

# 3D Kinetic Alpha Complexes and Their Implementation <sup>\*</sup>

Michael Kerber<sup>†</sup> and Herbert Edelsbrunner<sup>‡</sup>

September 25, 2012

## Abstract

Motivated by an application in cell biology, we describe an extension of the kinetic data structures framework from Delaunay triangulations to fixed-radius alpha complexes. Our algorithm is implemented using *CGAL*, following the exact geometric computation paradigm. We report on several techniques to accelerate the computation that turn our implementation applicable to the underlying biological problem.

**Keywords.** Computational geometry, Delaunay triangulations, alpha complexes, kinetic data structures, spatial sorting, exact geometric computation, implementation, software experiments.

---

<sup>\*</sup>This research is partially supported by NSF under grant DBI-0820624, by ESF under the Research Network Programme, and by the Russian Government under mega project 11.G34.31.0053.

<sup>†</sup>IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria; Max-Planck-Center for Visual Computing and Communication (MPC-VCC), Saarbrücken, Germany; Stanford University, Stanford, California

<sup>‡</sup>IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria; Departments of Computer Science and of Mathematics, Duke University, Durham, North Carolina; Delaunay Lab of Discrete and Computational Geometry, Yaroslavl' State University, Russia; Geomagic, Research Triangle Park, North Carolina.

# 1 Introduction

Consider a finite set of particles or points in  $\mathbb{R}^3$ , moving in time along continuous trajectories. Interpreting these points as the centers of moving objects, we are interested in the topological changes the configuration undergoes. Our interest in this problem originates in a sorting process that segregates cells during zebrafish development, as studied by Heisenberg and Krens [8]. The sorting process operates on intermixed configurations of cells in which different types have different physical properties. One example is a mix of two cell types, in which the cells of the first type have a strong preference for neighboring cells of the same type and a strong dislike of exposed boundary, while the cells of the second type have milder preferences and dislikes. The typical outcome is that the cells of the first type form a ball-like shape that is engulfed by a spherical shell consisting of cells of the second type (compare Figure 2).

In an effort to formalize the sorting process and to make it amenable to detailed and objective measurements, Heisenberg, Krens, and the authors of this paper introduced the *restricted Voronoi medusa* as a mathematical representation. It is a geometric body in 4-dimensional space-time obtained by stacking up restricted Voronoi regions in  $\mathbb{R}^3$  [5]. At any moment in time, the Voronoi region of a particle is intersected with a ball, and the resulting bodies are glued together to form the 4-dimensional structure. Applying persistent homology to the time function on this structure yields fine-grained information about the sorting process that is difficult to observe directly.

**Results.** For any moment in time, the dual of the restricted Voronoi tessellation is an alpha shape of fixed radius. Its evolution under continuous motion of the objects gives rise to a combinatorial representation of the medusa that carries most of its topological information (see [5]). We make two contributions in this context:

1. We describe an algorithm that maintains a fixed-radius alpha complex for points moving on piecewise algebraic trajectories in  $\mathbb{R}^3$ . The algorithm supports insertions and deletions of points and allows for piecewise algebraic trajectories.
2. We convert the kinetic algorithm into robust and efficient software. Basing the implementation on the CGAL package for kinetic data structures by Daniel Russel [12], it achieves correctness through the exact geometric computation paradigm.

Contribution 1 extends previous work for computing kinetic Delaunay triangulations; we believe this extension to find applications beyond our original problem. The requirement of correctly comparing algebraic numbers, without tolerance for inaccuracy or approximation in Contribution 2 seriously slows down the software, even for piecewise-linear trajectories. To counteract, we introduce techniques that speed-up the computations without sacrificing their correctness. We evaluate the effectivity of these techniques experimentally.

We point out that due to space limitations, this extended abstract omits the algorithm to construct the medusa out of the kinetic alpha shape. Most steps of this algorithm are straight-forward; special care is needed, however, when inserting or deleting trajectories during the process. We refer to [10] for a more extended version of our results.

**Outline.** Section 2 explains background from computational geometry and topology. Section 3 describes the kinetic algorithm for fixed-radius alpha complexes. Section 4 describes techniques to speed up the computations. Section 5 concludes the paper.

## 2 Background

We review the fundamental geometric data structures that are required in this work. Voronoi tessellations and Delaunay triangulations are treated in most computational geometry textbooks, including [2, 3], and alpha complexes are described in [4, 6]. For the most part, the discussion focuses on the 3-dimensional case. Most definitions and properties extend to higher dimensions as well as to the plane.

**Simplicial complexes.** We recall that a  $k$ -simplex is the convex hull of an affinely independent set of  $k + 1$  points in some Euclidean space. A *face* is a simplex defined by a subset of the  $k + 1$  points. It is *proper* if the subset is different from the set. Reversing the direction, we call the  $k$ -simplex a *coface* of its face. We define a *simplicial complex* as a finite collection of simplices that is closed under the face relation, with the additional property that any two simplices in the collection are either disjoint or their intersection is a face of both. The *boundary* of a  $k$ -simplex is the collection of its  $(k - 1)$ -faces. The simplices of dimension 0, 1, 2, and 3 are referred to as *vertices*, *edges*, *triangles*, and *tetrahedra*. The *star* of a  $k$ -simplex is the set of simplices that contain the  $k$ -simplex as a face. Noting that the star is in general not closed under the face relation, we define the *closed star* as the set of all simplices in the star and of all faces of these simplices. It is the smallest simplicial complex that contains the star. Finally, if  $\sigma$  is a  $k$ -simplex and  $u$  is a point that does not lie in the  $k$ -plane of the simplex, then the *join*, denoted as  $u * \sigma$ , is the  $(k + 1)$ -simplex that is the convex hull of  $u$  and the vertices of  $\sigma$ .

It is convenient to also introduce an abstract counterpart to the above geometric concept of a simplicial complex. Specifically, an *abstract simplicial complex* consists of a finite set of (abstract) elements and a collection of subsets that is closed under the subset relation. We may map each element to a point in some Euclidean space, and each subset to the convex hull of the points that correspond to its elements. If the dimension of the space is sufficiently high and the points are well chosen, this is a simplicial complex, which we refer to as a *geometric realization* of the abstract simplicial complex. Here is an example of this construction. Consider a finite set,  $X$ , of possibly overlapping bodies, and define the *nerve* as the collection of subsets of  $X$  with non-empty common intersection. We note that the nerve is an abstract simplicial complex. Indeed, the bodies are the elements, and if  $A \subseteq X$  is a set in the nerve, then every subset of  $A$  is also in the nerve. A useful result is the Nerve Theorem [4], which states that if the bodies in  $X$  are convex then every geometric realization of the nerve has the same homotopy type as the union of the bodies. Intuitively, this means that one can be transformed into the other by continuous transformations like bending, shrinking, and expanding, but without gluing and cutting.

**Voronoi tessellations and Delaunay complexes.** Consider now a finite set of points,  $U$ , in  $\mathbb{R}^3$ . The *Voronoi region* of a point  $u$  in  $U$  is the set of points  $x \in \mathbb{R}^3$  that have  $u$  as the closest point in  $U$ :

$$\text{vor}(u) = \{x \in \mathbb{R}^3 \mid \|x - u\| \leq \|x - v\|, \forall v \in U\}.$$

Note that  $\text{vor}(u)$  is convex. The *Voronoi tessellation* of  $U$  is the set of Voronoi regions of its points. If the points in  $U$  are in general position, by which we mean that no four lie in a common plane and no five lie on a common sphere, then the Voronoi regions intersect in a rather predictable pattern. Specifically, the intersection of any two is either empty or a (2-dimensional) polygon, the intersection of any three is either empty or a (1-dimensional) edge, and the intersection any four is either empty or a (0-dimensional) point. Furthermore, the intersection of five or more Voronoi regions is necessarily empty.

We get the dual *Delaunay complex* if we replace each non-empty intersection of Voronoi regions by the convex hull of the points that generate the Voronoi regions containing the intersection. That complex equals the nerve of the Voronoi tessellation, geometrically realized in the canonical way. Equivalently, we may define the Delaunay complex as the set of convex hulls of subsets of points that have the *empty sphere*

*property*. Specifically, this means that there exists a sphere that passes through the points of the subset and all other points in  $U$  lie strictly outside this sphere. We note that the center of this sphere belongs to the intersection of the corresponding Voronoi regions. Assuming general position, the Delaunay complex is a simplicial complex, which is generally referred to as the *Delaunay triangulation*. It is a geometric realization of the nerve of the Voronoi tessellation.

**Restricted Voronoi tessellations and alpha complexes.** Fixing a positive radius,  $\alpha_0$ , we define the *restriction* of a Voronoi region to be its intersection with the closed ball of radius  $\alpha_0$  centered at the generating point:

$$\text{res}(u) = \{x \in \text{vor}(u) \mid \|x - u\| \leq \alpha_0\}.$$

Again, we usually drop  $U$  from the notation. The *restricted Voronoi tessellation* of  $U$  is the set of restricted Voronoi regions of its points. In contrast to the unrestricted case, each restricted Voronoi region is bounded, and the tessellation covers only the union of balls and not the entire space.

As before, we assume general position so we can dualize by geometrically realizing the nerve. The resulting simplicial complex is called the *alpha complex*. Since  $\text{res}(u) \subseteq \text{vor}(u)$ , for each point  $u$  in  $U$ , the alpha complex is a subcomplex of the Delaunay triangulation. Next, we derive an equivalent condition for a Delaunay simplex to lie in the alpha complex which is more suitable for computations. Each  $k$ -simplex in the Delaunay triangulation has a unique circumscribed  $(k - 1)$ -sphere in its supporting  $k$ -plane. We call its center the *circumcenter*, its radius the *circumradius*, the ball in  $\mathbb{R}^3$  with this center and this radius the *circumball*, and the sphere that bounds the circumball the *circumsphere* of the  $k$ -simplex. Note that the circumsphere is the smallest sphere that passes through the vertices of the  $k$ -simplex. We call the  $k$ -simplex *short* if its circumradius is smaller than or equal to  $\alpha_0$ . Finally, we call the  $k$ -simplex *Gabriel* if its circumball has no point of  $U$  in its interior.

**Lemma 1 (Short&Gabriel Lemma)** *A simplex in the Delaunay triangulation of  $U$  belongs to the alpha complex, for radius  $\alpha_0$ , if and only if it is short and Gabriel, or it is the face of another Delaunay simplex that is short and Gabriel.*

The face of a short simplex is necessarily short, but the face of a Gabriel simplex is not necessarily Gabriel. It follows that all simplices in the alpha complex are short, but not all simplices need to be Gabriel. Also note that a tetrahedron is in the Delaunay triangulation iff it is Gabriel; therefore, it is in the alpha complex iff it is short. In our application, we use the restricted Voronoi tessellation to model a set of biological cells for which the positions of their nuclei are known. Indeed, a cell tends to minimize its surface area and usually does not grow larger than a certain size. Therefore, a restricted Voronoi region appears to be a good approximation of the actual cell shape and is still simple enough for our computational purposes.

### 3 Kinetic Alpha Complexes

In this section, we describe the algorithm that maintains the alpha complex for a fixed radius  $\alpha_0 > 0$ . We pay particular attention to the certificates that govern the sequence of operations needed to preserve the correctness of the structure at all times.

**The kinetic framework.** The input to our algorithm is a finite set of trajectories, each a continuous map  $u : [a, b] \rightarrow \mathbb{R}^3$  with  $0 \leq a < b \leq 1$ . For simplicity, we assume it to be piecewise linear, with  $a = t_0 < t_1 < \dots < t_k = b$  such that there are points  $a_j, b_j \in \mathbb{R}^3$  for which  $u(t) = (1 - t)a_j + tb_j$  for  $t_j \leq t \leq t_{j+1}$ . In other words, we can write  $u(t) = (f_1(t), f_2(t), f_3(t))$  such that between  $t_j$  and  $t_{j+1}$ ,

each  $f_i$  is a polynomial of degree 1. We call  $t_0, t_1, \dots, t_k$  the *bending events* of the trajectory. Furthermore, we assume that the trajectories do not meet each other, that is,  $u(t) \neq u'(t)$  for all  $u, u' \in U$  and all  $t$  for which both trajectories are defined.

Our task is to maintain a data structure that goes from an initial configuration, at time  $t = 0$ , to the final configuration, at time  $t = 1$ . For that, the data structure is constructed at time  $t = 0$ , and maintained through a sequence of update operations until the final configuration is reached. It is assumed that the number of updates is finite, and we call the time of an update an *event*. Events are detected by defining suitable *certificate functions*, also referred to as *certificates*. At any moment  $t$  different from any event, we have a collection of *active certificates*, all being non-zero at  $t$ . Importantly, they guarantee that as long as no certificate changes its sign, our data structure remains structurally unchanged. To detect the next event, the algorithm then finds the smallest root of any active certificate that is greater than  $t$ . It handles the event by updating the data structure and the collection of active certificates. Throughout this paper, we make the simplifying assumption that all events are distinct, that is, no two events happen at the same moment in time; see also Section 5.

**Maintaining the Delaunay triangulation.** Since we need it later, we begin by reviewing the kinetic algorithm for 3-dimensional Delaunay triangulations described in [12]. Besides changes brought about by insertions and deletions of points, and switches to new trajectory segments, there are only two configurations that trigger a structural change in the triangulation:

- five points of  $U$  lie on a common sphere, and the open ball bounded by this sphere contains no points of  $U$ ;
- four points of  $U$  lie on a common plane, and one of the open half-spaces bounded by this plane contains no points of  $U$ .

We call each such configuration a *degeneracy*. Consistent with the above assumption of distinct events, we assume that at every moment of time there is only one degeneracy, and that each degeneracy lasts only for a single moment. In other words, we can find a small open interval in time during which the given degeneracy exists at a single point in time, and it is the only degeneracy that occurs during this interval. We can therefore study the effect of the degeneracy by considering the non-degenerate local configurations right before and right after the degeneracy. Consider for example a degeneracy of the first type, which involves five points. Right before the degeneracy, the five points span two Delaunay tetrahedra with a common triangle, and right after the degeneracy they span three tetrahedra so that each pair shares a triangle and all three share an edge. Of course, it can also be the other way round. Importantly, we can transform one configuration to the other by flipping. In this particular case, we substitute three for two or two for three tetrahedra, calling the operation a *2-3-flip*; see Figure 1. To avoid a case analysis, we represent the triangulation using a vertex

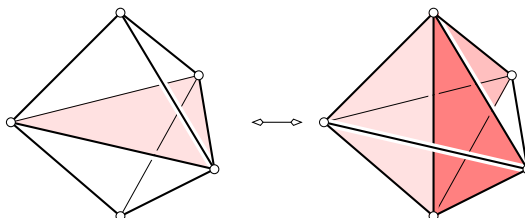


Figure 1: Illustration of a 2-3-flip that alters the triangulation of a triangular double pyramid. On the left, the five points span two tetrahedra meeting in a triangle. After the flip, the triangle is replaced by an edge and the three incident triangles that connect the edge to the remaining three points.

at infinity that is joined to every simplex in the boundary of the convex hull of  $U$ . Effectively, we embed

the triangulation on a 3-sphere. This way, we can add the point at infinity to the set of four points forming a degeneracy of the second type, thus getting a degeneracy of the first type, which is handled by a 2-3-flip, as described above.

**Flip events.** The transition of the Delaunay triangulation across degenerate configurations is controlled by two certificate functions. Let  $u^1, u^2, u^3, u^4, u^5$  be the five trajectories of the points that span two tetrahedra sharing a triangle or three tetrahedra sharing a common edge, as in Figure 1. If one of the trajectories belongs to the infinite vertex then we reorder them such that this trajectory is  $u^5$ . Let  $u^i(t) = (f_1^i(t), f_2^i(t), f_3^i(t))$  be the coordinate functions of the finite points, and recall that the squared norm of this point is the sum of the squares of its three coordinates. If all five points are finite, we create the certificate

$$\det \begin{bmatrix} 1 & f_1^1(t) & f_2^1(t) & f_3^1(t) & \|u^1(t)\|^2 \\ 1 & f_1^2(t) & f_2^2(t) & f_3^2(t) & \|u^2(t)\|^2 \\ 1 & f_1^3(t) & f_2^3(t) & f_3^3(t) & \|u^3(t)\|^2 \\ 1 & f_1^4(t) & f_2^4(t) & f_3^4(t) & \|u^4(t)\|^2 \\ 1 & f_1^5(t) & f_2^5(t) & f_3^5(t) & \|u^5(t)\|^2 \end{bmatrix}, \quad (1)$$

which is a univariate polynomial in  $t$  that is zero iff the five points are co-spherical. Assuming the coordinate functions are linear, the degree of the polynomial is 5. If the fifth point is at infinity, we create the certificate

$$\det \begin{bmatrix} 1 & f_1^1(t) & f_2^1(t) & f_3^1(t) \\ 1 & f_1^2(t) & f_2^2(t) & f_3^2(t) \\ 1 & f_1^3(t) & f_2^3(t) & f_3^3(t) \\ 1 & f_1^4(t) & f_2^4(t) & f_3^4(t) \end{bmatrix}, \quad (2)$$

which is zero iff the four point are coplanar. We call the polynomials in (1,2) *flip certificates* and their roots *flip events*.

After having constructed the initial certificates, at time  $t = 0$ , the algorithm finds the first positive flip event. It then performs a 2-3-flip, creating certificates for the (new) simplices inside the double pyramid, and updating the certificates of the simplices in the boundary of the double pyramid. The updating is necessary because the star of each boundary simplex changes during the flip. After these steps, both data structure and certificates are again valid, and the iteration continues with the next flip event.

**Radius events.** Next, we extend the kinetic algorithm from Delaunay triangulations to alpha complexes. As before, we use a fixed radius  $\alpha_0 > 0$ . We represent the alpha complex by equipping each Delaunay simplex with a flag that indicates whether or not the simplex belongs to the alpha complex. To construct these flags at time  $t = 0$ , we check every Delaunay simplex for being short and for being Gabriel. Following the Short&Gabriel Lemma in Section 2, we add all Delaunay simplices that are short and Gabriel, as well as all their faces, to the alpha complex. To maintain the flags, we construct a certificate for each edge, triangle, and tetrahedron whose roots are the times when the circumradius of the simplex equals  $\alpha_0$ . To simplify the discussion, we assume the generic case in which the circumradius changes from strictly smaller to strictly larger than  $\alpha_0$ , or vice versa. We call these functions *radius certificates* and their roots *radius events*. Whenever a Delaunay simplex is inserted or deleted, the algorithm also creates or removes the corresponding radius certificate. The certificate of an edge compares the length to  $2\alpha_0$ , and taking squares, we get a polynomial of degree 2. The radius certificates of a triangle and a tetrahedron are more complicated, but can be derived from suitable minors of the matrix that defines the circumsphere of the simplex; see [14] for the formula in the tetrahedral case. We will discuss the triangle case in Section 4.

After initializing the alpha complex and the certificates, the algorithm looks for the next event. If this is a flip event, we proceed as described above. In addition, we update the flags that identify the alpha complex as a subcomplex of the Delaunay triangulation. Because all tetrahedra involved in the 2-3-flip have the same circumsphere, they are either all short or all non-short. If they are short, all new Delaunay simplices are added to the alpha complex, and otherwise, none of them is added to the alpha complex. Second, consider the case in which the next event is a radius event. Let  $\sigma$  be the corresponding Delaunay simplex. If  $\sigma$  goes from non-short to short, then its proper cofaces are necessarily non-short. We check whether  $\sigma$  is also Gabriel, noting that this is always the case if  $\sigma$  is a tetrahedron. If so,  $\sigma$  is added to the alpha complex together with all its faces. On the other hand, if  $\sigma$  goes from short to non-short, then its proper faces are necessarily short. We remove  $\sigma$  from the alpha complex, unless it was not in the complex even before the event. If the event causes the deletion of  $\sigma$  from the alpha complex, then this may have consequences for its faces. In particular, if  $\sigma$  was the last proper coface of  $\tau$  in the alpha complex, and  $\tau$  is not Gabriel, then  $\tau$  is also deleted from the alpha complex. Afterwards, the algorithm continues with the next event.

**Redundancy of Gabriel events.** Perhaps surprisingly, flip and radius events suffice to maintain the alpha complex. Flip certificates monitor when tetrahedra become non-Delaunay, and radius events monitor when simplices become short or non-short. We do not need certificates that monitor when simplices become Gabriel or non-Gabriel. To understand why such certificates are not necessary, we call a time  $t$  *G-critical* for a simplex  $\sigma$ , if at that time,  $\sigma$  changes from Gabriel to non-Gabriel, or vice versa.

**Lemma 2 (G-criticality Lemma)** *Let  $t$  be a G-critical time for a short Delaunay edge or triangle. Then this edge or triangle has a proper coface that is in the alpha complex at time  $t$ .*

PROOF. Denote the edge or triangle by  $\sigma$  and consider its circumball,  $B_\sigma$ , at time  $t$ . No point of  $U$  lies in the interior of  $B_\sigma$ , but there is a point  $u$  on the bounding sphere that is not a vertex of  $\sigma$ . The join  $u * \sigma$  is another simplex in the Delaunay triangulation, and it is a proper coface of  $\sigma$ . It has the same circumball as  $\sigma$ , which implies that  $u * \sigma$  is short and Gabriel and therefore belongs to the alpha complex at time  $t$ .  $\square$

The lemma implies that when a short edge or triangle changes its Gabriel status, it is a face of a simplex in the alpha complex. The status change has therefore no impact on its membership in the alpha complex.

**Other events.** We briefly mention the remaining types of events supported by our algorithm. First, we consider a bending event, at which a trajectory starts a new segment. Such an event leaves the alpha complex unchanged, but all flip and radius certificates that involve the coordinates of the corresponding vertex are recomputed. These are the certificates associated to the simplices in the closed star of the vertex.

Second, we allow for insertions and deletions of points. The two operations are mostly symmetric, and we only discuss the insertion of a point  $u$ . We add  $u$  to the Delaunay triangulation by identifying all tetrahedra whose circumballs contain  $u$ , referring to their union as the *conflict region* of  $u$ . Since these tetrahedra no longer satisfy the empty sphere criterion, we remove them from the Delaunay triangulation, together with their faces in the interior of the conflict region. Next, we add  $u$  and connect it to all simplices in the boundary of the conflict region. After the operation, these simplices form the boundary of the closed star of  $u$ . Finally, the simplices in the closed star are checked for being in the alpha complex, and their certificates are created or updated.

## 4 Implementation and Experiments

In this section, we turn to implementation issues. In particular, we discuss how to implement the algorithm in a robust manner, we study the effect of practical choices, and we present experimental results obtained with our software.

**Robust computation.** Recall the basic structure of a kinetic data structure as explained in Section 3: it consists of certificate functions, which are polynomials in  $t$ , and each step advances the state to the smallest root larger than the current time. To maintain the certificates and to advance to the next event, the algorithm computes and compares real roots of univariate polynomials. These roots are algebraic numbers — irrational in general — which makes the computations non-trivial. The naive solution of approximating these roots by inexact floating-point numbers can have unpredictable effects. It is not true that the outcome is just slightly wrong, e.g. by switching the order of events that happen almost simultaneously, but the incorrect order can lead to inconsistent configurations, causing program crashes, non-termination, and non-sensical results. This problem is well-known in geometric contexts [11] and several approaches have been proposed. We follow the *exact geometric computation (EGC) paradigm*, popularized by Chee Yap [16]. It suggests that the basic primitives be mathematically correct, so that an algorithm using these primitives is in the position to compute provably correct results. Translated to our situation, we require that the events of our process are handled in the mathematically correct order. The price we pay for this interpretation of robustness is the burden to compute with algebraic numbers.

We implement our algorithm using the CGAL library<sup>1</sup>, which is designed in the spirit of the EGC paradigm. Another aspect of CGAL is its generic programming approach: algorithms access underlying data structures and primitives through a well-defined interface, so that these layers can be easily replaced with alternative implementations. More specifically, we make use of the kinetic data structures package [13], which provides an EGC implementation of kinetic Delaunay triangulations in two and three dimensions. Internally, the package contains an algebraic kernel, providing the low-level functionality needed to handle roots of polynomials, and a combinatorial layer, maintaining the data structure and the certificates over time. As mentioned earlier, we have extended the combinatorial layer to maintaining an alpha complex.

**Experimental set-up.** We use datasets obtained with the COMPUCCELL3D software<sup>2</sup>, which allows for the simulation of a 3-dimensional cell segregation process using a Monte-Carlo algorithm for energy minimization; see the companion paper [5] for more details. We focus on simulated as opposed to observed data because they offer a better control of the input size and the direct accessibility of the cell trajectories. In our particular example, the cells are colored blue or red, each color with probability one half, and the parameters of the simulation are chosen so that the blue cells eventually engulf the red ones; see Figure 2 for an illustration. We created datasets for several input sizes. In all cases, each trajectory represents the

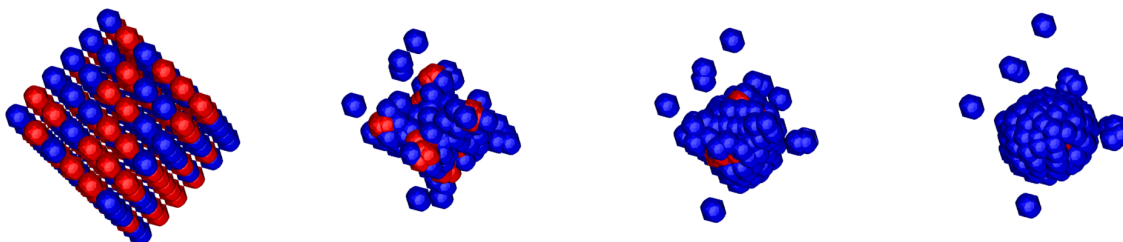


Figure 2: The restricted Voronoi tessellation at four moments in time. At the beginning, the cells form a cubical grid (upper-left). The cells move toward the center of the available space, and the blue cells begin to engulf the red cells (upper-right and lower-left), allowing for satellites while this happens. Finally, the blue cells form a sphere surrounding a ball of red cells (lower-right).

path of a cell nucleus which exists throughout the entire process. Hence, no new cells are ever inserted after the start of the process, and no old cells are deleted before the end of the process. The trajectories follow

<sup>1</sup>Computational Geometry Algorithms Library, [www.cgal.org](http://www.cgal.org).

<sup>2</sup>[www.compuccell3d.org/](http://www.compuccell3d.org/).



a global rhythm in which each trajectory starts a new segment at each value in a common sequence of bending events. In between two bending events, each trajectory is linear. All experiments are performed on a Intel Core 2 Dual CPU clocked with 2.4 GHz each, with 3 MB of cache size, and 4 GB of total memory. The code runs under Debian Squeeze, compiled with gcc-4.4.5 and CGAL version 3.9.

| INPUT #S |       | TIME IN SEC |        | #EVENTS |       |
|----------|-------|-------------|--------|---------|-------|
| traj     | bends | Del         | Alpha  | flips   | rad   |
| 20       | 20    | 7           | 740    | 512     | 631   |
| 20       | 40    | 29          | 1,550  | 1,011   | 1,335 |
| 20       | 80    | 81          | 3,205  | 2,019   | 2,503 |
| 20       | 160   | 188         | 6,473  | 3,978   | 4,506 |
| 10       | 40    | 7           | 487    | 369     | 554   |
| 20       | 40    | 29          | 1,549  | 1,011   | 1,335 |
| 40       | 40    | 79          | 3,975  | 2,874   | 2,171 |
| 80       | 40    | 229         | 9,897  | 7,856   | 4,977 |
| 160      | 40    | 495         | 21,516 | 17,667  | 5,998 |

Table 1: Columns from left to right: the number of trajectories and bending events per trajectory, the time to maintain the Delaunay triangulation and the alpha complex, and the number of flip and radius events.

| INPUT #S |       | TIME IN SEC |       |
|----------|-------|-------------|-------|
| traj     | bends | deg 10      | deg 6 |
| 20       | 20    | 361         | 151   |
| 20       | 40    | 770         | 342   |
| 20       | 80    | 1,579       | 734   |
| 20       | 160   | 3,142       | 1,515 |
| 10       | 40    | 248         | 105   |
| 20       | 40    | 770         | 344   |
| 40       | 40    | 1,892       | 912   |
| 80       | 40    | 4,882       | 2,374 |
| 160      | 40    | 10,181      | 5,157 |

Table 3: Timings for maintaining the alpha complex using a degree 10 versus a degree 6 certificate function for monitoring the circumradii of triangles.

| INPUT #S |       | TIME IN SEC |        | #CERTIFICATES |         |
|----------|-------|-------------|--------|---------------|---------|
| traj     | bends | before      | after  | before        | after   |
| 20       | 20    | 740         | 361    | 20,211        | 10,262  |
| 20       | 40    | 1,550       | 770    | 40,897        | 20,622  |
| 20       | 80    | 3,205       | 1,579  | 82,287        | 41,037  |
| 20       | 160   | 6,473       | 3,142  | 163,511       | 80,489  |
| 10       | 40    | 487         | 248    | 12,932        | 6,324   |
| 20       | 40    | 1,549       | 770    | 40,897        | 20,622  |
| 40       | 40    | 3,975       | 1,892  | 105,754       | 54,426  |
| 80       | 40    | 9,897       | 4,882  | 259,848       | 139,816 |
| 160      | 40    | 21,516      | 10,181 | 566,589       | 303,065 |

Table 2: Third and fourth columns: the time to maintain the alpha complex before and after Optimization 1. Fifth and sixth columns: the number of radius certificates before and after Optimization 1.

| INPUT #S |       | TIME IN SEC |       |        |       |
|----------|-------|-------------|-------|--------|-------|
| traj     | bends | kds         | ak_d  | filter | cache |
| 20       | 20    | 151         | 84    | 72     | 47    |
| 20       | 40    | 342         | 176   | 152    | 98    |
| 20       | 80    | 734         | 364   | 310    | 198   |
| 20       | 160   | 1,515       | 731   | 622    | 390   |
| 10       | 40    | 105         | 55    | 49     | 30    |
| 20       | 40    | 344         | 177   | 152    | 100   |
| 40       | 40    | 912         | 458   | 392    | 256   |
| 80       | 40    | 2,374       | 1,180 | 1,024  | 689   |
| 160      | 40    | 5,157       | 2,566 | 2,256  | 1,481 |

Table 4: Timings for maintaining an alpha complex using the kds kernel, the ak\_d kernel, the latter with Descartes filtering, and in addition with enabled cache.

In a first test, we compare the running times for maintaining the Delaunay triangulation and the alpha complex, see Table 1. Comparing the third and fourth columns of the table, we see that the radius events slow down the algorithm by more than a magnitude, in spite of the fact that their number is not much larger than the number of flip events. In the remainder of this section, we explain improvements of our algorithm aimed at reducing the performance gap between Delaunay and alpha complexes.

**Number of certificates.** The bottleneck is the construction of radius certificates and the computation of their real roots. Recall that in our original formulation, we maintain a radius certificate for each edge, triangle, and tetrahedron. Our first optimization is based on the observation that many of these certificates are not necessary: if a simplex is short, then all its faces are short, and if a simplex is non-short, then all its cofaces are non-short.

**Optimization 1** *Whenever a triangle or tetrahedron becomes short, we remove the radius certificates of its proper faces, and when an edge or triangle becomes non-short, we remove the radius certificates of its proper cofaces.*

Of course, this implies that we sometimes have to construct certificates that would otherwise still exist. For example, we construct the certificate of a triangle at the time its third edge becomes short. On the other hand, we avoid unnecessary certificates, for instance the certificates of the boundary edges of a triangle that stays short for the whole simulation. As we see in Table 2, the strategy saves time in practice. We observe that the constructions of radius certificates and the running time both decrease roughly by a factor of two.

**Degree.** We turn to the computation of certificates. Assuming piecewise-linear trajectories, the radius certificate of an edge is a polynomial of degree 2; compare with Section 3. There is a standard construction of a radius certificate of a tetrahedron, which is a polynomial of degree 8; see [14]. Our interest lies in the remaining triangle case. The current CGAL implementation computes the squared circumradius of a triangle  $\Delta$  in  $\mathbb{R}^3$  with an expression of the form

$$r_{\Delta}^2 = \frac{\text{num}_x^2 + \text{num}_y^2 + \text{num}_z^2}{4\text{den}^2}, \quad (3)$$

where  $\text{den}$  is the determinant of the matrix in (2), and  $\text{num}_x$ ,  $\text{num}_y$ ,  $\text{num}_z$  are expressions formed by minors of this matrix. The corresponding certificate,

$$\text{num}_x^2 + \text{num}_y^2 + \text{num}_z^2 - 4\alpha_0^2\text{den}^2,$$

is a polynomial whose degree is 10, which is higher than the degree for the tetrahedron. We replace (3) by a simpler expression. Writing  $u$ ,  $v$ ,  $w$  for the three vertices of the triangle, the circumradius can also be written as

$$r_{\Delta} = \frac{\|u - v\| \cdot \|u - w\| \cdot \|v - w\|}{2\|(u - w) \times (v - w)\|}, \quad (4)$$

a formula that is straightforward to derive using elementary matrix calculus; see also Wikipedia [15].

**Optimization 2** *Monitor the radius of a triangle using the following certificate function:*

$$\|u - w\|^2\|u - v\|^2\|v - w\|^2 - 4\alpha_0^2\|(u - w) \times (v - w)\|^2.$$

The degree of this certificate is 6. We see the effect of this improvement in Table 3. The running time improves by more than a factor of two.

**Algebraic kernel.** As already mentioned, the CGAL package for kinetic data structures contains an internal algebraic kernel, which, among other things, is used to isolate the roots of polynomials and sort them in the event queue. By the generic design of the package, the combinatorial layer communicates with the kernel via a small and well-defined interface, which makes it possible to replace the algebraic kernel with a different implementation.

In recent years, a mature and generic algebraic kernel for geometric computations has been developed [1]. It has been integrated into CGAL and is available since version 3.7 under the name `Algebraic_kernel_d`. We refer to it as the `ak_d` kernel. Both the internal `kds` and the `ak_d` kernels use subdivision methods for root isolation, but they differ in the strategy for detecting empty intervals and isolating intervals. The `kds` kernel uses Sturm theory [17, §7], while the `ak_d` kernel is based on Descartes' rule of sign [7], which leads to a better performance in practice; see [9] for a comparison of various root solvers. The difference between the third and fourth columns in Table 4 shows that exchanging the kernel yields another improvement of roughly a factor of two.

**Filter and cache.** We get further optimizations by exploiting the special structure of our experimental setup. For any certificate, we are only interested in the roots between the current time and the next bending event, when the certificate becomes invalid. Many certificates do not have roots in this interval, but may have roots outside. The current implementation first computes all real roots and thereafter discards the ones that lie outside the mentioned interval.

**Optimization 3** *We use Descartes' rule of sign to certify the non-existence of roots in the interval until the next bending event, and if successful, we skip the root isolation algorithm.*

The fifth column of Table 4 shows the improvement. More than ninety percent of the certificates that do not have a root before the next bending event are filtered out. As a final improvement, we avoid isolating the roots of the same polynomial multiple times.

**Optimization 4** *We store polynomials together with their real roots in the interval until the next bending event in cache, which is cleared at the next bending event.*

We see in the sixth column of Table 4 that the cache yields another substantial speed-up, which suggests that certificates are frequently devalidated and revalidated during the runtime of the algorithm. We remark that also the `kds` kernel would benefit from caching. Comparing the running times for maintaining the alpha complex before and after the four steps of optimization, we see that the performance improves by roughly a factor of 15. Moreover, compared to maintaining the Delaunay triangulation, the optimized algorithm is slower by a factor up to 4. It is no surprise that the extension to alpha complexes is expensive. After all, it requires additional radius certificates, which have higher degrees than the flip certificates needed to maintain the Delaunay triangulation. We have demonstrated that with some algorithmic engineering, the overhead needed for alpha complexes can be kept within a moderate bound.

## 5 Discussion

The main contributions of this paper are a kinetic algorithm for alpha complexes and its implementation with exact arithmetic. There is no theoretical obstacle to generalizing our algorithm and its implementation to the weighted case, in which different Voronoi regions are restricted to within different size balls. A more challenging problem is the extension to bodies different than balls, e.g. arbitrarily oriented ellipsoids.

We took a considerable effort to accelerate the implementation of the kinetic alpha complex algorithm, since this was necessary to compute examples of reasonable size in acceptable time; the instances computed in [5] each took about 4 hours with our best configuration. Nevertheless, there are opportunities to further speed up the software, in particular on the level of the algebraic kernel. For example, it would be desirable to restrict the root isolation method to within a given interval, without wasting any time on roots outside this interval. We believe that kinetic data structures are an important tool in the topological analysis of time-varying shapes. We hope that our work on cell segregation initiates further work on such data. To facilitate this research, it would be useful if our extension of CGAL's package on kinetic data structures is transformed from an experimental branch to a redesign of the package. It is desirable that such a redesign solves the problem of degeneracies in the implementation of kinetic Delaunay triangulations and alpha complexes. Except for some special cases, the current versions of both algorithms are not guaranteed to work correctly when two or more events happen at exactly the same time.

## Acknowledgments

The authors thank Viktoriia Sharmanska for discussions and help with a 2-dimensional prototype of our implementation.

## References

- [1] E. BERBERICH, M. HEMMER AND M. KERBER. A generic algebraic kernel for non-linear geometric applications. In “Proc. 27th Ann. Sympos. Comput. Geom., 2010”, 179–186.
- [2] M. DE BERG, O. CHEONG, M. VAN KREFELD AND M. OVERMARS. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2008.
- [3] H. EDELSBRUNNER. *Geometry and Topology for Mesh Generation*. Cambridge Univ. Press, Cambridge, England, 2001.
- [4] H. EDELSBRUNNER AND J. L. HARER. *Computational Topology. An Introduction*. Amer. Math. Soc., Providence, Rhode Island, 2010.
- [5] H. EDELSBRUNNER, C.-P. HEISENBERG, M. KERBER AND G. KRENS. *The medusa of spatial sorting: topological construction*. arxiv:1207.6474, 2012.
- [6] H. EDELSBRUNNER AND E. MÜCKE. Three-dimensional alpha shapes. *ACM Trans. Graphics* **13** (1994), 43–72.
- [7] A. EIGENWILLIG, V. SHARMA AND C. YAP. Almost tight recursion tree bounds for the Descartes method. In “Internat. Sympos. Symbol. Alg. Comput., 2006”, 71–78.
- [8] C.-P. HEISENBERG AND G. KRENS. Cell sorting in development. *Current Topics in Developmental Biology*, to appear.
- [9] M. HEMMER, E. TSIGARIDAS, Z. ZAFEIRAKOPOULOS, I. EMIRIS, M. KARAVELAS AND B. MOURRAIN. Experimental evaluation and cross-benchmarking of univariate real solvers. In “Proc. Conf. Symbol. Numeric Comput., 2009”, 45–54.
- [10] M. KERBER AND H. EDELSBRUNNER. *The medusa of spatial sorting: 3D kinetic alpha complexes and implementation*. Manuscript, 2012. Available at arxiv.org
- [11] L. KETTNER, K. MEHLHORN, S. PION, S. SCHIRRA AND C. YAP. Classroom examples of robustness problems in geometric computations. *Computational Geometry: Theory and Applications* **40** (2008), 61–78.
- [12] D. RUSSEL. *Kinetic Data Structures in Practice*. PhD Thesis, Stanford University, California, 2007.
- [13] D. RUSSEL. Kinetic Data Structures. *CGAL User and Reference Manual*, edition 4.0, 2012.
- [14] E. WEISSTEIN. Circumsphere. [mathworld.wolfram.com/Circumsphere.html](http://mathworld.wolfram.com/Circumsphere.html), accessed, 2011.
- [15] WIKIPEDIA, THE FREE ENCYCLOPEDIA. Circumscribed sphere. [en.wikipedia.org/wiki/Circumscribed\\_circle#Circumcircle\\_equations](http://en.wikipedia.org/wiki/Circumscribed_circle#Circumcircle_equations), accessed September 11, 2012.
- [16] C. YAP. Towards exact geometric computation. *Computational Geometry: Theory and Applications* **7** (1997), 3–23.
- [17] C. YAP. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, Oxford, England, 2000.