Constructing the Exact Voronoi Diagram of Arbitrary Lines in 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 $\mathbb{R}^3$. 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 on parametric surfaces together with advanced algebraic tools. Supplementary material and in particular the prototypical code of our implementation can be found in the website: http://acg.cs.tau.ac.il/projects/internal-projects/3d-lines-vor/project-page.

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 [1,2] and robot motion planing [3,4] apply Voronoi diagrams to encode point sets keeping maximal distance from atoms or obstacles, respectively. A related concept is the medial-axis transform [5], 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 [6] of work published up till 2000. Voronoi diagrams in $\mathbb{R}^2$ are well understood in almost all

⋆ 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.

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 $\Omega(n^2)$ [7], but the best known upper bound is[3] $O(n^{3+\epsilon})$ [8]. In the case of lines with a fixed number c of orientations the upper bound was improved to $O(c^4 n^{2+\varepsilon})$ [9]. A complete analysis of all possible combinatorial cases for three arbitrary lines is presented by Everett *et al.* [10,11].

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 [12], circles [13], ellipses [14], and more [15, §2]. In $\mathbb{R}^3$, an exact implementation of the Voronoi diagram of additively-weighted points was analyzed in [16], 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 [17,18]. 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) [19,20]. However, those approaches are not complete. For example, Culver's algorithm [19] does not handle singular trisector-curves.

Finally, Hanniel and Elber [21] provided an algorithm to construct the Voronoi cell of bounded planes, spheres, and cylinders in $\mathbb{R}^3$. Though it is in some aspects similar to ours, the approach is approximate, does not deal with degeneracies, and leaves robustness issues aside.

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 anticipate 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 Voronoi cell 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 (but not the same as) the well-known connection between Voronoi diagrams and lower envelopes [22]. Lower dimensional cells are represented several times, namely as part of the boundary of the

[3] 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.

VC of each line they are associated with. The implementation is developed in and based on CGAL, Computational Geometry Algorithms Library.[4]

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.* [11]: 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:

$$VC(S) = \{p \in \mathbb{R}^d \mid \forall s \in S, t \in \mathcal{O} \backslash S : d(p,s) < d(p,t) \text{ and } \forall s,t \in S : d(p,s) = d(p,t)\}.$$

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.

where $d(\cdot,\cdot)$ denotes the Euclidean distance function. When the cardinally of S is two or three it is called 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.

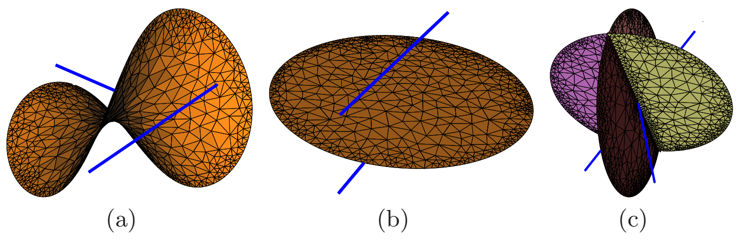


Fig. 1: Bisector of: (a) two generic lines; (b) two parallel lines; (c) two intersecting lines. The diagrams were created with our implementation (see Section 5), and were clipped by a sphere for convenience.

[4] 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 is perpendicular to ℓ_1 and ℓ_2 and passes through their intersection point.*

The main theorem of Everett *et al.* [10,11] 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 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 each three-dimensional *VC* as a lower envelope with respect to its line site ℓ_0. This lover envelope is represented as a minimization diagram which is conceptually embedded in in the uv-parameter space of the surface of an infinitesimally small cylinder around ℓ_0.[5] We utilize the divide-and-conquer algorithm for constructing lower envelopes [7] as it is implemented inCGAL [23, §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".

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

Note that the algorithm heavily relies on arrangement operations such as overlay, which are provided by [23, §8.1] and [24]. Though we treat this as a black box throughout most of the paper, some more 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, edge, or 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.

The two-dimensional package of Cgal 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 $\mathcal{X}(u, v, r) = p_0 + u \cdot \overrightarrow{b_1} + v \cdot r \cdot \overrightarrow{b_2} + r \cdot \overrightarrow{b_3}$. Note that $\mathcal{X}(u, v, \pm 1)$ defines two parallel planes (uv-planes) that sandwich ℓ_0, which we call the positive and the negative plane, respectively. Thus 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 $\pm(v_0 \cdot \overrightarrow{b_2} + \overrightarrow{b_3})$.

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. However, one can simply glue the arrangements together as long as the chosen frame is *generic*, that is, curves are not allowed to touch H^*, intersect in H^*, or even be contained in H^*. However, curves are allowed to transversely intersect H^*, where each intersection gives rise to a simple vertical asymptote in the projection.

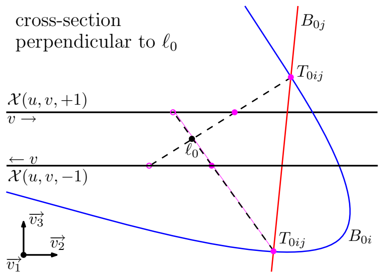
In order to avoid these critical cases, we generate the local frame by setting $\overrightarrow{b_2}$ to some random vector that is orthogonal to $\overrightarrow{b_1}$. Though this frame is generic with high probability, we also check in all relevant predicates that the frame is indeed generic. If necessary, 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 deferred to Appendix A. 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 [25]. However, 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[6] only 8. However, due to its algebraic nature the approach can not immediately distinguish between the positive and the negative parameter space. 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 segments using [24]. 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 they 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.

Further details and in particular how to guarantee that the chosen frame is generic, can be found in Appendix A.

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

[6] 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.

required predicates for these operations relate to planar algebraic curves only, which are provided by [24]. However, it remains to ensure that no intersection takes place in H^*. This boils down to testing that the leading coefficients with respect to v of two non overlapping (co-prime) curves have no common root. Thus, we provide a slightly modified set of predicates that additionally ensure this condition.

The predicates that are additionally 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 face, it is sufficient to select a rational point inside it and compare the surfaces along the corresponding ray. The point is chosen in a similar way to the approach used when sorting the trisectors to the positive and negative planes. For an edge we construct a vertex in its interior and compare along the corresponding ray. 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 we compare bisectors at a rational point *sufficiently close* to the vertex, where "sufficiently close" is again determined by a similar strategy as in sorting trisector curves.

3.3 Complexity

For the time complexity 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 [26]. 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 surfaces of maximum degree of 8 therefore, the time-complexity of the lower envelope algorithm is $O(n^{2+\varepsilon})$ (which is also the best known upper bound). Thus 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 p 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 p 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 $d(\ell, p)$ to $d(\ell', p)$ for one line $\ell' \in \mathcal{S}$. If $d(\ell, p)$ is less than or equal to $d(\ell', p)$ 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 p to the current line always decreases. We can locate the image of p inside the minimization diagram in expected $O(\log n)$ time by using point-location based on trapezoidal decomposition [27]. Combining this algorithm with the idea of landmarks [28] may already have good performance in practice. However, the algorithm has a worst-time 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, while each other layers contain the *VD* of only $1/k$ (random) lines of the preceding layer, where $k > 1$ is some constant. The highest layer (the root layer) contains only a constant number of lines, and the number of layers is $O(\log n)$. In order to locate a point p 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.

The following theorem summarizes the performance of the point-location structure (see [29,30] for similar analysis in 2D):

Theorem 2. *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.

We remark that some special cases are left out in this discussion for brevity (e.g., points that are contained in H^*), but they are completely handled in our software.

5 Implementation Details

Our implementation is based on CGAL, which follows the generic-programming paradigm [31]. 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 [32], in which these functionalities are defined. In particular, a user can employ an algorithm with his own types, constructions, and predicates by providing his 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 [23, §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 [24]. The approach reduces all construction and predicates to cylindrical algebraic decompositions of the plane for one or two curves. The approach uses additional resultant computation that projects intersection points onto the u-axis, that is, u-coordinates of intersection points are represented as real roots of this univariate resultant polynomial. Thereafter, the fibers above the roots are investigated in order to determine the arc of the curve on which the intersection takes place.

We essentially wrap the above traits class and add the required functionalities 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 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) [15, §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 that we wish to find therein. We could use a similar approach to the one used in [18]: By using the labels, we identify all possible candidates in $\mathcal{M}'$. This set contains only up to 8 representations and contains at least v'. We compute progressively 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 [33]. The oracle, which is required by the mesh generation, was written such that it only utilizes (and thereby tests) our point location structure. Lower dimensional features were approximated using the approach discussed above. In order to achieve sharp edges the protecting balls technique introduced by Boltcheva et al. [34] was applied. `medit` [35] was used 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 [24]. 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 a 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 parallel lines with coefficients in the range $[0, 2^{10}]$.[7] For each instance, we created 10 Voronoi diagram hierarchies, which where queried with 1000 random points in $[0, 2^{10}]^3$ each.

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

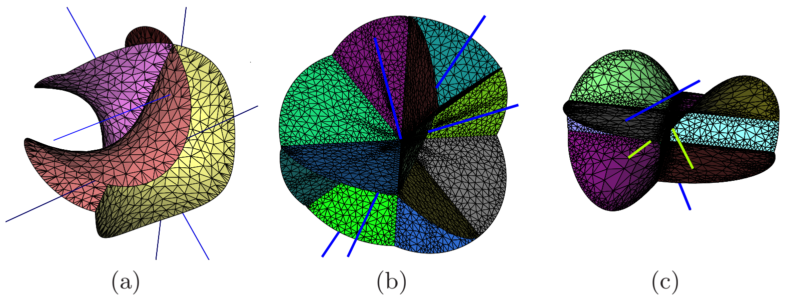


Fig. 2: Diagrams are clipped by a sphere for convenience. (a) *VD* of 4 lines, obtained by rotating one line around the z-axis. All bisectors meet in that axis. (b) *VD* of 4 lines intersecting in one point. (c) *VD* of 4 lines, two lines intersect and the others are parallel to each of them, respectively.

$N\backslash k$	2	4	8	16	20	24	28	$+\infty$
16	6.48	4.30	4.34	N/A	N/A	N/A	N/A	**3.94**
36	8.09	6.33	**5.33**	5.67	N/A	N/A	N/A	**5.62**
64	9.77	6.42	**5.73**	6.07	6.00	6.12	6.83	6.63
100	9.87	7.22	**6.18**	6.45	6.97	6.83	7.13	7.43

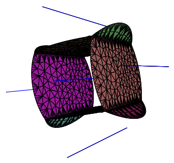


Table 1: Average number of visited cells per query, where k denotes the ratio of the hierarchy and N the number of lines. Entries for $n < 2k$ are repesented by the last column. To the right is depicted a Voronoi diagram of 5 parallel lines.

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 an average query time in $O(\sqrt{n})$, as one may also expect due to results in [36]. For larger instances, it seems that choosing k equal to 8 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 permits to answer point location queries in $O(\log^2 n)$ expected time. The implemented prototype is exact and can handle all degenerate cases. We refer to `http://acg.cs.tau.ac.il/projects/internal-projects/3d-lines-vor/project-page` for the most recent version and supplemental material.

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 computation of the Voronoi diagram of polyhedra in three dimensions.

Moreover, we expect that it will pave the way to devising a three-dimensional variant of the visibility-Voronoi complex [4], 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.

Our approach may also be generalized to spheres (see also [21]) which would open the door for innovative solutions to central problems in Structural Biology [1,2].

Acknowledgments: The authors thank S. Lazard and M. Yvinec for fruitful discussions.

References

1. Kim, D.S., Seo, J., Kim, D., Cho, Y., Ryu, J.: The beta-shape and beta-complex for analysis of molecular structures. In: Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence. Volume 158 of Studies in Computational Intelligence. Springer (2008) 47–66
2. 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., ACM Press (2008) 260–269
3. Halperin, D., Kavraki, L.E., Latombe, J.C.: Robotics. In: Handb. Disc. Comput. Geom. 2nd edn. Chapman & Hall/CRC (2004) 1065–1093
4. Wein, R., van den Berg, J.P., Halperin, D.: The visibility-Voronoi complex and its applications. Computational Geometry: Theory and Applications **36** (2007) 66–87
5. Blum, H.: A transformation for extracting new descriptors of shape. In: Models for the Perception of Speech and Visual Form, MIT Press (1967)
6. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Handb. Comput. Geom. Elsevier (2000) 201–290
7. Agarwal, P.K., Schwarzkopf, O., Sharir, M.: The overlay of lower envelopes and its applications. Disc. Comput. Geom. **15** (1996) 1–13
8. Sharir, M.: Almost tight upper bounds for lower envelopes in higher dimensions. Disc. Comput. Geom. **12** (1994) 327–345
9. Koltun, V., Sharir, M.: 3-dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. SIAM J. on Computing **32** (2003) 616–642
10. 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)
11. Everett, H., Lazard, S., Lazard, D., Din, M.S.E.: The Voronoi diagram of three lines. In: Proc. 23rd Annu. ACM Symp. Comput. Geom., ACM Press (2007) 255–264
12. Karavelas, M.I.: A robust and effient implementation for the segment Voronoi diagram. In: Int. Symp. on Voronoi Diagrams in Sci. and Engineering. (2004) 51–62
13. Emiris, I.Z., Karavelas, M.I.: The predicates of the Apollonius diagram: Algorithmic analysis and implementation. Comput. Geom. Theory Appl. **33** (2006) 18–57

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., ACM Press (2006) 227–236
15. Boissonnat, J.D., Teillaud, M., eds.: Effective Computational Geometry for Curves and Surfaces. Springer-Verlag, Mathematics and Visualization (2006)
16. Boissonnat, J.D., Delage, C.: Convex hull and Voronoi diagram of additively weighted points. In: ESA. (2005) 367–378
17. 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: Proc. 21st Annu. ACM Symp. Comput. Geom., ACM Press (2005) 99–106
18. Dupont, L., Hemmer, M., Petitjean, S., Schömer, E.: Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In: Proc. 15th Annu. Eur. Symp. Alg. Volume 4698 of LNCS., Springer (2007) 633–644
19. Culver, T., Keyser, J., Manocha, D.: Exact computation of the medial axis of a polyhedron. Computer Aided Geometric Design **21** (2004) 65–98
20. Milenkovic, V.: Robust construction of the Voronoi diagram of a polyhedron. In: Proc. 5th Canad. Conf. Comput. Geom. (1993) 473–478
21. Hanniel, I., Elber, G.: Computing the Voronoi cells of planes, spheres and cylinders in $\mathbb{R}^3$. Comput. Aided Geom. Des. **26** (2009) 695–710
22. Edelsbrunner, H., Seidel, R.: Voronoi diagrams and arrangements. Disc. Comput. Geom. **1** (1986) 25–44
23. The CGAL Project: CGAL User and Reference Manual. 3.6 edn. CGAL Editorial Board (2010)
24. Berberich, E., Hemmer, M., Kerber, M.: An algebraic kernel for CGAL. Submitted to ESA (2010)
25. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press (1999)
26. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Computing in Euclidean Geometry. Volume 1 of LNCS. 2nd edn. World Scientific, Singapore (1995) 452–492
27. Mulmuley, K.: A fast planar partition algorithm, I. In: Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci. (1988) 580–589
28. Haran, I., Halperin, D.: An experimental study of point location in planar arrangements in CGAL. ACM Journal of Experimental Algorithmics **13** (2008)
29. Devillers, O.: Improved incremental randomized Delaunay triangulation. In: Proc. 14th Annu. ACM Symp. Comput. Geom., ACM Press (1998) 106–115
30. Karavelas, M.I., Yvinec, M.: Dynamic additively weighted Voronoi diagrams in 2D. In: Proc. 10th Annu. Eur. Symp. Alg., Springer (2002) 586–598
31. Austern, M.H.: Generic Programming and the STL. Addison-Wesley (1999)
32. Myers, N.: Traits: A new and useful template technique. C++ Gems **17** (1995)
33. Rineau, L., Yvinec, M.: 3D surface mesh generation. In: CGAL User and Reference Manual. 3.5 edn. (2009)
34. Boltcheva, D., Yvinec, M., Boissonnat, J.D.: Feature preserving Delaunay mesh generation from 3D multi-material images. Computer Graphics Forum (2009)
35. Pascal J. Frey: MEDIT : An interactive Mesh visualization Software. Technical Report RT-0253, INRIA (2001)
36. Devroye, L., Lemaire, C., Moreau, J.M.: Expected time analysis for delaunay point location. Computational Geometry **29** (2004) 61 – 89

A Projection of Bisectors and Trisectors

Recall, from the discussion in Section 3 that the parametrization $\mathcal{X}(u,v,r)$ is defined with respect to a local frame $F = \{\overrightarrow{b_1}, \overrightarrow{b_2}, \overrightarrow{b_3}\}$ and that the plain H^*, which is orthogonal to $\overrightarrow{b_3}$, is not contained in $\mathcal{X}$.

In general this is not a problem since a curve (trisector) that transversely intersects H^* appears as a simple vertical asymptote in the uv-parameter space. For instance, if a projected curve leaves one plane as a vertical asymptote at plus infinity, then it reappears at minus infinity on the other plane. If this is the only case that happens it is clear that there is a one-to-on correspondence among the vertical asymptote (and thus edges) in the two minimization diagrams, which makes it possible to glue them together.

We call a frame for which this is possible a "generic frame", more precisely:

Definition 1 (Generic Frame). *We define the frame F to be* generic *if (i) no 1-dimensional components of the trisector (a conic or a line)*[8] *are contained in H^*, (ii) every intersection of a trisector with H^* is a transversal intersection, (iii) no two curves (trisectors) intersect in H^*, and (iv) the intersections of H^* with all bisectors B_{0i} are regular.*

In this section we analyze the cases that we encounter while projecting bisector boundaries and trisector curves. In particular, we discuss how we detect a non-generic frame.

A.1 Projection of Bisectors Boundary

The cases here correspond to the case distinction in Proposition 1.

Generic Case Let $B_{0i} \in \mathbb{Q}[x_1, x_2, x_3]$ be the polynomial representing the bisector between ℓ_0 (the base line) and some other line ℓ_i. In the generic case B_{0i} represents a hyperbolic paraboloid that is define in almost all directions of the projection. The only exception are rays that are perpendicular to ℓ_i and point away from ℓ_i. These directions are represented by a horizontal (representing rays in the same direction) line. The line is characterized by the leading coefficient of $B_{0i}(\mathcal{X}(u,v,r))$ with respect to r. On each of the uv-plane we interpret this surface as one or two uv-monotone surfaces. On the plane that contains the above line, we split the bisector into two uv-monotone surfaces; on the other plane there is exactly one uv-monotone surface.

Parallel Lines In case ℓ_0 and ℓ_i are parallel, B_{0i} is of degree 1 since it represents a plane. The leading coefficient of $B_{0i}(\mathcal{X}(u,v,r))$ with respect to r is a horizontal line that represents all rays that do not intersect B_{0i}. That is, on each of the uv-planes we obtain one uv-monotone surface, whose projected boundary is this line. The proper halfspace can be determined by a simple ray shoot.

[8] Note that a cubic can not be contained in H^*.

Intersecting Lines In case ℓ_0 and ℓ_i intersect, the bisector B_{0i} degenerates to two planes that intersect along a singular rational line ℓ_s, which is perpendicular to ℓ_0 and ℓ_i. Let $(u_0, v_0) \in Q^2$ be the parameter values for that line. All lines $\mathcal{X}(u_0, v, r), v \neq v_0$ have a double intersection with B_{0i}. The lines $\mathcal{X}(u, v_0, r), u \neq u_0$ do not intersect B_{0i}, since they are parallel to it. The parameter space is split up along $v = v_0$ and $u = u_0$, which results in uv-monotone surfaces, 4 one the positive and 4 on the negative side.

Ensuring a Generic Frame For all cases it holds that the construction may trigger a restart of the computation if one of the horizontal lines is not seen, that is, if the leading coefficient of $B_{0i}(\mathcal{X}(u, v, r))$ with respect to r has degree 0.

A.2 Projection of Trisectors

Besides the exactness of the method described in Section 3.1, it has the advantage that it is general and forgoes a huge case distinction. In particular, we do not factorize the polynomial into its factors that represent the different components mentioned in Theorem 1.

Two bisectors are compared above (below) a u-monotone arc c_u,[9] on the positive (negative) plane as follows. For a *sufficiently close* rational point $\overline{p}(u_0, v_0)$ above (below) c_u, the corresponding line is substituted into the two bisectors. This is at most two quadratic polynomials in r, each having at most one positive and one negative root. If present, the positive (negative) roots from the two bisectors are compared. In case that no root is present, there is no bisector above $\overline{p}$ and the comparison is not required.[10]

A special treatment is given for the case of three coplanar concurrent lines. The trisector in this case is a single perpendicular line to ℓ_0. Hence, its projection is just a rational point that is valid for both sides.

Ensuring a Generic Frame First, it is checked that no 1-dimensional component is contained in H^*, in that case $degree(res)$ is necessarily less than $2d_1d_2$ but the degree may also drop in few other cases. Thus if the degree is "suspicious", we simply intersect B_1 and B_2 with H^*. There is no 1-dimensional component if the degree of the gcd of the two resulting bivariate polynomials is constant. Moreover, we check that the projection has only simple vertical asymptote by checking that the leading coefficient of the square free part of res is square free. This ensures that the trisector intersects H^* transversely always.

[9] "Above" is the area to the left of the curve when going from its lexicographically smaller to its lexicographically larger end.

[10] This happens in case the bisector is just a simple plane.