The Theory and Practice of Robust Geometric Computation, or, How to Build Robust Solid Modelers

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Arthur James Kennedy Stewart
August 1991

The Theory and Practice of
Robust Geometric Computation,
or,
How to Build Robust Solid Modelers

Arthur James Kennedy Stewart, Ph.D.
Cornell University 1991

The field of solid modeling makes extensive use of a variety of geometric algorithms. When implemented on a computer, these algorithms often fail because the computer only provides floating point arithmetic, while the algorithms are expecting infinite precision arithmetic on real numbers. These algorithms are not *robust*.

This dissertation presents a theory of robustness. With this theory, a robust point location algorithm is developed. This algorithm determines whether a point lies inside a polygon, where both the point and the polygon are represented approximately.

The elegant theoretical approach to robustness is not viable in practice: algorithms like those used in solid modeling are generally too complex for this approach. This dissertation presents a practical alternative to the formal theory of robustness; this alternative is called *local robustness*.

Local robustness is applied to the design of a polyhedral intersection algorithm, which is an important component in many solid modelers. The intersection algorithm has been implemented, and, in extensive tests, has never failed to produce a valid polyhedron of intersection. A concise characterization of the locally robust intersection algorithm is presented; this characterization can be used to develop variants of the intersection algorithm, and to develop robust versions of other solid modeling algorithms.

Biographical Sketch

James was born on January 31, 1963. His early interest in computing survived exposure to the venerable computing machines of his teens, and he entered Queen's University in Kingston, Ontario in 1981. Four years later he left Queen's and decided to avoid graduate school for another year by moving to Switzerland, where in 1985 he worked as a programmer at the University of Geneva. Quickly fed up with real work, he returned in 1986 to enter graduate school at Cornell University in sunny Ithaca, New York. There he finally learned to play hockey (though some might disagree) and produced a thesis in his spare time. He reluctantly left Cornell in August of 1991.

To Alta and Alec,
the best of parents

Acknowledgements

Many thanks go to John Hopcroft, whose most important contribution was to teach me to recognize interesting problems. His intuition was a great help in avoiding blind alleys, and his insistence on clarity and brevity helped me to write good.

Many thanks also to Tom Coleman, who served as the other computer science member of the committee, and to the holy trinity of linguisitcs members: Susan Hertz, who actually supervised my linguistics work, John Kingston, who advised on courses and signed in the right places, and Abby Cohn, who kindly attended the defense and who chose not explore the depth of my knowledge of linguistics.

I really appreciate the efforts of my thesis readers: Jim Cremer read all of the details and didn't let me get away with too much sleight-of-hand, and Kjartan Stefansson read all of the hideous proofs and concluded with "Looks fine to me."

Lots of time was spent in interesting discussions — not always academic — with the various member of the modeling and simulation group: Paul Chew (who also provided Figure 1.1 in this dissertation), Jim Cremer, Baining Guo, John Hopcroft, Anne Neirynck, Dinesh Pai, Rick Palmer, Daniela Rus, and Rich Zippel.

My housemates deserve special commendation for putting up with me: Pat Stephenson, who in the end didn't get the kitchen painted blue, but who did pay off his bet, Kieran Herley, who was always tolerant of unwashed dishes and general untidiness, and whose brisk "walks" to and from school kept me from becoming too sedentary, Kjartan Stefansson, who taught me a useful Icelandic phrase (but never managed to teach me how to pronouce his name) and who shared my enthusiasm for woodworking, and Juan Elezgaray, who enriched my French vocabulary (oh putain!) and played wonderful piano pieces. I'll always remember the Albany Street parties and the late nights drinking beer on the porch.

I hope that all of the other friends won't be offended if they are not exhaustively enumerated, but they know who they are. Some are members of the AI drinking crowd, some I played hockey or bridge with, and others participated in the many rounds of dinner invitations. Anne Rogers has a special place; she introduced me to cooking and hockey (maybe a bit reluctantly) and shared some very special times with me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unprovided with original learning, unformed in the habits of thinking, unskilled in the arts of composition, I resolved to write a book. – Edward Gibbon

Geometric algorithms are used in many important applications, such as computer aided design and finite element analysis. These algorithms, though provably correct in a formal sense, often fail when implemented on a computer.

This failure is not simply the fault of “bad programming”. Failure occurs because the computer only provides floating point arithmetic where the program is expecting perfect arithmetic on real numbers. The absence of perfect arithmetic forces the program to make guesses where it should make accurate decisions, and often leads to failure.

Informally, an algorithm whose computer implementation doesn't fail is called *robust*. This dissertation develops both theoretical and practical approaches to the building of robust algorithms. On the theoretical side of things, a theory of robustness is presented and some algorithms are considered under this theory. On the practical side of things, it is argued that the formal notion of robustness is too rigorous for practical applications, and an alternative — called “local robustness” — is presented.

Local robustness is applied to the design of a polyhedral intersection algorithm. The intersection algorithm has been implemented, and, in over two hundred rigorous tests which would crash most intersection algorithms, has never failed to produce a valid polyhedron of intersection.

1.1 Geometric Algorithms Are Useful

There are many geometric algorithms with significant practical applications. For example, triangulation algorithms[1] are used to generate finite element meshes [5], such as the one shown in Figure 1.1. These meshes can be used in evaluating partial differential equations to compute such things as heat flow in solid objects and the performance of materials under physical stress.

Other practical applications include graphics, which uses hidden line removal algorithms, computer vision, which uses convex hull and Voronoi diagram algorithms, and robot path planning, which uses polyhedral intersection algorithms to determine potential

[1] See [9] or [31] for an explanation of these and the other algorithms mentioned here.

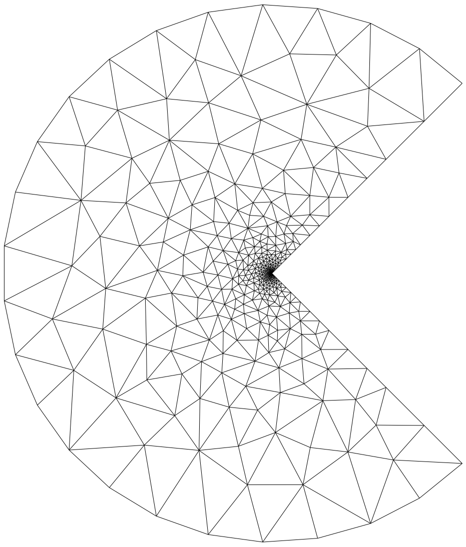

Figure 1.1: A Finite Element Mesh

collisions. But the application that is of greatest interest to the author is that of solid modeling.

Solid modelers — also known as CAD systems, for Computer Aided Design — are very useful in designing manufactured goods such as tools and automobiles. A solid modeler enables the designer to construct computer representations of objects, much like an electronic drafting board. The computer representation of an object can then be manipulated by other programs. For example, the computer representation of a shock absorber might be used by another program to simulate the action of the shock absorber on a bumpy road.

Many solid modelers use a "boundary representation" for their objects: each object is represented as collection of faces, where each face is some mathematical surface. The simplest such surface is a plane, and the objects representable with planar faces are called *polyhedra*.

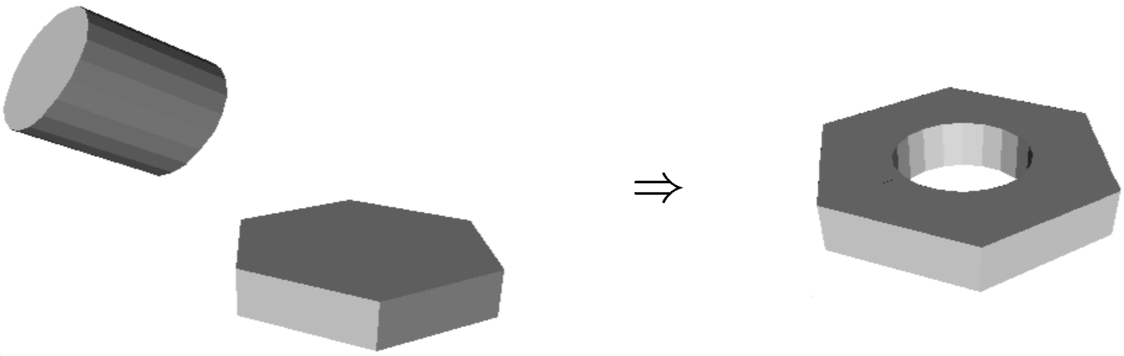

Figure 1.2: Constructing a Machine Nut

With a solid modeler, the designer builds an object by combining simpler objects. For example, the threadless machine nut of Figure 1.2 is created by intersecting a hexagonal slab with the complement of a solid cylinder.

A solid modeler has three basic operations with which to build objects: intersection, union, and complementation. The complement of an object is trivial to compute, whereas the union of two objects can be computed using the complementation and intersection operations. Thus, intersection is a necessary operation, and a *polyhedral intersection algorithm* is of fundamental importance to solid modeling.

1.2 Three Examples of Failure

Despite all of these useful geometric algorithms, very little attention has been paid to developing corresponding robust implementations. The implementations often fail because they neglect to consider the side effects of floating point arithmetic, such as roundoff error. This section will demonstrate what can happen when we implement algorithms without careful consideration of the underlying model of arithmetic.

A Z-buffer algorithm (see [11] or [32]) computes the image of a three dimensional scene for display on a computer screen. The output of one such Z-buffer algorithm is shown in Figure 1.3, in which there are two superimposed objects that differ by a few degrees. Figure 1.4 shows the output of the algorithm when the objects differ by about one hundredth of a degree. In this case the intersections of the planar faces are shown as wavy curves, instead of the straight lines which one would expect. This algorithm is not robust; it fails when presented with input that is nearly degenerate.

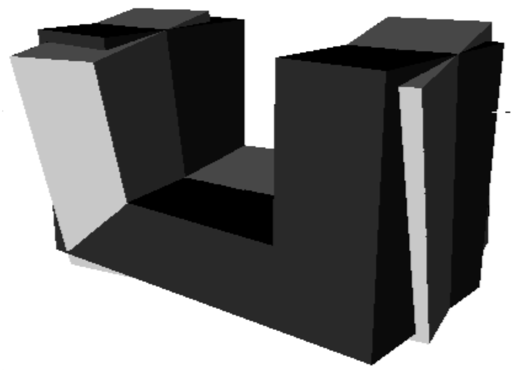

Figure 1.3: Objects Differing By 10°

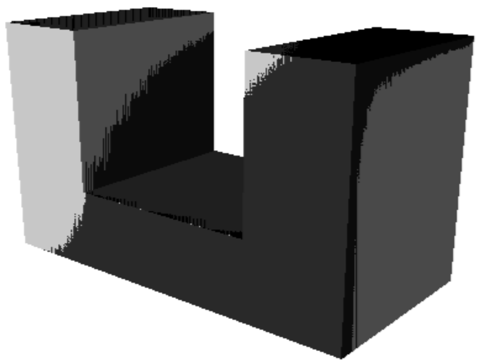

Figure 1.4: Objects Differing By 0.01°

The convex hull of a set of points is the smallest convex set containing those points. Intuitively, the boundary of the convex hull contains exactly those points that touch a rubber band stretched around the set of points, as shown in Figure 1.5. Convex hulls are used in applications as diverse as computer vision and statistics [31]. Convex hull algorithms are numerous; Graham [14] and Jarvis [21] give some of the first.

When the precision of the intermediate computations is reduced, a convex hull algorithm will either simply fail, or will output a hull that is not the true hull of the given points. The output may not even be convex, as shown in Figure 1.6. Algorithms that do this are not robust; they fail when executed with too little precision.

By far the worst failures occur in solid modelers; these programs either simply crash when presented with certain near-degenerate inputs, or generate output that doesn't correspond to any physical object. If the user is not careful, the error in the output may not be discovered until much later in the design cycle; this can be very expensive in terms of time and money.

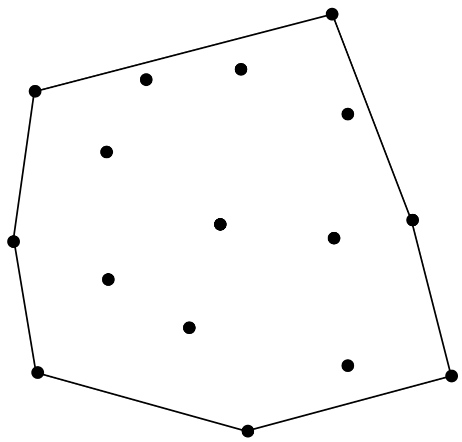

Figure 1.5: A Convex Hull

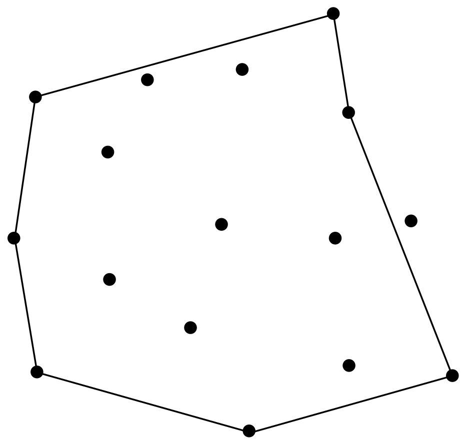

Figure 1.6: An Incorrect Convex Hull

A typical solid modeler can be made to crash by asking it to compute the intersection of two superimposed cubes, one of which has been rotated about some axis by some small angle — usually between one-half and one degree [18,22,23]. A procedure for inducing such a failure is discussed in Chapter 9. The typical output of a failed program is some meaningless error message, like

```
Segmentation fault (core dumped)
```

and can be very frustrating to the user.

These examples demonstrate what happens when we ignore the underlying model of arithmetic. In general, geometric algorithms which are provably correct for the model of real numbers will fail when executed with floating point arithmetic.

1.3 Why Do These Algorithms Fail?

The input to an algorithm can be thought of as a sequence of bits, $\overline{x}$, and the algorithm itself can be thought of as iteratively computing functions of this input, $f(\overline{x})$, then branching based upon the sign of $f(\overline{x})$ [31]. For example, an algorithm can determine whether a point, v, is below a plane, P, which has the equation $P(x) = 0$ by evaluating $P(v)$: if the sign of $P(v)$ is negative then the point lies below the plane; if it is positive then the point lies above the plane.

If an algorithm could compute with *real* arithemetic, then it could always compute $f(\overline{x})$ exactly and would always make the correct branch. However, an algorithm that operates with floating point arithmetic cannot be certain of the sign of $f(\overline{x})$ when

$$|f(\overline{x})| \; < \; \varepsilon,$$

for some small constant ε. We will call this situation an *ambiguous branch*.

The first cause of an ambiguous branch is *roundoff error*. A floating point number (for example, 1.234×10^{56}) is represented with two components: a mantissa (1.234) and an exponent (56). If two numbers, a and b, are represented with P-bit mantissas, then the product, ab, must in general be represented with a $2P$-bit mantissa. However, to keep the size of these numbers from growing, floating point arithmetic discards the least significant

P bits of ab. Thus, if we try to subtract a third number, c, from ab we cannot compute the sign of $ab - c$ when

$$|ab - c| \;<\; 2^{-P}.$$

The second cause of an ambiguous branch is *inaccurate input*. For example, suppose we represent a polyhedron with a set of plane equations and some topological information. It is possible that the topological information specifies that four planes are to meet at a vertex, but that the actual planes (defined by floating point coefficients) meet instead in a structure consisting of two vertices and a short edge [17], as shown in Figure 1.7.

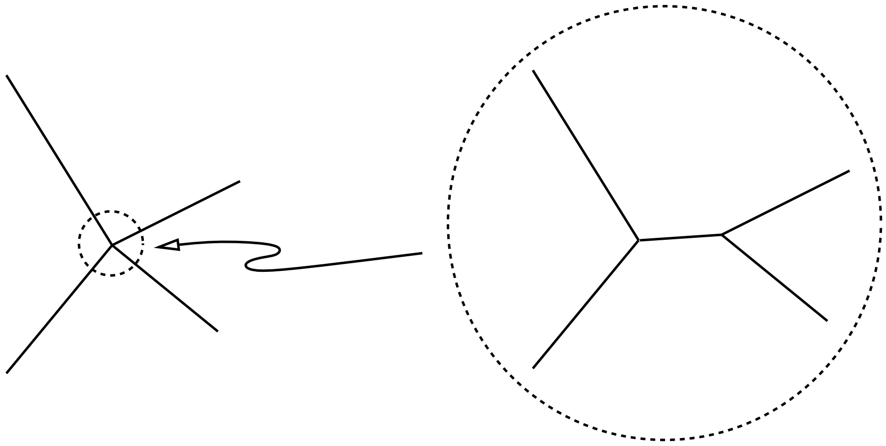

Figure 1.7: Four Planes Meeting

If the four planes do indeed meet at two vertices, then we must assume that the coefficients of the plane equations are inaccurate by some small amount δ. Thus, when a point v is sufficiently close to a plane P, the algorithm cannot determine by *any* numeric test whether v lies on, above, or below P.

When presented with an ambiguous branch the algorithm will make an arbitrary decision. Most algorithms use the rule that

$$|\tilde{f}(\overline{x})| \;<\; \varepsilon \;\Rightarrow\; f(\overline{x}) = 0,$$

where $\tilde{f}$ is the computed value of f and ε is some small constant. This is the principle of "if it's close to zero, then it *is* zero", and is a common source of failure in many geometric algorithms.

For example, consider Figure 1.8. On line ℓ_1, the points a and b are "close" to the line ℓ_2 (indicated by their gray regions of uncertainty), and by the aforementioned principle are considered to be ON ℓ_2. Since ℓ_1 and ℓ_2 share the distinct points a and b, these two lines must be coincident. But this is contradicted by the point c, which is ON ℓ_1 and OFF ℓ_2. Thus the algorithm has made contradictory decisions. Such decisions often lead to failure.

In summary, geometric algorithms that use floating point arithmetic are often faced with ambiguous branches. In such cases, the algorithm will usually make an arbitrary decision. If the algorithm makes a series of arbitrary decisions that are mutually contradictory, then the algorithm will probably fail. Robust algorithms do not make contradictory decisions.

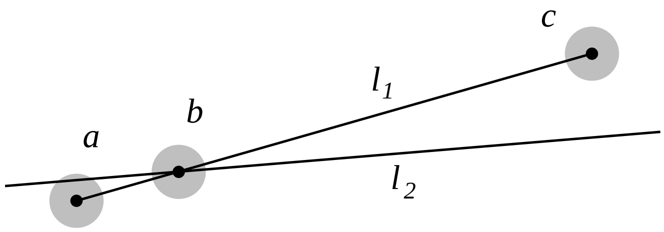

Figure 1.8: Misclassifying Points

1.4 Overview of the Thesis

We need a theory with which to build robust algorithms. Furthermore, this theory should be practical enough to be usable in the construction of real programs.

Chapter 2 reviews the approaches taken by other researchers in building robust algorithms. Chapter 3 presents a formal theory of robustness. Using this formal approach, Chapter 4 develops a robust algorithm to determine whether an approximately represented point lies inside an approximately represented polygon; this algorithm is useful in robust polyhedral intersection. Chapter 5 demonstrates that a particular problem having to do with approximate polygons is NP-hard.

In Chapter 6 it is argued that the formal notion of robustness is too rigorous for practical applications, and that an alternative approach — called *local robustness* — must be taken. Chapters 7 and 8 contain the local robustness theorems concerning the polyhedral intersection algorithm. Chapter 9 discusses how these theorems are incorporated in a polyhedral intersection algorithm, and presents empirical tests of the algorithm. The conclusions are presented in Chapter 10.

Chapter 2

Other Work

This chapter will review the approaches taken by other researchers in building robust algorithms. For another survey, the reader is directed to Hoffmann [16]. There are three general approaches:

- use exact, high precision arithmetic,
- compute on some implicit model, and
- maintain topological consistency.

2.1 Exact Arithmetic Methods

Algorithms in this category operate on exact computer representations of objects. For example, a polygon might be represented by an ordered list of vertices, the coordinates of which are given as exact points on an integer grid.

Given some exact input, $\overline{x}$, the algorithm can use high precision intermediate computations to compute the exact value of functions of the input, $f(\overline{x})$. Thus, an algorithm that uses exact input will never face an ambiguous branch, and will always succeed.

2.1.1 Greene and Yao

Greene and Yao [15] build a robust algorithm to report all segment intersections among a set of segments in the plane, where the segment endpoints are exactly represented as points on an integer grid. To maintain this exact representation, intermediate computations shift segment intersections onto grid points, and coerce segments to pass through certain other grid points in a manner that preserves the topology of the segments.

Since all intermediate high precision computations are done on exactly represented grid points, there are no ambiguous branches. Thus, the algorithm robustly computes the intersections of some set of polygonal line segments that are "close" to the original segments given as input to the algorithm.

2.1.2 Ottmann, Thiemt, and Ullrich

Ottmann, Thiemt, and Ullrich [30] also build a robust algorithm to report segment intersections. Like Greene and Yao, they also consider the segments as exact data. Their work gives a general method of computing arithmetic functions composed of the operators $+$, $-$, and $*$ in floating point arithmetic. Using their method, the least significant bit of the computed function is accurate. This method is shown to be applicable to many of the tests found in standard geometric algorithms.

2.1.3 Milenkovic

Milenkovic [25] builds a polygonal modeling system that operates on "normalized polygons". These normalized polygons obey a set of rules that allow the use of limited precision arithmetic in performing geometric tests. The process of converting an approximate polygon into a normalized polygon is called "accommodation" and involves shifting vertices and "cracking" edges to ensure a minimum feature separation, as shown in Figure 2.1. This approach is similar to that of Greene and Yao, except that the vertices are not shifted onto grid points and efficient, limited precision arithmetic is used.

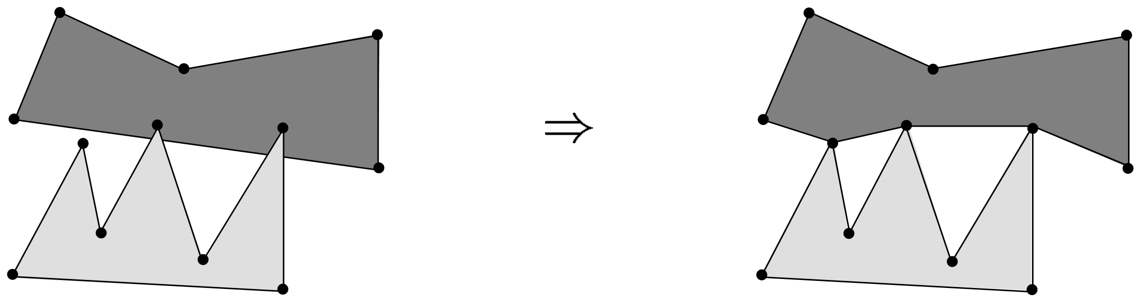

Figure 2.1: Edge Cracking

2.1.4 Sugihara and Iri

Sugihara and Iri [40] propose a polyhedral modeling system that uses exact input and high precision intermediate computations. A two dimensional version of the modeling system has been implemented, with promising results. The three dimensional polyhedral modeling system would allow an object to be defined by a set-theoretic combination of a restricted class of trihedral polyhedra[1] whose faces lie on a restricted set of exactly represented planes.

The basic geometric test performed by the modeling system would be to determine the orientation of four planes; this can be accomplished exactly with a high precision computation. The basic operation to construct objects would be $A \leftarrow A * T(P)$, where A is an object, P is a primitive object, T is a coordinate transformation like rotation or translation, and $*$ is a set-theoretic operation like intersection or union. It is shown that this operation does not restrict the set of representable objects.

The modeling system would allow rotations and translations to be applied to primitive objects. However, when such objects are transformed, their plane equations must generally be represented with more accuracy than before. Sugihara and Iri suggest that, since only

[1]A polyhedron is *trihedral* if each vertex is adjacent to exactly three faces.

trihedral primitive objects are transformed, these objects can be represented approximately (that is, rounded) without altering the topological structure. However, a later result of Milenkovic and Nackman [29] shows that, in general, trihedral polyhedra *cannot* be rounded efficiently. Thus, if we use exactly represented polyhedra, then we must find a restricted set of "roundable" rotations for these polyhedra.

One potential problem occurs when combining exactly represented polyhedra: the user must take care to ensure that many spurious "microfeatures" do not appear in the resulting polyhedron. These might appear if the user intended a vertex to be ON a face when in fact is was a small distance OFF the face, perhaps making a small "bump" in the resulting polyhedron.

2.2 Representation and Model Methods

Algorithms in this category operate on approximate computer representations of objects. For example, a line can be represented by a line equation $ax + by = 1$, where the coefficients a and b are each allowed to vary within a small range. Thus an approximate representation of a line could denote a whole family of real lines. In Figure 2.2, the coefficients of the line ℓ define the shortest vector from the origin to the line (called the *normal*); these coefficients are allowed to vary such that the head of the vector lies in the small square region. This representation denotes the family of lines that lie within the shaded region.

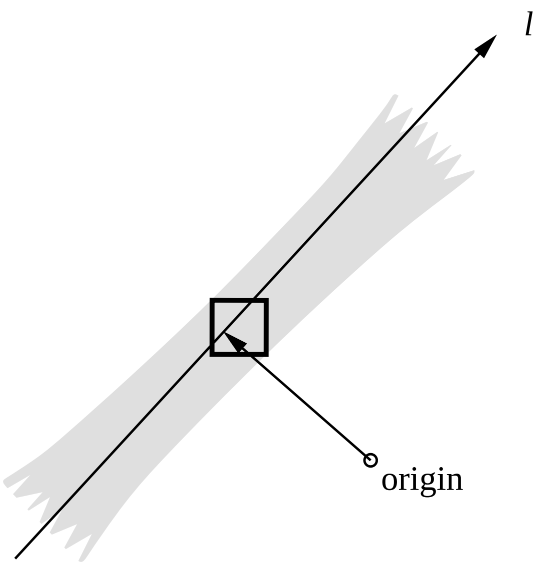

Figure 2.2: An Approximate Line Representation

At an abstract level, these algorithms operate as follows: given an approximate representation of the input object (or objects), the algorithm chooses one *real* object from the family of real objects denoted by the approximate representation, and computes on that real object. The real object is called the *model* and is never explicitly represented by the algorithm.

In reality, these algorithms make a series of decisions — and corresponding branches — that are consistent with one particular model M of the input. As long as ambiguous branches are decided in this consistent manner, the algorithm will not fail; it will make the same decisions as would an exact algorithm that operates on the exactly represented model M.

The difficulty in the implict model approach lies in proving that a model always exists which corresponds to the decisions of the algorithm. This is often a nontrivial task (as will be shown in Chapter 6) and is perhaps why the exact arithmetic approach is more popular with the implementors of geometric algorithms. Chapter 6 will provide a rigorous, yet tractable, alternative to the representation and model method which should be of interest to implementors.

2.2.1 Hoffmann, Hopcroft, and Karasick

Hoffmann, Hopcroft, and Karasick [17] introduce the representation and model method. They define an algorithm to be *correct* (or robust, in the terminology of this dissertation) if it computes a mapping from input representations R_i to output representations R_o such that there exist corresponding models M_i and M_o for which this is the correct mapping.

The problem of intersecting two polygons is considered, and it is shown that any vertex and edge that are "close" in the polygon representations can be considered coincident in the model polygons, for there will always exist some *real* model polygons in which all such incidences are realized. The model polygons are not necessarily simple, nor are their edges necessarily close to the edges of the polygon representation. Thus, the intersection algorithm does not guarantee an *a priori* bound on the perturbation of the model.

The problem of simultaneously intersecting *three* polygons is considered, and it is shown that this is sufficiently hard that it requires a thoerem prover for projective geometry. It is also shown that the problem of arranging the vertices of two polyhedra is at least as difficult as that of simultaneously intersecting three polygons.

2.2.2 Hopcroft and Kahn

Hopcroft and Kahn [19] present the first provably robust algorithm for intersecting an approximately represented convex polyhedron with a halfspace. Any vertices of the polyhedron that are close to the plane defining the halfspace are placed ON the plane. It is proven that there always exists a real polyhedron with the same incidences; however, this real polyhedron could be arbitrarily distant from the polyhedron representation.

This result could be used to build an algorithm to intersect two approximately represented convex polyhedra. However, the algorithm would not necessarily be robust in the representation and model sense (see the discussion in [19]).

2.2.3 Milenkovic

Milenkovic [25,26] presents a provably robust algorithm that computes the arrangement of a set of pseudolines[2]. The input consists of a set of lines with P-bit coefficients. The algorithm computes the arrangement of a set of implicit pseudolines that lie within a distance 2^{-P} of the lines. An important feature of the algorithm is that only $P + 5$ bits are required for intermediate computations.

[2] An *arrangement* of curves consists of a set of cells in the plane and the ordering of the curves around the boundary of each cell. *Pseudolines* are curves with the properties that two pseudolines intersect at most once, and a pseudoline never intersects itself.

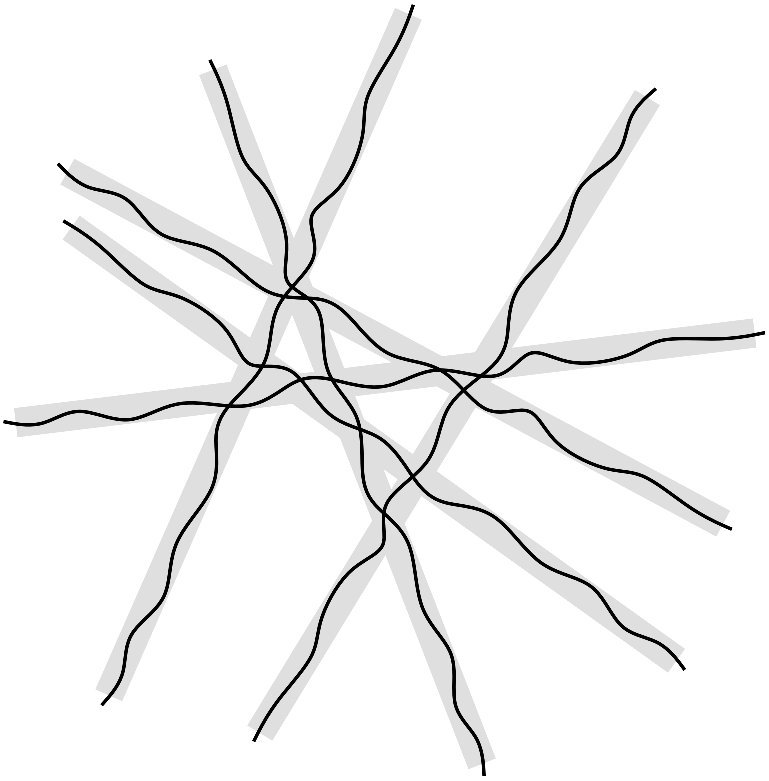

Figure 2.3: An Arrangement of Pseudolines Within Fixed Bounds

In later work [28], Milenkovic extends this to a more general approach called *double precision geometry*. This approach can be used to make robust any algorithm that calculates line and segment arrangements, as long as the algorithm can handle an arrangement of monotonic pseudolines instead of an arrangement of lines. The computed arrangement differs from the real line arrangement only in areas where vertices are close. For input lines that are represented with P-bit coefficients, the algorithm uses $2P + 5$ bits for intermediate computations.

In other work [27], this is extended to computing arrangements of planar curves, instead of pseudolines. Given a set of curves represented with P-bit coefficients, the arrangement of a set of implicit curves is instead computed. Each implicit curve $\tilde{\gamma}$ is guaranteed to lie within 2^{-P} of the real curve γ, is guaranteed to intersect the real curve whenever there is a sign change in the first derivative γ', and is guaranteed to be monotonic on the interval between intersections. The intermediate computations are performed with $(P+2\log D+c)$-bit precision, where D is the algebraic degree of the real curves and c is a small constant.

2.2.4 Li and Milenkovic

Li and Milenkovic [24] consider the problem of finding strongly convex hulls, which remain convex even when the vertices of the hull are perturbed slightly. Define an *ε-convex δ-hull* of a set of points S as a convex polygon P whose vertices are points in S, such that no point in S is more than δ outside of P, and P remains convex even if its vertices are arbitrarily perturbed by up to ε. Li and Milenkovic present the first robust algorithms that construct, given S and ε, an ε-convex $\mathcal{O}(\varepsilon)$-hull.

Two algorithms are given: an exact arithmetic algorithm which constructs the ε-convex 12ε-hull, and a rounded arithmetic algorithm which constructs a slightly less optimal ε-convex $(12\varepsilon + 288\sqrt{2}\mu)$-hull, where μ is the rounding error.

Note that building strongly convex hulls is a different issue from building robust algorithms. These convex hulls are a *tool* for building robust rounded arithmetic algorithms; such algorithms can rotate and translate a strongly convex hull without it becoming non-convex and without the previously enclosed points moving too far outside of it.

2.2.5 Fortune

Fortune [12] considers robustness in a manner similar to Hoffman, Hopcroft, and Karasick: an algorithm is *robust* if it always computes the solution to some real problem instance, and an algorithm is *stable* if the real problem instance is close to the given problem instance. This definition assumes exact input and inexact intermediate computations, although it could be easily modified for approximate input.

Fortunes presents two robust algorithms. The first computes the convex hull of an implicit set of n points perturbed by at most $\mathcal{O}(n\varepsilon)$ from a set of input points, where ε is approximately the rounding error. The second algorithm maintains a triangulation of an implicit set of n points which are perturbed at most $\mathcal{O}(n^2\varepsilon)$ from a set of input points.

2.2.6 Salesin, Stolfi, and Guibas

Salesin, Stolfi, and Guibas [34] and Salesin [33] present an approach to robustness called *epsilon geometry*, in which an algorithm computes on some perturbed version of the input. An interesting aspect of this work is that algorithms based upon epsilon geometry compute, dynamically, a bound on the perturbation.

Under this approach, an algorithm evaluates predicates $P(x)$ on the input x that compute, instead of TRUE or FALSE, an uncertainty interval $[l, h]$: no perturbation of x by less than distance l can make $P(x)$ true, and some perturbation of x by more than distance h can make $P(x)$ true (for some appropriate definition of "distance"). Predicates can be combined with the standard logical operations, and rules are given for combining their associated uncertainty intervals. Designing algorithms under this approach involves extensive manipulation of these uncertainty intervals.

Salesin [34] demonstrates this approach in implementing robust algorithms for point location in an approximately convex polygon[3], testing approximate and strong convexity of polygons, and constructing ε-convex δ-hulls of sets of points (similar to the convex hulls of Li and Milenkovic [24]).

[3] A polygon is *approximately convex* if there exists a perturbation of its vertices by up to some amount ε that makes it convex.

2.3 Topological Consistency Methods

Algorithms in this category maintain certain topological properties of the objects that they manipulate. For example, a topologically consistent line arrangement algorithm would never decide that two lines intersect more than once, and a topologically consistent Voronoi diagram algorithm would ensure that the resulting Voronoi diagram has a planar embedding.

In general, algorithms in this category deal with much harder problems than those in the other two categories. Algorithms in this category are usually *not* proved robust in the representation and model sense.

Karasick

Karasick [22] presents a polyhedral intersection algorithm that incorporates some *ad hoc* robustness rules. For example, one such rule states that a vertex of v one polyhedron cannot lie ON a face f of another polyhedron if the vertex is on a line ℓ that intersects that face at a distinct point $\ell \cap f \neq v$. This type of rule could prevent the inconsistent decisions that might be made in Figure 2.4 (as discussed in Section 1.3).

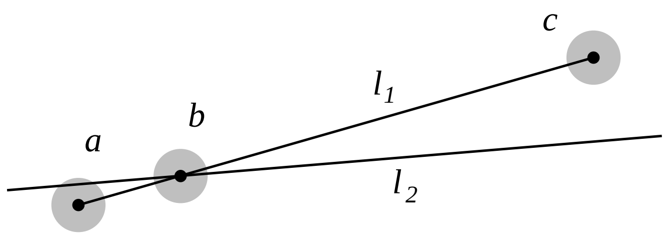

Figure 2.4: Preventable inconsistent decisions

A concise description of this work can be found in Hoffman, Hopcroft, and Karasick [18] (not to be confused with [17]). That paper also contains a discussion of the philosophy behind Karasick's topological consistency approach.

Two empirical tests were performed on Karasick's intersection algorithm. The first involved repetitively intersecting an object with a rotated version of itself: each intersection approximately doubled the number of faces in the object. Given an initial cube, Karasick's algorithm succeeded in performing twelve iterations before the machine crashed from lack of memory, resulting in a polyhedral approximation to a sphere which contained 3034 faces.

The second test involved intersecting two superimposed cubes, one slightly rotated from the other, and finding the range of angles in which the algorithm failed. Karasick's algorithm was found to have a range of failure of width anywhere from 10^{-2} to 10^{-10} degrees, depending upon the size of the regions of uncertainty around each polyhedral feature. For typical nonrobust algorithms, this range of failure is much greater: it is about half a degree [23].

While Karasick's intersection algorithm is not proved robust, these tests show that the approach of ensuring topological consistency dramatically increases the reliability of his algorithm.

2.3.1 Segal and Séquin

Segal and Séquin [37] develop a polyhedral intersection algorithm that maintains a region of uncertainty around each feature of the polyhedra. When features are close, they are merged and the regions of uncertainty are grown. If these regions grow too large a "tolerance error" is signalled to the user. Also, if a test about a vertex/plane relationship is required and tolerance regions of the vertex and the plane overlap, then a "tolerance error" is signalled and an arbitrary decision is made.

In a slightly more rigorous approach, Segal [35] incorporates four robustness criteria into his polyhedral intersection algorithm. When the algorithm discovers a configuration of features that contradicts the criteria, it backtracks, undoing decisions based on numeric data, until the contradiction is eliminated. No proof of robustness is given. However, a cube/cube test (similar to that of Karasick) reveals *no* region of error in Segal's algorithm.

2.3.2 Brüderlin

Brüderlin builds a polyhedral intersection algorithm that maintains a region of uncertainty around each feature [1,2]. In a similar manner to Segal and Séquin [37], his algorithm decides that close features are coincident, and merges their corresponding regions of uncertainty.

However, Brüderlin's algorithm keeps sufficient information about the regions of uncertainty to determine when its decisions to merge close features are potentially inconsistent. It is proven that if a potentially inconsistency is *not* detected, then there exists a model (in the "representation and model" sense) which corresponds to the decisions of the algorithm. However, if a potential inconsistency *is* detected, then the algorithm must be re-executed with smaller regions of uncertainty (and possibly more precision). Brüderlin states that, in practice, the algorithm rarely has to be run more than twice.

2.3.3 Sugihara and Iri

The philosophy of Sugihara and Iri is to eliminate redundant numeric tests. Their algorithms are not proved robust in the representation and model sense, but empirical tests show them to be immune from numerical error [41].

Sugihara and Iri [39,41] present a Voronoi diagram algorithm that is guaranteed to run to completion and to output a planar graph *even if* every numeric test is decided randomly (simulating 0-bit arithmetic). Obviously, the output in such a case is not too meaningful, but as the precision of the intermediate computations increases, the output of the algorithm converges to the true Voronoi diagram.

Sugihara, Ooishi, and Imai [42] extend this approach to a divide-and-conquer Voronoi diagram algorithm, and to an incremental algorithm for computing the Voronoi diagram of a collection of line segments. Sugihara [38] uses the incremental Voronoi diagram algorithm of Sugihara and Iri [39] to robustly compute all intersections among a set of line segments.

2.4 Other Related Work

This section briefly reviews some other work which is related to robustness.

2.4.1 Backtracking

Dobkin and Silver [7] demonstrate how numeric error can accumulate in cascaded computations, where the output of one computation is used as input by the next.

A "permutation/perturbation method" of avoiding numeric failure is suggested by Dobkin and Silver. Using this approach, the algorithm is run with various permutations of its numeric code and the results of the permutations are compared to find an average, empirical error in the computations. When the error becomes too large, the algorithm increases arithmetic precision and backtracks to some earlier point.

This is not applicable to polyhedral intersection, where there is no degredation of numeric information between the input and the output, since the faces of the polyhedron of intersection are a subset of the faces of the two polyhedra that were intersected.

2.4.2 Eliminating Degeneracy

Edelsbrunner and Mücke [10] present a technique — called "simulation of simplicity" — by which an algorithm can avoid considering degenerate cases of the input. A similar technique was simultaneously presented by Yap [45].

Using simulation of simplicity, the input, which is assumed to be exactly represented, is perturbed infinitesimally so that every numeric sign test which might return $+1$, -1, or 0 instead only returns $+1$ or -1. This perturbation is accomplished by a "black box" and is transparant to the algorithm.

This technique is particularly useful, since a large amount of the time spent on algorithm design is usually spent considering the degenerate cases of the input. For example, a Voronoi diagram algorithm that uses this approach could ignore the special case in which four points lie on the same circle.

However, this approach is not applicable to solid modeling, in which degenerate cases are often necessary: consider that four vertices on the same face are in a degenerate configuration.

Chapter 3

Robustness

This chapter introduces a theory of robustness based upon the representation and model approach of Hoffmann, Hopcroft, and Karasick [17].

3.1 Formalizing Our Notions

Consider the problem of multiplying two real numbers. There are three components involved: the pair of real numbers to be multiplied (called the *problem instance*), the real number that is the product (called the *problem solution*), and the mapping from instances to solutions (simply called the *problem*). The relationship among the three components is

$$\mathcal{P} : \mathcal{I} \to \mathcal{S}.$$

An *algorithm* is a method by which to compute the problem mapping. When an algorithm is executed on a computer it must use *representations* for both the problem instance and the problem solution, since the computer cannot operate on the abstract mathematical entities that are the real instance and real solution. An algorithm maps representations of problem instances onto representations of problem solutions,

$$A : \mathcal{I}_{rep} \to \mathcal{S}_{rep}.$$

For a given problem and algorithm, there are natural one-to-one mappings between the abstract entities and their corresponding representations (one-to-many mappings are discussed in the next section). For example, the computer representation of an integer maps onto a real, mathematical integer. The mappings that concern us are

$$\begin{aligned} \text{MODEL}_{\mathcal{I}} :&\ \ \mathcal{I}_{rep} \to \mathcal{I}, \ \text{ and} \\ \text{REP}_{\mathcal{S}} :&\ \ \mathcal{S} \to \mathcal{S}_{rep}. \end{aligned}$$

This is summarized in Figure 3.1.

An algorithm A for a problem $\mathcal{P}$ is *correct* if the diagram of Figure 3.1 commutes; that is, if for all $x_{rep} \in \mathcal{I}_{rep}$,

$$A(x_{rep}) = \text{REP}_{\mathcal{S}}(\ \mathcal{P}(\ \text{MODEL}_{\mathcal{I}}(x_{rep})\ )\ ).$$

However, this definition does not apply to algorithms that operate on approximate problem instances, in which the $\text{MODEL}_{\mathcal{I}}$ mapping is one-to-many.

$$\begin{array}{ccccc} & \mathcal{I} & \xrightarrow{\mathcal{P}} & \mathcal{S} & \\ \textsc{model}_{\mathcal{I}} & \uparrow & & \downarrow & \textsc{rep}_{\mathcal{S}} \\ & \mathcal{I}_{rep} & \xrightarrow{A} & \mathcal{S}_{rep} & \end{array}$$

Figure 3.1: Correctness

3.2 Robustness

There are two types of representations: *exact representations* denoting exactly one abstract entity and *approximate representations* denoting more that one abstract entity. For example, the binary encoding of an integer is an exact representation, whereas the floating point encoding of a real number is an approximate representation, since each floating point number can denote many real numbers.

An *exact algorithm* is one that operates on an exact problem instance, and an *approximate algorithm* is one that operates on an approximate problem instance. For a given approximate algorithm, there is a natural mapping from approximate problem instances onto sets of real problem instances,

$$\textsc{models}_{\mathcal{I}} : \mathcal{I}_{rep} \rightarrow 2^{\mathcal{I}}.$$

This is summarized in Figure 3.2, where the large arrow symbol indicates the one-to-many mapping of $\textsc{models}_{\mathcal{I}}$.

$$\begin{array}{ccccc} & \mathcal{I} & \xrightarrow{\mathcal{P}} & \mathcal{S} & \\ \textsc{models}_{\mathcal{I}} & \Uparrow & & \downarrow & \textsc{rep}_{\mathcal{S}} \\ & \mathcal{I}_{rep} & \xrightarrow{A} & \mathcal{S}_{rep} & \end{array}$$

Figure 3.2: Robustness

An algorithm A for a problem $\mathcal{P}$ is *robust* if, for every approximate problem instance $x_{rep} \in \mathcal{I}_{rep}$, there is a real problem instance $x \in \textsc{models}_{\mathcal{I}}(x_{rep})$ such that

$$A(x_{rep}) = \textsc{rep}_{\mathcal{S}}(\ \mathcal{P}(x)\ ).$$

For example, consider an approximate algorithm that determines whether two real numbers, represented as floating point numbers, are equal. If the two real numbers are close enough, then their representations will be identical floating point numbers. An algorithm for this problem would be considered robust if it outputs *either* "equal" or "not equal" when the two floating point representations are identical. For either response, there exist two real numbers (corresponding to the approximate floating point numbers) for which the response is correct, and hence the algorithm is robust by the above definition.

3.3 Consistency

The point location algorithm of Chapter 4 requires that the mappings of Figure 3.2 have a certain property called "consistency". For mappings with this property, each approximate problem instance maps to exactly one problem solution. More formally, mappings $\text{MODELS}_{\mathcal{I}}$, $\mathcal{P}$, and $\text{REP}_{\mathcal{S}}$ are *consistent* if

$$\forall\ x_{rep} \in \mathcal{I}_{rep},\ \forall\ x_1, x_2 \in \text{MODELS}_{\mathcal{I}}(x_{rep}),$$

$$\text{REP}_{\mathcal{S}}(\ \mathcal{P}(x_1)\ ) = \text{REP}_{\mathcal{S}}(\ \mathcal{P}(x_2)\ ).$$

For example, the mappings used in the problem of comparing two real numbers are not consistent; certain approximate problem instances can map to either of two solutions: "equal" or "not equal". However, the mappings used in the point location algorithm of Chapter 4 are consistent; in that chapter it will be shown that the consistency property is necessary to solve the point location problem.

3.4 Summary

The following three concepts have been defined in this chapter:

- An exact algorithm is *correct* if it computes the correct mapping.
- An approximate algorithm is *robust* if, given an approximate problem instance, the algorithm computes the correct mapping for any one of the corresponding real problem instances.
- A problem is *consistent* if, for each approximate problem instance, all corresponding real problem instances have the same solution.

This chapter's definition of robustness is similar to the definition of Fortune [12]. The representation and model approach is similar to Milenkovic's hidden variable approach [25], which he uses to construct arrangements of pseudolines that are constrained to lie within strips of fixed width. The strips can be considered as an approximate representation of the problem instance and the pseudolines as a model of the problem instance. Milenkovic's pseudoline arrangement algorithm can then be said to be provably robust in the sense of this chapter. It is interesting to note that, as with other algorithms of the representation and model variety, Milenkovic's algorithm never explicitly constructs an exact representation of the problem instance.

Chapter 4

Robust Point Location In Approximate Polygons

I hate definitions. – Benjamin Disraeli

Essential to the task of polyhedral intersection is an algorithm that determines whether a point lies INSIDE or OUTSIDE a polygon. This algorithm is used to determine whether an edge of one polyhedron intersects a face of another. If the point of intersection between the edge and the plane of the face is on the interior of the polygonal face boundary, then, since it is common to both polyhedra, it is one of the vertices of the polyhedron of intersection. This chapter presents a robust point location algorithm.

4.1 Assumptions

We will assume that polyhedra must be represented approximately, since polyhedra usually have overconstrained faces and vertices, and the *exact* locations of the vertices, edges, and faces of the polyhedra might require a very large number of bits to represent. Thus, we will represent the polygonal face boundary with an approximate polygon, as described in the next section.

We will also assume that the location of the point on the plane of the face is completely unknown, since this may be true in ill conditioned cases in which the intersecting edge lies very close to the plane of the face.

Since the location of the point is unknown, the algorithm must be provided with some additional information. This additional information consists of the *signature* of the point; that is, its position (LEFT or RIGHT) with respect to each directed edge of the boundary (the signature will be formally defined below). We will assume that the point does not lie ON any edge of the polygon.

It is a reasonable assumption that the signature is available since it is often derivable from logical information available to the intersection algorithm (for example, see Karasick's modeler [22]). The intersection algorithm that is developed in Chapter 9 can derive the signature.

4.2 Approximate Polygons

An "approximate polygon" is an approximate representation of some real polygon; an example is shown in Figure 4.1. The remainder of this section formally defines the approximate polygon.

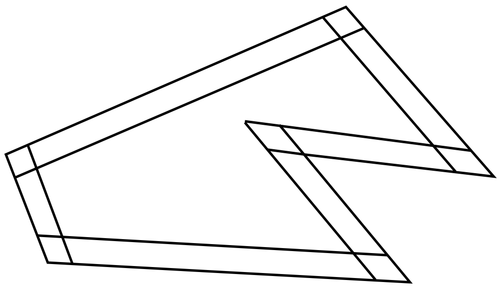

Figure 4.1: An Approximate Polygon

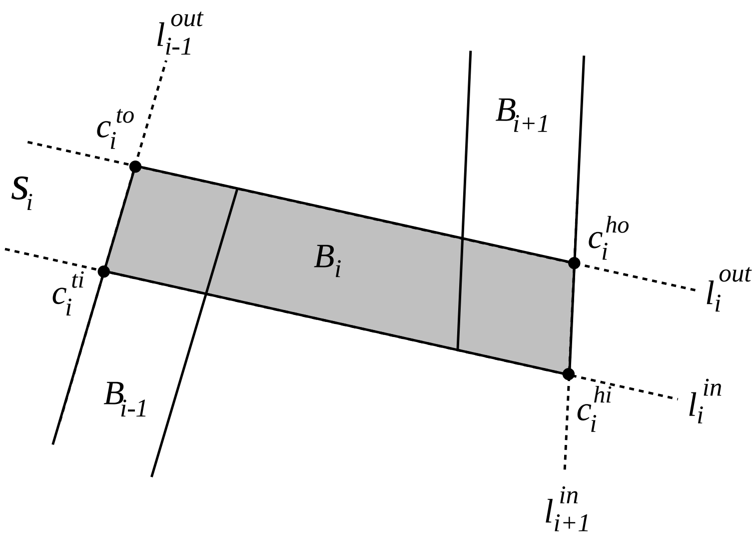

Figure 4.2: A Band

Define a *swath* S_i to be the region between two lines ℓ_i^{out} and ℓ_i^{in}, such that for all v, $\ell_i^{in}(v) \geq \ell_i^{out}(v)$. The line ℓ_i^{out} will always be chosen to lie on the exterior of the approximate polygon. These restrictions causes the lines to be parallel and defines the swath between them as

$$S_i = \{v \mid \ell_i^{in}(v) \geq 0 \ \wedge \ \ell_i^{out}(v) \leq 0\}.$$

Refer to Figure 4.2. Given an ordered list of swaths $(S_1, S_2, \ldots, S_n)$, define a *band* B_i as

$$B_i = S_i \cap E_i^{i-1} \cap E_i^{i+1},$$

where

$$E_i^j = \begin{cases} \{v \mid l_j^{out}(v) \leq 0\} & \text{if corner } i/j \text{ is convex} \\ \{v \mid l_j^{in}(v) \geq 0\} & \text{if corner } i/j \text{ is reflex.} \end{cases}$$

Define the corners of a band B_i as

$$\begin{aligned} c^{ti} &= l_i^{in} \cap L_i^{i-1}, \\ c^{hi} &= l_i^{in} \cap L_i^{i+1}, \\ c^{to} &= l_i^{out} \cap L_i^{i-1}, \text{and} \\ c^{ho} &= l_i^{out} \cap L_i^{i+1}, \end{aligned}$$

where

$$L_i^j = \begin{cases} l_j^{out} & \text{if corner } i/j \text{ is convex} \\ l_j^{in} & \text{if corner } i/j \text{ is reflex.} \end{cases}$$

Define an *approximate polygon* as an ordered list of bands,

$$P_{rep} \stackrel{\text{def}}{=} (B_1, B_2, \ldots, B_n),$$

with the property that $B_i \cap B_j = \emptyset$ if $|i - j| > 1$, and $B_i \cap B_{i+1} = S_i \cap S_{i+1}$, where S_i is the swath corresponding to band B_i.

Define the mapping $\text{MODELS}_{\mathcal{I}}$ such that $P \in \text{MODELS}_{\mathcal{I}}(P_{rep})$ if and only if P and P_{rep} meet the following conditions:

1. There is a one-to-one correspondence between the lines ℓ_i defined by the edges of P and the swaths S_i defined by the bands of P_{rep}. Assume that ℓ_i corresponds to S_i.

2. Each line $\ell_i(v) = 0$ must lie between the corners of band B_i. That is, it must satisfy the following four constraints (see Figure 4.2):

$$\ell_i(c^{ti}) \leq 0, \;\; \ell_i(c^{to}) \geq 0,$$

$$\ell_i(c^{hi}) \leq 0, \;\; \ell_i(c^{ho}) \geq 0.$$

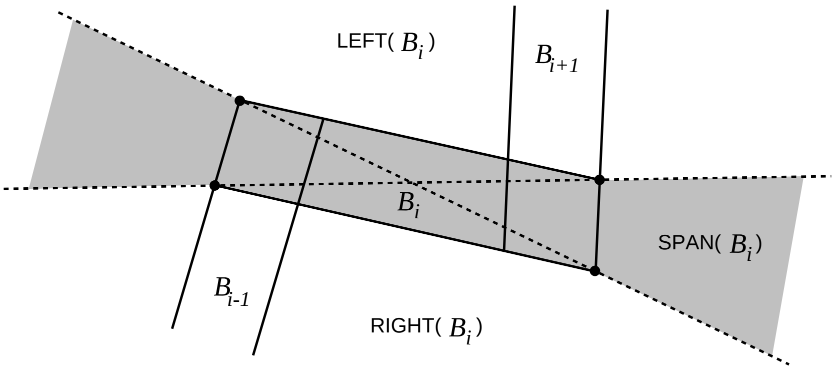

Figure 4.3: The SPAN of a Band

Refer to Figure 4.3. The *span* of a band is the set of points swept out by all lines which fit within the band; the *left* and *right* of a band are the set of those points to the left and right of the span. By convention, the bands are *directed* such that the interior of the approximate polygon is to the right of each band.

Table 4.1: Properties of Approximate Polygons

1. An approximate polygon is closed and simple.
2. Edge e_i of the real polygon lies completely within B_i.
3. Edges e_i and e_{i+1} of the real polygon intersect within $B_i \cap B_{i+1}$.
4. All models of an approximate polygon are simple.
5. All models of an approximate polygon have the same sinuosity (that is, the same sequence of left and right turns).
6. $\textsc{span}(B_i) \cap \textsc{span}(B_{i+1}) = B_i \cap B_{i+1}$.
7. If the point v is not in $\textsc{span}(B_i)$, then in all models, v lies to the same side of e_i.
8. If the point v is in $\textsc{span}(B_{i+1}) - B_i$, then in all models v lies to the same side of e_i.

4.2.1 Properties of Approximate Polygons

As a final note, some useful properties of approximate polygons are enumerated in Table 4.1. These properties are evident after a moment's consideration.

4.3 The Formal Problem

Define a *polygon* $P = (e_1, e_2, \ldots, e_n)$ as a set of directed edges, e_i, with the properties that consecutive edges intersect at one endpoint, nonconsecutive edges do not intersect, and the interior of the polygon lies to the RIGHT of each directed edge. Let ℓ_i be the directed line defined by edge e_i.

Define a *signature* as a string in $(\mathrm{L}|\mathrm{R})^n$. Let α_i denote the i^{th} component of α. A point v is said to *satisfy* a signature α with respect to a polygon $P = (e_1, e_2, \ldots, e_n)$ if, for all i in $1 \ldots n$,

- $\alpha_i = \mathrm{R}$ if v is RIGHT of ℓ_i, and
- $\alpha_i = \mathrm{L}$ if v is LEFT of ℓ_i.

Let P be a polygon and let α be a signature. Recalling the theory of robustness presented in Chapter 3, we define the POINT-LOCATION problem to be a mapping from (P, α) pairs onto the abstract concepts of INSIDE and OUTSIDE, such that

$$\text{POINT-LOCATION}(P, \alpha) = \begin{cases} \text{INSIDE} & \text{if } \exists\ v \text{ INSIDE } P \text{ satisfying } \alpha \\ \text{OUTSIDE} & \text{if } \exists\ v \text{ OUTSIDE } P \text{ satisfying } \alpha \\ \text{UNDEFINED} & \text{otherwise.} \end{cases}$$

Let us represent real polygons P with approximate polygons P_{rep}. Then a point location algorithm computes the mapping from (P_{rep}, α) pairs onto the computer outputs of `inside` and `outside`. However, we must consider the following question:

> Given any approximate polygon P_{rep} and any signature α, is it true that, for every pair of model polygons $P_1, P_2 \in P_{rep}$,
>
> $$\text{POINT-LOCATION}(P_1, \alpha) \ = \ \text{POINT-LOCATION}(P_2, \alpha)?$$

If the answer to this question were negative then the point location problem would not be *consistent* in the formal sense of Chapter 3; for some inputs of (P_{rep}, α) the algorithm could robustly output *either* `inside` or `outside`. Such an algorithm would not be very useful in a solid modeler. However, the Uniqueness Theorem which is presented later in the chapter proves that the point location problem *is* consistent.

4.4 The Point Location Lemmas

Refer to Figure 4.4 for the following. A *halfregion* is similar to a halfspace in $\Re^2$, except that it has a polygonal boundary. The following halfregions R_i and L_i consist of those points which, in *at least one* model $P \in \text{MODELS}(P_{rep})$, are either ON e_i or to the RIGHT or LEFT of e_i, respectively, in that model. Given α and P_{rep}, the halfregion H_i is that region in whose interior a point must lie if it is to satisfy α_i. The *interior* of the cell $\tilde{C}_\alpha$ consists of those points which satisfy α in *at least one* model.

$$R_i = \text{SPAN}(B_i) \cup \text{RIGHT}(B_i)$$

$$L_i = \text{SPAN}(B_i) \cup \text{LEFT}(B_i)$$

$$H_i = \begin{cases} R_i & \text{if } \alpha_i = \text{R} \\ L_i & \text{if } \alpha_i = \text{L} \end{cases}$$

$$\tilde{C}_\alpha = \bigcap_{i=1}^{n} H_i$$

The next two lemmas will be used to construct the point location algorithm. The first lemma shows that for each point in $\tilde{C}_\alpha$ there exists some model in which the point has signature α; the second lemma shows how to determine whether the point is INSIDE or OUTSIDE that model.

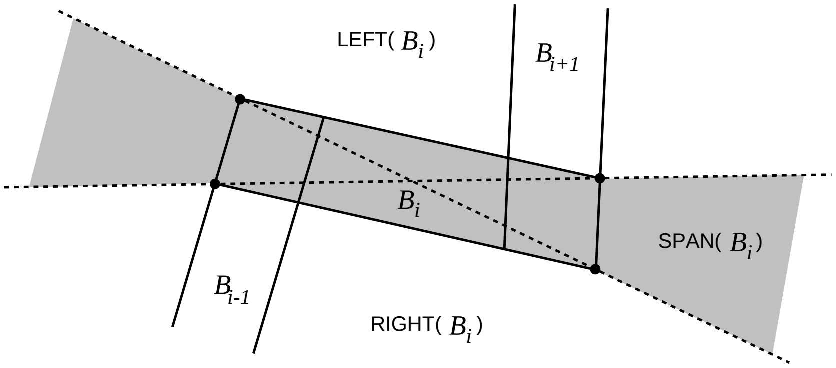

Figure 4.4: The SPAN of a Band

Lemma 1 (Model Existence)

Given an approximate polygon P_{rep} and a signature α, construct $\tilde{C}_\alpha$ as described above. Then for each point v on the interior of $\tilde{C}_\alpha$, there exists some model $P \in$ MODELS(P_{rep}) such that v satisfies α with respect to P.

Proof Since $v \in \tilde{C}_\alpha$, for each i, $v \in H_i$ and there is some line ℓ_i that lies between the corners of the band B_i, and that has v to the side specified by α_i. These lines intersect to form the edges of a model polygon P in which v satisfies α. □

Lemma 2 (Point Location)

Given an approximate polygon P_{rep}, a model polygon $P \in$ MODELS(P_{rep}), and a point v which has a signature α with respect to P, the following are true:

1. *If v is strictly to the interior of P_{rep} (that is, it does not lie on any band B_i) then v INSIDE P.*
2. *If v is strictly to the exterior of P_{rep} then v OUTSIDE P.*
3. *If $v \in B_i$, but $v \notin B_{i\pm1}$, then v INSIDE P* iff *$\alpha_i =$ R.*
4. *If $v \in B_i \cap B_{i+1}$ and the $i/i+1$ corner is* convex, *then v INSIDE P* iff *$\alpha_i =$ R and $\alpha_{i+1} =$ R.*
5. *If $v \in B_i \cap B_{i+1}$ and the $i/i+1$ corner is* reflex, *then v INSIDE P* iff *$\alpha_i =$ R or $\alpha_{i+1} =$ R.*

Proof In Figure 4.5 the cases 1 through 5 are demonstrated by the points x_1 through x_5, respectively. □

Given the Model Existence Lemma and the Point Location Lemma, a point location algorithm can be developed. This algorithm will construct the region $\tilde{C}_\alpha$, pick a point from its interior, and apply the rules of the Point Location Lemma to determine whether the point is INSIDE or OUTSIDE the model in which it satisfies signature α.

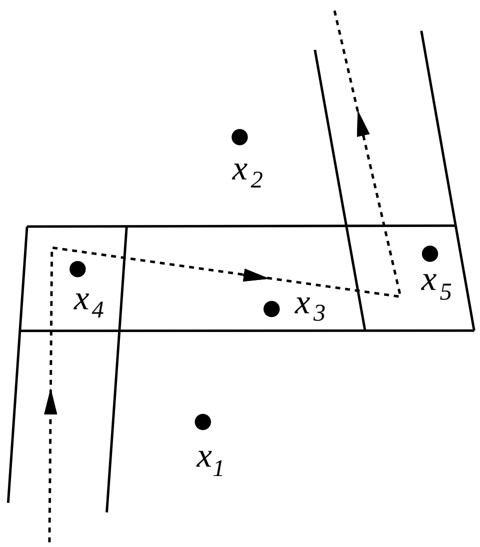

Figure 4.5: Cases for the Point Location Lemma

4.5 The Point Location Algorithm

The Model Existence Lemma, Point Location Lemma, and Uniqueness Theorem combine to form the following point location algorithm. Note that the algorithm is quite simple and never explicitly constructs the model polygon.

1. Compute $\tilde{C}_\alpha$.
2. If $\tilde{C}_\alpha = \emptyset$ then no model of P_{rep} induces a cell with signature α.
3. Pick a point w on the interior of $\tilde{C}_\alpha$.
4. Apply the Point Location Lemma to determine whether w is INSIDE or OUTSIDE of the models in which it has signature α.

Figure 4.6: Point Location Algorithm

Lemma 3 (Robustness)

The point location algorithm is robust.

Proof This follows from the Model Existence Lemma and Point Location Lemma. □

Lemma 4 (Consistency)

The approximate point location problem is consistent.

Proof This follows from the Uniqueness Theorem presented in the next section. □

Lemma 5 (Complexity)

The point location algorithm has time complexity $\mathcal{O}(n^2)$.

Proof
Step 1 of the algorithm can be accomplished by computing the arrangement of halfregions in $\mathcal{O}(n^2)$ time. This is done by computing the arrangement of the $3n$ lines which define the n halfregions H_i, then joining adjacent cells which are separated by a line segment which is *not* part of the boundary of some H_i. Those cells separated by a line segment which *is* part of the boundary of some H_i will have signatures which differ in a single position, so the signature of each cell can be found in constant time.

The remaining steps of the algorithm take constant time. Step 3 is easily accomplished given the convex decomposition of $\tilde{C}_\alpha$ which is computed simultaneously with $\tilde{C}_\alpha$ itself. Thus, the overall running time is $\mathcal{O}(n^2)$. □

4.6 The Uniqueness Theorem

The Uniqueness Theorem presented in this section shows that the point location problem is consistent in the formal sense of Chapter 3. The reader can skip to the next section, and come back later if interest warrants.

Theorem 1 (Uniqueness)

Given an approximate polygon P_{rep} and a signature α, if for some *model polygon in* MODELS(P_{rep}) *there is a point that satisfies α and that is* INSIDE *the polygon, then, for* every *model polygon, all points that satisfy α with respect to that polygon are* INSIDE *that polygon (similarly for* OUTSIDE*).*

Proof (By Contradiction):
Let $\alpha^k(x)$ be the signature of point x in model P^k. Let e_i^k be edge e_i in model P^k. Then assume the following:

$$\exists P^1, P^2 \in \textsc{models}(P_{rep}), \;\; \exists u, v \in \Re^2,$$

$$u \textsc{ inside } P^1 \;\; \wedge \;\; v \textsc{ outside } P^2 \;\; \wedge \;\; \alpha^1(u) = \alpha^2(v).$$

The theorem is proved by showing that there is some edge e_k that has u to one side in all models and v to the other side in all models. This violates the assumption, since $\alpha_k^1(u) \neq \alpha_k^2(v)$.

Lemma 6 *One of u or v must lie within the boundary of P_{rep}.*

Proof Assume that neither u nor v is within the boundary. Then by the initial assumption and the Point Location Lemma u and v lie on opposite sides of the boundary. Say u is inside and v is outside. Then the segment $\overline{uv}$ must traverse both of the parallel sides of some band B_i, as shown in Figure 4.7. From the definition of the corners of B_i and the definition of $\text{SPAN}(B_i)$, u and v lie to different sides of $\text{SPAN}(B_i)$. Then by Property 7 u lies to one side of all models of P_{rep} and v lies to the other side. Thus, in the models P^1 and P^2, $\alpha_i^1(u) \neq \alpha_i^2(v)$ and the initial assumption is contradicted. □

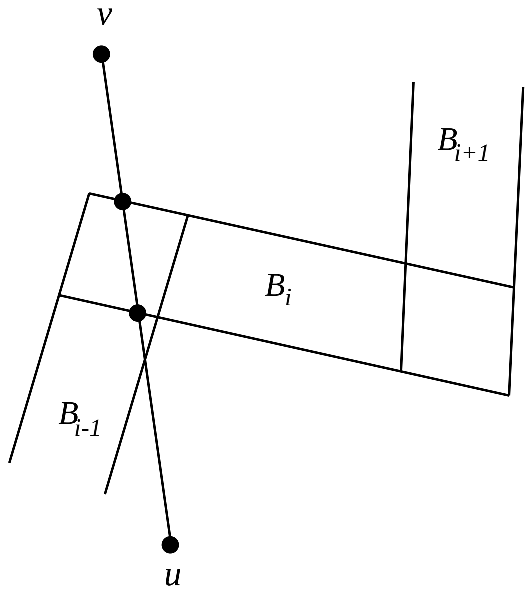

Figure 4.7: $\overline{uv}$ Crosses Some B_i

From Lemma 6 assume without loss of generality that u lies in the boundary of P_{rep}.

Lemma 7 *The point u lies in some band B_i such that $\alpha_i^1(u) = \text{R}$ and $\alpha_i^2(v) = \text{R}$.*

Proof u is INSIDE P^1 by the initial assumption and is in some band by Lemma 6. If $u \in B_k$ and $u \notin B_{k\pm1}$, then by the Point Location Lemma u is to the RIGHT of e_k^1. Choose $i = k$. If $u \in B_k \cap B_{k+1}$ then by the Point Location Lemma u is to the right of at least one of e_k^1 or e_{k+1}^1. Choose i to be k or $k+1$ to satisfy the lemma. Then by the initial assumption, $\alpha_i^1(u) = \text{R}$ means that $\alpha_i^2(v) = \text{R}$ also. □

Lemma 8 *On the segment $\overline{uv}$ there is some point $x \neq v$ that is inside P^2. Furthermore, e_i^2 does* not *intersect $\overline{uv}$ between x and v.*

Proof

Case 1: $u \in B_i$, but $u \notin B_{i\pm1}$.

If $\alpha_i^2(u) = \text{R}$, then u INSIDE P^2 (by the Point Location Lemma), so choose $x = u$ and we are done. Otherwise consider the case in which $\alpha_i^2(u) \neq \text{R}$.

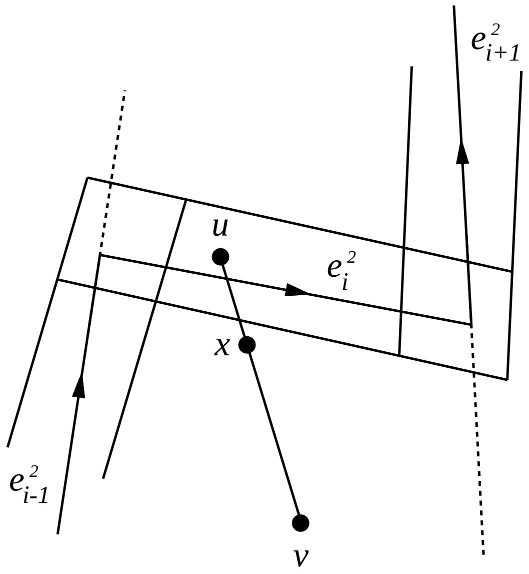

Figure 4.8: x Exists If $u \in B_i$

Refer to Figure 4.8. The point u lies in B_i and not in B_{i-1}, so by Property 8, in all models P^k, u is to the same side of e^k_{i-1}. In particular, $\alpha^1_{i-1}(u) = \alpha^2_{i-1}(u)$. By the initial assumption, $\alpha^1_{i-1}(u) = \alpha^2_{i-1}(v)$, so $\alpha^2_{i-1}(u) = \alpha^2_{i-1}(v)$. By a similar argument for e_{i+1}, $\alpha^2_{i+1}(u) = \alpha^2_{i+1}(v)$. Thus u and v lie between two lines touching the endpoints of e^2_i.

Since $\alpha^2_i(u) \neq \text{R}$ and since by Lemma 7 $\alpha^2_i(v) = \text{R}$, $\overline{uv}$ must cross the *line* defined by e^2_i. But u and v lie between the two lines touching the endpoints of e^2_i, so $\overline{uv}$ must cross the *edge* e^2_i. Since by Property 4 all models P are simple, there is a small neighborhood to the right of e^2_i which contains only points interior to P. The segment $\overline{uv}$ passes through this neighborhood, so there is some point $x \neq v$ on $\overline{uv}$ that is INSIDE P^2, and $\overline{xv} \cap e^2_i = \emptyset$.

Case 2: $u \in B_i \cap B_{i+1}$.

If u INSIDE P^2 then choose $x = u$ and we are done. Otherwise, consider Figure 4.9. By an argument similar to Case 1, u and v must lie to the same side of e^k_{i-1} in all models P^k, and must lie to the same side of e^k_{i+2} in all models P^k. If u OUTSIDE P^2 then the edges e^2_i and e^2_{i+1} must separate u and v. Thus the segment $\overline{uv}$ must cross either e^2_i or e^2_{i+1}, and in doing so must pass through a neighborhood of interior points to the right of the edge that it crosses. Therefore there is some point $x \neq v$ on $\overline{uv}$ that is INSIDE P^2, and $\overline{xv} \cap e^2_i = \emptyset$. □

Lemma 9 *In P^2 there is some edge e^2_j that crosses $\overline{uv}$ such that $\alpha^2_j(u) = \text{R}$ and $\alpha^2_j(v) = \text{L}$. Furthermore, e^2_j can be chosen such that no other e^2_k crosses $\overline{uv}$ between e^2_j and v.*

Proof From Lemma 8, $\overline{uv}$ contains a point x that is INSIDE P^2, and by the initial assumption, v OUTSIDE P^2. From the Jordan curve theorem we know that $\overline{xv}$ must cross the boundary of P^2 on some edge e^2_j with x to the inside (right) of e^2_j and v to the outside (left) of e^2_j. From the ordering of points along $\overline{uv}$ ($u \leq x < v$), if $\alpha^2_j(x) = \text{R}$ then $\alpha^2_j(u) = \text{R}$ also. If there are many candidates for e^2_j, choose that which is closest to v to satisfy the second part of the lemma. □

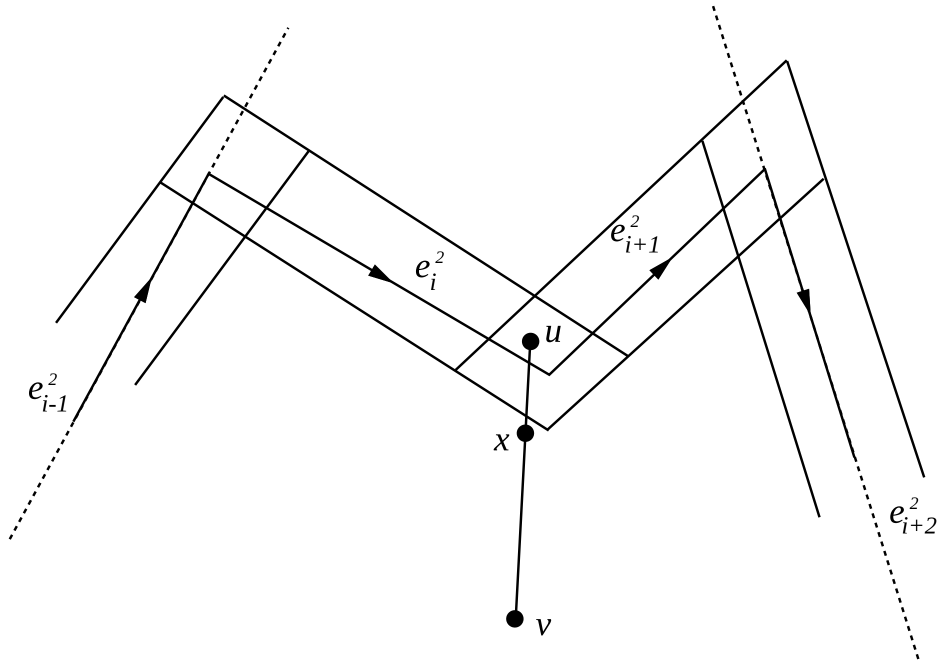

Figure 4.9: x Exists If $u \in B_i \cap B_{i+1}$

Lemma 10 $B_j \cap B_i = \emptyset$

Proof
If $|i-j| > 1$ then by the definition of an approximate polygon $B_i \cap B_j = \emptyset$. So we only have to consider $|i-j| \leq 1$. But by Lemma 9 e_j^2 intersects $\overline{uv}$ between x and v, and by Lemma 8 e_i^2 does *not* intersect $\overline{uv}$ between x and v. So $e_i^2 \neq e_j^2$ and hence $i \neq j$.

Assume that $j = i-1$. Then, by the initial assumption, $\alpha_j^2(v) = \text{L}$ means that $\alpha_j^1(u) = \text{L}$. By Lemma 9, $\alpha_j^2(u) = \text{R}$. Since u is to different sides of e_j in P^1 and P^2, u must lie in $\text{SPAN}(B_j)$. Since u also lies in B_i, by Property 6 u lies in the corner $B_i \cap B_j$.

Suppose corner i/j is convex. Since u INSIDE P^1, by the Point Location Lemma $\alpha_i^1(u) = \text{R}$ and $\alpha_j^1(u) = \text{R}$. But, by the initial assumption, $\alpha_j^1(u) = \text{R}$ means that $\alpha_j^2(v) = \text{R}$, contradicting Lemma 9. So corner i/j is not convex.

Suppose corner i/j is reflex. Refer to Figure 4.10. By Lemma 9, $\alpha_j^2(u) = \text{R}$ and $\alpha_j^2(v) = \text{L}$, and by Lemma 7, $\alpha_i^2(v) = \text{R}$. But if $\overline{uv}$ is to intersect e_j^2 then it must also intersect e_i^2 closer to v, as shown in the Figure. This contradicts Lemma 9, which states the e_j^2 is the closest intersection to v. So corner i/j is not reflex.

Thus the assumption is false that $j \neq i-1$. By a similar argument $j \neq i+1$. Therefore $B_j \cap B_i = \emptyset$. $\square$

Lemma 11 $u \in \text{SPAN}(B_j) - B_j$

Proof By Lemma 9, $\alpha_j^2(u) = \text{R}$ and $\alpha_j^2(v) = \text{L}$. By the initial assumption, $\alpha_j^2(v) = \text{L}$ means that $\alpha_j^1(u) = \text{L}$. Since $\alpha_j(u)$ is different in models P^1 and P^2, u must lie in $\text{SPAN}(B_j)$. From Lemma 10, u cannot lie in B_j since it already lies in B_i. $\square$

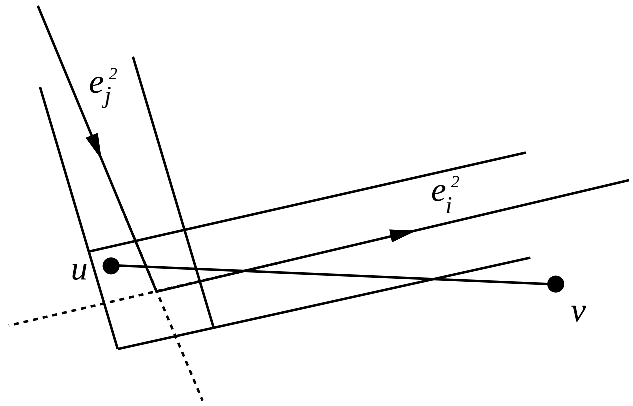

Figure 4.10: $\overline{uv}$ Must Intersect e_i^2 Closer To v

Lemma 12 *Define k such that $B_j \cap B_k$ is closest to u. Then u and v are on opposite sides of e_k^2.*

Proof Refer to Figure 4.11. Note that $k = j \pm 1$, otherwise B_j and B_k wouldn't intersect at all. In any model P the point u and the edge e_j are on opposite sides of the line defined by e_k ($u \in \text{SPAN}(B_j) - B_j$ by Lemma 11, and since B_k is closest to u, it separates u from the rest of B_j, which contains e_j). Thus $\overline{uv}$ must intersect e_j^2 at some point z to the side of e_k^2 that is opposite to u. From the ordering of points along $\overline{uv}$ ($u < z < v$), v must also be opposite to u. □

Lemma 13 $\alpha_k^1(u) = \alpha_k^2(u)$.

Proof By Lemma 12, $u \in \text{SPAN}(B_j) - B_j$. By Property 6, $\text{SPAN}(B_j) \cap \text{SPAN}(B_k) = B_j \cap B_k$, so with a bit of algebra we can conclude that $u \notin \text{SPAN}(B_k)$. Then by Property 7, $\alpha_k^1(u) = \alpha_k^2(u)$. □

To conclude the theorem, by Lemma 12 there is some edge e_k^2 in model P^2 such that $\alpha_k^2(u) \neq \alpha_k^2(v)$, and by Lemma 13, $\alpha_k^1(u) = \alpha_k^2(u)$. So $\alpha_k^1(u) \neq \alpha_k^2(v)$. But this contradicts the initial assumption, so the theorem is proved by contradiction.

□

4.7 Summary

This chapter has presented an algorithm for point location in an approximate polygon. The point location algorithm has been proved robust, and the point location problem has been shown to be consistent.

Ironically, this point location algorithm is *not* used in the polyhedral intersection algorithm of Chapter 9. A simpler alternative was found which makes use of additional information available from the polyhedral intersection algorithm. This is discussed further at the end of Section 9.1.2.

However, this example has served to demonstrate the theory of robustness presented in Chapter 3. The interesting thing to note is that the algorithm is fairly simple and never

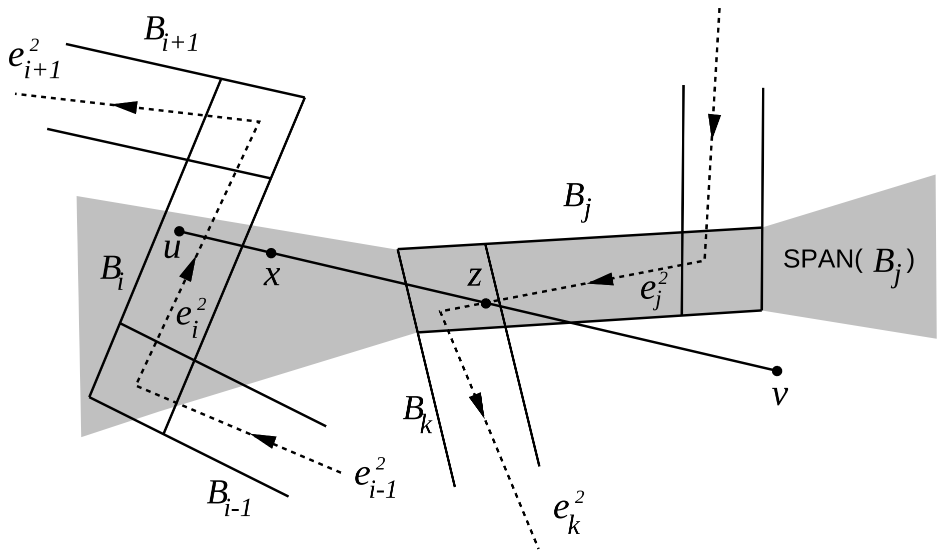

Figure 4.11: u and v On Opposite Sides of e_k^2

explicitly constructs a model polygon. The proof of robustness is quite straightforward; the real difficulty lies in the proof of consistency.

Not all problems with approximate polygons can be solved as easily. In fact, there are some apparently simple problems which turn out to be NP-hard, as will be demonstrated in the next chapter.

Chapter 5

Maximizing Incidences Between Approximate Polygons

He can compress the most words into the smallest idea of any man I ever met.
– Abraham Lincoln

Some apparantly simple problems having to do with approximate polygons are in fact quite difficult. This chapter demonstrates that the problem of maximizing incidences between approximate polygons is NP-hard [13].

5.1 Maximizing Incidences

Given two approximate polygons, P^1_{rep} and P^2_{rep}, we would like to find corresponding model polygons P^1 and P^2 having a maximum number of vertex/edge incidences between them. Let us represent an *incidence* as a triple of edges (e_i, e_j, e_k) which intersect at a point.

Incidence Maximizing Problem

Given two approximate polygons, P^1_{rep} and P^2_{rep}, determine a maximum set of incidences $\mathcal{I}$ such that there exist two corresponding models, P^1 and P^2, which have the incidences of $\mathcal{I}$.

In the theorem that follows it is shown that the Incidence Maximizing Problem in NP-hard. The proof consists of a reduction from 3SAT [13]: given a 3SAT formula, two approximate polygons are constructed which have a maximum number of incidences equal to some number k if and only if the 3SAT formula is satisfiable.

Theorem 2 (Incidence Maximizing)

The Incidence Maximizing Problem is NP-hard.

Proof

Given a 3SAT formula $\mathcal{F}$ with clauses c_i, terms t_j, and variables x_i, construct two approximate polygons as described below. These approximate polygons will achieve a maximum of k incidences (k is defined later) if and only if $\mathcal{F}$ is satisfiable. If $\mathcal{F}$ is not satisfiable then the maximum number of incidences will be fewer than k.

In the following, one of the approximate polygons will be designated as "red" and the other as "blue". In the figures, the red approximate polygon will be drawn with solid lines, and the blue one will be drawn with broken lines.

Variables

The key to the proof lies in building approximate polygons that force the corresponding model polygons to make choices between two alternatives, as follows. Let each variable x_i of the formula $\mathcal{F}$ be represented with a horizontal band from the red approximate polygon and four bands from the blue approximate polygon, as shown in Figure 5.1. Each blue band has zero width and the four blue bands intersect to form two blue corners, one touching the upper boundary of the red band and the other touching the lower boundary of the red band. The red band has some nonzero width.

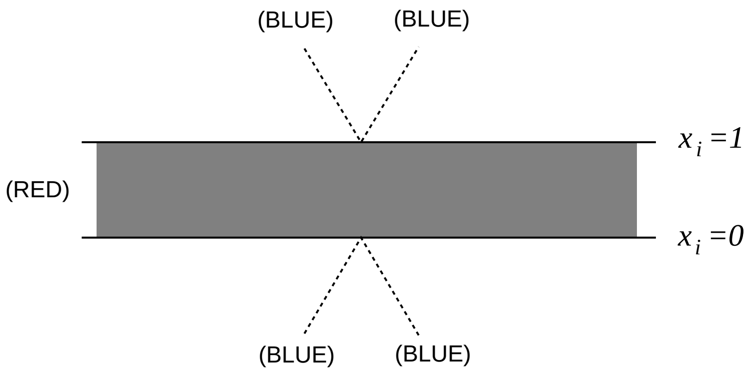

Figure 5.1: The Variable Structure

Lemma 14 *The structure of Figure 5.1 achieves a maximum of one incidence if and only if the edge of the red model polygon lies* ON *the top boundary or* ON *the bottom boundary of the red band.*

Proof Since the blue bands have zero width, the blue model edges must lie exactly on the blue bands, and the blue model vertices must lie exactly on the boundaries of the red band. The only red model edges which touch these vertices lie ON the top boundary or ON the bottom boundary of the red band. □

This combination of four zero width bands touching the boundary of some thick band will be called a *constraint mechanism*. Note that the constraint mechanism achieves a maximum of one incidence exactly when an edge lies on the boundary of the thick band.

For each variable x_i of the formula $\mathcal{F}$, construct a set of bands and put a constraint mechanism on each such band, as described above. All of the bands will later be joined to form two approximate polygons.

Terms

Each term t_j of the formula $\mathcal{F}$ is either a variable (x_i) or the negation of a variable ($\overline{x_i}$). Let each term be represented with a *vertical* blue band and a constraint mechanism in the same manner as a variable (where the constraint mechanism has *red* zero width bands. This is called a *term structure*. A blue edge on the right boundary of the blue band will signify $t_j = 1$. An edge on the left boundary will signify $t_j = 0$.

Lemma 15 *The term structure achieves a maximum of one incidence if and only if the blue model edge lies* ON *the right boundary or* ON *the left boundary of the blue band.*

Consistent Variables and Terms

The construction of the approximate polygons must ensure that the model edges representing the variables and and those representing the terms are consistent. For example, if $x_i = 1$ and $t_j = \overline{x_i}$ then $t_j = 0$. This is achieved with one of the two transfer mechanisms of Figure 5.2, in which the horizontal red band represents variable x_i and the vertical blue bands represent terms $t_j = x_i$ (on the left of the figure) and $t_k = \overline{x_i}$ (on the right of the figure). Each vertical blue band is connected at top to another slanted blue band of zero width, which has above it another vertical blue, also of zero width.

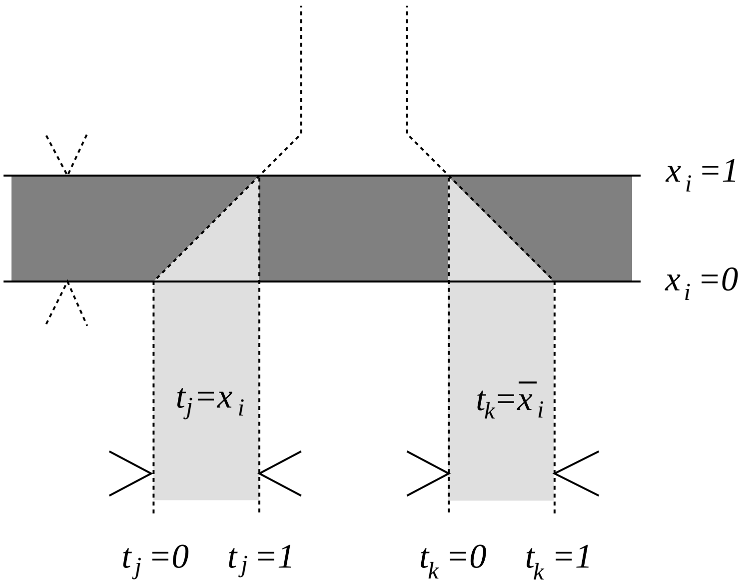

Figure 5.2: Transferring Logical Values

Lemma 16 *Assuming that the edges corresponding to x_i and t_j lie on band boundaries, the "transfer mechanism" of Figure 5.2 achieves a maximum of one incidence if and only if the model edges corresponding to x_i and t_j represent consistent logical values.*

Proof

(Only If) The blue model will have a vertex somewhere on the slanted blue band. To achieve one incidence, the red and blue edges corresponding to x_i and t_j must meet at this vertex. If the red edge represents $x_i = 1$ then the vertex must lie at the top of the slanted blue band, and the blue edge representing t_j must lie on the right of the blue band for the case that $t_j = x_i$ (seen on the left of the figure). Thus the edges representing x_i and t_j are consistent. Similar arguments apply for $t_j = \overline{x_i}$ and for a red edge on the bottom of the red band.

(If) Inspection of the figure shows that edges representing consistent values of x_i and t_j meet at the top or bottom of the slanted blue band and achieve one incidence. □

The transfer mechanism allows the creation of an array of horizontal red bands (representing variables) and vertical blue bands (representing terms) as shown in Figure 5.3. The blue bands — whose blue edges will represent logical values consistent with some choice of variable values — are used as input to a "clause mechanism". The output of the clause mechanism will be a single edge representing the logical OR of the three inputs.

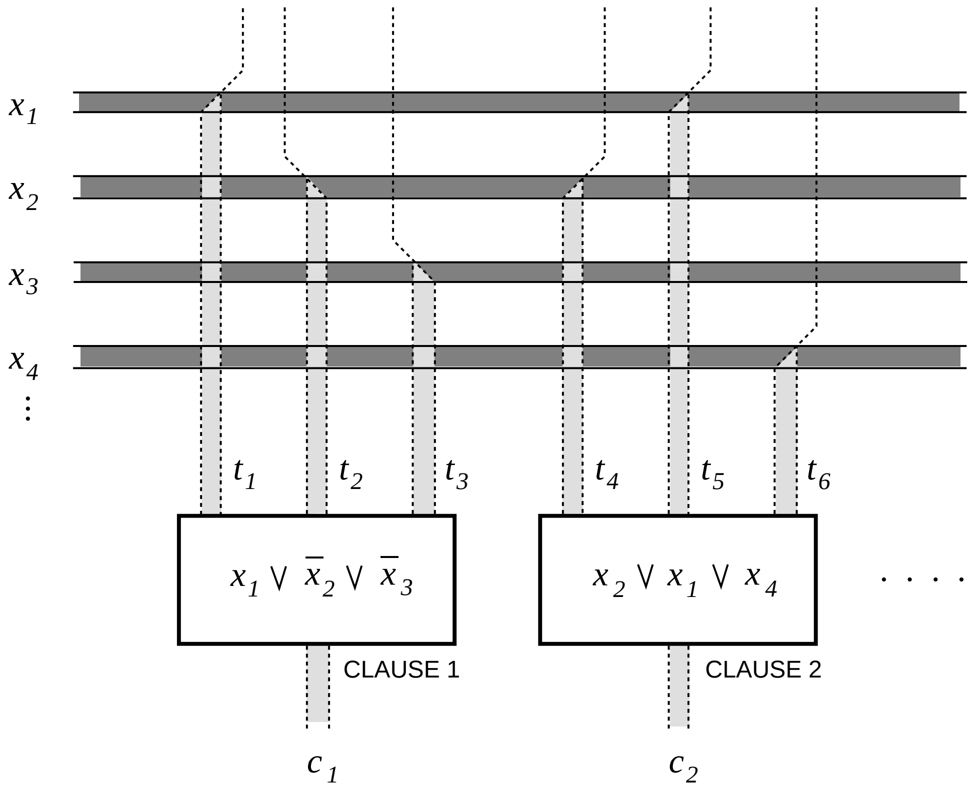

Figure 5.3: Array of Variables and Terms, With OR Gates

OR Gates

A two input OR gate will be used to build the clause mechanisms. This OR gate is shown in Figure 5.4 and has two blue input bands and one red output band.

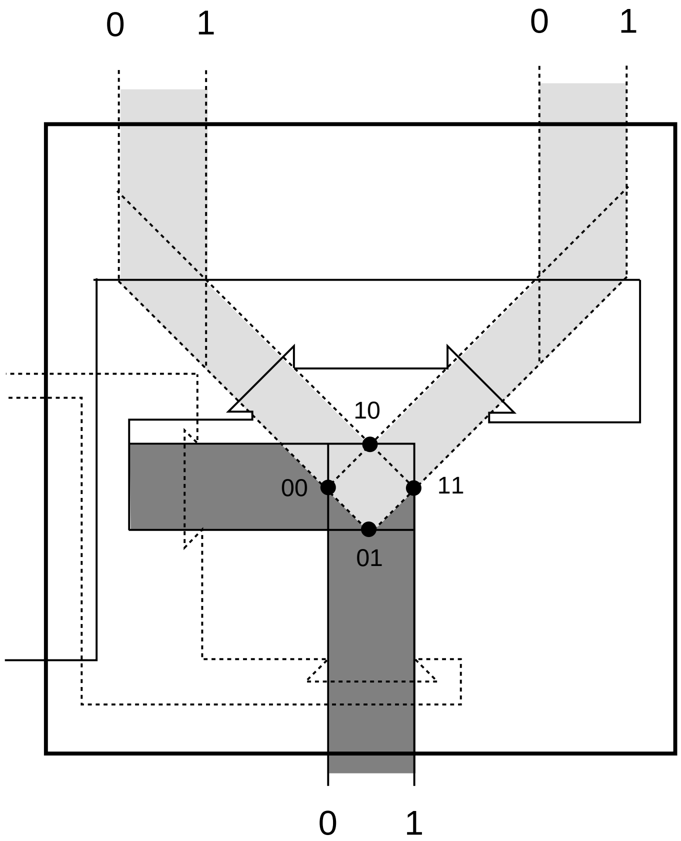

Figure 5.4: An OR Gate

Lemma 17 *Assuming that the edges corresponding to the input bands lie on the band boundaries, the OR gate of Figure 5.4 achieves a maximum of seven incidences if and only if the red output edge represents the logical OR of the two blue input edges.*

Proof The device consists of four thick blue bands, two thick red bands, and a variety of zero width bands of each color. The zero width bands are used to build four constraint mechanisms which constrain the models of the thick bands to lie on the band boundaries (note that the two *input* bands, coming from term structures of the array in Figure 5.3, already have this constraint). Also, a zero width red band (the uppermost, solid, horizontal line in the figure) acts as a transfer mechanism to transfer logical values from the two blue input bands to the two blue interior bands.

The two blue interior *edges* will intersect at a vertex which lies on one of the four points labeled 00, 01, 10, and 11 in the figure. The labeling indicates the logical input values which cause the blue vertex to appear at that location.

Recall that, to achieve a maximum number of incidences, the *red* edges are forced to lie on the boundaries of the red bands by the blue constraint mechanisms. This means that there are exactly four maximal red configurations. However, for each of the blue vertices at 00, 01, 10, or 11, one of the four configurations can achieve an *additional* incidence by touching that vertex.

- Blue vertex at 00: The horizontal red edge must be on the TOP of the horizontal red band and the vertical red edge must be on the LEFT of the vertical red band.
- Blue vertex at 01: The horizontal red edge must be on the BOTTOM and the vertical red edge must be on the RIGHT.
- Blue vertex at 10: The horizontal red edge must be on the TOP and the vertical red edge must be on the RIGHT.
- Blue vertex at 11: The horizontal red edge must be on the TOP and the vertical red edge must be on the RIGHT.

Thus, when achieving a maximum number of incidences in the OR gate, the red output edge appears on the right (logical 1) if and only if one of the blue input edges represents a logical 1. Otherwise the red output edge appears on the left (logical 0). Inspection of Figure 5.4 shows a maximum of seven simultaneous incidences (four from the constraint mechanisms, two from the transfer mechanisms, and one from incidence with the blue vertex). These seven incidences only occur when the output is the logical OR of the inputs. □

Figure 5.5 shows an abstract view of the OR gate. The dashes show the internal connections.

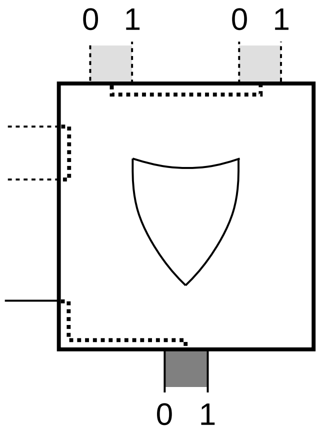

Figure 5.5: An OR Gate Abstraction

Clauses

For each clause c_i, two OR gates are joined to form a three input OR gate. This construction is shown in Figure 5.6. The only subtle aspect of the construction is that the second OR gate has two red inputs, one of which is transferred from the bottommost blue input. Thus the clause has three blue inputs and one blue output.

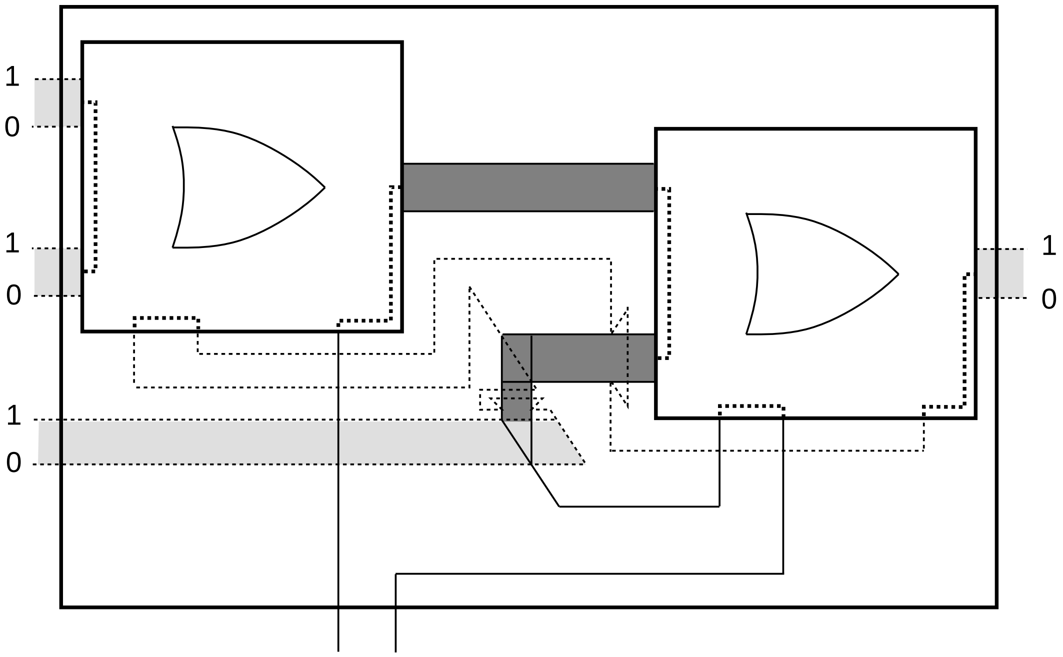

Figure 5.6: A Clause Structure

Lemma 18 *Assuming that the edges corresponding to the input bands lie on the band boundaries, the "clause mechanism" of Figure 5.6 achieves a maximum of eighteen incidences if and only if the blue output edge represents the logical OR of the three blue input edges.*

Proof From Lemma 17 each OR gate achieves a maximum of seven incidences exactly when its output is the logical OR of its inputs. The clause uses two constaint mechanisms and two transfer mechanisms to transfer the bottommost blue input to the bottom red input of the second OR gate. Each of the four mechanisms achieves a maximum of one incidence exactly when it operates correctly. Thus the clause achieves a maximum of eighteen incidences exactly when the output is the logical OR of the three inputs. □

Satisfying each Clause

If the formula $\mathcal{F}$ is satisfiable, then each clause c_i must be TRUE. In each corresponding clause mechanism, the blue output edge should appear on the right boundary of the blue output band. Thus we will add a single potential incidence to the *right side* of the blue output band of each clause, as shown in Figure 5.7. These incidences are achieved exactly when the output from each clause represents a logical 1.

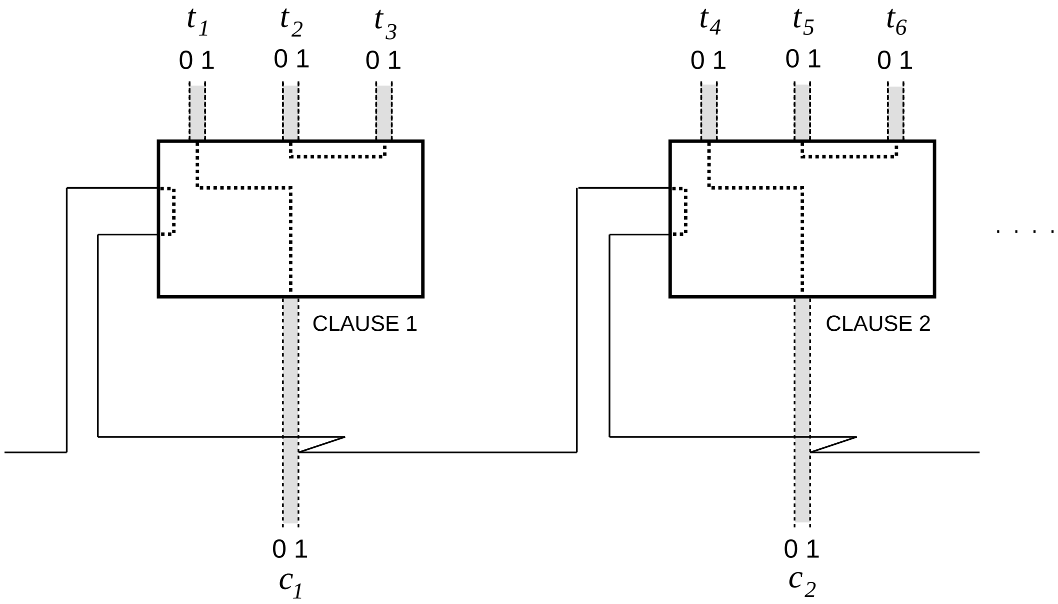

Figure 5.7: Forcing a Clause to be TRUE

Lemma 19 *The mechanism of Figure 5.7 achieves a maximum of one incidence per clause mechanism if and only if the output of the clause is a logical* 1*.*

Lemma 20 *The constructions of lemmas 14 through 19 can be joined to form two approximate polygons.*

Proof Left as an exercise to the reader. □

Finally we must count the maximum number of incidences possible. Lemma 14 gives one incidence per variable. Lemma 15 gives one incidence per term, and Lemma 16 gives one more per term at the transfer mechanism. From Lemma 18 each clause generates eighteen incidences, and from Lemma 19 each clause generates one more incidence on its output.

Lemma 21 *Let V be the number of variables and let C be the number of clauses. Then the maximum number of incidences obtainable from models of the two approximate polygons constructed above is $V + 25C$ if and only if the formula $\mathcal{F}$ is satisfiable.*

Proof

(If) If there is a satisfying assignment to $\mathcal{F}$, then this satisfying assignment has a corresponding set of red edges which lie on the boundaries of the red bands of the variable mechanisms. If all of the other edges are chosen to maximize the number of incidences, then these logical values of the variables will eventually propagate to the clause outputs, each of which will be a logical 1. Thus a maximum number of incidences will be acheived. This number is $V + 25C$, which comes from counting the maximum number of incidences per mechanism (recall that there are three terms per clause).

(Only If) The maximum number of possible incidences is $V + 25C$. If there are $V + 25C$ incidences between the two models of the two approximate polygons, then the edges represent a satisfying assignment of variables which results in each clause output representing a logical 1. Thus $\mathcal{F}$ is satisfiable. □

In conclusion, given a 3SAT formula, a pair of approximate polygons can be constructed such that the formula has a satisfying assignment if and only if there are models of the approximate polygons which have $V + 25C$ incidences between them. Thus the formula is satisfiable if and only if the size of the maximum set of incidences is $V + 25C$. The construction obviously takes polynomial time, so the Incidence Maximizing Problem is NP-hard.

□

5.2 Related Open Problems

A problem similar to the Incidence Maximizing Problem is that of arranging two approximate polygons. An *arrangement* of two polygons is a complete enumeration of all of the vertex/edge relations between the two polygons, where a vertex/edge relation is one of either LEFT, RIGHT, or ON.

Open Problem 1 (Approximate Polygon Arrangement)

Given an arrangement of the vertices and edges of two approximate polygons, do there exist two corresponding model polygons which have that arrangement?

It is discouraging that the Incidence Maximizing Problem is NP-hard, since this is of the same flavor as the Approximate Polygon Arrangement problem (although there is no mention of "maximizing" in the latter problem).

The second open problem is only slightly related to the Approximate Polygon Arrangement problem. Consider an approximate *polyhedron* to be the analogue of an approximate polygon, but in $\Re^3$.

Open Problem 2 (Approximate Polyhedron Validity)

Given an approximate polyhedron, does it have a model?

This problem is more subtle than would appear at first glance. While models trivially exist for any approximate *polygon*, the models of approximate polyhedra are overconstrained (that is, they can have more than three planes meeting at a vertex).

Open Problem 3 (Approximate Polyhedron Arrangement)

Given two approximate polyhedra, find an arrangement of the vertices and faces such that there exist two model polyhedra which have that arrangement.

It is unfortunate that this is an open problem, since we *must* solve this problem if we are to build a provably robust polyhedral intersection algorithm. In the next chapter it will be argued that since we have not been able to solve this and related problems robustly, we must settle for a less rigorous definition of robustness with which to build practical, "real world" algorithms, such as those found in solid modeling systems.

Chapter 6

Local Robustness

When writers refer to themselves as "we" and to the reader as "you", it's two against one. – Judith Rascoe

The previous chapter showed that the problem of maximizing incidences between two approximate polygons is NP-hard. Another problem, that of arranging two approximate polyhedra, was posed as an open problem. This problem must be solved in any robust solid modeler.

Since we have not yet solved this problem, we cannot yet build a robust solid modeler. However, we should not abandon the idea of robustness and blindly write programs which "usually work". Some middle ground must be found between an absolute proof of robustness and no proof at all. The middle ground proposed in this chapter is called *local robustness*, and involves proving robustness for characteristic subsets of the algorithm's input.

6.1 Robust Line Arrangement Is Difficult

That some geometric problems are inherently difficult to solve robustly was first proposed by Victor Milenkovic. He states that

> "All true line arrangements must satisfy a rich set of theorems of projective geometry" [25]

and illustrates this with the example of Pappus's Theorem from projective geometry:

Theorem 3 (Pappus)
Given nine points in the plane, call them $a, b, c, a', b', c', a'', b''$, and c'', if the following triples are collinear,

$$abc, a''b''c'', aa'b'', ab'c'', ba'a'', bc'c'', cb'a'', cc'b'',$$

then a', b', and c' are also collinear.

A robust line arrangement algorithm must obey Pappus's Theorem. That is, it cannot output an arrangement with the above labeling in which the first eight triples of points

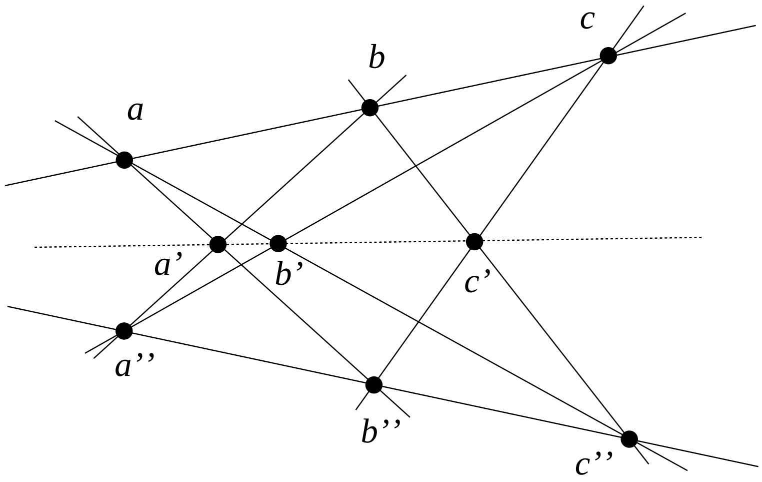

Figure 6.1: Pappus's Theorem

are collinear, and in which a', b', and c' are not collinear. Similarly, all other theorems of geometry must be obeyed.

The conclusion of Milenkovic is that a robust line arrangement algorithm *must* embody a large set of theorems of geometry. It is probably unreasonable to expect to build such an algorithm having less than exponential time complexity.

However, no practical application which uses line arrangements is going to require that all theorems of geometry be true of the arrangement. For example, a point location algorithm for points in line arrangements is probably not going to rely on Pappus's Theorem. A point location algorithm would probably work just as well on the "invalid" line arrangement shown in Figure 6.2 (which violates Pappus's theorem), as it would on the "valid" arrangement shown in Figure 6.1. We should be willing to believe that most algorithms are *not* going to fail because they make decisions which violate some complex theorem.

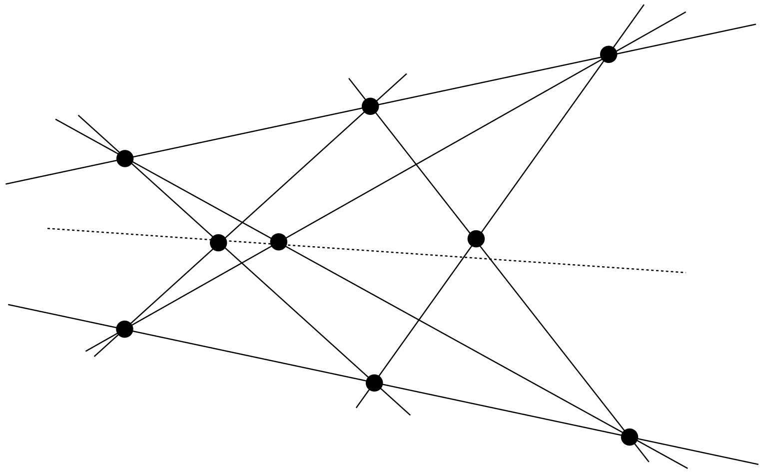

Figure 6.2: A Nonexistent Line Arrangement

6.2 Robustness In Practice

Consider the problem of line arrangements again. An algorithm which uses a line arrangement may make use of only a few properties of line arrangements. It might, for example, only rely on the properties that

(L1) two lines in the arrangement intersect exactly once, and

(L2) each cell of the arrangement is bounded by a simple cycle of edges.

If so, then we would just as well be satisfied by an arrangement of pseudolines, since these properties are also true of pseudolines. This is in fact what Milenkovic produces with his Robust Line Arrangement Algorithm [25]. Such an arrangement of pseudolines is just as useful to the majority of practical applications as an arrangement of straight lines, and, as Milenkovic demonstrated, can be computed robustly and efficiently.

Consider a second example: in building an algorithm to construct Voronoi diagrams, Sugihara and Iri [41,39] propose that the important properties that their algorithm must preserve in a Voronoi diagram are that

(P1) the Voronoi diagram is a planar graph which partitions the plane into as many cells as the generators, and

(P2) two cells share at most one common edge.

Their algorithm incrementally adds points to a Voronoi diagram, each time modifying the Voronoi diagram in a local area and ensuring that properties P1 and P2 are preserved within that area. Although not proved robust in the sense of Chapter 3, the algorithms of Sugihara and Iri [41,39] and of Sugihara, Ooishi, and Imai [42] are remarkable in their ability to withstand numerical error.

For example, one of their algorithms uses only single precision arithmetic to construct an approximation of the Voronoi diagram for over one hundred thousand points distributed in a unit square (having an average distance between points on the order of 2^{-8}). Another algorithm constructs with single precision an approximation of the Voronoi diagram for two thousand points distributed on a common circle. The conventional algorithms usually fail for both problems [42].

In conclusion, if we cannot build a provably robust algorithm for a particular problem, then we should build an algorithm which at least makes some consistent decisions; that is, decisions which maintain some small set of properties of the data. For example, in building line arrangements it may be sufficient to maintain properties L1 and L2, and in building Voronoi diagrams it is probably sufficient to maintain properties P1 and P2. The difficulty lies in determining which properties are necessary to ensure the success of a given algorithm.

6.3 Formalizing Local Robustness

So far, it has been argued that for some problems it is very difficult to build robust, efficient algorithms. It has also been argued that, for most *practical* problems, we don't need to build a provably robust algorithm; one which maintains certain properties of the problem is in most cases sufficient, and is in all cases better than one which doesn't maintain any properties at all. We should formalize this notion.

Reviewing the properties discussed above, we find that they are of a predominantly *local* nature. That is, when the data is viewed as a structure in $\Re^2$ or $\Re^3$, the properties involve only a small set of adjacent features.

For example, property L1 is concerned about a pair of lines in a line arrangement, and property L2 is concerned about a single cell in the arrangement. With Voronoi diagrams, property P1 requires that the complete diagram be a planar graph and that each cell contain exactly one generator, and property P2 requires that a pair of cells share at most one edge. At first glance property P1 is *not* a local property, since the planarity requirement involves the whole Voronoi diagram. However, the planarity requirement is equivalent to stating that, in any embedding of the graph, edges only intersect at their endpoints. This is a local property between pairs of edges.

Suppose that an algorithm only makes decisions which are consistent with a "sufficiently large" set of properties, where each property involves a small set of adjacent features of the data. Then the algorithm will probably not make *any* inconsistent decisions involving sets of those features. This is the philosophy behind local robustness.

Local Robustness

An algorithm is *locally robust* for a set of features if it is robust for all inputs consisting of exactly those features.

For example, if a line arrangement algorithm is provably robust for all inputs consisting of exactly five lines, then we would say that the algorithm is "locally robust for five lines". Say we developed a line arrangement algorithm and proved it locally robust for two, three, four, and five lines. Then the algorithm *will not fail* because of incorrect decisions involving up to five lines. The algorithm will only fail by making decisions which violate some complex theorem of geometry involving six or more lines. We should be much more confident of such an algorithm than of one which is not locally robust.

By proving the local robustness of an algorithm we are proving that the algorithm does not make inconsistent decisions involving small sets of features. The only possible errors that the algorithm can make are of a more complex nature:

Principle of Local Robustness

If an algorithm is locally robust for a sufficiently rich set of features, then it is unlikely that the algorithm will fail in practice. If it fails, then is does so in an interesting manner which involves a complex interaction of many features.

6.3.1 Comparing Locally Robust Algorithms

For a particular geometric problem, a partial order can be established on the algorithms which solves the problem, based upon the set of inputs for which the algorithm is provably robust. This allows us to compare robust algorithms and to state that one algorithm is "more robust" than another.

Comparative Robustness

For a given problem, let algorithm A_1 be provably robust for all inputs in a set I_1, and let algorithm A_2 be provably robust for all inputs in a set I_2. Then A_1 is *more robust* than A_2 if $I_1 \supset I_2$.

Chapter 7

Locally Robust Polyhedral Arrangement

The next step is to apply the notion of local robustness to the design of a polyhedral intersection algorithm. Such an algorithm contains the following high level components:

- A *point-in-polygon test* that determines whether a point of intersection between the two polyhedra is INSIDE or OUTSIDE a face.
- An *arrangement algorithm* that gives a complete enumeration of point/plane relations between the two objects being intersected.
- A *linear sorting algorithm* for ordering points along an edge.
- A *radial sorting algorithm* for ordering coplanar points around a center point.
- A *dihedral sorting algorithm* for ordering points in $\Re^3$ *around* an edge.

The first component, a robust point-in-polygon algorithm, was developed in Chapter 4. The second component, an arrangement algorithm, will be considered in this chapter. The remaining three components will be considered in Chapter 8.

7.1 An Arrangement Algorithm

Define a *feature* to be a point, a line, or a plane (corresponding to a vertex, an edge, or a face of a polyhedron). A plane is oriented to have an INSIDE and an OUTSIDE. Relations exist among sets of these features; four such relations are described below (where a and b are points, ℓ is a line, and p is a plane):

- INCIDENCE(a, b) $\in$ {ON, OFF}. A point a is either ON or OFF a point b.
- INCIDENCE(a, ℓ) $\in$ {ON, OFF}. A point a is either embedded ON a line ℓ, or is OFF ℓ.
- LOC(a, p) $\in$ {INSIDE, ON, OUTSIDE}. A point a is either INSIDE, ON, or OUTSIDE an oriented plane p.
- INCIDENCE(ℓ, p) $\in$ {ON, OFF}. A line ℓ is either embedded ON a plane p, or contains points that are OFF p.

The input to an arrangement algorithm consists of a set of features, F, some topological information relating the features, and a set of all "allowable" relations between the features, R_{allow}. Define a *complete* set of relations among the features of F as a set that contains exactly one relation for every group of features in F that can be related. Then the output of the arrangement algorithm consists of a complete set of relations, R. Furthermore, $R \subseteq R_{allow}$. Let us call R an *arrangement* of the features of F.

The algorithm is *robust* if there exists a set of real features, corresponding to those of F, that satisfy the relations of R. Call this set of real features the *model.* We shall assume that R_{allow} always admits a valid R.

For example, if the input to the algorithm consisted of two planes, their line of intersection, and a point, then

$$F = \{v, p_1, p_2, \ell\},$$

where v is the point, p_1 and p_2 are the planes, and $\ell = p_1 \cap p_2$. The set of allowable relations, R_{allow}, is a subset of

$$\begin{array}{ll} \{ & v \text{ INSIDE } p_1, \quad v \text{ ON } p_1, \quad v \text{ OUTSIDE } p_1, \\ & v \text{ INSIDE } p_2, \quad v \text{ ON } p_2, \quad v \text{ OUTSIDE } p_2, \\ & v \text{ ON } \ell, \quad v \text{ OFF } \ell, \\ & \ell \text{ ON } p_1, \\ & \ell \text{ ON } p_2 \ \}. \end{array}$$

Suppose that some other component of the polyhedral intersection algorithm had concluded, perhaps by an unambiguous numeric test, that v INSIDE p_1. Then the set R_{allow} would be defined as

$$\begin{array}{lll} R_{allow} = & \{ & v \text{ INSIDE } p_1, \\ & & v \text{ INSIDE } p_2, \quad v \text{ ON } p_2, \quad v \text{ OUTSIDE } p_2, \\ & & v \text{ ON } \ell, \quad v \text{ OFF } \ell, \\ & & \ell \text{ ON } p_1, \\ & & \ell \text{ ON } p_2 \ \}, \end{array}$$

and the task of the arrangement algorithm would be to pick a consistent, complete set $R \subseteq R_{allow}$. For example, the set

$$\begin{array}{lll} R = & \{ & v \text{ INSIDE } p_1, \\ & & v \text{ INSIDE } p_2, \\ & & v \text{ ON } \ell, \\ & & \ell \text{ ON } p_1, \\ & & \ell \text{ ON } p_2 \ \} \end{array}$$

would *not* be consistent, since there is no model for F in which v is not ON planes p_1 and p_2, yet *is* ON their line of intersection.

In this chapter we will characterize locally robust arrangement algorithms. The characterization will consist of a set of logical rules that are true of the algorithm's output R if and only if a model exists that satisfies R.

7.2 Arranging A Cone And Two Points

A cone is a precise geometric structure that corresponds to a polyhedron corner. Let us consider arranging a cone with two points. If we can characterize the corresponding valid arrangements, R, then we will have characterized arrangement algorithms that are locally robust for a cone and two points.

An example of a cone is shown in Figure 7.1. The cone consists of a set of faces that meet at a common vertex. A cone is defined by a vertex and a sequence of at least three faces,

$$\mathcal{C} = \{v, f_1, f_2, \ldots, f_n\}.$$

Each semi-infinte face is bounded by two edges. A pair of faces share an edge exactly when they are adjacent in the sequence. All other pairs of faces share exactly the vertex v. The face normals are oriented consistently; that is, the OUTSIDE and INSIDE surfaces of adjacent faces are themselves adjacent.

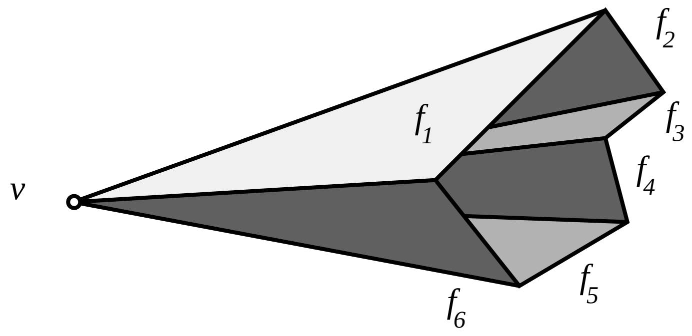

Figure 7.1: A Cone

In arranging a cone with two points, the input to the arrangement algorithm consists of a set of features, F, some topological information, and a set of allowable relations, R_{allow}. The features consist of the two points a and b, the point v which is the cone vertex, the planes p_i which correspond to the cone faces f_i, and the lines ℓ_i which correspond to the cone edges.

$$F = \{a, b, v, p_1, p_2, \ldots, p_n, \ell_1, \ell_2, \ldots, \ell_n\}$$

The topological information specifies that $\mathcal{C} = \{v, f_1, f_2, \ldots, f_n\}$ is a cone and that face f_i of the cone lies on plane p_i. The set of allowable relations, R_{allow}, constrains v to lie ON each p_i, defines $\ell_i = p_i \cap p_{i+1}$, and permits the points a and b to have any relations whatsoever to the planes p_i, to the lines ℓ_i, to the point v, and to each other. In the following, let i take on all values in $1 \ldots n$, and consider the subscripts *modulo* n:

$$
\begin{aligned}
R_{allow} = \{ \quad & v \text{ ON } p_i, \\
& \ell_i \text{ ON } p_i, \\
& \ell_i \text{ ON } p_{i+1}, \\
& a \text{ INSIDE } p_i, \quad a \text{ ON } p_i, \quad a \text{ OUTSIDE } p_i, \\
& b \text{ INSIDE } p_i, \quad b \text{ ON } p_i, \quad b \text{ OUTSIDE } p_i, \\
& a \text{ ON } \ell_i, \quad a \text{ OFF } \ell_i, \\
& b \text{ ON } \ell_i, \quad b \text{ OFF } \ell_i, \\
& a \text{ ON } v, \quad a \text{ OFF } v, \\
& b \text{ ON } v, \quad b \text{ OFF } v, \\
& a \text{ ON } b, \quad a \text{ OFF } b \ \}
\end{aligned}
$$

7.3 Nondegenerate Arrangements of a Cone and Two Points

Let us first consider those arrangements in which the points a and b do not lie ON any of the planes p_i. That is, the relations a ON p_i and b ON p_i are not present in R_{allow}. As we will prove in Theorem 4, the rules of Table 7.1 characterize those complete sets $R \subseteq R_{allow}$ that have models.

In Table 7.1, x and y are points, p is a plane, and f is any feature in F. The LOC relation was defined in Section 7.1. Rule 1 states that if points x and y are coincident then they have the same relation to each plane p. Rule 2 states that a feature lies ON two other features if and only if it lies ON their intersection.

Table 7.1: Nondegenerate Arrangement Rules

Rule 1 :	x ON $y \;\Rightarrow\;$ LOC$(x,p) =$ LOC(y,p)
Rule 2 :	x ON $f_1 \;\wedge\; x$ ON $f_2 \;\Leftrightarrow\; x$ ON $f_1 \cap f_2$

Consider the set of allowable relations, R_{allow}. The point a cannot be ON any line ℓ_i, as follows: if a is ON ℓ_i then, since $\ell_i = \ell_i \cap p_i$, a must also be ON p_i by Rule 2. But we have assumed that a is not ON any plane, so it cannot be ON any line, either. By a similar argument, it cannot be ON the point v. The same applies to the other point, b.

Let us assume that a is OFF b (otherwise, a and b will simply have the same relations to each plane p_i, by Rule 1). Then the set of allowable relations has been reduced to

$$
\begin{aligned}
R_{nondeg} = \{ \quad & v \text{ ON } p_i, \\
& \ell_i \text{ ON } p_i, \\
& \ell_i \text{ ON } p_{i+1}, \\
& a \text{ INSIDE } p_i, \quad a \text{ OUTSIDE } p_i, \\
& b \text{ INSIDE } p_i, \quad b \text{ OUTSIDE } p_i, \\
& a \text{ OFF } \ell_i, \\
& b \text{ OFF } \ell_i, \\
& a \text{ OFF } v, \\
& b \text{ OFF } v, \\
& a \text{ OFF } b \;\}.
\end{aligned}
$$

The only ambiguous relations in R_{nondeg} are $a \bullet p_i$ and $b \bullet p_i$. In the next section we will prove the following theorem.

Nondegenerate Arrangement Theorem

For any complete set of relations $R \subseteq R_{nondeg}$, there exist a cone and two points that satisfy R.

7.4 The Nondegenerate Arrangement Theorem

In this section we will prove that two points can be arbitrarily arranged with a cone, as long as no point lies ON a plane defined by a face of the cone. The reader can skip to the next section, and come back later if interest warrants.

Define a set

$$\Phi = \{ \begin{array}{l} (\text{INSIDE}, \text{INSIDE}), \\ (\text{INSIDE}, \text{OUTSIDE}), \\ (\text{OUTSIDE}, \text{INSIDE}), \\ (\text{OUTSIDE}, \text{OUTSIDE}) \ \}, \end{array}$$

and let $\rho \in \Phi^n$. Each $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$ corresponds to exactly one arrangement $R \subseteq R_{nondeg}$. Hence, ρ will also be called an *arrangement*. A real cone $\mathcal{C} = \{v, f_1, f_2, \ldots, f_n\}$ and two real points, a and b, are said to *satisfy* an arrangement ρ if

$$\rho_i = (\text{LOC}(a, p_i), \text{LOC}(b, p_i)),$$

where p_i is the plane defined by face f_i of the cone.

Thus we must prove that all arrangements $\rho \in \Phi^*$ are satisfiable. In the sections that follow, several preliminary results will be presented before this is proved.

7.4.1 Reducing The Problem To $\Re^2$

Just as a point can be INSIDE or OUTSIDE a plane in $\Re^3$, a point can be LEFT or RIGHT of a directed line in $\Re^2$. The relation between a point x and a directed line ℓ is

$$\text{SIDE}(x, \ell) \in \{\text{LEFT}, \text{ON}, \text{RIGHT}\}.$$

Define a set

$$\Sigma = \{ \begin{array}{l} (\text{RIGHT}, \text{RIGHT}), \\ (\text{RIGHT}, \text{LEFT}), \\ (\text{LEFT}, \text{RIGHT}), \\ (\text{LEFT}, \text{LEFT}) \ \}, \end{array}$$

and let $\sigma \in \Sigma^n$. Then $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is an *arrangement* in $\Re^2$. A real polygon $P = (e_1, e_2, \ldots, e_n)$ and two real points, x and y, are said to *satisfy* an arrangement σ if P is simple and if

$$\sigma_i = (\text{SIDE}(x, \ell_i), \text{SIDE}(y, \ell_i)),$$

where ℓ_i is the line defined by edge e_i of the polygon, and directed such that the interior of the polygon appears to the RIGHT of ℓ_i along the segment e_i.

We shall use the following abbreviations for elements of Σ:

$$\begin{array}{lcl} \text{RR} & \stackrel{\text{def}}{=} & (\text{RIGHT}, \text{RIGHT}), \\ \text{RL} & \stackrel{\text{def}}{=} & (\text{RIGHT}, \text{LEFT}), \\ \text{LR} & \stackrel{\text{def}}{=} & (\text{LEFT}, \text{RIGHT}), \text{ and} \\ \text{LL} & \stackrel{\text{def}}{=} & (\text{LEFT}, \text{LEFT}). \end{array}$$

Define REDUCE as a function from relation LOC in $\Re^3$ to the relation SIDE in $\Re^2$:

$$\begin{aligned}
\text{REDUCE}(\text{OUTSIDE}) &= \text{LEFT}, \\
\text{REDUCE}(\text{INSIDE}) &= \text{RIGHT}, \\
\text{REDUCE}(\text{ON}) &= \text{ON}, \\
\text{REDUCE}((x, y)) &= (\text{REDUCE}(x), \text{REDUCE}(y)) \text{ for } (x, y) \in \Phi, \text{ and} \\
\text{REDUCE}((\rho_1, \rho_2, \ldots, \rho_n)) &= (\text{REDUCE}(\rho_1), \text{REDUCE}(\rho_2), \ldots, \text{REDUCE}(\rho_n)).
\end{aligned}$$

In the following sections we shall be discussing polygons that lie on the surface of a sphere, having edges that lie on great circles of the sphere.

Lemma 22 (Sphere Reduction)

There exist a cone and two points that satisfy an arrangement $\rho \in \Phi^*$ if and only if *there exist a polygon and two points on the surface of a sphere that satisfy* REDUCE(ρ).

Proof

(Only If) Given a cone and two points satisfying ρ, construct a polygon satisfying σ by intersecting the faces of the cone with the surface of any sphere that is centered at the cone vertex. Project the two points onto the surface of the sphere.

(If) Given a polygon and two points on the surface of a sphere that satisfy REDUCE(ρ), construct a cone satisfying ρ by placing the vertex of the cone at the center of the sphere and placing the faces of the cone to embed the edges of the polygon. □

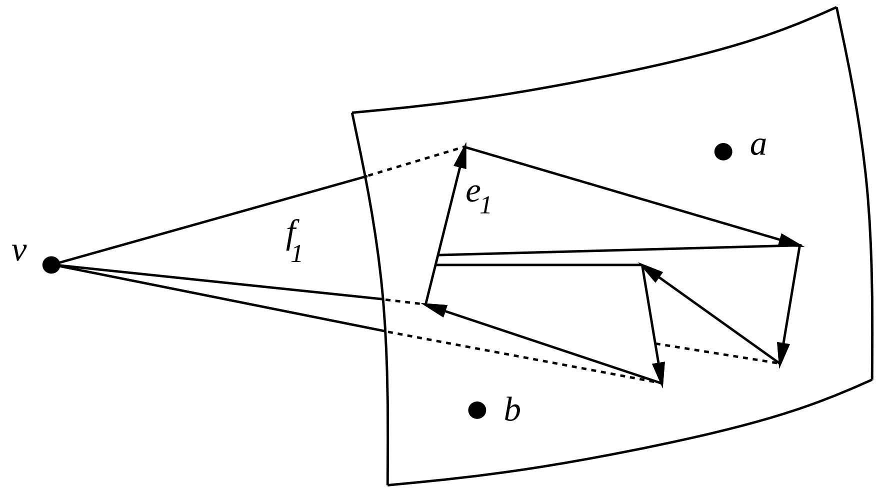

Figure 7.2: Constructing a Cone From a Polygon

For example, Figure 7.2 shows a polygon, $P = (e_1, e_2, \ldots, e_n)$, and two points, a and b, on the surface of a sphere, and also shows a cone $\mathcal{C} = (v, f_1, f_2, \ldots, f_n)$ whose vertex v is at the center of the sphere. The faces of the cone embed the edges of the polygon, which lie on great circles of the sphere. The polygon and two points satisfy the two dimensional arrangement

$$\sigma = (\text{RR}, \text{LR}, \text{RR}, \text{RL}, \text{LR}, \text{RL}),$$

whereas the cone and two points satisfy the three dimensional arrangement

$$\rho = (\quad (\text{INSIDE}, \text{INSIDE}), \ (\text{OUTSIDE}, \text{INSIDE}), \ (\text{INSIDE}, \text{INSIDE}), \\ (\text{INSIDE}, \text{OUTSIDE}), \ (\text{OUTSIDE}, \text{INSIDE}), \ (\text{INSIDE}, \text{OUTSIDE}) \ ).$$

7.4.2 Reductions of Arrangments $\sigma \in \Sigma^*$

The proof that follows will show how to construct a polygon and two points to satisfy any $\sigma \in \Sigma^*$. By the Sphere Reduction Lemma, this is equivalent to constructing a cone and two points to satisfy any $\rho \in \Phi^*$.

Given some $\sigma \in \Sigma^*$, a series of reductions will be applied to get a shorter σ'. The reductions will be chosen such that if a polygon exists satisfying the shorter σ' then one also exists satisfying the original σ.

With respect to the reductions that follow, there are only a few types of irreducible arrangements in Σ^*, each of which has a satisfying polygon. Since each of the irreducible arrangements has a satisfying polygon, all arrangements in Σ^* have a satisfying polygon.

For example, if an arrangement $\sigma' = (\text{LL}, \text{LR}, \text{RR}, \text{RL})$ has a satisfying polygon, as shown in Figure 7.3, then a satisfying polygon can also be constructed for the *longer* arrangement $\sigma = (\text{LL}, \text{LR}, \text{RR}, \text{RL}, \underline{\text{LL}, \text{RR}})$ (where σ' is obtained from σ by a reduction that deletes (LL, RR)). In this figure, the points a and b lie on the equator ℓ of the sphere, and the polygon lies in the northern hemisphere.

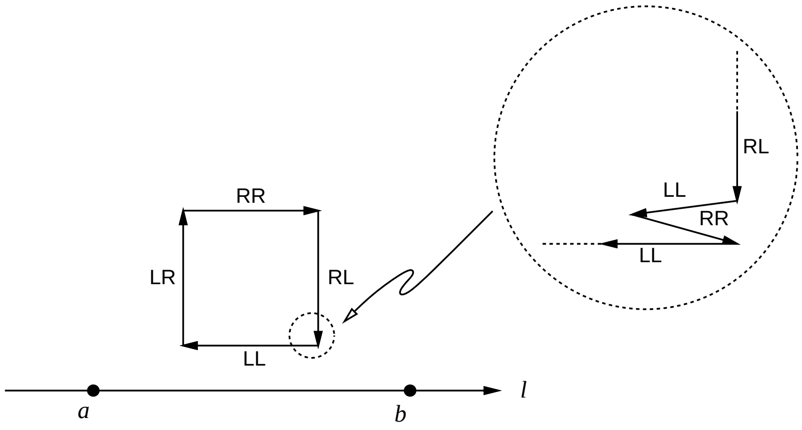

Figure 7.3: Reduction Example

In the example of Figure 7.3 the polygon satisfying σ' is modified in an arbitrarily small neighborhood around the RL-LL vertex. At this vertex two edges satisfying LL RR are added to get a polygon satisfying σ. Thus, at least in this example, one of the allowable reductions from σ to σ' is LL RR $\rightarrow \emptyset$, since if a satisfying polygon exists for σ' then one also exists for σ.

It turns out that most of the reductions are context sensitive. For example, the reduction LR RR $\rightarrow \emptyset$ is allowable in the context $\underline{\text{LL}}$ LR RR $\underline{\text{LR}}$, but is not allowable in the context $\underline{\text{LL}}$ LR RR $\underline{\text{RL}}$, as shown in Figures 7.4 and 7.5.

In Figure 7.4 two edges satisfying LR RR could be added at the LL-LR vertex, so the reduction $\underline{\text{LL}}$ LR RR $\underline{\text{LR}} \rightarrow$ LL LR is allowable. In Figure 7.5 no such edges can be added

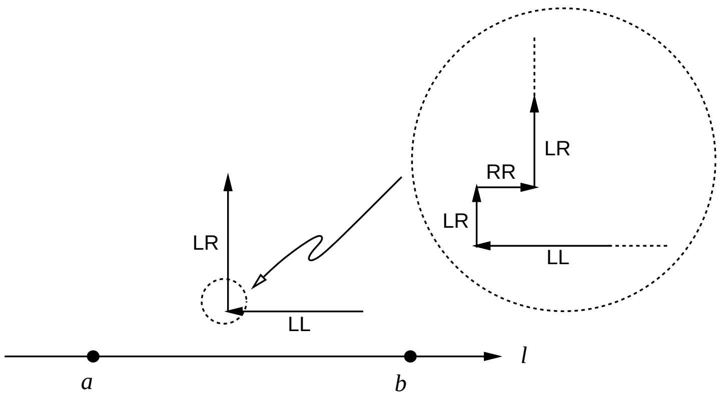

Figure 7.4: Legal Context For LR RR → ∅

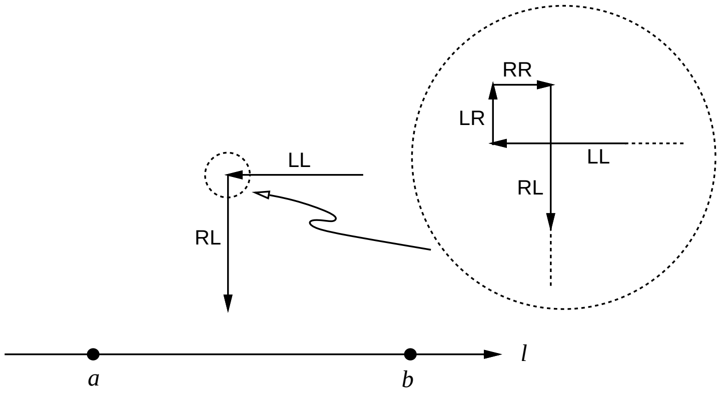

Figure 7.5: Illegal Context For LR RR → ∅

without resulting in a nonsimple chain. Thus, the different reduction LL LR RR RL → LL RL is not allowable.

In the following discussion let us assume that the edges of a polygon satisfying σ can be perturbed slightly and still satisfy σ. That is, around each edge there is a small envelope in which the edge may be positioned arbitrarily and still maintain the necessary relations (much like the edges of an approximate polygon). Such an envelope must exist since the polygon is simple and since a and b are not touched by any of the great circles on which the edges lie (since there are no ON relations).

7.4.3 Allowable Reductions

We would like to find a group of reductions that reduce long arrangements σ to shorter arrangements σ'. These reductions must have the property that if a polygon can be constructed satisfying σ' then one can be constructed satisfying σ. A reduction is *allowable* if it has this property.

The first class of reductions is that which removes "duplicate" adjacent relations from the arrangement.

Lemma 23 *For any $X \in \{\text{RR}, \text{RL}, \text{LR}, \text{LL}\}$, the reduction $X\ X \to X$ is allowable.*

Proof Let σ and σ' be any pair of arrangements such that σ' is derivable from σ by a single application of the reduction $X\ X \to X$. If there exists a polygon P' satisfying σ' then there is also a polygon P satisfying σ, as follows: construct P from P' by replacing the edge e of P' that satisfies X with two edges e_i and e_j that satisfy $X\ X$, as shown in Figure 7.6. □

Figure 7.6: Replacing One Edge in P' With Two Edges in P

Before continuing with more reductions, some things must be defined relating to the polygon on the sphere. Let a and b be the two points on the surface of the sphere that we are comparing to the polygon edges. Let the great circle passing through a and b define an equator ℓ, an "upper hemisphere" above ℓ, and a "lower hemisphere" below ℓ, as shown in Figure 7.7.

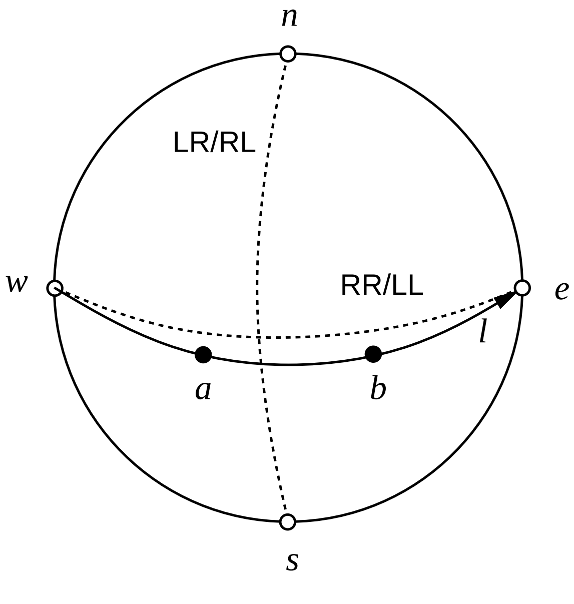

Figure 7.7: Definitions on the Sphere

Let the sphere center be v. Define two points e and w on ℓ such that e, v, and w are collinear and the distance $|w - a|$ is the same as the distance $|b - e|$. Define two points n and s as the north and south poles of the sphere.

In the following polygon constructions, the edges satisfying LR and RL will be restricted to lie on great circles that pass close to n and s, and between a and b, as shown in Figure 7.7. The edges satisfying LL and RR will be restricted to lie on great circles that pass close to e and w, and on either side of a and b.

Recall that $\Sigma = \{\text{RR}, \text{RL}, \text{LR}, \text{LL}\}$. In the following lemmas, let α and β be any string in Σ^*, and let X and Y be any symbol in Σ.

The next class of allowable reductions are those which remove an "opposite" pair of relations from the arrangement. For $X \in \Sigma$, define its opposite, $\overline{X}$, as follows:

X	$\overline{X}$
LR	RL
RR	LL
RL	LR
LL	RR.

Lemma 24 *For any $X \in \{\text{RR}, \text{RL}, \text{LR}, \text{LL}\}$, the reduction $X\,\overline{X} \to \emptyset$ is allowable.*

Proof (For LL RR $\to \emptyset$*. The other three reductions are similar.)*

Suppose a polygon P' exists that satisfies some arbitrary $\sigma' = \alpha\ X\ Y\ \beta$. Then a polygon P can be constructed to satisfy $\sigma = \alpha\ X$ LL RR $Y\ \beta$, as follows: Let edges e_x and e_y in P' satisfy relations X and Y in σ'. Let $x = e_x \cap e_y$. Assume for now that x is in the upper hemisphere. Figure 7.8 shows the approximate directions in which each edge must point to satisfy its particular relation if x lies in the upper hemisphere. Recall that the edges lie on great circles that pass close to n and s (for LR and RL edges) and close to e and w (for RR and LL edges).

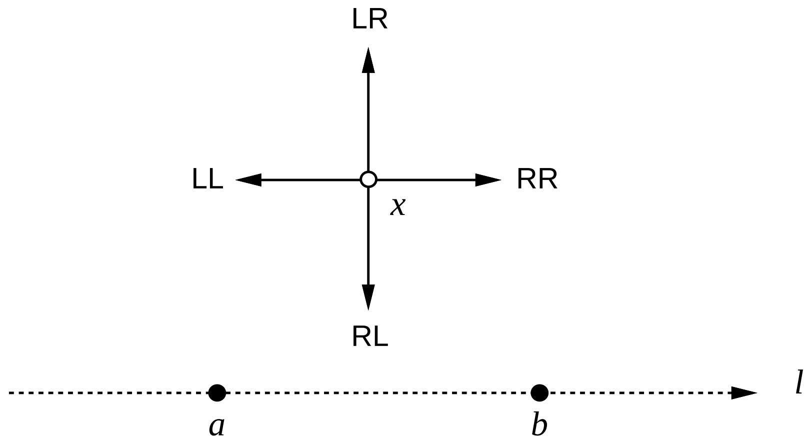

Figure 7.8: Edge Directions

The chart in Figure 7.9 shows, for each X and Y, how to transform edges satisfying $X\ Y$ into edges satisfying X LL RR Y. For example, in row RR and column LR of Figure 7.9 the edges satisfying RR LR are transformed into edges satisfying RR LL RR LR. This is shown in detail in Figure 7.10. Since such a transformation is possible in all contexts (as demonstrated by the chart of Figure 7.9), any polygon P' satisfying σ' can be transformed into a polygon P satisfying σ, and the reduction LL RR $\to \emptyset$ is allowable in all contexts.

Note

The transformation sometimes requires small perturbations of the original edges of P'. These perturbations can be made arbitrarily small so that no other part of the polygon is affected. Furthermore, the added edges can be made to lie on great circles that pass arbitrarily close to e and w (for LL and RR edges) or arbitrarily close to n and s (for LR and RL edges).

In the cases where X and Y are "opposites" (like LL-RR, RR-LL, LR-RL, RL-LR) the corresponding edges in P' make a turn of $\pi - \varepsilon$ radians to the right *or* to the left, as shown in Figure 7.11. Both cases must be considered when transforming P' into P. The table of Figure 7.9 is constructed so that the right turning case is given in row X, column Y, and the left turning case is given in the same row in the column labeled "left turn". For example, in row RL, column LR the right turning case is given; in the same row in the "left turn" column the left turning case is given. This is true of each row.

Thus the reduction LL RR $\rightarrow \emptyset$ is allowable. The remaining three reductions,

$$\begin{array}{lcl} \text{RR LL} & \rightarrow & \emptyset \\ \text{LR RL} & \rightarrow & \emptyset \\ \text{RL LR} & \rightarrow & \emptyset, \end{array}$$

are likewise allowable by symmetry. Thus each of the reductions is allowable under the initial assumption that the point x is in the upper hemisphere. If x is in the *lower* hemisphere then the same transformations apply, but they appear as though reflected about a vertical line. □

Lemma 25 *For any $X, Y \in \{\text{RR}, \text{RL}, \text{LR}, \text{LL}\}$, the reduction $X\ Y\ X \rightarrow X$ is allowable.*

Proof

Case 1: $Y = X$

In this case the allowable reduction $X\ X \rightarrow X$ can be applied twice:

$$X\ X\ X \rightarrow X\ X \rightarrow X.$$

Case 2: $Y = \overline{X}$

In this case the allowable reduction $X\ \overline{X} \rightarrow \emptyset$ can be applied once:

$$X\ \overline{X}\ X \rightarrow X.$$

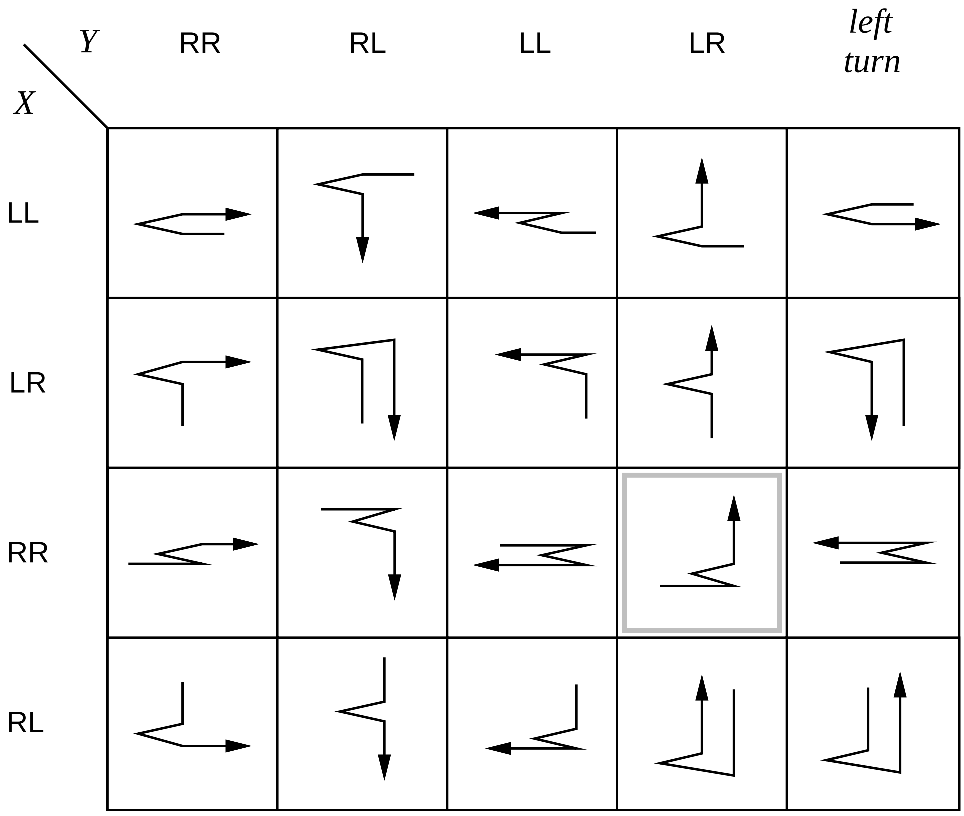

Figure 7.9: Adding LL RR Edges In All Contexts X and Y

Figure 7.10: Changing RR LR Edges Into RR LL RR LR Edges

Figure 7.11: Right and Left Turning Cases

Case 3:

Otherwise, X is in one of the two sets $\{$LR, RL$\}$ or $\{$LL, RR$\}$ and Y is in the other. Suppose X = RR and Y = LR (the other cases are similar). If there exists a polygon P' satisfying $\sigma' = \alpha$ RR β, then a polygon P satisfying $\sigma = \alpha$ RR LR RR β can be constructed as follows: Replace the edge e satisfying LR in P' with three edges satisfying RR LR RR, as shown in Figure 7.12.

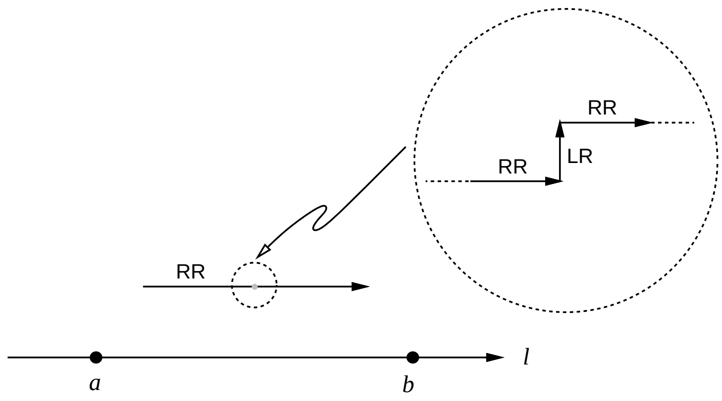

Figure 7.12: Replacing RR In P' With RR LR RR in P

The other seven instantiations of X and Y are similar. In each case an edge of P' is replaced by three edges, the middle one of which is arbitrarily short and approximately perpendicular to the other two. Thus the reduction $X\ Y\ X \rightarrow X$ is allowable. □

The three allowable reductions presented above are sufficient for our purposes. The allowable reductions are restated below.

$$\begin{array}{rcl} X\ X & \rightarrow & X \\ X\ \overline{X} & \rightarrow & \emptyset \\ X\ Y\ X & \rightarrow & X \end{array}$$

Define an *irreducible* arrangement as one to which none of the preceding reductions apply. It remains to characterize these irreducible arrangements.

Lemma 26 *The only irreducible arrangements of length three are*

$$
\begin{array}{l}
(\text{LR}, \quad \text{RR}, \quad \text{RL}),\\
(\text{RR}, \quad \text{RL}, \quad \text{LL}),\\
(\text{RL}, \quad \text{LL}, \quad \text{LR}),\\
(\text{LL}, \quad \text{LR}, \quad \text{RR}),\\
\\
(\text{LR}, \quad \text{LL}, \quad \text{RL}),\\
(\text{LL}, \quad \text{RL}, \quad \text{RR}),\\
(\text{RL}, \quad \text{RR}, \quad \text{LR}), \quad \text{and}\\
(\text{RR}, \quad \text{LR}, \quad \text{LL}).
\end{array}
$$

Proof By inspection. □

For any irreducible arrangement σ of length greater that three, every *subsequence* of length three, $(\sigma_i, \sigma_{i+1}, \sigma_{i+2})$, must also be irreducible. This leads to the following lemma.

Lemma 27 (Irreducible Arrangements)
An arrangement $\sigma \in \Sigma^*$ of length greater than three is irreducible if and only if

1. *σ is a subsequence of* $(\text{LR RR RL LL})^\infty$ *or of* $(\text{LR LL RL RR})^\infty$, *and*
2. *$|\sigma| = 4t$ for some integer t.*

Proof
(If) Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{4t})$. By inspection, every subsequence $(\sigma_i, \sigma_{i+1}, \sigma_{i+2})$ is one of the irreducible subsequences of Lemma 26, so none of our three allowable reductions can be applied and σ is irreducible.

(Only If, Part 1) If $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is irreducible then so are the subsequences $\sigma_a = (\sigma_i, \sigma_{i+1}, \sigma_{i+2})$ and $\sigma_b = (\sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3})$ for all $i \in \{1 \ldots n\}$. But σ_a and σ_b share σ_{i+1} and σ_{i+2}, and the only such irreducible subsequences are adjacent pairs of the two groups shown in Lemma 26. Thus σ must be a subsequence of $(\text{LR RR RL LL})^\infty$ or of $(\text{LR LL RL RR})^\infty$.

(Only If, Part 2) If, for example, $\sigma = (\text{LR}, \text{RR}, \text{RL}, \text{LL}, \ldots, \sigma_n)$ is irreducible, then so is any length three subsequence of σ. In particular, the subsequence $(\sigma_n, \text{LR}, \text{RR})$ is irreducible (recall that σ_n is adjacent to σ_1 in the arrangement). From inspection of the eight irreducible subsequences in Lemma 26, it must be that $\sigma_n = \text{LL}$. Generalizing this to any irreducible σ, it must be that $\sigma_n = \sigma_4$. Thus $|\sigma| = 4t$ for some integer t.
□

Now that irreducible arrangements of length greater than three have been characterized, it remains to show that polygons can be constructed satisfying those arrangements. Since the reductions may also result in arrangements of length two or length three, we must also show that polygons can be constructed satisfying those short arrangements, too. Arrangements of length less than two don't need to be considered, since the reductions remove at most two relations at a time and we can choose not to further reduce arrangements of length two and three.

Once it has been shown that satisfying polygons exist for arrangements of length two and three, and for irreducible arrangements of length greater than three, we will be able to conclude that satisfying polygons exist for *all* arrangements of length two or greater. This follows immediately from the property that the reductions are allowable.

Lemma 28 *There is a satisfying polygon for any $\sigma \in \Sigma^2$.*

Proof Construct the polygon as follows. Pick a point x on the surface of the sphere and place two edges at this point satisfying σ. Then extend the edges until they meet on the opposite side of the sphere to form a closed, simple polygon. □

Lemma 29 *There is a satisfying polygon for any $\sigma \in \Sigma^3$.*

Proof Construct the polygon for $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ as follows. Place a short initial edge satisfying σ_2 anywhere on the sphere, then place edges at its head and tail satisfying σ_3 and σ_1 and extend these edges until they meet. For a sufficiently short initial edge the resulting polygon is closed and simple. □

Lemma 30 *There is a satisfying polygon for any irreducible $\sigma \in \Sigma^*$, $|\sigma| > 3$.*

Proof
By Lemma 27, σ is a subsequence of either $(\text{LR RR RL LL})^\infty$ or $(\text{LR LL RL RR})^\infty$, and has length $4t$ for some integer t.

Assume that $\sigma = (\text{LR RR RL LL})^{4t}$ (the other subsequences of $(\text{LR RR RL LL})^\infty$ are cyclic permutations of σ). Then a polygon can be constructed by joining $4t$ chains of the form shown in Figure 7.13 and extending the head of the chain around the far side of the sphere until it meets the tail. An example is shown in Figure 7.14. The edges may be made arbitrarily short, so as many chains as necessary can be placed between a and b. The same argument applies if $\sigma = (\text{LR LL RL RR})^{4t}$, but the chain of Figure 7.13 must be reflected in a vertical line. □

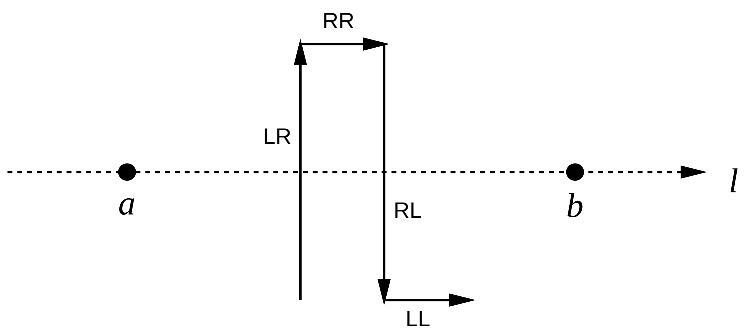

Figure 7.13: Chain Satisfying LR RR RL LL

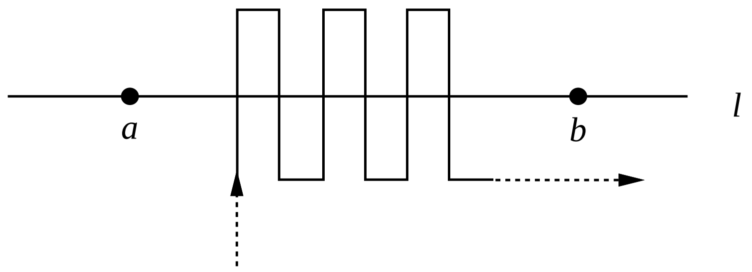

Figure 7.14: Polygon Satisfying (LR RR RL LL)^3

This leads to the Nondegenerate Arrangement Theorem. Recall that Φ^* is the set of point/plane relations corresponding to the set of point/edge relations Σ^*.

Theorem 4 (Nondegenerate Arrangement)

For any complete set of relations $R \subseteq R_{nondeg}$, there exist a cone and two points that satisfy R.

Proof

This is equivalent to showing that for any arrangement $\rho \in \Phi^*, |\rho| \geq 2$, there exist a cone and two points that satisfy ρ. From the definition of the cone in Section 7.2, we know that $|\rho| > 2$.

Invoke the Sphere Reduction Lemma and solve the equivalent problem on the sphere. That is, construct a polygon satisfying any $\sigma \in \Sigma^*$, $|\sigma| \geq 2$, as follows:

1. Apply the allowable reductions enumerated above to reduce σ to σ' until either $|\sigma| < 4$ or σ is irreducible.
2. If $|\sigma| < 4$ use Lemma 28 or Lemma 29 to construct a polygon P' satisfying σ'.
3. If σ is irreducible, use Lemma 30 to construct a polygon P' satisfying σ'.
4. From P' satisfying σ' construct a polygon P satisfying σ with the transformations of Lemmas 23, 24, and 25.

Since only allowable reductions were used to get σ' from σ, and since a polygon P' exists that satisfies σ', a polygon P exists that satisfies σ.

□

In conclusion, a cone and two points can be arbitrarily arranged, as long as no point is placed ON any plane defined by a face of the cone. Thus the rules of Table 7.1 characterize those nondegenerate arrangements of a cone with two points that have a model.

7.5 Degenerate Arrangements of a Cone and Two Points

Recall that we have restricted the input of the arrangement algorithm to a cone and two points with the features

$$F = \{a, b, v, p_1, p_2, \ldots, p_n, \ell_1, \ell_2, \ldots, \ell_n\}.$$

Recall that the set of allowable relations among these features is

$$\begin{aligned} R_{allow} = \{ \quad & v \text{ ON } p_i, \\ & \ell_i \text{ ON } p_i, \\ & \ell_i \text{ ON } p_{i+1}, \\ & a \text{ INSIDE } p_i, \;\; a \text{ ON } p_i, \;\; a \text{ OUTSIDE } p_i, \\ & b \text{ INSIDE } p_i, \;\; b \text{ ON } p_i, \;\; b \text{ OUTSIDE } p_i, \\ & a \text{ ON } \ell_i, \;\; a \text{ OFF } \ell_i, \\ & b \text{ ON } \ell_i, \;\; b \text{ OFF } \ell_i, \\ & a \text{ ON } v, \;\; a \text{ OFF } v, \\ & b \text{ ON } v, \;\; b \text{ OFF } v, \\ & a \text{ ON } b, \;\; a \text{ OFF } b \;\; \}. \end{aligned}$$

The Nondegenerate Arrangement Theorem showed that a model exists for every complete set of relations $R \subseteq R_{allow}$ which does not contain the degenerate relations a ON p_i, b ON p_i, or a ON b. In this section we will consider the case in which R contains degenerate relations (but we will still assume that a OFF b).

Two rules will be added to the characterization of valid arrangements. The revised rules are shown in Table 7.2, in which x and y are points, ℓ is a line, p is a plane, and f is any feature of F.

Table 7.2: Arrangement Rules

$$\begin{array}{ll} \textbf{Rule 1}: & x \text{ ON } y \;\Rightarrow\; \text{LOC}(x,p) = \text{LOC}(y,p) \\ \textbf{Rule 2}: & x \text{ ON } f_1 \;\wedge\; x \text{ ON } f_2 \;\Leftrightarrow\; x \text{ ON } f_1 \cap f_2 \\ \textbf{Rule 3}: & x \text{ ON } f_1 \;\wedge\; x \text{ ON } f_2 \;\Rightarrow\; f_1 \cap f_2 \in F \\ \textbf{Rule 4}: & x, y, v \text{ ON } l \;\wedge\; v \text{ ON } p_i, p_j \;\wedge\; \ell \text{ OFF } p_i, p_j \\ & \quad \Rightarrow\; (\text{LOC}(x,p_i) = \text{LOC}(y,p_i) \;\Leftrightarrow\; \text{LOC}(x,p_j) = \text{LOC}(y,p_j)) \end{array}$$

Rules 1 and 2 were described in the previous section. Rule 3 states that no feature can lie ON two other features unless their intersection is also a feature in F. Rule 4 states that if points x and y share a line ℓ which pierces several planes at some vertex v, then x and y are either always on the same side of the planes, or always on opposite sides of the planes. This is shown in Figure 7.15, in which the outward pointing normals of the planes p_i and p_j are shown as arrows.

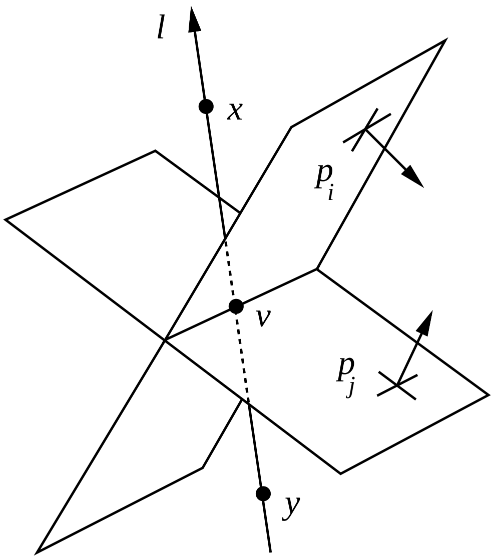

Figure 7.15: An Example of Rule 4

Rule 3 constrains the set of relations $R \subseteq R_{allow}$ such that each of the points a and b has one of the following relations to the planes of F:

1. the point is not ON any plane,
2. the point is ON exactly one plane, p_i,
3. the point is ON exactly two adjacent planes, p_i and p_{i+1}, and
4. the point is ON every plane.

In all other cases, the point would lie ON two nonadjacent planes, say p_i and p_j. Then by Rule 3, the line $p_i \cap p_j$ would have to be a feature of F. But the only lines in F are those defined by adjacent planes, $p_i \cap p_{i+1}$. So the point cannot lie ON two nonadjacent planes, and the four cases above are the only cases that are allowed by Rule 3.

We can now state the Arrangement Theorem, which will be proved following several preliminary results in the next section.

Arrangement Theorem

For any complete set of relations $R \subseteq R_{allow}$ that obeys the rules of Table 7.2, there exist a cone and two points that satisfy R.

7.6 The Arrangement Theorem

In this section we will prove that two points can be arbitrarily arranged with a cone. The reader can skip to the next section, and come back later if interest warrants.

Since either of the points a or b can be placed ON a plane, the set of point/plane relations must include the ON relation, and must be extended to

$$\begin{array}{llll}\Phi = \{ & (\text{INSIDE}, \text{INSIDE}), & (\text{INSIDE}, \text{ON}), & (\text{INSIDE}, \text{OUTSIDE}) \\ & (\text{ON}, \text{INSIDE}), & (\text{ON}, \text{ON}), & (\text{ON}, \text{OUTSIDE}) \\ & (\text{OUTSIDE}, \text{INSIDE}), & (\text{OUTSIDE}, \text{ON}), & (\text{OUTSIDE}, \text{OUTSIDE}) \; \}.\end{array}$$

For each complete set of relations $R \subseteq R_{allow}$ which satisfies the rules of Table 7.2, there is *exactly one* corresponding arrangement $\rho \in \Phi^*$ (since ρ is completely determined by the relations $a \bullet p_i$ and $b \bullet p_i$ of R, which themselves completely determine the other relations of R through the rules of Table 7.2).

Let $\mathcal{R} \subset \Phi^*$ be the set of all such arrangements. We must prove that every arrangement in $\mathcal{R}$ has a model that satisfies the corresponding set of relations, R.

The First Simplification

Let $\mathcal{R}_1 \subset \mathcal{R}$ be the set of arrangements in which neither of the points a or b lies ON every plane p_i defined by the faces of the cone (case 4 above). Then the next lemma will simplify matters slightly.

Lemma 31 (First Simplification)

If a satisfying model exists for every arrangement in $\mathcal{R}_1$, then one exists for every arrangement in $\mathcal{R}$.

Proof Let ρ be any arrangement in $\mathcal{R} - \mathcal{R}_1$. If, in the arrangement ρ, both a and b lie ON every plane, then a model exists trivially: construct a cone and place both a and b on its vertex.

Otherwise, assume that only a lies ON every plane. Then every component ρ_i of ρ is of the form (ON, X) for some $X \in \{\text{INSIDE}, \text{ON}, \text{OUTSIDE}\}$, since for every plane p_i, a ON p_i. Construct another arrangement, ρ', by replacing each such relation with (INSIDE, X). Then $\rho' \in \mathcal{R}_1$ (since neither a nor b is ON every plane in ρ').

Assume that a cone and two points exist which satisfy $\rho' \in \mathcal{R}_1$. Then the same cone satisfies the original ρ if the point b is left in position, and the point a is simply moved onto the cone vertex. $\square$

The Second Simplification

Let $\mathcal{R}_2 \subset \mathcal{R}_1$ be the set of arrangements in which the points a and b do not *both* lie ON an edge e_i of the cone. Then the next lemma simplifies matters again.

Lemma 32 (Second Simplification)

If a satisfying model exists for every arrangement in $\mathcal{R}_2$, then one exists for every arrangement in $\mathcal{R}_1$.

Proof Any arrangement in $\mathcal{R}_1 - \mathcal{R}_2$ has both of the points a and b lying ON a common edge e_i, and hence, ON two adjacent planes, p_i and p_{i+1}. Thus we can assume without loss of generality that for any $\rho \in \mathcal{R}_1 - \mathcal{R}_2$,

$$\rho = (\text{ON}, \text{ON}), (\text{ON}, \text{ON}), \rho_3, \rho_4, \ldots, \rho_n.$$

Construct another arrangement,

$$\rho' = (\text{INSIDE}, \text{ON}), (\text{INSIDE}, \text{ON}), \rho_3, \rho_4, \ldots, \rho_n.$$

Then $\rho' \in \mathcal{R}_2$ (since a lies ON no edge and b lies ON one edge in ρ').

Assume that a cone and two points exist which satisfy $\rho' \in \mathcal{R}_2$. Then the same cone satisfies the original ρ, as follows: since both points a and b lie ON a common edge in the arrangement ρ, Rule 4 of Table 7.2 applies and the relations of the two points to each plane of the cone (other than p_i and p_{i+1}) must be either always identical or always opposite in that arrangement ρ. If, in the model satisfying the arrangement ρ', the point a is moved onto the same line on which the point b lies, then a will have relations either identical to those of b (if a and b are on the same side of the cone vertex v), or opposite to those of b (if a and b are on opposite sides of v). Thus a model exists for any $\rho \in \mathcal{R}_1 - \mathcal{R}_2$. □

Thus it remains to prove that all arrangements in $\mathcal{R}_2$ have a satisfying model.

7.6.1 Reducing The Problem To $\Re^2$

Corresponding to the point/plane relations of Φ are the point/line relations

$$\Sigma = \{ \begin{array}{lll} (\text{RIGHT}, \text{RIGHT}), & (\text{RIGHT}, \text{ON}), & (\text{RIGHT}, \text{LEFT}) \\ (\text{ON}, \text{RIGHT}), & (\text{ON}, \text{ON}), & (\text{ON}, \text{LEFT}) \\ (\text{LEFT}, \text{RIGHT}), & (\text{LEFT}, \text{ON}), & (\text{LEFT}, \text{LEFT}) \; \}, \end{array}$$

which will be abbreviated as

$$\Sigma = \{\text{RR}, \text{RO}, \text{RL}, \text{OR}, \text{OO}, \text{OL}, \text{LR}, \text{LO}, \text{LL}\}.$$

Define the set of two dimensional arrangements $\mathcal{A}_2$ corresponding to those in $\mathcal{R}_2$ as

$$\mathcal{A}_2 = \{\sigma \mid \exists\, \rho \in \mathcal{R}_2, \;\; \sigma = \text{REDUCE}(\rho)\},$$

where the REDUCE function was defined in Section 7.4.1.

The Sphere Reduction Lemma can be trivially extended to include ON relations. By this extended lemma there exist a cone and two points which satisfy an arrangement $\rho \in \mathcal{R}_2$ if and only if there exist a polygon and two points on the sphere which satisfy the corresponding two dimensional arrangement, $\text{REDUCE}(\rho) \in \mathcal{A}_2$.

Thus it is sufficient to prove that every two dimensional arrangement in $\mathcal{A}_2$ has a model.

An arrangement in $\mathcal{A}_2$ may constrain each of the points a and b to lie ON certain edges of the polygon on the sphere. We will first divide the arrangements of $\mathcal{A}_2$ into two groups, based upon the number of common edges on which a and b both lie:

- a and b share no edge, and
- a and b share one edge.

Note that there is no arrangement in $\mathcal{A}_2$ in which a and b share more than two edges. Those arrangements would correspond to arrangements in $\mathcal{R}_2$ in which a and b share more than two planes. However, from the definition of $\mathcal{R}_2$ in the previous section, such arrangements are not present.

Each of the next sections will consider one of these cases.

7.6.2 Two Points Share No Edge

This section considers those arrangement in $\mathcal{A}_2$ in which the points a and b do not lie on any common edges. There are several subclasses of arrangements that must be dealt with here; these classes are characterized by the number of polygon edges on which each of a and b lie.

Each of the two points may lie ON zero, one, or two edges. If a point lies ON two edges then they must be adjacent (from the rules of Table 7.2), and hence the point lies ON a polygon vertex. Nine combinations of features can be enumerated and labeled as 00, 01, 02, 10, 11, 12, 20, 21, 22, where the first digit of the label is the number of edges that a is ON, and the second digit is the number of edges that b is ON.

These are demonstrated in Figure 7.16 where, for example, it is shown that arrangements in the 21 class are satisfiable by polygons which have one vertex touching the point a and one edge (or its extension) touching the point b. Note that the 00 class contains exactly the arrangements of the Nondegenerate Arrangement Theorem.

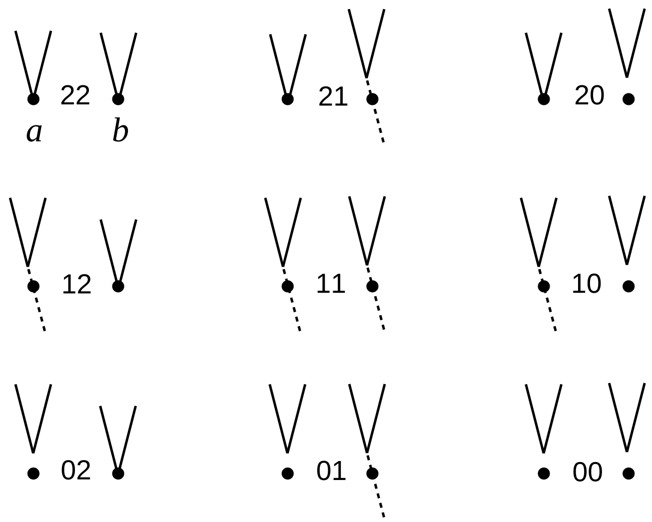

Figure 7.16: Two Points Share No Edge of the Polygon

Let $A = \{\text{OL}, \text{OR}\}$, $B = \{\text{LO}, \text{RO}\}$, and $\Psi = \{\text{LR}, \text{LL}, \text{RL}, \text{RR}\}$, and define

$$\begin{aligned}
\mathcal{C}_{00} &= \Psi^* \\
\mathcal{C}_{01} &= B\ \Psi^* \\
\mathcal{C}_{02} &= B\ B\ \Psi^* \\
\mathcal{C}_{10} &= A\ \Psi^* \\
\mathcal{C}_{11} &= A\ \Psi^* B\ \Psi^* \\
\mathcal{C}_{12} &= A\ \Psi^* B\ B\ \Psi^* \\
\mathcal{C}_{20} &= A\ A\ \Psi^* \\
\mathcal{C}_{21} &= A\ A\ \Psi^* B\ \Psi^* \\
\mathcal{C}_{22} &= A\ A\ \Psi^* B\ B\ \Psi^*
\end{aligned}$$

Define the set of all arrangements which share no edges as

$$\mathcal{C}_{noshare} = \mathcal{C}_{00} \cup \mathcal{C}_{01} \cup \mathcal{C}_{02} \cup \mathcal{C}_{10} \cup \mathcal{C}_{11} \cup \mathcal{C}_{12} \cup \mathcal{C}_{20} \cup \mathcal{C}_{21} \cup \mathcal{C}_{22}.$$

We must prove that each arrangement in $\mathcal{C}_{noshare}$ has a satisfying polygon.

Simplifying the Arrangements in $\mathcal{C}_{noshare}$

Consider the $\mathcal{C}_{22}$ and $\mathcal{C}_{21}$ classes of Figure 7.16. In any arrangement from $\mathcal{C}_{22}$ the point b lies on two adjacent edges of the satisfying polygon, whereas in any arrangement from $\mathcal{C}_{21}$ it lies on a single edge (or its extension). The polygons for $\mathcal{C}_{22}$ and $\mathcal{C}_{21}$ differ only in a single edge which has moved from being ON b in $\mathcal{C}_{22}$ to being OFF b in $\mathcal{C}_{21}$. It would seem that if we could satisfy some arrangement $\sigma_{22} \in \mathcal{C}_{22}$ then we could satisfy some *similar* arrangement $\sigma_{21} \in \mathcal{C}_{21}$ simply by perturbing a single edge of the polygon which satisfies σ_{22}.

This is captured in the next lemma. Here A is the set of those relations whose satisfying edges lie ON the point a and B is the set whose satisfying edges lie ON the point b.

Lemma 33 (Edge Perturbation)

Let $A = \{\text{OL}, \text{OR}\}, B = \{\text{LO}, \text{RO}\}, \Psi = \{\text{LR}, \text{LL}, \text{RL}, \text{RR}\}$, and let $\mathcal{S}$ be any subset of Σ^*. If there is a satisfying polygon for all $\sigma_a \in A\ \mathcal{S}$ or for all $\sigma_b \in B\ \mathcal{S}$, then there is also a satisfying polygon for all $\sigma \in \Psi\mathcal{S}$.

Proof Consider the case in which all $\sigma_a \in A\,\mathcal{S}$ have a satisfying polygon. If all $\sigma_a \in A\,\mathcal{S}$ have a satisfying polygon, then all $\sigma_{ol} \in \text{OL}\ \mathcal{S}$ and all $\sigma_{or} \in \text{OR}\ \mathcal{S}$ have a satisfying polygon. Consider one such σ_{ol}. Let P, a, and b be the polygon and two points on the sphere which satisfy σ_{ol} and let e_i be the edge of P satisfying OL. Let ℓ_i be the great circle defined by the edge e_i.

Consider Figure 7.17. Since P is simple (by the definition in Section 7.2) there is some small region around each endpoint of e_i in which the endpoint may be perturbed without affecting the simplicity of P. If these perturbations are restricted to lie on the great circles ℓ_{i-1} and ℓ_{i+1} defined by the adjacent edges e_{i-1} and e_{i+1}, then the directions of the adjacent edges will be unchanged, as will their relations to the points a and b.

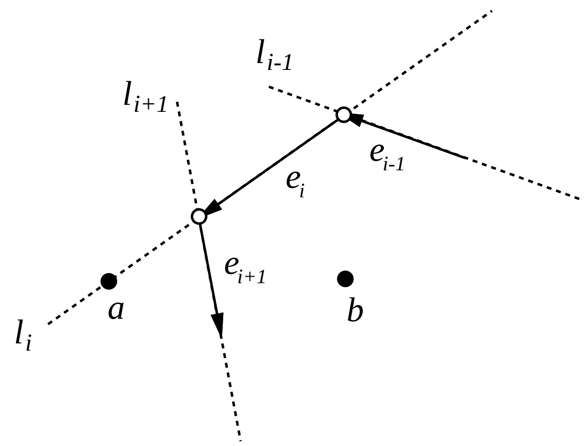

Figure 7.17: Perturbing the Endpoints of e_i

Since e_i satisfies OL the point a must lie ON the great circle ℓ_i. Arbitrarily small perturbations of the endpoints of e_i can cause a to lie to the RIGHT or LEFT of ℓ_i without changing the relation of the other point b to ℓ_i, since b is not ON ℓ_i. Thus, e_i, which originally satisfied OL, can be made to satisfy LL or RL. Furthermore, the perturbations of

$$\begin{array}{ccccc} 22 & \rightarrow & 21 & \rightarrow & 20 \\ \downarrow & & \downarrow & & \downarrow \\ 12 & \rightarrow & 11 & \rightarrow & 10 \\ \downarrow & & \downarrow & & \downarrow \\ 02 & \rightarrow & 01 & \rightarrow & 00 \end{array}$$

Figure 7.18: Reductions From $\mathcal{C}_{22}$

the endpoints of e_i can be chosen on ℓ_{i-1} and ℓ_{i+1} so as not to affect the remainder of the polygon.

Thus if there is a satisfying polygon for all arrangements in $\text{OL}\ \mathcal{S}$ there is also a satisfying polygon for all arrangements in $\{\text{LL}, \text{RL}\}\ \mathcal{S}$. By a similar argument, satisfying polygons for all arrangements in $\text{OR}\ \mathcal{S}$ yield satisfying polygons for all arrangements in $\{\text{LR}, \text{RR}\}\ \mathcal{S}$.

Thus if there is a satisfying polygon for all arrangements $\sigma_u \in A\ \mathcal{S}$, there is also a satisfying polygon for all arrangements $\sigma \in \Psi\mathcal{S}$. By a similar argument, if there is a satisfying polygon for all $\sigma_v \in B\ \mathcal{S}$, there is also a satisfying polygon for all $\sigma \in \Psi\mathcal{S}$. □

This result can be used to show that if satisfying polygons exist for all arrangements in $\mathcal{C}_{22}$, then satisfying polygons exist for all arrangements in each of the "lower" classes, with the proviso that the arrangements each have at least four relations.

Lemma 34 *If there is a satisfying polygon for every arrangement in $\mathcal{C}_{22}$ then there is also a satisfying polygon for every arrangement with at least four relations in $\mathcal{C}_{21}$, $\mathcal{C}_{20}$, $\mathcal{C}_{12}$, $\mathcal{C}_{11}$, $\mathcal{C}_{10}$, $\mathcal{C}_{02}$, $\mathcal{C}_{01}$, and $\mathcal{C}_{00}$.*

Proof We will prove that the $22 \rightarrow 21$ reduction can be done. Assume that all arrangements in $\mathcal{C}_{22}$ have a satisfying polygon. These are the arrangements $\sigma_{22} \in A\ A\ \Psi^* B\ B\ \Psi^*$. Then by Lemma 33 either of the two B's can be replaced by Ψ and there is a satisfying polygon for all arrangements $\sigma_{21} \in (A\ A\ \Psi^*\Psi B\ \Psi^*) \cup (A\ A\ \Psi^* B\ \Psi\Psi^*)$. Note that σ_{21} are those arrangements in $\mathcal{C}_{21}$ which have at least four relations. The other reductions are proved similarly.

Figure 7.18 shows the reduction steps. It can be seen that if all arrangements in $\mathcal{C}_{22}$ have a satisfying polygon, then all arrangements with at least four relations in the lower classes also have satisfying polygons. □

It remains to show that arrangements with fewer that four relations have satisfying polygons. A minimum of three edges is required for each polygon (by the definition in Section 7.2), so we only need to consider arrangements of three relations. Furthermore, if we can show that all length-three arrangements in $\mathcal{C}_{21}$ and $\mathcal{C}_{12}$ have satisfying polygons, then the reduction of Lemma 33 can be applied to get all "lower" classes.

Lemma 35 *Every arrangement of three relations in $\mathcal{C}_{21}$, $\mathcal{C}_{20}$, $\mathcal{C}_{12}$, $\mathcal{C}_{11}$, $\mathcal{C}_{10}$, $\mathcal{C}_{02}$, $\mathcal{C}_{01}$, $\mathcal{C}_{00}$ has a satisfying polygon.*

Proof It is sufficient to show that all arrangements of three relations in $\mathcal{C}_{21}$ have a satisfying polygon. By a symmetric argument all such arrangements in $\mathcal{C}_{12}$ have a satisfying polygon, and by the reduction of Lemma 33 all arrangements from the remaining lower classes have satisfying polygons.

Let $\tilde{\mathcal{C}}_{21}$ be the set of all arrangements with three relations: $\tilde{\mathcal{C}}_{21} = A\ A\ B$. Then for any arrangement $\sigma \in \mathcal{C}_{21}$ a satisfying polygon can be constructed by first placing edges e_1 and e_2 to satisfy the two ON relations of the point a. These edges will intersect at a. Then edge e_3 can be placed on the great circle passing through the point b and the head of e_2. Thus b will be ON e_3, and the direction of e_3 can be chosen to have a to its LEFT or its RIGHT. The head of e_3 and the tail of e_1 can be extended until they meet on the other side of the sphere, forming a simple polygon on the sphere. Thus a satisfying polygon exists for any $\sigma \in \tilde{\mathcal{C}}_{21}$, and all arrangements of three relations in $\mathcal{C}_{21}$, $\mathcal{C}_{20}$, $\mathcal{C}_{12}$, $\mathcal{C}_{11}$, $\mathcal{C}_{10}$, $\mathcal{C}_{02}$, $\mathcal{C}_{01}$, and $\mathcal{C}_{00}$ have a satisfying polygon. □

The next step is to prove that all arrangements in $\mathcal{C}_{22}$ are satisfiable.

Generating Satisfying Polygons For $\mathcal{C}_{22}$

Recall that the 22 class consists of those arrangements in the set

$$\mathcal{C}_{22} = A\ A\ \Psi^*\ B\ B\ \Psi^*,$$

where $A = \{\text{OL}, \text{OR}\}, B = \{\text{LO}, \text{RO}\}$, and $\Psi = \{\text{LR}, \text{LL}, \text{RL}, \text{RR}\}$. We want to prove that satisfying polygons exist for every arrangement in this class. Each arrangement σ in this class is of the form

$$\text{OX OX } \Psi^* \text{ XO XO } \Psi^*,$$

where each "X" can be replaced by either "R" or "L". In effect, there are sixteen subclasses (2^4) for which we must find satisfying polygons. We will give names to these subclasses as follows: let w, x, y, and z take on values in $\{\text{L}, \text{R}\}$ and define the complement of any variable x as

$$\bar{x} = \begin{cases} \text{R} & \text{if } x = \text{L} \\ \text{L} & \text{if } x = \text{R} \\ \text{O} & \text{if } x = \text{O} \end{cases}$$

and for each possible instatiation of w, x, y, and z define the subclass $\mathcal{C}_{wxyz}$ as

$$\mathcal{C}_{wxyz} = \text{O}w\ \ \text{O}x\ \ \Psi^*\ \ y\text{O}\ \ z\text{O}\ \ \Psi^*.$$

There are some simple reductions that show that satisfying polygons from some subclasses can be generated from the satisfying polygons of other subclasses.

Lemma 36 (Point Complementing)

If all arrangements in some class $\mathcal{C}$ are satisfiable, then so are all arrangements in $\mathcal{C}'$, defined as

$$\mathcal{C}' = \{\text{COMPLEMENT}_1(\sigma) | \sigma \in \mathcal{C}\} \cup \{\text{COMPLEMENT}_2(\sigma) | \sigma \in \mathcal{C}\},$$

where COMPLEMENT$_i(\sigma)$ *complements the* i^{th} *component of each relation in the arrangement* σ.

Proof We prove this for $\sigma' \in \{\text{COMPLEMENT}_1(\sigma) | \sigma \in \mathcal{C}\}$. A similar argument holds for $\sigma' \in \{\text{COMPLEMENT}_2(\sigma) | \sigma \in \mathcal{C}\}$. Take any arrangement $\sigma' \in \mathcal{C}'$ and form its "a-complement" σ by complementing the first component of each relation in σ. Then $\sigma \in \mathcal{C}$ and by the assumption there are two points a and b and a polygon on the sphere that satisfy σ. But if the point a is moved to the exact opposite position on the sphere then its relation to each edge of the polygon is reversed, and the new configuration satisfies the original arrangement $\sigma' \in \mathcal{C}'$. □

Corollary 1 *If all arrangements in some class $\mathcal{C}_{wxyz}$ are satisfiable, then so are all arrangements in $\mathcal{C}_{\bar{w}\bar{x}yz}$, $\mathcal{C}_{wx\bar{y}\bar{z}}$, and $\mathcal{C}_{\bar{w}\bar{x}\bar{y}\bar{z}}$.*

Lemma 37 (Point Exchanging)

If all arrangements in some class $\mathcal{C}$ are satisfiable, then so are all arrangements in $\mathcal{C}'$, defined as

$$\mathcal{C}' = \{\textsc{exchange}(\sigma) | \sigma \in \mathcal{C}\},$$

where $\textsc{exchange}(\sigma)$ *exchanges the first and second components of each relation in the arrangement σ.*

Proof Take any arrangement $\sigma' \in \mathcal{C}'$ and form its counterpart σ by exchanging the first and second components of each relation in σ'. Then $\sigma \in \mathcal{C}$ and by the assumption there are two points a and b and a polygon on the sphere which satisfy σ. But if the two points are exchanged then the new configuration satisfies the original arrangement $\sigma' \in \mathcal{C}'$. □

Corollary 2 *If all arrangements in some class $\mathcal{C}_{wxyz}$ are satisfiable, then so are all arrangements in $\mathcal{C}_{yzwx}$.*

With the Point Complementing Lemma and the Point Exchanging Lemma we can reduce the sixteen subclasses of $\mathcal{C}_{22}$ to only three subclasses.

Lemma 38 *If all arrangements in the subclasses $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$ are satisfiable, then all arrangements in $\mathcal{C}_{22}$ are satisfiable.*

Proof It is sufficient to show that each of the other thirteen subclasses can be reduced to one of these three subclasses through an application of either Corollary 1 or Corollary 2. This is shown in Figure 7.19, where the arrow superscripts a, b, and ab indicate point complementing of a, of b, and of both a and b. The arrow superscript e indicates point exchanging. □

From Lemma 38 we know that it is sufficient to prove that satisfying polygons exist for every arrangement in the subclasses $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$.

Satisfying the Arrangements in $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$

Consider the arrangements in $\mathcal{C}_{rrrr} =$ OR OR Ψ^* RO RO Ψ^*. These arrangements are equivalent to the "rotated arrangements" in OR Ψ^* RO RO Ψ^* OR, since a polygon satisfying an arrangement in one class will satisfy the corresponding "rotated arrangement" in the other class with a simple renaming of the edges. Thus we can prove satisfiablity for arrangements in OR Ψ^* RO RO Ψ^* OR, instead.

Consider the arrangements in OR Ψ^* RO RO Ψ^* OR. To show that all such arrangements are satisfiable, it is sufficient to show that for all strings $\alpha, \beta \in \Psi^*$ there is some chain of edges on the sphere that satisfy OR α RO, and another chain of edges that satisfy RO β OR, such that the two chains can be placed on the sphere to form a simple polygon satisfying OR Ψ^* RO RO Ψ^* OR.

A similar argument holds for the other subclasses $\mathcal{C}_{rrrl}$ and $\mathcal{C}_{lrrl}$. In order to show that all arrangements in the three subclasses are satisfiable, satisfying chains from each of the

$$
\begin{array}{lllll}
\mathcal{C}_{rrlr} & \xrightarrow{a} & \mathcal{C}_{rrrl} & & \\
\mathcal{C}_{rrll} & \xrightarrow{a} & \mathcal{C}_{rrrr} & & \\
\mathcal{C}_{lrlr} & \xrightarrow{a} & \mathcal{C}_{lrrl} & & \\
\mathcal{C}_{rlrl} & \xrightarrow{b} & \mathcal{C}_{lrrl} & & \\
\mathcal{C}_{llrr} & \xrightarrow{b} & \mathcal{C}_{rrrr} & & \\
\mathcal{C}_{llrl} & \xrightarrow{b} & \mathcal{C}_{rrrl} & & \\
\mathcal{C}_{lllr} & \xrightarrow{ab} & \mathcal{C}_{rrrl} & & \\
\mathcal{C}_{llll} & \xrightarrow{ab} & \mathcal{C}_{rrrr} & & \\
\mathcal{C}_{rlrr} & \xrightarrow{e} & \mathcal{C}_{rrrl} & & \\
\mathcal{C}_{rllr} & \xrightarrow{e} & \mathcal{C}_{lrrl} & & \\
\mathcal{C}_{rlll} & \xrightarrow{a} & \mathcal{C}_{rlrr} & \xrightarrow{e} & \mathcal{C}_{rrrl} \\
\mathcal{C}_{lrrr} & \xrightarrow{b} & \mathcal{C}_{rlrr} & \xrightarrow{e} & \mathcal{C}_{rrrl} \\
\mathcal{C}_{lrll} & \xrightarrow{ab} & \mathcal{C}_{rlrr} & \xrightarrow{e} & \mathcal{C}_{rrrl}
\end{array}
$$

Figure 7.19: Reductions in $\mathcal{C}_{22}$

following arrangements must be characterized in terms of their positions on the sphere:

OR Ψ^* RO,
RO Ψ^* OR,
LO Ψ^* OR, and
LO Ψ^* OL.

Consider, for example, Figure 7.20. If we could show that all satisfying chains for OR Ψ^* RO can be made to occupy some region R_1 of the sphere surface (R_1 includes the two edges satisfying OR and RO), and all satisfying chains for LO Ψ^* OL can be made to occupy some other region R_2 of the surface (R_2 includes the two edges satisfying LO and OL), then if $R_1 \cap R_2 = \{a, b\}$, satisfying polygons exist for every arrangement in OR Ψ^* RO LO Ψ^* OL. By a single right rotation, these polygons also satisfy every arrangement in the subclass $\mathcal{C}_{lrrl}$, so we would have proven that all arrangements in $\mathcal{C}_{lrrl}$ are satisfiable.

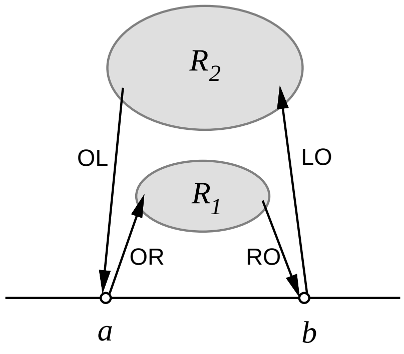

Figure 7.20: Regions For All Polygons Satisfying OR Ψ^* RO LO Ψ^* OL

Constraint Diagrams

The regions R_1 and R_2 described above are called *constraint regions*, since certain classes of satisfying polygons can be constrained to lie within them. A *constraint diagram* is the stereographic projection (onto a plane) of some constraint region on the surface of the sphere. A stereographic projection is a mapping of points on a sphere to points on a plane, as follows: a plane is placed on the equator of the sphere and a point x on the sphere is mapped to a point y on the plane if the three points x, y, and s (the south pole) are collinear.

From complex analysis, this is a *conformal mapping* and circles on the sphere map to circles on the plane. In particular, the equator is mapped to a circle on the plane, and all other great circles on the sphere are mapped to circles on the plane which intersect the image of the equator at two opposite points.

For an example of this mapping, consider Figure 7.21, which show a polygon on the sphere that satisfies the arrangement (LR RR RL LL)^3. The stereographic projection of this polygon is shown in Figure 7.22, which also shows the equator l, the two points a and b, and the two points a' and b' opposite a and b on the sphere.

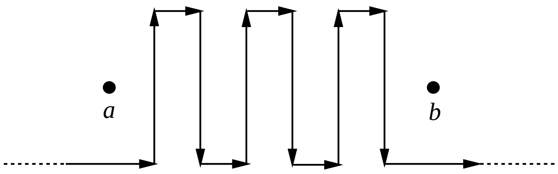

Figure 7.21: (LR RR RL LL)^3

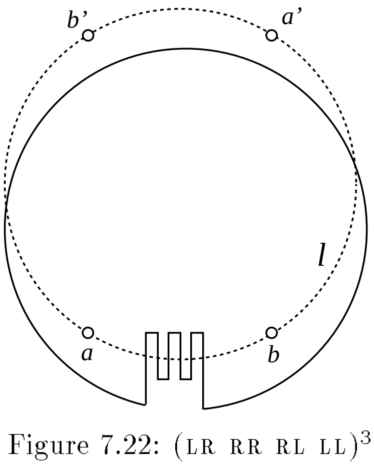

Figure 7.22: (LR RR RL LL)^3

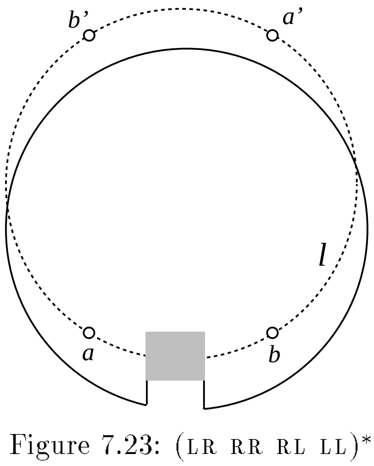

Figure 7.23: (LR RR RL LL)*

From Lemma 30, *any* arrangement in (LR RR RL LL)* can be satisfied by some polygon which has many "loops" between a and b, and which has one edge satisfying LL which extends around the far side of the sphere. Thus, one of the possible constraint regions for arrangements in (LR RR RL LL)* has the stereographic image shown in Figure 7.23, where as many loops as necessary are placed in the shaded region. This is just one of the many possible constraint diagrams for (LR RR RL LL)*.

The first constraint region to be constructed will be for the arrangements in the set OR Ψ^* RO. From the proof of Lemma 27, any arrangement in Ψ^* can be reduced to a subsequence of either (LR RR RL LL)$^\infty$ or (LR LL RL RR)$^\infty$.

Define a *right spiral* to be any subsequence of (LR RR RL LL)$^\infty$, and define a *left spiral* to be any subsequence of (LR LL RL RR)$^\infty$. Let the left turning symbol "↺" denote the set of all left spirals, and let the right turning symbol "↻" denote the set of all right spirals. Then we need to construct constraint regions for the two sets of arrangements OR ↻ RO and OR ↺ RO.

Constraint Regions For OR ↻ RO

There are four possible constraint regions for arrangements in OR ↻ RO, as will be proven in the next lemma. The first two constraint regions are shown in Figure 7.24 and Figure 7.25. Each contains exactly three edges satisfying OR LL RO. The third constraint region is shown in Figure 7.26, and contains edges satisfying all other sequences which don't contain RL LL LR as a subsequence. The fourth constraint region is shown in Figure 7.27 and contains edges satisfying all remaining sequences.

Lemma 39 *Any arrangement in* OR ↻ RO *can be satisfied by some chain of edges which lies in one of the constraint regions shown in Figures 7.24, 7.25, 7.26, or 7.27. A satisfying chain lies in the constraint region of Figure 7.24 if and only if one lies in the constraint region of Figure 7.25.*

Proof

Case 1: The constraint regions of Figure 7.24 and Figure 7.25 contain edges satisfying OR LL RO. This can be demonstrated by placing a single edge satisfying LL in the shaded part of either region.

Case 2: The longest arrangement in OR ↻ RO that doesn't contain RL LL LR is OR LL LR RR RL LL RO. Figure 7.28 shows how satisfying edges for this arrangement can be placed in the constraint region of Figure 7.26.

With the exception of OR LL RO, all other arrangements that don't contain relations RL LL LR can be satisfied by a subchain of the five edges which satisfy LL LR RR RL LL in the longest arrangement. Exactly the same edges are used as for the longest arrangement, and the edges satisfying OR and RO are formed by joining the tail of the chain to a (for OR) and by joining the head of the chain to b (for RO). For example, Figure 7.29 shows how to form a satisfying chain for OR RR RL LL RO.

Case 3: All remaining arrangements contain at least one subsequence RL LL LR. The longest arrangement which contains a *single* such subsequence is OR LL RL RR <u>RL LL LR</u> RR RL LL RO. Figure 7.30 shows how satisfying edges for this arrangement can be placed in the constraint region of Figure 7.27. As in Case 2, all *shorter* sequences containing RL LL LR can be realized with subchains of these edges.

All remaining arrangements contain n subsequences RL LL LR, where $n > 1$. Each such arrangement must contain $n-1$ adjacent subsequences RL LL LR RR. To construct a set of edges satisfying any such arrangement, first remove these $n-1$ subsequences. The resulting arrangement contains a *single* sequence RL LL LR and can be satisfied with an edge chain as described above. Then the $n-1$ subsequences are inserted between LL and LR in the subsequence RL LL LR, as demonstrated in Figure 7.31. Thus any arrangement which contains RL LL LR has satisfying edges which lie in the constraint region of Figure 7.27. □

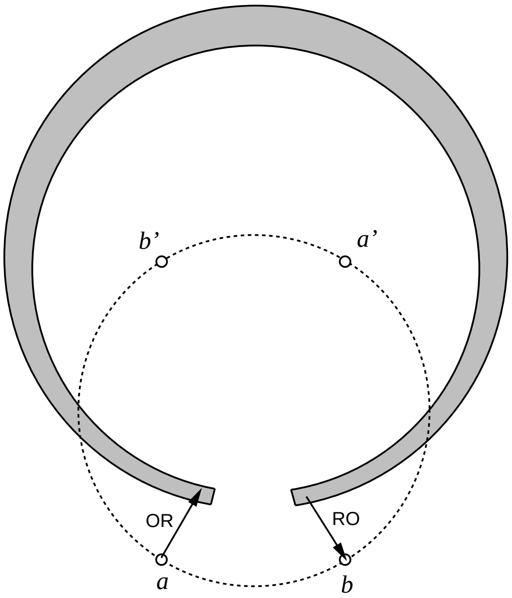

Figure 7.24: OR LL RO

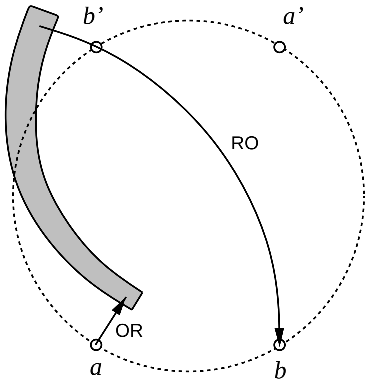

Figure 7.25: OR LL RO

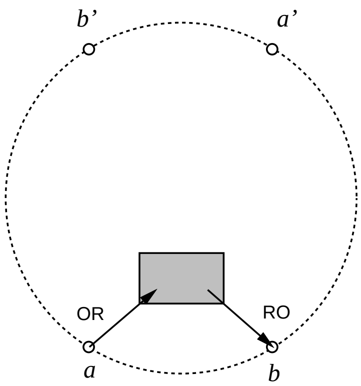

Figure 7.26: Without RL LL LR

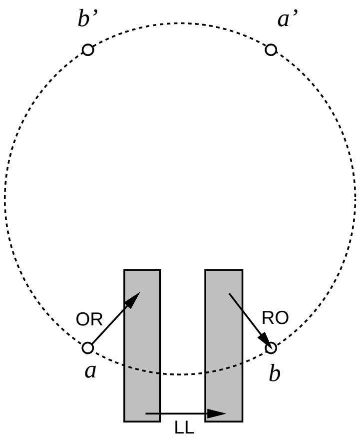

Figure 7.27: All Other Arrangements

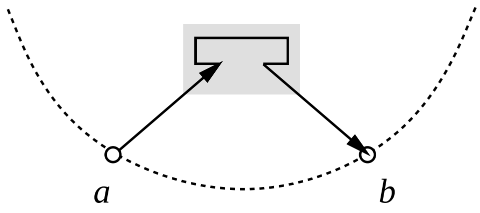

Figure 7.28: OR LL LR RR RL LL RO

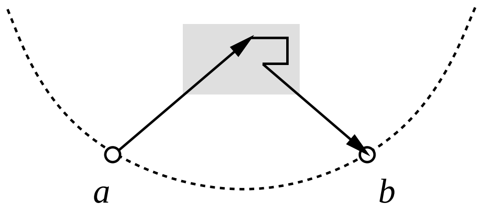

Figure 7.29: OR RR RL LL RO

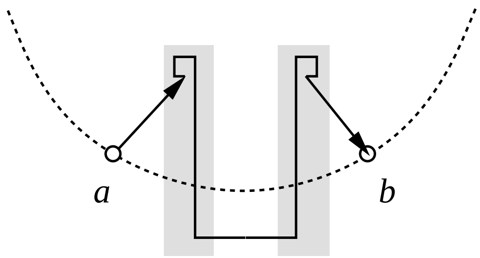

Figure 7.30: RL LL LR

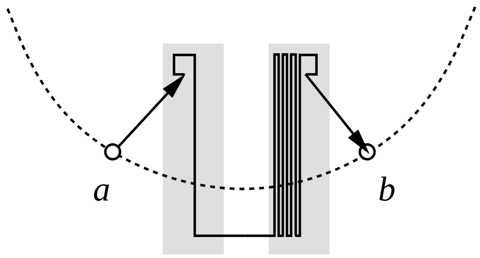

Figure 7.31: Adding (RR RL LL LR)*

Constraint Regions For OR ↺ RO

The same constraint regions are valid for arrangements in OR ↺ RO, as will be proven in the next lemma. The first constraint region is shown in Figure 7.26 and can be made to contain edges satisfying any left spiral which *doesn't* contain an LL relation. In Figures 7.24 and 7.25 are two constraint regions, either of which can be made to contain edges satisfying any left spiral which has at least one LL relation.

Lemma 40 *Any arrangement in* OR ↺ RO *can be satisfied by some chain of edges which lies in one of the constraint regions shown in Figures 7.24, 7.25 and 7.26. A satisfying chain lies in the constraint region of Figure 7.24 if and only if one lies in the constraint region of Figure 7.25.*

Proof

Case 1: The longest arrangement in OR ↺ RO which doesn't contain an LL relation is OR LR RR RL RO. Figure 7.32 shows how satisfying edges for this arrangement can be placed in the constraint region of Figure 7.26.

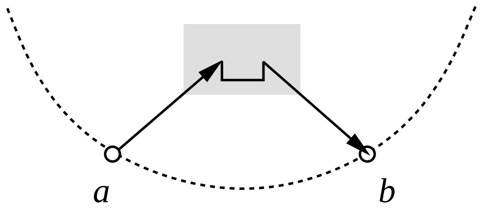

Figure 7.32: OR RL RR LR RO

As in Case 2 of Lemma 39, all *shorter* sequences not containing LL can be realized with subchains of these edges.

Case 2: All remaining arrangements contain at least one LL relation. The longest arrangement which contains a *single* LL relation is OR RL RR LR <u>LL</u> RL RR LR RO. Figure 7.33 shows how satisfying edges for this arrangement can be placed in the constraint region of Figure 7.24. As in Case 2 of Lemma 39, all *shorter* sequences containing at least one LL relation can be realized with subchains of these edges.

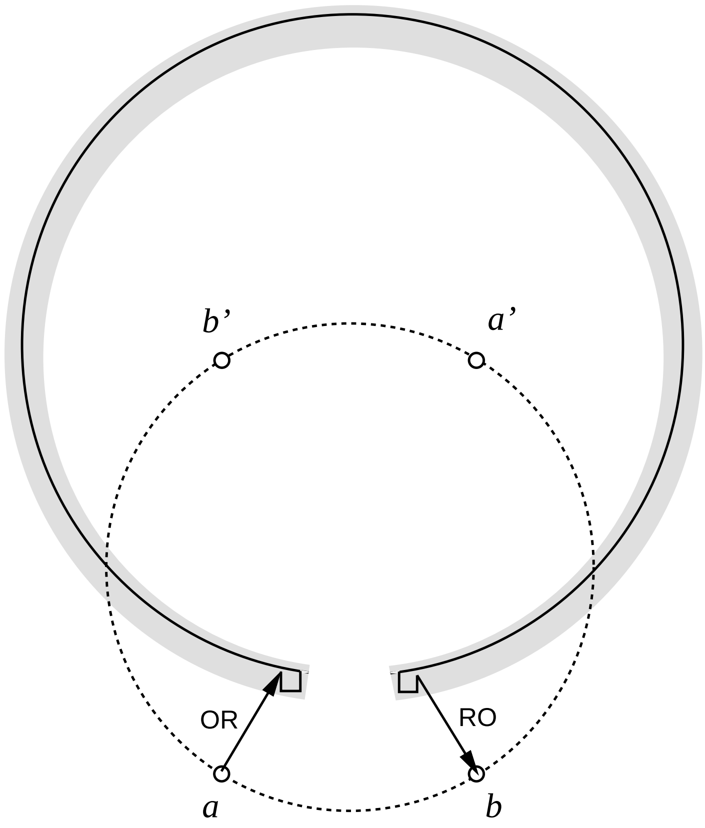

Figure 7.33: OR RL RR LR <u>LL</u> RL RR LR RO

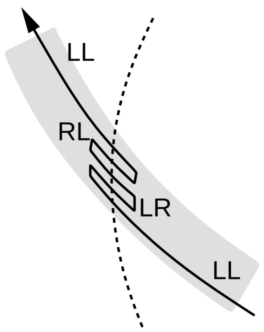

Figure 7.34: Adding (RL RR LR LL)*

All remaining arrangements contain n LL relations, where $n > 1$. Each such arrangement must contain $n-1$ adjacent subsequences RL RR LR LL. To construct a set of edges satisfying any such arrangement, first remove these $n-1$ subsequences. The resulting arrangement contains a *single* LL relation and can be satisfied with an edge chain as described above. Then the $n-1$ subsequences are inserted where the edge satisfying LL crosses the equator of the sphere, as demonstrated in Figure 7.34. Thus any arrangement which contains an LL relation has satisfying edges which lie in the constraint region of Figure 7.24.

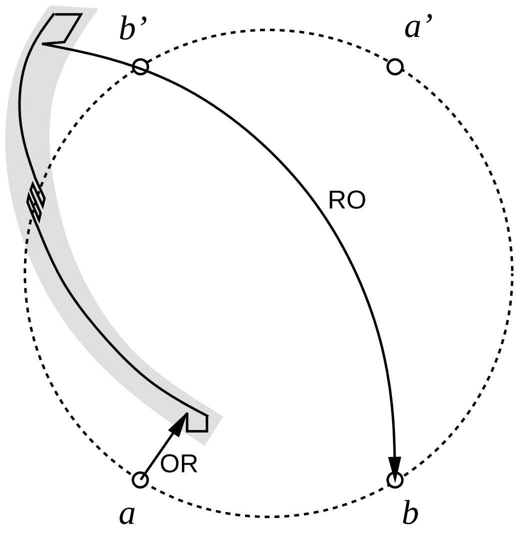

Figure 7.35: Adding (RL RR LR LL)*

By the same argument, all arrangements with one LL relation have satisfying edges which lie in the constraint region of Figure 7.25, as shown in Figure 7.35. □

Constraint Regions For LO Ψ^* OL

As demonstrated in Figure 7.36, an edge chain that satisfies some arrangement in OR Ψ^* RO is also a satisfying chain for some arrangement in LO Ψ^* OL if its edges are *reversed*.

This yields a one-to-one mapping between arrangements in each of these classes, as follows: an arrangement $(\phi_1, \phi_2, \ldots, \phi_n) \in$ OR Ψ^* RO is mapped to an arrangement $(\psi_1, \psi_2, \ldots, \psi_n) \in$ LO Ψ^* OL if $\phi_i = \text{REVERSE}(\psi_{n-i+1})$, where REVERSE($\psi$) replaces, in the relation ψ, each "L" with an "R" and each "R" with an "L". This leads to the following lemma.

Lemma 41 *The constraint regions of* LO Ψ^* OL *are exactly those of* OR Ψ^* RO.

Proof For any arrangement in LO Ψ^* OL, the above mapping yields an arrangement in OR Ψ^* RO which has a satisfying edge chain in one of the constraint regions of OR Ψ^* RO. When reversed, this edge chain satisfies the arrangement in LO Ψ^* OL. □

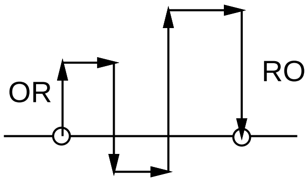

Figure 7.36: Corresponding Chains in OR Ψ^* RO and LO Ψ^* OL

Constraint Regions For RO Ψ^* OR

As demonstrated in Figure 7.37, an edge chain on the sphere which satisfies some arrangement in OR Ψ^* RO is also a satisfying chain for some arrangement in RO Ψ^* OR if it is *rotated* 180° about an axis which passes through the center of the sphere and comes out midway between the points a and b.

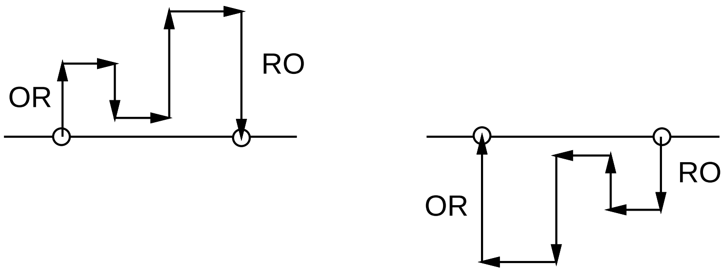

Figure 7.37: Corresponding Chains in OR Ψ^* RO and RO Ψ^* OR

As in the previous section, this yields a one-to-one mapping between arrangements in these two classes, as follows: an arrangement $(\phi_1, \phi_2, \ldots, \phi_n) \in$ OR Ψ^* RO maps to an arrangement $(\psi_1, \psi_2, \ldots, \psi_n) \in$ RO Ψ^* OR if $\phi_i = \text{ROTATE}(\psi_i)$, where $\text{ROTATE}(\psi)$ exchanges the first and second letters of the relation ψ. That is, LR and RL are exchanged, as are OR and RO, but LL and RR remain the same.

This means that the constraint regions of OR Ψ^* RO can be mapped to constraint regions of RO Ψ^* OR, simply by rotating them 180°. The constraint regions for RO Ψ^* OR are shown in Figures 7.38, 7.39, 7.40, and 7.41. Due to the nature of the stereographic projection they do not appear quite as one might expect. A few seconds of intense staring should convince the reader that they are the 180° rotations of the constraint regions of Figures 7.24, 7.25, 7.26, and 7.27.

Lemma 42 *Any arrangement in* RO Ψ^* OR *can be satisfied by some chain of edges which lies in one of the constraint regions shown in Figures 7.38, 7.39, 7.40, and 7.41. A satisfying chain lies in the constraint region of Figure 7.38 if and only if one lies in the constraint region of Figure 7.39.*

Proof The proof is identical to that of Lemmas 39 and 40, except that the relations LR and RL are exchanged and the relations OR and RO are exchanged throughout, and the figures used in Lemmas 39 and 40 are considered to be stereographic projections from the *north* pole rather than from the *south* pole. □

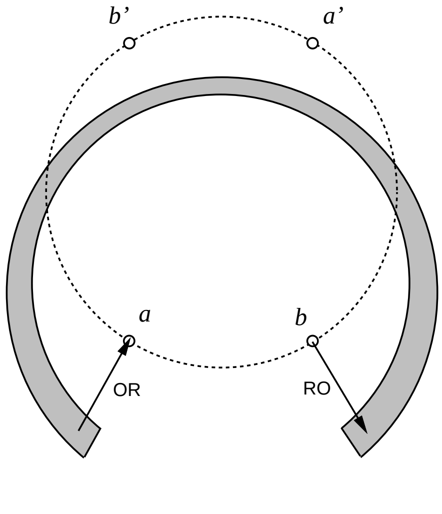

Figure 7.38: RO LL OR

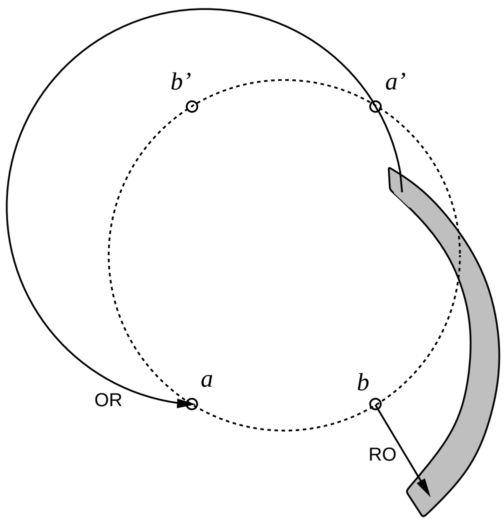

Figure 7.39: RO LL OR

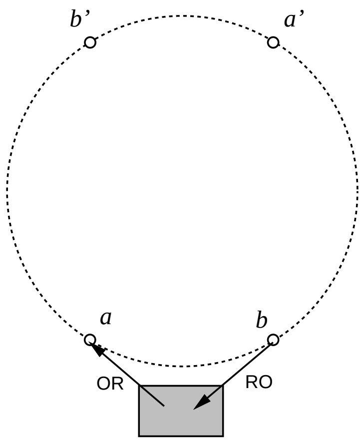

Figure 7.40: Without LR LL RL

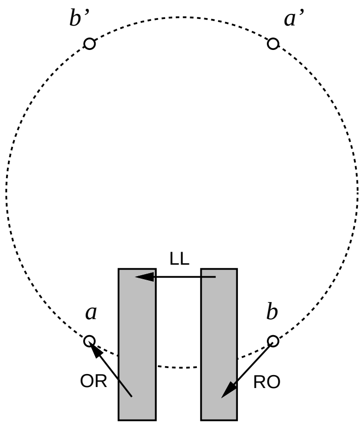

Figure 7.41: All Other Arrangements

Constraint Regions For LO ↻ OR

The arrangements in the LO Ψ^* OR are more complicated than the ones dealt with in the previous section; we must first consider the right spiral arrangements.

There are three possible constraint regions for arrangements in LO ↻ OR, as will be shown in the next lemma. The first constraint region is shown in Figure 7.42 and contains exactly three edges satisfying LO RL OR. The second constraint region, shown in Figure 7.43, contains edges satisfying all other arrangements which don't contain RR RL LL. The third constraint region, shown in Figure 7.44, contains edges satisfying all remaining arrangements.

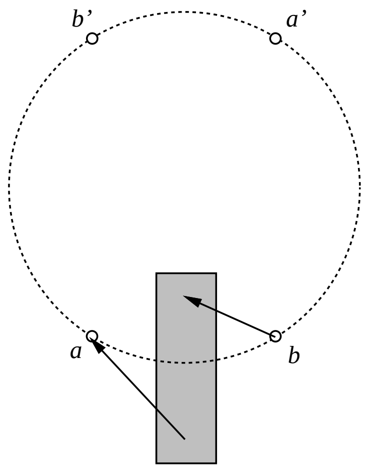

Figure 7.42: With RL

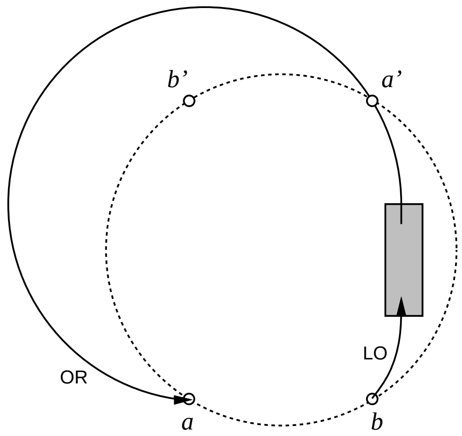

Figure 7.43: Without RR RL LL

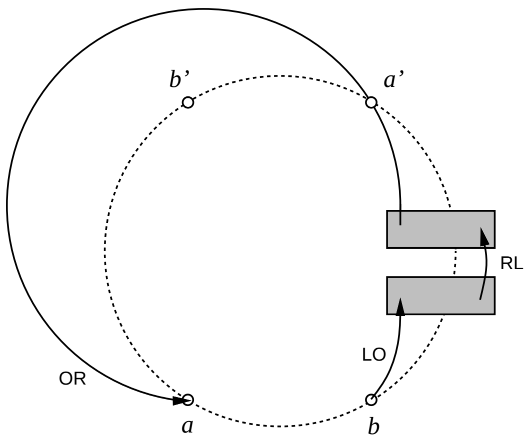

Figure 7.44: All Other Arrangements

Lemma 43 *Any arrangement in* LO $\circlearrowright$ OR *can be satisfied by some chain of edges which lies in one of the constraint regions shown in Figure 7.42, 7.43, or 7.44.*

Proof

The constraint region of Figure 7.42 contains edges satisfying LO RL OR. This is demonstrated by placing a single edge satisfying RL in the shaded part of the region.

Note that the constraint regions in Figures 7.43 and 7.44 are similar to those of Figures 7.26 and 7.27 in Lemma 39, except that they are rotated 90° counterclockwise and contain one extended edge. The proof of Lemma 39 also serves for this lemma, except that the figures must be rotated 90° counterclockwise and the relations must be replaced throughout as shown in Table 7.3. □

Table 7.3: Corresponding Relations

Lemma 39	RR	RL	LL	LR	OR	RO
Lemma 43	RL	LL	LR	RR	LO	OL

Constraint Regions For LO $\circlearrowleft$ OR

The constraint region of Figure 7.42 can be made to contain satisfying chains for all arrangements in LO $\circlearrowleft$ OR which contain at least one RL relation; the constraint region of Figure 7.43 handles arrangements without an RL relation.

Lemma 44 *Any arrangement in* LO $\circlearrowleft$ OR *can be satisfied by some chain of edges which lies in one of the constraint regions shown in Figure 7.42 or 7.43.*

Proof

Case 1: The longest arrangement in LO $\circlearrowleft$ OR which doesn't contain an RL relation is LO RR LR LL OR. It can be placed in the constraint region of Figure 7.43 as shown in Figure 7.45. As in Case 2 of Lemma 39, shorter arrangements not containing RL can be satisfied with the same edges.

Case 2: The longest arrangement in LO $\circlearrowleft$ OR which contains a *single* RL relation is LO RR LR LL <u>RL</u> RR LR LL OR. Figure 7.46 shows how a satisfying edge chain for this arrangement can be placed in the constraint region. As in Case 2 of Lemma 39, shorter arrangements containing a single RL can be satisfied with subchains of the same edges.

Any arrangement containing n RL relations ($n > 1$) must also contain $n-1$ subsequences RL RR LR LL. To construct an edge chain satisfying such an arrangement, the $n-1$ subsequences are removed and the resulting arrangement is constructed as described above. Then the $n-1$ subsequences can be inserted as shown in Figure 7.47. Thus any arrangement which contains the relation RL has satisfying edges which lie in the constraint region of Figure 7.42. □

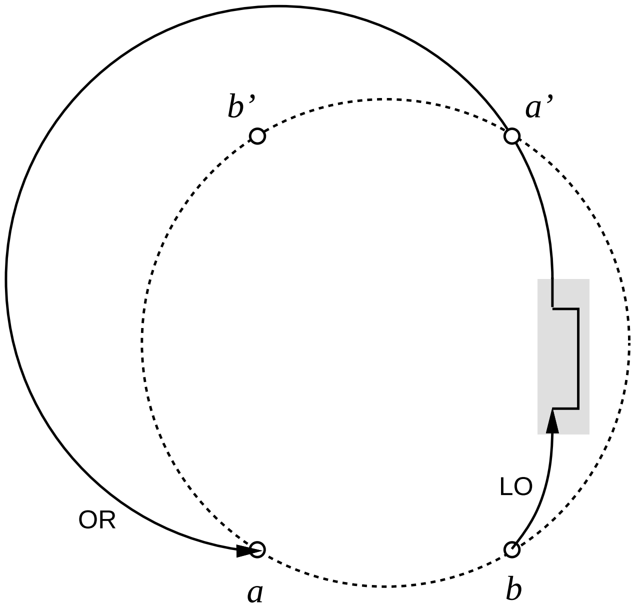

Figure 7.45: LO RR LR LL OR

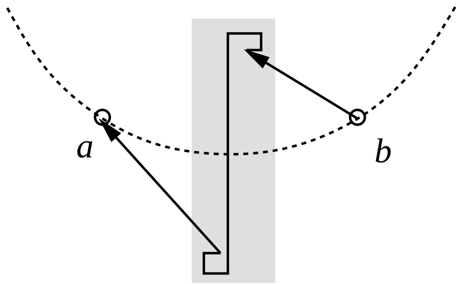

Figure 7.46: A Single RL Relation

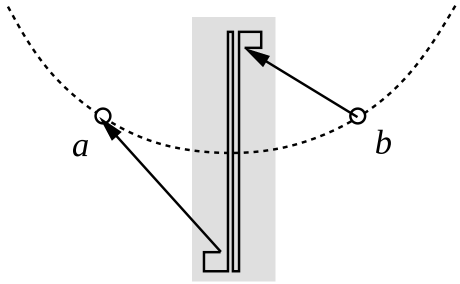

Figure 7.47: Adding (RL RR LR LL)*

Combining the Constraint Regions

Recall that the purpose of building the constraint regions is so that we can combine them to form constraint regions for each of the classes of arrangements $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$. These classes are

$$\begin{aligned} \mathcal{C}_{rrrr} &= \text{OR } \Psi^* \text{ RO RO } \Psi^* \text{ OR}, \\ \mathcal{C}_{rrrl} &= \text{OR } \Psi^* \text{ RO LO } \Psi^* \text{ OR}, \text{ and} \\ \mathcal{C}_{lrrl} &= \text{OR } \Psi^* \text{ RO LO } \Psi^* \text{ OL}. \end{aligned}$$

The arrangements in each of these classes can be considered as an arrangement in

$$\text{OR } \Psi^* \text{ RO}$$

concatenated with an arrangement in one of

$$\begin{aligned} &\text{RO } \Psi^* \text{ OR}, \\ &\text{LO } \Psi^* \text{ OR}, \text{ or} \\ &\text{LO } \Psi^* \text{ OL}. \end{aligned}$$

Consider every combination of two constraint regions, where one is a constraint region for OR Ψ^* RO and the other is a constraint region for one of RO Ψ^* OR, LO Ψ^* OR, or LO Ψ^* OL. If every such pair can be placed on the sphere such that they intersect at exactly the points a and b, then every arrangement in $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$ has a simple satisfying polygon on the sphere.

Lemma 45 *Every arrangement in $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$ has a simple satisfying polygon on the sphere.*

Proof
It is sufficient to show that every possible combination of two constraint regions R_1 and R_2 can be placed on the sphere such that they intersect at exactly a and b, where R_1 is a constraint region for OR Ψ^* RO and R_2 is a constraint region for one of RO Ψ^* OR, LO Ψ^* OR, or LO Ψ^* OL.

This is shown in Figures 7.48, 7.49 and 7.50. Some comments must be made about these figures: Since the edges of any satisfying polygon can be placed arbitrarily close together, we can shrink the shaded portions of the constraint regions to arbitrarily thin rectangles which will appear as lines in the constraint diagrams of Figures 7.48, 7.49 and 7.50. Furthermore, the satisfying regions can be perturbed as long as they maintain the same topology with repect to the equator and the four points a, b, a', and b'. Finally, for some arrangements, we can choose either of two constraint regions in which to place the satisfying edges (as proven in Lemmas 39 and 40). In Figures 7.48, 7.49 and 7.50 the choice of region will be evident.

From Figures 7.48, 7.49 and 7.50 it can be seen that every combination of constraint regions can be placed on the sphere such that they intersect at exactly a and b. Thus, every arrangement in $\mathcal{C}_{rrrr}$, $\mathcal{C}_{rrrl}$, and $\mathcal{C}_{lrrl}$ has a simple satisfying polygon on the sphere. □

Lemma 46 (Unshared Edge)
Every arrangement in $\mathcal{C}_{noshare}$ is satsfiable.

Proof This follows directly from Lemmas 34, 35, 38, and 45. □

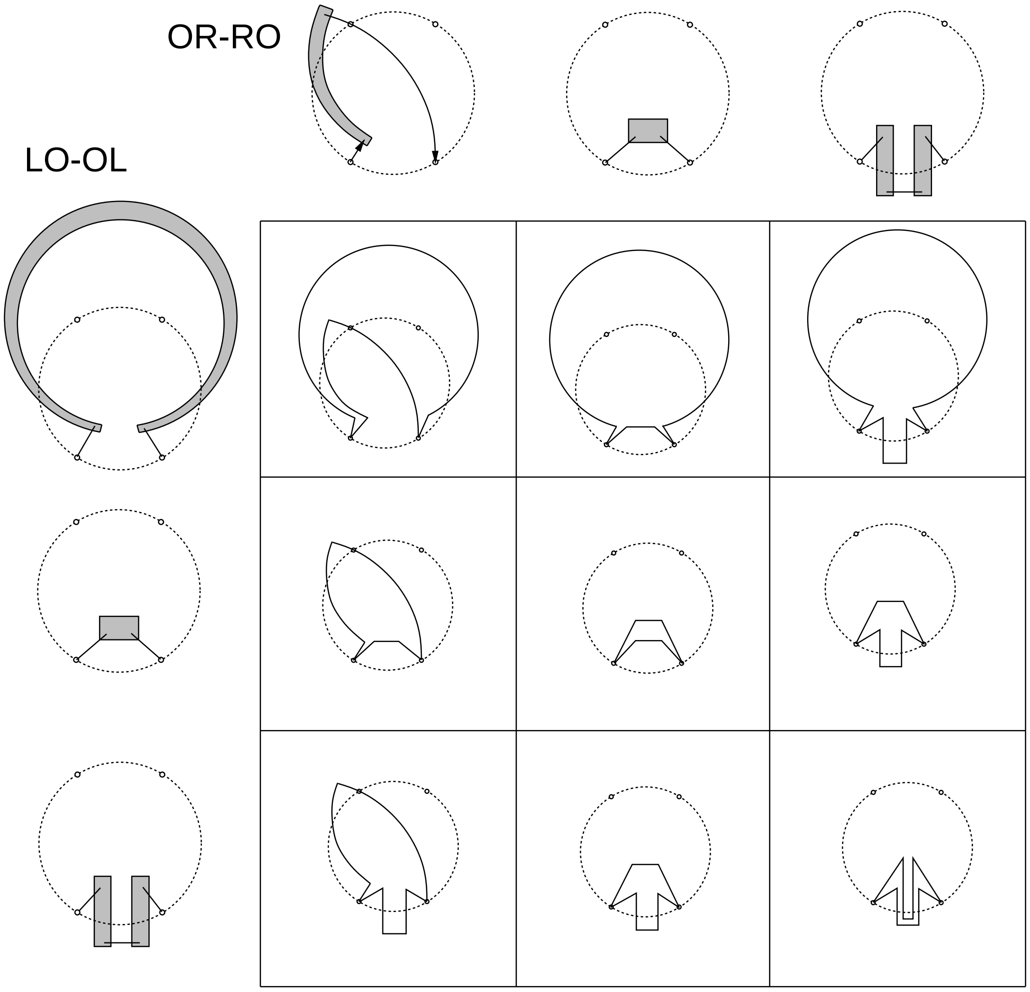

Figure 7.48: Satisfying regions for OR Ψ^* RO LO Ψ^* OL

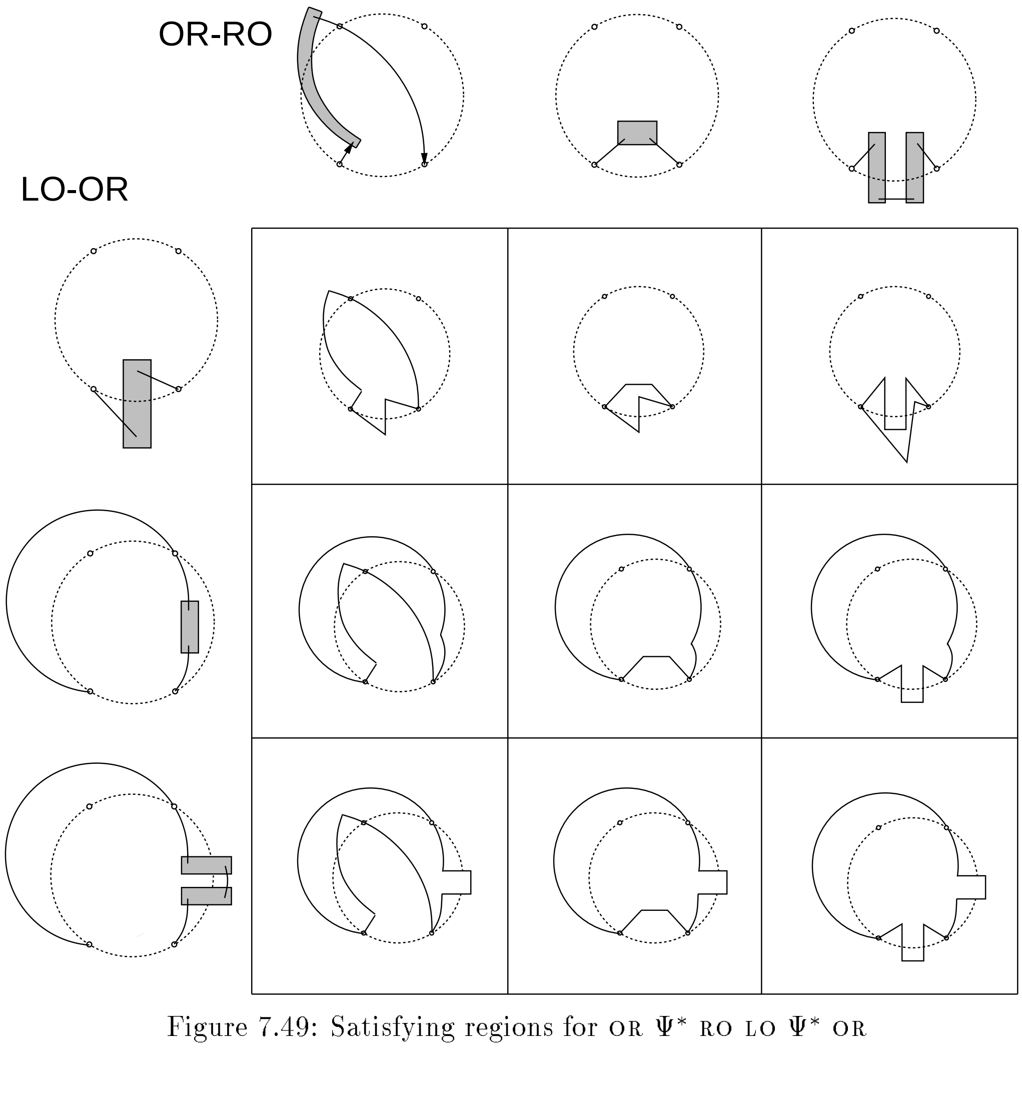

Figure 7.49: Satisfying regions for OR Ψ^* RO LO Ψ^* OR

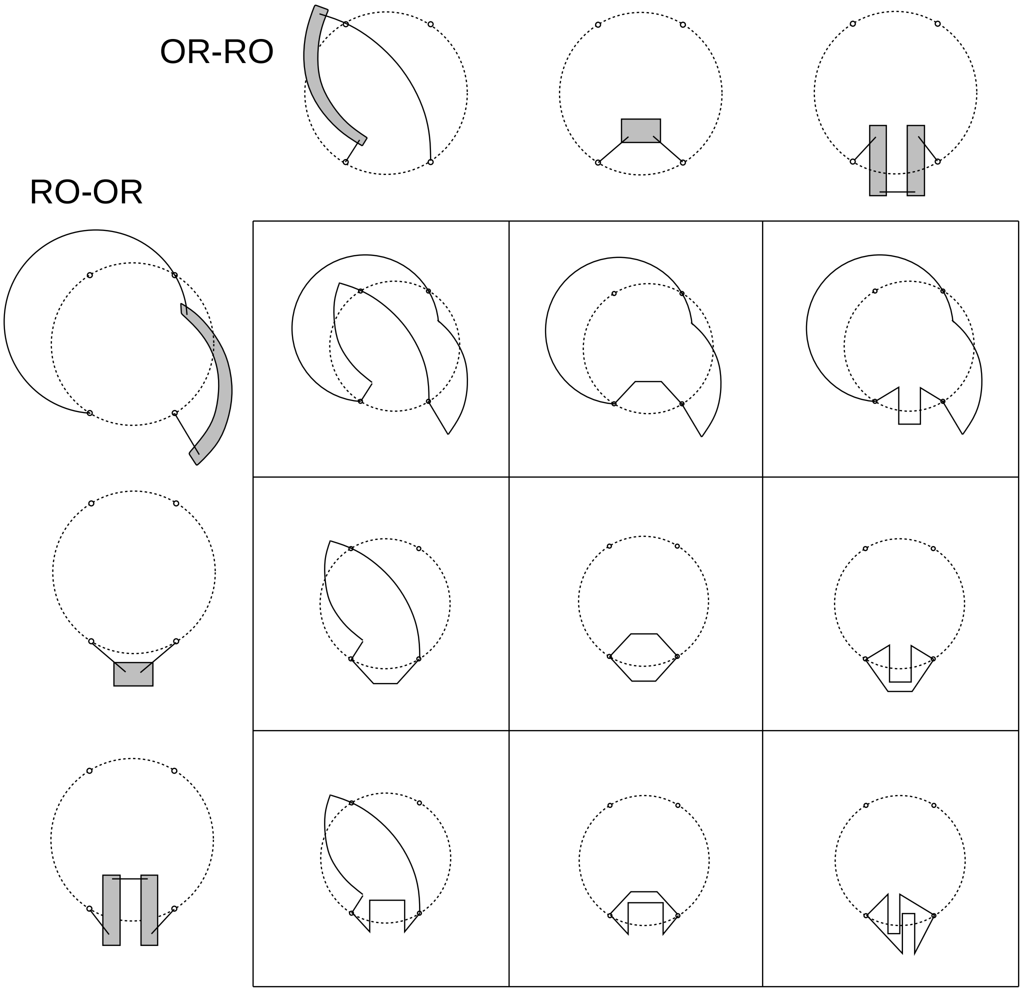

Figure 7.50: Satisfying regions for OR Ψ^* RO RO Ψ^* OR

7.6.3 Two Points Share One Edge

The previous section considered those arrangement in $\mathcal{A}_2$ in which the points a and b do not lie on any common edges. This section considers the other case: those arrangements in which the two points share a single edge.

Each such arrangement must contain exactly one OO relation, since there is exactly one edge on which both a and b lie. By Rule 3 of Table 7.2, the points a and b can each lie ON at most one other edge, and that edge must be adjacent to the common edge on which they both lie.

Let $A = \{\text{OL}, \text{OR}\}$, $B = \{\text{LO}, \text{RO}\}$, and $\Psi = \{\text{LR}, \text{LL}, \text{RL}, \text{RR}\}$, and define

$$\begin{aligned}
\mathcal{C}_{oo} &= \text{OO}\ \Psi^*, \\
\mathcal{C}_{a} &= \text{OO}\ B\ \Psi^* \ \cup\ B\ \text{OO}\ \Psi^*, \\
\mathcal{C}_{b} &= \text{OO}\ A\ \Psi^* \ \cup\ A\ \text{OO}\ \Psi^*, \text{and} \\
\mathcal{C}_{ab} &= A\ \text{OO}\ B\ \Psi^* \ \cup\ B\ \text{OO}\ A\ \Psi^*.
\end{aligned}$$

Then the set of all arrangements in which a and b share a polygon edge is

$$\mathcal{C}_{edge} = \mathcal{C}_{oo} \cup \mathcal{C}_a \cup \mathcal{C}_b \cup \mathcal{C}_{ab}.$$

By the Edge Perturbation Lemma, if there is a satisfying polygon for every arrangement in $\mathcal{C}_{ab}$ then there is also a satisfying polygon for every arrangement with at least three relations in $\mathcal{C}_{oo}$, $\mathcal{C}_a$, and $\mathcal{C}_b$. Furthermore, by the Point Exchanging Lemma, if all arrangements in $B\ \text{OO}\ A\ \Psi^*$ are satisfiable, then so are all arrangements in $A\ \text{OO}\ B\ \Psi^*$. Each arrangement in $B\ \text{OO}\ A\ \Psi^*$ is in one of the sets

$$\begin{aligned}
\mathcal{C}_{rr} &= \text{RO OO OR}\ \Psi^*, \\
\mathcal{C}_{rl} &= \text{RO OO OL}\ \Psi^*, \\
\mathcal{C}_{lr} &= \text{LO OO OR}\ \Psi^*, \quad \text{or} \\
\mathcal{C}_{ll} &= \text{LO OO OL}\ \Psi^*.
\end{aligned}$$

By the Point Complementing Lemma, if all arrangements in $\mathcal{C}_{rr}$ are satisfiable, then so are all arrangements in $\mathcal{C}_{rl}$, $\mathcal{C}_{lr}$, and $\mathcal{C}_{ll}$. So if a satisfying polygon exists for each arrangement in $\mathcal{C}_{rr}$, then a satisfying polygon exists for each arrangement in $\mathcal{C}_{edge}$.

Lemma 47 (Shared Edge)

Every arrangement in $\mathcal{C}_{edge}$ is satisfiable.

Proof It is sufficient to prove that every arrangement in $\mathcal{C}_{rr}$ is satisfiable. Consider the arrangements in OR Ψ^* RO, discussed in a previous section. These differ from the arrangements in $\mathcal{C}_{rr}$ = OR Ψ^* RO OO in that they are missing exactly one edge satisfying OO.

By Lemma 40 and Lemma 39, every arrangement in OR Ψ^* RO is satisfiable by some edge chain which lies in one of the constraint regions of Figures 7.24, 7.25, 7.26, or 7.27. To each of these constraint regions we can add a single edge between a and b which satisfies OO. In Figures 7.24, 7.25 and 7.26 the edge lies on the equator between a and b, whereas in Figure 7.27 the edge follows the equator around the far side of the sphere. Thus all arrangements in $\mathcal{C}_{rr}$ have a satisfying polygon. □

7.6.4 The Arrangement Theorem Proven

Theorem 5 *For any complete set of relations $R \subseteq R_{allow}$ which obey the rules of Table 7.2, there exist a cone and two points that satisfy R.*

Proof From the First Simplification Lemma and the Second Simplification Lemma it is sufficient to prove that every arrangement in $\mathcal{R}_2$ has a satisfying cone and two points. By the Sphere Reduction Lemma this is equivalent to proving that every two dimensional arrangement in $\mathcal{A}_2$ has a satisfying polygon and two points on the sphere.

The arrangements in $\mathcal{A}_2$ fall into two categories: those in which the points a and b share no edge of the polygon, and those in which they share a single edge. The Unshared Edge Lemma proves that a model exists for all arrangements in the first category; the Shared Edge Lemma shows this for the second category.

□

7.7 Polygons In The Plane

This section presents an interesting little theorem that can be taken as a footnote to this chapter. The reader may have wondered in the previous sections whether it was really necessary to make the complicated polygon constructions on the surface of a sphere, rather than simply on the plane. This section shows that constructions on the plane would not be sufficient.

There are some arrangements in $\{\text{LL}, \text{RR}, \text{LR}, \text{RL}\}^*$ which can be satisfied by two points and a simple polygon on the *sphere*, but not by two points and a simple polygon on the *plane*. Any example is the arrangment

$$\sigma = (\text{LR LL RL RR LR LL RL RR}).$$

A polygon on the plane is shown in Figure 7.51. This polygon satisfies σ but unfortunately is not simple. In Figure 7.52 another polygon is shown which satisfies σ, this time on the surface of the sphere. A single edge of this polygon wraps around the sphere, resulting in a simple polygon.

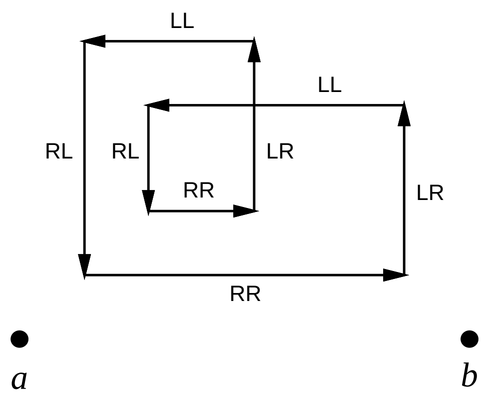

Figure 7.51: Nonsimple Polygon For $\sigma = (\text{ LR LL RL RR })^2$

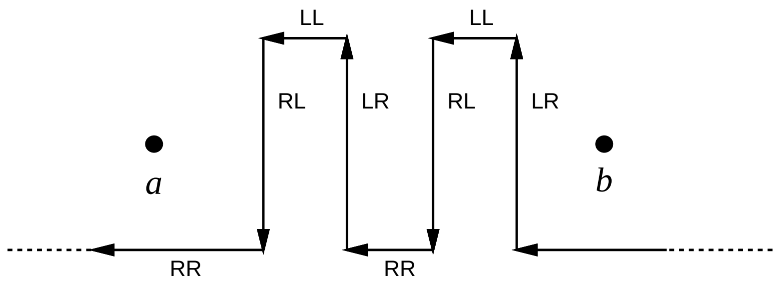

Figure 7.52: Simple Polygon For $\sigma = ($ LR LL RL RR $)^2$

Theorem 6 (Planar Insufficiency)

There is no simple polygon on the plane which satisfies $\sigma = ($ LR LL RL RR $)^2$.

Proof Let a and b be any points on the plane, and let ℓ be a directed line passing through these points, from a to b. Let ℓ_{in} be the segment of ℓ from a to b, and let $\ell_{out} = \ell - \ell_{in}$.

Assume that some polygon exists which satisfies σ. Then these lemmas follow:

Lemma 48 *Each adjacent pair of relations in σ (that is,* (LR LL), (LL RL), (RL RR), *and* (RR LR)*) can only be realized by a left turning pair of edges if the inflection point at the intersection of the two edges is above ℓ (that is, left of ℓ), and can only be realized by a right turn if the inflection point is below ℓ.*

Proof Figure 7.53 shows that this is so for the pair (RR LR). Inspection reveals the same for the other three pairs of adjacent relations in σ. □

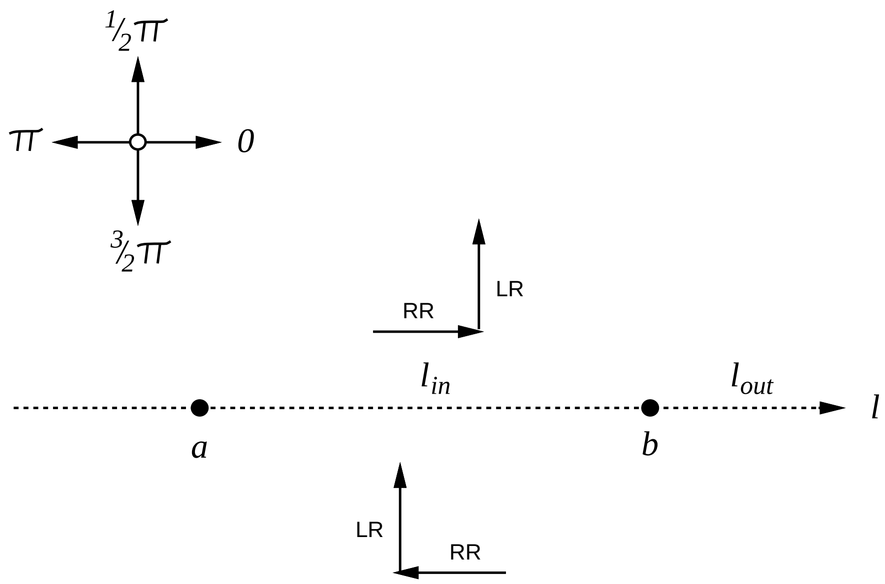

Figure 7.53: (RR LR) turns above and below ℓ.

Lemma 49 *No inflection point can lie* ON ℓ.

Proof No LR or RL edge can have its head or tail on ℓ_{out}, and no LL or RR edge can have its head or tail on ℓ_{in}. Since every inflection point is adjacent to two edges, one of which can't start or end on ℓ_{in} and the other of which can't start or end on ℓ_{out}, no inflection point can lie ON ℓ. □

The next two lemmas will show that all of the inflection points cannot lie on the same side of ℓ.

Lemma 50 *If all inflection points lie above (below)* ℓ *then the chain of edges satisfying the subsequence* (LR LL RL RR LR) *must turn more than* π *radians to the left (right).*

Proof Consider the case in which all points lie *above* ℓ. An edge satisfying LR must point in a direction in the range $(0 \ldots \pi)$ and an edge satisfying RL must point in $(\pi \ldots 2\pi)$ (see Figure 7.53). The edges satisfying (<u>LR</u> LL <u>RL</u> RR <u>LR</u>) must point successively through $(0 \ldots \pi)$, then $(\pi \ldots 2\pi)$, then $(0 \ldots \pi)$ again. If all inflection points are above ℓ then by Lemma 48 only left turns are permitted.

The minimum turning angle is achieved when the first LR edge points in direction $\pi - \varepsilon$, the first RL edges points in the range $\pi + \varepsilon \ldots 2\pi - \varepsilon$, and the second LR edge points in direction $2\pi + \varepsilon$. Thus if the inflection points are all above ℓ, the chain of edges satisfying (LR LL RL RR LR) must turn more than π radians to the left. □

Lemma 51 *The inflection points can't lie all above or all below* ℓ.

Proof If all points lie above ℓ then only left turns are possible. But to satisfy $\sigma =$ (LR LL RL RR LR LL RL RR) the edges must turn more than π radians from the first LR to the second LR, and must turn more than π *additional* radians to return from the second LR to the first. Thus the sum of the internal angles is greater than 2π. But the sum of internal angles in a simple polygon must be 2π, so there is no simple polygon which is completely to one side of ℓ and which satisfies σ. □

Now we can complete the proof of the theorem. From Lemma 51 we must conclude that the inflection points lie on both sides of ℓ. If this is so there must be some edges which cross ℓ. Let e_l be the leftmost such edge, and let e_r be the rightmost such edge (recall that we have assumed that the polygon lies on the *plane*, not the sphere).

Assume without loss of generality that e_l crosses ℓ from below to above (otherwise, the proof is symmetric). Then e_r must cross from above to below, since there must be an even number of crossings if the polygon is to be closed, and since e_l and e_r are the first and last in the sequence of crossings. Since e_l crosses ℓ to the left of e_r, there must be at least one right turn somewhere in the chain of edges between e_l and e_r. But this contradicts Lemma 48, which states that there are no right turns above ℓ.

Thus there is no simple polygon on the plane which satisfies the arrangement $\sigma =$ (LR LL RL RR LR LL RL RR), and the Planar Insufficiency Theorem is proven. □

7.8 Summary

In this chapter the notion of local robustness has been applied to a polyhedral arrangement algorithm. Four simple robustness rules have been proved sufficient to characterize algorithms which are locally robust for a cone and two points.

By the Principle of Local Robustness, any arrangement algorithm that obeys these robustness rules will probably not fail for reasons involving a cone and two points. These rules achieve more than that. Combined with rules to be developed in the next chapter, they ensure that *in practice* the author's polyhedral intersection algorithm never fails. This is demonstrated by the empirical tests presented in Chapter 9.

Chapter 8

Locally Robust Sorting

Any fool can make a rule. – Henry David Thoreau

Essential to the task of polyhedral intersection are three types of sorting algorithm for points in $\Re^3$: linear, radial, and dihedral. A linear sorting algorithm orders points along a line; a radial sorting algorithm orders points that share a common plane radially around a center point; a dihedral sorting algorithm orders points radially around a line.

Recall that an algorithm is robust if, for every approximate input, the algorithm generates output which is the correct output for some corresponding real input, called the *model*. An algorithm is *locally* robust if this is true of some restricted set of approximate inputs.

In this chapter we will characterize locally robust sorting algorithms. For each of the three sorting algorithms, the characterization will consist of a set of logical rules which are true of the algorithm's output if and only if a model exists which would generate that output.

8.1 Linear Sorting

In this section we consider ordering points along a line. Let a *feature* be a point, a line, or a plane. A line is *directed* and has an associated direction vector pointing in the positive direction along the line. A plane is *oriented* and has an associated plane normal pointing to the exterior of the plane.

Relations exist among sets of these features; three such relations are described below (where a and b are points, ℓ is a line, and p is a plane):

- $\text{LOC}(a, p) \in \{\text{INSIDE}, \text{ON}, \text{OUTSIDE}\}$. A point a is either INSIDE, ON, or OUTSIDE an oriented plane p.
- $\text{LINEAR-ORDER}(a, b, \ell) \in \{<, =, >\}$. A point a either precedes, is equal to, or follows a point b on a directed line ℓ.
- $\text{DIRECTION}(\ell, p) \in \{\text{INWARD}, \text{ON}, \text{OUTWARD}\}$. A directed line ℓ is either ON a plane p, or pierces the plane going INWARD or OUTWARD. For example, if $\text{DIRECTION}(\ell, p) =$ OUTWARD then the dot product of the line direction vector and the plane normal is positive.

The input to a linear sorting algorithm consists of a set of features F and a set of allowable relations between the features R_{allow}. The output consists of a *complete* set of relations R, in which every group of features of F which can be related has exactly one relation. Furthermore, $R \subset R_{allow}$.

Let us restrict the features of F to a single line ℓ, n points v_i, and n planes p_i. Let us restrict each point v_i to lie ON a corresponding plane p_i and ON no other plane p_j, and prevent line ℓ from lying ON any of the planes p_i. Then R_{allow} contains all possible relations, *except* for the following:

- v_i INSIDE p_i,
- v_i OUTSIDE p_i,
- v_i ON p_j for $i \neq j$, and
- ℓ ON p_i.

Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then by Theorem 7 below, the rules of Table 8.1 characterize those R which have models.

In Table 8.1, a, b, and c are points, ℓ is a line, and p_a is a plane touching point a. Note that Rules 7 through 10 are essentially the same: they ensure correct INSIDE/OUTSIDE labeling of points with respect to planes, as demonstrated in Figure 8.1.

Table 8.1: Linear Sorting Rules

Rule 5 :	$a <_\ell b \Leftrightarrow b >_\ell a$
Rule 6 :	$a <_\ell b \wedge b <_\ell c \Rightarrow a <_\ell c$
Rule 7 :	$a <_\ell b \wedge \ell$ OUTWARD $p_a \Rightarrow b$ OUTSIDE p_a
Rule 8 :	$a <_\ell b \wedge \ell$ INWARD $p_a \Rightarrow b$ INSIDE p_a
Rule 9 :	$a >_\ell b \wedge \ell$ OUTWARD $p_a \Rightarrow b$ INSIDE p_a
Rule 10 :	$a >_\ell b \wedge \ell$ INWARD $p_a \Rightarrow b$ OUTSIDE p_a

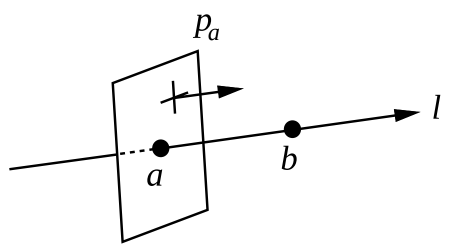

Figure 8.1: Applying Rule 7

Theorem 7 (Linear Sorting)

Define F and R_{allow} as described above. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a satisfying model if and only if the relations of R obey the rules of Table 8.1.

Proof

(Only if) By inspection, the rules are true of any model.

(If) Given some R, we want to construct a model for the features of F which satisfies R. By Rules 5 and 6, the LINEAR-ORDER relation between pairs of points is antisymmetric and transitive. Since *all* pairs of points are related in R, there exists a total ordering on the points. Choose an arbitrary line as a model for ℓ. Choose as models for the v_i any set of points on that line which satisfy the LINEAR-ORDER relations in R. Such a set of points exists since there is a total order on the v_i.

Choose as a model for each plane p_i any oriented plane passing through v_i which satisfies the relation $\text{DIRECTION}(\ell, p_i)$ in R. Then Rules 7 through 10 ensure that, for all other points v_j, the relation $\text{LOC}(v_j, p_i)$ in R is satisfied by this model. For example, suppose that $\text{DIRECTION}(\ell, p_i) = \text{OUTWARD}$. Then by Rule 7, for any point v_j which follows v_i on ℓ ($v_i <_\ell v_j$), the relation $\text{LOC}(v_j, p_i) = \text{OUTSIDE}$ is in R. This is satisfied by the model, as shown in Figure 8.1. □

We conclude that a linear sorting algorithm is locally robust for the features F and restricted relations R_{allow} (defined above) if its output R obeys the rules of Table 8.1.

Recall Rule 4 from the previous chapter on arrangements. The linear sorting rules presented in Table 8.1 make Rule 4 redundant (where the relations ℓ OFF p_i, p_j indicate that ℓ is not ON p_i or p_j).

$$\begin{aligned} \textbf{Rule 4:}\quad & x, y, v \text{ ON } l \;\wedge\; p_i, p_j \text{ ON } v \;\wedge\; \ell \text{ OFF } p_i, p_j \\ & \Rightarrow\; (\text{LOC}(x, p_i) = \text{LOC}(y, p_i) \;\Leftrightarrow\; \text{LOC}(x, p_j) = \text{LOC}(y, p_j)) \end{aligned}$$

8.2 Radial Sorting

In this section we consider ordering points in $\Re^3$ around a center point, where all of the points lie in a common plane. The only features of interest in this operation are points and lines. The relations of interest are the following (where the common plane is an implicit argument in the second and third relations):

- $\text{LINEAR-ORDER}(a, b, \ell) \in \{<, =, >\}$.
- $\text{SIDE}(a, \ell) \in \{\text{LEFT}, \text{ON}, \text{RIGHT}\}$. A point a is either ON a directed line ℓ, or is to its LEFT or RIGHT (as viewed from above the plane).
- $\text{RADIAL-ORDER}(a, b, c, x) \in \{\text{CW}, \text{CCW}, \text{DEGENERATE}\}$. A triple of points (a, b, c) is ordered either clockwise (CW) or counterclockwise (CCW) around a center point x (as viewed from above the plane), or is in a degenerate configuration. For example, in Figure 8.2 $\text{RADIAL-ORDER}(a, b, c, x) = \text{CCW}$. This can also be stated as $\text{CCW}(a, b, c, x)$.

As with the linear sorting algorithm, the input to a radial sorting algorithm consists of a set of features, F, and a set of allowable relations between the features, R_{allow}. The output consists of a complete set of relations, $R \subset R_{allow}$.

Let us restrict the features of F to a single center point x, n points v_i, and n lines ℓ_i. Let us restrict each line ℓ_i to pass through the center point x, and restrict each point v_i to lie on the corresponding line ℓ_i and on no other line ℓ_j. Then R_{allow} contains all possible relations, *except* for the following:

- x LEFT ℓ_i,
- x RIGHT ℓ_i,
- v_i LEFT ℓ_i,
- v_i RIGHT ℓ_i, and
- v_i ON ℓ_j, for $i \neq j$.

Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then by Theorem 8 below, the rules of Table 8.2 characterize those R which have models.

In Table 8.2, a, b, c, and x are points and ℓ is a line. A subscript on a line, like ℓ_a, indicates that the line passes through the point a (that is, a ON ℓ_a). Rules 12 and 13 are the radial equivalent of linear antisymmetry and transitivity, Rules 14 through 17 ensure correct LEFT/RIGHT labeling of triples of points (as demonstrated in Figure 8.2), and Rule 18 ensures correct labeling of pairs of points (as demonstrated in Figure 8.3).

Table 8.2: Radial Sorting Rules

Rule 11 :	$\text{RADIAL-ORDER}(a,b,c,x) = \text{RADIAL-ORDER}(b,c,a,x)$ $= \text{RADIAL-ORDER}(c,a,b,x)$
Rule 12 :	$\text{CCW}(a,b,c,x) \Leftrightarrow \text{CW}(a,c,b,x)$
Rule 13 :	$\text{CCW}(a,b,c,x) \wedge \text{CCW}(a,c,d,x) \Rightarrow \text{CCW}(a,b,d,x)$
Rule 14 :	$x <_\ell a \wedge \text{CCW}(a,b,c,x) \wedge c \text{ LEFT } \ell \Rightarrow b \text{ LEFT } \ell$
Rule 15 :	$x <_\ell a \wedge \text{CW}(a,b,c,x) \wedge c \text{ RIGHT } \ell \Rightarrow b \text{ RIGHT } \ell$
Rule 16 :	$x >_\ell a \wedge \text{CCW}(a,b,c,x) \wedge c \text{ RIGHT } \ell \Rightarrow b \text{ RIGHT } \ell$
Rule 17 :	$x >_\ell a \wedge \text{CW}(a,b,c,x) \wedge c \text{ LEFT } \ell \Rightarrow b \text{ LEFT } \ell$
Rule 18 :	$\text{LINEAR-ORDER}(x,a,\ell_a) = \text{LINEAR-ORDER}(x,b,\ell_b)$ $\Leftrightarrow \text{SIDE}(a,\ell_b) \neq \text{SIDE}(b,\ell_a)$

Theorem 8 (Radial Sorting)

Define F and R_{allow} as described immediately above. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a satisfying model if and only if the relations of R obey the rules of Table 8.2.

Proof

(Only if) By inspection, the rules are true of any model.

(If) We will prove this by induction on the number of points. For $n = 1$ a model exists trivially. Assume that a model exists for $n \leq k$. We must prove that a model exists for $n = k + 1$.

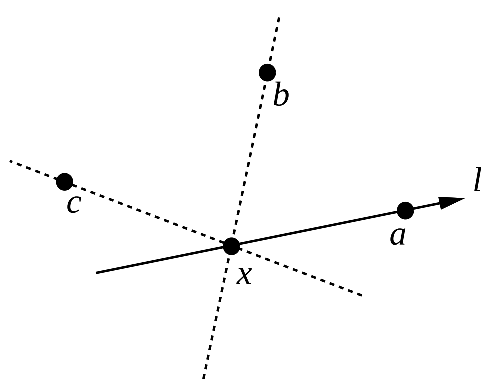

Figure 8.2: Applying Rule 14

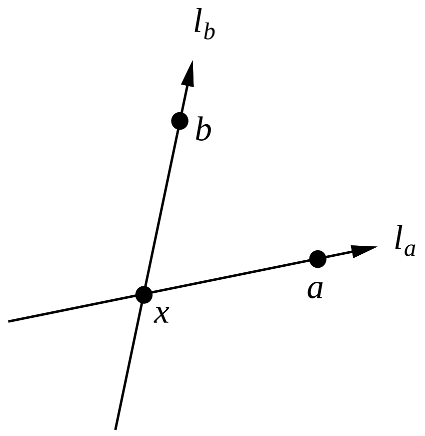

Figure 8.3: Applying Rule 18

Given a set of features F^{k+1} containing x, $v_1, \ldots, v_{k+1}$, $\ell_1, \ldots, \ell_{k+1}$, and a corresponding complete set of relations R^{k+1} which satisfy the rules of Table 8.2, we want to construct a model which satisfies R^{k+1}.

By Rules 12 and 13 there is a total (cyclic) order on the points $v_1, \ldots, v_{k+1}$. To see this, consider those instantiations of Rules 12 and 13 in which v_1 occurs as the first element of every quadruple of vertices:

$$\begin{aligned} &\text{Rule 12a}: \quad \textsc{ccw}(v_1, b, c, x) \;\Leftrightarrow\; \textsc{cw}(v_1, c, b, x) \\ &\text{Rule 13a}: \quad \textsc{ccw}(v_1, b, c, x) \;\wedge\; \textsc{ccw}(v_1, c, d, x) \;\Rightarrow\; \textsc{ccw}(v_1, b, d, x). \end{aligned}$$

Thus, the relation between pairs of points in $\{v_2, \ldots, v_{k+1}\}$ is antisymmetric and transitive. Since *all* pairs of points are related in R^{k+1}, there exists a total ordering on the points.

Assume without loss of generality that the points are ordered $v_1, v_2, \ldots, v_{k+1}$, such that $\textsc{ccw}(v_i, v_{i+1}, v_{i+2}, x)$. Also assume without loss of generality that $x <_{l_i} v_i$ for all i (otherwise use Rules 16 and 17 in place of Rules 14 and 15 in the following argument).

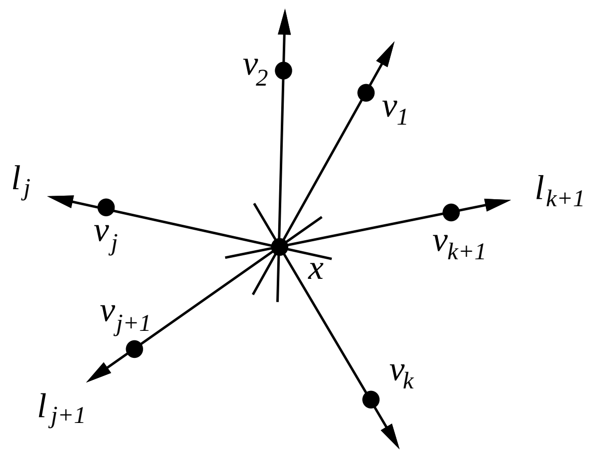

Figure 8.4: v_j LEFT $\ell_{k+1} \;\Rightarrow\; v_1$ RIGHT ℓ_j

Refer to Figure 8.4. Let j be the maximum index such that v_j LEFT ℓ_{k+1}. Obviously, CCW(v_{k+1}, v_1, v_j, x), which by Rule 12 implies CW(v_j, v_1, v_{k+1}, x). By Rule 18 and the fact that v_{k+1} is not ON ℓ_j (from the restriction in R_{allow}), v_j LEFT ℓ_{k+1} implies v_{k+1} RIGHT ℓ_j. From these two facts and Rule 15, we conclude that the relation

$$v_1 \text{ RIGHT } \ell_j \tag{8.1}$$

is present in R^{k+1}.

Since j is the maximum index such that v_j LEFT ℓ_{k+1}, and since v_{j+1} is not ON ℓ_{k+1} (from the restriction in R_{allow}), $j+1$ is the minimum index such that v_{j+1} RIGHT ℓ_{k+1}. By an argument similar to that above, we conclude that the relation

$$v_k \text{ LEFT } \ell_{j+1} \tag{8.2}$$

is also present in R^{k+1}.

We can now construct a model which satisfies the relations of R^{k+1}. First remove v_{k+1} and ℓ_{k+1} from F^{k+1} to get F^k, and remove any relations involving these features from R^{k+1} to get R^k. Since R^{k+1} satisfies the rules of Table 8.2, so does R^k, and by the inductive hypothesis a model M^k exists which satisfies R^k. The model satisfies Relations 8.1 and 8.2 since these relations are present in R^k.

Refer to Figure 8.5, which shows part of the model M^k. Since v_1 RIGHT ℓ_j in the model M^k, the positive angle (v_1, x, v_j) must be less than π radians (where a positive angle is counterclockwise). Similarly, the positive angle (v_{j+1}, x, v_k) must also be less the π radians.

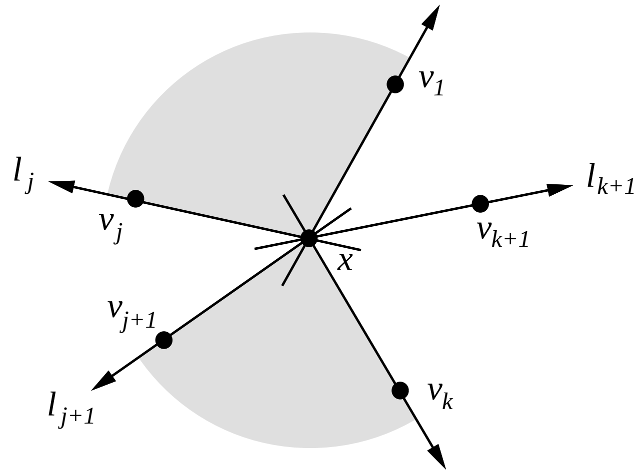

Figure 8.5: ANGLE$(v_1, x, v_j) < \pi$ and ANGLE$(v_{j+1}, x, v_k) < \pi$

Since each of these sectors spans less than π radians and only intersects at x, a line ℓ_{k+1} and a vertex v_{k+1} can be placed between them, as shown in Figure 8.5. This new model satisfies R^{k+1}, since in R^{k+1} all points $v_1, \ldots, v_j$ are LEFT of ℓ_{k+1}, all points $v_{j+1}, \ldots, v_k$ are RIGHT of ℓ_{k+1}, v_{k+1} is RIGHT of all lines $\ell_1, \ldots, \ell_j$, and v_{k+1} is LEFT of all lines $\ell_{j+1}, \ldots, \ell_k$. Thus a model exists which satisfies R^{k+1}. $\square$

We conclude that a radial sorting algorithm is locally robust for the features F and restricted relations R_{allow} (defined above) if its output R obeys the rules of Table 8.2.

8.3 Dihedral Sorting

In this section we consider ordering points in $\Re^3$ around a line. This is very similar to the radial sorting of the preceding section. The features of interest are planes, a line, and points. The relations of interest are the following:

- $\textsc{loc}(a, p) \in \{\textsc{inside}, \textsc{on}, \textsc{outside}\}$.
- $\textsc{side}(a, \ell, p_a) \in \{\textsc{left}, \textsc{on}, \textsc{right}\}$. A point a in $\Re^3$ is either ON the directed line ℓ, or is to its LEFT or RIGHT with respect to the plane p_a shared by a and ℓ. As shown in Figure 8.6, if $\hat{p}_a$ is the plane normal of p_a and $\hat{\ell}$ is the line direction vector for ℓ, then the cross product $\hat{p}_a \times \hat{\ell}$ points to the LEFT of ℓ with respect to p_a.

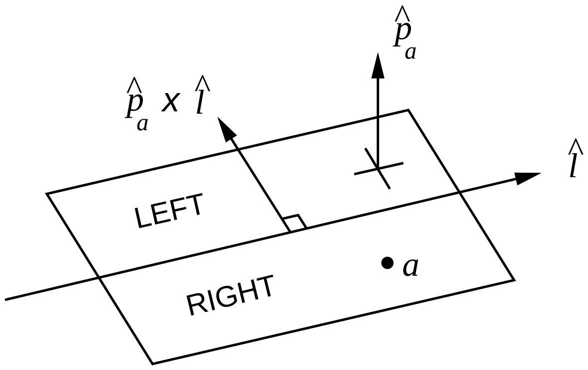

Figure 8.6: $\textsc{side}(a, \ell, p_a) = \textsc{right}$

- $\textsc{dihedral-order}(a, b, c, \ell) \in \{\textsc{rh}, \textsc{lh}, \textsc{degenerate}\}$. A triple of points (a, b, c) is ordered either "righthandedly" (RH) or "lefthandedly" (LH) about a line ℓ, or is in a degenerate configuration. For example, if $\textsc{rh}(a, b, c, \ell)$ then the points a, b, and c are positioned such that if one's right hand were placed on the line with the thumb pointing along the direction vector $\hat{\ell}$, and the fingers were then curled around the line, they would encounter a, then b, then c (or some cyclic permutation). The relation LH is defined similarly.

As with the other sorting algorithms, the input to a dihedral sorting algorithm consists of a set of features F and a set of allowable relations between the features R_{allow}. The output consists of a complete set of relations, $R \subset R_{allow}$.

Let us restrict the features of F to a single line ℓ, n points v_i, and n planes p_i. Let us restrict the line ℓ to lie on each plane p_i, and restrict each point v_i to lie on the corresponding plane p_i and on no other plane p_j. Then R_{allow} contains all possible relations, *except* for the following:

- ℓ INWARD p_i,
- ℓ OUTWARD p_i,
- v_i INSIDE p_i,
- v_i OUTSIDE p_i, and
- v_i ON p_j, for $i \neq j$.

Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then by Theorem 9 below, the rules of Table 8.3 characterize those R which have models.

In Table 8.3, a, b, and c are points, ℓ is the line, and p_a is the plane embedding both the line ℓ and the point a. Note that these rules are *exact analogues* of the radial sorting rules of Table 8.2.

Table 8.3: Dihedral Sorting Rules

Rule 19 :	$\text{DIHEDRAL-ORDER}(a,b,c,\ell) = \text{DIHEDRAL-ORDER}(b,c,a,\ell)$ $= \text{DIHEDRAL-ORDER}(c,a,b,\ell)$
Rule 20 :	$\text{RH}(a,b,c,\ell) \Leftrightarrow \text{LH}(a,c,b,\ell)$
Rule 21 :	$\text{RH}(a,b,c,\ell) \wedge \text{RH}(a,c,d,\ell) \Rightarrow \text{RH}(a,b,d,\ell)$
Rule 22 :	$a \text{ LEFT}_{p_a} \ell \wedge \text{RH}(a,b,c,\ell) \wedge c \text{ OUTSIDE } p_a \Rightarrow b \text{ OUTSIDE } p_a$
Rule 23 :	$a \text{ LEFT}_{p_a} \ell \wedge \text{LH}(a,b,c,\ell) \wedge c \text{ INSIDE } p_a \Rightarrow b \text{ INSIDE } p_a$
Rule 24 :	$a \text{ RIGHT}_{p_a} \ell \wedge \text{RH}(a,b,c,\ell) \wedge c \text{ INSIDE } p_a \Rightarrow b \text{ INSIDE } p_a$
Rule 25 :	$a \text{ RIGHT}_{p_a} \ell \wedge \text{LH}(a,b,c,\ell) \wedge c \text{ OUTSIDE } p_a \Rightarrow b \text{ OUTSIDE } p_a$
Rule 26 :	$\text{SIDE}(a,\ell,p_a) = \text{SIDE}(b,\ell,p_b) \Leftrightarrow \text{LOC}(a,p_b) \neq \text{LOC}(b,p_a)$

Theorem 9 (Dihedral Sorting)

Define F and R_{allow} as described immediately above. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a satisfying model if and only if the relations of R obey the rules of Table 8.3.

Proof

The proof is exactly analagous to the proof of the Radial Sorting Theorem. □

We conclude that a dihedral sorting algorithm is locally robust for the features F and restricted relations R_{allow} (defined above) if its output R obeys the rules of Table 8.2.

8.4 Summary

In this chapter the notion of local robustness has been applied to three sorting algorithms, and several robustness rules have been developed. By the Principle of Local Robustness, a sorting algorithm (linear, radial, or dihedral) which obeys the corresponding robustness rules will probably not fail for reasons involving the limited sets of features described above.

The next chapter discusses how these rules, along with the arrangement rules developed in the preceding chapter, were incorporated into the author's polyhedral intersection algorithm.

Chapter 9

A Robust Polyhedral Intersection Algorithm

When ideas fail, words come in very handy. – Goethe

The robustness rules of the preceding chapters were incorporated in a polyhedral intersection algorithm. The algorithm was extensively tested and never failed to produce a valid polyhedron of intersection.

The present chapter will briefly discuss how the robustness rules were incorporated in the intersection algorithm. Following this, we will discuss the tests that were performed on the algorithm.

The intersection algorithm is not a contribution of this dissertation and, hence, no attention was given to making it efficient (it is in fact quite slow). Rather, the intersection algorithm serves as a testbed to determine whether the local robustness rules are effective. As the tests demonstrate, the rules do work, and the local robustness approach does achieves robustness in practice.

9.1 Incorporating the Robustness Rules

At an abstract level, the author's polyhedral intersection algorithm operates as shown in Figure 9.1.

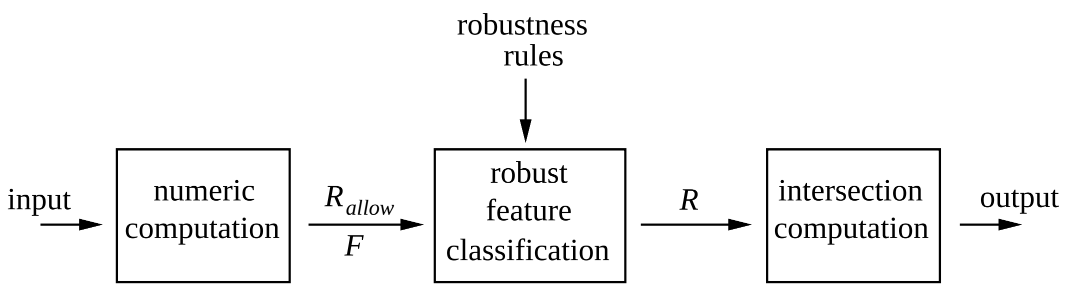

Figure 9.1: Abstract Intersection Algorithm

The raw input to the algorithm consists of symbolic data describing the topology of the two polyhedra to be intersected, and numeric data describing the approximate locations of the features of the polyhedra.

From this raw input, the first component of the algorithm (labeled "numeric computation" in Figure 9.1) computes a set of features F and a set of allowable relations among these features, R_{allow}. These are the same F and R_{allow} discussed in the preceding chapters.

For example, consider the POINT/PLANE relation: a point can be either INSIDE, ON, or OUTSIDE a plane. If the approximately represented point v is sufficiently distant from the approximately represented plane p, then there will be a single allowable relation between v and p in the set R_{allow}: either v INSIDE p or v OUTSIDE p. If the features are close then there will be three such relations in R_{allow}: v INSIDE p, v ON p, and v OUTSIDE p.

The second component of the algorithm (labeled "robust feature classification") takes F and R_{allow}, along with the robustness rules derived in the preceding chapters, and produces a complete set of relations $R \subset R_{allow}$ that satisfies the robustness rules.

The third component of the algorithm (labeled "intersection computation") computes the topology of the solid of intersection from the set of relations R and from the topological data of the raw input.

The output of the algorithm consists of the topology of the polyhedron of intersection and the relevant approximate plane equations taken from the raw input.

9.1.1 The Robust Feature Classification

Robust feature classification is achieved with a constraint propagation algorithm. In this section we will briefly discuss the algorithm. For the details of such an algorithm, the reader is directed to Waltz [44], Huffman [20], or Clowes [6].

The input to the algorithm consists of a set of features F, a set of allowable relations among these features, R_{allow}, and a set of robustness rules. The output is a complete set of relations $R \subset R_{allow}$ that satisfies the robustness rules.

R_{allow} may contain several alternative labelings for a given relation. For example, the relation between the line ℓ_i and the plane p_j might occur in R_{allow} as ℓ_i INWARD p_j, ℓ_i ON p_j, and ℓ_i OUTWARD p_j. The constraint propagation algorithm chooses one of the alternatives (that is, the relation is *instantiated*), removes the other alternatives from R_{allow}, and propagates the consequences of this instantiation by evaluating the robustness rules. This procedure is iterated until no alternative labelings remain in R_{allow}.

Constraint propagation has exponential time complexity because the algorithm may make an inconsistent choice and have to backtrack when this inconsistency is discovered. However, backtracking *never* occured in any of the tests performed on the algorithm. This is probably due to the following aspects of the algorithm:

- The only type of relation chosen for instantiation was the SIDE relation. Others types of relation, like the LINEAR-ORDER and DIRECTION relations, were only instantiated as a consequence of a deduction from one of the robustness rules.
- The robustness rules were slightly augmented: for example, for each rule $A \wedge B \Rightarrow C$, the logically equivalent rules $A \wedge \overline{C} \Rightarrow \overline{B}$ and $B \wedge \overline{C} \Rightarrow \overline{A}$ were added.

The advantage of the constraint propagation approach is that the intersection algorithm runs at normal speed when presented with unambiguous input, and only slows down when it is necessary to make deductions about ambiguous input.

9.1.2 The Intersection Computation

Since the intersection computation is not the focus of this work, it will only be discussed briefly. For details of various intersection algorithms, the reader is directed to any one of Karasick [22], Vanecek [43], Segal and Séquin [37], or Laidlaw, Trumbore, and Hughes [23]. Chapter 4 of Karasick's dissertation [22] also provides a good survey of other intersection algorithms.

The author's intersection algorithm compares every edge of one polyhedron with every face of the other polyhedron, and vice versa. If the endpoints of the edge span the plane of the face, then the intersection point is considered: if it is on the interior of the face then it is tagged as a vertex of the polyhedron of intersection. Due to the number of special cases which can arise in intersecting an edge and a face, the bulk of the algorithm lies in this step. Once the vertices have been classified they are sorted along each edge and the resulting data structure is traversed to compute the topology of the polyhedron of intersection.

Several parts of the intersection algorithm make use of the robust feature classification, as follows:

- Comparison of the edge endpoints with respect to the plane of each face relies upon *robust arrangement*.
- Sorting the vertices along each edge relies upon *robust linear sorting*.
- Traversal of the data structure relies upon *robust radial sorting* and *robust dihedral sorting*. For example, consider Figure 9.2. The algorithm must be able to determine, on a given face, the order of edges around a common vertex if it is to correctly reconstruct the boundary of the face. If the incoming and outgoing edges at a vertex are not correctly paired, then the edges of the face boundary may not all have the face interior to their right, as required.

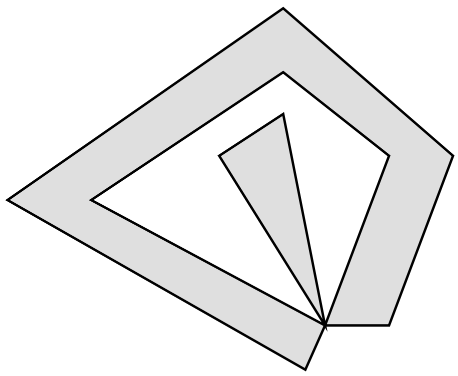

Figure 9.2: The Necessity of Robust Radial Sorting

It is interesting to note that the robust point location algorithm of Chapter 4 is *not* used in the intersection algorithm; a simpler algorithm, which uses additional information available from the robust feature classifiction, is used in its place. To determine whether a point v lies within the boundary of a face f, a standard "parity test" is performed: a line ℓ is selected that lies in the plane of f and that touches v, and the intersections of ℓ with the boundary edges of f are determined. The intersection points which *precede* v on ℓ are counted: if there is an even number of them, then v is on the exterior of f, otherwise it is on the interior. No part of this computation is done numerically; it relies soley upon the robust feature classification and the robust linear sorting.

It is also interesting to note that certain rules were *not* incorporated into the algorithm: Rules 11 through 17 and Rules 19 through 25 were not used. Despite this, the algorithm did not fail. This would seem to indicate that the RADIAL-ORDER and DIHEDRAL-ORDER relations are not really necessary; the simpler rules that do not involve these relations appear to be sufficient to ensure robustness in practice. Thus, even local robustness may be too strong a criterion for the success of a polyhedral intersection algorithm.

9.2 The Tests

Many tests were performed on the polyhedral intersection algorithm. In these tests the algorithm never failed to produce a valid polyhedron of intersection. Furthermore, the constraint satisfaction component never backtracked.

9.2.1 What Causes Failure?

A standard test to measure the robustness of an intersection algorithm is to intersect two identical objects [22,23]. Both objects are centered at the origin, but one is rotated by a small angle through an arbitrary axis.

With respect to this axis and the two objects, there are two distinguished angles: α and β. Below the angle α, the corresponding (approximate) features of the two objects are indistinguishable with numeric tests. Above the angle β, all relations between features can be determined with numeric tests, including those relations which involve features derived by combining the two objects (for example, points defined as the intersection of an edge of one object with a face of the other object). Note that α and β also depend upon the precision of the floating point arithmetic.

Any nonrobust intersection algorithm should successfully output the original object for angles below α and some different object for angles above β. It is in the *region of instability* $[\alpha, \beta]$ that nonrobust algorithms fail. They fail by making inconsistent decisions about those relations between features that cannot be determined numerically.

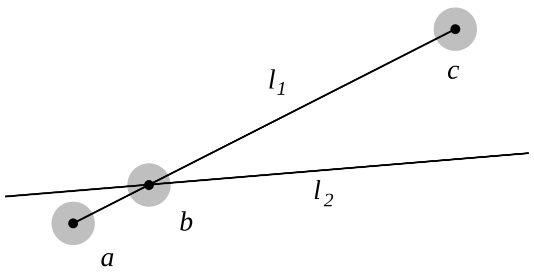

Figure 9.3: All Distinguishable

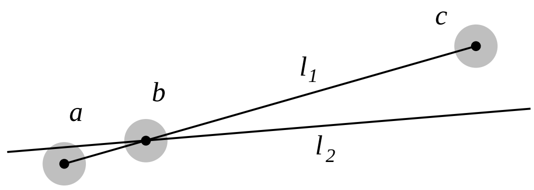

Figure 9.4: Some Indistinguishable

For example, consider Figure 9.3, in which the regions of uncertainty of each of the points a, b, and c are indicated with gray disks, and b is defined as $\ell_1 \cap \ell_2$. Assume that the positions of the lines are known exactly. Then an intersection algorithm can determine with numeric tests that both a and b are OFF ℓ_2.

However, in Figure 9.4, the region of uncertainty of a touches the line ℓ_2 and a numeric test cannot determine the relation between a and ℓ_2. If the intersection algorithm arbitrarily decides that a is ON ℓ_2, then it must also conclude that ℓ_1 is ON ℓ_2, since ℓ_1 and ℓ_2 would then share the two distinct points a and b. But the algorithm must also conclude the ℓ_1 is OFF ℓ_2, since the point c is not common to both lines.

This is an example of the type of failure that occurs in the region of instability $[\alpha, \beta]$. This is the type of failure that local robustness is designed to prevent.

9.2.2 How Can We Test For Robustness?

To empirically test an intersection algorithm for robustness, we must choose an arbitrary axis of rotation and determine the associated region of instability $[\alpha, \beta]$. Knowing this region, we can attempt to induce failure by excuting the intersection algorithm for various angles within this range.

The two simple procedures shown in Figure 9.5 can determine α and β. In these procedures, OBJ(θ, v) is the object rotated by angle θ about the axis v, and ε is the machine precision.

Some researchers suggest that their intersection algorithms are robust because they successfully compute α with the procedure of Figure 9.5, or because their algorithms fail during this computation when the distance between α and β is very small. While the procedures of Figure 9.5 can sometimes induce failure in nonrobust algorithms, completing either one of them successfully does not mean that the intersection algorithm is robust: it could be that the angles encountered in the search fortuitously avoided inconsistent decisions by the algorithm.

Since we cannot hope to test every representable angle between α and β, the best approach is to test a large random sample of them. In doing so, we hope to test many different combinations of ambiguous relations between features. Failure is more likely near the lower angle α, since it is near this angle that the interseciton algorithm must make more decisions about numerically ambiguous relations. Thus, more tests should be performed near α.

9.2.3 Testing the Locally Robust Intersection Algorithm

The author's polyhedral intersection algorithm was tested for a pair of unit cubes and a similar-sized pair of regular octahedra. The octahedron test is significant because — in the early stages of the algorithm's design — it would cause the algorithm to fail where the cube test could not[1].

The algorithm was tested on a Symbolics 3675 Lisp Machine using twenty-three bit floating point arithmetic. The features of the cube and octahedron were represented with an accuracy of 10^{-5}, slightly less accurate than the machine epsilon of $10^{-7} (\approx 2^{-23})$. Tables 9.1 and 9.2 summarize the tests. Each table includes the following information:

[1]The octahedron test is probably more rigorous because there is greater complexity when an edge of one octahedron intersects close to one of the valence-four vertices of the other octahedron. The valence-three vertices of the cubes are in some sense more "forgiving" since they are not overdetermined, as are the vertices of the octahedron.

```
Find-alpha( OBJ, v )

    β ← 10.0 degrees
    α ← 0.0 degrees
    while |β − α| > ε do
        θ ← (α + β)/2
        Intersect OBJ(0, v) and OBJ(θ, v).  If all corresponding
        features are numerically indistinguishable, then
            α ← θ
        else
            β ← θ
    Return α

Find-beta( OBJ, v )

    β ← 10.0 degrees
    α ← 0.0 degrees
    while |β − α| > ε do
        θ ← (α + β)/2
        Intersect OBJ(0, v) and OBJ(θ, v).  If all numeric
        tests were unambiguous, then
            β ← θ
        else
            α ← θ
    Return β
```

Figure 9.5: Determining α and β

- the number of vertices, edges, and faces in the resulting polyedron of intersection,
- the number of coincident vertices between the two rotated objects,
- the number of ambiguous numeric relations which were arbitrarily instantiated by the robust feature classification component of the algorithm,
- the number of ambiguous numeric relations which were deduced from the robustness rules, and which were of interest to the intersection computation component of the algorithm (others relations were deduced, but were not of direct interest), and
- the total number of ambiguous numeric relations.

The Cube Test

The axis of rotation for the cube test was the vector $[1, 2, 3]$. This was chosen to avoid rotating through any of the vertices or edges of the cube, in the hope that more interesting ambiguities would arise[2]. The computed α and β for the cube were 0.00066126644 and 0.96176076 degrees, respectively. The algorithm succeeded on all angles that were tested. From over one hundred such tests, a representative sample of 25 are summarized in Table 9.1.

Note that at angle β (shown at the bottom of Table 9.1) there are still 90 ambiguous relations, which appears to contradict the definition of β. However, these unknown relations were not of interest to the algorithm.

The Octahedron Test

The axis of rotation for the octahedron test was the vector $[1, 2, 3]$. The computed α and β for the octahedron were 0.001603792 and 3.481383 degrees, respectively. The algorithm succeeded on all angles that were tested. From over one hundred such tests, a representative sample of 25 are summarized in Table 9.2.

[2]A few tests were performed with an axis of $[1, 1, 1]$ passing through two opposite vertices of the cube. Fewer abmiguities arose in these tests than in those with the axis of $[1, 2, 3]$.

Table 9.1: Data From the Cube Test

angle	*result vertices*	*result edges*	*result faces*	*coincident vertices*	*relations instantiated*	*relations deduced*	*relations total*
0.0006612664	8	12	6	8	7	97	1200
0.00066126644	8	12	6	2	9	93	1349
0.0007324219	8	14	8	1	10	90	1307
0.0009765625	10	16	8	0	9	65	1820
0.0015415343	14	24	12	0	8	30	1969
0.0021935818	20	30	12	0	6	14	2172
0.00390625	20	30	12	0	6	12	1814
0.00443137	18	28	12	0	2	6	1225
0.0078125	18	28	12	0	2	14	1143
0.015625	18	28	12	0	2	14	924
0.03125	20	30	12	0	0	16	716
0.0625	20	30	12	0	0	16	624
0.125	20	30	12	0	0	16	624
0.13524757	20	30	12	0	0	16	624
0.25	20	30	12	0	0	16	624
0.3655514	20	30	12	0	0	16	578
0.47362018	20	30	12	0	0	8	453
0.5	20	30	12	0	0	8	443
0.52384853	20	30	12	0	0	8	443
0.6977829	20	30	12	0	0	8	439
0.750525	20	30	12	0	0	8	439
0.93798125	20	30	12	0	0	8	421
0.9617593	20	30	12	0	0	4	355
0.96176076	20	30	12	0	0	2	222
0.9617608	20	30	12	0	0	0	90

Table 9.2: Data From the Octahedron Test

angle	*result vertices*	*result edges*	*result faces*	*coincident vertices*	*relations instantiated*	*relations deduced*	*relations total*
0.0016037918	6	12	8	6	6	96	1638
0.001603792	6	12	8	1	6	92	2107
0.0016495044	6	12	8	1	6	89	2101
0.001953125	7	13	8	1	7	75	2354
0.0027060139	11	18	9	0	9	49	3127
0.00390625	13	21	10	0	8	44	3495
0.0042598094	15	24	11	0	7	32	3408
0.004904998	21	32	13	0	8	31	3813
0.0070919236	21	32	13	0	7	14	2898
0.008351261	21	32	13	0	6	10	2646
0.01	24	36	14	0	5	3	2775
0.018272676	24	37	15	0	4	8	1672
0.033572037	24	37	15	0	4	8	1585
0.058525786	25	38	15	0	3	5	1401
0.061003245	25	38	15	0	3	5	1401
0.08395079	26	39	15	0	2	2	1238
0.125	27	41	16	0	1	19	1166
0.19195558	28	42	16	0	0	20	972
0.5	28	42	16	0	0	20	862
1.0	28	42	16	0	0	12	587
1.6489155	28	42	16	0	0	8	408
2.3067856	28	42	16	0	0	4	235
2.505	28	42	16	0	0	4	737
3.481383	28	42	16	0	0	1	194
3.4813833	28	42	16	0	0	0	217

Chapter 10

Conclusion

One of the symptoms of an approaching nervous breakdown is the belief that one's work is terribly important. – Bertrand Russell

The philosophy of this work is to build practical, reliable algorithms on a firm, theoretical foundation. Local robustness provides this foundation, and the polyhedral intersection algorithm of Chapter 9 demonstrates that local robustness really does work in practice. The contributions of this dissertation are the following:

- Using the formal theory of robustness of Hoffmann, Hopcroft, and Karasick [17], a robust algorithm for point location in approximate polygons has been developed, and the point location problem has been proved consistent.
- It has been shown that maximizing incidences between approximate polygons is NP-hard.
- A new theory of local robustness has been presented which bridges the gap between the theory and practice of building robust algorithms.
- A set of simple robustness rules has been developed that characterize locally robust polyhedral intersection algorithms.
- An intersection algorithm has been implemented to demonstrate that the robustness rules work in practice. Unlike other intersection algorithms, which either crash or give up when presented with ambiguous input, the locally robust algorithm *always* computes a valid polyhedron of intersection.

10.1 What's Next?

Local robustness should be applied to other geometric problems, such as building Voronoi diagrams and finite-element meshes (however, for line arrangements the approach of Milenkovic [25] is probably better, since it has already been proven robust and has been efficiently implemented).

A better, locally robust polyhedral intersection algorithm should be developed. Instead of incorporating the robustness rules in an explicit database (as was done with the constraint propagation approach in this dissertation) a better algorithm would incorporate them implicitly and efficiently.

Local robustness should be applied to more aspects of solid modeling, not just to polyhedral intersection. The most important aspect would be a locally robust *rounding algorithm*: since coordinate transformations on a solid (like translation, rotation, or scaling) generally increase the size of the floating point numbers in the computer representation of the solid, these floating point numbers must be periodically rounded while maintaining the same solid topology. Since this rounding problem is NP-hard [29], it seems to be an ideal application of local robustness. The robustness rules presented in this dissertation would probably work for such a rounding algorithm. Another aspect of solid modeling is that of nonpolyhedral solids: a characterization of locally robust intersection algorithms for various classes of these solids should be developed.

10.2 And Finally ...

I believe that we should not spend so much energy on developing perfectly robust, perfectly efficient algorithms. Nor should we ignore the vagaries of floating point arithmetic, and hope that our "real arithmetic" algorithms will work when implemented. Local robustness provides the middle ground with which to build practical, reliable algorithms on a firm, theoretical foundation.

Appendix A

The Robustness Rules

This appendix summarizes the robustness rules which characterize locally robust polyhedral intersection algorithms. The input to such an algorithm consists of a set of features, some topological information, and a set of allowable relations.

The *features* consist of planes, lines, and points. The following *relations* exist among features (where a and b are points, ℓ is a line, p is a plane, and f is any feature):

$$\begin{aligned}
\text{INCIDENCE}(f_i, f_j) &\in \{\text{ON}, \text{OFF}\} \\
\text{LOC}(a, p) &\in \{\text{INSIDE}, \text{ON}, \text{OUTSIDE}\} \\
\text{LINEAR} - \text{ORDER}(a, b, \ell) &\in \{<, =, >\} \\
\text{DIRECTION}(\ell, p) &\in \{\text{INWARD}, \text{ON}, \text{OUTWARD}\} \\
\text{SIDE}(a, \ell, p) &\in \{\text{LEFT}, \text{ON}, \text{RIGHT}\} \\
\text{RADIAL} - \text{ORDER}(a, b, c, x, p) &\in \{\text{CW}, \text{CCW}, \text{DEGENERATE}\} \\
\text{DIHEDRAL} - \text{ORDER}(a, b, c, \ell) &\in \{\text{RH}, \text{LH}, \text{DEGENERATE}\}
\end{aligned}$$

A locally robust polyhedral intersection algorithm is one that obeys the robustness rules of Table A.1 (in Table A.1, F is the set of features of the two polyhedra). Note that Rule 4 (concerning polyhedral arrangement) is not included in this table since it is redundant. Also, note that the radial sorting rules, which apply to points and lines lying ON a common plane, have the plane as an explicit argument, whereas they did not have this when originally presented in Section 8.2. Finally, recall from Chapter 9 that Rules 11 through 17 and Rules 19 through 25 were not used in the author's robust polyhedral intersection algorithm.

Table A.1: Robustness Rules

Polyhedral Arrangement

Rule 1 : x ON $y \;\Rightarrow\; \text{LOC}(x,p) = \text{LOC}(y,p)$
Rule 2 : x ON $f_1 \;\wedge\; x$ ON $f_2 \;\Leftrightarrow\; x$ ON $f_1 \cap f_2$
Rule 3 : x ON $f_1 \;\wedge\; x$ ON $f_2 \;\Rightarrow\; f_1 \cap f_2 \in F$

Linear Sorting

Rule 5 : $a <_\ell b \;\Leftrightarrow\; b >_\ell a$
Rule 6 : $a <_\ell b \;\wedge\; b <_\ell c \;\Rightarrow\; a <_\ell c$
Rule 7 : $a <_\ell b \;\wedge\; \ell$ OUTWARD $p_a \;\Rightarrow\; b$ OUTSIDE p_a
Rule 8 : $a <_\ell b \;\wedge\; \ell$ INWARD $p_a \;\Rightarrow\; b$ INSIDE p_a
Rule 9 : $a >_\ell b \;\wedge\; \ell$ OUTWARD $p_a \;\Rightarrow\; b$ INSIDE p_a
Rule 10 : $a >_\ell b \;\wedge\; \ell$ INWARD $p_a \;\Rightarrow\; b$ OUTSIDE p_a

Radial Sorting

Rule 11 : $\text{RADIAL-ORDER}(a,b,c,x,p) = \text{RADIAL-ORDER}(b,c,a,x,p)$
$= \text{RADIAL-ORDER}(c,a,b,x,p)$
Rule 12 : $\text{CCW}(a,b,c,x,p) \;\Leftrightarrow\; \text{CW}(a,c,b,x,p)$
Rule 13 : $\text{CCW}(a,b,c,x,p) \;\wedge\; \text{CCW}(a,c,d,x,p) \;\Rightarrow\; \text{CCW}(a,b,d,x,p)$
Rule 14 : $x <_\ell a \;\wedge\; \text{CCW}(a,b,c,x,p) \;\wedge\; c \text{ LEFT}_p\, \ell \;\Rightarrow\; b \text{ LEFT}_p\, \ell$
Rule 15 : $x <_\ell a \;\wedge\; \text{CW}(a,b,c,x,p) \;\wedge\; c \text{ RIGHT}_p\, \ell \;\Rightarrow\; b \text{ RIGHT}_p\, \ell$
Rule 16 : $x >_\ell a \;\wedge\; \text{CCW}(a,b,c,x,p) \;\wedge\; c \text{ RIGHT}_p\, \ell \;\Rightarrow\; b \text{ RIGHT}_p\, \ell$
Rule 17 : $x >_\ell a \;\wedge\; \text{CW}(a,b,c,x,p) \;\wedge\; c \text{ LEFT}_p\, \ell \;\Rightarrow\; b \text{ LEFT}_p\, \ell$
Rule 18 : $\text{LINEAR-ORDER}(x,a,\ell_a) = \text{LINEAR-ORDER}(x,b,\ell_b)$
$\Leftrightarrow\; \text{SIDE}(a,\ell_b,p) \neq \text{SIDE}(b,\ell_a,p)$

Dihedral Sorting

Rule 19 : $\text{DIHEDRAL-ORDER}(a,b,c,\ell) = \text{DIHEDRAL-ORDER}(b,c,a,\ell)$
$= \text{DIHEDRAL-ORDER}(c,a,b,\ell)$
Rule 20 : $\text{RH}(a,b,c,\ell) \;\Leftrightarrow\; \text{LH}(a,c,b,\ell)$
Rule 21 : $\text{RH}(a,b,c,\ell) \;\wedge\; \text{RH}(a,c,d,\ell) \;\Rightarrow\; \text{RH}(a,b,d,\ell)$
Rule 22 : $a \text{ LEFT}_p\, \ell \;\wedge\; \text{RH}(a,b,c,\ell) \;\wedge\; c$ OUTSIDE $p \;\Rightarrow\; b$ OUTSIDE p
Rule 23 : $a \text{ LEFT}_p\, \ell \;\wedge\; \text{LH}(a,b,c,\ell) \;\wedge\; c$ INSIDE $p \;\Rightarrow\; b$ INSIDE p
Rule 24 : $a \text{ RIGHT}_p\, \ell \;\wedge\; \text{RH}(a,b,c,\ell) \;\wedge\; c$ INSIDE $p \;\Rightarrow\; b$ INSIDE p
Rule 25 : $a \text{ RIGHT}_p\, \ell \;\wedge\; \text{LH}(a,b,c,\ell) \;\wedge\; c$ OUTSIDE $p \;\Rightarrow\; b$ OUTSIDE p
Rule 26 : $\text{SIDE}(a,\ell,p_a) = \text{SIDE}(b,\ell,p_b) \;\Leftrightarrow\; \text{LOC}(a,p_b) \neq \text{LOC}(b,p_a)$

Bibliography

[1] B. Brüderlin. Detecting ambiguities: An optimistic approach to robustness problems in computational geometry. technical report UUCS-90-003, Computer Science Department, University of Utah, April 1990.

[2] B. Brüderlin. Robust regularized set operations on polyhedra. In *Hawaii International Conference on System Sciences*, January 1991.

[3] R. Byrne. *1,911 best things anybody ever said.* Ballantine Books, 1988.

[4] J. Canny. Some algebraic and geometric computations in pspace. In *Annual Symposium on the Theory of Computing*, pages 460–467, 1988.

[5] L. P. Chew. Guaranteed-quality triangular meshes. technical report 89-983, Computer Science Department, Cornell University, April 1989.

[6] M. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.

[7] D. Dobkin and D. Silver. Recipes for geometry & numerical analysis – Part I: An empirical study. In *ACM Annual Symposium on Computational Geometry*, pages 93–105, 1988.

[8] C. M. Eastman and K. Preiss. A review of solid shape modelling based on integrity verification. *Computer-Aided Design*, 16(2):66–80, 1984.

[9] H. Edelsbrunner. *Algorithms in Computational Geometry.* Springer-Verlag, 1987.

[10] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. In *ACM Annual Symposium on Computational Geometry*, pages 118–133, June 1988.

[11] J. D. Foley and A. Van Dam, editors. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

[12] S. Fortune. Stable maintenance of point set triangulations in two dimensions. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 494–499, 1989.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W H Freeman and Company, New York, 1979.

[14] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[15] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 143–152, October 1986.

[16] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31–41, March 1989.

[17] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *ACM Annual Symposium on Computational Geometry*, pages 106–117, June 1988.

[18] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, November 1989.

[19] J. E. Hopcroft and P. J. Kahn. A paradigm for robust geometric algorithms. technical report 89-1044, Computer Science Department, Cornell University, October 1989.

[20] D. Huffman. Impossible objects as nonsense sentences. In R. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 295–323. Elsevier, 1971.

[21] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21, 1973.

[22] M. Karasick. *On the Representation and Manipulations of Rigid Solids*. PhD thesis, McGill University, 1989.

[23] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. *Computer Graphics (SIGGRAPH Procedings)*, 20(4):161–170, 1986.

[24] Z. Li and V. J. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *ACM Annual Symposium on Computational Geometry*, pages 234–243, 1990.

[25] V. J. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*. PhD thesis, Carnegie Mellon University, 1988.

[26] V. J. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, 37:377–401, 1988.

[27] V. J. Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. In *ACM Annual Symposium on Computational Geometry*, pages 197–207, 1989.

[28] V. J. Milenkovic. Double precision geometry: A general technique for calculating line and segment intersections using rounded arithmetic. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 500–505, 1989.

[29] V. J. Milenkovic and L. R. Nackman. Finding compact coordinate representations for polygons and polyhedra. In *ACM Annual Symposium on Computational Geometry*, pages 244–252, 1990.

[30] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *ACM Annual Symposium on Computational Geometry*, pages 119–125, June 1987.

[31] F. P. Preparata and M. I. Shamos. *Computational Geometry, an Introduction*. Springer-Verlag, 1985.

[32] D. F. Rogers and R. A. Earnshaw, editors. *Techniques for Computer Graphics*. Springer-Verlag, 1987.

[33] D. Salesin. *Epsilon Geometry: Building Robust Algorithms From Imprecise Computations*. PhD thesis, Stanford University, 1991.

[34] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: Building robust algorithms from imprecise calculations. In *ACM Annual Symposium on Computational Geometry*, pages 208–217, 1989.

[35] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics (SIGGRAPH Procedings)*, 24(4):105–114, 1990.

[36] M. Segal and C. Séquin. Consistent calculations for solid modeling. In *ACM Annual Symposium on Computational Geometry*, pages 29–37, June 1985.

[37] M. Segal and C. Séquin. Partitioning polyhedral objects into nonintersecting parts. In *IEEE Computer Graphics and Applications*, pages 53–67, January 1988.

[38] K. Sugihara. A robust intersection algorithm based on delaunay triangulation. technical report 91-011, Computer Science Department, Purdue University, February 1991.

[39] K. Sugihara and M. Iri. Construction of the Voronoi diagram for over 10^5 generators in single-precision arithmetic. In *Abstracts of the First Canadian Conference on Computational Geometry*, page 42, August 1989.

[40] K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380–393, 1989.

[41] K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite-precision arithmetic. *Appl. Math. Lett.*, 2(2):203–206, 1989.

[42] K. Sugihara, Y. Ooishi, and T. Imai. Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms. In *Canadian Conference on Computational Geometry*, pages 36–39, August 1990.

[43] G. Vanecek, Jr. Obtaining boundaries with respect: A simple approach to performing set operations on polyhedra. technical report CSD-TR-920, Computer Science Department, Purdue University, October 1989.

[44] D. Waltz. Generating semantic descriptions from drawings of scenes with shadows. In P. Winston, editor, *The Psychology Of Computer Vision*, pages 19–92. McGraw-Hill, 1975.

[45] C. Yap. A geometric consistency theorem for a symbolic perturbation scheme. In *ACM Annual Symposium on Computational Geometry*, pages 134–142, June 1988.