TRACINGS AND THEIR CONVOLUTION: THEORY AND APPLICATIONS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

G.D. Ramkumar

March 1998

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas Guibas
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Lyle Ramshaw

Approved for the University Committee on Graduate Studies:

Abstract

The set of placements of a robot that collide with an obstacle is referred to as a C-space obstacle; in the case where the robot is allowed only to translate, it is typically modeled via a Minkowski sum. This thesis considers an alternative representation for it called the convolution that is smaller and easier to compute. We limit discussion in this thesis to (possibly non-convex) polygons and polyhedra.

In the planar case, the convolution operation was formulated in a seminal result of Guibas, Ramshaw, and Stolfi in 1983. They defined the operation for possibly self-intersecting boundaries of planar shapes called tracings. The first chapter of the thesis introduces the topic, discusses previous work, and presents an outline of the thesis.

In the second chapter, we present an algorithm that uses the convolution of two simple polygonal tracings to compute the outer face of their Minkowski sum. This algorithm exploits a new insight into the relationship between convolution cycles and the Minkowski sum. Although this algorithm is asymptotically slower than a general single face computation, it is practical and performs well for realistic polygon data.

In the third chapter, we define polyhedral tracings, three-dimensional extensions of polygonal tracings. Polyhedral tracings can be stored in a half-edge data structure for polyhedral boundaries enhanced with sign information. We define the operation of convolution between polyhedral tracings. The convolution is expressed similarly as a signed

half-edge data structure. We show that the half-edge structure of the convolution represents a valid set of closed polyhedral boundaries.

The fourth chapter discusses an algorithm to compute the convolution of polyhedral tracings efficiently. This algorithm makes use of a new method known as HeapSweep to overlay two planar subdivisions (one red and one blue) on the surface of a sphere and determine the bichromatic intersections without tracking all monochromatic intersections. This algorithm enables us to compute all features of the convolution in output-sensitive time with only poly-logarithmic overhead. The fifth chapter concludes the thesis.

Acknowledgements

First and foremost, I would like to thank my advisor Leonidas Guibas for his patient guidance and support throughout my graduate student years. Without his encouragement and assistance, this thesis would not be possible. I am deeply grateful to Julien Basch for being a colleague, a real friend, and a source of great inspiration; our collaboration led to integral parts of this thesis.

Special thanks to Rajeev Motwani for motivating advice and inspiration during and prior to this thesis work. My sincere thanks to Lyle Ramshaw for his invaluable advice on the conceptual and written content of the thesis. I am deeply indebted to thesis committee member Jean-Claude Latombe for providing resources for my work and for establishing a great academic environment at the Robotics Laboratory.

I would like to take this opportunity to thank our lab administrator Jutta Mccormick for her encouragement. Thanks to Carolyn Tajnai for her kind words. Many thanks are due to John Mitchell and Arthur Veinott for graciously agreeing to be on my orals committee.

Many friends made my stay at Stanford worthwhile — these include Anchal Jain, Anna Geevarghese, Ashish Gupta, Avijit Goswami, Danny Halperin, Dinesh Katiyar, Eric Veach, Frederic Cazals, Kim Roberts, Lydia Kavraki, Michael Goldwasser, Radhika Shah, Ranjit Konkar, Robert Kennedy, Sadia Najmi, Shyam Prabhakar, and Vivek Narayanan.

I would like to thank Arunashree for bringing fresh air into my life. Finally, I am deeply indebted to my brother, sister, mom, and dad for always being there.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computational Geometry is the study of the representation and manipulation of geometric objects in the computer [114, 127]. Discrete geometric computations are made possible using combinatorial descriptions of objects. The identification of relevant combinatorial structures, geometric predicates, and the design of algorithms that use these descriptions and predicates form the core of this discipline.

As the field of computational geometry evolved, it found applications in various applied fields. One of the key application areas that is a source of research problems in geometry is Robotics. For a robot R to move between an initial position and a goal, it is necessary to devise a motion plan that avoids collision between the robot and obstacles in the *workspace* [93]. The robot may be allowed to translate or rotate or has different types of joints imposing sliding or rotating constraints. The set of possible placements of the robot is a manifold whose dimension is the number of degrees of freedom of the robot. This set is called the *configuration space* of the robot. Amongst all possible placements, the *free space* is the set of placements of the robot for which it doesn't collide with any obstacle. The set of placements for which the robot collides with a given obstacle B is called the *C-obstacle* associated with B.

Hence, planning the motion of a robot in the workspace is equivalent to finding a path in its free space. We can solve this problem by decomposing the free space into “simple” cells, and create a graph of adjacencies between the cells. Then, in order to devise a plan, we (a) locate in which cells of the free space the initial and final placements of the robot lie, (b) find a path from the initial to the final placement in the graph of adjacencies, (c) devise an exact path in each cell. The definition of a “simple cell” is to make this latter step easy (for instance, if a cell is convex, a straight path gives a valid motion).

In the case of a robot allowed only to translate in two dimensions, the configuration space is the two-dimensional plane: a point (x, y) in the configuration space represents a placement of the robot translated by (x, y) from an arbitrary resting position. In this case, the C-obstacle associated to an obstacle B can be obtained by performing an operation called the *Minkowski sum* of the obstacle and the robot reflected through the origin. This operation is a natural set operation that has been used in several fields (see below for a review).

The Minkowski sum carries limited algebraic structure, and its definition is not adequate for computational purposes. Indeed, in order to perform discrete computations on infinite sets, it is necessary to obtain and manipulate only the combinatorial description of these sets. For instance, we represent a simple polygon by a circular list of its vertices: in this way, we have a finite, compact representation for the infinite set of points contained in the polygon, and this finite representation is amenable to the tools of discrete algorithms. In higher dimensions, a polytope will be typically represented as a *cell complex*, i.e. as a finite set of vertices, edges, and higher-dimensional simplices, together with some gluing information. Sometimes, as in the case of the polygon described above, only the *boundary* of the set is thus decomposed; this is the *boundary description* of the set. In three dimensions, several boundary descriptions have been proposed [73, 27].

Let us consider the two-dimensional case: it is easy to compute the boundary description of the Minkowski sum of two convex polygons, given their respective boundary descriptions. It is obtained by sorting the slopes of the edges of the two polygons, and matching vertices that overlap edges of the other polygon in this ordering. However, for more general polygons, it is not possible to directly define the Minkowski sum in terms of the boundaries of the polygons: a full cell decomposition, or a decomposition into convex polygons, is needed. The Minkowski sum has to be computed between all pairs of cells, and this introduces a quadratic cost in the computation. In order to address this problem, Guibas, Ramshaw and Stolfi [70] define an operation called *convolution* between polygon boundaries. More precisely, they define a class of objects called *tracings*, that includes boundaries of polygons, and show that this class equipped with the convolution operation has a rich algebraic structure that is also well suited for computational purposes. They also show how the convolution operation can be used as an intermediary step to compute the Minkowski sum between two polygons efficiently. The second chapter of this thesis presents a novel way in which the two-dimensional convolution can be used to compute important parts of the Minkowski sum.

Schapira [121] generalizes the idea of convolution to sets in spaces of arbitrary dimensions. He casts the operation as an algebraic operation on multi-sets. His work spells out the structure of the convolution, and clarifies its relation with the notion of free space for a translating robot. However, as it is based on decomposition of sets, it is not directly amenable to any type of computation based on boundary descriptions. The third chapter of this thesis defines a generalization of tracings to three dimensions (including the set of polyhedron boundaries) and defines the convolution in terms of boundary descriptions. The fourth chapter presents an efficient algorithm to compute it.

The convolution computation arises in the following specific application: projecting the convolution of two polyhedral tracings onto the sphere of directions; such an overlay is called a direction cone. This application arises in the context of automatic assembly of products [118] consisting of polyhedral parts. Assembly of parts using path planning algorithms was considered in [91], assembly of toleranced parts was studied in [94]. Schweikard et al. [126] considered assembly sequences of polyhedra.

1.1 Background

In the rest of this thesis, we use the following notation: the d-dimensional Euclidian space is denoted by $\mathcal{E}^d$, or by $\mathcal{E}$ if the dimension is clear from the context. The d-dimensional sphere is denoted by $\mathcal{S}^d$. For a set $A \in \mathcal{E}$, we denote by $-A$ the reflection of A through the origin, and by $A + x$ the set of all points of A translated by x, that is to say:

$$\begin{aligned} -A &= \{-a \mid a \in A\}, \\ A + x &= \{a + x \mid a \in A\}. \end{aligned}$$

We denote by 1_A the *characteristic function* (or *indicator*) of a set A, i.e. the function whose value is 1 on A and 0 elsewhere. In the rest of this thesis, we assume that objects are in general position unless specified otherwise. For instance, given a pair of polygons, we assume that no two edges are parallel.

1.1.1 Minkowski sum

Definition 1.1. *Let A, B be subsets of $\mathcal{E}^d$. The* Minkowski sum *of A and B, denoted $A \oplus B$ is defined as:*

$$A \oplus B = \{a + b \mid a \in A, b \in B\}.$$

The *Minkowski difference* of A and B, denoted $A \ominus B$, is the Minkowski sum of A and $-B$. There is an alternate way to define the Minkowski difference.

Proposition 1.2. *Let A, B be subsets of a vector space $\mathcal{E}^d$. Then the Minkowski difference of A and B is the set:*

$$A \ominus B = \{r \in \mathcal{E} \mid A \cap (B + r) \neq \emptyset\}.$$

Proof. Let $r \in \mathcal{E}$ such that $A \cap (B + r) \neq \emptyset$. That is, there exists $p \in \mathcal{E}$ such that $p \in A$ and $p \in B + r$, i.e. $p - r \in B$. Hence, by definition, $p - (p - r) \in A \ominus B$, or $r \in A \ominus B$.

Conversely, if $r \in A \ominus B$, then there exist $a \in A$ and $b \in B$ such that $r = a - b$. Hence, $a \in A$ and $a = b + r \in B + r$, implying that $A \cap (B + r) \neq \emptyset$. $\square$

With this alternate definition, we have the following corollary:

Corollary 1.3. *Let R be a robot allowed only to translate in a workspace $\mathcal{E}$. The C-obstacle corresponding to an obstacle B is the Minkowski difference $B \ominus R$.*

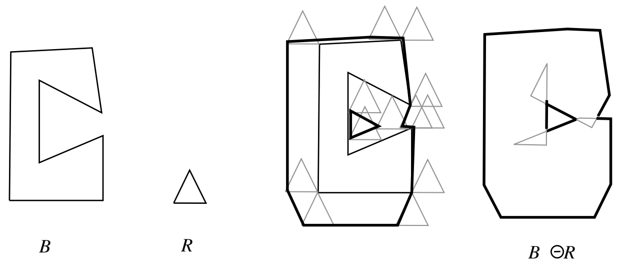

Figure 1.1: The Minkowski difference $B \ominus R$ between an obstacle B and a robot R is the set of placements of R at which it intersects with B.

Hence, in the situation of a robot allowed only to translate in the workspace, the free space can be computed by taking the complement of the union of the Minkowski difference of each obstacle with the robot. The above definition, however, does not give any indication on how to compute the Minkowski difference.

1.1.2 Complexity of representation and computation

For convex sets, we have the following property:

Proposition 1.4. *The Minkowski sum of two convex sets is convex.*

Proof. Let $x, y \in B \oplus R$, where B and R are convex sets, i.e. $x = b + r$ and $y = b' + r'$, with $b, b' \in B$, and $r, r' \in R$. Then for any α, β with $\alpha + \beta = 1$, we have:

$$\alpha x + \beta y = \alpha b + \beta b' + \alpha r + \beta r'.$$

Hence, the segment $[x, y]$ is in the Minkowski sum. $\square$

This immediately gives a way to compute the Minkowski sum of two convex polygons given their boundary representation. It is enough to find the extremal point in each direction and this extremal point is simply the sum of the extremal points of the two input sets. Hence, to compute the Minkowski sum of two convex polygons, we take the two circular lists of edge slopes (both oriented in counter-clockwise order) on $\mathcal{S}^1$ and merge them. Each arc in the merged order gives one vertex of the ouput polygon. This operation takes linear time if the input polygons are given by the circular list of their vertices.

More generally, the boundary representation of a convex polytope in $\mathcal{E}^d$ defines a convex subdivision on $\mathcal{S}^{d-1}$, where each feature of dimension k in the original polytope gives a feature of dimension $d-1-k$ in the subdivision of $\mathcal{S}^{d-1}$. The computation of the Minkowski sum reduces to the computation of the overlay of the two convex subdivisions, which can be done in optimal output-sensitive time in two dimensions. This solves the 3-dimensional problem for convex sets (but note that the complexity of the resulting polyhedron can be quadratic):

Theorem 1.5. *[71] Let P, Q be two convex polyhedra in $\mathcal{E}^3$ represented by their boundaries with a total of n features. If their Minkowski sum has k features, it can be computed in*

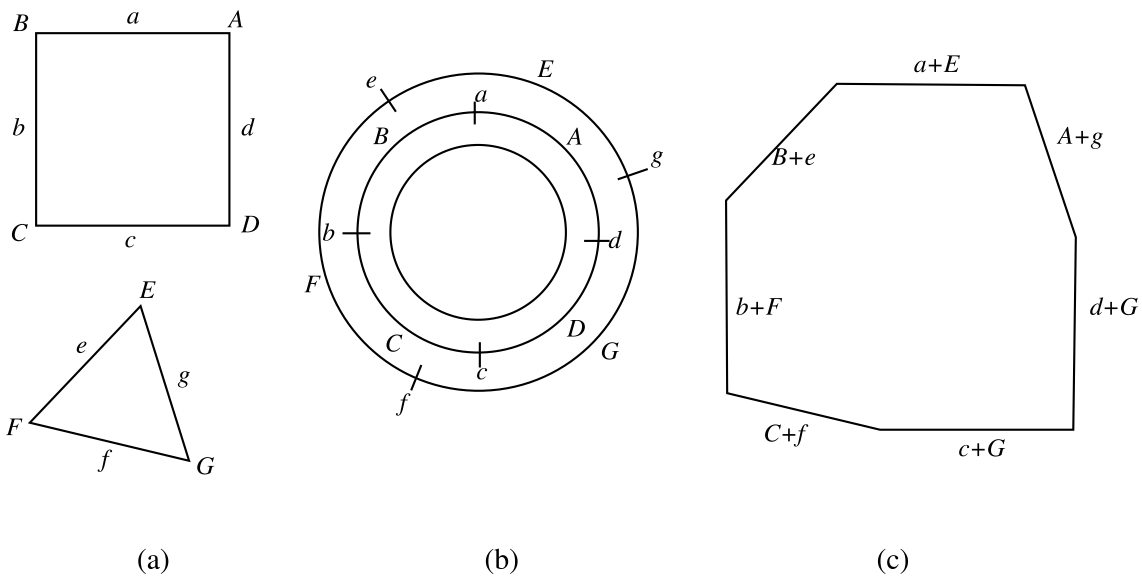

Figure 1.2: (a) Two convex polygons; (b) the orientations spanned by each feature on $\mathcal{S}^1$; (c) Their Minkowski sum.

optimal time $\Theta(n+k)$.

Let us consider what happens if we relax the convexity condition. Consider two simple polygons: the situation becomes dramatically different. Given two input polygons of size m, n let us look at the worst-case asymptotic size of the Minkowski sum. If the two polygons are convex, then the Minkowski sum has boundary of size $O(m+n)$ (each vertex of the first polygon can pair with one edge of the second and vice versa). The simple polygon case is harder, first, because we have to consider all mn pairs of vertices to get new vertices of the obvious type; and, second, because there can be many more additional vertices (Figure 1.3). In the general case:

Lemma 1.6. *[67] Let P, Q be two simple polytopes in $\mathcal{E}^d$ with m and n vertices respectively. Their Minkowski sum is of size $\Theta(m^d n^d)$ in the worst case.*

Note that in Figure 1.3, the resulting polygon has holes (it is impossible to move from one placement to another placement). If one focuses on a single face of the Minkowski sum

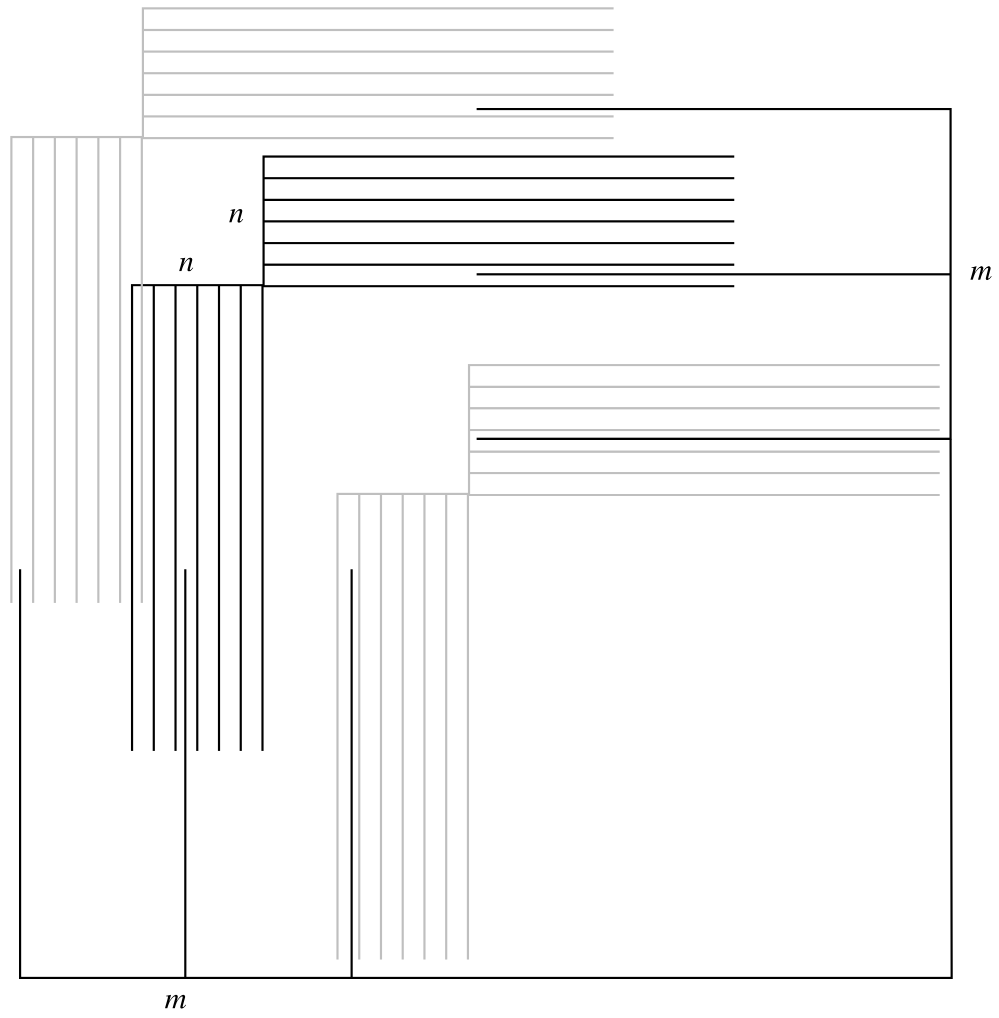

Figure 1.3: Two simple polygons of size $\Theta(m)$ and $\Theta(n)$. It is possible to place any of the large poles in any of the fine holes. The number of independent placements is $\Theta(m^2n^2)$. So the Minkowski difference $P \ominus Q$ has $O(m^2n^2)$ separate components.

complement, general results on the complexity of a single face in an arrangement of line segments [72] show that the worst-case complexity of a face is $O((m+n)\alpha(m+n))$, where α is the inverse of Ackermann's function. This bound is tight [78].

1.1.3 Paintings and their convolution

The set of polyhedral sets is closed under the Minkowski sum operator. We therefore have a semi-group, but that's the only algebraic structure we can obtain with this operator. A much richer structure can be obtained by representing sets as their characteristic functions, and generalizing characteristic functions to integer valued functions with certain regularity properties.

For the purpose of this thesis, we restrict our discussion to piecewise linear objects in three dimensions.

Definition 1.7. *A painting F is an integer-valued function on $\mathcal{E}^d$ such that F is non-zero in a bounded set and $\mathcal{E}^d$ can be partitioned into finitely many subsets of affine subspaces on which F has constant integral value.*

In practice, a painting can be represented by a cell decomposition of space, with an integer associated to each component.

Example 1.8. *A polytope can be represented by its characteristic function, i.e. the function whose value is* 1 *on points in the set and* 0 *elsewhere. The choice of the value on the boundary allows to represent a topologically open or closed set.*

The addition operator '+' of characteristic functions defines a group whose zero element is the identically 0 painting, i.e. the characteristic function of the empty set. The set of paintings along with the + operation is a group. Paintings can also be multiplied pointwise, which corresponds, in the case of characteristic functions, to the operation of intersections on the underlying sets.

Definition 1.9. *A standard decomposition of a painting F is a finite family $(C_i, \alpha_i)_i$ such that for all i, C_i is a compact contractible set, and*

$$F = \sum_i \alpha_i 1_{C_i}.$$

Theorem 1.10. *[121] Given a painting F and a standard decomposition (C_i, α_i), the quantity $\sum_i \alpha_i$ is called the integral of F and denoted $\int F$. It is independent of the decomposition.*

Example 1.11. *We can compute the integral of the characteristic function of the d-dimensional sphere. Let us call this quantity c_d. This function can be decomposed as follows: add two closed hemispheres and subtract their intersection, which is a $d-1$-sphere. Hence we have:*

$$c_d = 2 - c_{d-1},$$

and $c_1 = 2$ for the 0-dimensional sphere which is made of two points.

This example shows that some natural paintings (d-spheres for odd d) can have a 0 integral even when they are not empty.

Lemma 1.12. *[121] The integral of the characteristic function of a closed set is its Euler-Poincaré characteristic, i.e. the alternating sum of the number of elements of each dimension in a cell decomposition of the set.*

In this thesis we refer to the operation of convolution on paintings as the painting convolution; if the operands are clearly paintings from the context, we refer to it as the convolution. We are now ready to define the operation formally.

Definition 1.13. *The painting convolution $F * G$ of two paintings is defined by:*

$$(F * G)(x) = \int \left(\lambda y \cdot F(y) G(x-y)\right).$$

Lemma 1.14. *The convolution of two paintings is a painting.*

Lemma 1.15. *The convolution is a bilinear operation.*

There is a strong similarity between the convolution and the Minkowski sum. Consider the function Γ defined on the class of sets defined by $\Gamma(\emptyset) := 0$ and $\Gamma(S) := 1$ for $S \neq \emptyset$. The characteristic function at a point x of the Minkowski sum of two sets of characteristic functions F, G can be expressed as:

$$(F \oplus G)(x) = \Gamma\left(\lambda y \cdot F(y)G(x-y)\right).$$

In two dimensions, the convolution of the characteristic function of two closed convex sets is precisely their Minkowski sum, as the intersection of two closed convex sets is a closed convex set, of integral 1. Moreover, as the intersection of two simple polygons is made of simply connected components, it follows that:

Lemma 1.16. *The Minkowski sum of two simple polygons P, Q is the set of points where the convolution $1_P * 1_Q$ is non-zero.*

Example 1.17. *Consider a closed vertical segment V and an open horizontal segment H. If H_c denotes the closed horizontal segment and A, B its endpoints, then we have:*

$$1_V * 1_H = 1_V * (1_{H_c} - 1_A - 1_B).$$

Using bilinearity of $*$ and the above remark on the convolution of closed convex sets, we see that the resulting convolution is the characteristic function of a square which is closed on its bottom on top sides, and open on its left and right sides (the corners are not included in the resulting set).

The convolution has a number of interesting properties. In particular, every painting admits an inverse for the convolution. However, the above presentation doesn't give a way to compute the convolution of two polyhedra represented by their boundaries. Let us first get back to two dimensions where this question has been solved.

1.1.4 Boundary approach to the convolution

In the rest of this section, we restrict our discussion to the plane.

The boundary of a single polygon is a set of points, and this is a well defined topological concept: a point is in the boundary if it is in both the closure of the polygon and the closure of its complement. Is there a reverse operation, that allows one to obtain the polygon from its boundary? The Jordan closed curve theorem shows that a closed curve in the plane separates it in two pieces; the bounded piece is the original polygon. A closed curve, however, is not a set of points as was defined by the boundary operation. It is a continuous map $\phi : [0,1] \rightarrow \mathcal{E}^2$, so that at any point, there is a direction of traversal that is given. This direction of traversal is essential information in the proof of the theorem.

In fact, it is possible to obtain a consistent direction of traversal from the unstructured set of points: its topology, as induced from the topology of the plane, gives us enough information. So, is there any information lost in the process of forming the boundary? At least one: the original polygon, seen as a set of points, can be either topologically open or closed, or partly open and partly closed. The boundary is the same in all cases. In computational geometry, this distinction has never been made, although it is a natural one.

When it comes to represent paintings, the situation worsens (Figure 1.4). In 1983, Guibas, Ramshaw, and Stolfi [70] introduced an extended boundary representation for polygons that can represent any 2-dimensional painting.

Imagine that a car follows a certain trajectory in the plane, with, at every instant, a specific *position* and an *orientation* (a unit vector tangent to the trajectory). A pair (position, orientation) is called a *state*. In this setting, a closed curve is replaced by a continuous

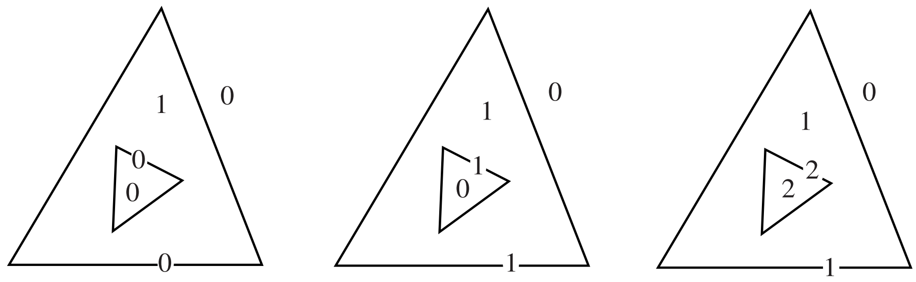

Figure 1.4: These three paintings have the same boundary regions. By orienting each triangle, it is possible to distinguish between the first and the third painting, but the values at the boundaries are still undetermined.

sequence of states, with a no-skidding condition, that the tangent to the trajectory (the sequence of positions) should be parallel to the orientation.

Tracings were introduced in [70] to augment the standard boundary representation of planar shapes via polygons or closed curves with certain additional information, to make them closed. A polygonal tracing is a set of closed trajectories (where the initial state is the same as the final state), with only two types of motion: *moves* that correspond to edges, and *turns* that correspond to vertices of a polygon.

The car moves in two ways: first, the car can go backward or forward. Second, if it stops, it can turn, either left or right, and therefore a vertex is more than the junction of two edges (the robot R2D2 would be a better analogy than a car). These two motions are necessary and sufficient to represent any painting.

An essential notion is that of the *winding number*, which is usually defined for a closed curve but can be defined for any closed trip. Consider a point and a ray out of this point. For now, assume that the ray crosses only edges of the polygonal trip, and that the starting point is not on the trip. Then the winding number is defined as the signed sum of all crossings of the ray by the trip, counting $+1$ when the car crosses from right to left, and -1 otherwise.

Theorem 1.18. *The winding number w.r.t. a closed trip depends only on the position and* *not on the direction of the ray.*

Hence, the winding number function defines a painting at all points not on the trajectory. It remains to see what happens for points that lie on the car trajectory. This is precisely the information that is given by the orientation (forward or backward) of the car and the direction (left or right) of a turn. If the orientation of the car is forward, then the painting is locally closed, that is the winding number on the boundary is 1. In Chapter 3, we give a precise way to obtain a tracing from a painting in three dimensions.

Definition 1.19. *Given two polygonal tracings P and Q, their convolution is the polygonal tracing obtained as follows: construct a) a turn for any two turns of P and Q with overlapping orientations equal to their overlap, equal to their overlap and placed at the sum of their positions and b) a move for every move of P and turn of Q (and vice versa) such that the orientation of the move falls into the set of orientations of the turn, equal to the move of P placed at the sum of their positions.*

This construction is a generalization of the construction of Figure 1.2 for the Minkowski sum of convex polygons. Note that if the tracings do not represent convex polygons, any pair of move and turn can generate a feature in the convolution, whose complexity is therefore the product of the complexities of the input polygons in the worst case. This object is called the convolution for the following reason:

Lemma 1.20. *The painting corresponding to the convolution of two tracings is equal to the convolution of their two corresponding paintings.*

Moreover, we have an algorithmic result for non-convex polygons:

Theorem 1.21. *[71] The convolution of two tracings of total size n can be computed in $O(n+k)$, where k is the size of the convolution.*

The intersection of two simple polygons can be made of any number of simply connected components, but its Euler characteristic is non-zero if and only if the two polygons intersect.

Hence, the Minkowski sum of two simple polygons is precisely the set of points at which the convolution is non-zero. The convolution is an implicit representation of the Minkowski sum. For two polygons with m and n features respectively, the worst case complexity of the convolution is only mn, which is much better than the m^2n^2 worst case of the Minkowski sum. Yet, it is not the case that the convolution is always smaller than the Minkowski sum.

Nevertheless, the convolution provides a good way to compute the Minkowski sum. Compared to the initial method of computing the Minkowski sum, which consists in matching all pairs of features together, the convolution can provide a significant saving, as it selects those pairs of features whose interaction may contribute to the boundary of the Minkowski sum.

1.1.5 Historical note

The definition of paintings and tracings have been presented in the above order for presentational reasons. Historically, the ideas were discovered in the following sequence.

Guibas, Ramshaw, and Stolfi [70] first formulated the convolution as an operation directly defined on boundaries. So as to make the definition independent of the representation, they defined a tracing as a multi-set of states. They showed a number of properties of these tracings, and how they could be used to solve some algorithmic problems. For instance, they showed how it is possible to compute the common tangents of two polygons by taking the convolution of their tracings in opposite orientations.

Then, in 1991, Schapira showed that the operation of convolution was naturally related to *constructible functions*, i.e. integer-valued functions on a smooth manifold with certain regularity properties. We presented here this theory in a more restricted framework.

1.2 Related work

In a series of three papers known as the Piano Movers series [122, 123, 124], Schwartz and Sharir introduced the idea of configuration spaces in motion planning. In [123], they adapted a general method of Collins [45] to obtain a cell decomposition (called a *cylindrical decomposition*) of the free space of a robot with arbitrary degrees of freedom, based on the decomposition of arrangements of algebraic surfaces in d dimensions. The scheme is doubly exponential in d. In the general setting, Canny [29] has since devised a more efficient method (called the *road map method*) that is only singly exponential in the dimension of the space. In the case where the number of degrees of freedom is not fixed, the motion planning problem has been shown to be PSPACE-complete [81, 82], so that the exponential dependence in d is likely to be unavoidable.

Given the high complexity of the previous general methods, a number of specialized algorithms were developed for special types of robots: a rod translating and rotating in the plane [122, 97, 130], an L-shaped polygon [75] and a general polygon in the plane [122]. The restriction of the problem to one of these robots limits the dimension of the configuration space, and the structure of the free space is easier to understand.

The Minkowski sum is a natural operation that appears in many different contexts. In Graphics, the *weighted Minkowski sum* provides a way to morph two shapes into an intermediate 'interpolating' shape [120].

Minkowski sums have been studied extensively in mathematical literature. Risler [117] studied the problem of placing curved polygons using Minkowski sums. Minkowski addition of polytopes was studied by Gritzmann and Sturmfels [67]. Kaul, et al. [85, 84] studied the computation of Minkowski sums in two and three dimensions using line and plane sweep methods.

As we have seen above, the computation of the Minkowski sum of convex polyhedra reduces to the problem of computing the overlay of two convex subdivisions of the plane. Since the result of Guibas and Seidel [71], a few interesting methods were found with practical improvements [30, 61].

In Computer Aided Design, a popular representation of solids is via their Constructive Solid Geometry (or CSG) representation. To avoid problems related to open/closed nature of sets, only *regular sets* are considered in this setting (i.e. those that are equal to the closure of their interior), and operations like union, intersection, set difference are defined on these sets. In general, the basic objects are convex polyhedra, and more complex shapes are obtained using these operations. These operations can be expressed as algebraic sums of characteristic functions. To compute the convolution of two solids represented in CSG, we can use the bilinearity of the convolution, and obtain readily a CSG representation of the convolution. Moreover, Edelsbrunner [58] shows how to convert the boundary representation of a polytope into its CSG representation, or, more precisely, how to represent the characteristic function of a polytope given by its boundary as an algebraic sum of characteristic functions of a linear number of convex polytopes.

The notion of convolution is closely related to that of geometric duality [131]. Indeed, the extended boundary representation of polygons by Guibas et al. [70] has a well-defined geometric dual in the two-sided projective plane. The notion of geometric duality was first developed in the algorithmic context by Chazelle, Guibas and Lee [38], and is an essential tool for algorithmic design in computational geometry [96, 15, 16, 50]

Most of the results in this thesis have appeared as publications. Part of Chapter 1 appeared in the 12th Symposium on Computational Geometry [115]. Part of Chapter 3 appeared in the 2nd Workshop on Algorithmic Foundations of Robotics [23]. Part of Chapter 4 appeared in the 4th European Symposium on Algorithms [21]. An application for assembly planning is described in [118].

1.3 Thesis overview

In the second chapter of this thesis, we focus on the convolution of simple polygons in the plane. We show how the structure of the convolution can be represented using a two-dimensional diagram called the *fiber product*, and propose a new algorithm to compute the outer face of the Minkowski sum from the convolution.

In the third chapter, we extend the work of Guibas et al [70] to three-dimensions: we introduce a boundary representation of three-dimensional paintings, called *polyhedral tracings* that is well suited for computational purposes, and define the convolution operation based on this representation.

In the fourth chapter, we present a nearly optimal algorithm to report all bichromatic intersections between two families of connected line segments. We show how this algorithm can be used to compute the convolution of two polyhedral tracings in nearly optimal output sensitive time.

The fifth chapter concludes this thesis with a summary and possible applications and extensions of this work.

Chapter 2

Planar Minkowski sum outer face computation

Minkowski sums arise commonly in robotics under the name of configuration space obstacles [98], when the robot motion is restricted to translation. If an obstacle and a robot are modeled by two polygons P and Q, the Minkowski difference $P \ominus Q$ is the set of placements of Q with respect to P that cause the two polygons to intersect (Figure 1.1). The *outer face* of this set is the infinite component of its complement; it represents the set of *free* placements of Q that can be reached by translation from infinity. In most robotics applications, it is assumed that the robot is initially far away from the obstacle and consequently has no way at all of reaching the cavities in the complement of the Minkowski difference, even though these cavities correspond to free placements of the robot.

As we have seen in the introduction, the Minkowski difference $P \ominus Q$ is equal to the Minkowski sum $P \oplus -Q$, which is itself closely related to the convolution (Lemma 1.16). In this chapter, we present an algorithm to compute the outer face of the Minkowski sum of two simple polygons P and Q by directly using the convolution of their associated tracings.

Planar tracings were introduced as generalizations of ordinary polygons by Guibas, Ramshaw, and Stolfi [71] as part of the *kinetic framework* for computational geometry. Along with the notion of tracings they introduced the convolution operation that takes two tracings as inputs and produces a new tracing. Ordinary simple polygons can be viewed as tracings by giving them an orientation and direction of traversal, say counterclockwise.

In this chapter, we show how to use the convolution to find the outer face of the Minkowski sum efficiently, in time $O((k + (m + n)\sqrt{\ell}) \log^2(m + n))$, where m and n denote the sizes of P and Q respectively, k is the size of the convolution $P * Q$, and ℓ is the number of cycles in the convolution tracing. Note that the convolution is always of size $O(mn)$, while the Minkowski sum can be of size $\Theta(m^2n^2)$ (Figure 1.3). We use an algorithm presented in [71] to compute the convolution of planar polygonal tracings in optimal time $O(m + n + k)$. The outer face of the Minkowski sum can be computed by applying the general algorithm of [37] to compute a single face in an arrangement of k line segments in time $O(k\alpha(k) \log k)$. It is shown in [78] that the complexity of a single face in the Minkowski sum can in fact be $\Theta(mn\alpha(mn))$ in the worst case.

Unlike that approach, which only needs the edges of the convolution as a set of line segments, our algorithm exploits the way in which the segments are linked together into cycles in the convolution. The key idea is to detect self-intersections and to excise the loops thus formed in the convolution. To detect self-intersections, we provide a novel scheme for performing ray-shooting against a dynamic polygonal line, using the geodesic triangulation structure of [35]. Our scheme relies on dividing the polygonal line into blocks of contiguous segments and storing a geodesic triangulation for each block. Ray-shooting queries are answered in amortized time $O(\log^2 k)$, where k is the size of the polygonal line. The problem of dynamic ray-shooting has been studied in a more general setting in [64], where the authors obtain a worst-case time bound of $O(\log^2 n)$ for ray-shooting against a dynamic planar subdivision of size n. Our technique works only for polygonal lines, but it is simpler and sufficient for our needs. Our outer-face algorithm, though again slightly inferior in asymptotic complexity to the single-face method, is easy to implement and helps to illuminate some new aspects of the connection between convolution and the Minkowski sum.

This chapter is organized as follows. Section 2.1 introduces basic terms and outlines the relationship between convolution, fiber product, and Minkowski sum.

In Section 2.2 we describe how to traverse a cycle of the convolution, detecting self-intersections, excising loops, and computing its outer boundary. In Section 2.3, we discuss how to connect multiple cycles of the convolution together and traverse them in sequence using the algorithm of Section 2.2 to find the outer face of the Minkowski sum. Section 2.4 concludes the chapter.

2.1 Convolution and Minkowski sum

We recall briefly some terms introduced in the kinetic framework [70]. A *state* is a pair consisting of a position $\dot{s}$ in the plane and a direction $\vec{s}$. A *move* is a set of states with constant direction and position varying along a line segment parallel to the direction. A *turn* is a set of states with constant position and direction varying along an arc of the circle of directions. A *polygonal trip* consists of a continuous sequence of moves and turns; the trip is closed if it starts and ends at the same state. A *polygonal tracing* is a collection of closed polygonal trips. We think of each loop of a tracing as being traversed by a car which always faces in the direction of the current state. The winding number of a point with respect to a tracing has the traditional meaning. A detailed exposition of the meanings of these terms was presented in Chapter 1 (also [70]).

Given a simple polygon P we denote as $\hat{P}$ the tracing corresponding to a counterclockwise traversal of its boundary with the car always moving forward. Let P, Q be two simple polygons with corresponding tracings $\hat{P}$ and $\hat{Q}$. The convolution of $\hat{P}$ and $\hat{Q}$ is defined as

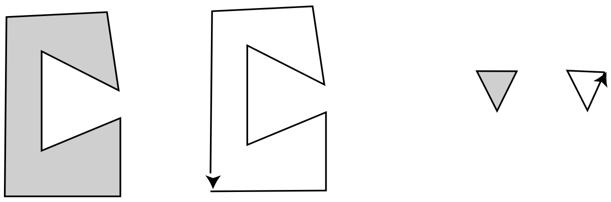

Figure 2.1: Two simple polygons P and Q with their associated counterclockwise tracings $\hat{P}$ and $\hat{Q}$ respectively.

follows. If p and q are states in $\hat{P}$ and $\hat{Q}$ such that $\vec{p} = \vec{q}$, then the state $c = (\dot{p} + \dot{q}, \vec{p})$ is a state in the convolution. In other words, the convolution is a set of vector sums of states in $\hat{P}$ and $\hat{Q}$ where the two cars face in the same direction. An example of two simple polygons, their tracings and their convolution are shown in Figures 2.1 and Figure 2.2.

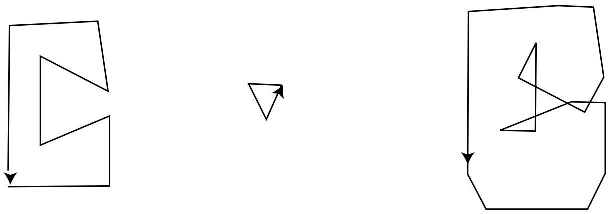

Figure 2.2: The convolution $\hat{P} * \hat{Q}$ of two simple polygonal tracings $\hat{P}$ and $\hat{Q}$.

2.1.1 Fiber product representation of the convolution

A convenient way to visualize the convolution is to depict it as the *fiber product* $\mathrm{FP}(P, Q)$ defined as follows (we use the term fiber product informally here; for a precise and more detailed exposition, see [24]). Lay out the perimeter of P on the horizontal axis and the perimeter of Q on the vertical axis, and plot all pairs of boundary points that produce a state in the convolution (note that this representation collapses information about matching states in turns). A vertical edge of the fiber product corresponds to a convolution move derived from $\hat{Q}$ while a horizontal edge is derived from an edge of $\hat{P}$ (Figure 2.3). An advantage of this representation is that the cycles of the convolution are well separated. That makes it easy to visualize the convolution. Let us now consider this in more detail.

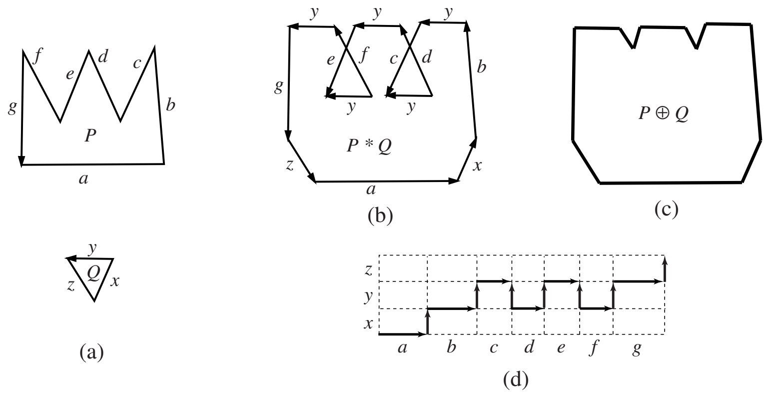

Figure 2.3: (a) Two tracings; (b) Their convolution; (c) Their Minkowski sum; (d) Their fiber product (opposite sides of the rectangle are identified).

In the subsequent discussion the two input polygons are referred to as "red" and "blue" polygons respectively. Laying out the boundary of the red polygon results in a sequence of edges and moves mapped onto horizontal rectangle edge. Similarly the other polygon boundary maps onto the vertical edge. A pair of points on the boundaries can correspond to one of three cases: (a) interior of a red edge and a blue vertex, or (b) interior of a blue edge and a red vertex, or (c) interiors of a red edge and a blue edge. In case (a), if the red edge and the blue corner combine to produce an edge in the convolution then the corresponding point is marked on the fiber product. Case (b) is similar. If no two edges are parallel, case (c) doesn't arise. The collection of marked points form edges in the rectangle comprising the fiber product. The edges form cycles and in fact the cycles are well separated. The separation is because a given pair of boundary points can only produce one feature on the fiber product. Hence no two edges of the fiber product diagram can intersect. In the case when the convolution has multiple cycles of edges, this makes it easy to visualize since the cycles don't interfere with each other on the fiber product diagram.

The Minkowski sum $P \oplus Q$ of polygons P and Q consists of pair-wise vector sums of all interior or boundary points of P and Q. Note that this includes all pairs used in the definition of the convolution. The Minkowski sum $P \oplus Q$ is thus the non-zero winding number region of $\hat{P} * \hat{Q}$ (Lemma 1.16). From this it follows that the convolution pairs cover the boundary of the Minkowski sum; other pairs must be inside or can be on the boundary, but they cannot be outside. On the other hand, edges of the convolution can be interior, wholly or in part, to the Minkowski sum, e.g., when they separate two regions both of which have non-zero winding number.

A key advantage of the convolution over the Minkowski sum is that it is stored as a collection of edge cycles without regard to how these cycles might intersect themselves or each other. This is the reason why the convolution is $O(mn)$ in size regardless of the geometry of the input polygons. The Minkowski sum, on the other hand, can be combinatorially much larger, since intersections of different convolution edges that are on the Minkowski sum boundary must be explicitly stored.

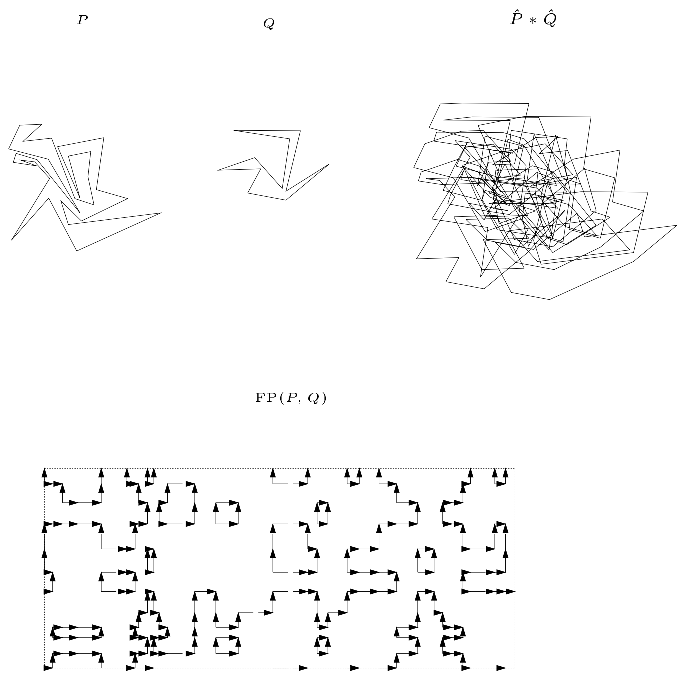

Figure 2.4: A larger example of the convolution and the fiber product (the fiber product is scaled to fit within the page).

A worst-case example of a Minkowski sum with complexity $\Theta(m^2n^2)$ was exhibited in the Introduction (Figure 1.3). Notice that in this case most of the vertices of the Minkowski sum are due to holes. In fact the outer boundary is $O(m+n)$ in size. It was shown in [78] that the outer face of the Minkowski sum is $O(mn\alpha(mn))$ in the worst case; they also show an example that achieves this bound tightly.

The above suggests that it makes sense to compute only the outer face of the Minkowski sum instead of the entire structure. In the context of Robotics, the outer face is the complement of the region of freedom of the robot that is reachable from infinity. For the purpose of most applications this is the region that is of interest. One usually does not care about the robot's local free space in positions that are not reachable from outside, like the hole inside the Minkowski sum of Figure 2.5. This space may only be reached from outside if the robot were allowed to penetrate the obstacle; this is not the case in most applications.

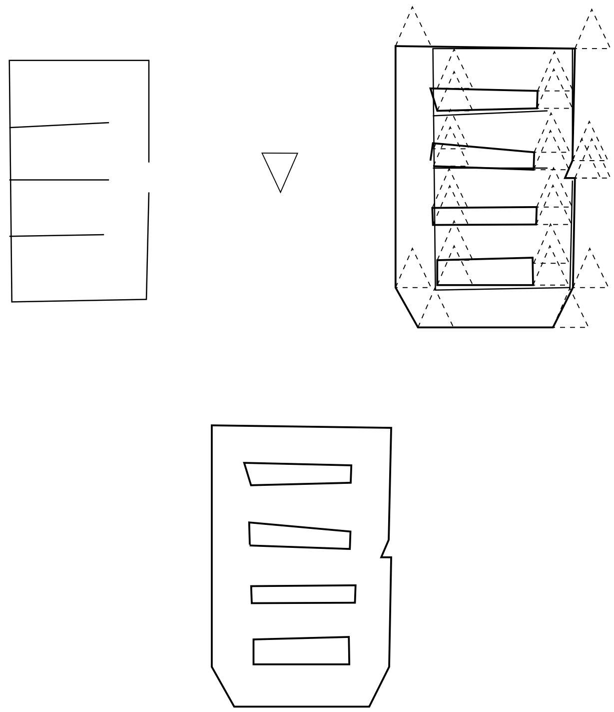

Figure 2.5: An example in which the Minkowski sum contains a small outer face but many interior faces. In this type of a case, it is advantageous to compute only the outer face instead of the entire Minkowski sum.

In this chapter we explore the algorithmic connection between the convolution and the Minkowski sum. We show how to use edge cycles of the convolution to derive the outer face of the Minkowski sum using a novel loop-cutting technique.

2.1.2 Cycles of the convolution

The convolution consists of a number of closed polygonal trips that we call *cycles*. They may overlap in the plane, producing intersections. In the fiber product, however, the cycles are spatially separated out. This is because a pair of boundary features of P and Q can only produce a single feature in the convolution. This property makes the fiber product a convenient tool for indexing the convolution, without the clutter introduced by its actual embedding in the plane. Figure 2.6 shows two simple polygons which produce a convolution containing two cycles. Each cycle contributes to the outer face of the Minkowski sum, requiring our algorithm to consider both in order to find the outer face. Figure 2.4 shows a large example of a convolution with cycles separated out in the corresponding fiber product. This example clearly shows the advantage of the fiber product visualization — it would be hard to distinguish the cycles in the convolution whereas it is easy to distinguish them in the fiber product.

Lemma 2.1. *If $\hat{Q}$ is a convex polygonal tracing and $\hat{P}$ is a single cycle polygonal tracing, the convolution $\hat{P} * \hat{Q}$ consists of a single cycle.*

Proof. Consider any two pairs of matching states in the convolution, say (p_1, q_1) and (p_2, q_2). Move the P car from p_1 continuously along P (in either direction) till p_2 is reached. Since Q is convex, the Q car always matches any arbitrary (left or right) turn the P car makes, simply by moving (forwards or backwards) along Q from its current state. Thus any two states on the convolution can be connected and therefore there can be only one cycle. □

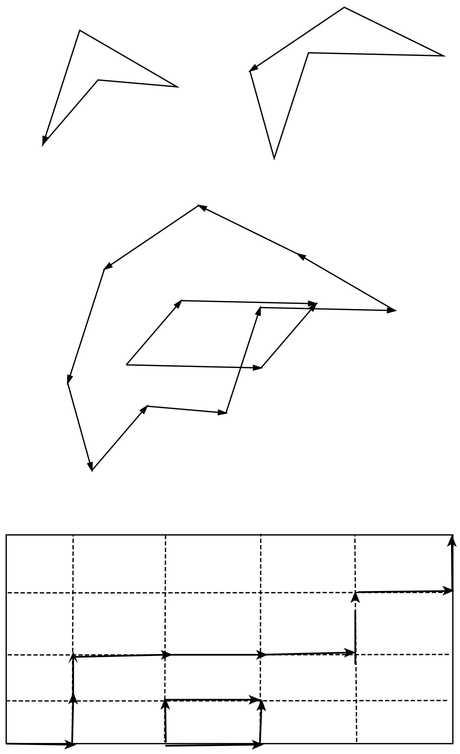

Figure 2.6: Shows two polygons, their convolution, and the fiber product; the convolution contains two cycles and each cycle contributes to the outer face of the Minkowski sum.

2.1.3 Computing the convolution

Guibas and Seidel [71] show how to compute the convolution of two simple polygonal tracings $\hat{P}$ and $\hat{Q}$ in optimal time $O(m+n+k)$, where m, n, and k are the sizes of $\hat{P}, \hat{Q}$, and $\hat{P}*\hat{Q}$ respectively. Their method consisted of storing the set of turn intervals of each polygon in a "hive-graph" data structure. A hive-graph uses the fractional cascading paradigm to store intervals in multiple levels of decreasing detail. They store the turns of $\hat{P}$ as intervals in a hive graph h_P and the turns of $\hat{Q}$ as intervals in another hive graph h_Q. Hive graphs allow efficient queries of the form — a) in which interval, if any, is angle θ in a hive graph h? and b) which intervals does a given interval $[\theta_1, \theta_2]$ intersect with in a hive graph h. For each move of $\hat{P}$, with say slope θ, the algorithm makes queries of the form — which intervals is angle θ in the hive graph h_Q. The responses to this query combined with the query move comprise pairs that generate convolution edges of the form "move of $\hat{P}$+turn of $\hat{Q}$". Similarly moves of $\hat{Q}$ are queried in the hive graph h_P. The responses constitute pairs (along with the move of $\hat{Q}$ that generated the query) corresponding to convolution edges of the form "turn of $\hat{P}$ + move of $\hat{Q}$".

In order to find turns of the convolution we make queries of type b) as follows. For each turn of $\hat{P}$ with slope set $[\theta_1, \theta_2]$ query the interval $[\theta_1, \theta_2]$ versus the hive graph h_Q. The responses constitute convolution turns of the form "turn of $\hat{P}$ + turn of $\hat{Q}$". This query could have been performed vice-versa (turn of $\hat{P}$ versus hive graph of Q) with the same result.

In the paper cited above, Guibas and Seidel compute the set of moves and turns comprising the convolution and sort them using radix sort to find the order in which they are traversed in the convolution. We show below an alternative way to find the traversal order using signs assigned to turns — this method follows as a consequence of results in the paper that introduced the kinetic framework. This alternative method is convenient and more efficient: it avoids the need to radix sort the moves and turns after finding them.

In the kinetic framework paper of [70] the authors show how to consistently assign positive and negative signs to the moves and turns of a tracing. We recall below the rules used to assign these signs. A left turn is assigned a positive sign (+1 multiplicity if the turn is smaller than 2π) and a right turn is assigned a negative sign (-1 multiplicity if the turn is smaller than 2π). The multiplicity of the sign of the turn increases in proportion to the magnitude of the turn in proportion to 2π. A move of a tracing is assigned positive (+1) sign if it is forward and negative (-1) sign if it is negative. The above rules are designed such that the convolution of two tracings derives consistent signs for its moves and turns. The derived sign of a move of the convolution is the product of the signs of the pair of move and turn that generated it. The derived convolution signs are consistent in the sense that the signed multiplicities of the convolution turns and the moves generate a properly ordered collection of cycles of a balanced tracing. For the purpose of illustration let us develop below the specific case of convolving two counterclockwise simple polygonal tracings.

For a counterclockwise simple polygonal tracing a) all moves are forward and hence positive, b) all left turns are positive, and c) right turns are negative. Hence the convolution of a move with a left turn is a forward (i.e. positive) move and the convolution of a move with a right turn is a backward (i.e. negative) move. It is then possible to traverse the convolution starting from any of its moves: if the sign of the move is positive traverse the cycle in the forward direction and if it is negative traverse it in the backward direction.

2.2 Outer boundary of a convolution cycle

In this section, we focus on the computation of the outer boundary of one cycle of the convolution, and show that the outer boundary of a cycle composed of k moves can be computed in time $O(k \log^2 k)$. The algorithm works as follows: starting from an outer-face vertex, we grow a simple polygonal chain, one move at a time, keeping a ray-shooting data structure that allows us to detect self-intersections. When a self-intersection occurs, we add the intersection point as a new vertex, and then "pinch off" and excise the edge cycle thus formed. We describe the algorithm in more detail below and show that it correctly computes the outer boundary of a cycle. We then show how the geodesic triangulation data structure of [35] can be adapted for efficient dynamic ray-shooting towards a growing simple polygonal chain.

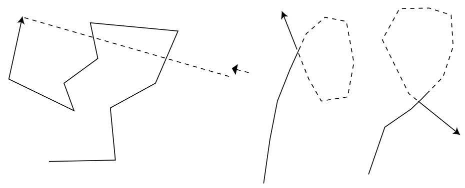

Figure 2.7: Snap shots of the running of the algorithm: left picture depicts the ray-shooting, and the middle and right pictures show excised clockwise and counterclockwise loops in dotted lines.

2.2.1 Description of the algorithm

The algorithm works as follows: start from an outer vertex and grow the chain one edge at a time. The new endpoint of the next edge added becomes the current active end of the chain where the next edge will go. For each edge being added, we shoot *backwards* from the free endpoint of the new edge towards the active vertex of the polygonal chain, to see if the edge intersects the polygonal chain grown so far – see Figure 2.7. If it does not, the edge is accepted and added to the chain. If it does, we break both the new edge and the polygonal chain at this first intersection found. Because we shot backwards, we are guaranteed to find the last intersection between the chain and the edge (if we were to imagine the new edge growing out of then active vertex). The part of the original chain between the intersection and the active vertex is removed (this is the cycle being pinched off), and then the remaining fragment of the new edge is added to give us a new active vertex.

Snapshots from a sample execution of the algorithm are shown in Figure 2.8. In this example, the convolution consists of one cycle and in a sequence of five steps of loop-cutting, the outer face of the Minkowski sum is found. The first five steps consist of excising counterclockwise loops of the convolution. The following step excises a clockwise loop which corresponds to a hole in the Minkowski sum. The next step excises another counterclockwise loop resulting in the outer face. The loops are shown in lighter print just prior to their excision.

It remains to show one can find an outer-face edge of the convolution to start the loop excision process. In linear time we find the left-most edge of P and the left-most turn of Q. The convolution of these is necessarily on the outer face of the convolution; in fact, it is the left-most edge of the convolution. Compute this edge and treat it as the first edge in the loop traversal.

2.2.2 Convolution loops

The proposed algorithm relies on an important observation, which justifies the pinching off: if the polygonal line starts from a point that lies on the outer face of the Minkowski sum, then no edge (or part of an edge) in a loop can lie in the outer face. For the argument below, recall that we have oriented all our polygons in a counterclockwise manner.

Lemma 2.2. *Let π be a path following a cycle of the convolution and starting from a point O known to lie on the outer face. If π self-intersects, then no point on the loop thus created can be on the outer face of that cycle.*

Proof. Clockwise loops are completely enclosed within the convolution because the interior of the convolution is always to the left of a convolution edge according to the way the edge is traversed in going around the cycle. This means that the region enclosed by the clockwise loop is completely within the outer face of the Minkowski sum.

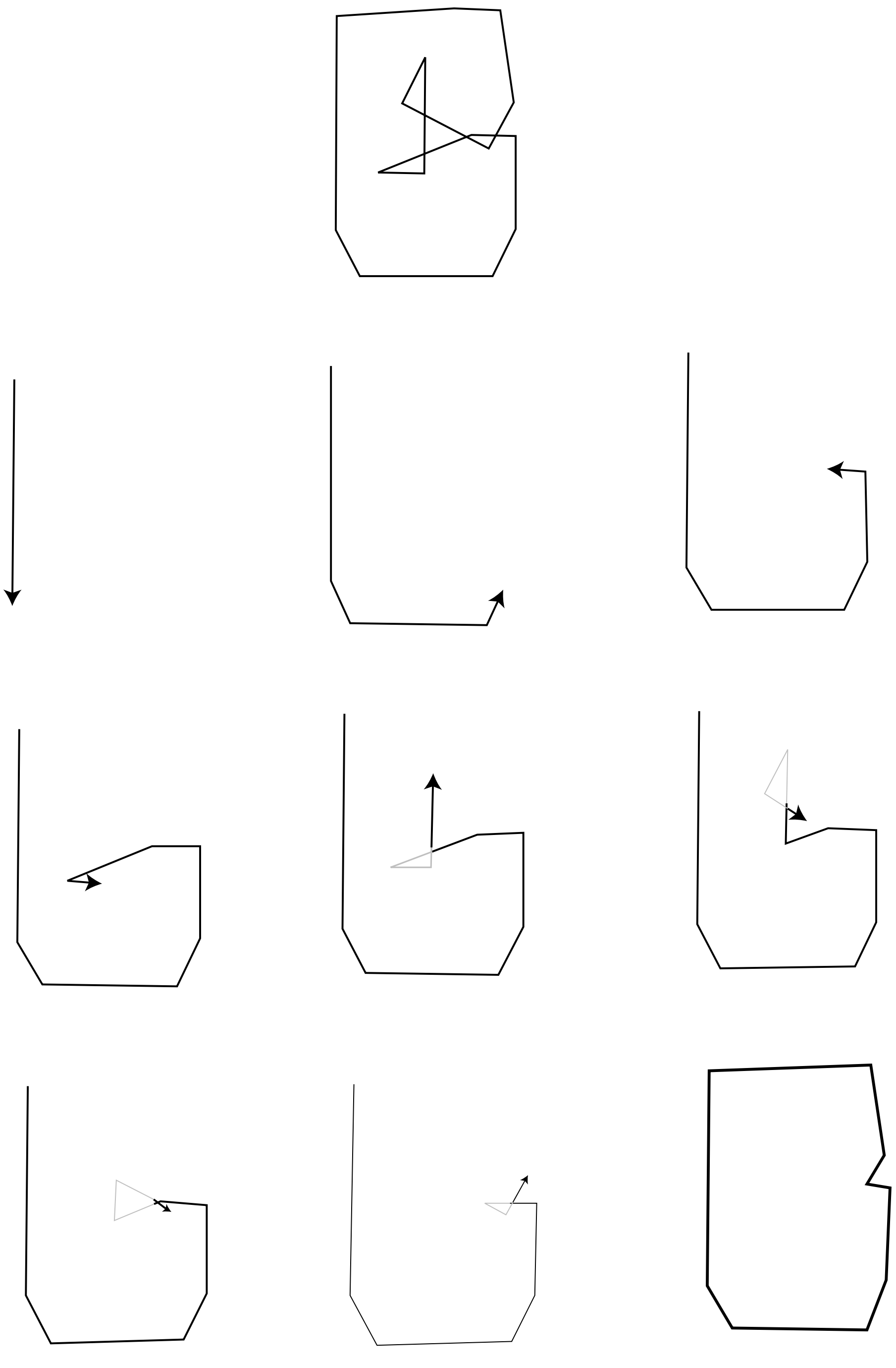

Figure 2.8: Snapshots of the execution of the loop excision algorithm: the entire convolution (top), and a sequence of states of the algorithm during the computation of the outer boundary of the convolution cycle.

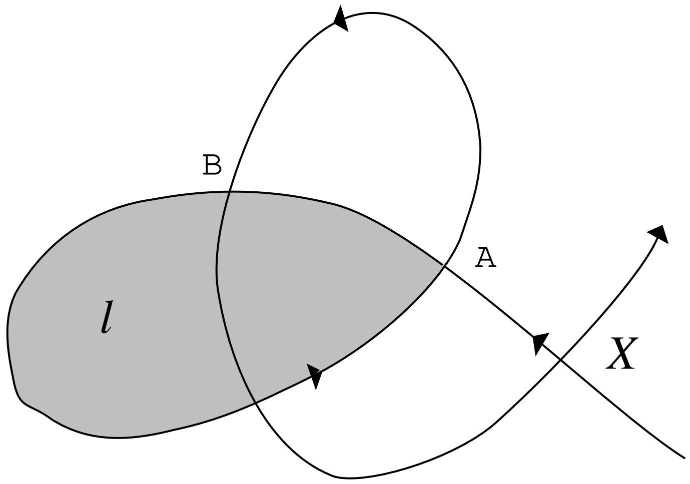

Figure 2.9: A counterclockwise loop of the convolution with the excised loop shown in grey.

Let us now consider a counterclockwise loop ℓ created by the first self-intersection of π at the point A. We can assume that A itself is on the outer face, for if π had already entered inside the convolution by crossing in at an earlier point X, then X itself defines another loop whose set of edges includes those of ℓ and to which our argument below can be applied: if no portion of that cycle can be on the outer face, *a fortiori* the same applies to ℓ.

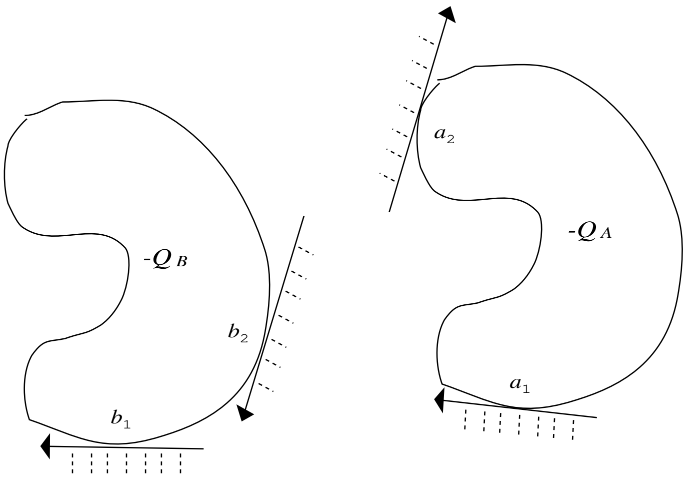

Figure 2.10: Placements $-Q_A$ and $-Q_B$ of $-Q$ corresponding to intersections A and B of Figure 2.9.

Now around A the loop cannot contribute to the outer face, for it separates regions of winding number 1 and 2 respectively – see Figure 2.9. So the only way for ℓ to appear on the outer face is for π to cross again into ℓ so as to bring the lowest of these winding numbers down to 0. Let the position of such a crossing be B. By reasoning as above, we can assume that B is on the outer face as well (if not, keep going till a B is found that is on the outer face). Points A and B are *double points* in the convolution: each of them denotes a placement of $-Q$ that causes it to touch the boundary of P in two places. Let us denote these placements as $-Q_A$ and $-Q_B$ respectively, and the tangency points as a_1, a_2 and b_1, b_2 respectively. The portion σ of the boundary of P of that generates the portion of π seen so far visits these points in the order a_1, b_1, a_2, b_2, as depicted in Figure 2.10. Since the cycle is counterclockwise, the interior of P must locally always be as depicted in Figure 2.10. Note also that, since A and B are on the outer face by assumption, the boundary of P cannot enter the interior of $-Q_A$ or $-Q_B$.

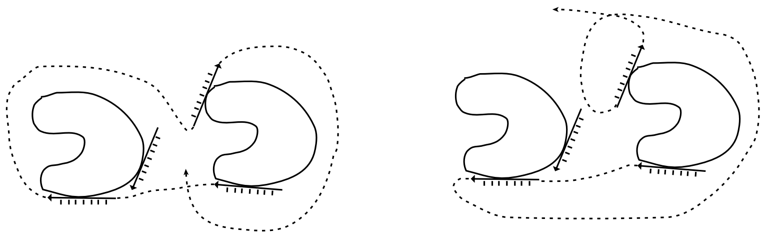

Figure 2.11: A counterclockwise loop cannot be on the outer face because connecting the end points results in a self intersection.

But now we have a topological impossibility: there is no way to connect the points a_1, b_1, a_2, b_2 into a cycle in the order given, without either causing self intersections or going through the regions $-Q_A$ or $-Q_B$. Some of the disallowed possibilities are shown in Figure 2.11. But this contradicts the assumption that there is a simple counterclockwise oriented polygon P passing through these points and avoiding regions $-Q_A$ and $-Q_B$. $\square$

It is worth noting that the proof relies on the input tracings representing simple polygons with only forward moves. This is the only case which is of any relevance to robotics: tracings with backward moves model locally open sets that do not normally arise when modeling obstacles and robots.

2.2.3 Dynamic ray-shooting structure

An efficient mechanism is needed to detect self-intersections of the dynamically updated polygonal line. The polygonal line is updated at one end and we need to efficiently check whether a segment added to the end of the polygonal line causes a self-intersection. The check is performed using a ray-shooting operation of the newly added segment versus the existing polygonal line. In this section we present a method to do this that uses multiple copies of a static ray-shooting data structure.

A geodesic triangulation based ray-shooting data structure for a simple polygon in the plane was presented in [35]. The data structure is computed in linear time and supports ray-shooting queries in $O(\log k)$ time where k is the size of the polygon. This technique can be extended for polygonal lines by computing geodesic triangulations that stay to the left and to the right of the lines respectively. Ray-shooting queries can be made to the left as well as the right of the polygonal line.

The ray-shooting data structure must handle updates of the polygonal line at one end. In order to efficiently process these updates we divide the polygonal line into contiguous blocks of segments and store a static ray-shooting structure for each block. We choose the number of blocks to be $O(\log k)$, where k is the size of the polygonal line. The ray-shooting cost summed over all the blocks is therefore $O(\log^2 k)$. We now show how to choose the lengths of the blocks so that the amortized cost of updates of the polygonal line is minimized.

Suppose the current length of the polygonal line is j. Let $\tilde{\jmath} = (\tilde{b}_\lambda, \ldots, \tilde{b}_0)$ be a *redundant* binary representation of j: where each $\tilde{b}_i$ is $0, 1$, or 2, and $j = \sum_i \tilde{b}_i 2^i$ (we denote by $\mathrm{Val}(\tilde{\jmath})$

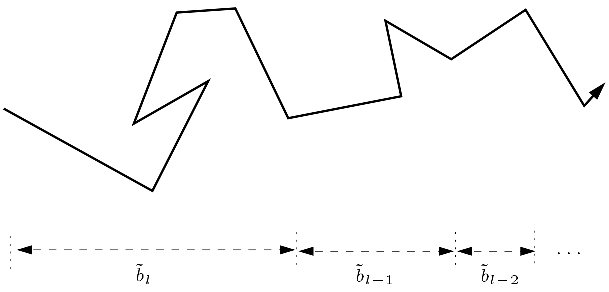

Figure 2.12: Dividing the polygonal line into blocks of segments.

the associated value j of $\tilde{\jmath}$). The polygonal line is divided into blocks of length $\tilde{b}_i 2^i$ with the active end stored in the smallest block. Since a bit is allowed to be 2, a number can have redundant representations: for instance 5 can be either 101 or 21. Addition involves carry propagation and subtraction involves borrow propagation using the following table.

Operation	0	1	2	10	20
INC	1	2	10	11	21
DEC	-	0	1	01	11

We provide some examples below of the increment and decrement operator to help the reader gain some intuition for this operation.

Operation	Operand	Result
INC	1222	2111
INC	2112	2121
INC	201	202
DEC	2000	1111
DEC	1210	1201
DEC	1202	1201

A given integer may have multiple redundant binary representations as shown in the following table. The maximum number of such multiple representations depends logarithmically on the size of the integer. Redundant representations of the first seventeen integers are presented in the following table. Notice that numbers of the form 2^i have exactly $i+1$ redundant representations — the dependency is logarithmic as claimed above.

Integer	Representations...				
0	0				
1	1				
2	10	2			
3	11				
4	100	20	12		
5	101	21			
6	110	102	22		
7	111				
8	1000	200	120	112	
9	1001	201	121		
10	1010	1002	210	202	122
11	1011	211			
12	1100	1020	1012	220	212
13	1101	1021	221		
14	1110	1102	222		
15	1111				
16	10000	2000	1200	1120	1112

2.2.4 Insert and delete operations

Let us now look at what happens when a segment insertion or deletion is performed at the active end of the polygonal line of length j. The redundant representation $\tilde{j}$ of j

is incremented or decremented and for each bit position $\tilde{b}_i$ that changes as a result, the corresponding block(s) is (are) reconstructed or destroyed according to the following rules: a) if the bit value changes from 0 to 1 (or 1 to 2) then a block (or two) are created, b) if the bit value changes from 1 to 0 (or 2 to 1) then a block (or two) are destroyed, c) if the bit changes from 1 to 2 then a block is created, and lastly d) if the bit changes from 2 to 1 then a block is destroyed. Since the length of the block (or blocks) is $\tilde{b}_i 2^i$, the cost of effecting such a change is $O(2^i)$. These rules are designed to preserve the invariant that the polygonal line is divided according to the bit values in the redundant binary representation $\tilde{\jmath}$. We note that cycle-cutting is implemented by deleting in turn each element of the discarded portion of the cycle.

The worst case insertion (or deletion) cost of the redundant binary counter occurs when every bit position changes as a result of a single insert (or delete). The worst case cost is linear in the number of segments in the polygonal line. However, one would expect that in the average case relatively few bit positions are changed as a result of a single operation. An interesting question in the context is: what is the average cost of an insert (or delete) operation averaged over a large number of operations?

We now perform an amortized analysis of the redundant binary counter under the assumption that the cost of changing bit position $\tilde{b}_i$ is $O(2^i)$. We denote by k be the total number of increment and decrement operations performed by the algorithm. Note that k is bounded by twice the size of the convolution. The worst case cost of an insert or delete is $O(k)$. The worst case corresponds to when the highest order bit is changed by the operation. However one would expect such an expensive operation to happen infrequently. In other words, we would expect that the amortized cost of an operation (averaged over all operations) is considerably less. We now show formally that the amortized cost of updates is $O(\log k)$ (indeed much less).

2.2.5 Amortized analysis of the redundant binary counter

We perform an analysis of the redundant binary counter in an abstract setting in which, to flip $\nu + 1$ bits (from position 0 to position ν) in the redundant counter, we incur a cost of 2^ν. This accurately models the dynamic ray shooting structure up to a constant factor.

Recall that k denotes the total number of increment and decrement operations. The number of bits in the counter at any time is bounded by $\lambda = \lceil \log k \rceil$. We show using a potential function based argument that the amortized cost of each operation on the counter is $O(\log k)$.

At a certain stage t of the algorithm, we construct a potential function ϕ as follows. Suppose that the redundant counter has representation $\tilde{\jmath} = (\tilde{b}_\lambda, \ldots, \tilde{b}_0)$ with a value of j. For each bit position $\ell, \ell \leq \lambda$, we associate a potential function $\phi_\ell(t)$ as follows (we use a subscript to 0 or 1 to indicate its position):

$$\begin{aligned} \phi_\ell(t) &= \left| \mathrm{Val}(0_\lambda, \ldots, 0_\ell, 1_{\ell-1}, \ldots, 1_0) - \mathrm{Val}(0_\lambda, \ldots, 0_\ell, \tilde{b}_{\ell-1}, \ldots, \tilde{b}_0) \right| \\ &= \left| 2^\ell - 1 - \sum_{i<\ell} \tilde{b}_i 2^i \right| . \end{aligned}$$

Note that the potential $\phi_\ell(t)$ is 0 in the safe situation where all bits to the right of ℓ are 1's (it will take a long time before bit ℓ can be flipped), while it is about 2^ℓ when bit ℓ risks being flipped, either by an INC or a DEC operation,

The total potential at step t is

$$\phi(t) = \sum_{\ell \leq \lambda} \phi_\ell(t) .$$

Lemma 2.3. *If an Increment operation from stage t to stage $t+1$ flips $\nu + 1$ bits in the representation of the redundant binary counter, the potential ϕ decreases by at least $2^\nu - \lambda$.*

Proof. If $\nu+1$ bits are flipped by an increment, the change of representation is:

$$(\ldots, \tilde{b}_\nu, 2, \ldots, 2) \longrightarrow (\ldots, \tilde{b}_\nu + 1, 1, \ldots, 1).$$

Before the increment, we have, for all $\ell \leq \nu$,

$$\begin{aligned} \phi_\ell(t) &= \left| 2^\ell - 1 - \sum_{i<\ell} 2 \cdot 2^i \right| \\ &= 2^\ell - 1. \end{aligned}$$

After the increment, the potential at the same position becomes:

$$\begin{aligned} \phi_\ell(t+1) &= \left| 2^\ell - 1 - \sum_{i<\ell} 1 \cdot 2^i \right| \\ &= 0. \end{aligned}$$

For $\ell > \nu$, we have:

$$\phi(t) = \left| 2^\ell - 1 - \sum_{\nu<i<\ell} \tilde{b}_i 2^i - \mathrm{Val}(0, \ldots, 0, \tilde{b}_\nu, 2, \ldots, 2) \right|$$

And $\phi(t+1)$ is the same with $\mathrm{Val}(0, \ldots, 0, \tilde{b}_\nu, 2, \ldots, 2)$ replaced by $\mathrm{Val}(0, \ldots, 0, \tilde{b}_\nu + 1, 1, \ldots, 1)$, which is just one more by definition. Hence, for such a position ℓ, the potential increases or decreases by 1.

Thus the change in total potential is

$$\begin{aligned}
\phi(t+1)-\phi(t) &= \sum_{\nu\le\ell\le\lambda}(\phi_\ell(t+1)-\phi_\ell(t)) \\
&\quad +\sum_{\ell<\nu}(\phi_\ell(t+1)-\phi_\ell(t)) \\
&\le \lambda-\nu+\sum_{\ell<\nu}(-2^\ell+1) \\
&= \lambda-2^\nu,
\end{aligned}$$

which is the claimed bound. □

We have an exactly parallel situation for a DEC:

Lemma 2.4. *If a Decrement operation from stage t to stage $t+1$ flips $\nu+1$ bits in the representation of the redundant binary counter, the potential ϕ decreases by at least $2^\nu-\lambda$.*

Proof. The decrement operation, if it flips $\nu+1$ bits, causes the following change:

$$(\ldots,\tilde{b}_\nu,0,\ldots,0)\longrightarrow(\ldots,\tilde{b}_\nu-1,1,\ldots,1)$$

and the potential changes as follows: Before the decrement, we have, for all $\ell\le\nu$,

$$\begin{aligned}
\phi_\ell(t) &= \left|2^\ell-1-\sum_{i<\ell}0\cdot 2^i\right| \\
&= 2^\ell-1.
\end{aligned}$$

After the decrement, the potential at the same positions become:

$$\begin{aligned}
\phi_\ell(t+1) &= \left|2^\ell-1-\sum_{i<\ell}1\cdot 2^i\right| \\
&= 0.
\end{aligned}$$

All potentials associated with positions $\ell > \nu$ increase or decrease by 1. Thus the difference in the total potential $\phi(t)$ is the same as for the Increment operation. $\square$

Theorem 2.5. *Consider a redundant binary counter, and assume that the cost of flipping $\nu+1$ bits is 2^ν. Then a sequence of k INC and DEC operations has amortized cost $O(\log k)$ per operation.*

Proof. The initial value $\phi(0)$ of the potential function is $\sum_{\ell\leq\lambda}(2^\ell - 1)$ which evaluates to $2^{\lambda+1} - 1 - \lambda = \Theta(k)$. This adds only a constant cost per operation.

For both INC and DEC, the flip cost is entirely paid for by the decrease in potential (by the previous two lemmas), except for at most $\lambda = \lceil \log k \rceil$, which is therefore the amortized cost of one operation. $\square$

Thus, the redundant binary counter based data structure dynamize the geodesic triangulation data structure with an amortized cost of $O(\log k)$ per operation.

2.3 Finding the cycle

The algorithm of Section 3 hinges on ray-shooting against a *connected* polygonal line. It is not obvious how to extend the algorithm to work within a comparable time bound when multiple disjoint cycles can be present. In order to apply our technique on multiple cycles of the convolution we connect these cycles together using extra paths to form a single cycle. We then use the loop excision process on the composite cycle thus formed. This method correctly outputs the outer face of the Minkowski sum iff each of the extra paths that were added lie inside the Minkowski sum (otherwise the output would contain areas covered by the extra paths that were not inside the Minkowski sum). Thus if the paths connecting the cycles are known to lie inside the Minkowski sum, then the outer face will remain unchanged when we apply the loop excision process to the composite cycle. The running time of the algorithm depends on the total length of the composite path, namely the size of the convolution plus the length of the connecting paths.

Let us now see how to connect the cycles of the convolution. It is easy to see this process visualized on the fiber product of the input polygons. If the input polygons P and Q are of sizes m and n the fiber product $\mathrm{FP}(P,Q)$ is visualized on a rectangle of size $m \times n$. A grid point in the rectangle is a pair of vertices (p,q) of P and Q, and corresponds to their vector sum $(p+q)$. By definition, $(p+q)$ lies in the Minkowski sum of P and Q. Thus, a path connecting two points in the fiber product corresponds to a path lying inside the Minkowski sum. Connecting the cycles of the fiber product hence amounts to connecting cycles of the convolution using paths that lie in the Minkowski sum.

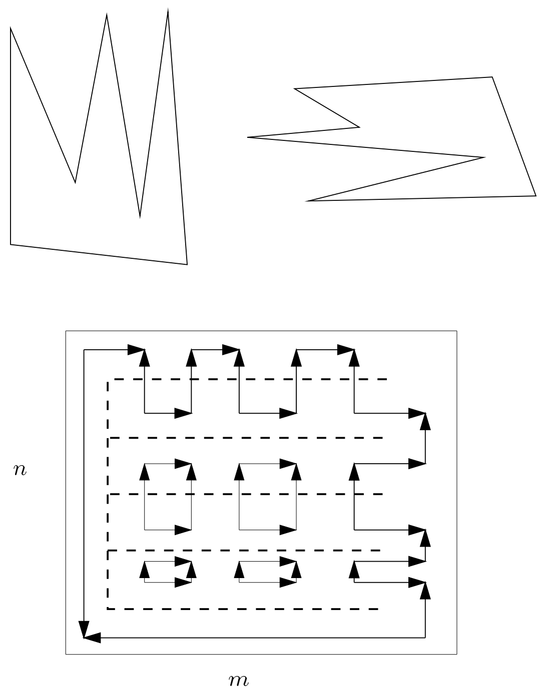

Figure 2.13: Two polygons and their convolution fiber product and a superimposed comb to connect the cycles.

Let us now see how to connect cycles of the fiber product. Suppose there are ℓ cycles in the fiber product. Overlay a comb with $\sqrt{\ell}$ horizontal teeth and a vertical spacing of $\frac{n}{\sqrt{\ell}}$, as in Figure 2.13. Now connect each cycle of the fiber product to its nearest tooth using a vertical segment. This construction connects together all the cycles. There are $\sqrt{\ell}$ teeth in the comb each of length m yielding a total comb length of $m\sqrt{\ell}$. Each of the l cycles can be connected to a tooth with a vertical segment smaller than $\frac{n}{\sqrt{\ell}}$, yielding a total length of at most $n\sqrt{\ell}$. The final length of all the added paths is therefore $(m+n)\sqrt{\ell}$.

We can show that this (the length of connecting paths that lie on the fiber product) is optimal to within a constant factor. To prove this claim we provide a lower bound: an arrangement of the fiber product cycles that require connecting paths of length at least $(m+n)\sqrt{\ell}$. In particular, if the fiber product cycles are arranged in a regular $\sqrt{\ell} \times \sqrt{\ell}$ sub-grid of the $m \times n$ fiber product grid, then the connecting paths contain at least $m\sqrt{\ell}$ horizontal segments and at least $n\sqrt{\ell}$ vertical segments adding up to a minimum total length of at least $(m+n)\sqrt{\ell}$ of any set of interconnecting paths (this can be seen from the fact that the interconnecting paths form a minimum spanning tree on the cycles — see [8] for lower bounds on the path length of minimum spanning trees). Figure 2.14 shows an example of a spanning tree that connects the cycles in this configuration. Using these connecting paths (and possibly doubling them over), we can now easily form a single cycle whose edges include all the original edges of the convolution cycles.

We can now compute the total running time of our loop-cutting algorithm for a convolution with multiple cycles. The length of the polygonal line traversed is $O(k+(m+n)\sqrt{\ell})$. Amortized cost of ray-shooting and updating the polygonal line is $O(\log^2(k+(m+n)\sqrt{\ell}))$ which simplifies to $O(\log^2(m+n))$ since $k \leq mn$. Hence the total cost of finding the outer face of the Minkowski sum is $O((k+(m+n)\sqrt{\ell})\log^2(m+n))$.

Note that this bound remains roughly quadratic even when $\ell = \Theta(k) = \Theta(mn)$. Though a quadratic number of cycles is possible, we expect that in most practical situations ℓ is likely to be quite small. As observed earlier this bound is, however, strictly weaker than the running time bound of a variant algorithm — namely, compute the convolution edges

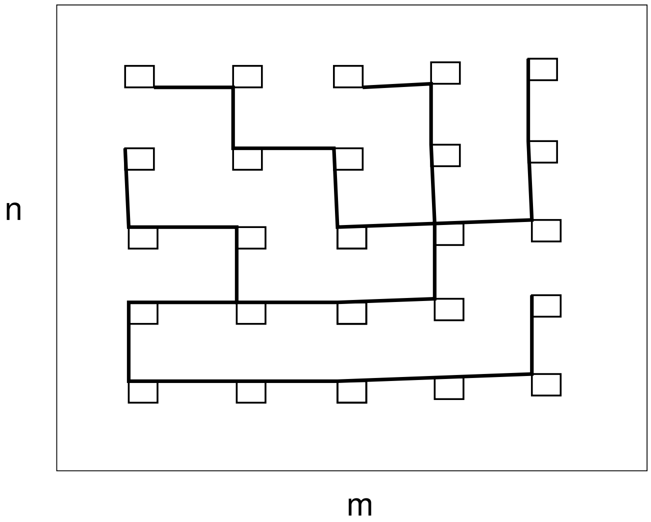

Figure 2.14: Picture of an $m \times n$ arrangement with a $\sqrt{\ell} x \sqrt{\ell}$ sub-grid and a "typical" spanning tree on this arrangement. It can be verified that the spanning tree is of the required length $O((m+n)\sqrt{(\ell)})$.

and find the outer face by a single face computation among a set of line segments using the method of [72]. This variant algorithm takes only time $O((k+m+n)\log(m+n)\alpha(m+n))$. Our algorithm performs well for most realistic input instances where k as well as ℓ are small.

2.4 Conclusion

In this chapter we have presented a method to use the convolution of two simple polygons to find the outer face of their Minkowski sum in time $O((k+(m+n)\sqrt{\ell})\log^2(m+n))$, where m, n are the sizes of the polygons, k is the size of their convolution, and ℓ is the number of cycles in the convolution. Three key ingredients went into our algorithm. We summarize each of these below.

We adapted the geodesic triangulation data structure for ray-shooting queries against a polygonal line of size k to work in amortized time $O(\log^2 k)$ in a dynamic context. The adaptation used a redundant binary counting system that allowed dynamic insertions and deletions in amortized logarithmic time. We showed how this system can be used to divide the polygonal line into multiple pieces, each storing a copy of the geodesic triangulation data structure. The amortization bound on the counting system ensures that the data structures are not created or updated too many times. In fact, the maintenance and queries are proved to take an amortized time bound of only $O(\log^2 k)$.

If one of the input polygons is convex, then the convolution has a single cycle. We proved a topological lemma that allows us to compute the outer face by traversing this cycle and discarding any loops formed by self-intersections. Loops are minimal cyclical portions of the convolution formed by self-intersections of the convolution boundary. The loops are discovered by performing ray shooting queries (amortized time $O(\log^2 k)$) on the dynamic ray shooting structure described above. The total running time to traverse a single cycle of the convolution and compute the outer face of the Minkowski sum using this technique is $O(k \log^2(m+n))$. This method works whenever the convolution has a single cycle (even if both polygons are non-convex).

In the general case, our algorithm uses an additional technique to connect the cycles of the convolution into a single cycle using paths inside the convolution of total length $O((m+n)\sqrt{\ell})$, and then applies the single-cycle technique, giving a running time of $O((k+(m+n)\sqrt{\ell}) \log^2(m+n))$. The method to connect the cycles uses a minimum spanning tree method to string together the cycles. The cycles are visualized on an alternative structure called the fiber product that makes it easier to visualize the cycles than in the convolution itself.

An interesting open problem here is whether the cycles can be connected using paths of length $O(m+n+l)$. Our method only considers paths in the fiber product, which correspond to pairs of points on the boundary of P and Q. For such pairs the lower bound example mentioned earlier applies, but it may be possible to do better if we allow pairing of points *inside* P and Q. Another problem raised is whether ray-shooting queries against a dynamic polygonal line of length k can be performed in $O(\log k)$ time. At least, it seems that a better balancing of the query time and update time could be achieved with a redundant counter where the "2" is replaced by a base dependent on n.

A theory of polyhedral tracings and their convolution is discussed in the next chapter [23]. An interesting question for polyhedral tracings is whether the relationship between the convolution and the Minkowski sum explored in this chapter can be extended to three dimensions, and whether a similar algorithm can be formulated to compute the outer face of the Minkowski sum. This question remains unanswered in this thesis.

Chapter 3

Polyhedral tracings

This chapter begins the formal development of the *kinetic framework* for Computational Geometry in three dimensions; we introduce the precise notion of polyhedral tracings. To help motivate the extension of tracings to three dimensions, we briefly recall the kinetic framework for Computational Geometry in two dimensions [70]. The authors of [70] augment the standard boundary representation of planar shapes via polygons or closed curves with certain additional information, to make them into closed *tracings*. The advantage of tracings is that they admit a multi-linear calculus of operations that generalizes standard boolean operations on shapes such as union, intersection, set difference, etc. A particular accomplishment of the kinetic framework was to define the operation of *convolution* on planar tracings and to illustrate its use in a variety of algorithmic problems. Loosely speaking, the convolution $B * R$ of two tracings B (the barrier) and R (the robot) is another tracing that encodes the interaction of B with all possible translations of a copy of R mirrored through the origin. The convolution allows us to transform problems involving two tracings B and R into problems about a point and a composite tracing, the convolution $B * R$. This composition concept, except for the advantageous multi-linear structure of the convolution, is familiar in robotics, where it appears under the names of Minkowski sum or configuration space obstacle [98, 93].

In two dimensions, a closed curve is normally defined via a continuous map from the unit circle to the plane. To augment the curve into a tracing, we make a value of this map be not only a point in the plane, but an associated direction as well, with the two related by a 'no-slipping' condition. Intuitively, we can think of a tracing as being drawn by a tiny car; the car can move forward, move backward, or turn while standing still, but may not skid sideways. A standard curve can be given an orientation at each point, corresponding to the sequence in which the points of the curve are traversed by the car. A tracing in addition specifies the direction the car is facing at each point, and that direction must be along the tangent to the curve whenever the latter is defined. The combination of (location, direction) is called a *state* in [70]. The direction of the car is the crucial new component that allows the definition of the convolution operation and gives it its desirable properties. The convolution is defined as a fiber product in [70], by matching states from the two factor tracings with the same direction, and then assembling those into a tracing.

Before this thesis almost nothing was known about the convolution operation for tracings in dimensions higher than two, except in the special case of convex bodies, where it is equivalent to the Minkowski sum; this case for 3-D polyhedral objects has been treated in [71, 17, 87, 119]. Unfortunately, the convex case is too special to suggest the proper extension of the concept of a tracing to three and higher dimensions. If, in two dimensions, one thinks of a tracing as a rule for assigning winding numbers to the elements (faces, but also edges, and vertices) of the planar subdivision defined by the tracing, then we get a structure we call a *painting*. Paintings or arbitrary dimension were studied by Schapira [121] under the name of *constructible functions*. Using results from sheaf theory, Schapira was able to define a convolution operation on constructible functions and to show how it corresponds to the convolution of tracings discussed above. Nevertheless, Schapira's method does not give a definition of a tracing in higher dimensions, and as such it is not directly useful in algorithmic problems. A tracing can be thought of as a boundary representation of a painting; because of this, it can be cheaper to represent, since it need not encode features, such as self-intersections, that are artifacts of the embedding of the domain manifold into the range space, and not features of the mapping itself.

The main contribution of this chapter is the notion of *polyhedral tracings*, which extends the classical notion of a polyhedron in exactly the same way that a polygonal tracing extends the notion of a polygon. We also define the convolution $B * R$ of two polyhedral tracings B and R. The technical challenge in getting this theory to work lies in how to extend normal polyhedra into tracings and, less obviously but equally importantly, how to orient the elements of the fiber product defining the convolution so that together they form another tracing manifold. We describe a data structure for representing polyhedral tracings based on the *half-edge* data structure of Muller and Preparata [106] augmented with a sign. The notation used in this presentation was originally presented for quad-edge structures of [73]. We give a set of rules to derive the representation of the convolution from the representations of two input polyhedral tracings.

We prove in Section 3.3.3 of this chapter that the set of rules we propose to compute the convolution of two polyhedral tracings defines a consistent augmented half-edge structure.

This chapter is organized as follows. Section 3.1 introduces the general framework of tracings and paintings. Section 3.2 defines polyhedral tracings and introduces a data structure for representing them. Section 3.3 describes the convolution of two polyhedral tracings in terms of these data structures. Section 3.4 concludes the chapter.

3.1 Tracings and paintings

In this section, we generalize to three dimensions the notion of *tracings* introduced in the kinetic framework, and relate that notion to Schapira's constructible functions, or *paintings*.

We briefly recall his convolution operation on paintings and its relation to the Minkowski sum, and then proceed to define an a convolution operation on tracings.

In order for the definitions below to make sense, we need to place ourselves in a consistent framework. Schapira chose the most general framework of sub-analytic sets and functions on compact real analytic manifolds. However, for the purpose of defining polyhedral tracings, we can assume that the domain manifold of a function has a finite cell decomposition such that in each cell the function can be expressed as an algebraic function.

3.1.1 Tracings

In the kinetic framework, a point on a curve is enriched into a state by the addition of a direction vector that can point either forward or backward along the curve. In three dimensions, we enrich a point on a surface into a state by adding a *whisker* vector at that point, that can point along either of the two directions normal to the surface.

More formally, a tracing R is given by an *oriented compact two-dimensional manifold without boundary* M_R, together with a pair of smooth functions: the location map $r : M_R \to \mathcal{E}^3$, and the whisker map $\dot{r} : M_R \to \mathcal{S}^2$. The two are related by a *no-slipping* condition:

$$\forall x \in M_R, \operatorname{Im} T_x r \perp \dot{r}(x),$$

where $\operatorname{Im} T_x r$ is the image of the differential of r at x. That is, the location map r is restricted locally to move only in directions orthogonal to $\dot{r}$.

The orientation of the manifold M_R can be thought of as a tiny circle rotating around each of its points. If the image under the location map of this circle around x is replaced by a rotating right-handed corkscrew, the direction in which it starts moving will be referred to as *locally outward* for M_R at x and the opposite direction will be called *locally inward*.

There are two senses in which the whisker map $\dot{r}$ gives more information than the differential of the location map:

- When the tangent space $\operatorname{Im} T_x r$ is a plane, the whisker provides exactly one bit of extra information, by selecting either the locally inward or the locally outward normal to this plane.
- The whisker map smoothly extends the notion of tangent plane to ridges and corners (where $\operatorname{Im} T_x r$ is only a line or a point). In these cases the whisker map defines an arc or a region in $\mathcal{S}^2$ respectively.

3.1.2 Paintings

A painting ϕ on $\mathcal{E}^3$ is a function that associates an integer to each point of $\mathcal{E}^3$, in a well-behaved fashion. The paintings we will deal with correspond to polyhedral subdivisions of $\mathcal{E}^3$ with an associated constant integer value for each feature (vertex, edge, face, chamber) of the subdivision. For instance, there are two natural paintings that can be associated with a polyhedron P. The *closed painting* of P has value 1 everywhere inside the polyhedron and on its boundary, and zero outside. The *open painting* of P is the same except for a value of 0 on the boundary. A painting ϕ is said to be *locally closed* at $p \in \mathcal{E}^3$ if there exists an open neighborhood A of p on which

$$\max_{q \in A} \phi(q) = \phi(p).$$

Likewise, we define *locally open* by replacing max with min. These definitions generalize the notions of open and closed sets to paintings. Since the overlay of spatial subdivisions is another spatial subdivision, multi-linear operations on paintings can be defined in an obvious point-wise fashion.

Any tracing F has an associated painting ϕ, defined via the *winding number* function. We can compute the winding number at a point p of $\mathcal{E}^3$ that does not lie on the range of the location map $r(M_R)$ of F as follows. Take a smooth path π from p to infinity, and count $+1$ or -1 each time the path crosses $r(M_R)$. We count the crossing as $+1$ if the 'velocity vector' of π at the crossing point is directed locally outwards (as defined by the local orientation), and -1 otherwise. The total count, which is the winding number, is independent of the path chosen, and thus defines a unique value except for those points that lie on $r(M_R)$. If p is such a point, one would like to perturb the location map by a tiny amount so as to move its image away from p, and proceed as before. One role of the whisker map $\dot{r}$ is to indicate the direction in which this perturbation should happen: the winding number is computed not with respect to r, but with respect to $r + \epsilon\dot{r}$, for some small enough $\epsilon > 0$. This perturbed tracing transforms corners and ridges into portions of spheres and cylinders. The winding number is thus established on every point of $\mathcal{E}^3$ and defines a painting ϕ associated with F.

The distinction between open and closed sets is crucial in Schapira's theory of constructible functions. Tracings also make this distinction, and it is one of their original features as a boundary representation of solids. Consider a point $p \in \mathcal{E}^3$ and assume, to simplify matters, that it has only one pre-image under r, say $x \in M_R$ with $p = r(x)$. If the whisker $\dot{r}(x)$ at p points locally outward, the associated painting is locally closed at p, otherwise it is locally open.

3.1.3 Convolution

Any polyhedral painting ϕ can be triangulated, and thus it is not hard to see that it can be expressed as a finite weighted sum of indicators of closed, contractible sets. Such a representation, however, is not unique. Schapira defines the *integral* of ϕ, $\int \phi$ as follows: represent ϕ as a finite sum of closed, contractible sets multiplied by coefficients. The algebraic sum of such coeficients is the integral of ϕ. Schapira shows that the integral is independent of the choice of representation. This integral has a familiar interpretation in certain cases. For example, if ϕ is the indicator of a closed set S, this integral is simply the Euler-Poincaré characteristic of the set S, i.e. in three dimensions, the number of components, minus the number of tunnels, plus the number of voids of S.

The painting convolution of two paintings ϕ, ψ is then defined with respect to this integral $\int$, as

$$(\phi * \psi)(p) = \int \lambda q \cdot \phi(q)\psi(p-q),$$

and it relates to intersection problems in the following way [121]:

Theorem 3.1. *If ϕ, ψ are indicators of closed sets B, R, then the value of $\phi * \psi$ at a point $p \in \mathcal{E}^3$ is the Euler-Poincaré characteristic of the intersection of B with R reflected through the origin and translated by p.*

We now wish to describe a convolution operation on tracings such that, given tracings B and R (blue and red) that define paintings ϕ and ψ, the convolution $Q = B * R$ defines the painting $\phi * \psi$. Disregarding the topology of tracings, and viewing them as bags of states as in [70], we define a convolution operation exactly like in two dimensions: any two points $x \in M_B, y \in M_R$ define a state in the convolution only if their whiskers match. In that case, the output feature keeps the same whisker and its location map is $b(x) + r(y)$.

In our framework for tracings, the convolution domain is defined to be that subset of $M_B \times M_R$ satisfying:

$$M_Q = \{(x, y) \in M_B \times M_R \mid \dot{b}(y) = \dot{r}(x)\}.$$

This is an instance of a general categorical construction known as a *fiber product* [92] applied to the two whisker maps $\dot{b}$ and $\dot{r}$. Under mild assumptions (*transversality*) on the pair $(\dot{b}, \dot{r})$, the fiber product M_Q is an oriented manifold of the same dimension as M_B and M_R, which makes the above definition a valid tracing. Warning: (1) the topology of the fiber product is not simply related to that of M_B and M_R; for instance, the fiber product can have any number of components even if M_B and M_R are connected, (2) deriving a consistent local orientation rule for M_Q is a non-trivial problem which is fully addressed in [24]. In this thesis, we limit ourselves to showing that the convolution operation for polyhedral tracing representations gives a consistent orientation rule, i.e. that the resulting object represents a manifold with consistent location and whisker maps.

Note that the winding number at a point with respect to the tracing was defined after perturbing the tracing slightly at every point. The perturbed tracing can be viewed as the convolution of the original tracing with a small ball of radius ϵ.

3.2 Polyhedral tracings

In this section, we define a polyhedral tracing as a specialization of a general tracing. We describe an efficient way to represent a polyhedral tracing with a *signed oriented half-edge* structure, and show how an ordinary polyhedron can be made into a tracing using a sound set of conventions.

3.2.1 Relation to previous discussion

Before we get into the technical details of what a polyhedral tracing is, we review briefly the definition of tracings as used in the previous chapter on planar tracings. We attempt to bring out clearly the extra complexity in defining polyhedral tracings.

In planar tracings, it was sufficient to store the set of cycles of moves that comprise the tracing. The turns of the tracing were implicit, assuming that they were less then π, and hence uniquely determined by the two moves around them. The cycle could be an ordinary circular linked list because it was simply a closed sequence of moves. Each move itself comprised a constant amount of information: the origin of the move, the destination, and the sign of the move, that is the direction in which the move was traversed (whether it was a forward move or a backward move). The turns also had signs specifying whether they were clockwise or counter-clockwise turns.

So, it is possible to store a planar tracing using a simple data structure as described above. There is no need for a specialized data structure. The situation changes for polyhedral tracings. In this chapter we will explain why special data structures are required to store polyhedral tracings; we then go on to explain what these data structures are.

In the case of planar tracings, it was not necessary to explicitly store the domain manifold of the tracing and the whisker map associated with the tracing. These are stored implicitly as long as the moves and their signs are specified. The domain manifold is simply a collection of cycles as described by the moves, and the whisker map consists of the set of turns at the vertices of the tracing. We note that a special case of a cycle is a whole turn at a vertex and a null set of moves. Both the manifold and the whisker map are quite simple and are easily derived from a linked list representation of the tracing.

For polyhedral tracings, however, there is a need to explicitly store the domain manifold. This is because the domain manifold is complex — a closed planar subdivision with edges, vertices, and faces. The whisker map also has to be explicitly stored at each of these features (features refer to edges, vertices, or faces). The domain manifold and the whisker are not easy derivatives of a simple representation of the tracing, as is the case in planar tracings. We explain how the domain manifold can be explicitly stored, and how the data structure can be augmented to store the whisker map along with the features of the tracing.

3.2.2 Polyhedral tracings: Intuitive approach

A polyhedral tracing is a restricted form of tracing in the framework of piecewise linear geometry. It is decomposable into three types of features: faces, edges, and vertices—just as a polyhedron is (the two-dimensional analogue of this is a polygonal tracing that is decomposable into moves/edges and turns/vertices). Using a tracing to describe a polyhedron has several advantages. Firstly, a tracing can encode both open, closed, and mixed subsets of $\mathcal{E}^3$, whereas no such distinction exists for a polyhedron. Secondly, a tracing can represent a larger variety of paintings with piecewise planar boundaries, and not just indicators of polyhedra. And thirdly, the convolution of two (closed) polyhedra is in general not a polyhedron. Polyhedral tracings have in fact just the right expressiveness to allow us to represent the convolution of general non-convex polyhedra. We present a formal definition of polyhedral tracings later in this chapter. We begin with a discussion of a boundary data structure called the half-edge structure [106] that we use to represent polyhedral tracings.

3.2.3 Half-edge structures

A half-edge structure [106] (recast in the postfix notations of [73]) is an algebra $(V, E, F, \mathrm{Lnext}, \mathrm{Org}, \mathrm{Lface}, \mathrm{Sym})$, where V (resp. E, F) is the set of vertices (resp. edges, faces), and

$$\begin{aligned} \mathrm{Lnext}: &\quad E \to E \\ \mathrm{Org}: &\quad E \to V \\ \mathrm{Lface}: &\quad E \to F \\ \mathrm{Sym}: &\quad E \to E \end{aligned}$$

where Lnext and Sym are one-to-one (so that their inverses are well defined), and with the following relations, called the *fundamental half-edge relations*:

$$e\,\mathrm{Sym}\,\mathrm{Sym} = e \tag{3.1}$$

$$e\,\mathrm{Lnext}\,\mathrm{Lface} = e\,\mathrm{Lface} \tag{3.2}$$

$$e\,\mathrm{Lnext}\,\mathrm{Org} = e\,\mathrm{Sym}\,\mathrm{Org} \tag{3.3}$$

A half-edge structure is *oriented* if, for any edge e, there is no edge e' with the same origin and with $e'\,\mathrm{Lface} = e\,\mathrm{Rface}$. In this case, the orientation of an edge determines its direction and vice-versa.

We now define some additional notation. In a half-edge structure, an edge has a *direction*, from its *origin* (Org) to its *destination* (Dest), and an *orientation*, given by its *left face* (Lface) and *right face* (Rface). The *symmetric* (Sym) of an edge is the same edge with direction and orientation reversed. The *next edge* (Lnext) to an edge e whose left face is a face f is the edge with same left face and whose Org is the Dest of e. See Figure 3.1.

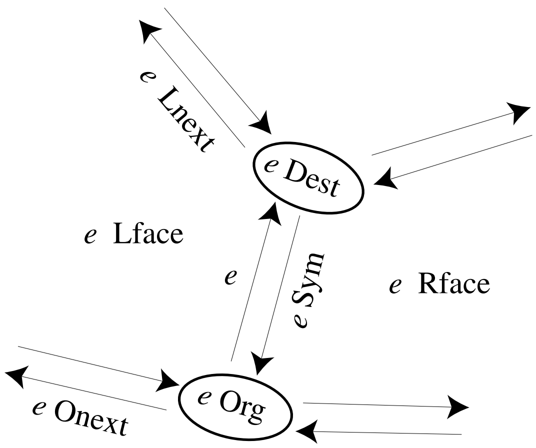

Figure 3.1: Operations that relate to an edge e of a half-edge structure.

The other primitives introduced above are syntactic sugar that can be defined with these four fundamental ones:

$$
\begin{aligned}
\mathrm{Rface} &= \mathrm{Sym}\,\mathrm{Lface} \\
\mathrm{Dest} &= \mathrm{Sym}\,\mathrm{Org} \\
\mathrm{Lprev} &= \mathrm{Lnext}^{-1} \\
\mathrm{Onext} &= \mathrm{Lprev}\,\mathrm{Sym} \\
\mathrm{Oprev} &= \mathrm{Onext}^{-1} \\
\mathrm{Dprev} &= \mathrm{Dnext}^{-1} \\
\mathrm{Dnext} &= \mathrm{Sym}\,\mathrm{Lprev}
\end{aligned}
$$

For an edge e, we also use the alternate notation $e\,\mathrm{Sym} = -e$. From the fundamental relations, several other relations can be deduced, for instance:

$$\begin{aligned}
\mathrm{Onext}\,\mathrm{Rface} &= \mathrm{Lface}\\
\mathrm{Onext}\,\mathrm{Org} &= \mathrm{Org}\\
\mathrm{Dnext}\,\mathrm{Dest} &= \mathrm{Dest}\\
\mathrm{Sym}\,\mathrm{Dnext} &= \mathrm{Onext}\,\mathrm{Sym}\\
\mathrm{Lprev}\,\mathrm{Sym}\,\mathrm{Org} &= \mathrm{Org}\\
\mathrm{Oprev}\,\mathrm{Org} &= \mathrm{Org}\\
\mathrm{Dprev}\,\mathrm{Dest} &= \mathrm{Dest}\\
\mathrm{Sym}\,\mathrm{Dest} &= \mathrm{Org}\\
\mathrm{Sym}\,\mathrm{Rface} &= \mathrm{Lface}\\
\mathrm{Sym}\,\mathrm{Onext} &= \mathrm{Dnext}\,\mathrm{Sym}\\
\mathrm{Oprev}\,\mathrm{Lface} &= \mathrm{Rface}\\
\mathrm{Onext}\,\mathrm{Sym}\,\mathrm{Lface} &= \mathrm{Lface}\\
\mathrm{Oprev}\,\mathrm{Sym}\,\mathrm{Rface} &= \mathrm{Rface}\\
\mathrm{Dnext}\,\mathrm{Lface} &= \mathrm{Rface}\\
\mathrm{Dprev}\,\mathrm{Rface} &= \mathrm{Lface}\\
\mathrm{Dnext}\,\mathrm{Sym}\,\mathrm{Lface} &= \mathrm{Lface}\\
\mathrm{Dprev}\,\mathrm{Sym}\,\mathrm{Rface} &= \mathrm{Rface}
\end{aligned}$$

Relations 3.3 and 3.4 can be put together into one single relation. If $s = \pm 1$,

$$(s(e\,\mathrm{Lnext}^s))\,\mathrm{Org} = (-se)\,\mathrm{Org}\,. \tag{3.4}$$

We use an oriented half-edge structure to represent the topology of the oriented manifold underlying a polyhedral tracing. In this representation, a vertex, an edge, and a face all represent two-dimensional features in the manifold. Notice the contrast from the original use of the half-edge where a vertex (resp. edge) represents a zero-dimensional (resp. one-dimensional) feature of a subdivision (Figure 3.2).

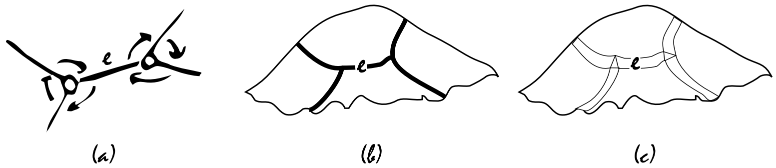

Figure 3.2: (a) A portion of a half-edge structure; (b) this structure viewed as a subdivision; (c) The domain manifold needed for a tracing.

We are now ready to formally define polyhedral tracings and discuss how to represent them.

3.2.4 Polyhedral tracings: Formal definition

A polyhedral tracing is a tracing such that its manifold can be decomposed into interior disjoint simply connected subsets of three types: vertex domains, edge domains and face domains, with connectivity properties similar to a subdivision defined in [73], and such that (Figure 3.3):

1. On a face domain, the whisker map is constant and the location map is a bijection whose range is a simple polygon,
2. On a vertex domain, the location map is constant; the whisker map is a bijection, and its range is a simple spherical polygon on $\mathcal{S}^2$ entirely contained within a hemisphere (*hemisphere assumption*),
3. On an edge domain, the location map range is a line segment (at the intersection of two faces) and the whisker map range is an arc of a great circle of angle less than π.

Thus, a polyhedral tracing is a tracing whose manifold is decomposable into disjoint simply connected subsets each of which is piecewise linear, that is a vertex, edge, or a face, and satisfies adjacency requirements: each vertex has a ring of faces around it, every edge has two faces adjacent to it, and every face has a set of vertices around it.

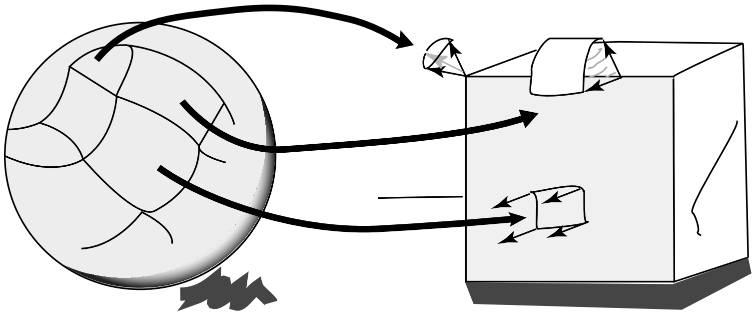

Figure 3.3: A cube tracing indexed by a sphere. Note that the inverse image under the whisker map of points on faces, edges, and vertices of the cube are zero-dimensional, one-dimensional, and two-dimensional respectively.

3.2.5 Representing polyhedral tracings

A polyhedral tracing can be compactly encoded in half-edge structure enriched with a single bit for each face. From this information, the full tracing can be reconstructed up to isomorphism of the indexing manifold. Recall that at a point where the location map is non-singular (a face), the whisker encodes exactly one bit of information: whether it points locally inward or outward. In an oriented half-edge, the outward direction can be obtained from the orientation of the ring of edges, by the "right-handed corkscrew rule". To represent a polyhedral tracing, we therefore need only a half-edge structure and a sign bit for each face. More precisely, we have a half-edge structure $(V, E, F, \dots)$, together with the following geometric information:

$$\begin{aligned} p: \; V &\rightarrow \mathcal{E}^3 \\ \sigma: \; F &\rightarrow \{-1, +1\} \\ \nu: \; F &\rightarrow \mathcal{S}^2 \end{aligned}$$

where p gives the location of each vertex, and ν gives the whisker at each face (a unit vector), such that $\sigma(f)\nu(f)$ points locally outward at f. One way to compute $\nu(f)$ is as follows: pick an edge e whose Lface is f, and let u be a vector that locally points from e towards f. Then $\nu(f)$ is the unit vector orthogonal to f such that the triplet

$$(e\,\mathrm{Dest} - e\,\mathrm{Org}, u, \sigma(f)\nu(f))$$

is a positively oriented basis.

Obviously, some information is lost in this representation: the original domain manifold of the polyhedral tracing is known only up to an isomorphism, but this is enough for our purposes. The location map can be smoothly extended from the data for vertices to edges and faces, and this extension is unique up to an isomorphism, due to the assumption that the image of the location map is a bijection on a face domain.

The reconstruction of the whisker map for edges and vertices from the whiskers of the faces is more delicate. The hemisphere assumption on the whisker map for vertices was introduced to allow for an unambiguous reconstruction at vertices. Consider first the case of an edge. Its whiskers will be an arc joining the whiskers of its two adjacent faces, but there are two such arcs. However, only the smallest of the two satisfies the requirement that the whisker of an edge be an arc of angle less than π (if the faces are flat and coincident, we choose the whisker set counterclockwise with respect to the edge). For a vertex domain, the assumption that the whisker map is a bijection allows its specification using the values on its boundary, but this boundary defines two polygons on the sphere. It is the hemisphere assumption that allows to choose one of the two without ambiguity — we choose the smaller polygon.

For a given arc on the boundary of such a simple spherical polygon, it is impossible to determine locally on which side of the arc the polygon lies. It may be convenient to store this information for each edge e, to know not only on which side its Lface is in the plane normal to $\nu(e\,\mathrm{Lface})$, but also on which side of its whisker set can be found the whisker set of its origin vertex. In the next subsection, we define a set of signs for edges and vertices that precisely encode this information, and are used to develop a sign rule for the convolution.

3.2.6 From a polyhedron to a polyhedral tracing

In most applications, a polyhedron is defined by its boundary, encoded by an oriented half-edge structure. This structure can be easily converted into a polyhedral tracing, but there are two issues which require some discussion: what signs to choose for the faces, and how to enforce the hemisphere requirements for the whisker map.

The choice of signs for faces depends on whether the polyhedron is intended to represent a topologically closed or open set. While this distinction is not often considered in computational geometry, it may have relevance in solid modeling. As was seen before, having the whisker map point locally outwards defines a locally closed painting. Thus, in order to represent a topologically closed polyhedron, we set the sign bit of all faces to $+1$. To represent an open polyhedron, we set the sign bit of all faces to -1. This sets the values of the associated painting on all faces, but it remains to check whether the edges and vertices also have the correct values. We address this question and verify the other requirements of a polyhedral tracing.

The choice of signs defines the whisker on each face. We take the convention that the whisker set of an edge is the shortest arc that joins the whiskers of its adjacent faces.

Therefore, there is no need to explicitly represent edges in the tracing. This convention mirrors a similar one used for turns in two dimensions [70]. It get its full relevance from the open/closed interpretation of tracings. Indeed, with this choice, an edge adjacent to two closed faces is closed (i.e. the painting has the same value on the edge as in the polyhedron interior). An edge adjacent to two open faces is open. Note that in the special case of adjacent coplanar faces with opposite whiskers, the whisker set is a full turn of π counterclockwise with respect to the interior.

The vertex case is more delicate. Indeed, the path stroked by the whiskers of the adjacent edges might not fit in a hemisphere (Figure 3.4), or might even self-intersect (Figure 3.5), hence requiring a *normalization process*. There is no canonical way to normalize, but the simplest is to cut the vertex in pieces by introducing zero-length edges between one of the nodes and all other nodes in the path. This leaves only triangles that are simple and contained within one hemisphere. All normalizations define the same painting, but they can affect the running time of later convolutions (see later discussion in the convolution section). The normalization process is required to restrict the tracing so that it satisfies the assumptions stated earlier. If the normalization was not performed, we would not be able to assume that each vertex whisker sets is included in a single hemisphere. See Figure 3.5 for two possible ways to normalize a vertex with a self-intersecting path of whiskers.

Once the tracing corresponding to a polyhedron has been defined, it remains to see whether the vertices have the desired value in the associated painting, i.e. whether they have value 1 for a closed polyhedron, and value 0 for an open polyhedron. This is indeed the case, provided that for each vertex, there exists a hemisphere that contains all the whiskers of the faces adjacent to this vertex. Indeed, consider the closed case, and observe this vertex so as to see the whiskers of all adjacent faces. Say that this direction is “up”. Whichever normalization is used (if any is needed), all whiskers of states close to this vertex will be visible from above, so that a convolution with a ball of radius ϵ will locally raise

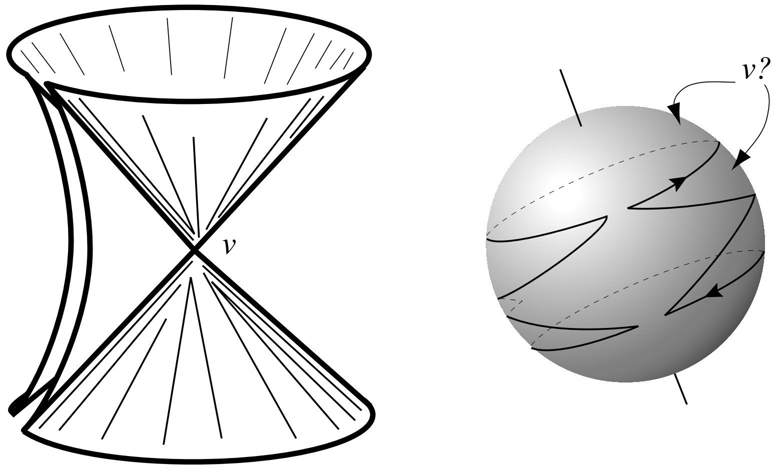

Figure 3.4: A vertex whose whisker map does not fit in a hemisphere. It is not clear which rule should be used to decide what is its inside and what is its outside. The two choices give different values for the corresponding painting at that vertex (0 or 1), and additional assumptions are required to decide which is the right choice. For instance, if we know the painting represents a closed polyhedron, we should get a value of 1 for the painting.

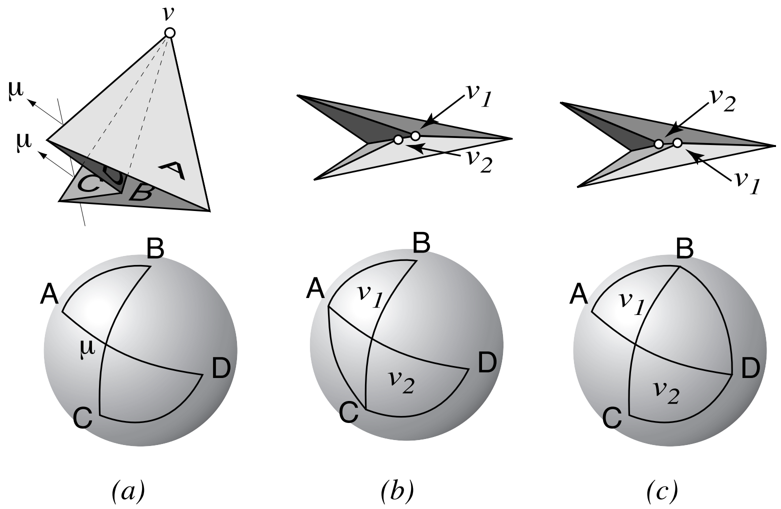

Figure 3.5: (a) The simplest four edges vertex that has a self intersecting path; (b-c) two ways to normalize it. Vertices v_1 and v_2 are both at v, but drawn slightly displaced to show the edge connectivity. Two different normalizations might give convolutions of very different size, as is seen in the next section.

the entire tracing. Hence the winding number at the vertex is the same as that of a point slightly inside the polyhedron. If the hemisphere requirement is not satisfied at a vertex (Figure 3.4), it is unclear how to choose the whisker map so that the corresponding painting has value 1 at that vertex.

3.3 Convolution of polyhedral tracings

Since a polyhedral tracing is a special case of a tracing as defined in Chapter 2, the convolution operation is well defined. Moreover, the result is also a polyhedral tracing. Given the signed half-edge representations of two input polyhedral tracings B and R, we show in this section how to obtain the convolution Q in the same representation. In the rest of the chapter, we refer to features of R, B, and Q as *red, blue,* and *purple* respectively. The underlying manifold of the convolution is the fiber product of the whisker maps of B and R. It is described implicitly by the signed half-edge structure of Q.

The features of the convolution are obtained by convolving pairs of matching features of B and R. A pair of features is said to *match* if the whisker sets of the features have a non-empty intersection. In order to keep the notation simple, we assume here that the intersection between the whisker sets of any two pairs is simply connected (and thus defines only one feature).

In two-dimensions, a forward move combined with a right turn becomes a backward move in the convolution. Likewise, in three dimensions, we expect that some faces will change sign in the convolution. Say a face f is convolved with a vertex v to generate a face f' in the convolution. As f and f' have the same whisker, a change of sign requires a reversal of the edge ring (Figure 3.6). This is not the only way one may obtain a face in the convolution: when a red and a blue arc intersect on the sphere, their corresponding edges generate a parallelogram face (the Minkowski sum of the two edges). The whisker of this new face is well defined, but its sign has to be determined.

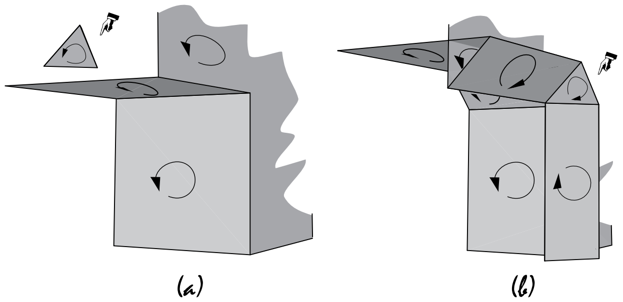

Figure 3.6: In this convolution, the orientation of the little triangle is switched to create a consistent tracing. This reversal is captured by a negative sign assigned to the saddle vertex. Some pairs of edges whose whisker sets intersect create parallelogram faces in the convolution.

3.3.1 Orienting the convolution

In the kinetic framework of two dimensions, the authors captured reversals of moves in a very simple sign rule: left turns have sign $+1$, right turns have sign -1, and the sign of an output feature is the product of the signs of the input features. Deriving a similar sign rule in three dimensions is more difficult. We would like to deduce the sign rule from the general convolution definitions by calculating the orientation of the fiber product manifold on the domains corresponding to each feature of the convolution. The topology of this manifold should also give the connectivity of its half-edge representation. However, in this work, we adopt a direct approach, which consists in guessing a sign rule and a way to connect convolutions features, then checking that these form a consistent tracing. We proceed to define signs for edges and vertices, which are defined for a tracing independently of any other, and thus can be stored in the signed half-edge structure.

1. The sign of an edge e, denoted σ_e, is positive (resp. negative) if the triplet of vectors $(e\,\mathrm{Dest} - e\,\mathrm{Org}, \nu(e\,\mathrm{Lface}), \nu(e\,\mathrm{Rface}))$ defines a positively (resp. negatively) oriented basis. In other words, it is the sign of determinant of the 3×3 matrix whose columns

are the vectors mentioned. Informally speaking, the sign σ_e of an edge e is positive (resp. negative) iff the shortest turn from $\nu(e\,\mathrm{Lface})$ to $\nu(e\,\mathrm{Rface})$ is counterclockwise (resp. clockwise) with respect to the normal defined by the edge going from $e\,\mathrm{Org}$ to $e\,\mathrm{Dest}$.

2. The sign of a vertex v, denoted σ_v, is positive (resp. negative) if the Onext ring of edges around v defines a counterclockwise (resp. clockwise) cycle of arcs around $\nu(v)$, the whisker set at v.

From the above definition, it follows that the sign of an edge e is the same as that of $e\,\mathrm{Sym}$. Also, when a tracing represents a closed polyhedron the above sign rule leads to giving convex edges a positive sign and concave edges a negative sign (much like left and right turns in polygonal tracings). Both convex and concave vertices have a positive sign, whereas a vertex like the one in Figure 3.6 has negative sign (Figure 3.7).

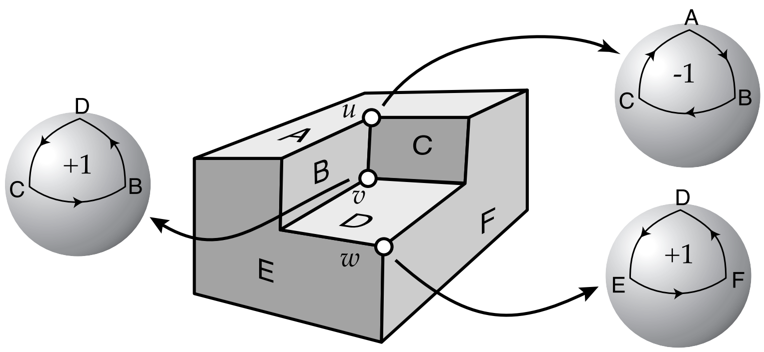

Figure 3.7: The three simplest types of vertices with their ring of edges in the sphere of directions and their signs.

We now know how to compute the sign of a face in the convolution: it is the product of the signs of the two features that generated that face. In the next section, we also define the connectivity information for the convolution tracing, and finally prove that this information gives a consistent tracing, i.e. that the relations (3.1–3.3) are satisfied.

3.3.2 Connectivity of the convolution

Let the vertex, edge, and face sets of B be V_b, E_b, and F_b, and those of R be V_r, E_r, and F_r. Consider a pair (h_b, h_r) of features of B and R. If h_b and h_r match, the resulting convolution feature is denoted by the unordered pair (h_b, h_r). We assume that all edge whisker sets are smaller than π in length. We also assume that all vertices of V_b and V_r each have three incident edges; consequently, their whisker sets are triangles on $\mathcal{S}^2$. Notice that these two assumptions are together stronger than the hemisphere assumption stated earlier (that all vertex whisker sets lie within a hemisphere).

Under the above assumptions, every pair of features produces at most one feature in the convolution. Note that in an implementation, an output feature would be identified by a pointer, so that the unambiguity assumptions would be unnecessary. For conciseness of notation, we denote $-e$ for $e\,\mathrm{Sym}$, allowing expression $\sigma_v\, e$, to refer to e or $e\,\mathrm{Sym}$ depending on the sign σ_v of v.

We now define the signed half-edge structure of Q by enumerating its features and their connectivity:

$$
\begin{aligned}
V(Q) &= \{(v_b, v_r) \in V_b \times V_r \mid \nu(v_b) \cap \nu(v_r) \neq \emptyset\} \\
E(Q) &= \{(e, v) \in E_b \times V_r \ \cup\ E_r \times V_b \mid \\
&\qquad\qquad\qquad\qquad \nu(e)\ \cap\ \nu(v) \neq \emptyset\} \\
F(Q) &= \{(f, v) \in F_b \times V_r \ \cup\ F_r \times V_b \mid \nu(f) \in \nu(v)\} \\
&\cup\ \ \{(e_b, e_r) \in E_b \times E_r \mid \nu(e_b) \cap \nu(e_r) \neq \emptyset\}
\end{aligned}
$$

The unordered pairs (e, e'), $(e\,\mathrm{Sym}, e')$, $(e', e\,\mathrm{Sym})$, and $(e\,\mathrm{Sym}, e'\,\mathrm{Sym})$ refer to the same parallelogram face. The goal in this section is: a) to define the primitives $\mathrm{Lface}, \mathrm{Lnext}, \mathrm{Org}$, and Sym for the convolution features based on the connectivity and b) to define signs of the input features. For instance, we can define Sym of an edge that is the convolution of edge $e \in E_b$ and vertex $v \in V_r$ as:

$$(e, v)\,\mathrm{Sym} = (e\,\mathrm{Sym}, v)$$

In this case, the symmetry of this operator (3.1) follows immediately from the fact that the same relation holds for tracing B.

Before we can define the other operators, we need to introduce some more notation. Consider the whisker set $\nu(e)$ of an edge e on $\mathcal{S}^2$. The vertex whisker set $\nu(e\,\mathrm{Org})$ lies in one of the two hemispheres delimited by the great circle that contains the arc. We denote by $\mathrm{ToOrg}(e)$ the vector orthogonal to this great circle and pointing towards the hemisphere in which $\nu(e\,\mathrm{Org})$ lies (Figure 3.8). If the signs of e and $e\,\mathrm{Org}$ are positive, then $\mathrm{ToOrg}(e)$ is $e\,\mathrm{Org} - e\,\mathrm{Dest}$. Flipping the sign of e results in negating $\mathrm{ToOrg}(e)$, since it changes the sense of the vector $e\,\mathrm{Org} - e\,\mathrm{Dest}$ with respect to $\nu(e)$. Flipping the sign of $e\,\mathrm{Org}$ changes the Onext order of edges around $\nu(e\,\mathrm{Org})$ from being clockwise to counter-clockwise (or vice versa), thus resulting in negating $\mathrm{ToOrg}(e)$. Putting these together, we get:

$$\mathrm{ToOrg}(e) = \sigma_{e\,\mathrm{Org}}\,\sigma_e(e\,\mathrm{Org} - e\,\mathrm{Dest}) \tag{3.5}$$

In order to express the connectivity of a face in the convolution resulting from a pair of edges (e, e'), we need to know the "sense" in which their whisker sets $\nu(e)$ and $\nu(e')$ intersect. For this, we need a function for the tangent to $\nu(e)$ (arc going from $\nu(e\,\mathrm{Lface})$ to $\nu(e\,\mathrm{Rface})$) at the intersection between $\nu(e)$ and $\nu(e')$. For an edge e and a normal μ that lies on $\nu(e)$, we define $\zeta(e, \mu)$ to be a vector tangent to $\nu(e)$ at μ, i.e.

$$\zeta(e, \mu) = \sigma_e(e\,\mathrm{Dest} - e\,\mathrm{Org}) \times \mu, \tag{3.6}$$

where $\times$ denotes the vector cross product.

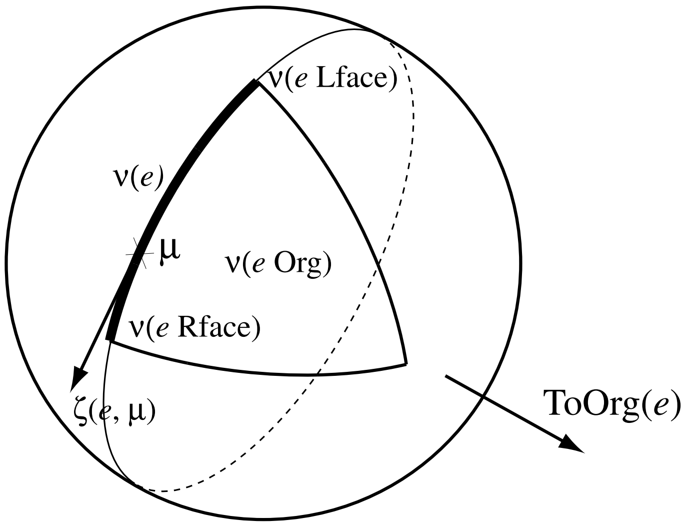

Figure 3.8: The vector $\zeta(e,\mu)$ and ToOrg(e) on $\mathcal{S}^2$.

The whisker sets of an edge e and a vertex v can intersect on S^2 in several ways (Figure 3.9):

1. $\nu\,(e)$ can lie entirely inside the triangle $\nu\,(v)$,
2. $\nu\,(e)$ can start inside, at $\nu\,(e\,\mathrm{Lface})$, and then exit,
3. $\nu\,(e)$ can start outside $\nu\,(v)$ and enter and then stop inside, and
4. it can start outside $\nu\,(v)$, pass through, and end outside again.

In cases 3 and 4, to define $(e,v)\,\mathrm{Lface}$ we need an operator that identifies the "first" intersection of $\nu\,(e)$ with $\nu\,(v)$ on S^2. The operator is called $\mathrm{Fedge}\,(e,v)$ (Figure 3.9): the *first edge* bounding $\nu\,(v)$ that one meets when going from $\nu\,(e\,\mathrm{Lface})$ to $\nu\,(e\,\mathrm{Rface})$. In other words, we require $\mathrm{Fedge}\,(e,v)$ to be an edge e' whose origin is v and such that at the intersection of $\nu\,(e)$ and $\nu\,(e')$, the arc from $\nu\,(e\,\mathrm{Lface})$ to $\nu\,(e\,\mathrm{Rface})$ points towards $\nu\,(v) = \nu\,(e'\,\mathrm{Org})$. Formal definition of $\mathrm{Fedge}\,(e,v)$ follows.

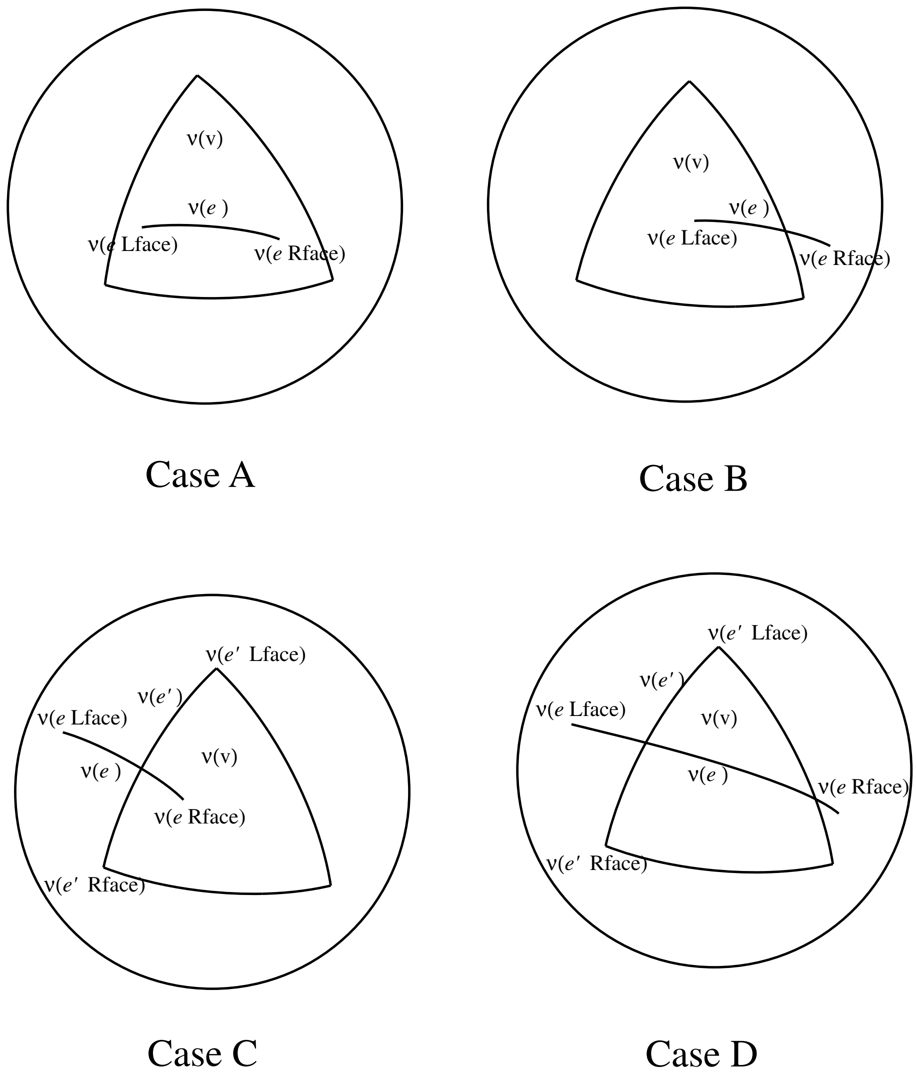

Figure 3.9: Four ways that the whisker sets of an edge e and a vertex v can intersect on S^2; in cases (c) and (d), the whisker set of $e' = \text{Fedge}\,(e, v)$ is shown assuming that σ_v is positive.

Definition 3.2. *Consider an edge $e \in E_b$ and a vertex $v \in V_r$ such that their whisker sets intersect, but $\nu\,(e\,\mathrm{Lface}) \notin \nu\,(v)$. We define* $\mathrm{Fedge}\,(e, v)$ *to be the unique edge $e' \in E_r$ such that:*

$$e'\,\mathrm{Org} = v \tag{3.7}$$

$$\zeta(e, \mu) \cdot \mathrm{ToOrg}(e') > 0 \tag{3.8}$$

where μ is the only point of $\mathcal{S}^2$ in the intersection of $\nu\,(e)$ and $\nu\,(e')$.

Relation (3.8) ensures that the tangent to $\nu\,(e)$ going from $\nu\,(e\,\mathrm{Lface})$ to $\nu\,(e\,\mathrm{Rface})$ at its intersection with e' points towards $\nu\,(v)$ (that is, $\nu\,(e'\,\mathrm{Org})$).

The method to compute $\mathrm{Fedge}\,(e, v)$ is only briefly addressed here. It will be an output of the sweep-line algorithm described in the next section. The rules for constructing the polyhedral tracing representing the convolution are summarized in Tables 3.1 and 3.2.

A word of caution: the definitions above depend on the signs of the features involved in a way that may not be particularly intuitive. The rules outlined in table 3.1 were found by trial and error. The other primitives ($\mathrm{Dest}, \mathrm{Rface}$, etc $\ldots$) can be deduced from the four fundamental primitives listed in the table. The rest of this section consists of detailed calculations that prove the consistency of the convolution.

Note that, just as in the two-dimensional case, the topological adjacencies between features in the convolution are induced by those of the factor tracings and the signs of its features are the products of the corresponding signs of the matching factor features. Similarly, we will show that the structure defined above is a consistent half-edge structure, in the sense that all the relations (3.1–3.3) are satisfied, and that its geometric information is that of a tracing. Except for the case of closed convex polyhedra, where it is the well-known Minkowski sum, the resulting convolution will in general self-intersect and may have locally open faces, as in Figure 3.6.

$$(e, v)\,\mathrm{Sym} = (e\,\mathrm{Sym}, v) \tag{3.9}$$
$$(e, v)\,\mathrm{Org} = ((\sigma_v\, e)\,\mathrm{Org}, v) \tag{3.10}$$

If $\nu\,(e\,\mathrm{Lface}) \in \nu\,(v)$:

$$(e, v)\,\mathrm{Lface} = (e\,\mathrm{Lface}, v) \tag{3.11}$$
$$(e, v)\,\mathrm{Lnext} = (e\,\mathrm{Lnext}^{\sigma_v}, v) \tag{3.12}$$

If $\nu\,(e\,\mathrm{Lface}) \notin \nu\,(v)$:

$$(e, v)\,\mathrm{Lface} = (e, \mathrm{Fedge}\,(e, v)) \tag{3.13}$$
$$(e, v)\,\mathrm{Lnext} = (w, \sigma_w\,\mathrm{Fedge}\,(e, v)) \text{ with } w = (-\,\sigma_v\, e)\,\mathrm{Org} \tag{3.14}$$

Table 3.1: Connectivity of the convolution of two polyhedral tracings.

$$\begin{aligned}
\mathrm{p}\,(v_b, v_r) &= \mathrm{p}\,(v_b) + \mathrm{p}\,(v_r) \\
\nu\,(v_b, v_r) &= \nu\,(v_b) \cap \nu\,(v_r) \\
\nu\,(e, v) &= \nu\,(e) \cap \nu\,(v) \\
\nu\,(f, v) &= \nu\,(f) \\
\nu\,(e_b, e_r) &= \nu\,(e_b) \cap \nu\,(e_r) \\
\sigma_{(f,v)} &= \sigma_v\,\sigma_f \\
\sigma_{(e_b,e_r)} &= \sigma_{e_b}\,\sigma_{e_r}
\end{aligned}$$

Table 3.2: Geometry of the convolution of two polyhedral tracings.

We describe later how to compute Fedge (e, v) operator systematically by overlaying the dual subdivisions of the two polyhedra on the sphere of directions. In the next chapter we show how to compute these operators using a much more efficient algorithm.

3.3.3 The convolution is a consistent tracing

Recall that we want to prove relations (3.1–3.3) for the connectivity defined in Table 3.1. There are essentially two cases: one for the ring of edges around a face resulting from convolving a face and a vertex (Lemma 3.3), and one for the ring of four edges around the face obtained by convolving two edges (Lemma 3.6).

Lemma 3.3. *For an edge $e \in E_r$ and a vertex $v \in V_b$ such that $\nu\,(e\,\mathrm{Lface}) \in \nu\,(v)$ and the definitions of Table 3.1, the fundamental half-edge relations (3.1–3.3) are satisfied.*

Proof. Examining the relations one by one, and using the fact that they hold for the original tracings, we have:

$$\begin{aligned}
(e, v)\,\mathrm{Sym}\,\mathrm{Sym} &= (e\,\mathrm{Sym}, v)\,\mathrm{Sym} && \text{Definition (3.9)}\\
&= (e\,\mathrm{Sym}\,\mathrm{Sym}, v) && \text{Definition (3.9)}\\
&= (e, v) && \text{(3.1) for } R.
\end{aligned}$$

$$\begin{aligned}
(e, v)\,\mathrm{Lnext}\,\mathrm{Lface} &= (e\,\mathrm{Lnext}^{\sigma_v}, v)\,\mathrm{Lface} && \text{Definition (3.12)}\\
&= (e\,\mathrm{Lnext}^{\sigma_v}\,\mathrm{Lface}, v) && \text{Definition (3.11)}\\
&= (e\ \ \mathrm{Lface}, v) && \text{(3.2) for } R\\
&= (e, v)\ \ \mathrm{Lface} && \text{Definition (3.11).}
\end{aligned}$$

$$
\begin{aligned}
(e,v)\,\mathrm{Lnext}\,\mathrm{Org} &= (e\,\mathrm{Lnext}^{\sigma_v}, v)\,\mathrm{Org} && \text{Definition (3.12)}\\
&= ((\sigma_v(e\,\mathrm{Lnext}^{\sigma_v}))\,\mathrm{Org}, v) && \text{Definition (3.10)}\\
&= ((-\,\sigma_v\,e)\,\mathrm{Org}, v) && \text{Relation (3.4)}\\
&= (-e, v)\,\mathrm{Org} && \text{Definition (3.10)}\\
&= (e,v)\,\mathrm{Sym}\,\mathrm{Org} && \text{Definition (3.9).}
\end{aligned}
$$

□

This proves the three fundamental half-edge relations for the convolution in the "easy case", i.e. for the ring of edges around a vertex-face convolution feature. The edge-edge case is more difficult, and we need some intermediary computations.

Lemma 3.4. *Let e, e' be two edges whose state sets intersect at μ. Let $e\,\mathrm{Org} = w, e'\,\mathrm{Org} = v$. Then:*

$$\sigma_v\,\zeta(e,\mu)\cdot\mathrm{ToOrg}(e') = -\,\sigma_w\,\zeta(e',\mu)\cdot\mathrm{ToOrg}(e).$$

Proof. The negative sign comes from the antisymmetry of the determinant. For any edge e, let $\delta(e)$ denote $\sigma_e(e\,\mathrm{Dest} - e\,\mathrm{Org})$. Using this abbreviation, we can rewrite Equation (3.5) as

$$\mathrm{ToOrg}(e) = -\,\sigma_{e\,\mathrm{Org}}\,\delta(e),$$

and Equation (3.6) as

$$\zeta(e,\mu) = \delta(e)\times\mu.$$

We have:

$$\begin{aligned}
\zeta(e,\mu)\cdot \mathrm{ToOrg}(e') &= (\delta(e)\times\mu)\cdot(-\sigma_v\,\delta(e')) \\
\zeta(e',\mu)\cdot \mathrm{ToOrg}(e) &= (\delta(e')\times\mu)\cdot(-\sigma_w\,\delta(e)) \\
(x\times y)\cdot z &= -(z\times y)\cdot x.
\end{aligned}$$

□

Lemma 3.5. *Let (e,v) be an edge-vertex pair such that $\nu\,(e\,\mathrm{Lface}) \notin \nu\,(v)$. If $e' = \mathrm{Fedge}\,(e,v)$ and $w = (-\,\sigma_v\,e)\,\mathrm{Org}$, then*

$$\nu\left((\sigma_w\,e')\right)\cap\nu\,(w)\neq\emptyset \tag{3.15}$$

$$\nu\left((\sigma_w\,e')\,\mathrm{Lface}\right) \notin \nu\,(w) \tag{3.16}$$

$$\mathrm{Fedge}\left(\sigma_w\,e',w\right) = -\,\sigma_v\,e \tag{3.17}$$

Proof. Let μ be the whisker common to e and e'. It is also a whisker common to $(\sigma_w\,e')$ and w, thus proving (3.15). To prove (3.16), we can show that the tangent at μ to the arc from $\nu\,((\sigma_w\,e')\,\mathrm{Lface})$ to $\nu\,((\sigma_w\,e')\,\mathrm{Rface})$ points towards w (this is sufficient as we assumed that all arcs were shorter than π). In other words, we show that $\zeta(\sigma_w\,e',\mu)$ is oriented in the same direction as $\mathrm{ToOrg}(-\,\sigma_v\,e)$. Taking the inner product:

$$\begin{aligned}
\zeta(\sigma_w\,e',\mu)\cdot\mathrm{ToOrg}(-\,\sigma_v\,e) &= \sigma_w\,\zeta(e',\mu)\cdot\mathrm{ToOrg}(-\,\sigma_v\,e) \\
&= -\,\sigma_v\,\zeta(-\,\sigma_v\,e,\mu)\cdot\mathrm{ToOrg}(e') && \text{Lemma 3.4} \\
&= \zeta(e,\mu)\cdot\mathrm{ToOrg}(e') \\
&> 0 && (3.8)
\end{aligned}$$

This proves (3.16). Moreover, the above calculations show that $-\,\sigma_v\,e$ satisfies the requirements of Definition (3.2) for the pair $(w,\sigma_w\,e')$, which proves (3.17). □

We are now ready to prove the lemma for the edge ring around an edge/edge face of the convolution:

Lemma 3.6. *For an edge $e \in E_r$ and a vertex $v \in V_b$ such that $\nu(e) \cap \nu(v) \neq \emptyset$, $\nu(e\,\mathrm{Lface}) \notin \nu(v)$, and the definitions of Table 3.1, the fundamental half-edge relations (3.1–3.3) are satisfied.*

Proof. Once again, we prove the three fundamental relations independently. The proof of the property $(e,v)\,\mathrm{Sym}\,\mathrm{Sym} = (e,v)$ is just as before, so we omit it. As above, let $e' = \mathrm{Fedge}\,(e,v)$ and $w = (-\,\sigma_v\, e)\,\mathrm{Org}$.

$$\begin{array}{rcll}
(e,v)\,\mathrm{Lnext}\,\mathrm{Lface} & = & (\sigma_w\, e', w)\,\mathrm{Lface} & \text{Definition (3.14)} \\
& = & (\sigma_w\, e', \mathrm{Fedge}\,(\sigma_w\, e', w)) & \text{Definition (3.13) because (3.16)} \\
& = & (\sigma_w\, e', -\,\sigma_v\, e) & \text{(3.17)} \\
& = & (e, e') & \text{The four pairs represent the same face} \\
& = & (e,v)\,\mathrm{Lface} & \text{Definition (3.13),}
\end{array}$$

$$\begin{array}{rcll}
(e,v)\,\mathrm{Lnext}\,\mathrm{Org} & = & (\sigma_w\, e', w)\,\mathrm{Org} & \text{Definition (3.14)} \\
& = & (w, e'\,\mathrm{Org}) & \text{Definition (3.10)} \\
& = & ((-\,\sigma_v\, e)\,\mathrm{Org}, v) & \text{(3.7)} \\
& = & (-e, v)\,\mathrm{Org} & \text{Definition (3.10)} \\
& = & (e,v)\,\mathrm{Sym}\,\mathrm{Org} & \text{Definition (3.9).}
\end{array}$$

$\square$

We now show that the geometry of the tracing is valid. For this, we need to prove that the vertices of a face are planar. The vertices of faces of the form (v, f) are obviously planar since they are translated copies of f. This is also clear for faces (e_1, e_2) since these are parallelograms formed by these edges in the plane containing the lines supporting the edges.

3.3.4 Computing Fedge (e, v) primitive

Computing the convolution by the above method involves computing the Fedge (e, v) primitive for all possible pairs v, e. This can be achieved by overlaying the subdivision of the first tracing onto the subdivision of the second and computing all cross intersections. For each cross intersection e, e' between an edge of R with an edge of B, we set up the Fedge (e, v) of the vertex v that includes the intersection with e to be e'. This can be computed by traversing the overlays of R and B in unison. The total time of a naive overlay method is $O(mn \log(m+n))$ where m is the complexity of R and n is that of B. Of this complexity, $O(mn)$ is the cost of looking at each pair of features of R and B and the extra $O(\log(m+n))$ factor is devoted to sorting the features of the result Q. This computation is the most obvious implementation and is not optimally efficient. In particular one need not look at all pairs of features of R and B — one can do better by systematically *traversing* the boundaries of R and B. The bulk of the next chapter is devoted to computing these primitives efficiently without the time complexity required by the naive algorithm described here.

3.3.5 Normalization and convolution size

When discussing output-sensitive algorithms, there is an issue of what should be considered to be the size of the output. This issue was discussed at length by Guibas and Seidel [71] for two-dimensional convolutions, and it is not surprising that it arises again in three dimensions. In the present setup, two tracings that have different topologies are always considered distinct, although they might define the same painting (and the same "bag of states" in the vocabulary of the original kinetic framework). In this sense, the output described above is the only one that properly describes the exact topology of the fiber product manifold of the convolution.

Note however that the objects we are ultimately interested in are the underlying paintings, and not one specific representation as a tracing. As in the two-dimensional case, it is possible to construct two tracings of size m, n, such that their convolution is of size $\Theta(mn)$, where $\Theta(m+n)$ features would be sufficient to represent the same painting.

We now make a conjecture that proposes to relate the convolution of tracings to the painting convolution.

Conjecture 1. *Let B, R be two polyhedral tracings and Q their convolution. Then the painting that Q represents is the convolution of the two paintings that B and R represent.*

3.4 From convolution to Minkowski sum

In this section, we briefly address the issue of how one can go from the convolution of two polyhedral tracings to the Minkowski sum of the corresponding polyhedral objects. We return to this issue in the next chapter, where we will address it in greater detail.

Based on the fundamental theorem as described for convolutions of tracings in [70] and for convolutions of paintings in [121], we conclude that the winding number with respect to the convolution gives useful information with respect to the Minkowski sum of the inverted robot and the obstacle. Given two tracings their convolution corresponds to the painting convolution of the two corresponding paintings. In other words, converting the convolution of tracings into a painting using the winding number function results in the painting convolution.

In three dimensions, if the intersection of the robot and obstacle is always simply connected, then the winding number with respect to the convolution is non-zero exactly when the robot placed at the corresponding position intersects with the obstacle (the Minkowski sum is exactly the non-zero winding number region of the convolution). However, if the intersection of the robot and obstacle may not be simply connected, then the complement of the Minkowski sum can be computed as follows: a) find all the zero winding regions of the convolution, b) for each region, pick any sample point and check if the obstacle and the robot placed at that point intersect, and c) if they do not intersect, then add the region to the complement of the Minkowski sum. The resulting union of non-intersecting zero winding number regions is the complement of the Minkowski sum.

It is now possible to compute the Minkowski sum by performing a plane sweep over the convolution collecting non-intersecting zero-winding number regions of the convolution. At any position of the sweep plane we are interested in knowing the convolution regions on the sweep plane that are non-intersecting and have zero-winding number. This information can be updated as new events are processed in the plane sweep. By combining non-intersecting zero-winding number regions on the sweep plane we can construct the entire Minkowski sum: an outer shell possibly with holes and tunnels.

In the next chapter we will explain how the convolution is computed and that will give us the necessary background and opportunity to explain the Minkowski sum computation in greater detail. There is also ground for further work in this area to compute the Minkowski sum more efficiently than is shown in this thesis.

3.5 Conclusion

In this chapter we introduced the notion of polyhedral tracings. We defined the operation of convolution on them which is the three-dimensional extension of the same operation in two dimensions. We showed that tracings and their convolution in three dimensions have roughly the same properties as in two dimensions. A polyhedral tracing is like a polyhedron except for the extra advantage it has in representing locally open, locally closed as well as mixed polyhedral sets. Like a turn at a vertex in a two-dimensional tracing we add extra "whisker sets" at edges and vertices of polyhedral tracings. Whiskers are features on the surface of the two-dimensional unit sphere $\mathcal{S}^2$ (a turn can be thought of as representing a feature, namely an interval, on the boundary of a one-dimensional unit sphere $\mathcal{S}^1$, a disc). At edges these correspond to an edge on the sphere $\mathcal{S}^2$ going from one face normal of a face incident to the edge to the other face normal. At vertices the whisker sets correspond to a spherical polygon with edges corresponding to whiskers of the edges incident at the vertex. The whisker at a face is simply the face normal, oriented outward or inward based on whether the face is locally closed or open, respectively.

We show how to store a polyhedral tracing in a half-edge structure corresponding to the boundary of the polyhedral tracing. Extra information required is a boolean at each face specifying whether the face whisker is outward or inward — in other words, whether the tracing is locally open or closed at that face. We showed how the whisker sets of edges and vertices can be derived from the whiskers at faces. We call this extension of the half-edge structure an oriented half-edge structure (with extra information of orientation at each face).

We introduced the notion of convolution of two polyhedral tracings R and B. The convolution is obtained by pairing features h_R and h_B of the two tracings such that the whisker sets of h_R and h_B intersect. Under certain assumptions we show how to represent the convolution within a simple system of notation and prove that it satisfies some desirable properties. The properties of the convolution make it a consistent "balanced" tracing. The convolution is representable by an oriented half-edge structure in the same way as the original tracings that were convolved. We provide rules by which the half-edge structure of the convolution can be derived from the half-edge structures of the original tracings.

By pairing every feature of the first tracing versus the second tracing the convolution can thus be computed in $O(mn \log(m+n))$ time (mn is for testing every pair of features and the extra logarithmic factor is for sorting the convolution features into an oriented half-edge structure). In the next chapter we show how to compute the convolution much more efficiently.

Chapter 4

Computation of Polyhedral Convolution

In this chapter, we describe an algorithm to compute the signed half-edge structure of a convolution $Q = B * R$ of two tracings R and B with a total of n vertices in output-sensitive time $O(k\alpha(n)\log^3 n)$, where k is the number of vertices of Q. We will assume that R nd B have all faces of sign $+1$, from which it follows that their whisker maps cover the sphere of directions $\mathcal{S}^2$, and therefore $k \geq n$. The key algorithmic problem we must solve is to find the matching features of R and B, by computing their overlay on $\mathcal{S}^2$.

In the case where R and B are convex tracings, i.e. when each whisker value appears only once per tracing, this problem has already been optimally solved [71] in optimal $O(n + k)$ time. The difficulty with general tracings is that their whisker maps can multiply cover $\mathcal{S}^2$ — they are effectively Riemann surfaces over $\mathcal{S}^2$. During the overlay operation we want to discover all bichromatic edge intersections, but we cannot afford to pay for looking at crossings of edges of the same color, which would arise if we were to 'flatten out' each Riemann surface onto $\mathcal{S}^2$ (Figure 4.1).

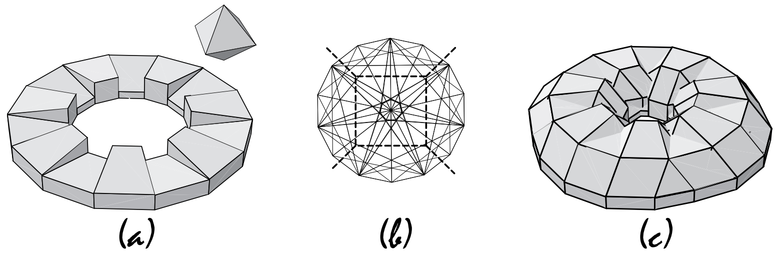

Figure 4.1: (a) A diamond and a synchronizing gear with $n = 7$ teeth; (b) the arcs defined by the edges of the gear have $\Theta(n^2)$ intersections on the sphere (north hemisphere shown); (c) but the convolution has only $\Theta(n)$ features.

In this chapter, we give a brief history of intersection problems in Section 1. From the complexity viewpoint, the dominant cost of an algorithm to compute the convolution is the computation of all the bichromatic edge intersections; we describe a solution for this sub-problem in Section 2, and analyze its running time in Section 3. At last, we show how it applies to the convolution computation in the Section 4, and conclude in Section 5.

4.1 Intersection problems

The problem of finding and reporting all pair-wise intersections in a set of line segments is among the first to have been studied in computational geometry; its solution established the use of sweep-line methods and introduced the notion of output-sensitive algorithms. A common variation is the *red-blue intersection problem*, where two families of line segments are given – a red and a blue – and only the bichromatic (*purple*) intersections are to be reported. Various assumptions can be made about each family. For our purposes, a family of line segments will be called *disjoint* if no pair of segments intersect (except possibly at their endpoints), *connected* if the union of all segments is connected as a point set, and *general* otherwise. Each assumption leads to different algorithms and running times. In what follows, we denote by n the number of (red and blue) input segments, and by k the number of purple intersections.

The case where each input set is disjoint has been extensively studied. In the early days of computational geometry, Bentley and Ottmann [25] introduced their well-known line sweep algorithm, which reports all purple intersections in time $O((n+k)\log n)$. Mairson and Stolfi [99] were the first to find an asymptotically optimal algorithm running in $O(n\log n+k)$ time. This second algorithm searches for and reports purple intersections at endpoint events during the sweep; the use of a clever 'red-blue cone' invariant allowed the authors to get rid of the $O(\log n)$ overhead on the number k of intersections. The same optimal time bound for reporting purple intersections was achieved later by several different methods [34, 36, 43, 113]. Especially worthy of mention among these is the hereditary segment tree data structure of Chazelle, Edelsbrunner, Guibas, and Sharir [36], which can also be used to *count* all purple intersections in $O(n\log n)$ time. Thus, in the disjoint case, asymptotically optimal and practical algorithms are known.

For general inputs, where arbitrary intersections are allowed within each family, a Bentley-Ottmann sweep reports all purple intersections in $O((n+k+k')\log n)$ time, where k' is the total number of monochromatic intersections. For certain inputs k' can be $\Omega(n^2)$, while the desired output can be much smaller. This version of the problem seems to be of about the same difficulty as counting all segment intersections in a given family. For this latter problem, Guibas, Overmars, and Sharir [69] gave a randomized $O(n^{4/3+\epsilon})$ solution using partitioning techniques (what has since become known as cuttings; derandomization tools developed later can also be applied). Agarwal [2] adapted this technique to the red-blue version of the problem, lowering the running time to $O(n^{4/3}\log n+k)$ for reporting and obtained a method for counting in time $O(n^{4/3}\log n)$ as well, at the cost of increasing the space requirement to $\Theta(n^{4/3})$. A significant improvement in these time bounds is unlikely, as the general case falls within a class of problems at least as hard as Hopcroft's problem of detecting an incidence between a set of n points and a set of n lines; the fastest known algorithm for Hopcroft's problem is due to Matoušek [49, 102], and runs in $O(n^{4/3}2^{O(\log^* n)})$

time. For a certain general class of algorithms, a lower bound of $\Omega(n^{4/3})$ for this problem was recently given by Erickson [59].

Given this situation, we were motivated to seek additional conditions which make easier the task of reporting the purple intersections. A natural condition is that of *connectedness* for each of the monochromatic inputs, as introduced above. This condition often pertains in situations where the purple intersection problem itself arises (e.g., in the overlay of line arrangements, or of simply-connected planar subdivisions – though in the latter case the disjoint methods above apply). Under the hypothesis of connectedness, Agarwal and Sharir [6] looked at the problem of detecting a *single* purple intersection. Their technique is based on the following idea: pick a point z on the red collection and compute the blue face $\mathcal{F}$ that contains z using a general algorithm to compute a single face in an arrangement of line segments. Next, pick a blue point on the boundary of $\mathcal{F}$ and compute the red face $\mathcal{F}'$ that contains it. If the blue and red segment sets are connected, the set of purple intersections is non-empty iff the boundaries of $\mathcal{F}$ and $\mathcal{F}'$ intersect, and this can be tested via a Bentley-Ottmann sweep. Using the recent randomized algorithm of Chazelle *et al.* [37] to compute a single-face, a purple intersection is detected in time $O(n\alpha(n)\log n)$, where $\alpha(n)$ is the slowly growing inverse of Ackermann's function.

Some of the algorithms described above are essentially combinatorial [25, 99, 6, 43], and work as well if the line segments are replaced by x-monotone algebraic arcs of bounded degree. Others [34, 37, 2] make a more essential use of the affine structure of the input and cannot be so adapted. When considering x-monotone arcs such that any pair intersects at most s times, it is common to encounter the function $\lambda_s(n)$, which denotes the almost-linear maximum length of an (n, s) Davenport-Schinzel sequence; for more material on Davenport-Schinzel sequences the reader is referred to [129]. Our contribution is HeapSweep: an efficient algorithm to compute the intersections between two sets of line segments both of which are connected as a point set on the plane.

4.2 The HeapSweep algorithm

We assume that the input is non-degenerate, i.e. that all end segment points and intersections occur at distinct x values. Standard perturbation techniques can be used to guarantee that this is always the case [56].

In the traditional Bentley-Ottmann line sweep, a balanced search tree is used to represent the state of the sweep line. This search tree stores the exact top-to-bottom ordering of the line segments intersecting the sweep line. In order to maintain this ordering as the sweep progresses, *all* intersections between line segments, namely red-red, blue-blue, and purple intersections have to be detected and processed. Our algorithm follows the general principle of the line sweep technique, with a global event queue $\mathcal{Q}$ for segment endpoints and intersections (which are scheduled as they are discovered), but it stores only a partial ordering of the segments on the sweep line.

We divide the set of line segments intersecting a given position of the sweep line into a sequence of contiguous monochromatic blocks $\Gamma_1, \Gamma_2, \dots, \Gamma_t$ (Γ_1 is the top block), and we keep the ordering of the blocks but not the exact order of segments within each block. In order to preserve this structure and to detect purple intersections as the sweep progresses, we need to look for possible intersections between the top and bottom segments of adjacent blocks along the sweep line. For this purpose, each block Γ_i is stored in a new data structure, called a *kinetic queue*, which is an enhancement of a standard priority queue (with efficient insertion, deletion, and access to the top priority element) allowing for data with continuously changing priorities. In our case, the priority of a segment is the vertical coordinate of its intersection with the sweep line; actually, two kinetic queues with opposite priorities are maintained for each block, allowing efficient access to the smallest and largest element within each block.

A kinetic queue can be implemented by a standard priority queue data structure, such as a binary heap. What is novel in the kinetic setting is that the priorities of the elements in the heap continuously change, as the sweep proceeds. Whenever the sweep-line encounters a monochromatic intersection between a parent-child pair in this heap, the heap structure needs to be updated – but fortunately this is relatively straightforward. Furthermore, these updates can be scheduled using the same mechanism used by the sweep as a whole, namely by the maintenance of a global event queue $\mathcal{Q}$ ordered by time. We will call such monochromatic intersection events processed by the algorithm *internal* to the corresponding kinetic queue. What makes the implementation of a kinetic queue challenging is the need to minimize the number of internal events processed. Kinetic data structures have received much attention recently in literature [20, 18, 19].

In order to process the events corresponding to the start of a new segment or the end of an old segment during the sweep, we will store the top and bottom elements of each block in a balanced binary search tree $\mathcal{T}$, according to the ordering of the blocks along the sweep line. Note also that the purple intersections of interest will always be between the bottom segment in one block and the top segment in the following block in the ordering. Thus each pair of consecutive bichromatic elements along $\mathcal{T}$ schedules a possible future purple intersection event, which is re-scheduled each time the top or bottom segment of one of these blocks changes (due to an internal event in one of the associated kinetic queues). Purple intersection events themselves typically create two new blocks of size one (Figure 4.2–a), which necessitates the modification and re-balancing of $\mathcal{T}$.

4.2.1 Endpoint events

In a traditional sweep line algorithm, segment endpoints are processed easily by locating their position along the sweep line, and inserting or deleting the corresponding segment at that position. The situation is more delicate when it comes to HeapSweep.

When an endpoint starting a blue segment is encountered during the sweep, the block in which it is located can be found using $\mathcal{T}$. If this block is blue, the segment can be inserted directly into the corresponding kinetic queue(s); if the block is red, however, the blue segment *splits* the red block into two pieces. Similarly, when a segment ends, it may cause a block to vanish and require a *merge* of two existing blocks of the same color (Figure 4.2–b). Both block splits and merges can require time proportional to the size of the blocks on which they are performed. In a standard fashion, this cost can be reduced to a number of kinetic queue insertion or deletion operations *proportional to the size of the smaller block involved.* A merge is implemented by inserting each segment from the smaller of the two blocks into the larger one. A split is implemented by alternately deleting segments from the top and the bottom of the block until we reach the position where the new segment has to be inserted. If this position is reached from the top first, say, then the segments deleted from the bottom are inserted back into the block, while those deleted from the top are inserted into a new (and initially empty) block.

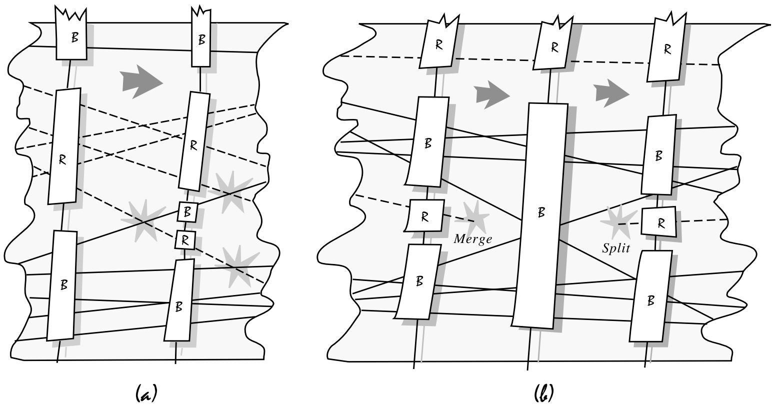

Figure 4.2: (a) A purple intersection: two new blocks are created, and new purple intersection events are scheduled. (b) A merge of two blue blocks at a red right endpoint followed by a split of a blue block caused by a red left endpoint.

In summary, the number of kinetic queue operations required to implement a split and a merge is proportional to the number of segments present in the smaller of the two blocks involved. This cost is still high in the worst case but, as we will see in Section 3, the *connectedness* of the input guarantees that the worst case doesn't occur too often. Without the connectedness hypothesis (even in the disjoint case) it is straightforward to construct an example with no purple intersections where HeapSweep would run in quadratic time.

4.2.2 Kinetic queue implementations

As was mentioned above, a natural way to implement a kinetic queue is as a standard binary heap. However, no satisfactory bounds are known on the number of internal events needed to maintain such a structure during the sweep [22]. Instead, we describe two different implementations of kinetic queues [21, 20], in which the number of internal events is within a log factor of the total number of objects present in the queue. We do not prove the correctness and running time of the kinetic queue implementations in this thesis; instead, we refer the reader to [21, 20].

The first implementation, called a *heater*, is randomized. The second, called a *kinetic tournament*, is deterministic.

To define a heater, we proceed as follows. If each object in a set is given two numbers, a *rank* and a *priority*, there is a unique binary tree which is both a search tree on the ranks and a heap on the priorities. Such a tree is well-known and called a *treap* – Aragon and Seidel [10] used it to create their popular randomized search tree data structure, which is a treap on objects with a given rank and a randomly assigned priority. The randomization guarantees that this structure is balanced with high probability. A heater is like a treap, but this time, priorities are given and ranks are random. When an element with a given priority is inserted in a heater, it is first assigned a random rank, and inserted at the appropriate leaf of the heater. It then bubbles up with a sequence of rotations until it reaches a position consistent with its priority. Deletions are implemented analogously.

When a heater is used to implement a kinetic queue in HeapSweep, we need to detect and process certain internal events in order to keep the heater consistent with the continuously changing priorities (Figure 4.3). These internal events are the intersections between parent-child segment pairs in the heater. When such an event occurs, a rotation involving the parent and child is performed to keep the heater consistent – this is sufficient, as at that moment the parent and child have equal priorities and no other ordering in the heater can be changing at the same time (by our non-degeneracy assumption). After the rotation, two parent-child relationships change in the heater. Thus up to two existing events in the sweep event queue $\mathcal{Q}$ may have to be de-scheduled (deleted), and two new events scheduled (inserted).

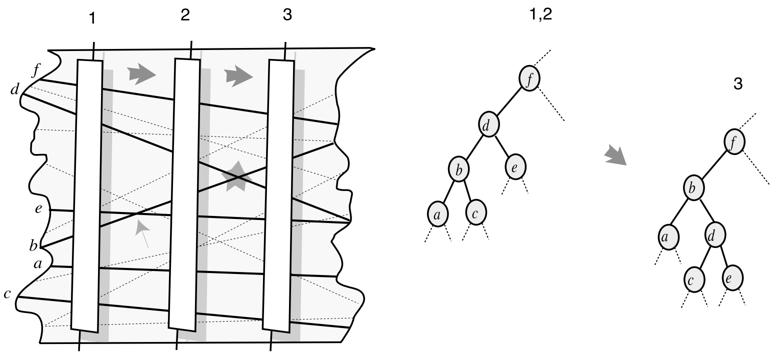

Figure 4.3: Three positions of the sweep line and the corresponding heaters with attention focused on line segments labeled (randomly) a through f. As the sweep line moves from position 1 to 2 passing the intersection between b and e, the heater remains unchanged (since b and e are not a parent-child pair). However, when it moves from position 2 to 3 passing the intersection between parent-child pair b and d, a rotation is performed that swaps the priorities of b and d but otherwise preserves the inorder sequence.

Lemma 4.1. *[21] Let H be a heater sweeping over s segments in the plane. Denote by $C(H)$ the number of intersections that cause an update of H. Then $E[C(H)] = O(s\alpha(s)\log s)$. The expected cost of the sweep is $O(t_Q s\alpha(s)\log s)$.*

A kinetic tournament is another implementation of a kinetic priority queue based on the static tournament tree structure for leader election. It is built on a perfectly balanced tree which represents a tournament among all elements which ever appear in the kinetic queue. Here also, to keep the structure consistent with the changing priorities during the sweep, an internal event has to be scheduled in $\mathcal{Q}$ for each game of the tournament whose outcome can change in the future. When such an event happens, the new winner has to be percolated up the tree. Similar percolations need to happen when a new object appears or an old one disappears. Thus, unlike the heater case, tournament events in $\mathcal{Q}$ can cause a number of de-schedulings and re-schedulings proportional to the height of the tournament tree.

The tournament tree can be made dynamic as follows. Imagine a very large sequence of leaves and a perfectly balanced prototypical tournament tree built upon that sequence. At any one time, the actual elements present in the tournament tree occupy a portion near the left-most leaf of that prototypical tree. Such a leaf is determined by the subtree rooted at the node where the current leader is determined. Insertions are processed by adding new leaves to the right of the current set of leaves, creating, on occasion, a new root to accommodate more elements. In order to hold the structure in an amount of space linear in the number of nodes present, upon deletion of a leaf element, the lowest node representing a game with that element absorbs and removes its other (non-deleted) child (this works just like the 'easy case' in binary tree node deletion – when the node being removed can just be short-circuited). Both insertions and deletions must then percolate up the tournament tree.

4.3 HeapSweep analysis

In this section, we obtain expected and worst-case time and space bounds for the algorithm HeapSweep in the connected case of the red-blue intersection reporting problem. We first examine the expected time complexity of the sweep of a set of segments by one heater. Then we perform a similar analysis for a tournament tree. Finally, we show how economical polygon regularization [21] can be used to chop all segments in pieces so as to reduce the analysis of the HeapSweep algorithm to that of a set of independent kinetic queues. For clarity, the worst case update time of the main event queue $\mathcal{Q}$ is denoted by t_Q (it is clear that this quantity is $O(\log n)$).

4.3.1 Time bounds for kinetic queues

Analysis of two Kinetic Queue implementations has been done in [21]. The main results is as below:

Lemma 4.2. *[20] Let H be a kinetic tournament sweeping over s segments in the plane. The worst-case cost of the sweep is $O(t_Q s\alpha(s)\log s)$.*

4.3.2 Economical polygon regularization

Analyzing the complexity of Merge and Split operations requires some background. We define a decomposition of a polygon in the presence of line segments that has the following property: the number of fragments caused by the decomposition is a) relatively small, and b) is closely related to the cost of corresponding Merge and Split operations.

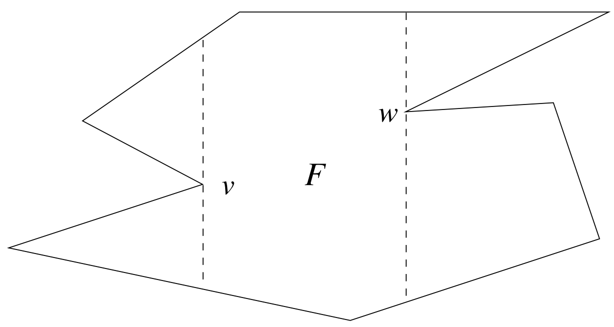

Figure 4.4: A simple polygon F regularized into x-monotone simply connected regions using vertical threads drawn from critical vertices v and w.

We call regularization the process of decomposing a polygon (or more generally, a subdivision) in the plane into x-monotone simply connected regions (Figure 4.4). In this section we look at the following scenario: we are given a simple polygonal face F (in an arrangement) containing a set R of (possibly intersecting) line segments. We show that F can be regularized via a number of vertical threads which cause only $O(|R| \log |R|)$ cuts of these segments. In the following subsections the cost of Merge and Split operations of the HEAPSWEEP algorithm will be shown to depend on these cuts.

A point on the boundary of the polygon F is called *critical* if it is locally x-extremal and concave (Figure 4.4). In other words, a critical point is a concave vertex of F where both incident edges are to the left or to the right of the vertex.

Definition 4.3. (Economical regularization) *The* R-economical regularization *of* F *is the decomposition of* F *into* x*-monotone regions obtained by the following procedure (Figure 4.4): from each critical vertex* v *of* F*, draw two threads vertically, up and down from* v*, until they reach the polygon boundary. Keep only the thread that intersects fewer elements of* R*.*

A crucial aspect here is that we only keep the 'shorter' of the two threads at each critical vertex. It is easy to see that economical regularization defines a monotone decomposition of P. We define the cost of the regularization to be the total number of intersections ('cuts') between segments of R and the selected threads. To analyze this cost, we use a lemma reminiscent of the classical analysis of the UNION-FIND [9] algorithm:

Lemma 4.4. *Let* $\mathcal{T}$ *be a tree and denote the left (resp. right) child of a node* ν *by* $\ell(\nu)$ *(resp.* $r(\nu)$*). Each node* ν *is given a (possibly negative) integer* weight w_ν*. The* subtree weight W_ν *of a node* ν *is the weight of its subtree:* $W_\nu = w_\nu + W_{\ell(\nu)} + W_{r(\nu)}$*. Define the* cost *of a node* ν *as the subtree weight of the lighter of its two subtrees, i.e.* $c_\nu = \min(W_{\ell(\nu)}, W_{r(\nu)})$, *and the* absolute weight *of $\mathcal{T}$ to be $S_{\mathcal{T}} = \sum_\nu |w_\nu|$. Then, if at least one node of $\mathcal{T}$ has a non-zero weight, we have:*

$$\sum_\nu |c_\nu| \leq S_{\mathcal{T}} \log S_{\mathcal{T}}.$$

Proof. By induction on the structure of the tree. □

4.3.3 Reachability tree

We proceed to define a tree decomposition of the simple polygon F, on which we later use the above lemma. In order to do so, critical vertices of F need to be separated into two different groups: the *split vertices* of F, which are locally x-minimal, and the *merge vertices*, which are locally x-maximal. We now focus only on the split vertices, and later invoke the same technique for the merge vertices by reversing the x-axis. By non-degeneracy, we may assume that no two critical vertices of F have the same x-coordinate.

Definition 4.5. (Up/down reachability) *Let F be a simple polygon, and v be a split vertex of F. A point $p \in F$ to the right of v is* up (resp. down) reachable *from v if there exists a path from v to p inside F which starts locally above (resp. below) v and remains to the right of v.*

Definition 4.6. (Up/down child, up/down region) *A point p, $p \in F$ is an* up (resp. down) child *of split vertex v if v is the rightmost split vertex from which p is reachable, and p is up (resp. down) reachable from v. The set of all up (resp. down) children of v is called the* up-region *(resp.* down-region*) of v and is denoted v^* (resp. v_*). The set r of points with no parent is called the* orphan region.

Since F is simple, no point can be simultaneously up and a down reachable from the same vertex, so that the regions defined above partition F. Each region is delimited by portions of the boundary of F and by portions of vertical lines passing through split vertices. Note that (i) a region need not be connected, and (ii) an up-child is not necessarily 'above' its parent in the plane (Figure 4.5–a).

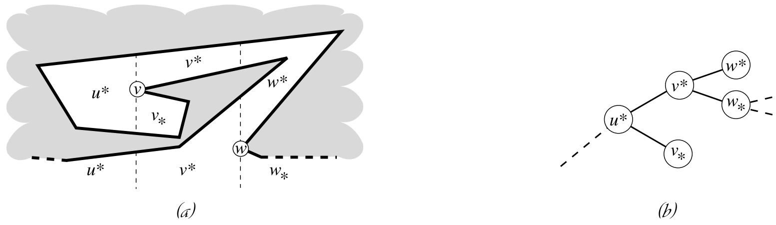

Figure 4.5: (a) A portion of a simply connected polygonal region (in white), and the regions associated with the split vertices (vertex u is not shown); (b) the associated portion of the reachability tree. Note that the merge vertices are ignored in this decomposition.

Lemma 4.7. *A region contains at most one split vertex on its right boundary.*

Proof. By contradiction. □

We now build a connectivity graph between these regions, with one node per region; two nodes are connected in this graph if they have a vertical boundary in common. By Lemma 4.7, this graph is a binary tree called the *reachability tree* of F, which we root at the orphan region. If the right boundary of a region contains a split vertex v, the associated node in the tree bears the up and down regions of v as its two children (Figure 4.5–b).

4.3.4 Cost of economical regularization

Lemma 4.8. *Let F be a simple polygon in the plane that contains a set R of (possibly intersecting) segments. The R-economical regularization of F cuts the segments of R a total of $O(|R|\log|R|)$ times.*

Proof. We consider only the cuts made by the split vertices of F, and prove that there are $O(|R|\log|R|)$ of them. The same argument holds for the merge vertices. We use below the vocabulary of Lemma 4.4.

Construct the reachability tree $\mathcal{T}$ of F. If a node ν of $\mathcal{T}$ has corresponding region ρ, set the weight w_ν of ν to be the number of segments that end in ρ minus the number of segments that start in ρ (a segment "starts" at its left endpoint and "ends" at its right one). Therefore, the subtree weight W_ν of ν is simply the number of segments crossing into ρ along its left boundary. If the region ρ terminates on the right with a split vertex v, the cost of ν is either the number of segments entering the up-region of v, or the number of segments entering the down-region of v, whichever is smaller.

The total absolute weight $S_{\mathcal{T}}$ of $\mathcal{T}$ is at most the number of end-points of all segments. It follows from Lemma 4.4 that the total cost of $\mathcal{T}$ is $O(|R| \log |R|)$.

For the R-economical decomposition of F, we choose to make a cut at each split vertex v with the thread (going up or down) that cuts the fewer segments till we reach the boundary of F again. Such a thread is a subset of the left boundary of either the up-region or the down-region of v. Therefore, it doesn't cut more segments than the cost (in the above sense) of the region to which v belongs. By proceeding like this we may choose to cut in the 'wrong' direction, but we will reach the boundary of F and stop cutting before we have spent more than we can afford. □

It is not hard to give an example showing that the bound in this lemma is tight to within a constant factor (a 'ruler function' polygon with one red segment per tick is such an instance – the two threads from each vertex cut the same number of red segments). Our purpose in defining the regularization is to bound the total cost of merge and splits in HeapSweep. This cost is captured by the following corollary:

Corollary 4.9. *Let B be a connected set of blue segments on the plane, and R be a set of red segments that don't intersect with B. The R-economical regularization of all faces of the arrangement defined by B cuts $O(|R| \log |R|)$ segments of R.*

Proof. Since the blue segments are connected, each face of their arrangement is a simple polygon, except for the outer face. To make the outer face simply connected, cut that face by drawing a vertical cut up from the leftmost blue vertex. All faces are now simply connected, and Lemma 4.8 applies to each (the linear cost of the extra cut in the outer face is absorbed in the overall bound). □

4.3.5 Time and space complexity of HeapSweep

It is time to conclude the analysis by putting together the regularization lemma and the analysis of kinetic queues. Recall that n is the total number of red and blue segments, and that k is the number of bichromatic intersections that the algorithm will output.

Theorem 4.10. *The* HeapSweep *algorithm reports all purple intersections between red and blue segments using linear space and $O((n+k)\alpha(n)\log^3 n)$ time.*

Proof. The working space requirement of HeapSweep is simple to assess. Each kinetic queue requires linear space in the number of segments it maintains, and it is responsible for a number of events in the global queue $\mathcal{Q}$ that is also linear in its size. Other than endpoints and kinetic queues internal events, the only other events in $\mathcal{Q}$ are the scheduled purple intersections, of which at any time there can be at most one per red-blue alternation along the sweep line. Thus the total size at any time of the search tree T for endpoints, of the event queue $\mathcal{Q}$, and of all the kinetic queues together, is $O(n)$. We use a straightforward binary heap implementation for $\mathcal{Q}$, so an update (insertion or deletion) in the global queue takes time $t_Q = O(\log n)$.

We now restrict our attention to the time complexity of the maintenance of red blocks. As each segment may appear in several kinetic queues during its life-time, it is necessary to chop all segments into a reasonable number of *fragments*, so that each fragment appears in only one kinetic queue.

The red segments are broken in two steps. First, red segments are broken at all purple intersections into *proto-fragments*. Such an intersection can now be seen (from the point of view of red blocks) as the deletion of a red proto-fragment from one kinetic queue, followed by an insertion of *another* proto-fragment, either in a new kinetic queue, or into an existing one (Figure 4.2–a). If there are n_r red segments, the number of red proto-fragments is exactly $n_r + k$. Second, the R-economical regularization of the faces defined by the blue segments, (Definition 4.3) further partitions the $n_r + k$ red proto-fragments into *fragments*, at the points where they are cut by the vertical threads. From Corollary 4.9, the proto-fragments (which don't intersect the blue segments anymore) are cut in $O((n_r + k) \log n)$ fragments. The merge of two kinetic queues can now be seen as the destruction of all fragments from the smaller one, together with the insertion of equally many *new* fragments into the larger one. Similarly, the split operation involves the deletion of some fragments from one kinetic queue and the creation of a new one made only of *new* fragments.

It is now possible, if we set aside the blue segments, to describe the situation as follows: We have a family S of $O((n_r + k) \log n)$ segments (fragments), partitioned into a number of disjoint subsets $(S_i)_i$, each swept over by a different kinetic queue. Using one of the prescribed kinetic queue implementations, the overall time of insertions, deletions, and internal updates is $\sum_i O(t_Q |S_i| \alpha(S_i) \log |S_i|) = O(t_Q (n_r + k) \alpha(n) \log^2 n)$. The same analysis applies to the cost related to the blue kinetic queues. The maintenance cost of the "interface" between blocks, i.e. the purple events, does not add to the complexity: the scheduling/de-scheduling of a purple event can only be caused by (and charged to) an internal update that modifies the top or bottom of one of the adjacent kinetic queues.

With the bound on t_Q mentioned above, this implies the desired bound on the running time of HEAPSWEEP, which is randomized or deterministic depending on the specific kinetic queue implementation. $\square$

4.4 Convolution computation

In this section, we modify HeapSweep to compute the convolution between two tracings. Three issues arise: a) we don't have segments on the plane, but great circle on the sphere, b) we not only need to detect all intersection between edges, but also output all pairs made of a vertex and a face, and c) we need to compute the connectivity between all the output features, which will implicitly define the indexing manifold of the convolution.

The first issue can be taken care of by projecting the sphere on the plane through an appropriate map, which maps great circles to lines. Once the computation is performed in the plane, it remains to do some additional gluing between both ends of the sweep, but we won't detail the process here. Instead, we focus on the two other more interesting issues in the rest of this section, assuming that the data is given in the plane.

4.4.1 Face-vertex features

The direction map of a polyhedral tracing can be thought of as a subdivision of a surface that folds over the plane possibly several times. This subdivision is made of vertices, edges, and faces, where a vertex (resp. face) corresponds to a face (resp. vertex) of the original polyhedral tracing. To compute the convolution between a red and a blue tracing, we need not only to find the bichromatic intersections between edges, but also all bichromatic vertex/face pairs that overlap. HeapSweep, in the version described above, doesn't keep track of the faces that intersect the sweep line. We cannot precisely keep track of the way faces cover the sweep line, since this would reintroduce the quadratic cost of computing all monochromatic intersections.

We present a solution here. For each red edge, keep all blue faces that are currently passing over it (and vice-versa). Call this list the *cover* of the edge (Figure 4.6). The list of faces in the cover of each edge must be maintained as the algorithm progresses. This

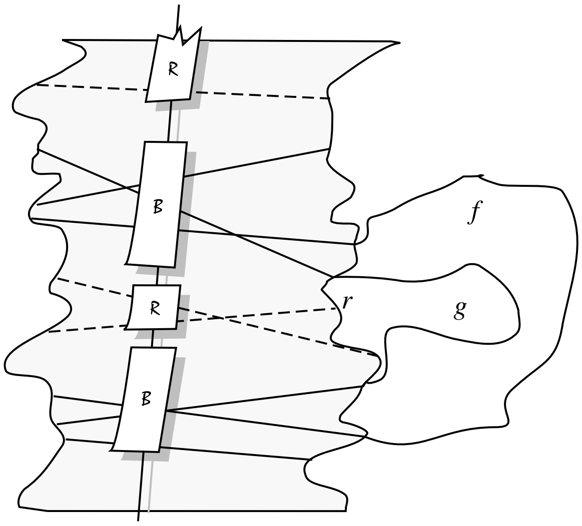

Figure 4.6: Shows an edge r in a red block and two faces f and g in its cover which enclose r.

requires the following modifications to the algorithm (we choose arbitrary colors for each event):

1. When sweeping through a vertex that is not a split or merge vertex, output one purple face per face in the cover of the edge to which this vertex belongs.

2. For a blue vertex b that causes a split of a red block, create the cover of the newly created edge and output one vertex per face in the cover. To compute the cover, first split the red block into two blocks. Consider first the smaller block. Let b' be the first blue vertex on the other side of the smaller block. Copy into the cover of b those faces in the cover of b' that cover b. Then add to the cover of b faces generated by edges of the smaller block.

3. If a blue vertex causes a merge, first output one vertex per face in the cover of its two incident edges that are to be destroyed. Then delete these two covers. The covers of all other edges remain the same.

4. At a bichromatic intersection between a red edge r and a blue edge b, two singleton

blocks are created containing r and b, respectively. The cover of r is created as follows: duplicate the cover of the red block that r belonged to and add the face adjacent to b if it covers r and delete it if it discontinues covering r.

In the above method, all operations can be charged to output features of the convolution. Note that one output feature was created for every pair of edge and face such that the face is in the cover of the edge, since their whisker maps then intersect. The algorithm still runs in nearly linear output-sensitive time, but the ouput is now the total size of the convolution. It is possible to construct an example where the convolution is quadratic with only $O(1)$ purple intersections. There is one drawback to this approach: the space used can be quadratic. We omit some details of this implementation here.

To avoid the quadratic space cost, several variants can be imagined. For instance, it is possible to maintain an *interval tree* [114] of all red faces that intersect with the sweep line, and query this tree with the position of a blue vertex when sweeping over one. However, it is not possible to maintain the exact interval tree, as, once again, this would reintroduce the quadratic cost of detecting all monochromatic intersections. Instead, one can collapse all end points of intervals that end in the same block, as though they all end at the same endpoint. This requires a data structure that handles degeneracies. Special operations still have to be performed during Splits and Merges. Since interval trees are of linear size, the space requirement of this solution is linear.

4.4.2 Connectivity

Computing the interconnectivity of the convolution features can be tricky. In general, we follow this strategy: compute the feature as soon as possible. Connect it (topologically) to all neighboring features that have already been computed. If a neighboring feature has not been computed leave the link suspended until it is. The link gets updated when that neighboring feature has actually been computed.

We store detected features of the convolution in a hash table. The hash key is a unique description of the feature; for example, under the assumptions of Section 3.3, the set of two factor features that generated this feature. When a feature of the convolution is detected, we check if it is already in the hash table. If it is, we update its links to neighboring features; otherwise, we add the feature to the table and add links to its existing neighbors. Once all the features of the convolution are computed, their connectivity links are also in place.

Theorem 4.11. *The convolution of two polyhedral tracings can be computed in time $O(k\alpha(k)\log^3 k)$, where k is the total number of features of the convolution.*

4.4.3 Minkowski sum

As was the case in two dimensions, the convolution is a useful step to compute the Minkowski sum of two polyhedra: it filters out pairs of features that cannot be part of the Minkowski sum, and thus avoids in some cases the quadratic cost of the naive algorithm that tries to pair up all features. Given all the faces of the convolution, a space sweep can be performed to compute the Minkowski sum.

We expect that many algorithms using the Minkowski sum can be speeded up by using the combinatorially smaller convolution directly instead. For example, in order to render the outer face of the Minkowski sum from an external viewpoint, it is sufficient to render the faces of the convolution in any sequence. The depth buffer of current graphics hardware will automatically ensure that only the visible sections of the Minkowski sum are produced. Similarly, algorithms that query the Minkowski sum in various ways, such as ray-shooting, planar sectioning, extrema and bounding volume computations, can all benefit by using the convolution instead of the Minkowski sum. Also, it is likely that the convolution can be used in novel ways along the lines of the second chapter of this thesis.

The winding number of a point $p \in \mathcal{E}^3$ with respect to a convolution $B * R$ of size k can be trivially computed in $O(k)$ time by a ray-shooting algorithm. Since such a ray can intersect $\Omega(k)$ facets of the convolution, this is optimal in the worst case. However in real-life, use of a more efficient ray-shooting structure may make sense.

4.5 Applications and related results

We mention below some extensions to the main algorithm. The proofs of the stated results are obtained from minor modifications of the above analysis.

Monotone arcs. Although the analysis has been performed for line segments for the sake of the exposition, the same algorithm applies for a more general class of arcs. The only difference is the complexity of the upper envelope of a family of arcs, which is then carried through the entire analysis, for both the randomized and the deterministic implementations of a kinetic queue mentioned in this chapter. The regularization lemma also holds if line segments are replaced by x-monotone arcs.

Theorem 4.12. *Let R and B be two sets of x-monotone arcs, such that each monochromatic family defines a connected point set on the plane. Assume that two pairs of arcs intersect at most s times. If n is the number of input arcs, the algorithm* HeapSweep *computes all k purple intersections in time $O(\lambda_{s+2}(n+k)\log^3 n)$, where $\lambda_s(m)$ is the length of the longest (m,s)-Davenport-Schinzel sequence.*

For instance, algebraic curves of bounded degree obey the requirements mentioned in this theorem (provided they are first decomposed into a constant number of x-monotone parts each).

Several components. HeapSweep can be used for more general input in which the blue set has c_b components and the red set has c_r components. In that case, the analysis of Corollary 4.9 needs to be modified: from each component, a thread needs to be drawn to obtain a simple arrangement, and the algorithm runs in time $O((c_b n_r + c_r n_b + k)\alpha(n)\log n)$. If one point from each component is given, it is possible to compute a spanning tree between these points [100] that has low stabbing number, and thus provide connected input to HeapSweep. The spurious intersections generated by the spanning trees cause a total running time of $O((n_r\sqrt{c_b} + n_b\sqrt{c_r} + k)\alpha(n)\log^3 n)$.

General case in linear space. In order to solve the general red-blue intersection problem in linear space, the cuttings technique of [69, 2] can be applied, but the running time becomes $O(n^{4/3+\epsilon} + k)$. The ϵ term can be made as small as desired, but at the cost of a increased hidden constant in the space bounds. Using HeapSweep, we obtain a $O((n^{4/3} + k)\alpha(n)\log^3 n)$ algorithm to report all intersections in the general case, with a linear space cost that has a small implied constant. This can be done by preprocessing the input to identify connected subsets of segments (see [69, 6]), on which it is then possible to use the variation of *HeapSweep* for known components mentioned above. Details are omitted.

j-level. If the j-level in an arrangement of n arcs has complexity k, the algorithm HeapSweep can be adapted to compute it in time $O(\lambda_s(n+k)\log^2 n)$, whether or not the input set is connected, by keeping two kinetic queues grouping all segments above and below the current j-level. This generalizes the results of Edelsbrunner and Welzl [57], who computed the j-level of an arrangement of line segments in $O((n+k)\log^2 n)$ time using a line sweep technique that can be retroactively be seen as a precursor of HeapSweep. Cole, Sharir, and Yap [44] improved these bounds to $O((n+k)\log^2 j)$.

Both methods mentioned above use the data structure of Overmars and van Leeuven [111] for dynamic maintenance of a convex hull with insertions and deletion at a cost of $O(\log^2 n)$ per operation. Mark de Berg pointed out that this data structure can also be used in HeapSweep, replacing the kinetic queue and giving an improved $O((n+k)\log^3 n)$ algorithm for reporting the k purple intersections between n red and blue line segments [47].

4.6 Conclusion

We have presented a new algorithm, HeapSweep, to report all *purple* intersections between red and blue possibly intersecting line segments on the plane. HeapSweep is a variation of the line sweep method that stores only a partial ordering of segments on the sweep line. The sweep line is divided into maximal contiguous blocks of monochromatic segments and the segments of each block are stored in a novel data structure called a kinetic queue. A kinetic queue keeps track of the top and bottom segments of a block and allows the detection of purple intersections, which can only occur between top and bottom segments of consecutive blocks. If the set of red segments (resp. blue segments) is connected, i.e. their union as a point set is connected, we have proved that HeapSweep reports all purple intersections in time $O((n+k)\alpha(n)\log^3 n)$, where n is the input size and k is the number of purple intersections.

We then showed how the red-blue algorithm could be adapted to compute the convolution of two polyhedral tracings. This requires an additional data structures to compute output features that come from vertex-face pairs, but the analysis of the running time carries through, and the convolution can be computed in nearly optimal output-sensitive time.

The connectedness assumption is interesting and practical, and yet there does not seem to be any obvious way to take advantage of it in traditional techniques such as divide-and-conquer, random sampling, and segment trees – since this property is not preserved in subsets of the input set. In contrast, HeapSweep makes interesting use of connectedness through the regularization lemma, which may find use in other applications.

There are still many logarithms in the running time of HeapSweep, in particular of the k term. In order to reduce this overhead, it would be nice to integrate our technique, which relaxes the vertical ordering of the sweep line, with the methods of Edelsbrunner and

Guibas [54] or Mairson and Stolfi [99], which relax the horizontal ordering of the sweep. The bounds we obtained also give hope that a filtering search technique [31] coupled with geometric partitioning could achieve an optimal $O(n \log n + k)$ running time (in a sense, HeapSweep is an instance of filtering search as it detects more intersections than necessary – but not too many more).

There doesn't seem to be any hope of adapting HeapSweep to solve the *counting* problem efficiently. The hereditary segment-tree approach [36] used to obtain an optimal algorithm for counting purple intersections in the disjoint case cannot be adapted either, since segment tree nodes may have a disconnected set of segments. In fact we believe that connectedness does not simplify the counting problem, and we might be able to see an $\Omega(n^{4/3})$ lower bound to substantiate this claim.

Chapter 5

Conclusion

5.1 Thesis summary

In this thesis, we investigated a framework for representing and computing convolutions of piecewise linear objects in two and three dimensions. For simple polygons, we presented a new algorithm to compute the outer face of their Minkowski sum making use of their convolution. This algorithm exploits a new property of the convolution of simple polygonal tracings; it is relatively simple to implement and has almost as good running time performance as previously known algorithms to compute the Minkowski sum.

Generalizing polygonal tracings to three dimensions, we develop a model and data structure representation for polyhedral tracings. We then extended the operation of convolution to polyhedral tracings: given two input polyhedral tracings, we specified in detail the output resulting from convolving the two tracings. The convolution is specified as a boundary half-edge data structure.

We then described a method to compute the convolution of two polyhedral tracings as a half-edge structure, given the input half-edge structures of two polyhedral tracings. The algorithm employs a novel method to intersect two sets of line segments to find intersections of segments between the two sets, without looking at intersections between segments within each set.

Finally, we described an application and implementation of polyhedral tracings in the context of assembly planning.

5.2 Open problems and future work

Further applications of the "cycle-cutting" paradigm used in the planar outer-face computation remain to be explored. There is also scope for making the method more efficient by finding a better way to connect the cycles of the convolution. The paradigm is fairly simple to implement in two dimensions, but is difficult to extend to three dimensions. Thus, it remains to find a efficient algorithms to compute the outer face of the Minkowski sum of polyhedra.

In this thesis, we generalized the theory of tracings to three dimensions. It remains to develop a theory of tracings for higher dimensions. There is scope for augmenting e.g. the cell tuple representation of higher-dimensional complexes [27] to create higher-dimensional tracings. However, the computational side is very challenging. For instance, in order to compute efficiently the convolution of two four-dimensional tracings, we would have to solve the following bichromatic problem: given a red and a blue family of triangles in space, compute all bichromatic edges. Would a connectedness hypothesis allow us to get a better running time than what could be achieved using general cutting techniques [102]?

In Chapter 3, we proved that the convolution of polyhedral tracings has well defined and balanced structure. Conjecture 1 stated that the convolution satisfies the fundamental theorem that relates it to the intersection of the corresponding polyhedral paintings. The fundamental theorem of convolutions was proved for planar tracings in [70]. Proving the theorem for higher dimensions remains an important open problem.

In the case of a robot allowed only to translate in space, the notion of C-obstacle (the Minkowski difference between the obstacle and the robot) was advantageously replaced by the convolution. The theory of tracings gave a firm computational basis to the convolution: the convolution, in the signed half-edge representation, is easier to compute than the Minkowski sum.

For more general robots with arbitrary degrees of freedom, the notion of *configuration space* and of *C-obstacle* is well defined. It would be simple to define an operation similar to the convolution of paintings in this context: for a given placement of the robot, one could replace the boolean value "does the robot intersect with the obstacle?" by the Euler-Poincaré characteristic of the intersection. We may expect with this definition to obtain a setting as rich as Schapira's paintings. However, the computational side seems more difficult. A boundary representation of C-obstacles in configuration space along with a convolution operation would be very useful.

With the help of the *winding number*, it is possible to compute the painting corresponding to a tracing. Also, there is a well defined topological operation that allows us to compute the boundary of a closed set. It is therefore natural to ask whether it is possible to devise a rule that gives a tracing corresponding to a painting.

Convolutions could make it possible to compute local regions of configuration spaces for polyhedral parts or robots in motion. Often, an initial configuration of the robot or part is known that is not very far from the desired final configuration. Local computation of configuration spaces allows for locally maneuvering a robot or a polyhedral part under the constraint of a physical barrier. Further applications of convolutions to motion planning and assembly planning remain to be studied.

Bibliography

[1] P. K. Agarwal. Partitioning arrangements of lines: I. An efficient deterministic algorithm. *Discrete Comput. Geom.*, 5:449–483, 1990.

[2] P. K. Agarwal. Partitioning arrangements of lines: II. Applications. *Discrete Comput. Geom.*, 5:533–573, 1990.

[3] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 67–75, 1994.

[4] P. K. Agarwal, D. Eppstein, and J. Matoušek. Dynamic half-space reporting, geometric optimization, and minimum spanning trees. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 80–89, 1992.

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

[6] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 70–80, 1988.

[7] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications t o motion planning and collision detection. *SIAM J. Comput.*, 19(2):297–321, 1990.

[8] Pankaj K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[9] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, MA, 1983.

[10] C. Aragon and R. Seidel. Randomized search trees. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 540–545, 1989.

[11] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):209–216, 1991.

[12] B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. *Combinatorica*, 10(2):137–173, 1990.

[13] B. Aronov and M. Sharir. On translational motion planning in 3-space. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994.

[14] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11:1171–1181, 1985.

[15] F. Aurenhammer. A new duality result concerning Voronoi diagrams. *Discrete Comput. Geom.*, 5:243–254, 1990.

[16] C. Bajaj and M. Li. On the duality of intersections and closest points. In *Proc. 21st Allerton Conf. Commun. Control Comput.*, pages 459–461, 1983.

[17] Chanderjit L. Bajaj and Myung-Soo Kim. Generation of configuration space obstacles: The case of moving algebraic curves. *Algorithmica*, 4:157–172, 1989.

[18] J. Basch, L. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[19] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[20] J. Basch, L. J. Guibas, and J. Hershberger. Data structure for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, January 1997.

[21] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th Annu. European Sympos. Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 302–319. Springer-Verlag, 1996.

[22] J. Basch, L. J. Guibas, and G.D. Ramkumar. Sweeping lines and line segments with a heap. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 469–471, 1997.

[23] J. Basch, L.J. Guibas, G.D. Ramkumar, and L. Ramshaw. Polyhedral tracings and their convolution. In *Proc. 2nd Workshop on Algorithmic Foundations of Robotics*, 1996.

[24] J. Basch and L. Ramshaw. Orienting transverse fiber products. *Manuscript.*

[25] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[26] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, 1991.

[27] E. Brisson. Representing geometric structures in d dimensions: Topology and order. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 218–227, 1989.

[28] K. Q. Brown. Comments on "Algorithms for reporting and counting geometric intersections". *IEEE Trans. Comput.*, C-30:147–148, 1981.

[29] J. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1987.

[30] T. M. Chan. A simple trapezoid sweep algorithm for reporting red/blue segme nt intersections. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 263–268, 1994.

[31] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986.

[32] B. Chazelle. Reporting and counting segment intersections. *J. Comput. Syst. Sci.*, 32:156–182, 1986.

[33] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[34] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[35] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. 18th Internat. Colloq. Automata Lang. Program.*, volume 510 of *Lecture Notes in Computer Science*, pages 661–673. Springer-Verlag, 1991.

[36] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[37] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments and related problems. *SIAM J. Comput.*, 22:1286–1302, 1993.

[38] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[39] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[40] S. W. Cheng and R. Janardan. Space-efficient ray shooting and intersection searching: algorithms, dynamization and applications. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 7–16, 1991.

[41] L. P. Chew and K. Kedem. A convex polygon among polygonal obstacles: Placement and high-clearance motion. *Comput. Geom. Theory Appl.*, 3:59–89, 1993.

[42] K. L. Clarkson. Randomized geometric algorithms. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 117–162. World Scientific, Singapore, 1992.

[43] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[44] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.

[45] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture notes in computer science*, volume 33, pages 135–183. Springer-Verlag, New York, 1975.

[46] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1990.

[47] M. de Berg. Personal communication. 1996.

[48] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 105–114, 1994.

[49] M. de Berg and O. Schwarzkopf. Cuttings and applications. Report RUU-CS-92-26, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, August 1992.

[50] O. Devillers, S. Meiser, and M. Teillaud. The space of spheres, a geometric tool to unify duality results on Voronoi diagrams. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 263–268, 1992.

[51] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Implicitly searching convolutions and computing depth of collision. In *Proc. 1st Annu. SIGAL Internat. Sympos. Algorithms*, volume 450 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 1990.

[52] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–33, 1993.

[53] H. Edelsbrunner. Algebraic decomposition of non-convex polyhedra. Technical report, University of Illinois at Urbana-Champaign, 95.

[54] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989. Corrigendum in 42 (1991), 249–251.

[55] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The complexity and construction of many faces in arrangements o f lines and of segments. *Discrete Comput. Geom.*, 5:161–196, 1990.

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

[57] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[58] Herbert Edelsbrunner. Algebraic decompositions of non-convex polyhedra. In *Proc. 36th Annu. IEEE Sympos. Found. Comput. Sci.*, page To appear, 1995.

[59] Jeff Erickson. New lower bounds for Hopcroft's problem. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 127–137, 1995.

[60] R. C. Evans, G. Koppelman, and V. T. Rajan. Shaping geometric objects by cumulative translational sweeps. *IBM J. Res. Develop.*, 31:343–360, 1987.

[61] Ulrich Finke and Klaus Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 119–126, 1995.

[62] P. K. Ghosh and S. P. Mudur. A computational theoretic framework for Minkowski addition and decomposition of boundary represented regions. Report ??, National Centre Softw. Develop. Comput. Tech., Tata Inst. Fund. Res., Bombay, India, 1985.

[63] Pijush K. Ghosh. *Vision, Geometry, and Minkowski Operators*, volume 119. 1991.

[64] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths via balanced geodesic triangulations. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 318–327, 1993.

[65] P. K. Gosh. A solution of polygon containment spatial planning and other related problems using Minkowski operators. *Comput. Vision Graph. Image Process.*, 49:1–35, 1990.

[66] R. H. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison Wesley, 1989.

[67] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: Computational complexity and applications. *SIAM J. Discrete Math.*, 6:246–269, 1993.

[68] Peter Gritzmann and Alexander Hufnagel. A polynomial time algorithm for Minkowski reconstruction. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 1–9, 1995.

[69] L. Guibas, M. Overmars, and M. Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. 1st Scand. Workshop Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 64–73. Springer-Verlag, 1988.

[70] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[71] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[72] L. J. Guibas, M. Sharir, and S. Sifrony. On the general motion planning problem with two degrees of freedom. *Discrete Comput. Geom.*, 4:491–521, 1989.

[73] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[74] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[75] D. Halperin, M. H. Overmars, and M. Sharir. Efficient motion planning for an L-shaped object. *SIAM J. Comput.*, 21:1–23, 1992.

[76] D. Halperin and M. Sharir. Arrangements and their applications in robotics: Recent developments. In K. Goldbergs, D. Halperin, J.-C. Latombe, and R. Wilson, editors,

Proc. Workshop on Algorithmic Foundations of Robotics. A. K. Peters, Boston, MA, 1995.

[77] D. Halperin and R. H. Wilson. Assembly partitioning along simple paths: the case of multiple translations. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1585–1592, 1995.

[78] Sariel Har-Peled, Timothy M. Chan, Boris Aronov, Dan Halperin, and Jack Snoeyink. The complexity of a single face of a Minkowski sum. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 91–96, 1995.

[79] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

[80] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[81] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: P-space hardness of the "Warehouseman's Problem". *Internat. J. Robot. Res.*, 3(4):76–88, 1984.

[82] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. *Planning, Geometry, and Complexity of Robot Motion.* Ablex Publishing, Norwood, NJ, 1987.

[83] Y. B. Karasik and M. Sharir. The power of geometric duality and Minkowski sums in optical computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 379–388, 1993.

[84] A. Kaul and M. A. O'Connor. Computing minkowski sums of regular polyhedra. Report RC 18891 (82557) 5/12/93, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1992.

[85] 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, 1991.

[86] A. Kaul, M. A. O'Connor, and V. Srinivasan. Computing Minkowski sums of regular polygons. Report RC 18523 (80982) 11/13/93, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1992.

[87] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. *Comput. & Graphics*, 16:107–116, 1992.

[88] S. Khanna, R. Motwani, and R. Wilson. On certificates and lookahead in dynamic graph problems. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 7–16, 1996.

[89] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1968.

[90] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 2nd edition, 1973.

[91] S.S Krishnan and A.C. Sanderson. Path-planning algorithms for assembly sequence planning. In *Proceedings of the SPIE*, volume 1571, pages 428–439, 1991.

[92] Serge Lang. *Differential Manifolds*. Addison-Wesley, 1972.

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

[94] JC. Latombe and R.H. Wilson. Assembly sequencing with toleranced parts. In 3^{rd} *ACM Symposium on Solid Modeling and Applications*, Salt Lake City, 1995.

[95] J.-P. Laumond. Obstacle growing in a nonpolygonal world. *Inform. Process. Lett.*, 25:41–50, 1987.

[96] D. T. Lee and Y. T. Ching. The power of geometric duality revisited. *Inform. Process. Lett.*, 21:117–122, 1985.

[97] D. Leven and M. Sharir. An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 221–227, 1985.

[98] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[99] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume F40 of *NATO ASI*, pages 307–325. Springer-Verlag, Berlin, West Germany, 1988.

[100] J. Matoušek. Spanning trees with low crossing number. *Informatique Théorique et Applications*, 25(2):103–123, 1991.

[101] J. Matoušek. Geometric range searching. Tech. Report B-93-09, Fachbereich Mathematik und Informatik, Free Univ. Berlin, 1993.

[102] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.

[103] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14:257–276, 1985.

[104] K. Mehlhorn and S. Näher. Implementation of a sweep line algorithm for the straight line segment intersection problem. Report MPI-I-94-160, Max-Planck-Institut Inform., Saarbrücken, Germany, 1994.

[105] David M. Mount and Ruth Silverman. *Combinatorial and Computational Aspects of Minkowski Decompositions*, volume 119. 1991.

[106] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.

[107] K. Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete Comput. Geom.*, 6:307–338, 1991.

[108] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[109] T. Ottmann and D. Wood. Dynamical sets of points. *Comput. Vision Graph. Image Process.*, 27:157–166, 1984.

[110] M. Overmars, H. Schipper, and M. Sharir. Storing line segments in partition trees. *BIT*, 30:385–403, 1990.

[111] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[112] J. Pach and M. Sharir. On vertical visibility in arrangements of segments and the queue size in the Bentley-Ottman line sweeping algorithm. *SIAM J. Comput.*, 20:460–470, 1991.

[113] Larry Palazzi and Jack Snoeyink. Counting and reporting red/blue segment intersections. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 530–540, 1993.

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

[115] G. D. Ramkumar. An algorithm to compute the Minkowski sum outer-face of two simple polygons. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 234–241, 1996.

[116] G. D. Ramkumar. Algorithm to compute the minkowski sum outer-face of two simple polygons. In *Proc. 12th Annu. ACM Sympos. Comput. Geometry*, 1996.

[117] Jean-Jacques Risler. Placement of curved polygons. Report LMENS - 91 -2, Laboratoire de Mathématiques,Ecole Normale Supérieure, UA 762 du CNRS, 1991.

[118] Bruce Romney, Cyprien Godard, Michael Goldwasser, and G. Ramkumar. An efficient system for geometric assembly sequence generation and evaluation. In *Proceedings of the ACME International Computers in Engineering*, pages 699–712, 1995.

[119] J. Rossignac and A. Kaul. Agrels and bips: Metamorphosis as a bézier curve in the space of polyhedra. In *Eurographics '94 Proceedings*, volume 13, pages 179–184, 1994.

[120] G. Sapiro, R. Kimmel, D. Shaked, B. B. Kimia, and A. M. Bruckstein. Implementing continuous-scale morphology via curve evolution. Technical Report CIS 9302, Technion IIT, Haifa, Israel, January 1993.

[121] Pierre Schapira. Operations on constructible functions. *Journal of Pure and Applied Algebra*, 72:83–93, 1991.

[122] J. T. Schwartz and M. Sharir. On the "piano movers" problem I: the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun. Pure Appl. Math.*, 36:345–398, 1983.

[123] 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.

[124] J. T. Schwartz and M. Sharir. On the "piano movers" problem V: the case of a rod moving i n three-dimensional space amidst polyhedral obstacles. *Commun. Pure Appl. Math.*, 37:815–848, 1984.

[125] Otfried Schwarzkopf. Computing convolutions on mesh-like structures. pages ??–?? IEEE Computer Society, 1993.

[126] A. Schweikard and R.H. Wilson. Assembly sequences for polyhedra. *Algorithmica*, 13(6):539–552, June 1995.

[127] M. I. Shamos. *Computational Geometry.* Ph.D. thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978.

[128] M. Sharir. On k-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.

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

[130] S. Sifrony and M. Sharir. A new efficient motion-planning algorithm for a rod in two-dimensional polygonal space. *Algorithmica*, 2:367–402, 1987.

[131] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations.* Academic Press, New York, NY, 1991.

[132] R. H. Wilson and A. Schweikard. Assembling polyhedra with single translations. Technical Report STAN-CS-91-1387, Dept. Comput. Sci., Stanford Univ., Stanford, CA, October 1991.