Constructing the Exact Voronoi Diagram of Arbitrary Lines in Three-Dimensional Space[*] with Fast Point-Location

Michael Hemmer[1], Ophir Setter[2], and Dan Halperin[2]

[1] INRIA - Sophia Antipolis, France
Michael.Hemmer@sophia.inria.fr
[2] Tel-Aviv University, Israel
{ophirset,danha}@post.tau.ac.il

Abstract. We introduce a new, efficient, and complete algorithm, and its exact implementation, to compute the Voronoi diagram of lines in space. This is a major milestone towards the robust construction of the Voronoi diagram of polyhedra. As we follow the exact geometric-computation paradigm, it is guaranteed that we always compute the mathematically correct result. The algorithm is complete in the sense that it can handle all configurations, in particular all degenerate ones. The algorithm requires $O(n^{3+\varepsilon})$ time and space, where n is the number of lines. The Voronoi diagram is represented by a data structure that permits answering point-location queries in $O(\log^2 n)$ expected time. The implementation employs the CGAL packages for constructing arrangements and lower envelopes together with advanced algebraic tools.

Keywords: Voronoi Diagrams, Point Location, Lower Envelopes, Robust Geometric Computing, Computational Geometry, CGAL.

1 Introduction

The Voronoi diagram (VD) is among the most fundamental structures in Computational Geometry, and is known to be a useful tool in a variety of domains. For instance, structural biology [19], [34] and robot motion planing [25], [35] apply Voronoi diagrams to encode point sets keeping maximal distance from atoms or obstacles, respectively. A related concept is the medial-axis transform [6], which is considered fundamental in solid modeling and applied to problems such as finite element meshing, shape morphing, and feature recognition. Yet, the adaptation of complex three-dimensional Voronoi diagrams in professional tools has been very slow. Their use is hindered by the difficulty of designing and implementing reliable geometric algorithms for complex structures in three-dimensional space.

Voronoi diagrams have been the subject of a tremendous amount of research. We refer the reader to the survey by Aurenhammer and Klein [2] of work

[*] This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

published up till 2000. Voronoi diagrams in $\mathbb{R}^2$ are well understood in almost all aspects, that is, in terms of complexity and optimal algorithms as well as in terms of robust and efficient implementations. In $\mathbb{R}^3$ much less is known, even for simple objects such as lines, segments, or polyhedra. For example, a tight bound on the combinatorial complexity of the *VD* of n lines or line segments in $\mathbb{R}^3$ is unknown; it is conjectured that the complexity is near-quadratic; the known lower bound is $\Omega(n^2)$ [1], but the best known upper bound is[1] $O(n^{3+\epsilon})$ [32]. In the case of lines with a fixed number c of orientations the upper bound was improved to $O(c^4 n^{2+\varepsilon})$ [26]. A complete analysis of all possible combinatorial cases for three arbitrary lines is presented by Everett *et al.* [16], [17].

Today, there are many published results on robust constructions of different types of Voronoi diagrams in $\mathbb{R}^2$. Not only Voronoi diagrams of points are considered, but also Voronoi diagrams of line segments [23], circles [15], ellipses [14], and more [8, §2], [31]. In $\mathbb{R}^3$, an exact implementation of the Voronoi diagram of additively-weighted points was analyzed in [7], but we are not aware of any exact, complete, and implemented algorithm that computes Voronoi diagrams of lines, line segments, or polyhedra. Nevertheless, progress has been made toward the exact computation of the arrangement of quadrics [5], [12]. Each Voronoi cell of the diagram of lines in space can be represented as the union of cells of such an arrangement. Other approaches explicitly aim for an exact or robust computation of the Voronoi diagram (or the medial axis) [9], [27]. However, those approaches are not complete. For example, Culver's algorithm [9] does not handle singular trisector-curves.

Finally, Hanniel and Elber [20] provided an algorithm to construct the Voronoi cell of bounded planes, spheres, and cylinders in $\mathbb{R}^3$. Similar to ours, the approach utilizes lower envelopes but leaves robustness issues aside, since the use of parametrized intersection curves hinders an efficient implementation using exact arithmetic. Moreover, it does not consider point location.

We present an exact and complete (and thus robust) algorithm for computing the Voronoi diagram of arbitrary lines in three dimensions with respect to the Euclidean metric. The algorithm requires $O(n^{3+\varepsilon})$ time and space, where n is the number of input lines. The data structure admits answering of point-location queries in $O(\log^2 n)$ time. We believe that the nature of the algorithm and the general approach of its implementation constitute a major milestone towards an exact and robust construction of the Voronoi diagram of polyhedra in $\mathbb{R}^3$.

We utilize the fact that in Euclidean space the boundary of Voronoi cell (*VC*) can be considered as a lower envelope since the cell essentially has a certain "star shapedness" property: For any point p inside the Voronoi cell of a specific line site ℓ, the line segment connecting p to its projection p_ℓ onto ℓ, is fully contained in the cell. This observation enables us to represent the Voronoi cell of ℓ as a minimization diagram, which is (conceptually) embedded on an infinitesimally small cylinder around ℓ. This observation is similar to the well-known connection

[1] A bound of the form $O(f(n) \cdot n^{\varepsilon})$ means that the actual upper bound is $C_{\varepsilon} f(n) \cdot n^{\varepsilon}$, for any $\varepsilon > 0$, where C_{ε} is a constant that depends on ε, and generally tends to infinity as ε goes to 0.

between Voronoi diagrams and lower envelopes [13]. Lower dimensional cells are represented several times, namely as part of the boundary of the *VC* of each line they are associated with. The implementation is developed in and based on CGAL, the Computational Geometry Algorithms Library.[2]

The paper is organized as follows. Section 2 discusses preliminary subjects, such as properties of bisectors and trisectors of lines in space and the lower envelope algorithm. Section 3 describes the details of the construction of a Voronoi cell. Section 4 discusses the point location algorithm and its analysis. Section 5 gives implementation details and presents preliminary experimental results that were obtained with our software.

2 Preliminaries

Let $\mathcal{O} = \{s_1, s_2, \ldots, s_n\}$ be a set of objects in $\mathbb{R}^d$, also referred to as sites. We follow the Voronoi diagram definition by Everett *et al.* [17]: The *Voronoi diagram* $VD(\mathcal{O})$ is the subdivision of $\mathbb{R}^d$ into cells, where each cell $VC(S)$ is associated with a subset $S \subseteq \mathcal{O}$, such that every point in $VC(S)$ is strictly closer to all sites in S than to all other sites in $\mathcal{O}$ and is equidistant from all sites in S. The formal definition is:

$$VC(S) = \left\{ p \in \mathbb{R}^d \;\middle|\; \begin{array}{l} \forall s \in S, t \in \mathcal{O} \setminus S : d(p,s) < d(p,t) \\ \forall s,t \in S : d(p,s) = d(p,t) \end{array} \right\}$$

In the context of this paper, $\mathcal{O}$ denotes a set of arbitrary rational lines in $\mathbb{R}^3$ and $d(\cdot,\cdot)$ denotes the Euclidean distance function. The set of points that is of equal distance to two or three sites is called a bisector or trisector, respectively.

2.1 Properties of Bisectors and Trisectors

We next state some properties of bisectors and trisectors of the Voronoi diagram of lines in $\mathbb{R}^3$ that are used throughout this paper. Proposition 1 gives properties of bisectors; see Figure 1 for illustrations.

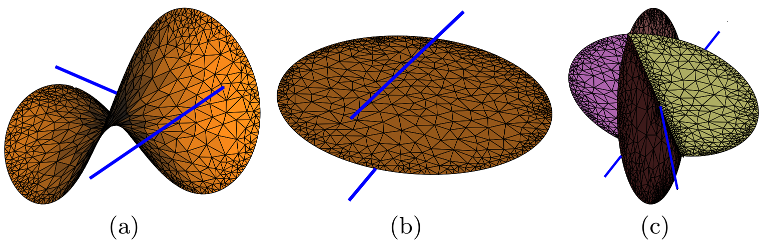


Fig. 1. Bisector of: (a) two generic lines; (b) two parallel lines; (c) two intersecting lines. The diagrams were created with our implementation (see Section 5), and were clipped by a sphere for convenience.

[2] http://www.cgal.org

Proposition 1. *The bisector of two lines ℓ_1 and ℓ_2 in three-dimensional space is either (a) a hyperbolic paraboloid (a surface of algebraic degree 2), if ℓ_1 and ℓ_2 are skew, (b) a plane, if ℓ_1 and ℓ_2 are parallel, or (c) a pair of orthogonal planes, if ℓ_1 and ℓ_2 are concurrent. In the latter case, the singular locus of the bisector is a line that perpendicularly intersects ℓ_1 and ℓ_2 in their intersection.*

The main theorem of Everett *et al.* [16] provides a good overview of the different cases of the trisector:

Theorem 1 (Everett et al.). *The trisector of three lines is either (i) a non-singular quartic, if the three lines are pairwise skew but not all parallel to a common plane nor lie on the surface of a hyperboloid of revolution, (ii) a cubic and a line that do not intersect, if the three lines are pairwise skew and lie on the surface of a hyperboloid of revolution, (iii) a nodal quartic, if the three lines are pairwise skew and all parallel to a common plane, (iv) one parabola or hyperbola, if there is exactly one pair of coplanar lines which are parallel, (v) two parabolas or hyperbolas that intersect, if there is exactly one pair of coplanar lines that intersect, (vi) between 0 and 4 lines, if there are two pairs of coplanar lines, or (vii) one line, in the case of three coplanar concurrent lines, the common singular locus of the bisectors.*

We use a corollary of the above theorem in Section 3, where we describe the construction of a Voronoi cell in the diagram of lines.

2.2 Lower Envelope Algorithm

Again, we regard the boundary of each three-dimensional *VC* as a lower envelope with respect to its line site ℓ_0. This lower envelope is represented as a minimization diagram which is conceptually embedded in the uv-parameter space of the surface of an infinitesimally small cylinder around ℓ_0.[3] We utilize the divide-and-conquer algorithm for constructing lower envelopes [1] as it is implemented in CGAL [33, §8.5], which we briefly describe next.

Since the algorithm projects bisectors into the parameter space, all bisectors are initially split up into uv-monotone surfaces. The algorithm then splits the resulting set $\mathcal{G}$ into two subsets $\mathcal{G}_1$ and $\mathcal{G}_2$ of roughly equal size, and recursively computes their minimization diagrams $\mathcal{M}_1$ and $\mathcal{M}_2$. In the conquer step, the two diagrams are merged into one. First, the overlay of $\mathcal{M}_1$ and $\mathcal{M}_2$ is computed, where each feature is labeled with up to two sets of labels L_1 and L_2 of candidate surfaces from both diagrams. Thereafter, the arrangement is further refined such that each feature can either be labeled with L_1, L_2, or $L_1 \cup L_2$. In particular, each face that is labeled with two bisectors is refined by the corresponding projected trisector curve. Note that this step can also split up edges. After the comparison of bisectors the algorithm removes redundant edges and vertices, which yields the final diagram. The complexity of the above algorithm is $O(n^{2+\varepsilon})$, with the condition that the bisector surfaces are "well-behaved".

Note that the algorithm heavily relies on arrangement operations such as overlay, which are provided by [33, §8.1] and [4]. Though, we treat these as a black

[3] See Section 3 for details on the uv-parameter space setting.

box throughout most of the paper, some details can be found in Section 5. The additional constructions and predicates required by the lower envelope algorithm are: the construction of the projected boundary of uv-monotone surfaces, the construction of the projected intersection of two uv-monotone surfaces, and the comparison of two bisectors above a face, an edge, or a vertex.

3 Computing a Voronoi Cell

This section discusses the computation of the *VC* of one line, referred to as the base line and denoted by ℓ_0.

CGAL's arrangement package has the infrastructure to compute envelopes over cylinders. However, for the efficiency of the implementation it is important to keep the algebraic degree of the projected curves as low as possible. Therefore, we project the curves on two parallel planes that "sandwich" the base line, while keeping the projection direction normal to the cylinder. This reduces the maximum degree of a projected trisector curve from sixteen down to eight.

3.1 Parametrization and Projection

Let $F = \{\overrightarrow{b_1}, \overrightarrow{b_2}, \overrightarrow{b_3}\}$ be an orthogonal basis of $\mathbb{R}^3$ which is chosen such that $\overrightarrow{b_1}$ is the direction of the base line ℓ_0. Moreover, let p_0 be some rational point on ℓ_0. Now, consider the parametrization $\mathcal{X}(u, v, r) = p_0 + u \cdot \overrightarrow{b_1} + v \cdot r \cdot \overrightarrow{b_2} + r \cdot \overrightarrow{b_3}$. $\mathcal{X}(u, v, \pm 1)$ defines two parallel planes (uv-planes) that sandwich ℓ_0. A point $\mathcal{X}(u_0, v_0, \pm 1)$ represents a ray that originates from point $p_0 + u_0 \cdot \overrightarrow{b_1}$ on ℓ_0 with direction $\pm(v_0 \cdot \overrightarrow{b_2} + \overrightarrow{b_3})$. Projecting along these rays onto $\mathcal{X}(u, v, \pm 1)$, we denote $\mathcal{X}(u, v, \pm 1)$ as the positive and the negative projection plane, respectively.

Note that the plane $H^* = \{x \in \mathbb{R}^3 | (x - p_0)^T \cdot \overrightarrow{b_3} = 0\}$ is not covered by the parametrization. But it is straightforward to glue the two minimization diagrams on the two planes together as long as the chosen frame F is *generic*, that is, curves are not allowed to touch H^*, intersect in H^*, or even be contained in H^*. However, curves are of course allowed to transversely intersect H^*, each intersection giving rise to a single vertical asymptote on each projection plane.

In order to avoid these critical cases, we generate the local frame by setting $\overrightarrow{b_2}$ to some random vector that is orthogonal to $\overrightarrow{b_1}$. Though this frame is generic with high probability, we also check in all relevant predicates that the frame is indeed generic. If necessary, we restart the computation choosing another random frame. We chose the standard strategy that increases the number of random bits used for each iteration. This way we guarantee termination and a small number of additional bits due to the randomization.

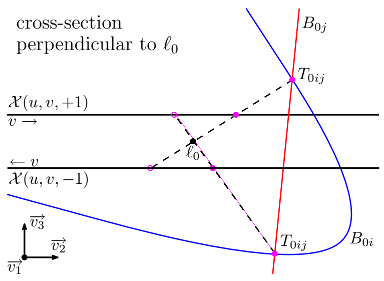
We highlight below several major issues in the projection of a trisector. The projection of a bisectors' boundary and a detailed case analysis is not addressed due to lack of space. We rely merely on the generic frame and on the following corollary that directly follows from Theorem 1:

Corollary 1. *The set of points where the trisector does not represent a transversal intersection of the bisectors is a 0-dimensional set, namely, the singular* *points of the trisector. The only exception is the case of three coplanar concurrent lines; in this case the trisector is the common singular locus (line) of the three bisectors.*

For a trisector T_{0ij} let $B_{0i}, B_{0j}, B_{ij} \in \mathbb{Q}[x_1, x_2, x_3]$ be the three trivariate polynomials of the relevant bisectors. Now let B_1 and B_2 be the two bisectors of minimal degree, d_1 and d_2, respectively. The projection is carried out by a resultant computation [18]. Since we wish to project towards ℓ_0 we first substitute $\mathcal{X}(u, v, r)$ into B_1 and B_2 and compute the resultant with respect to r.

$$res(u, v) := \text{resultant}(B_1(\mathcal{X}(u, v, r), B_2(\mathcal{X}(u, v, r)), r) \in \mathbb{Q}[u, v] \; .$$

This is at most a bivariate polynomial of degree $2d_1d_2$. Thus, in the worst case (the generic case) this is an irreducible polynomial of degree[4] only 8. However, due to its algebraic nature the approach can not immediately distinguish between the parameter spaces of the positive and the negative planes. The Figure to the right illustrates how the resultant projects T_{0ij} into the positive and negative plane. We first split up the projected curve into u-monotone arcs using [4]. In particular, curves are split up at vertical asymptotes. In order to decide that an arc α is on a certain plain we utilize Corollary 1, namely the observation that in all but one exception (which is handled explicitly) two bisectors must intersect transversely along the trisector curve, which implies that B_{0i} and B_{0j} must interchange their order while passing the projected trisector.



This is detected by two ray shoots at rational points right above and below α. Let $\overline{p}$ and $\underline{p}$ be these two points, respectively. To ensure that both points are chosen sufficiently close, we construct a rational vertical line L that intersects α in its interior, say at point p_α. We choose the points on L such that they isolate the arc from all other intersections of L with res. Now consider the path on L from $\overline{p}$ (or $\underline{p}$) to p_α. $\overline{p}$ is sufficiently close to α since this path does not intersect res until it reaches α. In case α is vertical, we choose L to be horizontal.

3.2 Lower Envelope Predicates

A core part of the envelope algorithm is the representation of minimization diagrams as labeled arrangements and the overlay of such arrangements. The required constructions and predicates for these operations relate to planar algebraic curves only, which are provided by [4]. However, it remains to ensure that no intersection takes place in H^*. This boils down to testing that the leading

[4] More precisely, it is a bivariate polynomial of bi-degree at most $(4, 4)$. For a standard rational parametrization of the cylinder, we would obtain a polynomial of bi-degree $(8, 8)$ or 16 in total.

coefficients with respect to v of two non overlapping (co-prime) curves have no common root. Thus, we provide a slightly modified set of operations that ensure this condition in addition.

The remaining predicates that are required by the envelope algorithm are the comparison of two bisectors above a face, an edge, or a vertex, respectively; see also Section 2.2. For a vertex, which may not have rational coordinates, we first check whether the point is on the projected intersection of the two bisectors and report equality if it is indeed the case. Otherwise it is sufficient to compute a rational point that is close enough to the vertex, and to compare the surfaces along the corresponding ray. In order to compare above an edge we construct a vertex in its interior and compare at that vertex as described above. For a face, it is sufficient to compare (again via ray-shooting) at a rational point in its interior. In each case, the rational point is constructed using strategies similar to the one discussed at the end of Section 3.1.

3.3 Complexity

For the time and space complexity analysis we ignore additional costs that may arise due to variable bit-length of various implementations adhering to the exact computation paradigm [36]. We also ignore the additional run-time that can result from a poor choice of a generic frame (Section 3.1), as it is not the general case, and has no impact on performance in expectation.

The bisector surfaces are algebraic and thus comply with the definition of "well-behaved" surfaces required in [32]. Thus, the time-complexity of the lower envelope algorithm is $O(n^{2+\varepsilon})$ (which is also the best known upper bound). Overall, the run-time complexity of computing the cells for all n lines is $O(n^{3+\varepsilon})$, which also bounds the space complexity.

4 Fast Point Location

Given a query point q we wish to find the closest line to it. Consider the following point-location strategy: We start with a random line site ℓ. First we project q on ℓ and locate its image in the minimization diagram of ℓ. The image is located on a feature of the minimization diagram which is labeled with a (in general not empty) set of line sites $\mathcal{S}$. We then compare the distance $d(\ell, q)$ to $d(\ell', q)$ for one line $\ell' \in \mathcal{S}$. If $d(\ell, q)$ is less than or equal to $d(\ell', q)$ we report ℓ or $\mathcal{S} \cup \ell$, respectively. Otherwise we continue in the cell of ℓ'. This walk through the Voronoi diagram terminates since there is only a finite number of cells and the distance of q to the current line always decreases. We can locate the image of q inside the minimization diagram in expected $O(\log n)$ time by using point-location that is based on trapezoidal decomposition [28]. Combining this algorithm with the idea of landmarks [21] may already have good performance in practice. However, the algorithm has a worst-case time complexity $O(n \log n)$.

We turn it into an algorithm with a time-complexity $O(\log^2 n)$ by combining it with a strategy that is similar to skip lists. We build a hierarchy of Voronoi diagrams. The lowest layer contains the VD of the full set of lines. The lines for the other layers are from the previous layer, each chosen with probability $1/k$, where $k > 1$ is constant. The highest layer (the root layer) contains only a constant number of lines ($\leq k$). The expected number of layers is $O(\log n)$. In order to locate a point q we first locate it in the root layer using the walk strategy described above. We then proceed to the next layer starting at the line that was found in the preceding layer.

We remark that some special cases are left out in this discussion for brevity (e.g., query points in H^*), but they are completely handled in our software. The following theorem summarizes the performance of the point-location structure (see [10] and [24] for a similar analysis in 2D):

Theorem 2. *For any query point q the expected running time of the point-location query in the hierarchical VD structure is $O(\log^2 n)$.*

Proof. The number of cells visited at the root layer is obviously at most k. For all other layers, consider the the path backward, from its target to the source: For every cell the probability that it is already the source is $1/k$. Thus, the expected length of a path is $\sum_{i=1}^{n} \frac{i}{k}(\frac{k-1}{k})^{i-1} \leq k$.

That is, the expected running-time is $k \sum_{i=1}^{\log_k n} T(k^i)$, where $T(m)$ is the expected time spent on the point location in the minimization diagram of m lines. Thus we obtain an expected running-time of $O(\log^2 n)$ in total.

5 Implementation Details

Our implementation is based on CGAL, which follows the generic-programming paradigm [3]. Algorithms are formulated and implemented such that they are abstract from the actual types, constructions, and predicates. Thus, the implementation of every algorithm and data structure in CGAL is parametrized by a so-called traits class [29], in which these functionalities are defined. In particular, users can employ an algorithm with their own types, constructions, and predicates by providing their own traits class. This way it is possible to achieve a great amount of flexibility. At the extreme, it is possible to even partially change the nature of an algorithm, as we do here for the three-dimensional lower envelope class [33, §8.5].

The core of our implementation is the traits class for the lower envelope algorithm, which also needs to be a valid traits class for CGAL's arrangement package. The required functionalities by the arrangement package are provided by the traits class presented in [4]. The approach reduces all construction and predicates to cylindrical algebraic decompositions of the plane for one or two curves. We essentially wrap this traits class and add the auxiliary functionalities required by the envelope algorithm; see also Section 3. In case we detect that the current frame is not generic an exception is thrown, which is then caught by our primary class that computes a new frame and restarts the computation

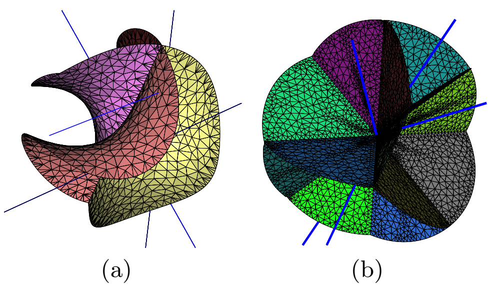


Fig. 2. Degenerate Voronoi diagrams of lines. The diagrams are clipped by a sphere for convenience. (a) *VD* of 4 lines, obtained by rotating one line around the z-axis. All bisectors meet in that axis. (b) *VD* of 4 lines intersecting in one point. (c) *VD* of 4 lines, two lines intersect and the others are parallel to each of them, respectively.

of the cell. For each Voronoi cell we keep a separate instance of the traits class, which is used for both planes. This allows caching of relevant results.

Approximation of the three-dimensional coordinates of a vertex, is based on multi-precision floating-point interval arithmetic (MPFI) [8, §8]. Since this is a certified approximation, we obtain a bounding box that contains the vertex. This could be used to easily establish the adjacency among lower dimensional cells. For instance, let v denote a vertex in a minimization diagram $\mathcal{M}$. The label of v points to all other minimization diagrams that contain a representation of it. Let $\mathcal{M}'$ be one of these diagrams and v' be the representation of v that we wish to find therein. We could use a similar approach to the one used in [12]: By using the labels, we identify all possible candidates in $\mathcal{M}'$. Since the bisector surfaces are at most of algebraic degree two, this set contains only up to 8 representations and contains at least v'. We progressively compute more precise bounding boxes for all candidates until only one (the one of v') overlaps the bounding box of v.

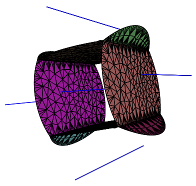
Our implementation can handle arbitrary rational lines, in particular, it can handle all possible degenerate cases. Figure 2 depicts degenerate Voronoi diagrams. Each mesh was generated using CGAL's package for labeled mesh domains [30]. The oracle, which is required by the mesh generation, was written such that it only utilizes (and thereby tests) our point location structure. We used `medit` [22] for the final visualization.

Since we aim to eventually incorporate our code into a CGAL package the software is developed within the revision control system of the project. All experiments within this section where carried out on an internal CGAL release `CGAL-3.7-Ic-27`, which already comprises all the necessary algebraic tools [4]. However, the trapezoidal map is currently not available for minimization diagrams due to ongoing changes in the arrangement package (it is anticipated soon), which forces us to resort to simpler point location strategies for now.

Finally, we present preliminary results obtained with our software. The point location structure as it is discussed in Section 4 leaves the ratio k among levels undetermined. In order to show the impact of k we created random instance of

Table 1. Average number of visited cells per query, where k denotes the ratio of the hierarchy and N the number of lines. Cases where the hierarchy would only consist of one level are marked with 'n/a', the times would correspond to those in the last column. To the right is depicted a Voronoi diagram of 5 parallel lines.

$N \backslash k$	2	4	8	12	16	20	24	28	$+\infty$
16	6.48	4.30	4.34	n/a	n/a	n/a	n/a	n/a	**3.94**
36	8.09	6.33	**5.33**	6.23	5.67	n/a	n/a	n/a	**5.62**
64	9.77	6.42	**5.73**	7.34	6.07	6.00	6.12	6.83	6.63
100	9.87	7.22	**6.18**	7.10	6.45	6.97	6.83	7.13	7.43
144	11.56	8.14	8.47	**7.46**	7.81	9.64	9.75	8.68	12.72



parallel lines[5] with coefficients in the range $[0, 2^{10}]$. For each instance, we created 10 Voronoi diagram hierarchies, which where queried with 1000 random points in $[0, 2^{10}]^3$ each.

Table 1 shows the average number of visited cells per query depending on the number of lines and the chosen value for k. The last column shows the pure walk without a hierarchy, which suggests (for the case of parallel lines) an average query time in $O(\sqrt{n})$, as one may also expect due to results in [11]. For larger instances, it seems that choosing k between 8 and 12 is appropriate.

6 Conclusions

We have presented an exact, complete, and thus robust, algorithm that computes the Voronoi diagram of arbitrary rational lines in $\mathbb{R}^3$. The algorithm requires $O(n^{3+\varepsilon})$ time and space, where n is the number of lines. The introduced data structure admits answering point-location queries in $O(\log^2 n)$ expected time. The implemented prototype is exact and can handle all degenerate cases.[6]

The algorithm is intentionally designed such that it avoids tedious case distinctions, which makes it implementable, maintainable and, in particular, extensible to other primitives such as points, line segments, and triangles. Thus, we consider our approach as a major milestone towards the exact computation of the Voronoi diagram of polyhedra in three dimensions.

The approach may also be generalized to spheres (see also [20]) which would open the door for innovative solutions to central problems in Structural Biology [25], [35]. Moreover, we expect that it will pave the way to devising a three-dimensional variant of the visibility-Voronoi complex [34], a structure that enables to trade-off clearance and path length in robot motion planning, and has proved to be especially useful in the plane.

Acknowledgments. We thank S. Lazard and M. Yvinec for fruitful discussions.

[5] Since the trapezoidal map is not yet available for envelopes, we had to resort to instances that keep the complexity of a cell small.

[6] For the most recent version and supplemental material we refer to: `http://acg.cs.tau.ac.il/projects/internal-projects/3d-lines-vor/project-page`

References

1. Agarwal, P.K., Schwarzkopf, O., Sharir, M.: The overlay of lower envelopes and its applications. Disc. Comput. Geom. 15(1), 1–13 (1996)
2. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J., Urrutia, G. (eds.) Handb. Comput. Geom., ch. 5, pp. 201–290. Elsevier, Amsterdam (2000)
3. Austern, M.H.: Generic Programming and the STL. Addison-Wesley, Reading (1999)
4. Berberich, E., Hemmer, M., Kerber, M.: A generic algebraic kernel for non-linear geometric applications. Research Report 7274, INRIA (2010)
5. Berberich, E., Hemmer, M., Kettner, L., Schömer, E., Wolpert, N.: An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In: Mitchell, J., Rote, G., Kettner, L. (eds.) Proc. 21st Annu. ACM Symp. Comput. Geom., pp. 99–106. ACM Press, Pisa (2005)
6. Blum, H.: A transformation for extracting new descriptors of shape. In: Wathen-Dunn, W. (ed.) Models for the Perception of Speech and Visual Form. MIT Press, Cambridge (1967)
7. Boissonnat, J.D., Delage, C.: Convex hull and Voronoi diagram of additively weighted points. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 367–378. Springer, Heidelberg (2005)
8. Boissonnat, J.D., Teillaud, M. (eds.): Effective Computational Geometry for Curves and Surfaces. Mathematics and Visualization. Springer, Heidelberg (2006)
9. Culver, T., Keyser, J., Manocha, D.: Exact computation of the medial axis of a polyhedron. Computer Aided Geometric Design 21(1), 65–98 (2004)
10. Devillers, O.: Improved incremental randomized Delaunay triangulation. In: Proc. 14th Annu. ACM Symp. Comput. Geom., pp. 106–115. ACM Press, New York (1998)
11. Devroye, L., Lemaire, C., Moreau, J.M.: Expected time analysis for Delaunay point location. Computational Geometry 29(2), 61–89 (2004)
12. Dupont, L., Hemmer, M., Petitjean, S., Schömer, E.: Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 633–644. Springer, Heidelberg (2007)
13. Edelsbrunner, H., Seidel, R.: Voronoi diagrams and arrangements. Disc. Comput. Geom. 1, 25–44 (1986)
14. Emiris, I.Z., Tsigaridas, E.P., Tzoumas, G.M.: The predicates for the Voronoi diagram of ellipses. In: Proc. 22nd Annu. ACM Symp. Comput. Geom., pp. 227–236. ACM Press, New York (2006)
15. Emiris, I.Z., Karavelas, M.I.: The predicates of the Apollonius diagram: Algorithmic analysis and implementation. Comput. Geom. Theory Appl. 33(1-2), 18–57 (2006)
16. Everett, H., Gillot, C., Lazard, D., Lazard, S., Pouget, M.: The Voronoi diagram of three arbitrary lines in $\mathbb{R}^3$. In: Abstracts of 25th Eur. Workshop Comput. Geom. (2009)
17. Everett, H., Lazard, S., Lazard, D., Din, M.S.E.: The Voronoi diagram of three lines. In: Proc. 23rd Annu. ACM Symp. Comput. Geom., pp. 255–264. ACM Press, New York (2007)
18. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (1999)

19. Halperin, D., Kavraki, L.E., Latombe, J.C.: Robotics. In: Goodman, J.E., O'Rourke, J. (eds.) Handb. Disc. Comput. Geom., 2nd edn., ch. 48, pp. 1065–1093. Chapman & Hall/CRC, Boca Raton (2004)
20. Hanniel, I., Elber, G.: Computing the Voronoi cells of planes, spheres and cylinders in $\mathbb{R}^3$. Comput. Aided Geom. Des. 26(6), 695–710 (2009)
21. Haran, I., Halperin, D.: An experimental study of point location in planar arrangements in CGAL. ACM Journal of Experimental Algorithmics 13 (2008)
22. Frey, P.J.: : MEDIT : An interactive Mesh visualization Software. Technical Report RT-0253, INRIA (December 2001)
23. Karavelas, M.I.: A robust and effient implementation for the segment Voronoi diagram. In: Int. Symp. on Voronoi Diagrams in Sci. and Engineering, pp. 51–62 (2004)
24. Karavelas, M.I., Yvinec, M.: Dynamic additively weighted Voronoi diagrams in 2D. In: Proc. 10th Annu. Eur. Symp. Alg., pp. 586–598. Springer, London (2002)
25. Kim, D.S., Seo, J., Kim, D., Cho, Y., Ryu, J.: The beta-shape and beta-complex for analysis of molecular structures. In: Gavrilova, M.L. (ed.) Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence. Studies in Computational Intelligence, vol. 158, pp. 47–66. Springer, Heidelberg (2008)
26. Koltun, V., Sharir, M.: 3-dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. SIAM J. on Computing 32(3), 616–642 (2003)
27. Milenkovic, V.: Robust construction of the Voronoi diagram of a polyhedron. In: Proc. 5th Canad. Conf. Comput. Geom., pp. 473–478 (1993)
28. Mulmuley, K.: A fast planar partition algorithm, I. In: Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci., pp. 580–589 (1988)
29. Myers, N.: Traits: A new and useful template technique. C++ Gems 17 (1995)
30. Rineau, L., Yvinec, M.: 3D surface mesh generation. In: CGAL Editorial Board CGAL User and Reference Manual (ed.), 3.5 edn. (2009)
31. Setter, O., Sharir, M., Halperin, D.: Constructing two-dimensional Voronoi diagrams via divide-and-conquer of envelopes in space. Transactions on Computational Sciences (to appear, 2010)
32. Sharir, M.: Almost tight upper bounds for lower envelopes in higher dimensions. Disc. Comput. Geom. 12(1), 327–345 (1994)
33. The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 3.6 edn. (2010), `http://www.cgal.org/`
34. Wein, R., van den Berg, J.P., Halperin, D.: The visibility-Voronoi complex and its applications. Computational Geometry: Theory and Applications 36(1), 66–87 (2007); special Issue on the 21st European Workshop on Computational Geometry - EWCG 2005
35. Yaffe, E., Halperin, D.: Approximating the pathway axis and the persistence diagram of a collection of balls in 3-space. In: Proc. 24th Annu. ACM Symp. Comput. Geom., pp. 260–269. ACM Press, New York (2008)
36. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K. (eds.) Computing in Euclidean Geometry, 2nd edn. LNCS, vol. 1, pp. 452–492. World Scientific, Singapore (1995)