Robust, Generic and Efficient Construction of Envelopes of Surfaces in Three-Dimensional Spaces*

Michal Meyerovitch**

School of Computer Science
Tel Aviv University
gorgymic@post.tau.ac.il

Abstract. Lower envelopes are fundamental structures in computational geometry, which have many applications, such as computing general Voronoi diagrams and performing hidden surface removal in computer graphics. We present a generic, robust and efficient implementation of the divide-and-conquer algorithm for computing the envelopes of surfaces in $\mathbb{R}^3$. To the best of our knowledge, this is the first exact implementation that computes envelopes in three-dimensional space. Our implementation is based on CGAL (the Computational Geometry Algorithms Library) and is designated as a CGAL package. The separation of topology and geometry in our solution allows for the reuse of the algorithm with different families of surfaces, provided that a small set of geometric objects and operations on them is supplied. We used our algorithm to compute the lower and upper envelope for several types of surfaces. Exact arithmetic is typically slower than floating-point arithmetic, especially when higher order surfaces are involved. Since our implementation follows the exact geometric computation paradigm, we minimize the number of geometric operations, and by that significantly improve the performance of the algorithm in practice. Our experiments show interesting phenomena in the behavior of the divide-and-conquer algorithm and the combinatorics of lower envelopes of random surfaces.

1 Introduction

Lower envelopes are fundamental structures in computational geometry, which have many applications. Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a collection of n (hyper)surface patches in $\mathbb{R}^d$. Let $x_1, \ldots, x_d$ denote the axes of $\mathbb{R}^d$, and assume that each s_i is monotone in $x_1, \ldots, x_{d-1}$, namely every line parallel to the x_d-axis intersects s_i in at most one point. Regard each surface patch s_i in $\mathcal{S}$ as the graph of a

* This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

** This work is part of the author's M.Sc. thesis under the guidance of Prof. Dan Halperin.

partially defined $(d-1)$-variate function $s_i(\bar{x})$. The *lower envelope* $\mathcal{E}_{\mathcal{S}}$ of $\mathcal{S}$ is the pointwise minimum of these functions: $\mathcal{E}_{\mathcal{S}}(\bar{x}) = \min\, s_i(\bar{x}),\ \bar{x} \in \mathbb{R}^{d-1}$, where the minimum is taken over all functions defined at $\bar{x}$. Similarly, the *upper envelope* of $\mathcal{S}$ is the pointwise maximum of these functions.

The *minimization diagram* $\mathcal{M}_{\mathcal{S}}$ of $\mathcal{S}$ is the subdivision of $\mathbb{R}^{d-1}$ into maximal connected cells such that $\mathcal{E}_{\mathcal{S}}$ is attained by a fixed subset of functions over the interior of each cell. In the same manner, the *maximization diagram* of $\mathcal{S}$ is the subdivision of $\mathbb{R}^{d-1}$ induced by the upper envelope of $\mathcal{S}$.

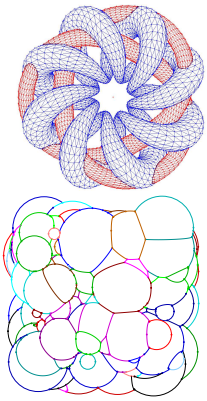
The complexity of the lower envelope of a set of surfaces is defined as the complexity of its minimization diagram. The maximum combinatorial complexity of the lower envelope of n x-monotone Jordan arcs in the plane such that each pair intersects in at most s points, for some fixed constant s, is near linear [22]. The combinatorial complexity of the lower envelope of a set $\mathcal{S}$ of n *well-behaved* (see e.g., [3]) surface patches in $\mathbb{R}^d$ is $O(n^{d-1+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε, d and some surface-specific constants [12, 21].

In this paper we deal only with envelopes in $\mathbb{R}^3$. Many algorithms for computing envelopes in three-space exist. Agarwal et al. [2] presented a divide-and-conquer algorithm with running time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. This time bound is based on their result that the combinatorial complexity of the overlay of two minimization diagrams of two collections of a total of n surface patches is also $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. Boissonnat and Dobrindt presented a randomized incremental algorithm with the same running-time bound [7]. Another randomized incremental approach [18] leads to a "quasi output sensitive" algorithm whose expected running time is a sum of weights associated with all intersections of projected objects edges, where the weight of an intersection is inversely proportional to the number of objects "hiding" that intersection from the viewing point. Obtaining output sensitive algorithms that compute envelopes is a major challenge. Such algorithms exist for special cases only. de Berg et al. [9] presented an output-sensitive algorithm for polyhedral objects. Using the data structure of Agarwal and Matoušek [1] its running time is $O(n^{2/3+\varepsilon}k^{2/3})$ for any $\varepsilon > 0$, where k is the output size and n is the number of faces of the polyhedra. Katz et al. [14] presented an output-sensitive algorithm which runs in time $O((U(n)+k)\log^2 n)$, where k is the complexity of the output map and $U(n)$ is a super-additive bound on the maximal complexity of the union of the projections on the viewing plane of any n objects. The method assumes that the surfaces can be ordered by depth from the viewing point and its efficiency shows up when $U(n)$ is small. For a comprehensive summary of results see [8].

Transforming a geometric algorithm into a computer program is not a simple task. An algorithm implemented from a textbook is susceptible to robustness issues, and thus may yield incorrect results, enter infinite loops or crash (see e.g., [15, 20]). This is mainly due to two assumptions that are often made in the theoretical study of geometric algorithms, which are not realistic in practice. First, the general position assumption excludes all degenerate inputs. Secondly, the real RAM model is assumed, which allows for infinite precision arithmetic operations on real numbers. Moreover, every operation on a constant number of

simple geometric objects is assumed to take constant time. This is of course not true in computer programs that use finite precision numbers.

CGAL, the Computational Geometry Algorithms Library,[1] is a product of a collaborative effort of several sites in Europe and Israel, aiming to provide a robust, generic and efficient implementation of geometric data structures and algorithms. It is a software library written in C++ following the generic programming paradigm. The *arrangement* of a set $\mathcal{C}$ of planar curves is the subdivision of the plane induced by the curves in $\mathcal{C}$ into maximally connected cells. CGAL provides a robust implementation for constructing planar arrangements of arbitrary bounded curves and supporting operations on them [10, 23]. Robustness is achieved both by handling all degenerate cases, and by using exact number types. The CGAL arrangement package contains a class-template, which represents a planar arrangement, and is parameterized by a traits class that encapsulates the geometry of the family of curves it handles. Robustness is guaranteed as long as the traits class uses exact number types for the computations it performs. Among the number-type libraries that are used are GMP[2] for rational numbers, and CORE[3] [13] and LEDA[4] [16, Chapter 4] for algebraic numbers.



We devised an exact and generic implementation of the divide-and-conquer algorithm for constructing the envelope of surface patches in $\mathbb{R}^3$. Our solution is complete in the sense that it handles all degenerate cases, and at the same time it is efficient. To the best of our knowledge, this is the first implementation of this kind. Our implementation is based on the CGAL library and is designated as a CGAL package. The problem of computing the envelope is somewhat two-and-a-half dimensional, since the input is three-dimensional, but the output is naturally represented as a two-dimensional object, the minimization diagram. We use the CGAL two-dimensional arrangement package for the representation of the minimization diagram. The separation of topology and geometry in our solution allows for the reuse of the algorithm with different families of surfaces, provided that a small set of geometric objects and operations on them is supplied. We used our algorithm to compute the lower or upper envelope of sets of triangles, of sets of spheres and of sets of quadratic surfaces [6]. The figures above show examples of minimization diagrams of a set of triangles and of a set of spheres as computed by our program. Our implementation follows the exact geometric computation paradigm. Exact arithmetic is typically slower than floating-point arithmetic, especially when higher order surfaces are involved. One of the main contributions of our work is minimizing the number of geometric operations, and by that significantly improving the performance of

[1] See the CGAL project homepage: `http://www.cgal.org/`.
[2] Gnu's multi-precision library `http://www.swox.com/gmp/`.
[3] `http://www.cs.nyu.edu/exact/core_pages/intro.html`.
[4] `http://www.algorithmic-solutions.com/enleda.htm`.

the algorithm in practice. Our experiments show interesting phenomena in the behavior of the divide-and-conquer algorithm and the combinatorics of lower envelopes of random surfaces. In particular, they show that on some input sets the algorithm performs better than the worst-case bound, and the combinatorial size of the envelope is typically asymptotic much smaller than the worst-case bound.

The rest of this paper is organized as follows. In Sect. 2 we briefly review the divide-and-conquer algorithm. In Sect. 3 we explain how we separate the geometry and topology, and provide the details on the geometric part of our implementation. Section 4 lists the methods we use to reduce the amount of geometric computation and improve the performance of the algorithm. In Sect. 5 we present experimental results and discuss the practical performance of the algorithm. More details on the work can be found in [17]. In the following sections, to simplify the exposition, we refer to *lower* envelopes. However, our code is generic and capable of computing envelopes in any direction, including upper envelopes.

2 The Divide-and-Conquer Algorithm

We are given as input a set $\mathcal{F}$ of n surface patches in $\mathbb{R}^3$. The first step is to extract all the xy-monotone portions of these surfaces that are relevant to the envelope. We denote this set by $\mathcal{G}$. Henceforth, for convenience, we only work on these xy-monotone surfaces in $\mathcal{G}$. (Some of our methods described in Sect. 4 are valid only under the assumption that the surfaces are xy-monotone. Yet, this assumption does not contradict the generality of the algorithm since $\mathcal{E}_{\mathcal{F}} = \mathcal{E}_{\mathcal{G}}$.) The output of our program is a minimization diagram, represented as a planar arrangement where each arrangement feature (vertex, edge or face) is labelled with the set of xy-monotone surfaces that attain the minimum over that feature. The label can contain a single surface, several surfaces, or no surface at all, in which case we call it the no surface label.

When $\mathcal{G}$ consists of a single xy-monotone surface, we construct its minimization diagram using the projection of its boundary. When $\mathcal{G}$ contains more than one xy-monotone surface, we split $\mathcal{G}$ into two sets $\mathcal{G}_1$ and $\mathcal{G}_2$ of (roughly) equal size, recursively construct the minimization diagrams $\mathcal{M}_1$ and $\mathcal{M}_2$ of these sets respectively, and finally merge these two diagrams into the final minimization diagram $\mathcal{M}$.

The merge step is carried out as follows. First, we overlay the two planar arrangements underlying the minimization diagrams $\mathcal{M}_1$ and $\mathcal{M}_2$ to obtain the arrangement $\mathcal{O}$, where each feature is a maximal connected portion of the intersection of one feature f_1 of $\mathcal{M}_1$ and one feature f_2 of $\mathcal{M}_2$. For each feature in $\mathcal{O}$ we keep two pointers to f_1 and f_2.

Next, we determine the structure of the minimization diagram over each feature in $\mathcal{O}$, to obtain the arrangement $\mathcal{O}'$, which is a refinement of the arrangement underlying $\mathcal{M}$. We then label each feature of $\mathcal{O}'$ with the correct envelope surfaces. We should consider here the two relevant features in $\mathcal{M}_1$ and $\mathcal{M}_2$ and their labels l_1 and l_2, respectively. The non-trivial case is when both

labels represent non-empty sets of xy-monotone surfaces. These surfaces are defined over the entire current feature f, and their envelope over f is the envelope of $\mathcal{G}_1 \cup \mathcal{G}_2$ there. Since all the xy-monotone surfaces of one label l_i overlap over the current feature f, it is possible to take only one representative surface s_i from each label and find the shape of their minimization diagram over f. Here we should consider the intersection between the representative surfaces s_1 and s_2, or more precisely, the projection $\mathcal{C}$ (which is, in general, a set of curves) onto the xy-plane of this intersection, which may split the current feature, if it is an edge or a face. We then label all the features of the arrangement of $\mathcal{C}$ restricted to f (where f is considered relatively open) with the correct label, which might be either one of the labels l_1 and l_2, or $l_1 \cup l_2$ in case of an overlap.[5] A face of the overlay handled in this step, and hence the arrangement restricted to the face, can be very complicated, with arbitrary topology (including holes, isolated points and "antennas") and unbounded complexity.

Finally, we apply a cleanup step in order to remove redundant features (edges or vertices) of $\mathcal{O}'$ and obtain the minimization diagram $\mathcal{M}$ of $\mathcal{G}$.

The algorithm that we implemented (and described in detail below) is similar to the one presented by [2] with one main difference — it handles the complex faces directly instead of performing a vertical decomposition to get simple constant-size faces.

3 The Geometric Traits Class

Our algorithm is parameterized with a *traits* class [5, 19], which serves as the geometric interface to the algorithm. The traits class encapsulates the geometric objects the algorithm operates on, and the predicates and constructions on these objects used by the algorithm. In this manner the algorithm is made generic and independent of the specific geometry needed to handle a special type of surfaces.

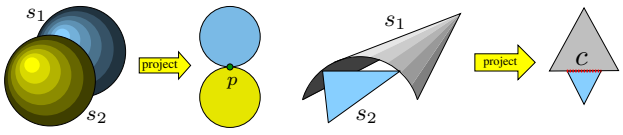
The set of requirements from a traits class forms a *concept* [5]. The geometric-traits concept for the envelope algorithm refines (extends) the concept for building planar arrangements of general bounded curves. The latter concept, which is described in detail in [23], defines three object types: planar points, x-monotone curves and general curves, and operations on them. The envelope concept adds to these requirements two more object types: three-dimensional xy-monotone surfaces and general surfaces, and the following operations on them:

1. Given a general surface, extract maximal continuous xy-monotone patches of the surface which contribute to its envelope.
2. Construct all the planar curves that form the boundary of the vertical projection of a given xy-monotone surface onto the xy-plane.
3. Construct all the planar curves and points, which compose the projection (onto the xy-plane) of the intersection between two xy-monotone surfaces s_1 and s_2, or return an empty set in case these surfaces do not intersect. If

[5] With a slight abuse of notation we use the label l_i to denote the corresponding set of surfaces.

possible, indicate, for each projected intersection curve, whether the *envelope order* of s_1 and s_2 changes when crossing that curve, or not. The envelope order of s_1 and s_2 indicates whether s_1 is below/coincides with/is above s_2. This information (referred to as the intersection type information) is optional — when provided, it is used to improve performance of the algorithm, as is explained in Sect. 4.

4. Given two xy-monotone surfaces s_1 and s_2, and a planar point p, which lies in their xy definition range, determine the envelope order of s_1 and s_2 at the xy-coordinates of p. This operation is only used in degenerate cases. A situation where this operation is used is illustrated in Fig. 1(a).
5. Given two xy-monotone surfaces s_1 and s_2, and a planar x-monotone curve c, which is a part of their projected intersection, determine the envelope order of s_1 and s_2 immediately above (similarly below) the curve c (in the plane). Note that c is a curve in the plane, and we refer to the region above/below c in the *plane*, here and in the description of Operation 6.
6. Given two xy-monotone surfaces s_1 and s_2, and a planar x-monotone curve c, which lies fully in their common xy range, such that s_1 and s_2 do not intersect over the interior of c, determine the envelope order of s_1 and s_2 in the interior of c. Figure 1(b) illustrates a situation where this operation is used.



(a) (b)

Fig. 1. (a) Two tangent spheres s_1 and s_2 will be compared over the only point p in their common xy-range. (b) The surfaces s_1 and s_2 will be compared over the interior of the line segment c.

Making the traits-class concept as tight as possible, by identifying the minimal number of required methods, is crucial. It can make the whole difference between being able to implement a traits class for a specific type of surfaces or not, and may have a major effect on the efficiency of the algorithm, especially for non-linear objects. This observation has guided us in our design. Our algorithm does not require the traits class to define any three-dimensional types except the surface types. The interface with the traits class contains only the necessary types for the input and the output of the algorithm, and a small set of operations defined on these types.

Our implementation of the divide-and-conquer algorithm is completely independent of the direction in which the envelope is to be computed. The traits is responsible for controlling this direction. All our traits classes support the computation of a lower and an upper envelope. It is also possible to use our algorithm to compute the envelope seen from a direction that is not parallel to the z-axis, provided that the traits-class correctly implements all operations with respect to that direction.

4 Reducing the Number of Algebraic Operations

We invested considerable effort in trying to minimize the amount of algebraic computation whenever possible, as such computation is usually very costly. This actually means substituting calls to the different traits-class methods by the propagation of information (in the form of labels) between incident features.

In the rest of this section we use the following notation. We merge two minimization diagrams $\mathcal{M}_1$ and $\mathcal{M}_2$ (representing the lower envelopes $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively) of two sets of xy-monotone surfaces $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. The result of the merge is a minimization diagram $\mathcal{M}$ (representing the lower envelope $\mathcal{E}$) of the set $\mathcal{G}_1 \cup \mathcal{G}_2$.

We use two types of information caching to avoid recomputing some geometric information, (for more details on these caches we refer the reader to [17]).

- Cache for projected intersections of pairs of xy-monotone surfaces (since the projected intersection of the same pair of surfaces may arise in the algorithm more than once).
- Cache for comparison results of pairs of disjoint xy-monotone surfaces, where the projection onto the xy-plane of each of the surfaces is convex.

Let $\mathcal{O}'$ be the arrangement, which is created by overlaying the arrangements underlying $\mathcal{M}_1$ and $\mathcal{M}_2$, and determining the shape of the minimization diagram over each feature of the overlay. $\mathcal{O}'$ is a refinement of the arrangement underlying $\mathcal{M}$. For each feature f of $\mathcal{O}'$ the envelope $\mathcal{E}$ is attained by the same set of surfaces over all points of f. $\mathcal{O}'$ is the input to the labelling step, which determines the correct label of all the features of $\mathcal{O}'$. For each feature f of $\mathcal{O}'$ we have to decide between two labels from the two minimization diagrams currently being merged; we say that f is associated with a *decision*. A *decision* can be one of three values: first, when the feature should be labelled with all the surfaces of the first label, second, when the feature should be labelled with all the surfaces of the second label and both, when the surfaces of both labels overlap over the feature. We work with decisions in the labelling step, and after the cleanup step, we translate the decisions into the relevant labels.

Obviously, in order to make a decision for a feature, we can use one of the three types of comparison operations described in Sect. 3. However, we can use the following observations to significantly reduce the number of such operations. These savings are demonstrated in the experiments reported in Sect. 5.

- No need to compare xy-monotone surfaces over their projected intersection, since they are equal there.
- Information on intersection type can be used when available, to avoid the comparison of two xy-monotone surfaces on one side of a curve (which is a part of their projected intersection) if their envelope order on the other side of that curve is known. Recall that such information is (optionally) given by the traits operation which constructs the projected intersection of two xy-monotone surfaces. Similar information is extremely helpful in constructing two-dimensional arrangements of curves [11].

– Information on the continuity or discontinuity of the two envelopes currently being merged can be used in order to conclude the decision for a feature from a decision on an incident feature. We regard the following as incident features: (i) a face and an edge on its boundary, (ii) a face and a vertex on its boundary, and (iii) an edge and its endpoint vertices.

Using continuity or discontinuity information, as we explain in the rest of this section, it is possible to carry a decision over from a face to a boundary edge and vice versa. To best exploit this property, we traverse the faces in a breadth-first order, moving from a face to its neighboring faces.

Using Continuity or Discontinuity Information

We consider the xy-monotone surfaces as graphs of partially defined bivariate continuous functions, and their envelope as a function defined over the entire xy-plane. In addition, we consider the boundary of an xy-monotone surface to be part of this surface. For lack of space, we omit proofs here; the interested reader can find them in [17].

Definition 1. *Let $\mathcal{E}$ be an envelope and $\mathcal{M}$ its minimization diagram. Let f and e be two incident features of $\mathcal{M}$, such that e lies on the boundary of f. We say that $\mathcal{E}$ meets f and e continuously if $\mathcal{E}$ restricted to $f \cup e$ is continuous over e.*

Observation 1. *Let $\mathcal{E}$ be an envelope of a set $\mathcal{S}$ of surfaces and $\mathcal{M}$ its minimization diagram. Let f and e be two incident features of $\mathcal{M}$, such that e lies on the boundary of f. $\mathcal{E}$ meets f and e continuously if and only if there exists an xy-monotone surface $s \in \mathcal{S}$ which appears over f and e on the envelope $\mathcal{E}$.*

We use this observation in our implementation to decide where an envelope meets two incident features continuously. This information in then used to reduce the number of geometric comparisons according to the following lemma.

Lemma 1. *Let f be a face of $\mathcal{O}'$ and e be an edge on its boundary. Suppose that: (i) a decision over the face f is known, and (ii) both envelopes $\mathcal{E}_1$ and $\mathcal{E}_2$ meet f and e continuously. Let s_1 and s_2 be the xy-monotone surfaces that appear over f and e on these envelopes respectively. Suppose further that e is not part of the projected intersection of the surfaces s_1 and s_2. Then the decision made on f is valid also on e. Similar statements can be used for other types of incident features.*

Sometimes, we can use also the *discontinuity* information to deduce a decision for a feature without comparing the relevant surfaces.

Observation 2. *Let $\mathcal{E}$ be a lower envelope and $\mathcal{M}$ its minimization diagram. Let f and e be two incident features of $\mathcal{M}$, such that e lies on the boundary of f. Let s_f and s_e be representative xy-monotone surfaces of $\mathcal{E}$ over f and e respectively. $\mathcal{E}$ does not meet f and e continuously if and only if s_e lies below s_f over e (note that s_f is defined over e).*

Applying Observation 2 to the labelling step we get:

Lemma 2. *Let f be a face of $\mathcal{O}'$ and e be an edge on its boundary. Assume that the lower envelope $\mathcal{E}_1$ meets f and e continuously, but the lower envelope $\mathcal{E}_2$ does not. (i) If $\mathcal{E}_2$ is below $\mathcal{E}_1$ over f, then $\mathcal{E}_2$ is below $\mathcal{E}_1$ also over e. (ii) If $\mathcal{E}_1$ is below $\mathcal{E}_2$ over e, then $\mathcal{E}_1$ is below $\mathcal{E}_2$ also over f. Similar arguments apply to all other incidence relationships.*

Figure 2 illustrates how the observations above are used in the labelling step. Table 2 in Sect. 5 demonstrates the significant savings in geometric operations by the techniques presented above. For example, for a set of 1000 random triangles the total number of comparison operations reduces from 265,211 to 13,620, and for a set of 1000 random spheres the total number of such operations reduces from 48,842 to 2,457, which is roughly a saving of 95% in either case.

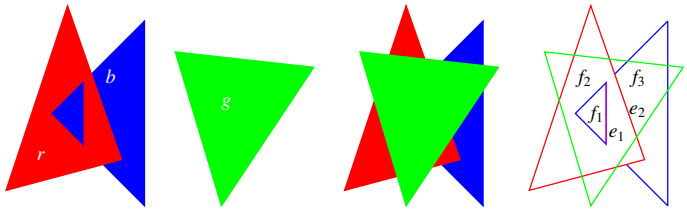


Fig. 2. Applying continuity or discontinuity arguments to carry on a decision between incident features: (a) the envelope of two triangles r and b, (b) the envelope of one triangle g, (c) the envelope of r, b and g, (d) the overlaid arrangement before the labelling step. To label face f_1 with g we compare triangles b and g. Using Lemma 1, we label all the features on the boundary of f_1 with g. To label face f_2, where triangles r and g should be compared, we use Lemma 1 and the edge e_1 to conclude that g appears on the envelope, without actually comparing r and g. Using Lemma 1, we label all the features on the boundary of f_2 with g. Finally, we use Lemma 2 and the edge e_2 to set the label of face f_3 to g. To summarize, we need only compare triangles b and g once, and all the other decisions follow.

5 Experimental Results

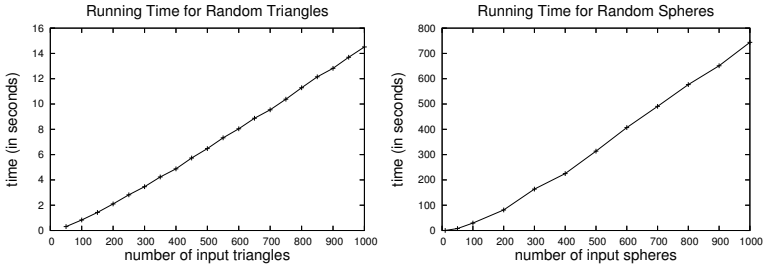
In this section we present experimental results that demonstrate the performance of our algorithm and show its behavior on various input sets. Many more experimental results are available in [17]. The running times reported in this section were obtained on a single 3 GHz PC with 2 Gb of RAM, running under Linux.

Our experiments were conducted on the following input sets:

- `rnd_triangles_`n n triangles, each of which was generated by choosing the coordinates of its three vertices uniformly at random as integers in the cube $[0, 10000]^3$.
- `rnd_small_`p`_triangles_`n n triangles, each of which was generated by first choosing the coordinates of one corner of the triangle uniformly at random

in the cube $[0, 10000]^3$, then choosing two random points in the sphere with radius $p * 10000$ around this point. All the vertex coordinates are integers.
- `rnd_small_spheres_`n n spheres, where the centers were chosen with integer coordinates uniformly at random in the range $[-1000, 1000]^3$, and the integer radii were chosen uniformly at random in the range $[1, 250]$.
- `rnd_spheres_`n n spheres, where the centers were chosen with integer coordinates uniformly at random in the range $[-1000, 1000]^3$, and the integer radii were chosen uniformly at random in the range $[1, 500]$.

We measured the running time of our algorithm on various types of examples, to investigate the behavior of the algorithm in practice. Some of the results are shown in Fig. 3. Our results show that on some inputs sets the algorithm performs better than the worst-case estimate.



(a) (b)

Fig. 3. The running time of computing the envelope for different input sizes: (a) `rnd_triangles_`n, (b) `rnd_spheres_`n.

We investigated the behavior of the size of the lower envelope of specific input sets; results graphs are available in [17]. For the `rnd_triangles_n` input sets, the results show that the size of the minimization diagram is roughly[6] $\Theta(n^{2/3})$. For other input sets the minimization diagrams are sub-linear in the input size as well.

Table 1 shows the actual running time for different input sets, each consisting of 1000 input surfaces. In the last three columns we give statistics of the whole process, which can give an idea about the amount of work that is carried out during the whole execution. It can be seen that the algorithm is much slower when run on non-linear input than on linear input; this is expected when using exact arithmetic, since with linear input, rational arithmetic suffices, whereas with non-linear input, algebraic numbers should be used. For lack of space we omit the statistics on the size of the minimization diagrams, which demonstrate the huge variance in output size for the same (combinatorial) input size; they can be found in [17].

[6] This bound of roughly $\Theta(n^{2/3})$ is inspired by recent results of Alon et al. [4] for envelopes of segments in the plane. It seems that some of their results extend to 3D implying this bound.

Table 1. Results for different input sets. *Intermediate* is the total sum of the combinatorial size of all the minimization diagrams computed during the recursion. *Intersections* is the number of intersections between pairs of surfaces that were found by the algorithm. *2d-Intersections* is the number of two-dimensional intersections between projected x-monotone curves that were found during the entire run of the algorithm.

Input File (1000 surfaces)	Time (seconds)	Process details:		
		Intermediate	Intersections	2d-Intersections
rnd_triangles	14.073	190,942	12,007	52,990
rnd_small_0.1_triangles	2.369	94,906	117	11,134
rnd_small_0.5_triangles	6.532	144,383	2,676	29,093
rnd_small_spheres	249.111	60,465	842	7,472
rnd_spheres	654.044	53,188	1,565	8,547

Table 2 shows the algorithm running time and the number of calls to the three types of comparison methods made by the algorithm in the labelling process, comparing between the naïve approach and our approach. The naïve approach means comparing surfaces over all features, except over projected intersections. Our approach is described in Sect. 4, and uses the intersection type and continuity/discontinuity information together with a breadth-first traversal of the faces. Both approaches use the caching information described in Sect. 4. It can be seen that the reduction in the number of operations is highly significant for all the input sets. We remark that the number of comparison operations over a two-dimensional point reduces to zero in our approach since these examples do not contain degeneracies in which this operation is invoked.

Table 2. Comparing the number of comparison operations used by the algorithm with and without our means for reducing the number of algebraic operations. The *Pt., Cv. and Cv.-side* columns represent the number of calls made to the appropriate version of the comparison method: comparison over a planar point, comparison over a planar x-monotone curve and comparison above/below a planar x-monotone curve respectively. The construction time is given in seconds.

Input File (1000 surfaces)	Naïve solution				Using our improvements			
	Pt.	Cv.	Cv.-side	Time	Pt.	Cv.	Cv.-side	Time
rnd_triangles	85,091	166,405	13,715	25.028	0	10,333	3,287	14.073
rnd_small_0.3_triangles	42,090	76,763	1,934	6.861	0	8,244	791	5.263
rnd_small_0.5_triangles	51,598	96,662	3,703	9.593	0	8,851	1,325	6.532
rnd_small_spheres	14,901	25,853	1297	399.327	0	2,466	450	249.111
rnd_spheres	14,965	25,630	2247	1116.430	0	1,840	617	654.044

References

1. P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

2. P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.
3. P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
4. N. Alon, D. Halperin, O. Nechushtan, and M. Sharir. The complexity of the outer face in arrangements of random segments. Manuscript, 2006.
5. M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1999.
6. E. Berberich and M. Meyerovitch. Computing envelopes of quadrics. In preparation.
7. J.-D. Boissonnat and K. T. G. Dobrindt. On-line construction of the upper envelope of triangles and surface patches in three dimensions. *Comput. Geom. Theory Appl.*, 5(6):303–320, 1996.
8. M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *LNCS*. Springer-Verlag, 1993.
9. M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994.
10. E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *J. of Experim. Alg.*, 5:1–23, 2000.
11. E. Fogel et al. An empirical comparison of software for constructing arrangements of curved arcs. Technical Report ECG-TR-361200-01, Tel-Aviv Univ., 2004.
12. D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.
13. V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proc. Symp. on Computational Geometry 1999*, pages 351–359, 1999.
14. M. J. Katz, M. H. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.*, 2:223–234, 1992.
15. L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. In *Proc. European symp. on Algorithms 2004*, volume 3221 of *LNCS*, pages 702–713, 2004.
16. K. Mehlhorn and S. Näher. LEDA*: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.
17. M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. M.Sc. thesis, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, July 2006.
18. K. Mulmuley. An efficient algorithm for hidden surface removal, II. *J. Comput. Syst. Sci.*, 49(3):427–453, 1994.
19. N. Myers. "Traits": A new and useful template technique. In S. B. Lippman, editor, *C++ Gems*, volume **5** of *SIGS Reference Library*, pages 451–458. 1997.
20. S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.
21. M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.
22. M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge-New York-Melbourne, 1995.
23. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. In *Proc. Library-Centric Software Design 2005*, 2005.