Robust Geometric Computing in Motion

Dan Halperin, *Tel Aviv University, ISRAEL*

Transforming a geometric algorithm into an effective computer program is a difficult task. This transformation is particularly made hard by the basic assumptions of most theoretical geometric algorithms concerning complexity measures and (more crucially) the handling of robustness *issues, namely issues related to arithmetic precision and degenerate input. The paper starts with a discussion of the gap between the theory and practice of geometric algorithms, together with a brief review of existing solutions to some of the problems that this dichotomy brings about.*

We then turn to an overview of the CGAL *project and library. The* CGAL *project is a joint effort by a number of research groups in Europe and Israel to produce a robust software library of geometric algorithms and data structures. The library is now available for use with significant functionality. We describe the main goals and results of the project.*

The central part of the paper is devoted to arrangements (i.e., space subdivisions induced by geometric objects) and motion planning. We concentrate on the maps and arrangements part of the CGAL *library. Then we describe two packages developed on top of* CGAL *for constructing robust geometric primitives for motion algorithms.*

1 Introduction

For over two decades research in Computational Geometry has yielded numerous efficient algorithms and data structures for solving problems arising in a diversity of areas from statistics and chemistry to GIS and robotics [46],[61],[80]. Whereas the original focus of this field was almost exclusively theoretical, these days a lot of attention is given to practical solutions and their software implementation. Transforming a theoretical geometric algorithm into an effective running program is a hard task. The difficulties in this process are rooted in the assumptions of the theoretical study concerning complexity measures, the arithmetic model and the treatment of degenerate input.

In this paper we discuss the gap between the theory and practice of geometric algorithms. We then describe efforts to settle this gap and facilitate the successful implementation of geometric algorithms in general and of algorithms for geometric arrangements and motion planning in particular.

Most of the algorithms developed in Computational Geometry aim for efficient worst-case asymptotic running time and storage. The computational model used is the so-called *real* RAM model that allows for infinite precision arithmetic operations [76]. Every operation on a (small) constant number of simple geometric objects (such as line segments, circles, planes) takes 'unit' time. In addition the input to the algorithm is assumed to be in *general position*, namely degenerate configurations are excluded. For example, no three of a set of input points may lie on a single line. We collectively refer to the issues of arithmetic precision and the treatment of degenerate input as *robustness issues.*

Each of the assumptions we have just mentioned needs to be revised in practice. Some of the problems are not unique to geometric algorithms, such as the fact that the standard asymptotic measures of algorithm performance may hide prohibitively large constants. This pitfall has been observed many times in geometry, for example in the context of range searching [39], vertical decomposition [56], and construction of Minkowski sums [4]; the latter example will be described in more detail in the sequel.

Sometimes the worst-case resource bounds are deterringly high since they cover the treatment of even the most pathological non-realistic input instances. This has led to the study of realistic input models (a.k.a. "fatness"). Making additional assumptions on the input (based on the specific nature of a problem at hand) and tailoring the algorithms accordingly can result in much more efficient algorithms that are often simpler [5],[30],[53],[90],[91].

The unit-cost model for operations on a constant number of simple geometric objects is also questionable, especially when we aim for robust computation. Already in two-dimensions and for collections of simple algebraic curves (e.g., circular arcs), algorithms need to determine the sign of a polynomial of a rather high degree [16]. Carrying out one such operation robustly may be costly. Redesigning algorithms so that they use lower-degree predicates was recently proposed as a means to enhance robustness [15],[16].

Indeed, robustness issues seem to be the most critical in the passage from theory to practice in geometric algorithms. Ignoring these issues can result in unreliable or incorrect programs. This has led to an intensive study in recent years. We briefly mention the main approaches to handling robustness issues next and refer the reader to recent surveys on the topic [81],[94] for further information.

Let us illustrate the problem of arithmetic precision in geometric computing. Consider the two polygons on the left-hand side of Figure 1: does the small polygon fit inside the cavity in the larger polygon? We find the answer by computing the Minkowski sum of the larger polygon and a copy of the smaller polygon rotated by 180° degrees (the details are deferred to Section 4.1). Suppose the answer is yes as the figure implies. To give a correct answer we must be able to identify a singular point on the boundary of the Minkowski sum (in the middle of the sum on the right-hand side of Figure 1) which is the intersection point of many line segments. The standardly available computer arithmetic and number types could not in general give such answers.

A solution is given by using *exact arithmetic* (see, e.g., [19],[20],[27],[71],[95]). We can use rational numbers and maintain the numerator and denominator as unlimited length integers (supported by several software packages). Precise predicates are also supported when they involve square roots or even taking the k-th root [21],[22],[62],[69],[70].

Exact arithmetic predicates can be rather time consuming. There are several adaptive evaluation schemes that save computing time by resorting to expensive computations only when they cannot determine the

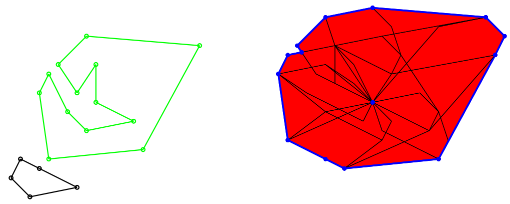


Figure 1: The small polygon can fit into the cavity in the larger polygon (left-hand side) as indicated by a singular point in the middle of the Minkowski sum of the larger polygon with a rotated copy of the small one (right-hand side)

correct answer by simpler means like the standard computer arithmetic. One form of adaptive evaluation is a *floating-point filter* which makes use of the hardware-supported floating-point arithmetic at the initial stage. For the use of such techniques in exact geometric computing see for example [32],[43],[63],[71],[88]. The packages described in Sections 3 and 4 make extensive use of LEDA's filters [71] to speed up their running time; see for example the experimental results reported in [42] for a comparison of exact arithmetic with or without filters.

If we are not using exact arithmetic, precision problems become especially noticeable at or near degeneracies. Burnikel et al. [23] propose to handle degeneracies directly[1] for certain two-dimensional problems. In three and higher dimensional problems however, directly handling degeneracies is an extremely tedious and error-prone task. Symbolic perturbation was suggested as a general means to overcome degeneracies [34],[36],[85],[93] provided that exact arithmetic is available.

If one insists on using only fixed-precision arithmetic then several methods that approximate the geometric objects were proposed to guarantee robustness and/or remove degeneracies; see for example, [47],[49],[54],[56],[59],[72],[89]. The need for approximating (or rounding) geometric objects arises even if we use exact arithmetic, once we wish to output numerical values for newly constructed features, such as the (coordinates of a) point of intersection of two curves.

[1]By handling degeneracies *directly* we mean allowing degenerate input and (re)designing the algorithm or code to cope with degeneracies. This is in contrast with indirect handling of degeneracies by first abolishing them through, say, perturbation.

Because of all the difficulties in implementing geometric algorithms (as discussed above) several computational-geometry groups have decided to put up a carefully designed and implemented library with an emphasis on robustness and generality [75]. This resulted in CGAL: The Computational Geometry Algorithms Library.

The CGAL project[2] officially started in September 1996. The actual work on the library's kernel started earlier [37] and drew on the experience from still earlier projects including: the geometric part of LEDA [70], C++gal [3], PlaGeo and SpaGeo[4], and the XYZ-workbench [82]. In the next section we give an overview of the CGAL project and library, the main design goals that guided the development of the library and a brief description of its various parts.

CGAL is definitely not the only software development effort of computational geometry algorithms. Implementations of geometric algorithms have been reported in the literature and were made available for various problems; see [7] and [8]. These include libraries and workbench efforts. Schirra [81] comments that as far as libraries/workbenches are concerned, two have paid special attention to robustness issues: The XYZ-workbench and LEDA.

This paper concentrates on *arrangements* and *motion planning*. Arrangements are space subdivisions induced by geometric objects and they are closely related to the study of collision-free path planning for objects moving among obstacles. Each of these topics, as well as the connection between them, has been the subject of extensive research. We give more background about these areas and point to bibliography in the relevant sections. In Section 3 we focus on the maps and arrangements part of the CGAL library which has been developed at Tel Aviv University. We explain the connection between arrangements and motion planning in Section 4 and describe two packages developed on top of CGAL that provide robust primitives for motion algorithms. Concluding remarks and suggestions for further research are given in Section 5.

2 The CGAL Project and Library

The Computational Geometry Algorithms Library CGAL is a software library of robust geometric algorithms and data structures.

The participating groups in the CGAL project are from ETH Zurich (Switzerland), The Free University Berlin (Germany), INRIA Sophia-Antipolis (France), Max Planck Institute Saarbrücken (Germany), RISC Linz (Austria), Tel Aviv University (Israel), Trier University (Germany) and Utrecht University (The Netherlands).

Among the major goals in the design of the library were: Robustness, generality, and efficiency [18],[37],[38]. In general **robustness** is achieved in CGAL through exact computation of geometric predicates and constructors[5], which often require the use of special number types. The user is free to choose different number types such as the machine float or double, but then the algorithms are not always guaranteed to produce the correct output. There is a clear indication in CGAL under which conditions the algorithms are certified to work correctly.

Number types could be easily imported from other packages such as LEDA or *The GNU Multiple Precision Arithmetic Library* [48] and smoothly used with CGAL. The ability of the user to specify different number types for the same algorithm is one aspect of the **generality** of the library. The project has adapted [18] the so-called *generic programming* paradigm that makes heavy use of the C++ template mechanism [11]. In addition to number types users can choose a representation type for point coordinates: Cartesian or homogeneous. Major flexibility and generality is achieved through the use of 'traits classes'. A traits class brings together all the data types and operations needed by a certain algorithm and is passed as an additional parameter to the algorithm. In Section 3.2 we illustrates

[2] The CGAL project and its successor project GALIA were both funded by the European Union. Throughout the paper we refer to both projects as the CGAL project.

[3] F. Avnaim, C++gal: A C++ library for geometric algorithms, INRIA Sophia-Antipolis, 1994.

[4] G.-J. Giezeman, PlaGeo, a library for planar geometry, and SpaGeo, a library for spatial geometry, Utrecht University, 1994.

[5] A geometric predicate typically computes the sign of an expression. A constructor produces a new geometric object such as the intersection point of two lines. The preciseness of predicate evaluation is crucial to the correct flow (branching) of the program control. Notice however that imprecise constructors could also be detrimental to the flow of control once the result of an imprecise construction is used in a predicate.

how traits classes enhance the generality of the code for representing two-dimensional subdivisions.

For a detailed discussion of the above and additional goals in the design of CGAL see [18],[37],[38].

Notice that there is no global strategy for handling degenerate input in CGAL. If an algorithm is claimed to perform correctly as long as exact predicates are provided, this means in particular that it could handle degenerate input. While this is often doable and even desirable for two-dimensional problems [23] it may be unnecessary or extremely difficult for higher-dimensional problems. In Section 4.2 we propose an efficient method for removing degeneracies from three-dimensional arrangements.

CGAL consists of three parts. The first is the kernel, which consists of primitive constant-size geometric objects and predicates on them (e.g., points and orientation test for points). The second part, the basic library, consists of a large collection of fundamental algorithms (e.g., constructing convex hulls) and data structures (e.g., kd-trees) parametrized by trait classes. The maps and arrangements package that we describe in the next section is part of the basic library. The third part incorporates non-geometric support facilities such as I/O support for debugging and for interfacing CGAL to various visualization tools. The full functionality of the library can be found in the manuals of the various parts [26]. Additional documentation includes an installation guide and a "getting started" manual guiding the novice how to use the library through a series of program examples.

CGAL can be used as a stand-alone library. However, there are close connections between CGAL and LEDA—the library of efficient data structures and algorithms. There are provisions in CGAL for easy use of LEDA's special number types, graphical window and graphical output to a postscript file. For a succinct listing of the principal differences between CGAL and the geometric part of LEDA see [71, Section 9.11].

3 Maps and Arrangements

Given a finite collection $\mathcal{S}$ of geometric objects such as hyperplanes or spheres in $\mathbb{R}^d$, the *arrangement* $\mathcal{A}(\mathcal{S})$ is the decomposition of $\mathbb{R}^d$ into connected open cells of dimensions $0, 1, \ldots, d$ induced by $\mathcal{S}$. Figure 2 illustrates a planar arrangement of segments which consists

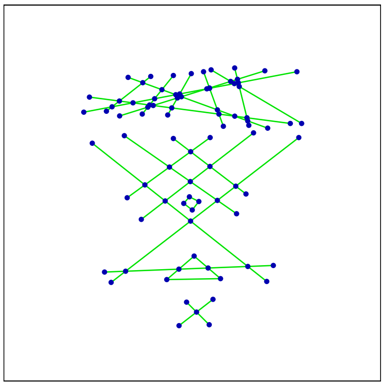


Figure 2: An arrangement of segments

of vertices, edges and faces: a *vertex* is either a segment endpoint or the intersection point of two (or more) segments, an *edge* is a maximal portion of a segment not containing any vertex, and a *face* is a maximal region of the plane not containing any vertex or edge. We assume below some familiarity with arrangements; for more information about arrangements see, for example, [3],[33],[51],[86].

Arrangements are a useful tool in the study of geometric problems in robotics and other areas [51],[86]. We use them to partition a space into cells such that certain invariants are maintained in each cell. The objects in S that define the arrangement are the loci of critical points in space where the invariants change. Arrangements lead to efficient solutions in, among other areas, robot motion planning (as we will explain in more detail in Section 4.1), assembly planning [52], and visibility problems [29].

Motivated by these applications the Tel Aviv site of the CGAL project has been responsible for the development of a package to support the construction and manipulation of arrangements of general curves. Our two-dimensional package is described in this section; many technical details that we omit here can be found in the CGAL user manual [26, Part II, Chapters 9 through 12] and in [42] and [57],[58]. In Section 4.2 we describe implementations for three-dimensional arrangements of triangles and of polyhedral surfaces built on top of CGAL.

Arrangements of curves have also been studied with an emphasis on the algebraic perspective (see, e.g., [9],[10]). MAPC [67] is a recent library effort which is complementary to the work describe here. See [67], [45] and the references therein for related work.

3.1 From Maps to Arrangements

Subdivisions could be used in CGAL in one of four levels, where each higher level is built on top of the previous one: (i) topological maps (which are not necessarily planar), (ii) planar maps, (iii) planar maps with intersections, and (iv) arrangements. We will restrict our description here to planar maps and arrangements.

A planar map in CGAL is defined as the subdivision of the plane into cells by a collection of pairwise interior-disjoint x-monotone curves. The choice to restrict the curves to be x-monotone at this level makes it easy to apply algorithmic methods such as the *vertical decomposition* [31] to otherwise general curves. The planar map level is the natural level to represent a simple plane subdivision such as a geographic map or a room with polygonal obstacles.

Planar maps in CGAL support traversal over faces, edges and vertices of the map, traversal over the edges of a face and around a vertex and efficient *point location* (namely identifying the feature of a map where a query point is located). They are dynamic and support insertion of new edges into or removal of edges out of an existing map. The representation is based on the *Doubly Connected Edge List* (DCEL) [31, Chapter 2]. This representation belongs to a family of edge-based data structures in which each edge is represented as a pair of opposite *halfedges*. The representation supports inner components (holes) inside the faces.

The curves at the arrangement level are the most general: they may intersect and are not necessarily x-monotone. The arrangement level employs the planar map level and preprocesses the input curves by decomposing each one of them into x-monotone subcurves, and further subdivides them at their intersection points with the other curves. A data structure which we call the *hierarchy tree* maintains the curve-history of each of the edges in the final planar map (namely the original curves and sub-curves in which this edge is contained). In addition to the planar-map functionality, the arrangement level supports overlapping curves and the traversal of all the curves containing a query edge.

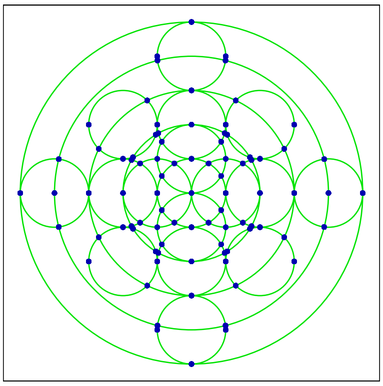


Figure 3: An arrangement of circles [58]

3.2 The Separation of Combinatorics and Numerics

As mentioned in Section 2, algorithms in CGAL are passed a so-called traits class that contains the needed data types and operations. The traits class is an abstract interface of predicates and functions that wraps the access of an algorithm to the geometric (rather than combinatorial) inner representation. We have formulated the requirements for the planar map's traits so they make as little assumptions on the curves as possible. Packing those predicates and functions under one traits class gives flexibility in choosing the coordinate representation of the objects (Homogeneous, Cartesian) and flexibility in choosing the geometric kernel (LEDA, CGAL or a user-defined kernel). The traits class enables the users to employ the maps and arrangements package with different kinds of curves that the users define (as required by their applications). The only restriction is that they obey the predefined interface.

The planar map traits class defines the two basic geometric objects of the map: the point and the x-monotone curve. In addition several types of predicates are required: For example, a predicate that returns whether a point is above, below or on a given curve, and a predicate that compares the y-coordinate of two curves at a given x-coordinate.[6]

We supply ready-made traits for line segments, circular arcs, and polylines while using various number

[6] The full list of requirements can be found in [26].

types. We mainly rely on LEDA's special number types for efficient exact arithmetic.

3.3 Point Location

Point location is a basic service required from a subdivision data structure: Given a query point p report the face of the subdivision containing p. Planar maps (and the levels above them) provide this service. We provide three algorithms for point location as well as a mechanism for the users to supply their own point-location algorithm.

The algorithms we have implemented are: (i) a naive algorithm — goes over all the edges in the map to find the location of the query point; (ii) an efficient algorithm, the default one, which is based on the Randomized Incremental Construction of a search structure through the vertical decomposition of the map [17],[73],[84]—we call it RIC for short, and (iii) a "walk-along-a-line" (walk, for short) algorithm that is an improvement over the naive approach: finds the point's location by walking along a vertical line from "infinity" towards the query point. We remind the reader that our point location implementation handles general planar maps. The subdivision is not necessarily *monotone* (each face's boundary is a union of x-monotone chains) nor connected. In addition the input may be degenerate (see for example Figure 3 where four circles intersect in a single point).

We have observed an interesting behavior of the point-location algorithms: during the construction of a map (or an arrangement) using the walk algorithm is faster than using the presumably fast RIC algorithm. The latter answers queries in logarithmic time whereas the walk algorithm may require up to linear time per query. The explanation to this phenomenon is the significant overhead in constructing the search structure (which does not express itself as strongly in the asymptotic bounds). After the map is constructed, the RIC algorithm is much faster than the other algorithms. One of our major current efforts is speeding up the point-location algorithms. See [42] for a preliminary comparison of the performance of these algorithms.

Besides the naive algorithm, implementing the point-location algorithms for possibly degenerate arrangements of arbitrary curves is non-trivial.[7] Already the walk algorithm raises difficulties; Mehlhorn and Näher describe these difficulties for their walk point-location in (straight edge) Delaunay triangulations [71]. The naive point location was used in our implementation to verify the correctness of the other algorithms through the use of a so-called 'double checker': it is a debug point-location strategy[8] that to the planar map package looks as a regular strategy but for every query it runs two different algorithms, compares their results and notifies the developer in case of a mismatch.

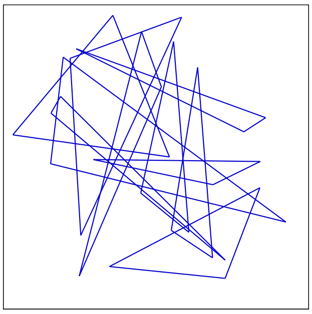
We have also developed an adaptive point location scheme for arrangements of parametric curves [58], where the construction of exact arrangements is a difficult task. The idea is to bound a curve inside a polygon (which serves as a polygonal approximation of the curve) and perform operations on the resulting arrangement of polygons. If we are unable to tell which original curves lie on the boundary of a face containing a query point, we refine the polygonal approximation until we can answer the query; see Figure 4 for an illustration on a collection of Bézier curves. This scheme makes use of the arrangement polygonal-line traits. Our work builds upon and extends the work by Neagu and Lacolle [74] who gave an algorithm to approximate the combinatorial structure of arrangements of parametric curves by arrangements of polygons.

3.4 On-line Zone Construction

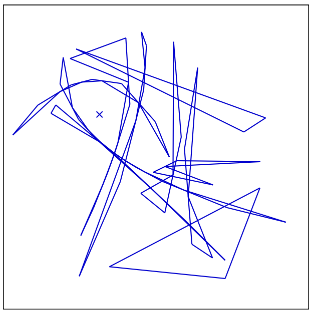
We conclude this section with another problem on planar arrangements of lines for which we implemented and compared four different algorithms: identifying the faces of an arrangement that are intersected by an additional curve γ (these faces are called the *zone* of γ); the curve is given to the algorithm on-line piece by piece. This work is motivated by the efficient exploration of the area bisectors of a polygon [13] related to using MEMS arrays for part orienting [14]. The experiments led to useful observations regarding the use of number types in arrangement computation and about the benefits of caching the results of geometric computation, both constructions and predicates, to save exact computation time (for example, once computed we retain the intersection point of two lines for later use by the algorithm instead of recomputing it). An experimental comparison between the algorithms is reported in [6].

[7] The implementation of the walk algorithm spans about 900 lines of C++ code, and the code of the RIC algorithm is about 5,000 lines.

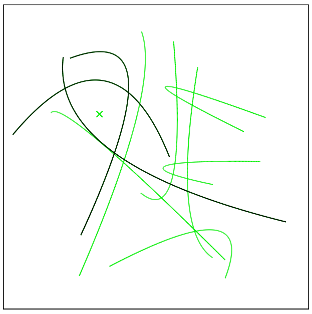
[8] The use of 'strategy' here refers to the *strategy pattern* [44].



(a)



(b)



(c)

Figure 4: An arrangement of polygons each bounding a Bézier curve before (a) and after (b) the first point location; (c) displays the underlying arrangement of Bézier curves and in bold line the curves that bound the face containing the query point

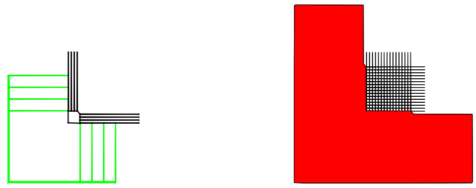


Figure 5: P and Q are polygons with horizontal and vertical teeth with m and n vertices respectively (one polygon's boundary is displayed in grey and the other's in black). The complexity of $P \oplus Q$ is $\Theta(m^2n^2)$.

4 Robust Motion Primitives

Arrangements are useful in solving motion planning problems. In this section we explain this connection and describe two tools that are based on arrangements for producing robust primitives for motion algorithms.

The first tool constructs exact polygonal Minkowski sums. Minkowski sums describe configuration space obstacles for translational motion planning and related problems. Following the typical CGAL approach robustness of the tool is achieved through the use of exact arithmetic. Moreover, we directly handle degeneracies; the solution depicted in Figure 1 exemplifies the handling of a degenerate configuration by our package.

The second tool computes *approximate* three-dimensional swept volumes, which describe the space occupied by a polyhedron as it is moved along a trajectory in space. Swept volumes have many applications including the verification that the motion of an object along a certain trajectory can be carried out without collision with the environment objects. In applications where very high precision is not required we propose a method which perturbs the surfaces that determine the swept volume so as to remove all possible degeneracies and which allows for computing robustly when using the standard floating-point arithmetic.

4.1 Exact Polygonal Minkowski Sums

Given two sets P and Q in $\mathbb{R}^2$, their *Minkowski sum* (or vector sum), denoted by $P \oplus Q$, is the set $\{p + q \mid p \in P, q \in Q\}$. Minkowski sums are used in a wide range of applications, including robot motion planning [68], assembly planning [52], and computer-aided design and manufacturing (CAD/CAM) [35].

Consider for example a planar setting with an obstacle P and a robot Q that moves by translation. We can choose a reference point r rigidly attached to Q and suppose that Q is placed such that the reference point coincides with the origin. If we let Q' denote a copy of Q rotated by 180° degrees, then $P \oplus Q'$ is the locus of placements of the point r where $P \cap Q \neq \emptyset$. In the study of motion planning the space of all possible placements of the reference point is called the *configuration space* of Q and this sum is called a *configuration space obstacle* because Q collides with P when translated along a path π exactly when the point r, moved along π, intersects $P \oplus Q'$.

We distinguish three types of points in the configuration space according to the placements of the robot that they represent: *free placements,* where the robot does not intersect any obstacle, *forbidden placements,* where the robot intersects the interior of an obstacle, and *semi-free placements,* where the robot is in contact with the boundary of an obstacle, but does not intersect the interior of any obstacle. The collection of all the points in the configuration space that represent semi-free robot placements partitions the configuration space into *free regions* and *forbidden regions.* In a motion-planning problem with two degrees of freedom, for instance, the points representing semi-free placements of the robot lie on several curves in a two-dimensional configuration space. We call these curves *constraint curves*. Note that each such curve is induced by the contact of a robot boundary feature and an obstacle boundary feature. We are therefore interested in studying the partitioning of 2D space by a collection of constraint curves and this is where motion planning and the study of arrangements meet. This discussion extends to motion planning with more degrees of freedom and higher dimensional arrangements and has been the subject of an intensive study; see e.g., [24],[50],[55],[83],[86].

There has been much work on obtaining sharp bounds on the size of the Minkowski sum of two sets in two and three dimensions, and on developing fast algorithms for computing Minkowski sums. If P is a polygonal set with m vertices and Q is another polygonal set with n vertices, then $P \oplus Q$ is a portion of the arrangement of $O(mn)$ constraint segments, where each segment is the Minkowski sum of a vertex of P and an edge of Q, or vice versa. Therefore the size of $P \oplus Q$ is $O(m^2n^2)$ and it can be computed within that time; this bound is tight in the worst case [64]; see Figure 5. The sum has lower worst-case complexity when one of the polygons or both are convex.

We devised and implemented three algorithms for computing the Minkowski sum of two polygonal sets [4],[40],[41]. Our main goal was to produce a *robust* and exact implementation. This goal was achieved by employing the CGAL planar map package (Section 3) while

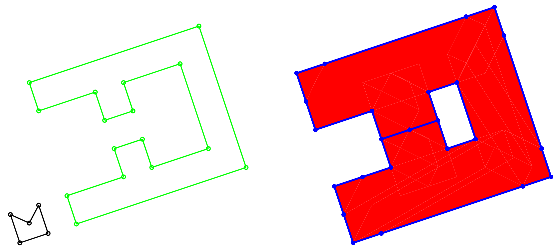


Figure 6: Tight passage: the desired target placement for the small polygon is inside the inner room defined by the larger polygon (left-hand side). In the configuration space (right-hand side) the only possible path to achieve this target passes through the line segment emanating from the hole in the Minkowski sum [41].

using exact number types. We are currently using our software to solve translational motion planning problems in the plane. We are able to compute collision-free paths even in environments cluttered with obstacles, where the robot could only reach a destination placement by moving through tight passages, practically moving in contact with the obstacle boundaries. See Figure 6 for an illustration. In motion planning lingo, our solution is *complete.* This is in contrast with most existing motion planning software for which tight or narrow passages constitute a significant hurdle. We remark that our tool can only handle translational motion; steps towards robust handling of polygon rotation are discussed in [25].

The input to our algorithms consists of two *polygonal sets* P and Q (each being an arbitrary collection of simple polygons), with a total of m and n vertices respectively. Our algorithms consist of the following three steps: **(1)** Decompose the polygons of P into convex subpolygons $P_1, P_2, \ldots, P_s$ and the polygons of Q into convex subpolygons $Q_1, Q_2, \ldots, Q_t$, **(2)** For each $i \in [1..s]$ and for each $j \in [1..t]$ compute the Minkowski *subsum* $P_i \oplus Q_j$, and **(3)** Construct the union of all the subsum polygons $P_i \oplus Q_j$ computed in Step 2; the output is represented as a planar map. The three algorithms differ in the way they compute the union (Step 3). For details on the algorithms and a comparison of their performance, see [40],[41]. Below we touch on two specific issues: (i) how degeneracies are treated, and (ii) what constitute good polygon decompositions (Step 1) for efficient construction of the sums.

4.1.1 Handling Degeneracies

The ability to discover the semi-free placement of Figure 1 or the tight passage of Figure 6 requires the program to identify the simultaneous intersection of many edges or the overlap of edges.

The simplest of the union algorithms (Step 3) is the so-called *arrangement* (union) algorithm. We next sketch the algorithm, demonstrate how degeneracies are treated by the algorithm and how CGAL facilitates this treatment.

Let $\mathcal{R}$ denote the collection of Minkowski subsums computes in Step 2 of the algorithm. The arrangement algorithm proceeds in two steps. First, we construct the arrangement $\mathcal{A}(\mathcal{R})$ of these polygons. Then, we traverse the features of the arrangement and for each vertex, edge, or face we decide whether it belongs to the Minkowski sum or to its boundary. The difficulties arise when we need to detect isolated semi-free features. We demonstrate next how we identify an isolated semi-free vertex. In our description we assume familiarity with the DCEL structure.

We keep for each face f its *inside count*, denoted $IC(f)$, which is the number of polygons of $\mathcal{R}$ in which it lies. We start from the unbounded face whose *inside count* is zero. We skip the details of how $IC(f)$ is updated and concentrate on vertices. A vertex is on the boundary of the union if it is an endpoint of an edge that is on the boundary of the union, or it may be disconnected from the rest of the boundary (an isolated vertex). We should know for each pair (v, f) of a vertex v and an incident face f on how many boundaries of polygons of $\mathcal{R}$ that contain f, v lies. Let us call this number the *slice count* of (v, f). Since each vertex can have many incident faces, an appropriate place to keep this information is the halfedge on the boundary of f that is targeted at v. Note that there is exactly one halfedge $e_{(v,f)}$ for which $face(e_{(v,f)}) = f$ and $target(e_{(v,f)}) = v$. We denote the *slice count* of the pair (v, f) by $SC(e_{(v,f)})$. Therefore, the following holds: *v is semi-free if and only if there is a face f such that* $[IC(face(e_{(v,f)})) - SC(e_{(v,f)})] = 0$. We maintain the *slice count* during the insertion of polygons of $\mathcal{R}$ into the map: If e_1 and e_2 are adjacent halfedges on the boundary of a polygon in $\mathcal{R}$, sharing a vertex v, we increase $SC(e)$ for each halfedge e that lies clockwise between e_1 and e_2 around v, and such that $target(e) = v$.

The full technical details of handling degeneracies are given in [40, pp. 53–60]. We believe that these procedures are valuable beyond the construction of Minkowski sums. Similar procedures will apply to handling degeneracies in other algorithms such as the computation of boolean operations of planar regions or the construction of *minimization diagrams* [51] describing the lower envelopes of surfaces in 3-space.

In what ways does CGAL facilitate the treatment of degeneracies? First, through the possibility to plug in exact number types in the traits classes. Second, the arrangement package allows for curve overlap; any number of curves can partly or fully overlap and one can iterate on all the curves that overlap a specific point of the arrangement. Also, it allows for the intersection of any number of curves at a point, and the traversal over these curves in clockwise or counterclockwise order (not only for segments but also for other curves, e.g., the circular arcs around the center point in Figure 3). Furthermore, in the maps and arrangements packages (as well as in other packages in CGAL) it is easy to augment vertices, (half)edges, and faces with extra information as is necessary in the procedure for detecting the semi-free vertices described above.

Since we deal here with line segments and polygons only it is not difficult to verify that we indeed treat all possible degeneracies that arise in the algorithm.

4.1.2 Efficient Decompositions

As the bound $\Theta(m^2n^2)$ indicates, even for input polygons with a moderate number of vertices the output can be huge. This together with our choice to use exact arithmetic necessitates speeding up the algorithm in other ways. We discovered that the choice of the decomposition in Step 1 can have a dramatic effect on the running time of the Minkowski-sum algorithms. (Notice that in the asymptotic complexity measures the choice of decomposition is meaningless: every reasonable decomposition, including an arbitrary triangulation, produces subpolygons of total complexity $O(k)$ where k is the complexity of the polygon.) This led us to investigate what constitute good decompositions for efficient construction of Minkowski sums [4].

We studied and experimented with various well-known decompositions as well as with several new decomposition schemes. Among our findings are that in general: (i) triangulations are too costly (although they can be produced quickly, they considerably slow down

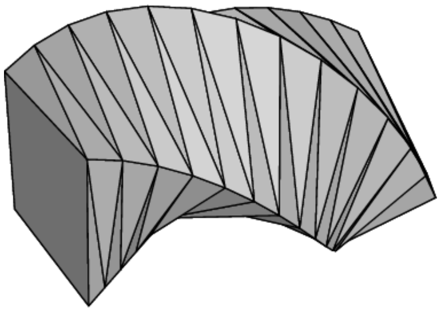


Figure 7: The volume swept by a square moving along a helical trajectory

the Minkowski-sum computation), (ii) what constitutes a good decomposition for one of the input polygons depends on the other input polygon—consequently, we developed a procedure for simultaneously decomposing the two polygons such that a "mixed" objective function is minimized, (iii) there are optimal decomposition algorithms that significantly expedite the Minkowski-sum computation, but the decomposition itself is expensive to compute — in such cases simple heuristics that approximate the optimal decomposition perform very well. The decomposition methods, the experiments and the conclusions drawn from them are reported in [4].

4.2 Degeneracy-Free Approximate Swept Volumes

A *swept volume* is the geometric space occupied by an object moving along a prescribed trajectory in a given time interval. The motion can be translational and rotational. We call the moving object a *generator* and its motion a *sweep*. See Figure 7 for an illustration.

Swept volumes play an important role in many geometric applications, such as geometric modeling, robot workspace computation, numerical control cutter path generation, and assembly design. There is rich literature on swept volumes which is beyond the scope of this paper to review; see, e.g., [1] for a bibliography.

Consider the following question: Given an assembly of parts, a specific part P in the assembly, and a potential path π for removing P out of the assembly—can P be moved along π without colliding with the other parts of the assembly? We can answer this question by computing the swept volume of P along π and check it for collision with the other parts. We were specifically motivated by problems in large assemblies and manual removal of parts for maintenance and repair.

We chose to implement an algorithm proposed by Abrams and Allen [2] which handles polyhedral bodies moving along an arbitrary trajectory in a motion that can be translational and rotational, and outputs a polyhedral approximation of the appropriate swept volume; we will refer to this algorithm as ASW (approximate swept volume). The algorithm generates a set $\mathcal{P}$ of polyhedral surfaces such that the boundary of the desired swept volume is the boundary of the outer cell of the arrangements $\mathcal{A}(\mathcal{P})$. (Notice that ASW (i) ignores voids, and (ii) is not conservative, namely the approximate swept volume does not necessarily contain the exact swept volume; for details see [2].) Our overall plan was: (i) use ASW to produce the set $\mathcal{P}$ and then (ii) use an implementation of the space sweep algorithm [28], adapted from triangles to polyhedral surfaces, to compute the outer cell of $\mathcal{A}(\mathcal{P})$.

However, the algorithm described in [28] assumes general position (its implementation [87] already has to handle a large number of cases even when assuming general position). Abrams and Allen also report that they could not find robust software for arrangements that would lead to a robust implementation of their solution [2]. Unlike two-dimensional arrangements where directly handling degeneracies is a satisfying solution (as in Section 4.1 above), full treatment of degeneracies in three-dimensional arrangements is an arduous task. Moreover, because of the nature of the problem at hand and the approximate solution we have chosen for it, the recognition of all the degeneracies in itself is irrelevant. Identification of all the degeneracies is only meaningful as a way to extract a consistent topology of the outer cell boundary—but we will obtain the same goal in a much simpler way. We apply a 'controlled' perturbation to the surfaces in $\mathcal{P}$ so that all degeneracies are removed [78],[79]. One of our main goals here is to allow to manipulate the swept volume robustly with standard floating-point arithmetic. We describe our scheme next.

Since we aim to use standard floating-point arithmetic, we are unable to tell for sure whether a degeneracy exists. We can only tell that a *potential degeneracy* exists. To define such a potential degeneracy formally, we use a *resolution parameter*, $\varepsilon > 0$, which is a small positive real number. Two polyhedral features (e.g., a vertex and a non-incident facet) are assumed too close (and therefore potentially degenerate) whenever they are less than ε away from each other. We assume that ε is given as an input parameter according to the machine precision, the type of the arithmetic operations and the depth of the *expression tree* [81]. In our algorithm this tree's depth is a small constant, thus ε is a constant that can be determined and bounded independently of the combinatorial input size, namely independently of the overall number of features (vertices, edges and faces) in the input polyhedral surfaces.

Next we define a perturbation radius δ, which is proven to be sufficiently small (δ depends on ε and the input polyhedral surfaces). Any polyhedral surface in $\mathcal{P}$ or any feature thereof will be perturbed by at most δ. Specifically, each vertex of any surface in $\mathcal{P}$ is moved by a Euclidean distance of at most δ from its original placement.

Our perturbation scheme is 'controlled' in two ways. First, by determining the size of δ we set a tradeoff that controls the magnitude of the perturbation versus the efficiency of the computation of the perturbation— the larger the δ the faster the computation. Second, unlike various popular heuristic perturbation schemes (e.g., 'heuristic epsilons' [81]), our perturbation guarantees that the resulting collection of polyhedral surfaces is degeneracy free by carrying out a controlled incremental insertion process where we do not proceed to the next iteration before the arrangement induced by the subcollection of surfaces or subsurfaces of $\mathcal{P}$ we inserted so far is degeneracy free. Successful completion of the process is guaranteed for δ values above a threshold which is determined by the analysis of the procedure.

The thesis [77] contains a detailed report on all the degeneracies arising in arrangements of polyhedral surfaces and the swept volume computation, gives bounds on the perturbation radius as a function of ε and the input polyhedral surfaces, and describes the implementation of the scheme together with experimental results.

Our implementation of the scheme uses CGAL in various ways (geometric primitives, predicates, affine transformations, assertions and preconditions testing). Most notably it uses the *polyhedral surface* package developed by Kettner [66]. The availability of a large number of well-tested tools in a uniform framework considerably eased our development task.

This work is an extension of an earlier work on arrangements of spheres as they arise in molecular modeling [56]. In both cases (molecular modeling and swept volumes) we show by experiments [56], [79] that also on highly degenerate input (as illustrated in Figure 8) the controlled perturbation scheme works efficiently even when δ is fairly small.

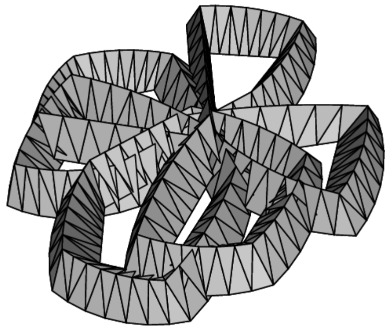


Figure 8: Swept volume with many intersections (and potential degeneracies) in the middle [78]

5 Conclusions

We described advancement in *robust* implementation of geometric algorithms. After reviewing the CGAL project and library we concentrated on the maps and arrangements part of CGAL and showed how this and other parts of CGAL are useful for devising robust primitives for motion planning.

Regarding arrangements, the obvious next goal would be to develop packages for three- and higher dimensional arrangements comparable in their robustness and generality to the two-dimensional package described in Section 3. A major obstacle to achieving this goal is the treatment of degeneracies (as far as precision is concerned we could use exact arithmetic wherever possible). We mention next several possible directions to confront degeneracies in arrangements. Although we point out the difficulties and shortcomings of each direction we only mention directions which we believe are feasible, perhaps in restricted form (that is, only in three dimensions or only for arrangements of certain simple types of objects).

- Certifying a correct result only for input in general position. This is feasible (in the sense that asserting general position is easier than computing a degenerate arrangement) but impractical since quite often the input of physical-world problems is not in general position.

- Directly handling all degeneracies (as we do in the two-dimensional case). This seems to be a tremendous task already for arrangements of triangles in three dimensions. What is missing is a systematic and concise way to express and compute the topology of the arrangement at the neighborhood of degeneracies. This could possibly be achieved by employing lower dimensional arrangements at such neighborhoods.
- Applying symbolic perturbation (see the Introduction). In this solution the difficulty seems to be in the interpretation of the output ('postprocessing') [81],[85].
- Tightly approximating the arrangement, possibly using only fixed precision arithmetic (as in Section 4.2). There are many conceivable ways to achieve this goal, one of which is extending the scheme that we propose above, controlled perturbation, to other objects and to higher dimensions. Implementing the scheme is not difficult. However giving a theoretical guarantee for its viability for arrangements of complex objects is a difficult task [77].

As for motion planning, supporting only two- and (currently partially) three-dimensional arrangements means we could solve robustly and accurately problems with a limited number of degrees of freedom. Progress in the implementation of algorithms for higher-dimensional arrangements would result in robust algorithms for systems with more degrees of freedom. Notably, probabilistic roadmap (PRM) techniques already offer a solution for motion planning problems with many degrees of freedom [12],[65]. However, densely cluttered environments and narrow passages in the workspace are difficult for PRM techniques [60],[92]; in fact tight passages are impossible to identify without exact predicates. We anticipate that hybridizing the two approaches, namely PRMs and exact (or tightly approximate) arrangements, will lead to stronger motion planning algorithms.

Finally, notice that CGAL offers much additional functionality that could be useful for algorithms in robotics and motion planning, including convex hulls, Voronoi diagrams, triangulations of point sets, various optimization algorithms (e.g., smallest enclosing sphere) and multi-dimensional search structures.

Acknowledgement

The work described in this paper was done by or has been made possible by the efforts of many individuals—the participants of the CGAL and GALIA projects. The full list of participating sites is given at the beginning of Section 2.

The contributions at the Tel Aviv site which are described in Sections 3 and 4 have been made by Eti Ezra, Eyal Flato, Iddo Hanniel, Shai Hirsch, Chaim Linhart, Eugene Lipovetsky, Oren Nechushtan, Sigal Raab, Hayim Shaul, and Ron Wein. Also contributed at Tel Aviv: Sariel Har-Peled, Eli Packer, and Michal Ozery. The author also thanks Stefan Schirra and Lutz Kettner for their comments on an earlier draft of this paper, and the anonymous referees for helpful reviews.

Work reported in this paper has been supported in part by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces), by ESPRIT IV LTR Projects No. 21957 (CGAL) and No. 28155 (GALIA), by the USA-Israel Binational Science Foundation, by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), by a Franco-Israeli research grant "factory of the future" (monitored by AFIRST/France and The Israeli Ministry of Science), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

References

[1] K. Abdel-Malek. Swept volumes: Bibliography. www.icaen.uiowa.edu/~amalek/sweep/bibliog.htm.

[2] S. Abrams and P. Allen. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proc. IEEE Intl. Sympos. on Assembly and Task Planning*, pages 188–193, 1995.

[3] P. Agarwal and M. Sharir. Arrangements. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

[4] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. In *Proc. 8th European Symposium on Algorithms*, volume 1879 of *Lecture Notes in Computer Science*, pages 20–31. Springer-Verlag, Saarbrücken, 2000. To appear in *Computational Geometry: Theory and Applications*.

[5] P. K. Agarwal, M. Katz, and M. Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom. Theory Appl.*, 5:187–206, 1995.

[6] Y. Aharoni, D. Halperin, I. Hanniel, S. Har-Peled, and C. Linhart. On-line zone construction in arrangements of lines in the plane. In *Proc. of the 3rd Workshop of Algorithm Engineering*, volume 1668 of *Lecture Notes Comput. Sci.*, pages 139–153. Springer-Verlag, 1999. Full version: www.cs.tau.ac.il/~danha/papers/onlinezone.ps.gz.

[7] N. Amenta. Directory of computational geometry software. www.geom.umn.edu/software/cglist/.

[8] N. Amenta. Computational geometry software. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 52, pages 951–960. CRC Press LLC, Boca Raton, FL, 1997.

[9] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM J. Comput.*, 13(4):865–877, 1984.

[10] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition II: The adjacency algorithm for the plane. *SIAM J. Comput.*, 13(4):878–889, 1984.

[11] M. Austern. *Generic Programming and the STL — Using and Extending the C++ Standard Template Library.* Addison-Wesley, 1999.

[12] J. Barraquand, L. E. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. *Internat. J. Robot. Res.*, 16(6):759–774, 1997.

[13] K.-F. Böhringer, B. Donald, and D. Halperin. On the area bisectors of a polygon. *Discrete Comput. Geom.*, 22:269–285, 1999.

[14] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Upper and lower bounds for programmable vector fields with applications to MEMS and vibratory plate part feeders. In J.-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 255–276. A. K. Peters, Wellesley, MA, 1996.

[15] J.-D. Boissonnat and F. P. Preparata. Robust plane sweep for intersecting segments. Report TR 3270, INRIA, Sophia Antipolis, Sept. 1997.

[16] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. *Comput. Geom. Theory Appl.*, 16(1):35–52, 2000.

[17] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, UK, 1998.

[18] H. Brönniman, L. Kettner, S. Schirra, and R. Veltkamp. Applications of the generic programming paradigm in the design of CGAL. Technical Report MPI-I-98-1-030, Max-Planck-Insitut für Informatik, 66123 Saarbrücken, Germany, 1998.

[19] H. Brönnimann, I. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theoret. Comput. Sci.*, 210(1):173–197, 1999. Special Issue on Real Numbers and Computers.

[20] H. Brönnimann and M. Yvinec. Efficient exact evaluation of signs of determinants. *Algorithmica*, 27:21–56, 2000.

[21] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Efficient exact geometric computation made easy. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 341–350, 1999.

[22] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000.

[23] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 16–23, 1994.

[24] J. Canny. *The Complexity of Robot Motion Planning.* ACM – MIT Press Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1987.

[25] J. Canny, B. R. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 251–260, 1992.

[26] *The CGAL User Manual, Release 2.3*, 2001. www.cgal.org.

[27] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, Oct. 1992.

[28] M. de Berg, L. J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete Comput. Geom.*, 15:35–61, 1996.

[29] M. de Berg, D. Halperin, M. Overmars, and M. van Kreveld. Sparse arrangements and the number of views of polyhedral scenes. *Internat. J. Comput. Geom. Appl.*, 7:175–195, 1997.

[30] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 294–303, 1997.

[31] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Heidelberg, Germany, 1997.

[32] O. Devillers and F. P. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete amd Computational Geometry*, 20:523–547, 1998.

[33] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[34] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[35] G. Elber and M.-S. Kim, editors. *Special Issue of Computer Aided Design: Offsets, Sweeps and Minkowski Sums*, volume 31. 1999.

[36] I. Z. Emiris, J. F. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19(1–2):219–242, Sept. 1997.

[37] A. Fabri, G. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. The CGAL kernel: A basis for geometric computation. In M. C. Lin and D. Manocha, editors, *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 191–202. Springer-Verlag, 1996.

[38] A. Fabri, G. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, the Computational Geometry Algorithms Library. *Software - Practice and Experience*, 30:1167–1202, 2000.

[39] P.-O. Fjällström, J. Petersson, L. Nilsson, and Z. Zhong. Evaluation of range searching methods for contact searching in mechanical engineering. *Internat. J. Comput. Geom. Appl.*, 8:67–83, 1998.

[40] E. Flato. Robust and efficient construction of planar Minkowski sums. Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2000. http://www.cs.tau.ac.il/∼flato.

[41] E. Flato and D. Halperin. Robust and efficient construction of planar Minkowski sums. In *Abstracts 16th European Workshop Comput. Geom.*, pages 85–88, Eilat, 2000.

[42] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *The ACM Journal of Experimental Algorithmics*, 5, 2000. Also in LNCS Vol. 1668 (WAE '99), Springer, pp. 154–168.

[43] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[45] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 264–273, 2001.

[46] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press LLC, Boca Raton, FL, 1997.

[47] M. Goodrich, L. J. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 284–293, 1997.

[48] T. Granlund. *GNU MP, The GNU Multiple Precision Arithmetic Library, Edition 3.1.1*, Sept. 2000.

[49] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 143–152, 1986.

[50] D. Halperin. Robot motion planning and the single cell problem in arrangements. *Journal of Intelligent and Robotic Systems*, 11:45–65, 1994.

[51] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[52] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.

[53] D. Halperin and M. H. Overmars. Spheres, molecules and hidden surface removal. *Comput. Geom. Theory Appl.*, 11:83–102, 1998.

[54] D. Halperin and E. Packer. Iterated snap rounding. To appear in *Computational Geometry: Theory and Applications.* A preliminary version appeared in *Abstracts 17th European Workshop Comput. Geom.*, Berlin 2001.

[55] D. Halperin and M. Sharir. Arrangements and their applications in robotics: Recent developments. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 495–511. A. K. Peters, Wellesley, MA, 1995.

[56] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[57] I. Hanniel. The design and implementation of planar arrangements of curves in CGAL. M.Sc. thesis, Dept. Comput. Sci., Tel Aviv University, Tel Aviv, Israel, 2000.

[58] I. Hanniel and D. Halperin. Two-dimensional arrangements in CGAL and adaptive point location for parametric curves. In *Proc. of the 4th Workshop of Algorithm Engineering*, volume 1982 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 2000.

[59] J. Hobby. Practical segment intersection with finite precision output. *Comput. Geom. Theory Appl.*, 13:199–214, 1999.

[60] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Proc. 3rd Workshop Algorithmic Found. Robot.*, pages 141–153, Wellesley, MA, 1998. A. K. Peters.

[61] W. D. Jones. Computational geometry, a community bibliography. Overview at compgeom.cs.uiuc.edu/~jeffe/compgeom/.

[62] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *15th ACM Symp. on Computational Geometry, 1999*, pages 351–359, 1999.

[63] M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulations using rational arithmetic. *ACM Trans. Graph.*, 10(1):71–91, Jan. 1991.

[64] A. Kaul, M. A. O'Connor, and V. Srinivasan. Computing Minkowski sums of regular polygons. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 74–77, Aug. 1991.

[65] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. and Autom.*, 12:566–580, 1996.

[66] L. Kettner. Designing a data structure for polyhedral surfaces. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 146–154, 1998.

[67] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: a library for efficient manipulation of algebraic points and curves. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 360—369, 1999. http://www.cs.unc.edu/~geom/MAPC/.

[68] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[69] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2001.

[70] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. *The LEDA User Manual, Version 4.1.* Max-Planck-Insitut für Informatik, 66123 Saarbrücken, Germany, 1999.

[71] K. Melhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[72] V. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. In D. Kapur and J. L. Mundy, editors, *Geometric Reasoning.* North-Holland, Amsterdam, Netherlands, 1988.

[73] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[74] M. Neagu and B. Lacolle. Computing the combinatorial structure of arrangements of curves using polygonal approximations. In *Abstracts 14th European Workshop Comput. Geom.*, pages 121–123, 1998.

[75] M. H. Overmars. Designing the Computational Geometry Algorithms Library CGAL. In *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 113–119. Springer-Verlag, May 1996.

[76] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[77] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. M.Sc. thesis, Dept. Comput. Sci., Bar Ilan University, Ramat Gan, Israel, 1999.

[78] S. Raab. Controlled perturbation of arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1999.

[79] S. Raab and D. Halperin. Controlled perturbation of arrangements of polyhedral surfaces with application to swept volumes. Full version, manuscript, Tel Aviv University, 2001.

[80] J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry.* North-Holland, 1999.

[81] 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.

[82] P. Schorn. *Robust algorithms in a program library for geometric computation.* Ph.D. thesis, ETH Zürich, Switzerland, 1991. Report 9519.

[83] J. T. Schwartz and M. Sharir. On the "piano movers" problem II: General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.*, 4:298–351, 1983.

[84] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, 1991.

[85] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19:1–17, 1998.

[86] M. Sharir and P. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, 1995.

[87] H. Shaul. Improved output-sensitive construction of vertical decompositions of triangles in three-dimensional space. M.Sc. thesis, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, 2001.

[88] J. Shewchuk. Adaptive robust floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18:305–363, 1997.

[89] K. Sugihara. On finite-precision representations of geometric objects. *J. Comput. Syst. Sci.*, 39:236–247, 1989.

[90] A. F. van der Stappen. *Motion Planning amidst Fat Obstacles.* Ph.D. dissertation, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1994.

[91] A. F. van der Stappen, D. Halperin, and M. H. Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom. Theory Appl.*, 3:353–373, 1993.

[92] S. Wilmarth, N. Amato, and P. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. In *15th ACM Symp. on Computational Geometry, 1999*, pages 173–180, 1999.

[93] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40(1):2–18, 1990.

[94] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997.

[95] C. K. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.