Advanced Space Filtering for the Construction of 3D Additively Weighted Voronoi Diagram

Michal Zemek, Martin Maňák, Ivana Kolingerová
Department of Computer Science and Engineering
University of West Bohemia
Pilsen
mzemek@kiv.zcu.cz, manak@kiv.zcu.cz, kolinger@kiv.zcu.cz

***Abstract*—Spatial relationships among 3D spheres can be described by an additively weighted Voronoi diagram and this diagram can be used for advanced spatial analysis. The diagram can be constructed by an edge tracing algorithm. The problem is that tracing an edge is a time consuming operation, where many spheres are tested. Former approaches make it faster by using space filters and searching for spheres intersecting the filter. But they are inefficient when the spheres have very different radii. Our approach presented in this paper is designed to be fast even on this kind of data. It is based on modified space filters and the search for spheres intersecting the filter is performed in a power diagram.**

***Keywords*-computational geometry, additively weighted Voronoi diagram, geometric filter**

I. Introduction

Space partitioning schemes are often used in the analysis of space among 3D spheres. The analysis can include but is not limited to the computation of the volume and surface occupied by their union, inspecting the free volume or finding internal voids. This is especially important in the study of molecules or granular materials. To solve these tasks effectively, it is fundamental to have a good space partitioning scheme. Several kinds of Voronoi diagrams and their dual triangulations have been used in this area [1]. A diagram partitions the space into regions and each region belongs to a generator (in our case - a sphere). Especially aw-Voronoi diagrams (additively weighted) turned out to be useful [2], because they describe spatial relationships among spheres very well. But their regions can have non-linear boundaries, so their computation is not so easy as in the case of diagrams with linear boundaries.

The aw-Voronoi diagram can be constructed by the edge tracing algorithm [3], which we will describe later in Section IV-B. The basic variant of the algorithm is slow (at least quadratic time complexity), therefore, filtering approaches have been devised to decrease the number of spheres tested in the algorithm by considering only the spheres intersecting some filter (e.g., the union of a filled sphere and a half-space). Previous filtering approaches have been devised primarily for molecular data, where spheres have very similar radii and cannot be too close or too far. But in general data, there can be very large spheres as well as very small and the distribution of their positions can be nonuniform. Previous approaches are inefficient on such data.

A. Contribution

In this paper we will present another filtering approach. The main idea is to compute a simpler power diagram for the input spheres in a preprocessing step and then use it in the construction of the aw-diagram when spheres intersecting a filter need to be found. Our filters are a modification of previous filters described in [4]. Both these filters perform well on non-uniformly distributed data, which is an advantage over the previous grid based filtering approach [5]. In contrast to [4], filters proposed in this paper are not affected by the size of the largest sphere in the input data. This is possible, because we use power diagrams and our improved version of a method for intersection detection [6] to search our filters for intersecting spheres. This way smaller and thus more effective filters than in the previous approaches can be used, but there is also some additional cost of searching in the power diagram. On some datasets, e.g., for spheres with very similar radii, our approach is slower than the previous approaches, but on dense sets of spheres with various radii, it substantially outperforms its predecessors.

B. Paper Outline

Section II summarizes the work related to the acceleration of the edge tracing algorithm. Section III describes intersection detection based on power diagrams. Section IV is dedicated to aw-Voronoi diagrams, the edge tracing algorithm and space filters. Our new filtering approach, which combines the filters and the intersection detection method, is described in Section V. Experimental results are given in Section VI and Section VII concludes the paper.

II. Related Work

There are several algorithms for the construction of an aw-Voronoi diagram [3], [7], [8], [9], [10], [11] or its approximation [12]. The edge tracing and similar algorithms [3], [7], [8] are less complex and easier to implement than the others. They are based on a simple idea of tracing

 ISBN: 978-1-61208-172-4

Voronoi edges [13], which has been also used to construct the Voronoi diagram of convex objects [14].

Because the basic variant of the edge tracing algorithm [3] is slow, space filters are used to make it faster [5]. The basic idea is to compute a space filter, which limits the spheres tested by the edge tracing algorithm to the spheres intersecting the filter. In [5], a filter is the union of a filled sphere and a half-space. In the preprocessing, all input spheres are stored in a regular grid. To search the filter for intersecting spheres, grid cells covering the filter are visited and spheres associated to these cells are tested for an intersection with the filter. The grid based approach is limited to uniformly distributed input data. Another approach overcomes this problem by using Delaunay triangulation to search the filter [4]. The idea is to compute the Delaunay triangulation of sphere centers in the preprocessing and expand the size of each filter by the radius of the greatest sphere. The search for spheres intersecting the filter is performed in the triangulation and is limited to spheres having their centers inside the filter. The filter reduces its volume during the search. This approach can be inefficient for data with spheres of very different sizes.

III. Power Diagrams for Space Filtering

Our approach for the intersection detection between spheres and aforementioned filters (thoroughly described in Section IV-C) utilizes power diagrams and it is based on a method [6] (it is used as a black-box for the intersection detection in Section V, its details are not necessary for understanding the rest of the paper, but are inevitable for the implementation). First, let us remind the concept of power diagrams, then we will describe the idea of the intersection detection.

Let $S = \{s_1, ..., s_n\}$ be a set of spheres in $\mathbb{R}^3$, where each sphere s_i has a center $c_i \in \mathbb{R}^3$ and a non-negative radius r_i. Let $||x-y||$ denote the Euclidean distance between points $x, y \in \mathbb{R}^3$. Power diagrams employ a so called *power distance*. The power distance of a point x from a sphere s_i is defined as $d_{pow}(s_i, x) = ||c_i - x||^2 - {r_i}^2$. The *power region* of $s_i \in S$ is the set of points $PR(s_i) = \{x \in \mathbb{R}^3 : d_{pow}(s_i, x) \leq d_{pow}(s_j, x), \forall s_j \in S, s_j \neq s_i\}$. We say that s_i is the generator of $PR(s_i)$. If $PR(s_i)$ is empty, s_i is called *redundant*. The *power diagram* of S is the set of all power regions and is denoted by $PD(S)$. A power region $PR(s_i)$ is a convex polyhedron (possibly unbounded). If two power regions $PR(s_i)$ and $PR(s_j)$ share a face, we say that they are *neighbouring* and s_i is a *neighbor* of s_j.

A. Intersection Detection

The detection of intersections between spheres and a general object using power diagrams is discussed in [6], but we have slightly improved the described approach. The idea is as follows. A power diagram is traversed a region by region, for each explored region a decision is made, which of its neighbors have to be also scheduled and explored. Step by step, the explored area is enlarged until a certain condition, guaranteeing that there is no unknown sphere intersecting Q, is fulfilled. Now let us formalize this condition.

We assume $Q \subset \mathbb{R}^3$ is a solid without cavities. Such Q will be referred to as a *query object* Q. Let us have $PD(S)$ and a query object Q and let R be a subset of faces of $PD(S)$ forming one or more bounded or unbounded polygonal surfaces without holes, such that these surfaces do not intersect Q. Then the set R will be referred to as the *rampart of* Q, R_Q for short (see Figure 1). The *interior* of R_Q is the part of $\mathbb{R}^3$ bounded by R_Q and containing Q. The *exterior* of R_Q is $\mathbb{R}^3 \setminus (R_Q \cup$ interior of $R_Q)$. The generators of all non-empty power regions, which are adjacent to R_Q and lie in the interior of R_Q, are the *guardians* of R_Q. The generators of all non-empty power regions, which lie in the exterior of R_Q, are the *invaders* of R_Q.

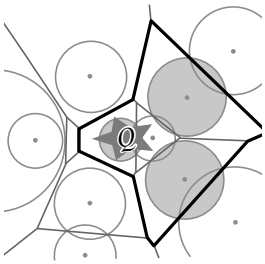


Figure 1. A rampart of Q – a thick black polyline, guardians – shaded circles (2D analogy).

Lemma 1: Given a set of spheres S and a query object Q. Let R_Q be a rampart of Q such that each guardian s_i of R_Q fulfills at least one of these conditions:

- s_i does not intersect Q, or
- s_i does not intersect any of its neighboring invaders.[1]

Then no invader of R_Q intersects Q.

Proof: is an analogy to the proof described in [6] and, for a sake of brevity, it is omitted here. ■

This lemma leads to the following algorithm. At the beginning, all the regions are supposed to be unknown. Starting with a region containing some point of Q, the power diagram is traversed by a breadth-first search. Each time an unknown region $PR(s_i)$ is explored, it is marked as known and its generator and the faces shared with its unknown neighbors are tested. Let f be a face shared by the known region $PR(s_i)$ and its unknown neighbor $PR(s_j)$. The region $PR(s_j)$ is *not* scheduled if and only if both the following conditions are met:

- $Q \cap f = \emptyset$,
- $Q \cap s_i = \emptyset$ or $s_i \cap s_j = \emptyset$.

[1]This condition is the difference from the original lemma in [6]. It reduces the size of explored area for a minimal additional cost.

 ISBN: 978-1-61208-172-4

This is done repeatedly until no scheduled region remains. In the end, the faces shared by the known and unknown regions make up the rampart R_Q satisfying Lemma 1 and all spheres intersecting Q have been found.

This algorithm does not deal with redundant generators. This issue is discussed in [6].

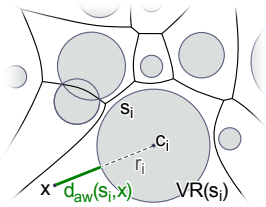
IV. Construction of AW-Voronoi Diagram

In this section, we will define the additively weighted Voronoi diagram, mention an algorithm for its construction and then describe filters that will help the algorithm to run faster. These filters are based on [4], but are smaller and thus more effective.

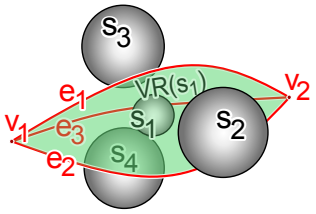
A. Additively Weighted Voronoi Diagram

The *aw-distance* of a point $x \in \mathbb{R}^3$ from $s_i \in S$ is defined as $d_{aw}(s_i, x) = ||c_i - x|| - r_i$. The *aw-Voronoi region* of $s_i \in S$ is the set of points $VR(s_i) = \{x \in \mathbb{R}^3 : \forall s_j \in S, s_j \neq s_i, d_{aw}(s_i, x) \leq d_{aw}(s_j, x)\}$. The *aw-Voronoi diagram* of S is the set of regions $VD(S) = \{VR(s_1), \ldots, VR(s_n)\}$. These terms are illustrated on a 2D analogy in Figure 2(a).

In 3D, the boundary of a region consists of faces, their boundary consists of edges and the boundary of edges consists of vertices. An example of a region with boundary edges is shown in Figure 2(b). Without further restrictions, some regions, faces and edges can be unbounded, because they continue to infinity. This would complicate diagram representation and algorithms, therefore we will always assume this is handled, e.g., by a virtual sphere representing infinity. Under this assumption, all regions, faces and edges are bounded, with the only exception of pure elliptic edges.



(a) 2D aw-Voronoi diagram and aw-distance



(b) 3D aw-Voronoi region $VR(s_1)$, its boundary edges e_1, e_2, e_3 and their bounding vertices v_1 and v_2

Figure 2. Additively weighted Voronoi diagram

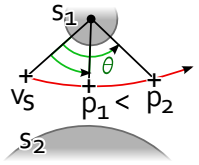
From the geometric point of view, each face is a part of a plane or a hyperboloid, each edge is a part of a line, a hyperbola or an ellipse and each vertex is a point. But the geometrical interpretation of boundary elements is not so important. What matters is the information which input spheres define their geometry. Each region is given by one input sphere, each face by two, each edge by three and each vertex by four spheres. We will always assume that this information (which spheres define the geometry) is available for each boundary element. Only edges and vertices are necessary to describe the diagram. Faces can be determined from edges. This simplifies the representation of a diagram to a graph of vertices and edges.

B. Diagram Construction Algorithm

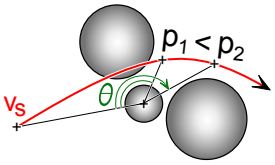
The graph of Voronoi vertices and edges can be constructed by the *edge tracing algorithm* [3], [7], which works as follows. First, an initial vertex is found and four edges incident to this vertex are pushed onto a stack. Then, until the stack is empty, edges are popped from the stack and for each edge, the second bounding vertex is found. If it is a new vertex, three more edges are pushed onto the stack.

The algorithm finds the whole component of edge connectivity. Finding an initial vertex [7], elliptic edges without bounding vertices, handling infinity and discovering all components of the diagram is not necessary for understanding our paper. Important is to know how an edge is traced for the second bounding vertex, so let us describe it now.

When an edge is pushed onto a stack, only one of its two bounding vertices is known - the *start vertex* v_s. From the position of v_s and its four defining spheres, three spheres defining the edge and the direction of the edge can be determined. A candidate to the other bounding vertex can be found by taking a sphere and, together with the three spheres, computing its position. The candidate, which has its position closest to v_s, creates the other bounding vertex - the *end vertex* v_e. Comparing distances along edge can be realized via *angular distance* shown in Figure 3.



(a) 2D example



(b) 3D example

Figure 3. Comparing distances along an edge via angular distance θ - the point p_1 is closer to the vertex v_s than the point p_2

C. Space Filters

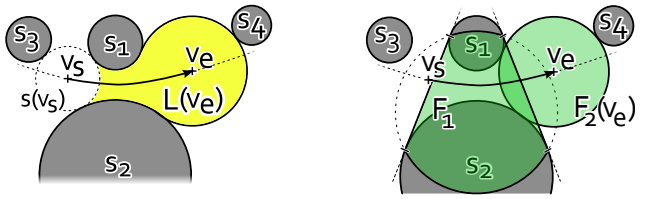
In a brute force approach, the end vertex is found by trying all spheres except the three spheres defining the edge, computing vertex candidates corresponding to these spheres and taking the one with minimal angular distance. This can be improved by using spatial filters. The idea [5] is to limit the space, which the sphere for the end vertex must intersect.

Each filter is defined individually for an edge from the knowledge of the start vertex v_s, the end vertex candidate v_e and the three spheres defining the geometry of the edge.

Let us first define an *ideal filter* $L(v_e)$. Its 2D analogy is shown in Figure 4(a). Each point x of the edge is the center of a filled ball $s(x)$ tangent to the defining spheres (or a half-space if x is at infinity). The ideal filter is

defined as the union of all such filled balls with centers $x \in [v_s, v_e]$ on the edge without the ball at the start vertex: $L(v_e) = \bigcup_{x \in edge[v_s,v_e]} s(x) \backslash s(v_s)$. The ball $s(v_s)$ cannot be intersected by any sphere since v_s is a true Voronoi vertex. If $L(v_e)$ is not intersected by another sphere, then v_e is the true end vertex, else the sphere intersecting $L(v_e)$ defines a candidate v'_e closer to v_s and another $L(v'_e) \subset L(v_e)$.

Because the shape of $L(v_e)$ is not trivial, we will cover it by a bigger *filter* $F(v_e)$, which is simpler. Let us first describe the filter for a non-elliptic edge with 2D analogy depicted in Figure 4(b). It is the union of two filters $F(v_e) = F_1 \cup F_2(v_e)$. The first filter F_1 is computed (in 3D) from supporting planes tangent to the spheres defining the edge as the filled ball passing through the corresponding tangent points, cut off by the supporting planes. The filter F_1 is static - it does not depend on v_e. The second filter $F_2(v_e) = s(v_e)$ is the filled ball centered at the end vertex candidate v_e or a half-space if v_e is at infinity. When the edge is an ellipse, we compute $F(v_e)$ as the smallest ball enclosing the union $\bigcup_{x \in ellipse} s(x)$. In this case, the filter is also static.[2]

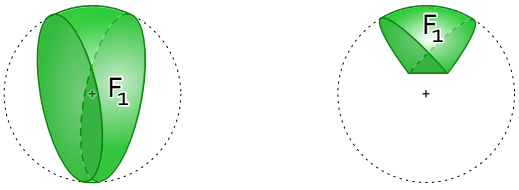


(a) Ideal filter $L(v_e)$

(b) Filter $F(v_e) = F_1 \cup F_2(v_e)$ that covers $L(v_e)$

Figure 4. 2D analogy of filters for the given edge – s_1 and s_2 define the geometry of the edge, v_s is the start vertex and v_e is the end vertex candidate.

The situation in 3D is similar to the 2D analogy. The filter F_2 is a sphere or a half-space, the filter F_1 is a sphere without two half-spaces. Notice that the center of the sphere may or may not be included in F_1 as shown in Figure 5.



(a) F_1 includes the center

(b) F_1 does not include the center

Figure 5. Two possible cases of F_1 in 3D

[2]Unlike the original filters [4], the proposed filters F_1, F_2 are not expanded by the radius of the biggest sphere of S. Another improvement is the reduction of the volume of F_1 by the two cutting planes.

V. Proposed Method

Here we will describe our filtering approach for end vertex search. It combines the aforementioned filters with the described method for the intersection detection.

First, a power diagram of S is computed in a preprocessing step. Then, during the construction of $VD(S)$, a filter $F(v_e)$ is created for each end vertex search. Using $PD(S)$ and the method for intersection detection described in III-A, a search for spheres intersecting $F(v_e)$ is performed. If such a sphere is not found, then v_e is the real end vertex. But if some sphere s_i intersecting $F(v_e)$ is found, it must be tested whether it defines a candidate v'_e closer to v_s than v_e. This is done by computing the positions of possible end vertex candidates v'_e and testing their angular distance against the angular distance of v_e. If v'_e is closer, this means that $L(v_e)$ is intersected by s_i, so $F(v_e)$ is replaced by $F(v'_e)$, possibly reducing the space to be further searched.

Now let us describe this processing of a filter $F(v_e)$ in detail. The filter consists of two parts, of filters F_1 and F_2, and the search for spheres intersecting them is done independently of each other (with one exception – if a sphere intersects both filters F_1 and F_2, its corresponding end vertex candidate is computed only once). A filter F_1 is static, it does not change during one end vertex search. A filter F_2 is dynamic – if a sphere intersecting F_1 or F_2 is found and it defines a closer end vertex candidate v'_e (i.e. with a smaller angular distance), the filter F_2 is updated. Its center is moved into the new vertex candidate v'_e and its radius is reduced so the resulting F_2 is tangent to the spheres defining v'_e. Let us note that although F_2 changes during its processing, the used algorithm for intersection detection still works correctly.[3]

Let us show the algorithm on an example in Figure 6. For the sake of simplicity, it is just a 2D analogy. The initial configuration is shown in Figure 6(a). There is a set of generators $\{s_1, \ldots, s_{12}\}$, an edge e and a start vertex v_s. The start vertex v_s is given by a set of three (in 3D four) generators $\{s_1, s_2, s_3\}$ and a sphere inscribed to these generators. The edge e is given by a set of two (in 3D three) generators $\{s_1, s_2\}$ and the orientation. The task is to find another generator, different from s_1 and s_2, which, together with s_1 and s_2, creates the end vertex v_e, i.e., the vertex candidate on e closest (in terms of the angular distance) to v_s. The generator will be found among generators intersecting appropriate space filters. The initial filter $F(v_\infty) = F_1 \cup F_2(v_\infty)$ can be computed from the initial configuration. The filter F_1 is static, so it can be searched through first. In Figure 6(b), the search through F_1 starts from a region intersecting F_1, e.g., the region $PR(s_1)$,

[3] The main idea of the proof of this claim is as follows. If a sphere s_i causes an update of a filter, the updated F_2 is tangent to s_i. If F_2 intersects some sphere, then there must be a neighbor s_j of s_i such that either F_2 intersects the face generated by s_i and s_j, or $s_i \cap s_j \neq \emptyset$ and thus the region of s_j, intersecting F_2, is to be scheduled and F_2 cannot 'run off" the searched area.

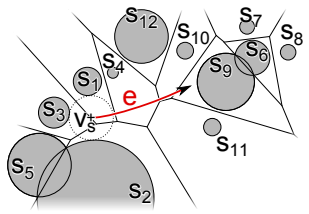
 ISBN: 978-1-61208-172-4

and explores generators $s_1, \ldots, s_5$. Although $s_4 \cap F_1 = \emptyset$, s_4 is explored because $PR(s_4) \cap F_1 \neq \emptyset$. Although the other explored generators intersect F_1, none of them can create a suitable vertex candidate to reduce the dynamic filter. Next in Figure 6(c), the search through the dynamic filter $F_2(v_\infty)$ starts from a region intersecting $F_2(v_\infty)$. We have selected $PR(s_6)$ for this example, but the real selection should be based on some heuristic, which will be discussed later. So s_6 is the first generator scheduled for exploring. In Figure 6(d), s_6 is explored. Because s_6 intersects $F_2(v_\infty)$, it is used to compute a vertex candidate $v_1 \in e$. The angular distance of v_1 from v_s is less than the angular distance of v_∞ from v_s, therefore the filter is updated to $F_2(v_1)$. This is a big reduction - from a half-space to a sphere. From now on it is sufficient to check only generators intersecting $F_2(v_1)$. Because $PR(s_6) \cap F_2(v_1) = \emptyset$ and $s_6 \cap F_2(v_1) \neq \emptyset$, only neighbors of s_6 intersecting s_6 must be scheduled for exploring - at least one of them will have its power region intersecting $F_2(v_1)$. In this case, only s_9 is scheduled. Next in Figure 6(e), s_9 is explored, a closer vertex candidate v_2 computed and the filter updated to $F_2(v_2)$. Neighbors s_{10} and s_{11} are scheduled, because the edge (face in 3D) between s_9 and s_{10} and between s_9 and s_{11} intersect $F_2(v_2)$. Neither s_{10} nor s_{11} intersects $F_2(v_2)$, therefore they cannot create a closer end vertex candidate. Exploring s_{10} and s_{11} results in scheduling s_4, s_{12}, and s_2. Next in Figure 6(f), s_4 is explored, a closer end vertex candidate v_3 computed and the filter updated to $F_2(v_3)$. The remaining generators s_{12}, s_2, s_1 and s_3 are explored, but none of them creates a closer candidate, therefore v_3 is the end vertex v_e.

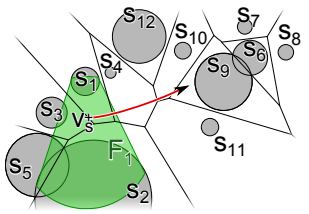
Because a sphere intersecting F_1 can cause a reduction of F_2, but not vice versa, it is better to process the filter F_1 first and then F_2. It is also possible that F_2 becomes fully absorbed by F_1 after a filter update. In such a case the rest of the processing of F_2 can be skipped, the current end vertex candidate is the real end vertex. So it is useful to perform the $O(1)$ test whether $F_2 \subseteq F_1$ before the processing of F_2 and after each update of F_2 during its processing.

Still, filters F_2 are often very big at the beginning of their processing. We have developed two simple heuristics, each of them quickly and substantially reduces the size of F_2. One of them tests the intersections between F_2 and all neighbors of the three spheres generating the current edge. Often, some of these neighbors intersect F_2 and thus reduce it.

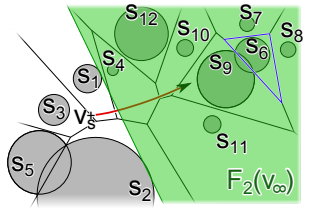
The other heuristic localizes the center of F_2 in $PD(S)$ by traversing from a region to region, using a visibility walk [15]. The walk starts in the region of one of the spheres generating the current edge and locates the region containing the center of F_2. A generator of each region explored during the walk is tested, whether it intersects F_2. If so, F_2 is updated, the destination point is changed to the new center of F_2 and the walk continues from the lastly explored region. A performance of both heuristics is similar in practice.



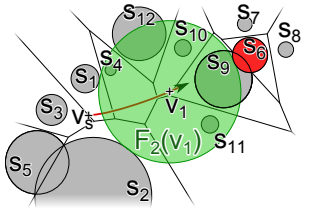
(a) The initial configuration with a start vertex v_s given by $\{s_1, s_2, s_3\}$ and with an edge e given by $\{s_1, s_2\}$. We want to find the end vertex $v_e \in e$ closest to v_s.



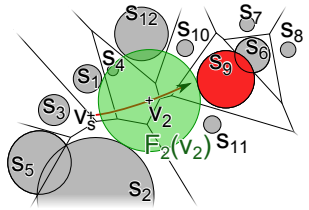
(b) The search through F_1 explores s_1, s_2, s_3, s_4, and s_5, but no suitable end vertex candidate is found.



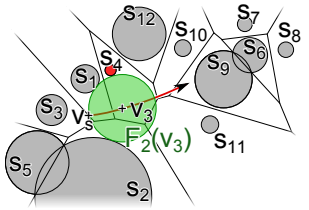
(c) The the search through $F_2(v_\infty)$ starts from, e.g., $PR(s_6)$, so s_6 is scheduled for exploring.



(d) When s_6 is explored, a candidate $v_1 \in e$ is computed and the filter updated to $F_2(v_1)$.



(e) When s_9 is explored, a candidate $v_2 \in e$ is computed and the filter updated to $F_2(v_2)$. Neighbors s_{10} and s_{11} are scheduled, then explored and s_4, s_{12}, and s_2 scheduled.



(f) When s_4 is explored, a candidate v_3 is computed and the filter updated to $F_2(v_3)$. No closer candidate is found after exploring generators s_{12}, s_2, s_1, and s_3, so $v_e = v_3$.

Figure 6. Algorithm example in 2D

VI. Experiments and Results

We compared the proposed method against the method [4], based on Delaunay triangulation (hence it will be referred to as Delaunay method), because it is the best competitive method for general input data. We did not compare it against the brute force method [3], because it is too slow to be interesting. Both implementations are written in C# 3.5 and the experiments were done on Intel Core i7, 3.07GHz with 6GB RAM running Windows 7. We were interested in the running time consumed by the construction of aw-Voronoi diagrams and the average number of generators explored during one end vertex search. We distinguish a *touched* generator and a *tested* generator. To *touch* a generator means to find out whether the generator is already known (which takes $O(1)$ time). To *test* a generator means to compute its intersection with one part of a filter. Moreover in the proposed method,

 ISBN: 978-1-61208-172-4

it also includes a check of a local neighborhood of the generator, as described in III-A. For datasets used in our experiments, this additional check runs in $O(1)$ time in average case. But the hidden multiplicative constant is big – the test of one generator in the proposed method is roughly 10 times slower in comparison with Delaunay method. In consequence, for sets of spheres with similar radii and for sparse sets of spheres, the proposed method is roughly 2 times slower than Delaunay method. But with increasing differences in radii and increasing density of spheres, the filters used in Delaunay method (as well as the regular grid-based filtering approach mentioned in II) become ineffective, while the performance of the proposed method is almost unaffected.

This is illustrated in the following two experiments. In the first experiment, the density of the dataset is constant and the difference of radii changes. The used dataset contains 10000 points randomly scattered in a unit box and one sphere lying outside the box. The radius r of the sphere changes from 0 to 1. Figure 7 shows dependency of the construction time of the aw-Voronoi diagram on r, Figure 8 the average number of performed tests per one edge of the diagram. For a small value of r, Delaunay method outperforms the proposed method. But while the running time (as well as the number of tests) of the proposed method is minimally affected by r, the filters used in Delaunay method become inefficient for increasing r.

In the second experiment, the difference of radii is constant and the density changes. The dataset is formed by a unit box containing 10 congruent spheres with a radius 0.1 and n points, where $n \in \langle 100, 64000 \rangle$. Figure 9 shows that while the construction based on the proposed method runs approximately in $O(n^{1.2})$ time, the construction based on Delaunay method runs in $O(n^{1.9})$. Note that the proposed method outperforms Delaunay method starting at $n = 250$.

This corresponds with the average numbers of performed tests per one edge, shown in Figure 10. On the given range of n, the numbers of generators tested and touched by the proposed method increase less than 3 times, whereas the growth for Delaunay method is more than 500 times.

Now let us mention datasets, which are suitable neither for the proposed, nor for the previous filtering approaches. The filtering approaches are inefficient for datasets with many mutually intersecting spheres. The dataset used in the last experiment consists of n unit spheres with centers randomly scattered in a unit box and of one point. This point prevents the improving of the performance with a trick, where the radius of each sphere is reduced by the minimal radius. In consequence, filters are intersected by almost all spheres and the diagram construction runs approximately in $O(n^2)$, regardless of the used filtering approach. Here the original edge tracing algorithm outperforms the filter-based methods, nevertheless, it also runs in quadratic time. Recall that our filter is an approximation encapsulating the ideal filter.

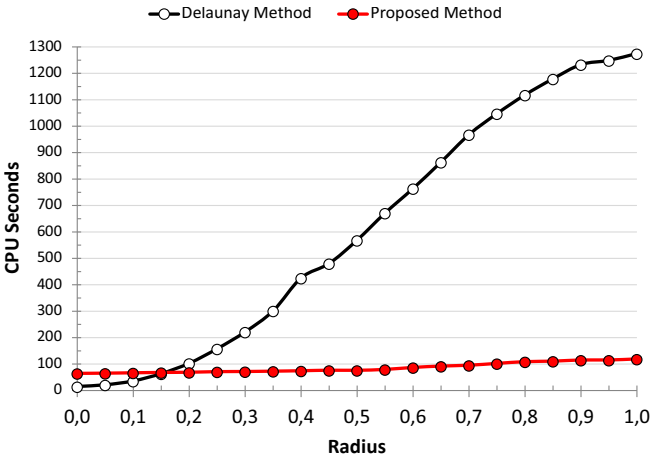


Figure 7. Aw-Voronoi diagram computation time – 10000 random points in a unit box and one sphere of the given radius.

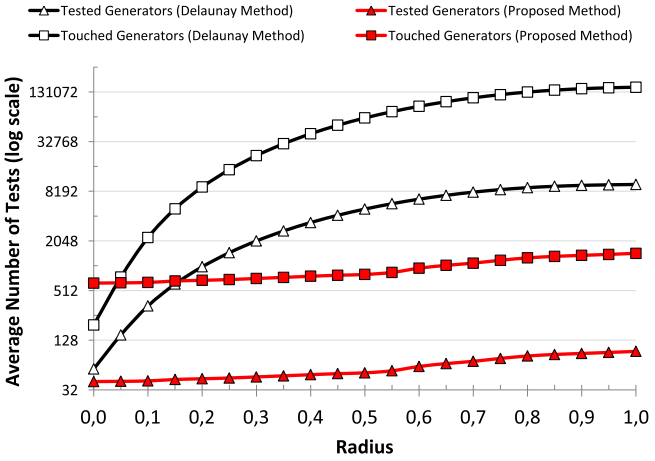


Figure 8. Average number of tests per one aw-Voronoi edge – 10000 random points in a unit box and one sphere of the given radius.

Another example of data unsuitable for our filtering approach is data with high probability that the power region of a sphere intersects the approximation filter but the sphere does not intersect the ideal filter. In this case many spheres will be explored but only a few of them will actually reduce the filter.

Our implementation runs in only one CPU thread. When a diagram is being constructed, many edges must be traced. With the increasing number of computation cores in modern hardware, it might pay off to trace more edges in parallel, using our filtering approach to speed up the tracing.

VII. Conclusion

We have proposed a new filtering approach for a construction of aw-Voronoi diagrams. The proposed approach combines filters based on [4] and a method for the intersection detection [6]. We have slightly improved both these two techniques. We have shown that on dense sets of

 ISBN: 978-1-61208-172-4

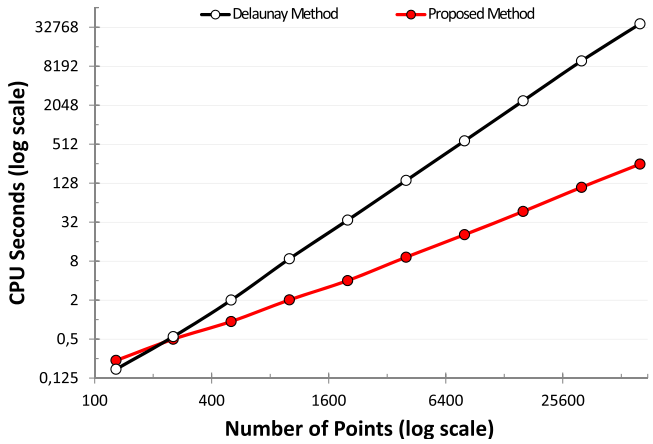


Figure 9. Aw-Voronoi diagram computation time – 10 spheres with a radius 0.1 together with an increasing count of points in a unit box.

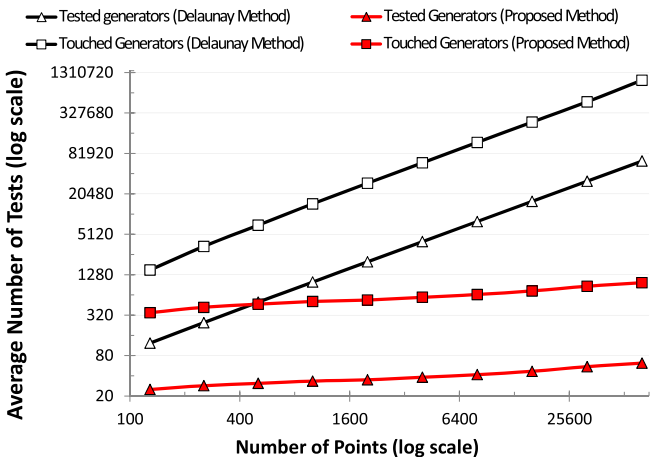


Figure 10. Average number of tests per one aw-Voronoi edge – 10 spheres with a radius 0.1 together with an increasing count of points in a unit box.

spheres with various radii the proposed filtering approach significantly outperforms the previous Delaunay approach.

As a future work, we recommend to investigate a hybrid method combining the proposed and Delaunay approaches together with the original edge tracing algorithm. First, a simple statistic on an input set of spheres would be computed and according to its result, the most suitable technique would be chosen for the diagram construction. Another open problem is to design filters suitable for a set of mutually intersecting spheres.

ACKNOWLEDGMENT

This work has been supported by the Czech Science Foundation under the project P202/10/1435 and by the University of West Bohemia under the project SGS-2010-028.

REFERENCES

[1] A. Poupon, "Voronoi and Voronoi-related tessellations in studies of protein structure and interaction," *Current Opinion in Structural Biology*, vol. 14, no. 2, pp. 233 – 241, 2004.

[2] D.-S. Kim, "A beta-complex solves all geometry problems in a molecule," in *The 6th International Symposium on Voronoi Diagrams in Science and Engineering*, 2009, pp. 254–260.

[3] D.-S. Kim, Y. Cho, and D. Kim, "Euclidean Voronoi diagram of 3d balls and its computation via tracing edges," *Computer-Aided Design*, vol. 37, pp. 1412–1424, 2005.

[4] M. Manak and I. Kolingerova, "Fast discovery of Voronoi vertices in the construction of Voronoi diagram of 3d balls," *International Symposium on Voronoi Diagrams in Science and Engineering*, vol. 0, pp. 95–104, 2010.

[5] Y. Cho, D. Kim, H.-C. Lee, J. Y. Park, and D.-S. Kim, "Reduction of the search space in the edge-tracing algorithm for the Voronoi diagram of 3d balls," in *ICCSA (1)*, 2006, pp. 111–120.

[6] M. Zemek and I. Kolingerová, "Power diagrams and intersection detection," in *ICCSA*, vol. 3, 2011, pp. 163–173.

[7] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova, "An algorithm for three-dimensional Voronoi S-network," *Journal of Computational Chemistry*, vol. 27, no. 14, pp. 1676–1692, 2006.

[8] D.-S. Kim, D. Kim, Y. Cho, and K. Sugihara, "Quasi-triangulation and interworld data structure in three dimensions," *Computer-Aided Design*, vol. 38, no. 7, pp. 808–819, 2006.

[9] J.-D. Boissonnat and C. Delage, "Convex hull and Voronoi diagram of additively weighted points," in *ESA*, 2005, pp. 367–378.

[10] D. Kim, Cho, and D.-S. Kim, "Region expansion by flipping edges for Euclidean Voronoi diagrams of 3d spheres based on a radial data structure," in *ICCSA (1)*, 2005, pp. 716–725.

[11] D. Kim and D.-S. Kim, "Region-expansion for the Voronoi diagram of 3d spheres," *Computer-Aided Design*, vol. 38, pp. 417–430, 2006.

[12] E. Yaffe and D. Halperin, "Approximating the pathway axis and the persistence diagram of a collection of balls in 3-space," in *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*. New York, NY, USA: ACM, 2008, pp. 260–269.

[13] M. Tanemura, T. Ogawa, and N. Ogita, "A new algorithm for three-dimensional Voronoi tessellation," *Journal of Computational Physics*, vol. 51, no. 2, pp. 191 – 207, 1983.

[14] V. A. Luchnikov, N. N. Medvedev, L. Oger, and J.-P. Troadec, "Voronoi-delaunay analysis of voids in systems of nonspherical particles," *Phys. Rev. E*, vol. 59, no. 6, pp. 7205–7212, Jun 1999.

[15] O. Devillers, S. Pion, and M. Teillaud, "Walking in a triangulation," *Internat. J. Found. Comput. Sci*, vol. 13, pp. 106–114, 2001.

 ISBN: 978-1-61208-172-4