3, 4, 5$) sub-regions. Each sub-region $\Omega_i$ is assigned a GPU based sweep circle thread, which outputs the Voronoi diagram inside the corresponding beach
curve. Although the neighboring beach curves have overlap when the sweep circles progress, all the threads can run independently without any data conflicts or synchronization.

Our algorithm can be applied to 3D surface by using parameterization [27]. Fig. 7 shows two genus-0 3D models which are conformally parameterized to disc and square. We generate 200 and 600 random sites on the 3D face and sheep model. Then we run the parallel sweep circle algorithm on the 2D parametric domain using the Euclidean distance and the parameterization induces the Voronoi diagram on the 3D surface. See Fig. 11. An interesting work is to use the geodesic distances instead of the Euclidean distance, which would result in an intrinsic geodesic Voronoi diagram on surface. As this is beyond the scope of this paper, we will address it in the future work.

The proposed sweep circle framework can be easily extended to the additively weighted Voronoi diagram, for which the conventional Euclidean distance is modified by the weights assigned to each site. We represent each site \( s_i \) as a disk of radius \( r_i \) centered at \( s_i \), where \( r_i \) is the weight of site \( s_i \). The active edge structure is also modified to contain both the line segments and hyperbolic segments, as shown in Fig. 10.

7. Discussion

Generally speaking, a divide-and-conquer Voronoi algorithm needs to overcome two difficulties: (1) computing the Voronoi diagram in a sub-domain; and (2) merging the individual results into a complete diagram. Taking Fig. 2 for example, a parallel sweepline algorithm needs to distinguish the diagram in the green square from other part for merging purpose, because the computed result outside the green square may be incorrect. Visually, the computed diagrams can be glued in a seamless manner. However, the implementation of such merging in the parallel sweepline algorithm is complicated. For example, one has to check whether or not each bisector intersects the centered green region. It is worth noting that our algorithm does not require any special treatment in the merging step. This is because our algorithm maintains a global table for pairing the adjacent sites in the final Voronoi diagram. Each sweepcircle deals with some sites inside it and then updates the table without data conflicts, since the computed Voronoi diagram behind the beach curve is globally correct. After all sweepcircle processes are done, the pairing table is ready, which induces the Voronoi diagram immediately. This feature distinguishes our parallel sweepcircle algorithm with the parallel sweepline algorithm.

We observed two degenerate cases in our experiments: (1) inserting a new elliptic arc segment into a very small and to-be-vanished arc; and (2) two consecutive elliptic arcs vanish at the same time. These cases happen when more than 3 Voronoi cells share a common vertex. In our implementation, we used a tolerance \( 1e-7 \) to identify such degeneracy and then treated the vanishing point as a high-order Voronoi vertex.
References


8. Conclusion

This paper presents the untransformed sweepcircle algorithm for computing 2D Voronoi diagrams. Our algorithm has the optimal O(n log n) time complexity and O(n) space complexity. The classical sweep line algorithm is the degenerate form of our algorithm when the circle center is at infinity. Our algorithm is flexible in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously, which naturally leads to parallel implementation. It is easy to implement, without complicated numerical calculation. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

Acknowledgment

This work was supported by NRF2008IDM-IDM004-006, Fraunhofer IDM@NTU, NRF BeingThere Project, and NSF No. 60933007. We would like to thank Shuang-Min Chen for her help on the video and figures.

Thurston [6] observed the sweepcircle technique allows to compute the Voronoi diagram locally, which shows the Voronoi diagram can be parallelized in nature. Dehne and Klein [8] applied the sweepcircle to compute a type of transformed Voronoi diagram. Although their algorithm also used multiple sweep circles, it is fundamentally different to our untransformed sweepcircle algorithm. First, in the traditional sweepcircle algorithm, the transformed edges are very complicated in representation form. By contrast, our algorithm is easy to implement since the most complicated operation is nothing but an arc-cosine calculation. Second, the beach curve consists of elliptic segments in our untransformed sweepcircle algorithm, while the beach curve is rather complicated in Dehne and Klein’s sweepcircle algorithm (they didn’t mention the representation form of the beach curve in [8]). Third, our algorithm is very natural to extend onto 2D weighted Voronoi diagrams, while it is not clear whether it is easy or not for Dehne and Klein’s sweepcircle algorithm to support 2D weighted Voronoi diagrams. Last, Dehne and Klein’s sweepcircle algorithm can be easily extended to the Voronoi diagram on the surface of a cone, while it seems difficult for our algorithm.

The classical sweepline algorithm is the degenerate form of our algorithm when the circle center is at infinity. Our algorithm is flexible in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously, which naturally leads to a parallel implementation. It is easy to implement, without complicated numerical calculation. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

8. Conclusion

This paper presents the untransformed sweepcircle algorithm for computing 2D Voronoi diagrams. Our algorithm has the optimal O(n log n) time complexity and O(n) space complexity. The classical sweep line algorithm is the degenerate form of our algorithm when the circle center is at infinity. Our algorithm is flexible in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously, which naturally leads to a parallel implementation. It is easy to implement, without complicated numerical calculation. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

Acknowledgment

This work was supported by NRF2008IDM-IDM004-006, Fraunhofer IDM@NTU, NRF BeingThere Project, and NSF No. 60933007. We would like to thank Shuang-Min Chen for her help on the video and figures.

References


8. Conclusion

This paper presents the untransformed sweepcircle algorithm for computing 2D Voronoi diagrams. Our algorithm has the optimal O(n log n) time complexity and O(n) space complexity. The classical sweep line algorithm is the degenerate form of our algorithm when the circle center is at infinity. Our algorithm is flexible in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously, which naturally leads to parallel implementation. It is easy to implement, without complicated numerical calculation. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

Acknowledgment

This work was supported by NRF2008IDM-IDM004-006, Fraunhofer IDM@NTU, NRF BeingThere Project, and NSF No. 60933007. We would like to thank Shuang-Min Chen for her help on the video and figures.

References


8. Conclusion

This paper presents the untransformed sweepcircle algorithm for computing 2D Voronoi diagrams. Our algorithm has the optimal O(n log n) time complexity and O(n) space complexity. The classical sweep line algorithm is the degenerate form of our algorithm when the circle center is at infinity. Our algorithm is flexible in that it allows multiple circles at arbitrary locations to sweep the domain simultaneously, which naturally leads to parallel implementation. It is easy to implement, without complicated numerical calculation. We demonstrate the efficacy of our parallel sweep circle algorithm using a GPU.

Acknowledgment

This work was supported by NRF2008IDM-IDM004-006, Fraunhofer IDM@NTU, NRF BeingThere Project, and NSF No. 60933007. We would like to thank Shuang-Min Chen for her help on the video and figures.