Accurate Sampling-Based Algorithms for Surface Extraction and Motion Planning

by
Gokul Varadhan

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2005

Approved by:

Dinesh Manocha, Advisor

Ming Lin, Reader

Shankar Krishnan, Reader

Jack Snoeyink, Committee Member

Pankaj Agarwal, Committee Member

ABSTRACT
GOKUL VARADHAN: Accurate Sampling-Based Algorithms for Surface Extraction and Motion Planning.
(Under the direction of Dinesh Manocha.)

Boolean operations, Minkowski sum evaluation, configuration space computation, and motion planning are fundamental problems in solid modeling and robotics. Their applications include computer-aided design, numerically-controlled machining, tolerance verification, packing, assembly planning, and dynamic simulation. Prior algorithms for solving these problems can be classified into exact and approximate approaches. The exact approaches are difficult to implement and are prone to robustness problems. Current approximate approaches may not solve these problems accurately. Our work aims to bridge this gap between exact and approximate approaches. We present a sampling-based approach to solve these geometric problems. Our approach relies on computing a volumetric grid in space using a sampling condition. If the grid satisfies the sampling condition, our algorithm can provide geometric and topological guarantees on the output.

We classify the geometric problems into two classes. The first class includes surface extraction problems such as Boolean operations, Minkowski sum evaluation, and configuration space computation. We compute an approximate boundary of the final solid defined using these geometric operations. Our algorithm computes an approximation that is guaranteed to be topologically equivalent to the exact surface and bounds the approximation error using two-sided Hausdorff error. We demonstrate the performance of our approach for the following applications: Boolean operations on complex polyhedral models and low degree algebraic primitives, model simplification and remeshing of polygonal models, Minkowski sums and offsets of complex polyhedral models, and configuration space computation for low degrees of freedom objects.

The second class of problems is motion planning of rigid or articulated robots translating or rotating among stationary obstacles. We present an algorithm for complete motion planning, i.e., finding a path if one exists and reporting a failure otherwise. Our algorithm performs deterministic sampling to compute a roadmap that captures the connectivity of free space. We demonstrate the performance of our algorithm on challenging environments with narrow passages and no collision-free paths.

ACKNOWLEDGMENTS

First and foremost, I thank my advisor, Dinesh Manocha; without his guidance and support, I would not have come anywhere close to finishing a PhD. I owe Shankar Krishnan for being there as a coauthor, a friend, and a guide; his role has been nothing short of a co-advisor. Many thanks to Ming Lin for her involvement during the course of the research and for investing time and energy into it. I thank Jack Snoeyink and Pankaj Agarwal for their support and constructive feedback as committee members.

Most of my work has been done jointly with a number of collaborators. I am indebted to Young Kim for his role; it was our joint work that eventually led to my disseration topic. I also thank TVN. Sriram and Liangjun Zhang for their efforts. Many thanks to all the members of the GAMMA and Walkthru groups for their participation.

My work would not have been possible without the involvement of the excellent students, faculty, and staff of the Department of Computer Science. I thank Russell Taylor and members of nanoManipulator group for their support during my first year in graduate school. I am grateful to the staff – in particular, Missy Wood, Janet Jones, Karen Thigpen, and Charlie Bauserman – for their help. I thank the funding sponsors of my work: ARO, NSF, ONR, DARPA, and Intel Corporation.

I am indebted to all my teachers for their support and encouragement but most of all for inspiration. I thank all my friends in Chapel Hill – I do not dare to name all of them. Without them, my stay here would not have been worthwhile. I thank my uncle, Sridhar Raghavan, who has been a lifelong role model; my decision to do a PhD was probably an attempt to emulate him. Most of all, my parents and brother for their unconditional love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Geometric algorithms play a fundamental role in numerous application domains – computer graphics, solid modeling, bio-informatics, geographic information systems, robotics, computer vision, and others. Two key domains that are a rich source of geometric problems are solid modeling and robotics.

Solid modeling deals with the design and representation of physical objects. To model complex objects, a commonly used approach is to combine a set of simpler primitives using geometric operations. An important class of geometric operations are the Boolean operations – union, intersection, and difference. These operations serve as fundamental building blocks of solid modeling, and are used to design complex objects. Fig. 1.1 shows an example.

Robotics is the study of the technology associated with the design, theory and application of robots. One of the ultimate goals of robotics is to build autonomous robots (Latombe, 1991). Such robots will accept a high-level description of tasks and execute them without any human intervention. As the robot executes the tasks, it will need to move in the environment while avoiding collision with the obstacles in the environment. Thus, one of the requirements of an autonomous robot is that it should be able to plan its own motion, and research in motion planning is important for building autonomous robot systems.

A basic version of the motion planning problem assumes that the robot is the only moving object in a static workspace. The obstacles are rigid and static. The robot may translate, rotate, or have different types of joints imposing sliding or rotating constraints. The geometry of both the robot and the obstacles is known. Even this seemingly simple problem is challenging from a computational standpoint and has been a subject of considerable amount of research.

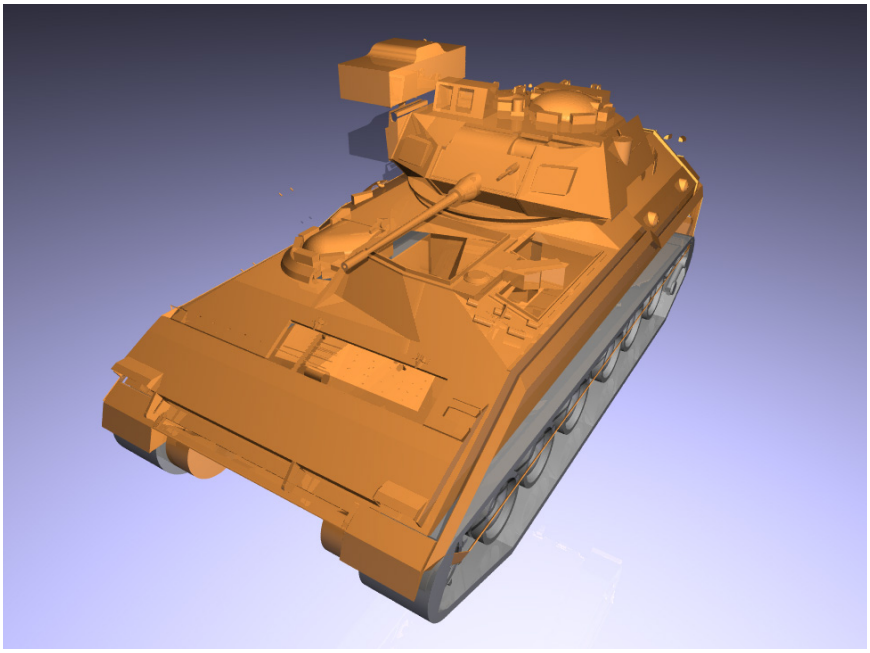


Figure 1.1: Bradley Fighting Vehicle: *This image shows a view of the Bradley Fighting Vehicle. This model has over* $8,500$ *solids generated entirely using Boolean operations.*

The above motion planning problem is often studied using a tool called the *configuration space*. The configuration space of a robot is the set of all possible positions and orientations that the robot can assume. Its dimension is equal to the number of degrees of freedom of the robot. Fig. 1.2 shows an example of a 2D configuration space. The underlying idea of configuration space is to represent the robot as a point and to map the obstacles to this space. The obstacles map to a region called the configuration space obstacle (C-obstacle). If a configuration belongs to C-obstacle, then the robot placed at that configuration collides with some obstacle. The complement of the C-obstacle is called the *free space* of the robot. Configuration space reduces the problem of motion planning of a dimensioned robot into the problem of planning the motion of a point within the robot's free space. Configuration space has played a crucial role in helping understand motion planning problems, and led to the development of many motion planning algorithms.

An important instance of the motion planning problem is when the robot is only allowed to translate. In this case, the C-obstacle is obtained by performing an operation called the *Minkowski sum* of the obstacle and the robot reflected about the origin. The Minkowski sum is a natural set operation that has been used in several fields.

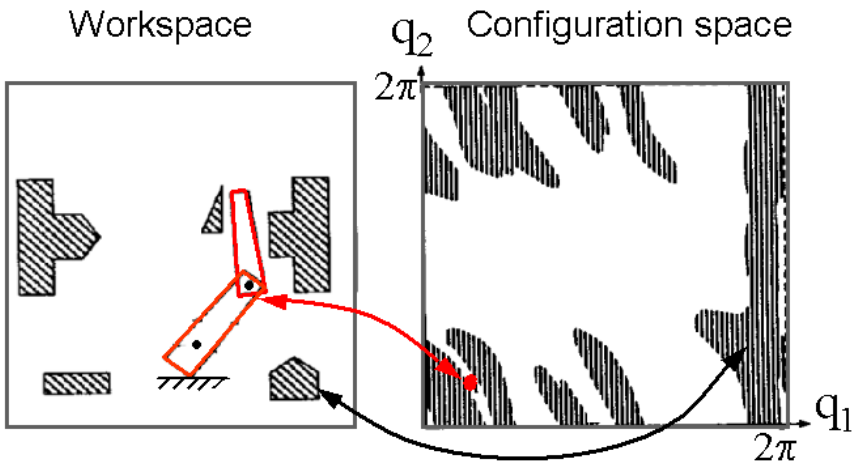


Figure 1.2: Configuration Space: *The left figure shows a planar articulated robot moving among polygonal obstacles (modified from (Latombe 1991)). The robot has two revolute joints. The right figure shows the 2-dimensional configuration space of the robot. As indicated by the arrows, robot (in its present configuration) maps to a point in the configuration space, and the obstacle maps to a forbidden region.*

Intuitively, the Minkowski sum of two objects can be considered as expanding one object by sweeping it over the other. Fig. 1.3 shows a 2D example. Formally, the Minkowski sum of two objects $\mathcal{P}$ and $\mathcal{Q}$ is defined as the set-sum, $\mathcal{P} \oplus \mathcal{Q} = \{\mathbf{p}+\mathbf{q} \mid \mathbf{p} \in \mathcal{P},\ \mathbf{q} \in \mathcal{Q}\}$. Apart from robotics, Minkowski sum has numerous applications in CAD/CAM as well.

The above geometric problems – Boolean operations, motion planning, configuration space computation, and Minkowski sum operation – are interesting not only because of the many applications in which they play a central role but also because of the computational challenges they present. We begin by examining each of the problems and their applications in more detail. We describe the prior work on these problems,

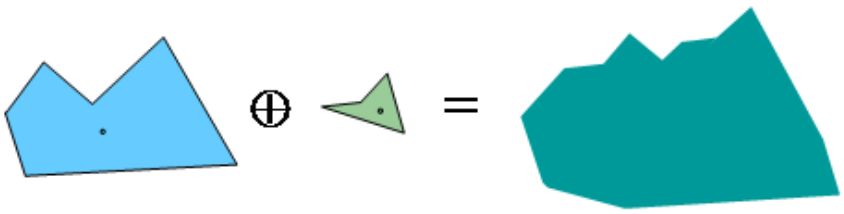


Figure 1.3: 2D Minkowski sum

and discuss the issues and challenges in their computation. Finally, we present an overview of our algorithms to solving these problems, and summarize the new results of this dissertation.

1.1 Boolean operations

Boolean operations play a fundamental role in solid modeling. By performing Boolean operations on simple objects, we can design more complex objects. This is the approach taken by Constructive Solid Geometry (CSG), a popular representation in solid modeling. CSG represents objects as a sequence of operations defined over a set of primitive solids. The primitive solids typically consist of simple objects such as a box, sphere, cylinder, torus, etc. The operations involve either rigid transformations or Boolean operations such as union, intersection, or difference. Fig. 1.1 shows a model of the *Bradley Fighting Vehicle*. This model has over $8,500$ solids generated entirely using Boolean operations.

The CSG representation uses a *regularized* version of the Boolean operations. The regularized Boolean operations differ from their corresponding set-theoretic counterparts in that the result is the closure of the operation on the interior of the two solids. Regularized operations are used to eliminate "dangling" lower-dimensional structures (Hoffmann, 1989a). If *op* denotes a set operation, the regularized version of *op* is defined as:

$$A op^* B = \text{cl}(\text{int } A op \text{ int } B)$$

where int S and cl S denote the interior and closure of a set S.

Apart from CSG, another representation commonly used in solid modeling is the boundary representation (B-rep). B-reps describe solids as a set of vertices, edges, and faces with topological relations among them. Both CSG and B-reps have different inherent strengths and weaknesses, and for most applications both are desired. For instance, a CSG object is always valid in the sense that its surface is closed, orientable and encloses a volume, provided the primitives are valid in this sense. A B-rep object, on the other hand, can be easily rendered on a graphic display system and is useful for visual feedback in solid design. A B-rep is also used for collision detection, dynamic simulation, and model verification.

Algorithms for performing the regularized union, intersection, or set difference of two solids can be used to convert solids represented by CSG representation to an equivalent

B-rep. Hence algorithms for Boolean operations on B-reps are also called *boundary evaluation* algorithms.

Boolean operations are also used as building blocks for other geometric operations. For example, the union operation can be used to implement a number of operations: the Minkowski sum operation; the *sweep* operation that moves an object along some trajectory while undergoing translational and rotational motion; the *offset* operation that grows an object outwards by a fixed distance.

1.2 Minkowski Sum

The Minkowski sum of two objects $\mathcal{P}$ and $\mathcal{Q}$ is formally defined as:

$$\mathcal{P} \oplus \mathcal{Q} = \{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in \mathcal{P},\ \boldsymbol{q} \in \mathcal{Q}\}$$

Intuitively, the Minkowski sum of two objects can be considered as expanding one object by sweeping it over the other.

Minkowski sum has numerous applications in robotics and manufacturing. One of the most important uses of the Minkowski sum is for motion planning of a robot translating among stationary obstacles (Lozano-Pérez, 1983). Minkowski sum is used to compute the free space of the robot. Then the problem of motion planning reduces to finding a path within the free space. Fig. 1.14 shows an example of 3D translational motion planning using the Minkowski sum.

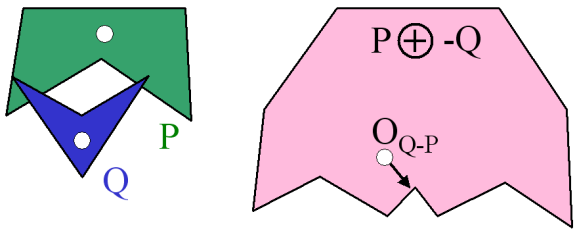


Figure 1.4: Penetration Depth: *The problem of penetration depth can be expressed in terms of Minkowski sum. The penetration depth is equal to the distance between the origin and the boundary of the Minkowski sum (Kim et al. 2002).*

Minkowski sum is also used to compute the penetration depth between two objects (Cameron, 1997; Kim et al., 2002). The penetration depth of two intersecting objects $\mathcal{P}$ and $\mathcal{Q}$, denoted as $PD(\mathcal{P}, \mathcal{Q})$, is the minimum translational distance that one of the

polyhedra must undergo to render them disjoint. Formally, $PD(\mathcal{P}, \mathcal{Q})$ is defined as:

$$\min\{\| \boldsymbol{d} \| \mid interior(\mathcal{P} + \boldsymbol{d}) \cap \mathcal{Q} = \emptyset\} \tag{1.1}$$

Here, $\boldsymbol{d}$ is a vector in $\mathcal{R}^3$. The problem of penetration depth computation arises in robotics, dynamic simulation, computer gaming, virtual environments, etc. One can reduce the problem of computing penetration depth between $\mathcal{P}$ and $\mathcal{Q}$ to a minimum distance query on the surface of their Minkowski sum, $\mathcal{P} \oplus -\mathcal{Q}$ (Cameron, 1997). See Fig. 1.4.

An important use of Minkowski sum is to perform *mathematical morphological operations*. The primary morphological operations, from which many others are derived, are dilation and erosion. Dilation of an object $\mathcal{P}$ by an object $\mathcal{Q}$, denoted as $\mathcal{D}(\mathcal{P}, \mathcal{Q})$, is same as the Minkowski sum $\mathcal{P} \oplus \mathcal{Q}$. $\mathcal{Q}$ is usually referred to as the structuring element. Erosion of an object $\mathcal{P}$ by the structuring element $\mathcal{Q}$, denoted as $\mathcal{E}(\mathcal{P}, \mathcal{Q})$, selects the locus of points swept by the origin of $\mathcal{Q}$ where $\mathcal{P}$ entirely contains the translated $\mathcal{Q}$. Erosion can be expressed in terms of the Minkowski sum operation as:

$$\mathcal{E}(\mathcal{P}, \mathcal{Q}) = \overline{\overline{\mathcal{P}} \oplus \mathcal{Q}'}$$

where $\overline{S}$ denotes the complement of a set S, and $\mathcal{Q}'$ denotes a copy of $\mathcal{Q}$ reflected about the origin.

The dilation and erosion operations can be composed to define other mathematical morphological operations such as *opening* and *closing*. Opening is an erosion followed by dilation, while closing is a dilation followed by erosion. They are defined as:

$$\begin{aligned} \text{Opening}(\mathcal{P}, \mathcal{Q}) &= \mathcal{E}(\mathcal{D}(\mathcal{P}, \mathcal{Q}), \mathcal{Q}) \\ \text{Closing}(\mathcal{P}, \mathcal{Q}) &= \mathcal{D}(\mathcal{E}(\mathcal{P}, \mathcal{Q}), \mathcal{Q}) \end{aligned}$$

While these operations have been mostly used in image processing (Serra, 1982), they are becoming increasingly popular in geometry processing (Rossl et al., 2000; Museth et al., 2002; Williams and Rossignac, 2004).

An interesting case of morphological operations is where the structuring element Q is a ball. In this special case, dilation creates an *offset* surface a distance r outwards from the original surface (Rossignac and Requicha, 1986). Here r is the radius of the

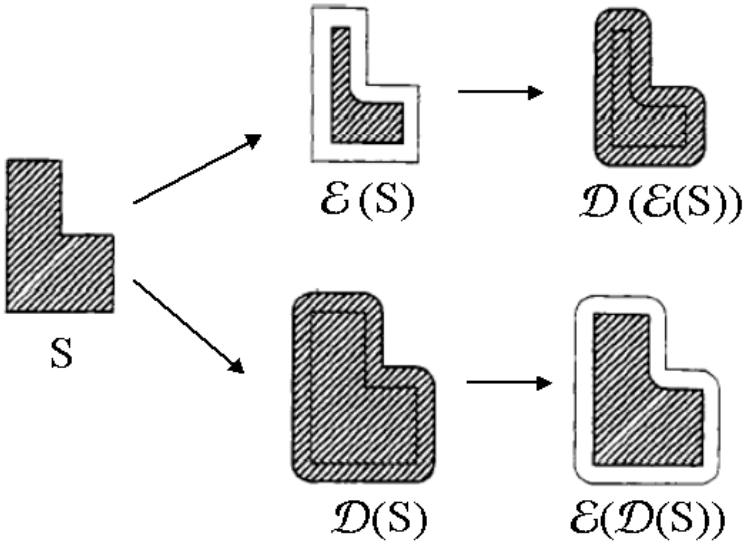


Figure 1.5: Minkowski sums for constant-radius rounding and filleting: (modified from (Rossignac and Requicha 1986)). *The top row of figures shows an example of a rounding operation, while the bottom row shows filleting.* $\mathcal{D}(S)$ *and* $\mathcal{E}(S)$ *denote dilation and erosion operations on a set* S *where the structuring element is a constant radius ball*

ball. Mathematically, this is defined as

$$\text{Offset}(\mathcal{P}) = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \exists \boldsymbol{p} \in \mathcal{P},\ \|\boldsymbol{x} - \boldsymbol{p}\| \leq r\}$$

See Fig. 1.12.

Similarly, erosion creates an offset surface a distance r inwards from the original surface. The opening and closing operations with a ball reduce to constant-radius *rounding* and *filleting* operations respectively (Rossignac and Requicha, 1986). See Fig. 1.5. These operations have applications in numerical control verification, tolerance analysis, mold/die making, and rapid prototyping.

Mathematical morphological operations are also used for postprocessing of models obtained by 3D scanning. The process of scannning is susceptible to both noise and insufficient scanning in certain parts of the model. Consequently, the model can have many artifacts: small holes, gaps, or isolated surface components. Opening and closing operations can be used to repair such models. Opening can be applied to eliminate unwanted small pieces or thin appendages. Closing operation can be applied to close small gaps or holes within objects (Museth et al., 2002).

Minkowski sum has been used for packing and layout in CAD/CAM (Boissonnat

et al., 1997; Daniels and Milenkovic, 1997); it can be used to determine if an object can be placed within a container without colliding with the other objects in the container. Other applications include morphing (Kaul and Rossignac, 1991) and friction modeling (Goyal et al., 1991).

1.3 Configuration Space

This section introduces the configuration space formulation for robot motion planning (Lozano-Pérez and Wesley, 1979; Lozano-Pérez, 1983). Let $\mathcal{R}$ be a robot consisting of a collection of rigid subparts moving in a Euclidean space W, called *workspace*, represented as $\mathbb{R}^d$. Let $\mathcal{O}_1, \ldots, \mathcal{O}_q$ be fixed rigid obstacles embedded in W. Assume that the geometry of $\mathcal{R}, \mathcal{O}_1, \ldots, \mathcal{O}_q$ is accurately known, and that there are no kinematic constraints to limit the motion of $\mathcal{R}$. The position and orientation of the subparts define the *configuration* of $\mathcal{R}$. The set of all configurations of $\mathcal{R}$ defines a configuration space $\mathcal{C}$. See Fig. 1.2.

Each position and orientation of the robot maps to a point in the configuration space. Every obstacle $\mathcal{O}_i, i = 1, \ldots, n$, in W maps to the region

$$\mathcal{CO}_i = \{\mathbf{q} \in \mathcal{C} : \mathcal{R}(\mathbf{q}) \cap \mathcal{O}_i \neq \emptyset\},$$

in $\mathcal{C}$, where $\mathcal{R}(\mathbf{q})$ is the subset of W occupied by $\mathcal{R}$ at the configuration $\mathbf{q}$.

The union of all $\mathcal{CO}_i$, $\bigcup_{i=1}^{q} \mathcal{CO}_i$ is called *C-obstacle region* or *forbidden region*. The set

$$\mathcal{F} = \mathcal{C} \setminus \bigcup_{i=1}^{q} \mathcal{CO}_i.$$

is called the *free configuration space* or the *free space*. Any configuration belonging to $\mathcal{F}$ is called a *free configuration*.

If the robot is only allowed to translate, then the C-obstacle of an obstacle $\mathcal{O}_i$ is given by the Minkowski sum: $\mathcal{CO}_i = \mathcal{O}_i \oplus -\mathcal{R}$. An example of such a C-obstacle in 3D is shown in the rightmost image of Fig. 1.14.

The problem of configuration space computation is to compute the free space $\mathcal{F}$. It is a classical problem in algorithmic robotics and computational geometry. It arises in several important applications such as motion planning (Avnaim and Boissonnat, 1989; Halperin, 2002a; Lozano-Pérez and Wesley, 1979; Sacks, 1999; Sacks, 2001), collision detection and distance computation (Gilbert et al., 1988; Dobkin et al., 1993),

layout and containment problems in manufacturing (Daniels and Milenkovic, 1995; Milenkovic, 1998), spatial reasoning (Xavier and LaFarge, 1997), assembly and task planning (Thomas et al., 2003), tolerance analysis and mechanism design (Joskowitz and Sacks, 1995).

1.4 Motion Planning

Motion planning is a fundamental problem in robotics. It is an essential part of an autonomous robot system. Motion planning also finds applications outside the realm of robotics, such as surgical planning, drug docking, molecular binding and protein folding, design for manufacturing, graphic animation, and computer games (Latombe, 1999).

In the basic version of the motion planning problem, it is assumed that the robot - a rigid or articulated object - is the only moving object in the workspace. The obstacles are rigid and static. It is assumed that the geometry of both the robot and the obstacles is known. The goal is to find a *collision-free path* - a path along which the robot can move without colliding with any obstacle. The robot may translate, rotate or have different types of joints imposing sliding or rotating constraints. Fig. 1.14 shows an example of a robot translating in 3D.

The motion planning problem can be formally stated in terms of the configuration space formulation. A *collision-free path* or a *free path* between two free configurations, $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, is a continuous map $\tau : [0,1] \rightarrow \mathcal{F}$ with $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$. Given an initial and a goal configuration, the basic motion planning problem is to generate a free path between the two configurations, if one exists, and to report a failure otherwise.

1.5 Prior Work and Challenges

In all the problems except motion planning, our goal is to compute *a surface*: In Boolean operations and Minkowski sum computation, we wish to compute a surface corresponding to the boundary of the final solid defined by the operation; in configuration space computation, we wish to compute the boundary of the free space. Hence we refer to this set of problems as *surface extraction* problems and treat them collectively. We will refer to the surface as the final surface and denote it as $\mathcal{E}$.

On the other hand, the problem of motion planning is different because there the goal is to determine whether a path exists between a start and a goal configuration. Hence we treat this problem separately.

In this section, we present the prior work on surface extraction problems and motion planning. We discuss many issues and challenges in their computation.

1.5.1 Surface Extraction Problems

At a broad level, the prior algorithms for surface extraction problems can be classified into exact or approximate approaches – depending on whether they compute an exact or an approximate representation of the final surface $\mathcal{E}$. We outline these approaches, and discuss their merits and limitations.

Exact Approach for Surface Extraction Problems

Boolean Operations: Boundary evaluation is a well studied problem in solid modeling. The heart of all boundary evaluation algorithms is a method for intersection curve computation. Given two primitives, these algorithms subdivide the surfaces of the primitives along the curve where the surfaces intersect. The subdivision yields a set of surface components. Thereafter, the boundary of the final solid is generated by combining certain surface components (Hoffmann, 1989a).

Most boundary evaluation algorithms follow a general framework. This framework was first introduced by Requicha and Voelcker (Requicha and Voelcker, 1985) to perform Boolean operations on polyhedra. Given two polyhedra, A and B, the conceptual structure of the algorithm (Hoffmann, 1989a) is as follows:

1. Determine which pairs of faces $f \in A$ and $g \in B$ intersect. If there are none, test for containment only and skip all other steps.

2. For each face $f \in A$ that intersects faces $g_i \in B$, compute the intersecting line segments between f and g_i's. The set of all intersecting line segments partitions the surface of face f . Determine the partitions of f that contribute to some of the surface area of the resulting solid.

3. Perform the same for all faces of B.

4. Assemble all the faces into the new solid.

This framework can be extended to also accommodate curve surface domains.

Boundary evaluation algorithms are not difficult conceptually, but their implementation requires substantial work for several reasons (Hoffmann, 1989a). Layers of primitive operations have to be designed. Accounting for the many special positions of incident structures in three dimensions can be tedious. Moreover, evaluating the intersection of curved surfaces is a difficult problem as it introduces nontrival mathematical problems and robustness problems.

Minkowski Sum Computation: Minkowski sum computation is a challenging problem. The 3D Minkowski sum of two non-convex objects with n features (vertex, edge, face) can have a combinatorial complexity as high as $O(n^6)$ (Dobkin et al., 1993). There are two general approaches for computing the Minkowski sum of general polyhedral models.

The first approach for computing Minkowski sum of general polyhedra reduces the problem to arrangement computation (Guibas et al., 1983; Kaul and Rossignac, 1991; Kaul and O'Connor, 1992; Ghosh, 1993; Basch et al., 1996). The *arrangement* of a finite collection Γ of geometric objects in $\mathbb{R}^d$, denoted as $\mathcal{A}(\Gamma)$, is the decomposition of $\mathbb{R}^d$ into relatively open connected cells of dimensions $0, \ldots, d$ induced by Γ, where each cell is a maximal connected set of points lying in the intersection of a fixed subset of Γ (Agarwal and Sharir, 2000). The first approach for Minkowski sum computation enumerates a set of surface primitives that form a superset of the boundary of the Minkowski sum. For example, the algorithm by (Guibas et al., 1983; Basch et al., 1996) is based on "convolution computation" defined on "polyhedral tracings". The convolution is a superset of the surface of the Minkowski sum, and can be computed in $O(n^2)$ time in the worst case where n is the number of faces of each polyhedron. In order to compute the actual boundary of the Minkowski sum, a 3D arrangement of the convolution needs to be computed. The set of 2-dimensional cells in the arrangement induces a partition of the convolution into a set of *surface components*. Given such a partition, it is possible to select a subset of the surface components to obtain the boundary of the Minkowski sum (Basch et al., 1996).

The second approach for computing Minkowski sum of general polyhedra reduces the problem to union computation (Lozano-Pérez, 1983; Evans et al., 1992). These algorithms decompose the polyhedral models into simpler objects whose Minkowski sum can be easily computed. In particular, there are efficient algorithms for computing the Minkowski sum of convex objects. Specifically, the algorithm by Guibas and Seidel

(Guibas and Seidel, 1987) can compute the Minkowski sum of two convex polyhedra in $O(n \log n + k)$ time, where n is the number of features (vertex, edge, face) in the two convex polyhedra and k is is the number of faces in the output. Hence one common approach is to resort to *convex decomposition* (Lozano-Pérez, 1983). This approach uses the following property of Minkowski sum: If $P = P_1 \cup P_2$, then $P \oplus Q = (P_1 \oplus Q) \cup (P_2 \oplus Q)$. The approach combines this property with convex decomposition for general polyhedral models:

1. Compute a convex decomposition for each polyhedron
2. Compute the pairwise Minkowski sums between all possible pairs of convex pieces in each polyhedron.
3. Compute the union of pairwise Minkowski sums.

After the second step, there are $O(mn)$ pairwise Minkowski sums where m and n are the number of convex pieces of the two polyhedra. The pairwise Minkowski sums are convex and their union can have a combinatorial complexity $O(k^3 + sk \log(s))$, where $k = mn$ and s is the number of total number of faces in the pairwise Minkowski sums (Aronov et al., 1997).

Configuration Space Computation: A general approach for configuration space computation proceeds by enumerating *contact surfaces* for every pair of features from the robot $\mathcal{R}$ and the obstacle $\mathcal{O}$. A contact surface of a geometric feature (vertex, edge, face) of $\mathcal{R}$ and a similar feature (vertex, edge, face) of $\mathcal{O}$ is defined as the set of points in the configuration space that represent configurations of $\mathcal{R}$ at which contact is made between these specific features.

The set Γ of contact surfaces define an arrangement $\mathcal{A}(\Gamma)$. The free space $\mathcal{F}$ consists of some cells in this arrangement. Therefore, $\mathcal{F}$ can be computed by computing $\mathcal{A}(\Gamma)$ (Avnaim and Boissonnat, 1989; Latombe, 1991; Halperin and Sharir, 1995b; Sacks, 1999). The combinatorial complexity of the entire $\mathcal{F}$ can be $O(n^k)$ where n is the number of contact surfaces in Γ and k is the dimension of the configuration space (Sharir, 1997). For many applications of configuration space computation (e.g. motion planning), it is not necessary to compute the entire $\mathcal{F}$; it may be sufficient to compute just a single connected component. Halperin and Sharir (Halperin and Sharir, 1995a) showed that the combinatorial complexity of a single cell of $\mathcal{A}(\Gamma)$ in three dimensions is $O(n^{2+\epsilon})$, for any $\epsilon > 0$, where the constant of proportionality depends on ϵ and on the maximum degree of the surfaces. They also proposed an algorithm than can compute

a single cell in $O(n^{2+\epsilon})$ time. Subsequently, Basu (Basu, 1998) extended the result of Halperin and Sharir to higher dimensions. Basu showed that the combinatorial complexity of a single connected component of $\mathcal{A}(\Gamma)$ in k dimensions is $O(n^{k-1+\epsilon})$. Furthermore, under a certain geometric assumption on the objects, this bound can be improved to $O(n^{k-1})$.

Both boundary evaluation and union computation can be treated as special cases of arrangement computation. Thus, we can define a common framework for all surface extraction problems:

1. Enumerate a set S of surface primitives that forms a superset of the final surface $\mathcal{E}$
2. Compute the arrangement $\mathcal{A}(S)$ of the surfaces in S.
3. Retain "appropriate" portions of $\mathcal{A}(S)$ to obtain $\mathcal{E}$.

Challenges/Issues: 3D arrangement computation is a major bottleneck in using the above framework. Arrangement computation requires computing intersections between pairs of surface primitives and is prone to problems in accuracy and robustness (Hoffmann, 2001). In our context, two additional factors contribute to the difficulty of arrangement computation. First, the number of surface primitives in the arrangement can be high. In Minkowski sum and configuration space computation, the arrangement may have $O(n^2)$ surfaces, where n is the number of features in the two objects. In offset computation, the arrangement consists of $O(n)$ surfaces. In our applications, the arrangement may consist of several thousands of surface primitives (see Chapters 4 & 5). Second, the surfaces in the arrangement are non-linear in configuration space and offset computation. Computing intersections between non-linear primitives is difficult to implement and expensive in practice.

Abrams and Allen (Abrams and Allen, 2000) report robustness problems in computing union of polyhedral models for swept volume computation. They use a commerical CAD system (ACIS geometric modeling system) to compute the union. They report a failure of the system when performing union of thousands of polyhedra.

To overcome accuracy problems, some researchers have proposed the use of exact arithmetic. These attempts have met with some success for polyhedral (Fortune, 1997) and low degree algebraic models (Keyser et al., 1997). However, these algorithms have been applied to inputs with only a small number of primitives. It is unclear whether these algorithms can scale upto inputs with thousands of primitives.

Overall, 3D arrangement computation remains a major bottleneck in exact computation of Minkowski sum, offset and configuration space. Consequently, all the practical algorithms for these problems have been limited to either inputs with a small combinatorial complexity or restricted cases (e.g. planar objects, convex objects).

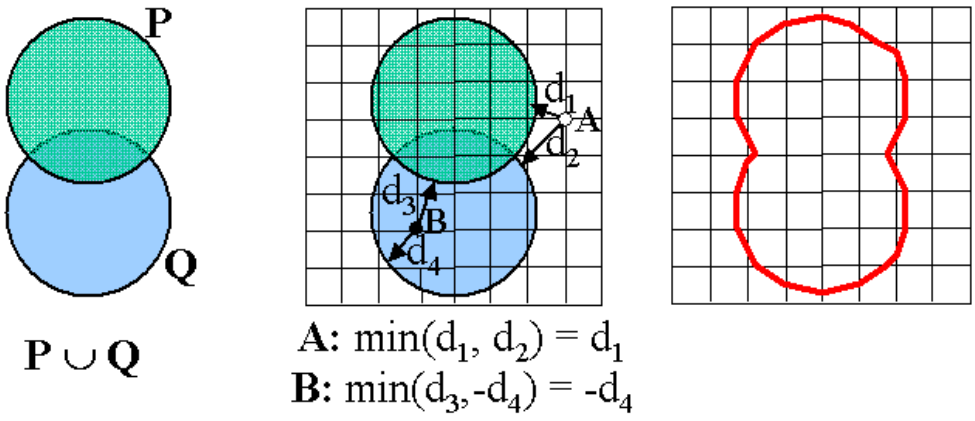
Approximate approach

An alternative to the exact approach is to use an approximate approach based on implicit representation. This representation uses a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to represent a closed surface. The function f is known as the *implicit function* or the *scalar field*. A commonly used scalar field is the *signed distance field*. For a closed surface $\mathcal{S}$, the signed distance field $D : \mathbb{R}^d \rightarrow \mathbb{R}$ is a continuous function that at a point $\boldsymbol{p}$ measures the distance between $\boldsymbol{p}$ and $\mathcal{S}$. This value is positive or negative depending on whether the point lies outside or inside $\mathcal{S}$. The distance can be defined under any reasonable norm (e.g., Euclidean, max-norm). $\mathcal{S}$ is the set of points $\boldsymbol{p}$ where $f(\boldsymbol{p}) = 0$ and is referred to as the *implicit surface*. Given a continuous scalar field f and a scalar value s, the *isosurface* with *isovalue* s is the set, $\{\boldsymbol{x} \in \mathbb{R}^d \mid f(\boldsymbol{x}) = s\}$, of points with identical scalar value s. Thus, the implicit surface is an isosurface of f with zero isovalue.

A common way of representing the scalar field is to discretize the continuous scalar field into discrete samples – to compute the value of the scalar field at the vertices of a volumetric grid. We refer to this step as *sampling* of the scalar field. The grid is an approximate representation of the scalar field; the accuracy of the approximate representation depends on the *rate of sampling* – the resolution of the grid.

An explicit boundary representation of the implicit surface can be obtained by extracting the zero-level isosurface using Marching Cubes (MC) (Lorensen and Cline, 1987) or any of its variants (Kobbelt et al., 2001; Ju et al., 2002; Varadhan et al., 2003b). We refer to these algorithms collectively as *MC-like* algorithms. The output of an MC-like algorithm is an approximation, usually a piecewise linear approximation, of the implicit surface. We refer to this step as *reconstruction* of the implicit surface.

Implicit surface representations are easy to use to perform geometric operations like union, intersection, difference, blending and warping. Specifically, they map Boolean operations into simple minimum/maximum operations on the scalar fields of the primitives. Suppose we have two primitives $\mathcal{P}_1$ and $\mathcal{P}_2$ with scalar fields f_1 and f_2. Then



(a) Union (b) Sampling and Operation (c) Reconstruction

Figure 1.6: Sampling and reconstruction: *This figure shows how to perform a union operation using the sampling and and reconstruction approach. The sampling step generates a volumetric grid (shown as a uniform grid). At each grid point, it computes a signed distance to the boundaries of each of the two primitives,* **P** *and* **Q**. *Next, a minimum operation is performed on the two signed distances. This generates another distance field on which the reconstruction is performed using an MC-like algorithm. The rightmost figure shows the reconstruction, which is an approximation to the union.*

we have

$$\begin{aligned} \boldsymbol{p} \in \partial(\mathcal{P}_1 \cup \mathcal{P}_2) &\iff \min(f_1(\boldsymbol{p}), f_2(\boldsymbol{p})) = 0 \\ \boldsymbol{p} \in \partial(\mathcal{P}_1 \cap \mathcal{P}_2) &\iff \max(f_1(\boldsymbol{p}), f_2(\boldsymbol{p})) = 0 \\ \boldsymbol{p} \in \partial(\mathcal{P}_1 \setminus \mathcal{P}_2) &\iff \max(f_1(\boldsymbol{p}), -f_2(\boldsymbol{p})) = 0 \end{aligned}$$

Because of the above property, implicit representations are frequently used to perform Boolean operations. They have been used for different applications including geometric modeling, volume rendering, morphing, path planning, swept volume computation, and sculpting digital characters (Wyvill et al., 1986; Bloomenthal and Ferguson, 1995; Pasko et al., 1995; Wyvill and van Overveld, 1996; Breen et al., 2000; Kobbelt et al., 2001; Perry and Frisken, 2001; Ju et al., 2002; Ohtake et al., 2003).

The above desirable properties of implicit surface representations make them suitable for surface extraction problems. We exploit these properties in sampling and reconstruction steps to obtain a boundary representation for the final surface $\mathcal{E}$. Consider

a surface $\mathcal{E}$ defined by Boolean operations over a set of primitives. We represent $\mathcal{E}$ implicitly – as an isosurface of a scalar field obtained by performing minimum/maximum operations over the scalar fields associated with the primitives. The overall approach proceeds as follows:

1. **Sampling:** Generate a volumetric grid and compute a scalar field (e.g, a signed distance field) at its corner grid points.

2. **Operation:** For each geometric operation (union/intersection), perform an analogous operation (e.g., min/max) on the scalar fields of the primitives. At the end of this step, the scalar values at the grid points define a sampled scalar field for $\mathcal{E}$.

3. **Reconstruction:** Perform isosurface extraction using Marching Cubes (Lorensen and Cline, 1987) or its variant to obtain a piece-wise linear approximation of $\mathcal{E}$.

This approach is illustrated in Fig. 1.6. The above approach can be extended to apply to all surface extraction problems.

Two important advantages of the above approach are simplicity and efficiency. Each step is easy to implement. A uniform grid or an adaptive grid (e.g. octree) may be chosen. Geometric operations such as union or intersection are cheap – we only need to perform simple min/max operations on the corresponding scalar fields; there is no need to perform expensive arrangement computation. Isosurface extraction is also reasonably straightforward; the MC-like algorithms are both simple and fast. Many public domain implementations of MC-like algorithms (Schroeder et al., 1997) are available.

Challenges/Issues: The above approach produces an approximation to the final surface $\mathcal{E}$. The accuracy of the approximation mainly depends on the resolution of the underlying grid. Insufficient grid resolution can result in a poor approximation. The output of Marching Cubes may suffer from various kinds of geometric and topological errors. Small components or handles present in $\mathcal{E}$ may not be captured in the output. The process of reconstruction may also introduce "extraneous topology", i.e., the reconstructed surface may have unwanted additional components or undesirable handles that were not present in $\mathcal{E}$. Fig. 1.7 shows an example of such a situation.

The above problems occur on account of inadequate resolution of the grid. Therefore, to alleviate these problems, many applications generate samples on a fine grid. However, the use of fine grid can result in three problems. First, there may still be no guarantees on the accuracy of the reconstructed isosurface. Second, a fine grid increases

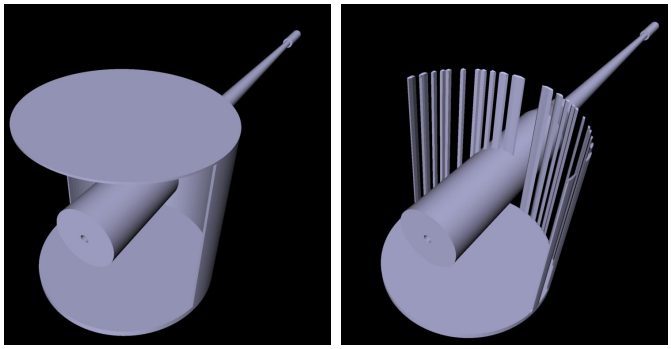


Figure 1.7: Accuracy problems with MC-like methods: *This figure highlights the errors that can be present in the output when MC-like methods are used to reconstruct surfaces with thin features. The left image shows the "gun model" of the Bradley Fighting Vehicle, which is generated using 8 Boolean operations. The right image shows the output of a MC-like algorithm (dual contouring) on a distance field sampled on a uniform* $64 \times 64 \times 64$ *grid. The output has many artifacts such as unwanted holes and extraneous handles.*

the storage overhead and the reconstructed surface can have a high number of polygonal primitives. Finally, it is computationally expensive to use a fine grid. Recent work on adaptive grid generation and subdivision algorithms overcomes some of these problems (Frisken et al., 2000; Perry and Frisken, 2001). However, none of these algorithms give rigorous guarantees on the accuracy of the reconstructed isosurface.

1.5.2 Motion Planning

The prior algorithms for motion planning can be classified as *complete* or *approximate* algorithms.

Complete Algorithms

Some of the early work was on developing algorithms for *complete* motion planning. An algorithm is complete, if it is guaranteed to find a solution when one exists and to return a failure otherwise. In 1979, Reif showed that path planning for a 3-D linkage made of polyhedral links is PSPACE-hard (Reif, 1979b). His analysis provides strong evidence that any complete planner will run in exponential time in the number of degrees of freedom (dofs).

There are two general algorithms for complete motion planning. The first algorithm, proposed by Schwartz and Sharir (Schwartz and Sharir, 1983a), is based on a *cylindrical*

algebraic decomposition of the robot's free configuration space (known as the Collin's decomposition). This algorithm takes time doubly exponential time in the number of degrees of freedom. Later, Canny (Canny, 1988) developed an improved algorithm based on computing a representation of the robot's free space as a network of one-dimensional curves. This representation is called the *roadmap* of the free space. This method runs in time singly exponential in the number of dofs. We are not aware of any implementations of either of these algorithms.

Many complete algorithms have been developed for specific instances of the motion planning problem, mainly for robots with 2-3 dofs. A number of algorithms have been proposed for path planning of polygonal robots moving among polygonal obstacles (Kedem and Sharir, 1990; Avnaim and Boissonnat, 1989; Sacks, 1999; Flato and Halperin, 2000). Algorithms have also been proposed for planning the motion of a polyhedron translating in 3-space amidst polyhedral obstacles (Lozano-Pérez and Wesley, 1979; Aronov and Sharir, 1994). Some of these specific algorithms have been implemented. These algorithms rely either on arrangement or "criticality" computation (Latombe, 1991). These computations are inefficient and may not be robust to floating-point approximations. As a result, the applicability of these algorithms has been limited (Latombe, 1999).

Approximate Algorithms

The high complexity and difficulty of implementing complete path planners have motivated the development of approximate planners. Two approximate approaches were introduced in the 80's: approximate cell decomposition (Latombe, 1991) and potential field method (Latombe, 1991). Both approaches are *resolution-complete*, i.e., whenever a path exists, they find one if the resolution parameters of the underlying grid are chosen fine enough. However, due to insufficient resolution, they may fail to find a path even if one exists. They have been used to solve complex path planning problems in 2-D and 3-D configuration spaces. However, they do not extend well to robots with more than 4 or 5 dofs.

The need to perform high dof planning motivated the development of randomized sampling based planners (Kavraki and Latombe, 1994; Overmars and Svestka, 1995). This approach starts by randomly computing a set of *samples*, i.e., a set of collision-free configurations in the configuration space. Then nearby samples are interconnected by computing "local paths", thus creating a probablistic roadmap (PRM) of the free space. This approach avoids the expensive computation of an explicit representation of

the free space. Therefore, this approach is simple, general, and efficient in practice. It has been successfully applied to do motion planning for robots with very high degrees of freedom. Many of the recent practical motion planning algorithms have adopted the randomized sampling approach (Amato et al., 1998; Wilmarth et al., 1999; Simeon et al., 2000; LaValle and Kuffner, 2000).

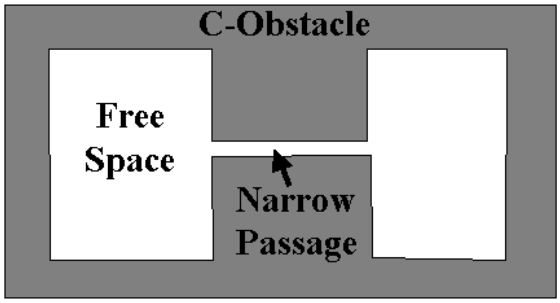


Figure 1.8: Narrow Passage: *The figure shows an example of a narrow passage in free configuration space.*

Challenges/Issues: There are two main issues with the randomized sampling based methods. First, these methods may not be able to find paths through "narrow passages" in the free space. See Fig. 1.8. Intuitively, a narrow passage is a small region whose removal changes the connectivity of the free space. A formal characterization of narrow passages is given in (Barraquand et al., 1997; Hsu et al., 1999). To capture the connectivity of the free space accurately, the planner must sample configurations in the narrow passages. This is difficult, because narrow passages have small volumes, and the probability of drawing random samples from small sets is low. Consequently, the randomized sampling based methods may not be able to find a valid path even though one may exist. A number of extensions have been proposed to improve the sampling to better handle narrow passages (Amato et al., 1998; Wilmarth et al., 1999; Hsu et al., 2003a). While these methods can find paths through narrow passages in many situations, they do not give guarantees. The second issue with randomized sampling based methods is that they do not terminate when no path exists between the initial and goal configuration. This is because the planner cannot detect non-existence of collision-free path.

1.6 Goals

In the previous sections, we briefly described prior approaches for surface extraction problems and motion planning. We looked at both exact and approximate approaches, and discussed their merits and limitations. We described two approximate algorithms: implicit modeling method for surface extraction problems and randomized sampling based method for motion planning. Both share some similarities; the underlying principle in both algorithms is the same – sampling. Hence we will refer to them as sampling based algorithms.

Neither exact nor sampling based approach by themselves are satisfactory. While the exact approach provides accuracy, it lacks in simplicity, and is prone to robustness problems. On the other hand, the sampling based approach is simple to implement but suffers from accuracy problems. Thus, there exists a gap between the two approaches, and our work aims to bridge this gap. Our goal is to achieve an accuracy "close to" that of the exact approach while retaining the simplicity of the sampling based approach.

We use the sampling based approach for both surface extraction problems and motion planning. Our goal is to design sampling based algorithms that can provide guarantees on the output. For surface generation problems, we wish to obtain an approximation that is topologically equivalent to the exact surface and has a bounded two-sided *Hausdorff error*.

Two objects $\mathcal{P}$ and $\mathcal{Q}$ are topologically equivalent if there exists a continuous bijective mapping between them with a continuous inverse (Munkres, 1975). The *one-sided Hausdorff* distance between $\mathcal{P}$ and $\mathcal{Q}$ is defined as follows:

$$h(\mathcal{P}, \mathcal{Q}) \quad = \quad \max\{\min d(\boldsymbol{p}, \mathcal{Q}) \mid \boldsymbol{p} \in \mathcal{P}\}$$

where $d(\boldsymbol{p}, \mathcal{Q})$ is the distance between a point $\mathbf{p}$ and a set $\mathcal{Q}$. The *two-sided Hausdorff* distance is defined as the maximum of $h(\mathcal{P}, \mathcal{Q})$ and $h(\mathcal{Q}, \mathcal{P})$. We use the term *Hausdorff error* to mean the two-sided Hausdorff distance between the approximation and the exact surface.

Guarantees such as bounds on the Hausdorff error and correct topology are not only important from a theoretical point of view, but also from a practical standpoint. The Hausdorff error measures the deviation of the approximation from the exact surface. Therefore, bounding this error is important to ensure a geometrically close approximation. Preserving the topology is also important in many applications. In CAD, topological features such as tunnels often correspond to distinguishing characteristics

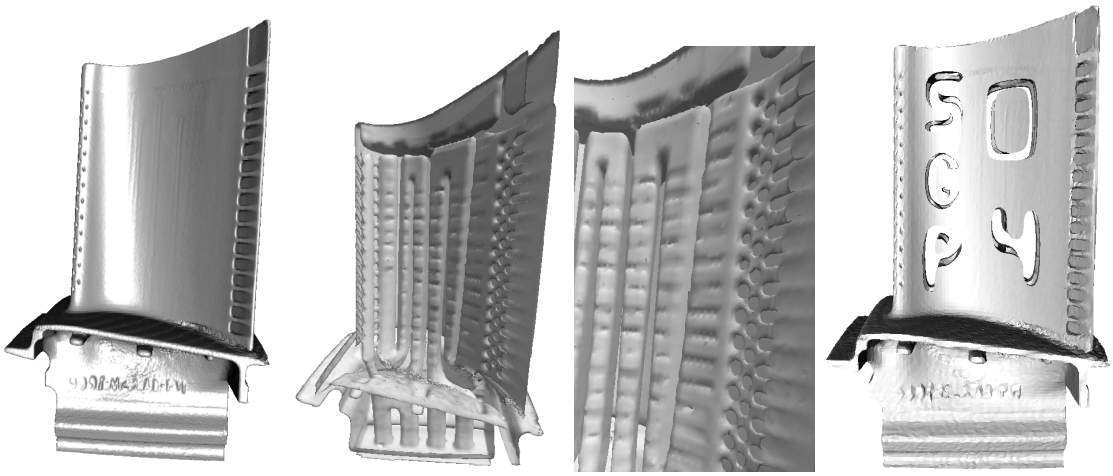


Figure 1.9: Simplification and Boolean operations: *This* $1.7M$ *triangle model of a turbine has a high genus and many features in the interior. We highlight the application of our algorithm to simplification and Boolean operations on this complex model. The simplified model of the turbine has* $511K$ *triangles and we show a zoomed view in the center-right image. We perform five difference (Boolean) operations on the turbine model and reconstruct the boundary of the final solid. Our algorithm produces a geometrically close and topology preserving approximation to the final solid. Overall, our algorithm can perform such geometric computations on complex models in tens of seconds and give rigorous guarantees in terms of preserving the topology of the final surface.*

of the model (Fig. 1.10). The geometric models used to represent the organs in medical datasets often consist of handles (Fig. 3.26). Retaining these topological features can be necessary in order to preserve the anatomical structure of the organ. This can be crucial for visualization and analysis. The geometric model of a molecule may consist of tunnels, which often act as atomic sieves that can aid the biochemical processes. Preserving these features can be important for rational drug design. In robotics, the tunnels may correspond to narrow passages through which the robot may pass to reach the goal.

Apart from capturing important features present in the original surface, guaranteeing topology is important for another reason. An algorithm that preserves topology avoids the introduction of extraneous topology; its output does not have unwanted additional components or handles, and is immune from the kinds of errors shown in Fig. 1.7.

For motion planning, our goal is to design a *complete* sampling based motion planning algorithm. Such a planner will be able to find a collision-free path whenever one

Figure 1.10: Remeshing: *The left image shows a brake hub model, which has many "skinny" triangles with poor aspect ratios (center image). We used our algorithm to compute an improved triangulation of the model (right image). The original brake hub model has* $14K$ *triangles. Our algorithm took* 1.85 *secs to perform remeshing and generate a mesh with* $7K$ *triangles at a relative Hausdorff error of* $1/32$.

exists; in particular, it will be able to find paths even through narrow passages. The planner will also be able to detect the non-existence of collision-free path and terminate.

1.7 Thesis Statement

Sampling algorithms can solve surface extraction and motion planning problems, and provide geometric and topological guarantees on the output.

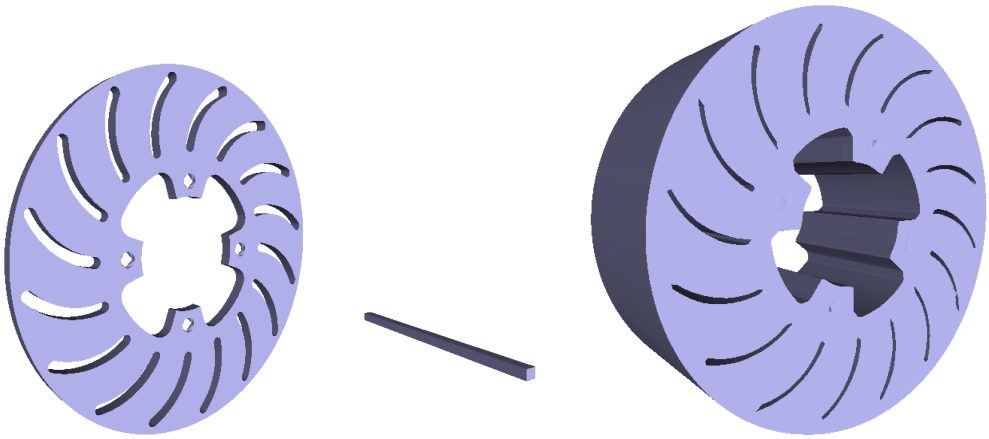
1.8 New Results

We now present the main results of this thesis.

A Sampling Based Method for Topology Preserving Isosurface Extraction

We propose a sampling condition for topology preserving isosurface extraction. Our sampling condition requires that every grid cell satisfy three criteria – a *complex cell* criterion, a *star-shaped* criterion, and a *non-degeneracy* requirement. These criteria will be defined in Chapter 3. We show that these criteria are sufficient to ensure that the isosurface reconstructed from the grid using MC-like methods is topologically equivalent to the exact isosurface.

Furthermore, we can augment the above sampling condition to also bound the geometric error in the reconstructed isosurface. Given any $\epsilon > 0$, we can ensure that the two-sided Hausdorff distance between the reconstructed isosurface and the exact isosurface is less than ϵ.

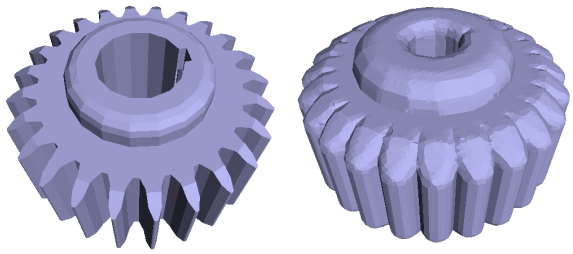


(a) Brake Hub (b) Rod (c) Minkowski Sum Approximation

Figure 1.11: Minkowski Sum Computation: *This example shows the application of our Minkowski sum approximation algorithm. The rightmost image shows an approximation to the Minkowski sum of* Brake Hub *and* Rod *models, which consist of* $4,736$ *and* 24 *triangles respectively. The final Minkowski sum has a number of narrow tunnels that contribute to a high genus. Our algorithm produced an approximation that preserved these features. It took* 140 *secs to compute a topologically correct approximation with a relative two-sided Hausdorff error of less than* $1/64$*. The relative Hausdorff error is defined as the ratio of the absolute Hausdorff error to the maximum length of a "tight" axis-aligned bounding box around the object.*

We generate a volumetric grid satisfying the sampling condition by using an adaptive subdivision algorithm. The subdivision algorithm generates an adaptive volumetric grid by a recursive application of the sampling condition. If a grid cell satisfies the sampling condition, the subdivision algorithm returns the grid cell as a leaf node of the adaptive volumetric grid. Otherwise, the grid cell is subdivided and the sampling condition is recursively applied to each of the children cells. In the absence of degeneracies, the adaptive subdivision algorithm will terminate once all the grid cells satisfy the sampling condition. The isosurface extraction is extracted from the resulting grid using an MC-like method. The output of isosurface extraction has a bounded two-sided Hausdorff error and is topologically equivalent to the exact isosurface.

The adaptive subdivision algorithm does not terminate in certain degenerate cases. We discuss this further and suggest possible ways of dealing with them. We also analyze the behavior of the subdivision algorithm and provide sufficient conditions for its termination.



(a) Gear (b) Offset Approximation

Figure 1.12: Offset Computation: *This example shows the application of our Minkowski sum approximation algorithm to offset computation of polyhedral models. The images show a gear-shaped model and an approximate offset computed by our algorithm. This model has* $2,382$ *triangles. Our approximation algorithm took* 84 *secs to compute a topologically correct approximation with a relative two-sided Hausdorff error of less than* $1/64$.

Our algorithm is relatively simple to implement. It primarily relies on *max-norm* distance computation, linear programming, and interval arithmetic. These computations can be performed efficiently for polyhedral and low-degree algebraic primitives.

We use our algorithm for accurate boundary evaluation of Boolean combinations of polyhedra and low degree algebraic primitives, model simplification, and remeshing. See Figs. 1.9 and 1.10.

The running time of our algorithm varies between a few seconds for simple models composed of a few thousand triangles to a few tens of seconds for complex polyhedral models represented using hundreds of thousands of triangles.

Accurate Minkowski Sum Approximation

We present an algorithm to approximate the 3D Minkowski sum of polyhedral objects. Our algorithm decomposes the polyhedral objects into convex pieces, generates pairwise convex Minkowski sums, and computes their union. We compute the union approximately by applying our sampling based isosurface extraction algorithm. The geometric and topological guarantees of our isosurface extraction algorithm apply to the Minkowski sum approximation as well.

The number of primitives in the union operation tends to be $O(n^2)$ where n is the

number of polygons in the input polyhedral objects. Our input models require a union of tens of thousands of primitives. The high number of primitives can pose a major overhead to the isosurface extraction algorithm. We reduce this overhead by employing two types of culling techniques. Our algorithm employs *primitive culling*, and performs efficient distance and inside/outside queries by considering only a small subset of primitives, while preserving the correctness of these queries. Our algorithm also performs *cell culling* to eliminate the grid cells that do not contain a part of the Minkowski sum boundary. In practice, these culling techniques improve the performance of the algorithm by more than *two orders* of magnitude.

Our algorithm is relatively simple to implement. We have used it approximate the Minkowski sum and offset of complex polyhedral models. Figs. 1.11 and 1.12 highlight some of our results.

Accurate Configuration Space Approximation

We present an approximate algorithm for configuration space computation. The free space is formulated as an arrangement over a set of contact surfaces. We enumerate the contact surfaces, and compute an approximate free space boundary expression using our sampling based isosurface extraction algorithm. We present computational techniques for applying the complex cell and star-shaped tests to the free space without explicit free space computation. The geometric and topological guarantees of our isosurface extraction algorithm apply to the free space approximation as well.

We apply our algorithm to two specific instances of motion planning. In each instance, the robot has three degrees of freedom (dofs).

- **3T :** A 3D robot translating among polyhedral obstacles. This instance reduces to computing the Minkowski sum of the robot (reflected about its origin) and the obstacles. Hence we use our Minkowski sum approximation algorithm for this instance.

- **2T+1R :** A planar rigid robot with 2 translational and 1 rotational dofs moving among polygonal obstacles.

See Figs. 1.14 and 5.7.

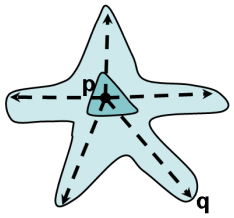


Figure 1.13: Star-shaped Primitive: *The figure shows a star-shaped primitive. Point P is a guard of the kernel. In general, the guard in not unique and can be any point belonging to a set called the* kernel *(shaded region in the interior of the primitive).*

Star-shaped Roadmap Method for Complete Motion planning

We present a new sampling based method for complete motion planning. Our method relies on computing a *star-shaped* roadmap of the free space. Informally, a region is *star-shaped* if there exists a point (called the guard) in the region that can *see* every point in the region. See Fig. 1.13. Sec. 3.1 provides a precise definition. The star-shaped roadmap is constructed by computing a star-shaped decomposition of the free space. This produces a set of *guards* that capture the *intra-region* connectivity – the connectivity between points belonging to the same star-shaped region. The *inter-region* connectivity is captured by computing *connectors* that connect guards of adjacent regions. The guards and connectors are combined to obtain the star-shaped roadmap.

We use an adaptive subdivision algorithm for constructing the star-shaped roadmap without explicit computation of the free space. The adaptive subdivision algorithm applies the star-shaped criterion in a recursive manner. In the absence of degeneracies in the free space, the adaptive subdivision algorithm will terminate and produce a star-shaped roadmap as the output. The resulting star-shaped roadmap captures the complete connectivity of the free space, thus enabling complete path planning. The roadmap can be used not only to find a path, but also detect non-existence of any collision-free path.

We apply our algorithm to three specific instances of the general motion planning problem. Two of these are **3T** and **2T+1R**. Additionally, we also apply our algorithm to **3R** - a planar articulated robot with 3 revolute joints moving among polygonal obstacles. We demonstrate the performance of our algorithm on environments containing narrow passages or no collision-free paths. Figs. 1.14, 5.7, 5.10 highlight the application of our algorithm to different environments.

1.9 Organization

The rest of the thesis is organized as follows:

- **Chapter 2** surveys the related work in the areas of arrangements, boundary evaluation, isosurface extraction, Minkowski sum computation, motion planning, and free space computation.
- **Chapter 3** presents an algorithm for topology preserving isosurface extraction.
- **Chapter 4** presents an approximate algorithm for Minkowski sum computation.
- **Chapter 5** presents an algorithm for configuration space approximation and the star-shaped roadmap method for motion planning.
- **Chapter 6** suggests directions for future research and concludes this dissertation.

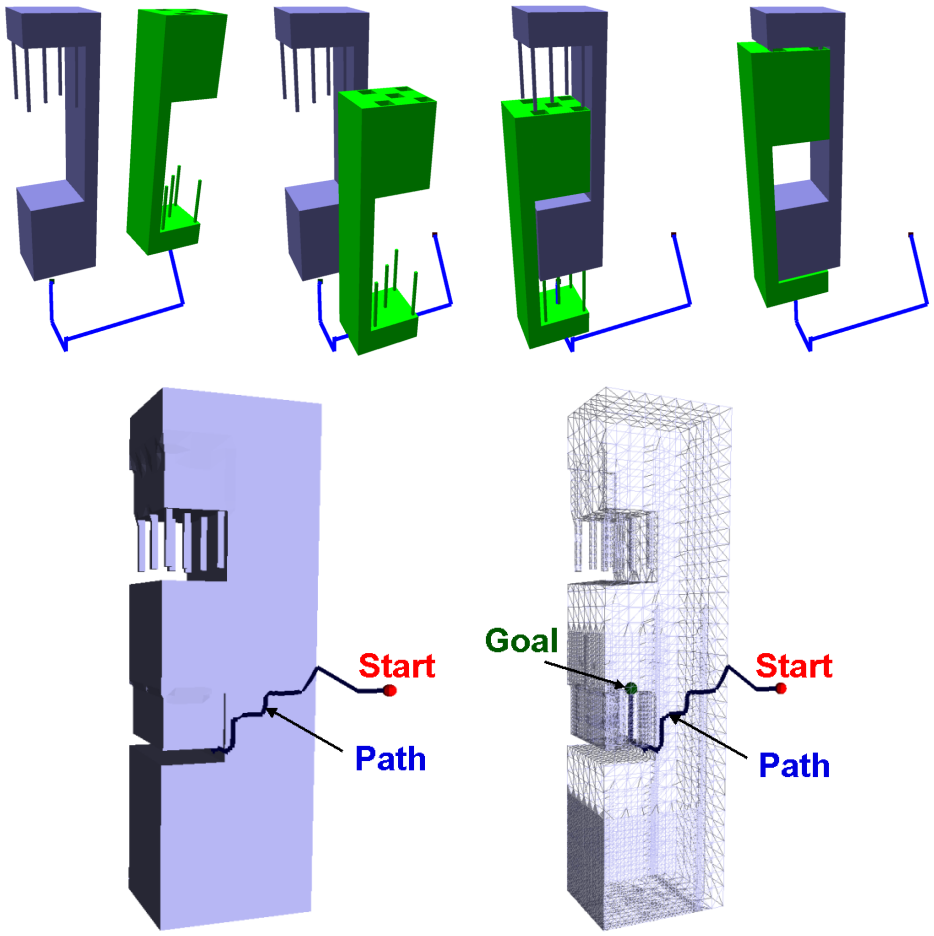


Figure 1.14: Motion Planning and Configuration Space Computation: *This example demonstrates the application of our motion planning and configuration space approximation algorithms to assembly planning. There are two parts each with pegs and holes. The goal is to assemble the two parts so that the pegs of one part fit into the holes of the other. The part is allowed to translate in 3D. This problem can be reduced to a motion planning problem by treating one of the parts as a robot and the other as the obstacle. The top row of images show a collision-free path computed by our motion planning algorithm. Our algorithm took* 15.9 *secs to construct a star-shaped roadmap and was able to find a collision-free path (shown in blue) in* 0.22 *secs. We also applied our configuration space approximation algorithm to this example. The bottom row of images show two views of the free configuration space approximation. The collision-free path computed by our motion planning algorithm is also shown in these images. This is a challenging example because the goal configuration is lodged within a narrow passage in the free configuration space.*

Chapter 2

Related Work

In this chapter, we discuss the prior work on boundary evaluation, arrangements, isosurface extraction, Minkowski sum evaluation, configuration space computation, and motion planning. Each of these problems have been a subject of a considerable amount of research. An exhaustive survey of all the methods is too extensive for this dissertation. Hence we focus only on specific instances of these problems and refer the reader to more comprehensive surveys.

2.1 Boundary Evaluation

Boundary evaluation refers to the process of performing Boolean operations on solids and recovering a boundary representation of the final solid defined by the operation. There has been a great deal of work on boundary evaluation. We only give a brief overview of the subject and refer the reader to (Hoffmann, 1989b; Krishnan, 1997; Keyser, 2000) for a more comprehensive treatment.

A large number of algorithms for boundary evaluation of polyhedra have been proposed over the years (Okino et al., 1973; Voelcker, 1974; Hillyard, 1982; Sugihara and Iri, 1989; Fortune, 1997). Few algorithms for curved solids have also been presented (Fang et al., 1993; Chun-Yi et al., 1996; Krishnan et al., 1997; Keyser et al., 1997). Most boundary evaluation algorithms follow a common approach. This approach involves several steps. First, each patch of the first solid is intersected with each patch of the second solid. To improve performance, additional data structures such as bounding boxes may be used to prune out many non-intersecting pairs of patches. A pair of intersecting patches results in an intersection curve in the domain of each patch. Next, the intersection curves from different patches are merged together. The merged inter-

section curve partitions the boundary of the solid into various components. Finally, a subset of these components are retained to obtain the boundary of the final solid.

While boundary evaluation algorithms are not difficult conceptually, two issues make their implementation difficult in practice. First, the underlying computations are prone to accuracy and robustness problems. Second, evaluating the intersection of curved surfaces is a difficult problem as it introduces nontrival mathematical problems, the problem of numerical precision, and stability of calculations (Hoffmann, 1989b; Hoffmann, 2001). We discuss each issue separately and give an overview of the prior work.

2.1.1 Accuracy and Robustness Problems

The robustness problem refers to the tendency of boundary evaluation algorithms to fail on certain inputs. There are two main types of robustness problems: numerical error and degenerate data.

Numerical Error

The first type of problem occurs due to the use of floating-point arithmetic in geometric computations. This problem has been summed succinctly by Hoffmann in his survey paper (Hoffmann, 2001):

"In geometric computations, logical facts such as incidence, separation, tangency, etc., are deduced based on numerical calculations. Further inferences are drawn from these deductions. The imprecision of floating-point arithmetic makes the conclusions drawn from the numerical calculations unreliable. In particular, the same logical questions may arise repeatedly, in the form of different floating-point computations giving contradictory answers. Unless this problem is avoided, it is not possible to write fully correct/robust/reliable code for geometric operations using standard floating-point arithmetic."

The above problem has received considerable amount of attention in recent years (Hoffmann, 2001). A common method for dealing with numerical error is to use tolerances, which allow two values that are within a certain fixed amount of each other to be considered the same (Segal, 1990; Jackson, 1995). While this can often reduce robustness problems, there are still many problems with using tolerances in boundary evaluation. It may not be possible to define a global value of tolerance that works well for all cases. Defining several values of tolerances and managing them can require a

large amount of user effort and may still not altogether eliminate robustness problems.

To completely eliminate numerical errors, many methods use exact arithmetic (Sugihara and Iri, 1989; Yap and Dubé, 1995; Fortune, 1997; Keyser et al., 1997). Using exact arithmetic, all decisions based on numerical data are guaranteed to be made as if the numerical computations are exact. The main drawback of exact arithmetic is that it can be much slower than standard floating point arithmetic or other fixed-precision numbers that make direct use of the hardware.

Few methods have been proposed to speed up exact arithmetic by using floating point filters (Fortune and Van Wyk, 1993), interval arithmetic (Chun-Yi et al., 1996), and affine arithmetic (de Figueiredo, 1996). Most of these methods have been developed for linear domains. Exact arithmetic in the non-linear domain is much more expensive. The paper by Keyser et al. (Keyser et al., 1997) describes an exact implementation for low degree algebraic surfaces.

Degeneracies

A second source of robustness problem is degenerate data. Many algorithms assume that the input data is in general position and do not handle certain "degenerate cases". For example, a boundary evaluation algorithm may assume that the input primitives are manifold and/or they intersect each other transversally. Failure to satisfy the assumption can lead to program failure.

Handling degeneracies is a very challenging problem and has been studied in both computational geometry and solid modeling. Many approaches have been proposed to handle them. One option is to resort to "special case handling" (Yu, 1992). This involves enumerating all the possible types of degeneracies and add code to explicitly detect and resolve them. While this approach can be useful in many situations, it leads to more complexity in the underlying algorithms and representations, e.g., these algorithms need to work with non-manifold representations.

A different approach to handling degeneracies is to use perturbation methods. The main idea behind perturbation methods is to modify the input data such that it is guaranteed to have no degeneracies. A large number of perturbation techniques have been proposed (Edelsbrunner and Mücke, 1987; Emiris and Canny, 1991; Yap, 1990; Raab, 1999; Ouchi and Keyser, 2004). There are a few issues with using perturbation methods (Seidel, 1994). One of the arguments against perturbation methods is that degenerate cases are often present on purpose. Thus, perturbing the data to make it non-degenerate actually invalidates the input data. A second argument against per-

turbation methods is that applying perturbation changes the problem – therefore the solution to the perturbed problem is not actually a solution of the original problem. In some cases, it is possibly to transform the computed solution to recover the solution to the original problem. However, in general, this transformation can be difficult. Finally, perturbation methods require some form of exact computation in order to be successful. This makes implementation of perturbation methods slower than those that do not use exact computation.

2.1.2 Surface Intersection

There is a large body of literature addressing the surface intersection problem. A survey of various surface intersection techniques can be found in (Pratt, 1986; Patrikalakis, 1993; Hoffmann, 1989b; Krishnan, 1997).

The intersection of two surfaces can be complicated in general, with a number of boundary segments, closed loops, and self-intersections (singularities). A good surface intersection algorithm should, in theory, be able to detect all such features of the intersection curve.

Early approaches to surface intersection were based on recursive subdivision techniques (Lane and Riesenfeld, 1980; Pratt, 1986). The basic idea of these methods is to decompose the problem into similar problems which are much simpler. These methods subdivide the surface into small pieces until each piece satisfies some flatness criterion. However the main problem with the approach lies in constructing the topology of the intersection curve from the individual intersection pieces. This approach is not robust and may fail in the presence of singularities.

An alternative approach for evaluating surface intersections is curve tracing. The main idea behind tracing techniques is to compute a starting point on each loop of the curve and use the local geometry of the curve to evaluate successive points. In this class of methods, identifying a starting point on each loop is a difficult problem. This had led to the development of a large number of techniques for loop detection (Sinha et al., 1985; Sederberg and Meyers, 1988; Kriezis et al., 1990; Hohmeyer, 1991; Zundel and Sederberg, 1993). Most of these methods subdivide each surface until sufficient conditions for the nonexistence of loops are satisfied. Other methods perform loop detection by mapping the intersection curve onto a different domain where the mapped curve is always connected (Krishnan, 1997).

Two problems that must be addressed when tracing a curve are *component jumping*

and *backtracking*. Component jumping occurs when the tracing algorithm takes a step that is too large and proceeds on a different component of the curve. Backtracking occurs when for some reason the tracing algorithm generates points out of order. To prevent either of these occurrences, the algorithm must choose appropriate step sizes to march along the intersection curve (Hohmeyer, 1991; Krishnan, 1997).

2.2 Arrangements

The arrangement of a finite collection of geometric objects in $\mathbb{R}^d$, denoted as $\mathcal{A}(\Gamma)$, is the decomposition of the space into relatively open connected cells of dimensions $0, \ldots, d$ induced by Γ, where each cell is a maximal connected set of points lying in the intersection of a fixed subset of Γ (Agarwal and Sharir, 2000). They serve as the underlying structure for many geometric problems arising in a wide range of applications including robotics, computer graphics, molecular modeling, and computer vision. Halperin (Halperin, 1997) and Agarwal and Sharir (Agarwal and Sharir, 2000) present a comprehensive survey on the subject of arrangements and their applications.

We restrict ourselves to arrangements of (hyper)surfaces in three or higher dimensions. Let the coordinate axes of $\mathbb{R}^d$ be denoted as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$. Let $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ be a collection of (hyper)surface patches in $\mathbb{R}^d$. Usually, the following assumptions are made about the surface patches (Halperin, 1997):

1. Each surface patch is contained in an algebraic surface of constant maximum degree.

2. The boundary of each surface patch is determined by at most some constant number of algebraic surface patches each of constant maximum degree.

3. Every d-tuple of surface patches in Γ meet in at most s points.

4. Each surface patch is monotone in $\mathbf{x}_1, \ldots, \mathbf{x}_{d-1}$, namely every line parallel to the $\mathbf{x}_d$ axis intersects the surface patch in at most one point

5. The surface patches in Γ are in general position.

We will use the term *arrangement of surfaces* to refer to arrangements whose defining objects satisfy the above assumptions. For additional remarks regarding the above assumptions, see (Halperin, 1997).

The combinatorial complexity of the arrangement refers to the total number of cells of various dimensions in the arrangement. Given a collection Γ of n surfaces in $\mathbb{R}^d$, as defined above, the maximum combinatorial complexity of the arrangement $\mathcal{A}(\Gamma)$ is $O(n^d)$ where the constant of proportionality depends on d and on the maximum degree of the surfaces and of their boundaries (Halperin, 1997).

2.2.1 Complexity of a Single Cell

A cell is a maximal connected subset of the intersection of a fixed (possibly empty) subset of surface patches that avoids all other surface patches. The combinatorial complexity of a single cell in the arrangement is the total number of cells of any dimension on the boundary of the cell.

The problem of computing a single cell in the arrangement arises in motion planning (Chapter 5). In this problem, Γ denotes a set of *contact surfaces* that arise due to contact between pairs of geometric features of the robot and the obstacle (see Sec. 2.5 and Chapter 5). The free space $\mathcal{F}$ consists of a collection of cells in the arrangement $\mathcal{A}(\Gamma)$; a single connected component of $\mathcal{F}$ is a single cell in $\mathcal{A}(\Gamma)$. For motion planning, it is sufficient to compute just the connected component containing the initial configuration and check if the goal configuration belongs to the same connected component.

Halperin and Sharir (Halperin and Sharir, 1995a) showed that the combinatorial complexity of a single cell of $\mathcal{A}(\Gamma)$ in three dimensions is $O(n^{2+\epsilon})$, for any $\epsilon > 0$, where the constant of proportionality depends on ϵ and on the maximum degree of the surfaces. They also proposed an algorithm than can compute a single cell in $O(n^{2+\epsilon})$ time. Subsequently, Basu (Basu, 1998) extended the result of Halperin and Sharir to higher dimensions. Basu showed that the combinatorial complexity of a single connected component of $\mathcal{A}(\Gamma)$ in k dimensions is $O(n^{k-1+\epsilon})$. Furthermore, under certain geometric assumptions on the objects, this bound can be improved to $O(n^{k-1})$.

2.2.2 Union Computation

Let $\mathcal{K} = \{\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n\}$ be a set of n connected d-dimensional sets in $\mathbb{R}^d$. We consider the complexity of $\mathcal{K} = \cup_{i=1}^{n} \mathcal{K}_i$ in three dimensions.

An important application of union is to the problems of Minkowski sum computation and translational motion planning. These problems can be reduced to computing a union of a set of convex polyhedra $\mathcal{K}_i, i = 1, \ldots, n$ where each $\mathcal{K}_i$ is defined as a Minkowski sum of two convex polyhedra (see Chapter 4).

Aronov et al. (Aronov et al., 1997) proved that the complexity of the union of n convex polyhedra in $\mathbb{R}^3$ with a total of s faces is $O(n^3 + ns \log n)$. The bound was improved by Aronov and Sharir (Aronov and Sharir, 1997) to $O(ns \log n)$ and $\Omega(ns\alpha(n))$ when the given polyhedra are Minkowski sums of a fixed convex polyhedron with n pairwise-disjoint convex polyhedra.

Very recently, Ezra (Ezra, 2005) has shown that a single cell in the arrangement of convex polyhedra in $\mathbb{R}^3$ has a complexity $O(sn^{1+\epsilon})$ for any $\epsilon > 0$. Thus their result gives an upper bound on the combinatorial complexity of a single component of $\mathcal{F}$ for the case of 3D translational motion planning. Ezra (Ezra, 2005) also presents a deterministic algorithm that constructs a single cell in time $O(sn^{1+\epsilon} \log^2 s)$ for any $\epsilon > 0$.

Agarwal and Sharir (Agarwal and Sharir, 1999) considered the problem of planning the motion of a ball $\mathcal{B}$ moving among a set of pairwise-disjoint polyhedral obstacles with a total of n vertices in $\mathbb{R}^3$. They reduce the free space $\mathcal{F}$ to a union operation and show that the maximum combinatorial complexity of $\mathcal{F}$ is $\mathcal{B}$ is $O(n^{2+\epsilon})$ for any $\epsilon > 0$. They also presented a randomized algorithm to compute the boundary of $\mathcal{F}$ whose expected running time is $O(n^{2+\epsilon})$ for any $\epsilon > 0$.

A useful consequence of the above result is that it gives an upper bound on the combinatorial complexity of the (solid) offset of a polyhedron. The offset operation, an important operation in solid modeling, is identical to the Minkowski sum operation with a ball. We will discuss offsets in more detail in Sec. 2.4.5.

2.3 Isosurface Extraction

Given a continuous scalar field $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and a scalar value s, the *isosurface* with *isovalue* s is the set, $\{\boldsymbol{x} \in \mathbb{R}^d \mid f(\boldsymbol{x} = s\}$, of points with identical scalar value s. Isosurface extraction refers to the process of constructing a (usually piecewise linear) approximation to the isosurface. In this chapter, we focus only on isosurface extraction in $\mathbb{R}^3$. Even this topic has been extensively studied, and we do not attempt to survey the entire literature on this topic. We refer the reader to (Sutton et al., 2000; Carr, 2004) for additional pointers. For work on isosurface extraction in higher dimensions, see (Weigle and Banks, 1998; Bhaniramka et al., 2000; Bhaniramka et al., 2004).

The problem of isosurface extraction originated in two disparate domains – volumetric visualization and implicit modeling. In volumetric visualization, isosurface extraction was used as a tool to visualize volumetric datasets acquired experimentally. Whereas in the field of implicit modeling, it was used to compute an explicit boundary

representation of surfaces represented using mathematical functions. The early work in these two domains was done independently. Ultimately, it was realized that general techniques for isosurface extraction could be applied to either domains.

We start by giving a brief overview of isosurface extraction methods in volumetric visualization and implicit modeling (Sections 2.3.1 and 2.3.2). Next we describe Marching Cubes (Lorensen and Cline, 1987) – the most popular method for isosurface extraction – and many of its variants (Section 2.3.3). Then we discuss various topological considerations during isosurface extraction (Section 2.3.4). We discuss a number of isosurface extraction methods that provide some form of topological guarantees on their output.

2.3.1 Volumetric Visualization

In the field of volumetric visualization, isosurface extraction techniques were developed due to the need to visualize volumetric datasets acquired experimentally, e.g., medical datasets obtained using magentic resonance imaging (MRI), Computed Tomography (CT) (Lorensen and Cline, 1987; Payne and Toga, 1990a; Lorensen, 1995). These data sets consist of scalar values on a voxel grid where the type of scalar values depend on the application; typically they represent some kind of intensity/density corresponding to the part being scanned. Often the data points are organized on a structured grid, e.g., a $L \times M \times N$ grid. Then the numbers L, M, and N define the *resolution* of the data set. Isosurface extraction is used to visualize the boundary of some important feature – typically, an anatomical organ. The boundary corresponds to a region of constant value in the volumetric data.

Volumetric datasets are also generated during scientific simulations in computational fluid dynamics (CFD) (Favre, 1997; Heermann, 1998) and molecular dynamics (Monks et al., 1996; Lanzagorta et al., 1998). In these applications, the data sets consist of scalar values on a voxel grid where the scalar values represent the values of various simulation parameters, e.g., temperature, pressure, etc. The objective is to expose contours of constant value for understanding the structure of the scalar field. These contours isolate surfaces of interest, focusing attention on important features in the data such as material boundaries and shock waves.

Early volume visualization methods adopted two approaches to represent regions of constant value within the volumetric data. The first approach, the *cuberille method*, considered the subdivision of space created by the sample points and represented con-

stant value regions as a set of cubes that included the values of interest (Herman and Liu, 1979; Artzy et al., 1981; Herman and Udupa, 1983).

The second approach used isosurface extraction. The desired surface was approximated by choosing an isovalue and extracting a surface which partitioned the data points (also called sample points) into two sets – those above the isovalue, and those below the isovalue. The surface was extracted by interpolating the scalar values between data points. Early isosurface methods first generated two-dimensional contour lines for parallel planes running through the data, and then connected the contour lines into three dimensional isosurfaces (Fuchs et al., 1977; Christiansen and Sederberg, 1978; Wright and Humbrecht, 1979; Cook et al., 1983). Subsequent methods created the isosurface by examining the three dimensions at once (Koide et al., 1986; Lorensen and Cline, 1987; Payne and Toga, 1990a; Wilhelms and Gelder, 1990a).

2.3.2 Implicit Modeling

Implicit modeling uses a mathematical function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to represent an object. The function f is a scalar field; in the context of implicit modeling, it is also known as an *implicit function*. The implicit surface is the set of points $\boldsymbol{p}$ where $f(\boldsymbol{p}) = 0$. Thus, the implicit surface is an isosurface of f with zero isovalue.

Implicit representations became popular because of their simplicity and versatility in performing a wide variety of geometric operations. For example, implicit representations map Boolean operations on a set of primitives into minimum/maximum operations on the scalar fields of the primitives (Ricci, 1973). Many other operations such as offsetting, blending, warping, and sweep can also be expressed elegantly using implicit representations (Blinn, 1982; Wyvill et al., 1986; Wyvill and van Overveld, 1996; Pasko et al., 1995; Bloomenthal, 1997). Because of these advantages, implicit modeling techniques have been used by a large number of applications including geometric modeling (Breen et al., 2000; Kobbelt et al., 2001; Perry and Frisken, 2001; Ju et al., 2002; Ohtake et al., 2003), volume rendering (Wang and Kaufman, 1994), surface reconstruction (Hoppe et al., 1992; Curless and Levoy, 1996; Carr et al., 2001), remeshing (Kobbelt et al., 2001; Wood et al., 2002), swept volume computation (Schroeder et al., 1994), animation (Wyvill et al., 1986), and sculpting digital characters (Perry and Frisken, 2001).

Despite the above advantages of the implicit representation, many applications such as graphical rendering, collision detection, dynamic simulation, and model verification

use an explicit representation such as a polygonal mesh representation. Isosurface extraction techniques are used to convert implicit surfaces to an "explicit form", i.e., a polygonal mesh approximating the implicit surface.

Wyvill et al. (Wyvill et al., 1986) developed one of the early isosurface extraction methods for polygonizing implicit surfaces. They computed samples in a three-dimensional rectangular lattice and constructed a polygonal surface in each cell in this lattice individually. Bloomenthal (Bloomenthal, 1988) developed another method that adaptively sampled the implicit function by surrounding the implicit surface by an octree. This method generated the octree by adaptively subdividing those cells of the octree that contained highly curved regions. Then a piecewise polygonal representation was extracted from the resulting octree. Subsequently, a large number of methods were developed for polygonizing implicit surfaces (Hall and Warren, 1990; Velho, 1990; Ning and Bloomenthal, 1993; Bottino et al., 1996; Stander and Hart, 1997; Kobbelt et al., 2001; Perry and Frisken, 2001; Ohtake et al., 2001; Ju et al., 2002; Varadhan et al., 2003b; Zhang et al., 2004; Boissonnat et al., 2004; Nielson, 2004; Schaefer and Warren, 2004).

We note one important difference between isosurface extraction methods for volumetric visualization and implicit modeling. The volume data set that is input to the volumetric visualization methods is only a sampled version of a continuous scalar field. Typically, these methods do not have knowledge of the continuous scalar field from which the data set has been experimentally acquired. As a result, they have to use some form of interpolation to obtain the scalar value at a point other than the data points. On the other hand, in the case of implicit modeling methods, the implicit function f is usually available; consequently, they have the ability to resample the implicit function.

2.3.3 Marching Cubes

The Marching Cubes algorithm (MC), proposed by Lorensen and Cline (Lorensen and Cline, 1987), is a standard method to extract an isosurface from a volumetric dataset with scalar values. It performs reconstruction by extracting surfaces separately in every cell in a volumetric cubic grid. The algorithm iterates through all cells in the grid, hence the term marching cubes. In each cell, each vertex is classified as "positive" or "negative" depending on whether the scalar value of the vertex is greater or less than the isovalue. For each edge of the cube whose vertices have opposite signs, a point

is generated by linear interpolation along the edge. These edge points are then used to construct one or more polygonal surface separating the vertices with opposite signs. Since each of the 8 vertices of the cubic cell can be either positive or negative, there are $2^8 = 256$ possible sign configurations. Lorenson & Cline used symmetries between different sign configurations to reduce them to 15 basic cases (Fig. 3.2). They stored each of these cases in a look-up table, and used it to find the polygonal approximation of the isosurface in a given cell.

The original Marching Cubes algorithm examined all cells in the data set even though typically the isosurface intersects only a small subset of the cells. If N is the number of cells in the grid, MC takes $O(N)$ time to generate the isosurface. Itoh and Koyamada (Itoh and Koyamada, 1995) estimated that the number of cells intersected by the isosurface is typically $N^{2/3}$ since the isosurface is two-dimensional and the volumetric data is three-dimensional. Wilhelms and Gelder (Wilhelms and Gelder, 1990a) estimated that MC spent between 30% and 70% of the total time examining empty cells that do not intersect the isosurface. A tremendous amount of research has focused on reducing the number of cells visited while constructing an isosurface (Wilhelms and Gelder, 1990a; Bajaj et al., 1996; Cignoni et al., 1996; Livnat et al., 1996). These methods utilize auxiliary data structures to examine only those cells that contain a portion of the isosurface. While the search structures introduced by many of these methods increase the storage requirements, the acceleration gained by the isosurfacing technique offsets this overhead.

Another drawback of the original MC algorithm was that it generated an excessively large number of triangles to represent the isosurface. To overcome this drawback, many methods were developed for performing isosurface extraction adaptively using hierarchies such as octrees or k-D trees (Bloomenthal, 1988; Hall and Warren, 1990; Velho, 1990; Shekhar et al., 1996; Westermann et al., 1999; Frisken et al., 2000; Gerstner and Pajarola, 2000; Ju et al., 2002). These methods use properties such as local curvature to perform an adaptive polygonization of the surface, thus producing an isosurface with fewer triangles.

A large number of variants of MC have also been developed that suggest alternative ways of reconstructing the isosurface within the cell (Montani et al., 1994; Kobbelt et al., 2001; Perry and Frisken, 2001; Ju et al., 2002; Varadhan et al., 2003b; Zhang et al., 2004; Nielson, 2004; Schaefer and Warren, 2004).

Implicit surfaces defined in terms of Boolean operations usually have sharp edges or corners. When MC is used for polygonizing such implicit surfaces, the output usually

has aliasing artifacts in the vicinity of the sharp features. Recently, a few extensions have been proposed that can reconstruct sharp features and reduce aliasing artifacts in the reconstructed model (Kobbelt et al., 2001; Ohtake et al., 2001; Ju et al., 2002; Varadhan et al., 2003b).

2.3.4 Topological Considerations in Isosurface Extraction

Topological Ambiguity

In the original Marching Cubes algorithm, some of the base cases were ambiguous. Given a face of a cube with two diagonally opposite corners above the surface, and the other two below the surface, some of the basic cases assumed that the higher corners fell inside the same connected component of the surface, while others assumed that they did not. Since each face of a cube was shared with an adjacent cube, it was possible to generate surfaces with holes by accident. This was noted by Durst (M.J.Durst, 1988), and many solutions were proposed (Wilhelms and Gelder, 1990b; Nielson and Hamann, 1991; Natarajan, 1994; Cignoni et al., 2000; Lopes and Brodlie, 2003).

Nielson & Hamann's solution (Nielson and Hamann, 1991), the *asymptotic decider*, was based on assuming that the scalar field can be modeled as a bilinear interpolation on each face of the cube. The known topology of contours of bilinear functions was invoked to ensure that the assumption was consistent in both cells. The value of the scalar field at the saddle point of the bilinear function was computed and used to distinguish between the two possible solutions. Since this test gave the same result in both cubes, consistent treatment was assured, and the holes disappeared.

Natarajan (Natarajan, 1994) extended the asymptotic decider by replacing bilinear interpolation by trilinear interpolation. This enabled the algorithm to test for body saddle points inside the cube. Cignoni et al. (Cignoni et al., 2000) and Lopes & Brodlie (Lopes and Brodlie, 2003) presented further extensions of this idea.

The above algorithms deal with the problem of extracting a surface from a fixed volumetric data set. The continuous scalar field is not available for resampling. Since the scalar values are available only on the vertices of the cell, they model the scalar field in the interior of the cell. For example, the algorithms by (Natarajan, 1994; Cignoni et al., 2000; Lopes and Brodlie, 2003) model the scalar field as a trilinear function that interpolates the scalar values at the vertices of the cell. Then they extract a surface that is topologically equivalent to the isosurface of the trilinear function. However, due to inadequate resolution of the data set, the trilinear function may not be an accurate

model of the continuous scalar field. Hence the output of these algorithms need not be topologically equivalent to the isosurface of the continuous scalar field.

Topology Control and Simplification

Many volumetric approaches have used topological properties for generating an isosurface without additional handles or cavities from scanned data sets (Guskov and Wood, 2001; Wood et al., 2002; Bischoff and Kobbelt, 2002). Often the input data contains noise due to the scanning process. During isosurface extraction, the inaccuracies in the input data result in a surface whose genus is much higher than the actual surface. In many applications, the topological type of the object under consideration is known beforehand, e.g., the cortex of a human brain is always homeomorphic to a sphere (Bischoff and Kobbelt, 2002). These methods exploit such a priori knowledge to eliminate unwanted handles that arise from the noise.

Topology Preserving Implicit Surface Polygonization

Few methods based on Morse theory (Bottino et al., 1996; Stander and Hart, 1997; Boissonnat et al., 2004) can guarantee a topology preserving polygonization of implicit surfaces. These methods assume that the implicit surface is smooth and require computation of all the critical points of the implicit surface.

The "shrinkwrap" algorithm by Bottino et al. (Bottino et al., 1996) and the algorithm by Stander and Hart (Stander and Hart, 1997) follow the same general approach. Both methods first find an isovalue such that the isosurface at that isovalue can be easily polygonized. The initial polygonization is then progressively transformed into the desired one, by computing intermediate level sets. Morse theory is used to perform topological changes to the polygonization when critical points are encountered. The shrinkwrap algorithm begins with a polygonization of a large sphere that shrinks inwards until it adheres to the final implicit surface. In contrast, the algorithm by Stander and Hart starts with an "empty" polygonization that expands outwards towards the final implicit surface.

The algorithm by Boissonnat et al. (Boissonnat et al., 2004) uses a different approach. Their algorithm surrounds the implicit surface with a collection of tetrahedral cells. They show that if these cells satisfy a set of conditions based on critical points, then a topology preserving polygonization can be computed. They also describe an algorithm for building such a mesh.

Snyder (Snyder, 1992) presented an adaptive subdivision method for computing a isotopic approximation of an implicit curve. This method checks whether the implicit curve is locally parametrizable with respect to one of the axes of the cell. The condition for parametrizability is verified by performing interval analysis on the gradient of the implicit function. If a cell fails the condition, it is subdivided and the algorithm is recursively applied to the subdivided cells. Recently, Plantinga and Vegter (Plantinga and Vegter, 2004) have presented a similar method for computing isotopic approximation of both implicit curves and surfaces.

All the above methods assumes that the implicit surface is smooth. It is not clear how to apply them to perform Boolean combinations where the final surface is not smooth and whose topology can be very different from those of the input primitives.

2.4 Minkowski Sum and Offset Computation

The Minkowski sum operation was introduced by Hermann Minkowski in his classic work in 1903 on integral geometry (Minkowski, 1903). Intuitively, the Minkowski sum of two objects can be considered as expanding one object by sweeping it over the other. Formally, the Minkowski sum of two objects $\mathcal{P}$ and $\mathcal{Q}$ is defined as the set-sum,

$$\mathcal{P} \oplus \mathcal{Q} = \{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in \mathcal{P},\ \boldsymbol{q} \in \mathcal{Q}\}$$

Minkowski sums have found applications in a diverse set of domains. Hadwiger (Hadwiger, 1957) discussed their fundamental properties in the context of classical geometry. Matheron (Matheron, 1975) and Serra (Serra, 1982) studied them in mathematical morphology. Lozano-Perez and Wesley (Lozano-Pérez and Wesley, 1979) used them to define configuration space obstacles for path planning. Minkowski sums have also been used for penetration depth computation (Cameron, 1997; Agarwal et al., 2000), packing and layout (Boissonnat et al., 1997; Daniels and Milenkovic, 1997), and for implementing solid modeling operations such as constant radius offsetting, rounding, filleting, translational sweep (Rossignac and Requicha, 1986; Evans et al., 1987).

In this section, we briefly review the prior work on 2D and 3D Minkowski sum computation. We make a distinction based on whether the input primitives are polygonal/polyhedral or curved objects.

2.4.1 Minkowski Sum of Polygons

The Minkowski sum $\mathcal{P} \oplus \mathcal{Q}$ of two polygons, $\mathcal{P}$ and $\mathcal{Q}$, with m and n features (vertex or edge) can have a worst case combinatorial complexity $O(m^2n^2)$. This upper bound is tight as there exist such examples (Agarwal et al., 2002).

When both $\mathcal{P}$ and $\mathcal{Q}$ are convex, the size of $\mathcal{P} \oplus \mathcal{Q}$ is only $O(m+n)$. For this case Schwartz (Schwartz, 1981) and Lozano-Perez (Lozano-Pérez, 1983) presented a linear time algorithm to compute $\mathcal{P} \oplus \mathcal{Q}$. The main idea behind this algorithm is as follows. In a first step, the features of $\mathcal{P}$ and $\mathcal{Q}$ are sorted based on the slope of their edges. The second step performs a counter-clockwise (or clockwise) sweep to combine features of $\mathcal{P}$ and $\mathcal{Q}$. This step produces a succession of features belonging to $\mathcal{P} \oplus \mathcal{Q}$. Provided the sorted lists of features of $\mathcal{P}$ and $\mathcal{Q}$ are available, this algorithm takes $O(m+n)$ time. The algorithm can also be applied to non-convex polygons by decomposing them into convex polygons. This approach, which will be described in Sec. 2.4.3, has been taken by many subsequent methods (Kaul et al., 1991; Flato and Halperin, 2000).

An operation closely related to the Minkowski sum is the *convolution* operation (Guibas et al., 1983). Like Minkowski sum, convolution is a general operation that can be defined in any dimension. Convolution between two geometric objects is defined on their respective "tracings", which is obtained by augmenting the each object with a "direction map". For example, to obtain a tracing of a planar curve, each point $\mathbf{p}$ on the curve is associated with a unit vector $\vec{\mathbf{p}}$ tangent to the curve. This definition is further extended to define a tracing of a polygon as follows. If $\mathbf{p}$ lies on an edge of $\mathcal{P}$, then $\vec{\mathbf{p}}$ is parallel to the edge. On the other hand, if $\mathbf{p}$ is a vertex of $\mathcal{P}$, then $\vec{\mathbf{p}}$ can vary along an arc of the circle of directions (Guibas et al., 1983). A pair $(\mathbf{p}, \vec{\mathbf{p}})$ is called a *state* and the tracing $\widehat{P}$ is the set of all the states of the points $\mathbf{p} \in \mathcal{P}$. The convolution of two tracings $\widehat{P}$ and $\widehat{Q}$ is defined as follows:

$$\widehat{P} * \widehat{Q} = \{(\mathbf{p}+\mathbf{q}, \vec{\mathbf{p}} \mid (\mathbf{p}, \vec{\mathbf{p}}) \in \widehat{P}, (\mathbf{q}, \vec{\mathbf{q}}) \in \widehat{Q}, \vec{\mathbf{p}} = \vec{\mathbf{q}}\}$$

We note a few properties of convolution that are important not only in their own right but also in the context of Minkowski sum computation. First, the worst case combinatorial complexity of $\widehat{P} * \widehat{Q}$ is only $O(mn)$, which is much better than the $O(m^2n^2)$ worst case complexity of $\mathcal{P} \oplus \mathcal{Q}$. Second, $\widehat{P} * \widehat{Q}$ can be computed in $O(m+n+k)$ time where k is the size of $\widehat{P} * \widehat{Q}$. Third, it can be shown that $\mathcal{P} \oplus \mathcal{Q}$ corresponds to a subset of $\mathcal{P} * \mathcal{Q}$. Therefore, the convolution provides a good way to compute the Minkowski sum. It provides a condition for selecting pairs of features of $\mathcal{P}$ and $\mathcal{Q}$ whose

interaction may contribute to the boundary of the Minkowski sum. This property has been utilized by many subsequent algorithms. Finally, in the case where both $\mathcal{P}$ and $\mathcal{Q}$ are convex, $\widehat{P} * \widehat{Q}$ is a tracing of a convex polygon that is identical to $\mathcal{P} \oplus \mathcal{Q}$. In this case, $\widehat{P} * \widehat{Q}$ can be computed in linear time.

2.4.2 Minkowski Sum of Planar Curves

Many authors have studied the problem of computing the Minkowski sum of planar curves. Bajaj and Kim (Bajaj and Kim, 1987) and Kaul and Farouki (Kaul and Farouki, 1995) studied the convolution of algebraic curves. Lee et al. (Lee et al., 1998) treated the case where the input curves were rational.

All of these methods follow a common approach. They first compute the convolution of the two input curves. If the input curves are algebraic curves, then their convolution is also an algebraic curve. In general, the convolution curve is self-intersecting and is a superset of the actual Minkowski sum boundary. To obtain the boundary of the Minkowski sum, these methods trim away the redundant parts of the convolution curve. There are two issues with this approach. First, the degree of the convolution curve is usually quite high which introduces many numerical problems (Lee et al., 1998). Second, the trimming of the convolution curve is difficult in practice. It is related to the problem of computing arrangement of planar curves and there are no practical implementations for arrangements of high degree planar curves. Typically, these methods compute a polygonized approximation to the convolution curve. This reduces the trimming problem to computing arrangement of a collection of line segments.

The convolution of two rational curves need not be rational. Lee and Kim (Lee et al., 1998) presented a method for computing a rational approximation to the convolution curve which was then used to compute an approximation to the Minkowski sum boundary.

2.4.3 Minkowski Sum of Polyhedral Primitives

The combinatorial complexity of the Minkowski sum of two convex polyhedra is $O(mn)$ where m and n denote the number of features (vertex, edge, face) in the two polyhedra. On the other hand, the Minkowski sum of two non-convex polyhedra can have a complexity as high as $O(m^3n^3)$ (Dobkin et al., 1993).

We first discuss specialized algorithms for computing the Minkowski sum of convex polyhedra, and then discuss general methods for non-convex polyhedra.

Convex Polyhedra

There are a number of efficient algorithms for computing the Minkowski sum of convex polyhedra.

Convex hull method: This method is based on the following property:

$$P \oplus Q = \mathbf{CH}(\{\mathbf{v}_i + \mathbf{v}_j | \mathbf{v}_i \in V_P, \mathbf{v}_j \in V_Q\}) \tag{2.1}$$

Here, **CH** denotes the convex hull operator, and V_P, V_Q represent the sets of vertices, respectively in polyhedra P and Q. Based on this fact, we compute the Minkowski sum as follows:

1. Compute the vector sum between all possible pairs of vertices from each polytope.
2. Compute their convex hull.

Step 1 produces mn points. Computing their convex hull can take $O(mn \log(mn)$ time (de Berg et al., 2000).

Convolution method: A more efficient method is presented by Guibas and Seidel (Guibas and Seidel, 1987). This method by is based on the convolution operation defined on *polyhedral tracings*. A polyhedral tracing is the 3D counterpart of a polygonal tracing. It extends a polyhedron by augmenting the position with a direction map defined at each point of the polyhedron. The direction map is defined in terms of the Gauss map.

The Gauss Map $\mathbb{G}$ of a compact convex polyhedron $\mathcal{P}$ in Euclidean three-dimensional space $\mathbb{R}^3$ is a set-valued function from $\mathcal{P}$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p}$ the set of outward unit normals to support planes to $\mathcal{P}$ at $\mathbf{p}$. The entire facet f of $\mathcal{P}$ is mapped under $\mathbb{G}$ to a single point – the outward unit normal to f. An edge e of $\mathcal{P}$ is mapped to an arc of a great circle $\mathbb{G}(e)$ on $\mathbb{S}^2$, whose length is the exterior dihedral angle at e. A vertex v of $\mathcal{P}$ is mapped by $\mathbb{G}$ to a spherical polygon $\mathbb{G}(v)$, whose sides are the images under $\mathbb{G}$ of edges incident to v.

The polyhedral tracing is defined as follows:

$$\widehat{P} = \{(\mathbf{p}, \vec{\mathbf{p}}) \mid \mathbf{p} \in \mathcal{P}, \vec{\mathbf{p}} \in \mathbb{G}(\mathbf{p})\}$$

Then the convolution operation can be defined on two polyhedral tracings as in the 2D case. $\widehat{P} * \widehat{Q}$ is another polyhedral tracing. Furthermore, if both $\mathcal{P}$ and $\mathcal{Q}$ are convex, then $\widehat{P} * \widehat{Q}$ is a tracing of a convex polyhedron that is identical to $\mathcal{P} \oplus \mathcal{Q}$. This fact

is used to compute the Minkowski sum using convolution computation (Guibas and Seidel, 1987).

A key property of a convex polyhedron is that the Gauss map of the associated tracing corresponds to a convex decomposition of the unit sphere. This property is utilized to compute the convolution (and hence the Minkowski sum) in the following manner:

1. Compute the Gauss maps of $\mathcal{P}$ and $\mathcal{Q}$
2. Compute an overlay of $\mathbb{G}_{\mathcal{P}}$ and $\mathbb{G}_{\mathcal{Q}}$.
3. The overlay can be further processed to obtain $\mathcal{P} * \mathcal{Q}$, which is identical to $\mathcal{P} \oplus \mathcal{Q}$ (Guibas and Seidel, 1987).

This method is very efficient and has an output-sensitive time complexity. It can compute the Minkowski sum of two convex polyhedra in $O(m + n + k)$ time, where k is the number of faces in the output. The output size k can vary between $O(m + n)$ and $O(mn)$.

A number of variants of the above method have been proposed. Ghosh (Ghosh, 1993) presented a method based on computing a *slope diagram* that is obtained by projecting the Gauss map of a polyhedron onto a plane. The Minkowski sum of two polyhedra is obtained by merging their slope diagrams. Bekker and Roerdink (Bekker and Roerdink, 2001) and Wu et al. (Wu et al., 2003) provided a number of improvements to this method. Recently, Fogel and Halperin (Fogel and Halperin, 2005) presented a method based on a *Cubical Gauss Map* representation that maps the Gauss map onto a cube.

Incremental Surface Expansion: This method was proposed as a part of a polyhedral morphing system based on Minkowski sums (Kaul and Rossignac, 1991). This method classifies the faces on the boundary of the Minkowski sum into three types. The first type is a face/vertex (FV) type, where a face of one polyhedron is combined with a vertex of the second polyhedron to obtain the face on the boundary of the Minkowski sum. Similarly, a vertex/face (VF) and edge/edge (EE) types are defined. The algorithm starts with any candidate face on the boundary of the Minkowski sum, and incrementally expands the surface by finding the next candidate face. The incremental expansion uses every possible FV, VF, EE combination that the current candidate face can be extended to. The worst case asymptotic time complexity of this algorithm is

$O(mn)$, but it works well in practice, with just a few degeneracies such as co-planar faces that need special handling.

Non-convex polyhedra

Prior methods for Minkowski sum of non-convex polyhedral models can be classified into two categories: union based methods and arrangement based methods.

Union Based Methods: The first class of methods reduce the problem to a union computation (Lozano-Pérez, 1983; Evans et al., 1992). These algorithms decompose the polyhedral models into simpler objects whose Minkowski sum can be easily computed.

Lozano-Perez (Lozano-Pérez, 1983) proposed a method based on *convex decomposition*. This method uses the following property of Minkowski sum: If $P = P_1 \cup P_2$, then $P \oplus Q = (P_1 \oplus Q) \cup (P_2 \oplus Q)$. The method combines this property with convex decomposition for general polyhedral models:

1. Compute a convex decomposition for each polyhedron
2. Compute the pairwise Minkowski sums between all possible pairs of convex pieces in each polyhedron.
3. Compute the union of pairwise Minkowski sums.

The pairwise Minkowski sums can be computed using one of the methods presented in the previous subsection. After the second step, there are $O(kl)$ pairwise Minkowski sums where k and l are the number of convex pieces of the two polyhedra. The pairwise Minkowski sums are convex, and their union can have a combinatorial complexity $O(p^3 + sp\log(p))$ where $p = kl$ and s is the total number of faces in the pairwise Minkowski sums (Aronov et al., 1997).

Evans et al. (Evans et al., 1992) presented a different method based on computing the union of *translational sweeps*. Their method follows three steps:

1. Compute the Minkowski sum of each face of $\mathcal{P}$ and every non-parallel edge of $\mathcal{Q}$. The Minkowski sum of a single face and a non-parallel edge yields a triangular prism. This step may produce $O(mn)$ prisms.
2. Generate copies of $\mathcal{Q}$ translated by every vertex of $\mathcal{P}$.
3. Generate copies of $\mathcal{P}$ translated by every vertex of $\mathcal{Q}$.

Steps 2 and 3 produce m and n copies respectively. The collection of all the polyhedra obtained in the above steps is unioned to obtain $\mathcal{P} \oplus \mathcal{Q}$.

Both the above methods (Lozano-Pérez, 1983; Evans et al., 1992) typically need to perform a union of a large number (e.g. thousands) of primitives.

Arrangement Based Methods: The second class of methods reduce the problem to arrangement computation (Kaul and Rossignac, 1991; Ghosh, 1993; Basch et al., 1996). These algorithms are similar to the ones presented for convex polyhedra. However, for non-convex polyhedra, the output of these methods is a superset of the boundary of the Minkowski sum. To obtain the actual boundary of the Minkowski sum, the arrangement of the superset has to be computed.

Kaul and Rossignac (Kaul and Rossignac, 1991) adapted their incremental surface expansion method for the case of non-convex polyhedra. They enumerate different possible FV, VF, EE combinations that can potentially contribute to the Minkowski sum boundary. They generate a surface for each combination, thus yielding a superset of the boundary of the Minkowski sum. Basch et al. (Basch et al., 1996) extended the convolution method (Guibas and Seidel, 1987) to compute convolution of tracings of non-convex polyhedra, which also produces a superset. Ghosh's slope diagram based method (Ghosh, 1993) can be applied to non-convex polyhedra. However, in this case merging the slope diagrams can be nontrivial.

2.4.4 Minkowski Sum of Curved Surfaces

Bajaj and Kim (Bajaj and Kim, 1990) presented algebraic methods to compute the convolution of a pair of curved convex objects bounded by patches of algebraic surfaces. They show that the convolution can be represented exactly as an implicit algebraic surface. However, the resulting algebraic degrees are usually very high (Lee et al., 1998).

In general, computing the 3D Minkowski sum of curved surfaces is considered difficult. One source of complication is the difficulty in deriving an explicit parametrization of the convolution. This has led to the development of algorithms for a restricted class of surfaces. Kim and Sugihara (Kim and Sugihara, 2001) and Seong et al (Seong et al., 2002) considered the case where the input surfaces are surfaces of revolution or linear extrusion generated by slope-monotone closed curves. Muhlthaler and Pottmann (Mhlthaler and Pottmann, 2003) considered the case where the input surfaces are ruled surfaces. These algorithms take advantage of the fact that for these classes of surfaces,

an explicit parametrization of the convolution can be derived. However, computing the actual boundary of the Minkowski sum requires a trimming operation, which is difficult to implement robustly.

2.4.5 Offset Computation

Two definitions of offsets are found in the literature – *normal offset* (n-offset) of a surface (Maekawa, 1999) and *solid offset* (s-offset) of a solid (Rossignac and Requicha, 1986). The normal offset of a curve/surface is defined as the locus of the points which are at a constant distance r along the normal from the generator curve/surface. The definition of n-offset require a well-defined normal at each point on the surface. Therefore, they are not well-defined if a surface is not smooth. Furthermore, even when a curve/surface is smooth, n-offsetting may lead to cusps and self-intersections.

Rossignac and Requicha (Rossignac and Requicha, 1986) extended the above definition by defining a solid offset (s-offset). The s-offset of a solid S is obtained by adding to S all the points exterior to S that lie within a distance r of the boundary of S. Mathematically, it is defined as:

$$\text{s-offset}(S) = \{\boldsymbol{x} \in \mathbb{R}^d \mid \exists \boldsymbol{s} \in S,\ \|\boldsymbol{x} - \boldsymbol{s}\| \leq r\}$$

A solid offset produces a solid that is an expanded version of the original solid. It can be considered as a special case of Minkowski sum where one of the objects is a ball (disk in 2D) (Rossignac and Requicha, 1986). The s-offset is well defined for a solid even if its boundary is not smooth. In particular, it is well-defined for a polyhedron. Rossignac and Requicha showed that the the boundary of the s-offset of a solid S is a subset of a collection of surfaces generated by offsetting the faces, edges, and vertices of ∂S They extended the definition of n-offset to generate offsets of edges and vertices of ∂S.

We now give a brief overview of some of the prior work on offsets of curves and surfaces. To keep the presentation simple, we do not distinguish between whether these methods compute an n-offset or an s-offset; we use the term offset to refer to both. We refer the reader to (Pham, 1992; Maekawa, 1999) for a comprehensive survey of the literature.

There has been a lot of work on computing offsets of planar algebraic curves (Lee et al., 1997; Maekawa, 1999). The offset of a planar algebraic curve is algebraic, but can have a very high degree. For example, the offset of a cubic Bezier curve can have

an algebraic degree of 10 (Lee et al., 1997). These fundamental limitations have led offset research to develop various offset approximation techniques (Lee et al., 1997). The offset of a rational curve/surface need not be rational, except in special cases (line, circle, plane, sphere, cylinder, etc). Therefore, some researchers have focused on a special class of curves, e.g., *Pythagorean hodograph* (PH) curves, whose offsets are rational curves (Farouki and Sakkalis, 1990). The notion of PH curves has also been extended to 3D to define PH surfaces which have rational offsets (Pottmann, 1995).

One of the main issues with offset computation is that the offset curve/surface may self-intersect. Two types of self intersections can occur. First, a self intersection may occur *locally* when the absolute value of the offset distance exceeds the minimum radius of curvature in the concave regions. Second, a self-intersection may also occur *globally* when the distance between two distinct points on the curve/surface reaches a local minimum (Maekawa, 1999). It is an essential task of an offset computation algorithm to detect the self-intersections and generate the *trimmed* offset curve/surface.

There is considerable amount of prior work on offsets of surfaces in 3D (Farouki, 1985; Rossignac and Requicha, 1986; Forsyth, 1995; Elber and Cohen, 1997). Rossignac and Requicha (Rossignac and Requicha, 1986) studied offsets based on regularized sets, while Elber and Cohen (Elber and Cohen, 1997) studied them in the context of filleting and rounding based on Bezier and NURBS surface representation. All these methods compute offset surfaces of models first, then trim or extend these offset surfaces to reconstruct a closed 3D model. Due to the complexity of trimming and extending operations, they are difficult to implement robustly.

The difficulty of trimming has prompted many methods to adopt alternative representations such as implicit and distance field based representations (Breen and Mauch, 1999; Perry and Frisken, 2001; Williams and Rossignac, 2004), ray representations (Hartquist et al., 1999), to efficiently compute approximations to the offset. However, most of these algorithms are developed for displaying purpose in computer graphics domain. No accuracy analysis of the offset approximation is presented. Recently, Chen et al. (Chen et al., 2005) presented an approximate algorithm based on point based representations. Their algorithm can produce an approximation with a bounded geometric error.

2.5 Motion Planning and Free Space Computation

Motion planning is an extensively studied problem. It is beyond the scope of this dissertation to provide a complete survey of the prior work. We refer the reader to the following books and survey papers. The book by Latombe (Latombe, 1991) is a standard reference for the work prior to 1991. A survey paper by Sharir (Sharir, 1997) covers algorithmic motion planning, emphasizing the theoretical analysis of the problem. A recent book by Choset et al. (Choset et al., 2004) covers the more recent advances in the field.

We restrict ourselves to the basic version of the motion planning problem which assumes that the robot $\mathcal{R}$ – a rigid or an articulated object – is the only moving object in a static workspace cluttered with rigid, stationary obstacles $\mathcal{O}$. $\mathcal{R}$ may translate, rotate or have different types of joints imposing sliding or rotating constraints. It is assumed that the geometry of both $\mathcal{R}$ and $\mathcal{O}$ is known and that there are no kinematic constraints to limit the motion of $\mathcal{R}$. This version of the problem ignores issues such as uncertainty, nonholonomic constraints, control issues, etc. The goal is to find a *collision-free path* – a path along which $\mathcal{R}$ can move from an initial configuration $\mathbf{q}_{init}$ to a goal configuration $\mathbf{q}_{goal}$ without colliding with $\mathcal{O}$. A collision-free path can be defined using the configuration space formulation. Let $\mathcal{C}$ denote a k-dimensional configuration space. The free space $\mathcal{F}$ is defined as the set of all the configurations of $\mathcal{R}$ for which $\mathcal{R}$ does not intersect $\mathcal{O}$ (see Sec. 1.3 for a precise definition). A collision-free path is a continuous map $\tau : [0, 1] \rightarrow \mathcal{F}$ with $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$. Sometimes the above problem definition is modified slightly to allow a *semi-free* path that is allowed to touch $\partial\mathcal{F}$, the boundary of $\mathcal{F}$. A motion planning algorithm is *complete* if is guaranteed to find a collision-free path if one exists and return a failure otherwise.

$\mathcal{F}$ can be expressed in terms of an arrangement of a set of *contact surfaces* (Sharir, 1997). A contact surface of a geometric feature (vertex, edge, face) of $\mathcal{R}$ and a similar feature (vertex, edge, face) of $\mathcal{O}$ is defined as the set of points in $\mathcal{C}$ such that represent configurations of $\mathcal{R}$ at which contact is made between these specific features. Let Γ denote the set of all the contact surfaces defined by all pairs of features of $\mathcal{R}$ and $\mathcal{O}$ that can be involved in a contact with each other. Then $\mathcal{F}$ consists of a collection of cells in the arrangement $\mathcal{A}(\Gamma)$. A single connected component of $\mathcal{F}$ is a single cell in $\mathcal{A}(\Gamma)$.

We start by presenting some the theoretical results on the computational complexity bounds for path planning (Sec. 2.5.1). We discuss both general solutions as well as

algorithms for specific instances of the problem. All these algorithms are based on computing some sort of free space representation. We cover only a small subset of all the results on the subject. We refer the reader to Latombe (Latombe, 1991) and Sharir (Sharir, 1997) for a more comprehensive survey.

Most of the prior methods for motion planning can be classified into a few general approaches. We discuss these approaches in Sec. 2.5.2.

2.5.1 Computational Complexity

General Solutions

The first upper bound on the complexity of the motion planning problem problem was established by Schwartz and Sharir (Schwartz and Sharir, 1983a). Their algorithm used Collins cylindrical algebraic decomposition (Collins, 1975) to perform an exact cell decomposition of the free space. This algorithm has a time complexity doubly exponential in the configuration dimension. The algorithm takes randomized expected time $O((nd)^{3^k})$ where the contact surfaces are defined by a total of n polynomials of maximum degree d and k is the configuration space dimension.

This doubly exponential time complexity bound was improved by Canny (Canny, 1987) to a singly exponential time. Canny proposed an algorithm that took expected randomized $O(n^k(\log n)d^{O(k^2)})$ time. Rather than compute an exact cell decomposition of the free space, Canny's algorithm computed a one-dimensional network of curves in the free space, called the *roadmap* (Sec. 2.5.2). The original Canny's algorithm made assumptions about the contact surfaces being in general position and the availability of the generic projection plane. The issue of general position has been resolved by Basu et al (Basu et al., 1996), whose algorithm produces a roadmap in time $O(n^{k+1}d^{O(k^2)})$.

The maximum complexity of the entire $\mathcal{F}$ can be $O(n^k)$. Therefore, ignoring the dependence on the degree d, Canny's algorithm is close to optimal in the worst case, provided some representation of the entire $\mathcal{F}$ has to be output, since there are examples where $\mathcal{F}$ has $\Omega(n^k)$ connected components (Sharir, 1997). However, for motion planning, it is not necessary to compute the entire $\mathcal{F}$; it is sufficient to compute the connected component of $\mathcal{F}$ that contains $\mathbf{q}_{init}$ and check if $\mathbf{q}_{goal}$ lies in the same connected component. Halperin and Sharir (Halperin and Sharir, 1995a) showed that the combinatorial complexity of a single cell of $\mathcal{A}(\Gamma)$ in three dimensions is $O(n^{2+\epsilon})$, for any $\epsilon > 0$, where the constant of proportionality depends on ϵ and on the maximum degree of the surfaces. Recent work by Basu (Basu, 1998) extends this result to higher

dimensions. Basu shows that the combinatorial complexity of a single connected component of $\mathcal{A}(\Gamma)$ in k dimensions is $O(n^{k-1+\epsilon})$. Furthermore, under a certain natural geometric assumption on the objects, this bound can be improved to $O(n^{k-1})$.

The first lower bound on motion planning was established by Reif (Reif, 1979a) for the complexity of generalized mover's problem. The goal of this problem is to plan a free path for a robot made of an arbitrary number of polyhedral bodies connected together at some joint vertices, among a finite set of polyhedral obstacles. Reif proved that this problem is PSPACE-hard. PSPACE-hardness has also been established for a variety of other path planning problems (Latombe, 1991). These results strongly suggest that the complexity of path planning increases exponentially with the dimension of the configuration space. In light of these results, two different approaches have been taken. The first approach focuses on specific planning problems in low dimensional configuration spaces, where algorithms of reasonable complexity can be designed. An alternative approach is to resort to heuristic or approximate algorithms that are incomplete. These algorithms may fail to find a path even if one exists. We will discuss the first approach next, and the second approach in Sec. 2.5.2.

Specific Solutions

We focus on the complexity of motion planning problems with three degrees of freedom. See (Sharir, 1997) for complexity of motion planning problems with two degrees of freedom.

In a three dimensional configuration space, the maximum complexity of the entire $\mathcal{F}$ can be $O(n^3)$ (Sharir, 1997). Halperin and Sharir (Halperin and Sharir, 1995a) showed that the complexity of a single component is only $O(n^2 + \epsilon)$. They also propose an algorithm than can compute a single cell in $O(n^{2+\epsilon})$ time. We now consider certain specific cases where it is possible to design algorithms with better worst case asymptotic bounds. In the case where $\mathcal{R}$ is a robot that is capable of only translation, $\mathcal{F}$ can be computed using the algorithms for Minkowski sum computation (Sec. 2.4). We now consider certain cases involving rotation.

A line segment in a planar polygonal environment: Consider the case where $\mathcal{R}$ is a line segment ("rod," ladder") translating and rotating in a planar polygonal environment with m edges. The naive bound on the complexity of $\mathcal{F}$ is $O(m^3)$ However, one can show that the maximum combinatorial complexity of $\mathcal{F}$ is only $\Theta(m^2)$ (Sharir, 1997). Many near-quadratic algorithms have been developed including an algorithm

based on constructing a Voronoi diagram in $\mathcal{F}$ (Ó'Dúnlaing et al., 1987).

A convex polygon in a planar polygonal environment: Here $\mathcal{R}$ is a convex polygon bounded by p edges, free to translate and rotate in an arbitrary polygonal environment bounded by m edges. $\mathcal{F}$ is 3-dimensional, and there are at most $2pm$ contact surfaces, of maximum degree 4. The naive bound on the complexity of $\mathcal{F}$ is $O((pm)^3)$. Using *Davenport-Schinzel sequences*, one can show that the complexity of $\mathcal{F}$ is $O(pm\lambda_6(pm))$ where $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s composed of n symbols, and is nearly linear in n for any fixed s (Sharir, 1997). Agarwal et al. (Agarwal et al., 1997) presented an algorithm to compute $\mathcal{F}$ that takes $O(pm\lambda_6(pm)\log(pm))$ time.

A nonconvex polygon in a planar polygonal environment: Here $\mathcal{R}$ is an arbitrary polygonal region (not necessarily connected) bounded by p edges, translating and rotating in a polygonal environment bounded by m edges, as above. One can show that the maximum complexity of $\mathcal{F}$ is $\Theta((pm)^3)$ (Sharir, 1997). A single connected component of $\mathcal{F}$ can be computed in time $O((pm)^{2+\epsilon})$ using the algorithm by Halperin and Sharir (Halperin and Sharir, 1995a).

Many algorithms compute $\mathcal{F}$ by using a discrete number of slices along the orientation parameter (rotational degree of freedom). The boundary of $\partial\mathcal{F}$ is composed of ruled surface patches generated by contacts between a moving vertex $\mathcal{R}$ and an obstacle edge of $\mathcal{O}$ or between a moving edge of $\mathcal{R}$ and an obstacle vertex of $\mathcal{O}$. Avnaim et al. (Avnaim and Boissonnat, 1989) presented an algorithm for constructing $\mathcal{F}$ in $\Theta((pm)^3 \log pm)$ time. They computed the contact surfaces by tracing their generating line segments through a range of orientations. They explicitly computed the critical orientations which represent changes in the number of segments or their endpoints. The adjacent surfaces were linked in order to compute the boundary of $\mathcal{F}$.

Brost (Brost, 1991) presented a similar algorithm where the contact surfaces are computed by tracing the boundary curve segments wherever vertex-vertex contacts occur. This is followed by exact intersection of surfaces to compute the contact space.

Based on the above formulation, Sacks (Sacks, 1999; Sacks, 2001) presented the first complete motion planning algorithm that is practical for real-world applications. Sacks's algorithm is applicable to polygonal as well as curved primitives. For polygonal primitives, testing for a criticality reduces the problem to solving only a quadratic equation. In the worst case, the algorithm may need to test for $O(m^3n^3)$ criticality conditions.

2.5.2 Planning Approaches

At a broad level, the prior methods for motion planning can be classified into a few general approaches: *roadmap*, *cell decomposition*, *sampling*, and *potential field*. In this sub-section, we discuss roadmap methods, sampling based methods, and two types of cell decomposition methods – *exact cell decomposition* and *approximate cell decomposition*. Potential field methods have been surveyed in (Latombe, 1991; Choset et al., 2004).

Roadmap Methods

The idea underlying this approach is to convert the path planning problem in a k-dimensional configuration space to path planning in a network of one-dimensional curves while maintaining the connectivity in the robot's free space. This construction reduces the basic task into three subtasks: finding a subpath from $\mathbf{q}_{init}$ to the roadmap, finding a subpath on the roadmap and finding a subpath from roadmap to $\mathbf{q}_{goal}$. The various types of roadmaps proposed to achieve this task are *visibility graph*, *Voronoi diagram or retraction approach*, and *silhouettes*.

Visibility Graph Method: Visibility graph method is one of the earliest path planning methods (Nilsson, 1969) and has been widely used to implement path planners for mobile robots. The visibility graph is the undirected graph $G = (V, E)$ such that V consists of $\mathbf{q}_{init}$, $\mathbf{q}_{goal}$ and all the $\mathcal{C}$-obstacle vertices, while E consists of all line segments connecting two nodes that do not intersect the interior of the $\mathcal{C}$-obstacle region. G can be searched for a semi-free path between $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. This path, if exists, is a polygonal line connecting $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$ through C-obstacle vertices. In the case of a two-dimensional configuration space with polygonal C-obstacles, the resulting path is also the shortest semi-free path between $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. Using this approach, Lozano-Perez and Wesley (Lozano-Pérez and Wesley, 1979) proposed an $O(n^3)$ algorithm, which was improved to $O(n^2 \log n)$ by Lee (Lee, 1978) and to $O(n^2)$ by Guibas and Hershberger (Guibas and Hershberger, 1985) and Edelsbrunner (Edelsbrunner, 1987). An output sensitive algorithm of $O(k+n \log n)$, where k is the output size, was proposed by Ghosh and Mount (Ghosh and Mount, 1987). In a three-dimensional configuration space with polyhedral C-obstacles, the visibility graph can still be applied, but the computed path may not be the shortest semi-free path between $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. This is because, in general, the shortest path is a polygonal line whose vertices lie along the edges of the

C-obstacle.

Laumond (Laumond, 1987) presented an extension to the visibility graph method in the case where the C-obstacles are "generalized polygons", i.e., regions bounded by straight segments and/or circular arcs. Such C-obstacles occur when $\mathcal{R}$ is a generalized polygon translating at a fixed orientation among obstacles which are also modeled as generalized polygons.

A limitation of the visibility graph method is that it is not directly applicable to robots with rotational degrees of freedom. In practice, its applicability has been mainly limited to two-dimensional configuration spaces with a polygonal C-obstacle region (Latombe, 1991).

Voronoi Diagrams and Retraction: The retraction approach to motion planning uses the concept of retraction in topology to define a continuous mapping of the robot's free space $\mathcal{F}$ onto one-dimensional network of curves lying in $\mathcal{F}$. A retraction is a continuous map of a space onto a subspace leaving each point of the subspace fixed. Let X be a topological space and Y a subspace of X. If there exists a continuous map $r : X \rightarrow Y$ such that $r(y) = y$ for all $y \in Y$, then we say Y is a retract of X and r is a retraction. When $\mathcal{C} = R^2$ and the robot and obstacles are polygonal, the Voronoi diagram is a roadmap obtained by retraction (Ó'Dúnlaing et al., 1983). This approach provides the additional property that the obtained paths maximize the clearance between the robot and the obstacles.

(Ó'Dúnlaing et al., 1987) described a retraction method for planning the motion of a line segment moving among polygonal obstacles. In this method, the three dimensional free space is first retracted onto a two-dimensional variant of the Voronoi diagram. In a second step, this diagram is retracted onto a network of one-dimensional curves in $\mathcal{F}$. Canny and Donald (Canny and Donald, 1988) proposed a "simplified Voronoi diagram" which is easier to extend to higher-dimensional configuration spaces than the classical generalized Voronoi diagram.

Vleugels and Overmars (Vleugels and Overmars, 1997) presented a retraction method for 3D translational motion planning of a convex polyhedral robot translating among convex polyhedral obstacles. Instead of computing the exact Voronoi diagram, they used a spatial subdivision algorithm to compute a discretized approximation of the Voronoi diagram and use it for motion planning. Provided the subdivision algorithm performs a sufficient level of subdivision, the resulting approximate Voronoi diagram preserves the connectivity of the free space and can be used to guarantee complete

motion planning.

Silhouette Method: The silhouette method, proposed by Canny (Canny, 1987), is one of the most general methods of solving the basic motion planning problem. This is a complete algorithm that runs in single exponential time in the configuration space dimension. This method constructs the roadmap recursively. First, $\mathcal{F}$ is projected onto a generic two-dimensional plane and the *silhouette* of $\mathcal{F}$ under this projection is computed. Next, the critical values of the projection on some line are found, and a roadmap is constructed recursively within each slice of $\mathcal{F}$ at these critical values. The resulting "sub-roadmaps" are then merged with the silhouette to obtain the final roadmap.

Exact Cell Decomposition Methods

The crux of these methods is to partition the robot's free space into a collection of non-overlapping cells and to construct a connectivity graph representing the adjacency between the cells. A path is found by a simple graph search after locating $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ in the connectivity graph.

Schwartz and Sharir (Schwartz and Sharir, 1983a) proposed the first exact cell decomposition method for solving the general path planning problem, based on Collins *cylindrical algebraic decomposition* of the free space. This method has a time complexity doubly exponential in the configuration dimension and serves mainly as a proof of existence of a general path planning method (Latombe, 1991).

Many exact cell decomposition methods have been proposed for specific instances of the motion planning problem. For instance, Schwartz and Sharir (Schwartz and Sharir, 1983b) and Leven and Sharir (Leven and Sharir, 1985) presented methods for planning the motion of a robot modeled as a line segment ("ladder") translating and rotating among polygonal obstacles.

Stappen and Overmars (van der Stappen and Overmars, 1994) presented an efficient and simple paradigm for motion planning in the presence of fat obstacles. Here fatness means that there exists a constant $k > 0$ such that for all hyperspheres S centered inside the object E and not fully containing E, we have $k\ volume(E \cap S) > volume(S)$. The definition forbids fat obstacles to be long and thin or to have long or thin parts.

Approximate Cell Decomposition Methods

The approximate cell decomposition methods partition the configuration space into a collection of cells that have a predefined simple shapes (e.g., rectangloids). Unlike exact cell decomposition methods, these methods do not represent the free space exactly. Instead, they compute a conservative approximation to the free space. This approach was first introduced by Lozano-Perez and Brooks (Lozano-Pérez, 1981; Brooks and Lozano-Pérez, 1985). Subsequently, a large number of algorithms based on this approach have been proposed (Kambhampati and Davis, 1986; Donald, 1987; Zhu and Latombe, 1991).

These methods classify the cells into three types: *empty* cells that lie completely in free space, *full* cells that are completely within C-obstacle, and *mixed* cells that contain the boundary of the free space. The set of empty cells provide a conservative approximation of the free space and are used for path computation. The approximate cell decomposition methods are *resolution-complete*, i.e., they can find a path if one exists provided the resolution parameters are selected small enough (Latombe, 1991). These methods have been applied mostly to robots with a small number of degrees of freedom ($k \leq 4$). Other cell decomposition algorithms divide the configuration space using octrees or a lattice of points (Brooks and Lozano-Pérez, 1985; Donald, 1984; Zhu and Latombe, 1990). These algorithms are are also resolution-complete.

Sampling Based Planners

The need to perform planning in high degree configuration spaces motivated the development of probabilistic roadmap (PRM) planners, which resulted from the work of independent research groups (Kavraki and Latombe, 1994; Overmars and Svestka, 1995; Kavraki et al., 1996b).

In a PRM planner, the configuration space is sampled randomly according to various sampling strategies. Samples in the free space, called milestones, are kept; the samples in collision with obstacles are usually discarded. Nearby pairs of milestones are then connected in a simple manner. For example, if we know for every milestone its configuration space distance to an obstacle, we can use this information to argue that a straight line path (in configuration space) between two nearby milestones is fully in free space. These local connections transform the milestones into an undirected graph that constitutes the probabilistic roadmap for the problem. Given an initial and a final configuration of the robot, we can first connect them to the roadmap and then search

this graph for a path between the two configurations. Variations of this basic idea have been demonstrated to work on a number of challenging problems. The simplicity of this randomized method and its early successes have made it a popular subject of study in the motion planning community in recent years. A number of different sampling strategies have been developed, including shrinking the obstacles (Hsu et al., 1998), sampling near the free space boundaries (Amato et al., 1998; Boor et al., 1999) or medial axis of the free space (Wilmarth et al., 1999), and using visibility to reject unwanted samples (Simeon et al., 2000). We will refer to all of these planners collectively as sampling based planners.

An important characteristic of sampling based planners is that they can construct a roadmap of the free space without explicitly constructing the boundaries of the configuration space obstacles or representing the cells of $\mathcal{F}$. This enables them to be applicable to high dimensional configuration spaces.

Sampling based planners are not complete: They may fail to find a path even if one exists. Under certain assumptions, they achieve a weaker form of completeness. They have been shown to be *probabilistically complete*, i.e., if a solution path exists, the planner will eventually find it (Kavraki et al., 1996a).

One drawback of randomized sampling based methods is that the running time of the algorithm is variable; it depends on the the random samples chosen during the sampling process. Branicky et et al. (Branicky et al., 2001) proposed the use of quasi-random (also called deterministic) sampling techniques. This type of sampling has several advantages. First, the running time of the algorithm is guaranteed to be the same due to the deterministic nature of the sampling process. Second, they can minimize certain statistical measures such as discrepancy and dispersion. Finally, the resulting planners can be shown to be resolution-complete.

It is well known that there are two issues with sampling based planners. The first problem is the so called "narrow passage" problem. It refers to the difficulty of the planner to find paths through narrow passages in the free space (Fig. 1.8). The second problem is that the planner does not terminate when no path exists between the initial and goal configuration. This is because the planner cannot detect non-existence of any collision-free path. We discuss each of these problems further.

Narrow passage problem The original PRM planner sampled configurations in the free space using a uniform probability distribution (Choset et al., 2004). This type of sampling exhibits a poor performance when the free space has narrow passages.

To capture the connectivity of the free space accurately, the planner must sample configurations in the narrow passages. This is difficult, because narrow passages have small volumes, and the probability of drawing random samples from small sets is low. Consequently, the planner may not be able to find a valid path even though one may exist.

A large number of different sampling strategies have been proposed to alleviate the narrow passage problem (Amato et al., 1998; Hsu et al., 1998; Wilmarth et al., 1999; Boor et al., 1999; Hsu et al., 2003b). These methods bias the sampling in order to obtain a greater number of samples in the vicinity of the narrow passages. While these methods may be able to find paths through narrow passages in many scenarios, none of them can guarantee that a path will be found when one exists. We give a brief overview of some of these methods.

Hsu et al. (Hsu et al., 1998) sample a "dilated" $\mathcal{F}$ by allowing the robot to penetrate the obstacles by some small constant distance. The dilation of $\mathcal{F}$ widens the narrow passages, making it easier for the planner to capture the connectivity of the free space. The resulting samples that do not lie in $\mathcal{F}$ are then pushed onto $\mathcal{F}$ by performing local resampling operations.

OBPRM (Amato et al., 1998) generates samples near the boundary of configuration space obstacles. The motivation behind this kind of sampling is that narrow passages can be considered as narrow corridoors in $\mathcal{F}$ surrounded by obstacles. Initially, OBPRM generates many configurations at random from a uniform distribution. For each configuration $\mathbf{q}$ found in collision, it chooses a random direction, and moves along that direction until $\mathbf{q}$ becomes free, which is then added to the roadmap.

MAPPRM (Wilmarth et al., 1999) attempts to generate samples that are inside the narrow passages but as far away as possible from the obstacles. It computes samples near the Voronoi diagram of the free space to achieve this property. Although computing the Voronoi diagram in high dimensional configuration spaces is not practical, it is possible to find samples near the Voronoi diagram without computing it explicitly. MAPPRM uses a bisection method that moves each sample configuration until it is equidistant from two points on the boundary of $\mathcal{F}$. Similar techniques are also used by other methods that first compute the Voronoi diagram in the workspace and then use it to perform sampling in the configuration space (Choset et al., 2004).

Non-termination problem: If no path exists between the initial and goal configurations, then sampling based planners do not have an effective way of terminating.

Most of them simply pre-select a maximum number of milestones to be sampled and report that no path exists, if that number is reached, and no path has been found. This creates an ambiguity: We do not know whether there is actually no path between the two configurations, or whether we simply failed to generate a set of milestones sufficient for discovering the path. We will refer to this problem as the *non-termination problem*.

We are aware of only one work that addresses the non-termination problem. This method (Basch et al., 2001) provides "disconnection proofs" for motion planning, i.e., proofs showing that no path is possible between two robot configurations in a given environment. The authors present their method in a special setting: the goal is to move a polyhedral robot through a gate – a polygonal hole in an infinite planar wall in 3D. The main idea behind this method is as follows. Suppose the robot is able to pass through the gate. Then there will be a moment when exactly half of the volume of the robot remains on one side of the wall. Let d denote the orientation of the robot at that moment and S_d denote the section of the robot defined by the plane of the wall. Suppose we have a proof that for a certain specific orientation d, the section S_d does not fit in the gate. Then the same proof is likely to work for a neighborhood of orientations around d. While many of the ideas presented in this work are general, the method as well as the proposed techniques are currently specialized for the simple setting of a polyhedral robot moving through a gate.

Chapter 3

Topology Preserving Isosurface Extraction

In Chapter 1, we observed that all the surface extraction problems – Boolean operations, Minkowski sum operation, and configuration space computation – can be defined in terms of an arrangement $\mathcal{A}(\Gamma)$ of a set Γ of primitives, where each primitive in Γ is a polyhedral or an algebraic object. The primitives in Γ form a superset of the desired surface $\mathcal{E}$ – in Boolean operations and Minkowski sum computation, $\mathcal{E}$ corresponds to the boundary of the final solid defined by the operation; in configuration space computation, $\mathcal{E}$ corresponds to the boundary of the free space. In all the surface extraction problems, our goal is to compute an explicit boundary representation of $\mathcal{E}$. In this chapter, we will focus on the case where $\mathcal{E}$ is defined in terms of Boolean operations. Minkowski sum and configuration space computation will be the subjects of Chapters 4 and 5 respectively.

We adopt the sampling and reconstruction approach that was presented in Chapter 1. This approach represents $\mathcal{E}$ implicitly – as an isosurface of a scalar field obtained by performing minimum/maximum operations over the scalar fields associated with the primitives. Therefore we will also refer to $\mathcal{E}$ as the *exact isosurface*. Recall that this approach consists of the following steps:

1. **Sampling:** Generate a volumetric grid and compute a scalar field (e.g, a signed distance field) for each primitive at the grid points.

2. **Operation:** For each geometric operation (union/intersection/difference), perform an analogous operation (e.g., min/max) on the scalar fields of the primitives. At the end of this step, the scalar values at the grid points define a sampled scalar field for $\mathcal{E}$.

3. **Reconstruction:** Perform isosurface extraction on the grid using an MC-like algorithm

The output of isosurface extraction is a polygonal approximation $\mathcal{A}$ to $\mathcal{E}$. We refer to $\mathcal{A}$ as the *reconstructed isosurface*. This approach is illustrated in Fig. 1.6.

The accuracy of $\mathcal{A}$ is mainly governed by the *rate of sampling* – the resolution of the underlying volumetric grid. Due to inadequate sampling, $\mathcal{A}$ may suffer from various kinds of inaccuracies. Significant features, small components or handles present in $\mathcal{E}$ may not be captured in $\mathcal{A}$ (Figs. 3.4(a),3.4(b)). Insufficient sampling could also introduce "extraneous topology", i.e., $\mathcal{A}$ may have unwanted additional components or undesirable handles that were not present in $\mathcal{E}$ (Fig. 1.7).

In this chapter, we present an isosurface extraction algorithm that provides geometric and topological guarantees on $\mathcal{A}$. Our algorithm guarantees that $\mathcal{A}$ is topologically equivalent to $\mathcal{E}$ and has a bounded two-sided Hausdorff error. Our algorithm primarily relies on two geometric criteria – *complex cell criterion* and *star-shaped criterion* (defined in Sec. 3.4). These two criteria are combined to design a *sampling condition*. We show that if the volumetric grid satisfies this sampling condition, then an MC-like algorithm can be applied to the grid and produce a reconstruction $\mathcal{A}$ that is topologically equivalent to $\mathcal{E}$. Furthermore, we augment the sampling condition to also bound the two-sided Hausdorff error in $\mathcal{A}$. To generate a volumetric grid satisfying the sampling condition, we perform adaptive subdivision by a recursive application of the sampling condition. In the absence of degeneracies, the adaptive subdivision will terminate once all the grid cells satisfy the sampling condition. Then isosurface extraction can be performed on the resulting grid using an MC-like algorithm.

A significant part of the work described in this chapter was done jointly with Shankar Krishnan and T.V.N Sriram, and is described in (Varadhan et al., 2004). The chapter is organized as follows. Section 1 introduces the notation and definitions. Section 2 presents an overview of the Marching Cubes algorithm. In Section 3, we analyze the errors in the output of Marching Cubes that can be caused due to inadequate resolution of the grid. In Section 4, we present a sampling condition on the volumetric grid to ensure topology preservation. In Section 5, we present an adaptive subdivision algorithm to generate a volumetric grid. Section 6 analyzes the behavior of the adaptive subdivision algorithm and presents conditions for its termination. Section 7 discusses degenerate cases for our algorithm. In Section 8, we present a simple technique to guarantee a tight geometric error bound on the approximation. Section 9 disusses a few issues that arise when performing isosurface extraction on adaptive grids. In Section 10,

we present techniques to improve the performance of the algorithm. Section 11 discusses the performance of the algorithm. Section 12 describes the implementation of the algorithm, and presents three different applications: Boolean operations, simplification, and remeshing.

3.1 Notation and Preliminaries

Input

We assume that the exact surface $\mathcal{E}$ is obtained by performing Boolean operations (union, intersection, difference, complement) on a set of primitives in $\mathbb{R}^3$. The input to our algorithm is a Boolean expression and a set $\Gamma = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ of primitives.

We assume that each primitive $\mathcal{P}_i$ is a closed 2-manifold, and is either a polyhedral or an algebraic object. We assume each $\mathcal{P}_i$ bounds a solid, which we denote as $\widetilde{\mathcal{P}}_i$. The Boolean operations are performed on the solids $\widetilde{\mathcal{P}}_i$, which yields a final solid $\widetilde{\mathcal{E}}$. The exact surface is the boundary of $\widetilde{\mathcal{E}}$ and is denoted as $\mathcal{E}$. We assume that $\mathcal{E}$ is a closed 2-manifold.

Output

The output of our algorithm is a polygonal approximation $\mathcal{A}$ to the exact surface $\mathcal{E}$.

Notation

We introduce the notation used in the rest of this chapter.

- We use lower case bold letters such as $\boldsymbol{p}, \boldsymbol{q}$ to refer to points in $\mathbb{R}^3$.
- We use upper case letters such as $\mathcal{P}, \mathcal{Q}, \mathcal{P}_1$ to refer to geometric primitives. We assume that each primitive is a closed manifold. Each primitive bounds a solid, which we will refer to as the **primitive solid**, and denote it as $\widetilde{P}$.
- Boolean operations are defined on primitive solids. $\widetilde{\mathcal{P}}_1 \cup \widetilde{\mathcal{P}}_2$, $\widetilde{\mathcal{P}}_1 \cap \widetilde{\mathcal{P}}_2$, $\widetilde{\mathcal{P}}_1 \setminus \widetilde{\mathcal{P}}_2$ denote union, intersection, and difference operations on $\widetilde{\mathcal{P}}_1$ and $\widetilde{\mathcal{P}}_2$ respectively. We will assume that these operations are *regularized* (Sec. 1.1).

  Let $\widetilde{\mathcal{S}}$ denote a solid defined by Boolean operations. By a slight abuse of notation, we will use $\widetilde{\mathcal{S}}$ to also refer to the **Boolean expression** of the solid, which is the expression that defines $\widetilde{\mathcal{S}}$ in terms of Boolean operations over a set of primitive solids, e.g., $\widetilde{\mathcal{S}} = \widetilde{\mathcal{P}}_1 \cup \widetilde{\mathcal{P}}_2$. A Boolean expression of a surface $\mathcal{S}$ is defined as the Boolean expression of the corresponding solid $\widetilde{\mathcal{S}}$.

- The abbreviation *w.r.t* means *with respect to*.
- ∂S, $\overline{S}$, int S, and cl S respectively denote the boundary, complement, interior and closure of a set S.
- Let $d \geq 1$ be an integer. Let $\mathbf{o}$ be the origin of $\mathbb{R}^d$. $\mathbb{S}^{d-1}$ and $\mathbb{B}^d$ denote the $(d-1)$-dimensional sphere and d-dimensional ball respectively. They are defined as:

$$\begin{aligned} \mathbb{S}^{d-1} &= \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{xo}| = 1\} \\ \mathbb{B}^d &= \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{xo}| < 1\} \end{aligned}$$

  Also define the 0-ball $\mathbb{B}^0 = \{\mathbf{o}\}$.
- A **scalar field** is a function, $f : \mathbb{R}^3 \to \mathbb{R}$, which assigns each $\boldsymbol{x} \in \mathbb{R}^3$ a scalar value $f(\boldsymbol{x})$. Given a continuous scalar field and a scalar value s, the **isosurface** with **isovalue** s is the set, $\{\boldsymbol{x} \in \mathbb{R}^3 \mid f(\boldsymbol{x} = s\}$, of points with identical scalar value s. Unless otherwise stated, the isovalue is assumed to be zero.
- For a closed polyhedral primitive, the scalar field is typically defined as a **signed distance field**. For a closed surface $\mathcal{S}$, the signed distance field $D : \mathbb{R}^3 \to \mathbb{R}$ is a continuous function that at a point $\boldsymbol{x}$ measures the distance between $\boldsymbol{x}$ and $\mathcal{S}$. This value is positive or negative depending on whether $\mathbf{x}$ lies outside or inside $\mathcal{S}$. The distance can be defined under any reasonable norm (e.g., Euclidean, max-norm). For an algebraic primitive defined by a polynomial function $g(\mathbf{x}) = 0$, the scalar field is defined as g.
- A **sampling** of a scalar field f will refer to a volumetric grid in $\mathbb{R}^3$ such that every grid vertex $\mathbf{p}$ stores the value $f(\mathbf{p})$ of the scalar field at point $\mathbf{p}$. A sampling of a primitive is defined as the sampling of the associated scalar field. We will use the term *sampling* to also refer to the process of generating such a volumetric grid.
- The term **reconstruction** refers to the process of isosurface extraction by using Marching Cubes (MC) algorithm (Lorensen and Cline, 1987) or its variants (Montani et al., 1994; Kobbelt et al., 2001; Ju et al., 2002; Varadhan et al., 2003b). We will collectively refer to all these algorithms as **MC-like methods**.
- The exact surface $\mathcal{E}$ and the approximation $\mathcal{A}$ may also be referred to as the **exact isosurface** and the **reconstructed isosurface** respectively.

- $\mathcal{G}$ denotes a volumetric grid in $\mathbb{R}^3$. Unless otherwise stated, the grid is assumed to be an octree (Samet, 1989). In a few instances, we will also consider tetrahedral grids (Shewchuk, 1998); we will then state this explicitly. The letter C will be used to refer to a single cell in $\mathcal{G}$. When referring to the cell as a geometric primitive, we will refer to it as a **voxel**. The boundary of a cell consists of faces, edges, and vertices. A cube-shaped grid cell consists of one voxel, six faces, twelve edges, and eight vertices. All of them are assumed to be closed sets. The symbols ϑ, f, e, and v will refer, respectively, to a voxel, a face, an edge, and a vertex.

  Let c be an edge/face/voxel of a cell C. The **size** of c, denoted as $\|c\|$, is the maximum distance between any two vertices in c. The **size** of C, denoted as $\|C\|$, is the size of its voxel. The **width** of c is the minimum distance between any two vertices in c. The **width** of C is the width of its voxel.

- By a restriction of a set S w.r.t another set T, we mean $S \cap T$, which is denoted as S_T. In particular, we will use the following notation frequently:

  The restrictions of $\mathcal{E}$ w.r.t a cell C, voxel ϑ, a face f, or an edge e are denoted as $\mathcal{E}_C$, $\mathcal{E}_\vartheta$, $\mathcal{E}_f$, and $\mathcal{E}_e$ respectively. Similarly, we can define $\mathcal{A}_C$, $\mathcal{A}_\vartheta$, $\mathcal{A}_f$, and $\mathcal{A}_e$. The restriction w.r.t the cell is defined as the restriction w.r.t the voxel of the cell.

- A **homeomorphism** is a continuous bijective mapping with a continuous inverse (Munkres, 1975). Two objects $\mathcal{P}$ and $\mathcal{Q}$ are **topologically equivalent** if there exists a homeomorphism $\mathcal{H} : \mathcal{P} \to \mathcal{Q}$. We denote this as $\mathcal{P} \approx \mathcal{Q}$.

  We will call an object $\mathcal{P}$ a **topological disk** if $\mathcal{P} \approx \mathbb{B}^d$ for $d > 0$.

  An object is **d-manifold** if every point has a neighborhood that is topologically equivalent to $\mathbb{R}^d$.

  A manifold is **connected** if for any two points on the manifold, there exists a path between them in the set. If two points $\mathbf{p} \in S$ and $\mathbf{q} \in S$ are connected in a set S, we denote this as $\mathbf{p} \xleftrightarrow{S} \mathbf{q}$.

  A manifold is said to be **simply connected** if any simple closed curve on the manifold can be shrunk to a point continuously in the set.

- Let $d(\boldsymbol{p}, \boldsymbol{q})$ denote the distance (in a suitable metric) between two points $\boldsymbol{p}$,$\boldsymbol{q} \in \mathcal{R}^n$. Unless explicitly stated, the metric is assumed to be Euclidean. Given a set

$\mathcal{Q}$, we define the distance between a point $\mathbf{p}$ and $\mathcal{Q}$ as follows:

$$d(\boldsymbol{p}, \mathcal{Q}) = \min\{d(\boldsymbol{p}, \boldsymbol{q}) \mid \boldsymbol{q} \in \mathcal{Q}\}$$

The diameter of a set P is defined as:

$$diam(\mathcal{P}) = \max\{d(\boldsymbol{p_1}, \boldsymbol{p_2}) \mid \boldsymbol{p_1}, \boldsymbol{p_2} \in \ \mathcal{P}\}$$

The one-sided Hausdorff distance between two sets $\mathcal{P}$ and $\mathcal{Q}$ is defined as follows:

$$h(\mathcal{P}, \mathcal{Q}) \quad = \quad \max\{\min d(\boldsymbol{p}, \mathcal{Q}) \mid \boldsymbol{p} \in \mathcal{P}\}$$

Note that the above definition is not symmetric, i.e., $h(\mathcal{P}, \mathcal{Q})$ is not necessarily equal to $h(\mathcal{Q}, \mathcal{P})$. $h(\mathcal{P}, \mathcal{Q})$ and $h(\mathcal{Q}, \mathcal{P})$ are also referred to as the forward and backward Hausdorff distances respectively. The **two-sided Hausdorff** distance is defined as:

$$H(\mathcal{P}, \mathcal{Q}) \quad = \quad \max(h(\mathcal{P}, \mathcal{Q}), h(\mathcal{Q}, \mathcal{P}))$$

- By a **surface**, we refer to a 2-manifold in $\mathbb{R}^3$.
  - A set $\mathcal{P} \subseteq \mathcal{S}$ is a **component** of surface $\mathcal{S}$ if $\mathcal{P}$ is connected and there exists no point $\mathbf{p} \in \mathcal{P}$ such that $\mathbf{p} \overset{\mathcal{S}}{\longleftrightarrow} \mathbf{q}$ for some point $\mathbf{q} \in \mathcal{S} \setminus \mathcal{P}$.
  - We call a component $\mathcal{P}$ a **component with boundary** if it has a nonempty boundary. Otherwise, we call $\mathcal{P}$ a **closed component**.
- Let S be a nonempty subset of $\mathbb{R}^n$. The set Kernel(S) consists of all $\boldsymbol{s} \in S$ such that for any $\boldsymbol{x} \in S$, we have $\boldsymbol{sx} \subseteq S$. S is **star-shaped** if Kernel(S) $\neq \emptyset$. We call a point belonging to Kernel(S) as a **guard** of S. See Fig. 3.7

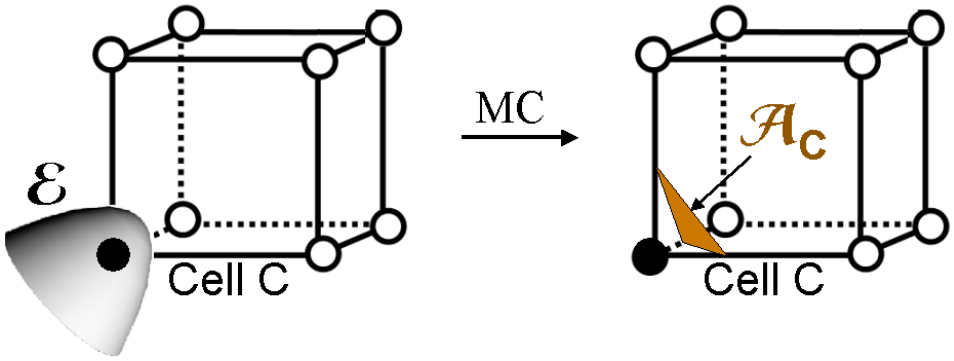


Figure 3.1: Marching Cubes: *This figure shows a cell intersecting the isosurface. The output of Marching Cubes algorithm is shown on the right.*

3.2 Overview of Marching Cubes

Given a continuous scalar field, $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ and a scalar value s, an isosurface with isovalue s is the set, $\{\boldsymbol{p} \mid f(\boldsymbol{p}) = s\}$, of points with identical scalar value s. In the following discussion, we will assume that the isovalue s is zero. The Marching Cubes algorithm (MC) (Lorensen and Cline, 1987) is a simple method for generating a polygonal reconstruction $\mathcal{A}$ of an isosurface $\mathcal{E}$ in $\mathbb{R}^3$. The input to MC is a volumetric cubic grid with a scalar value at each grid vertex. MC performs reconstruction by extracting surfaces separately in every grid cell. The algorithm iterates through all grid cells, hence the term marching cubes. The algorithm operates on a single grid cell C to produce a polygonal approximation $\mathcal{A}_C$ of $\mathcal{E}_C$.

1. **Classification:** Classify each vertex of C as inside or outside $\mathcal{E}$. We refer to this inside/outside classification as the *sign* of the vertex. The sign of a point $\boldsymbol{p}$ is defined as the sign of the scalar value $f(\boldsymbol{p})$. The signs at the 8 vertices of C define a *sign configuration* $(s_1, \ldots, s_8)$ where s_i is 1 if the i^{th} vertex is positive and 0 otherwise, $i = 1, \ldots, 8$.

2. **Detection:** Test if $\mathcal{E}$ intersects C, i.e., if $\mathcal{E}_C = \mathcal{E} \cap C \neq \emptyset$. This test is performed by checking if C exhibits a *sign change*, i.e., not all the vertices of C have the same sign. In this case, proceed to Step 3. On the other hand, if all the grid vertices of C have the same sign, then assume that $\mathcal{E}_C = \emptyset$ and do not perform a reconstruction within C.

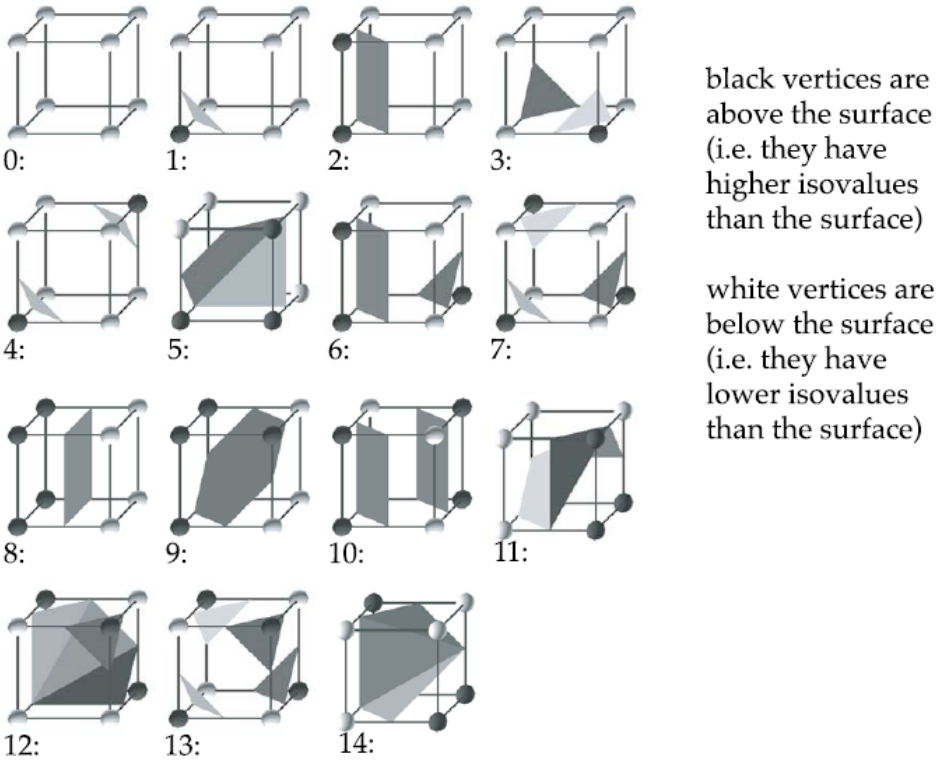


Figure 3.2: Marching Cubes Cases (Courtesy Hamish Carr).

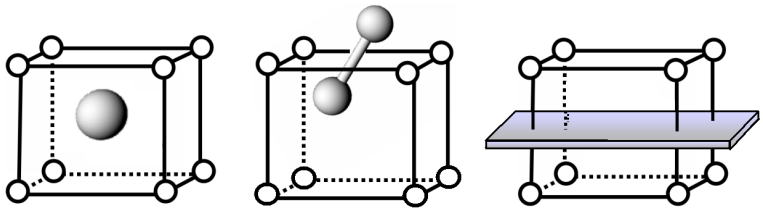
3. **Reconstruction:**

   - For each edge of C whose endpoints have different signs, estimate an edge point by linear interpolation of the scalar field along the edge.
   - Use the edge points enumerated in Step 2 to construct one or more polygonal facets separating the vertices with different signs. This is done as per the sign configuration of C. See Fig. 3.1. Since each of the 8 vertices of a cube can be either positive or negative, there are $2^8 = 256$ possible sign configurations. Lorenson & Cline (Lorensen and Cline, 1987) used symmetries between different sign configurations to reduce them to 15 basic cases. Fig. 3.2 shows all the 15 cases. They stored each of these cases in a look-up table, and used them to find $\mathcal{A}_C$.

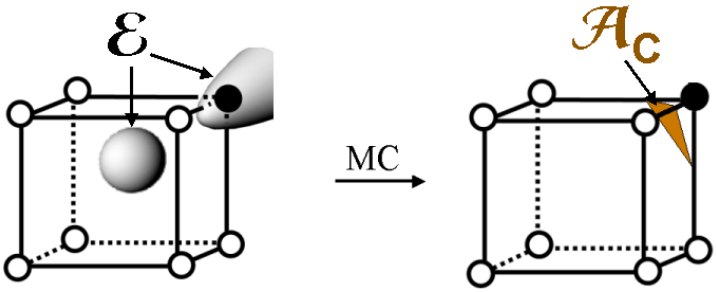
The union of $\mathcal{A}_C$ over all the grid cells produces the reconstructed isosurface $\mathcal{A}$.

There are a large number of extensions of MC (Montani et al., 1994; Kobbelt et al., 2001; Ju et al., 2002; Varadhan et al., 2003b), and all of them follow the same general

approach. We refer to these algorithms collectively as **MC-like methods**.



(a) Unreliable Detection



(b) Unreliable Reconstruction

Figure 3.3: Errors in MC-like reconstruction: *When the isosurface has complicated features, MC-like methods are unreliable, and may produce inaccurate output. Fig. (a) shows three cases where the isosurface intersects the cell, but the MC-like methods cannot detect the presence of the isosurface with the cell. In these cases, MC-like methods produce no output within the cell. Fig. (b) shows a case where they output a polygon, but they do not reconstruct the surface component in the interior of the cell.*

3.3 Geometric and Topological Errors

MC-like methods rely on the sign configuration of a cell C for two tasks: (a) to *detect the isosurface*, i.e., if $\mathcal{E}_C \neq \emptyset$ and (b) to reconstruct $\mathcal{E}_C$ – the portion of the isosurface within the cell. The reliance on sign configuration is merely a heuristic and not a fool-proof test.

There can be two kinds of problems:

- **Unreliable Detection:** MC-like methods may wrongly assume that $\mathcal{E}$ does not intersect C and perform no reconstruction within C (Fig. 3.3(a)).
- **Unreliable Reconstruction:** MC-like methods produce $\mathcal{A}_C$ by indexing into a lookup table using the sign configuration of C. The sign configuration of C may not adequately capture $\mathcal{E}_C$. As a result, $\mathcal{A}_C$ may be a poor approximation to $\mathcal{E}_C$ (Fig. 3.3(b)).

The above two problems can lead to both geometric errors and topological errors in $\mathcal{A}$. We discuss each of them separately.

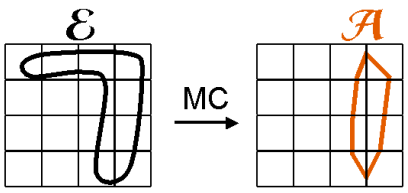
3.3.1 Geometric Errors

The inability of MC-like methods to detect $\mathcal{E}$ reliably can lead to a large geometric error in $\mathcal{A}$. See Fig. 3.4(a). One way of measuring the geometric error of $\mathcal{A}$ is to compute the Hausdorff distance between $\mathcal{E}$ and $\mathcal{A}$. For a definition of Hausdorff distance, see Sec. 3.1. From now on, we will assume that the geometric (or Hausdorff) error of $\mathcal{A}$ is equal to the two-sided Hausdorff distance $H(\mathcal{A}, \mathcal{E})$.

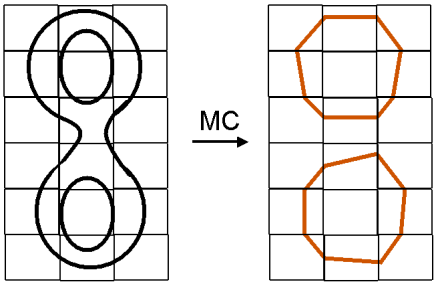
While it is possible for MC-like methods to guarantee a bound on the one-sided Hausdorff distance $h(\mathcal{A}, \mathcal{E})$, this guarantee is applicable only *one-way*: They do not bound the backward distance $h(\mathcal{E}, \mathcal{A})$; there can be points on $\mathcal{E}$ that are *far* from $\mathcal{A}$. See Fig. 3.4(a). Thus the geometric error $H(\mathcal{A}, \mathcal{E})$ can be quite large; consequently, $\mathcal{A}$ can be a poor approximation to $\mathcal{E}$.

3.3.2 Topological Errors

The problems of unreliable detection and unreliable reconstruction can cause topological errors in $\mathcal{A}$. Due to unreliable detection, MC-like methods may miss small surface components or handles present in $\mathcal{E}$. As a result, these features may not be captured in $\mathcal{A}$. See Fig. 3.4(b).



(a) Geometric Error



(b) Topological Error

Figure 3.4: *This figure shows 2D examples where MC-like methods produce output with geometric and topological errors. In Fig (a), a feature present at the top of $\mathcal{E}$ has not been captured in $\mathcal{A}$. In Fig (b) $\mathcal{A}$ does not capture all the components present in $\mathcal{E}$*

Due to the problem of unreliable reconstruction, the topology of $\mathcal{A}_C$ in a cell C may not match the topology of $\mathcal{E}_C$. This can introduce "extraneous topology" in $\mathcal{A}$, i.e., $\mathcal{A}$ may have unwanted additional components or undesirable handles that were not present in $\mathcal{E}$ (Fig. 1.7).

3.3.3 Sampling Issues

The above geometric and topological errors occur due to insufficient resolution of the volumetric grid. These errors could be avoided by choosing a grid with a sufficiently high resolution. The main question that arises is how much resolution is sufficient to ensure an accurate approximation. Our goal is to come up with a condition for what constitutes a *sufficient resolution* of the volumetric grid.

Our overall approach proceeds by sampling and reconstruction. In any approach based on sampling and reconstruction, the key to an accurate output is to ensure that the sampling satisfies the requirements of the reconstruction. The nature of require-

ments depend on the reconstruction method. To ensure accuracy of MC-like reconstruction methods, we impose the following requirements on the sampling.

We require that the every cell in the volumetric grid must satisfy two requirements:

1. **Reliable Detection:** $\mathcal{E}_C \neq \emptyset$ if and only if C exhibits a sign change.
2. **Reliable Reconstruction:** $\mathcal{E}_C \approx \mathcal{A}_C \approx \mathbb{B}^2$.

Failure to satisfy the above requirement implies an insufficient rate of sampling for MC-like reconstruction methods, which can cause both geometric and topological errors in $\mathcal{A}$. We avoid these errors by using a sampling condition that enforces the requirement. We present this condition in the following section.

3.4 Sampling Condition

We present a sampling condition that ensures accuracy during isosurface extraction. If the volumetric grid satisfies the sampling condition, then we can apply an MC-like method to obtain an approximation $\mathcal{A}$ with a bounded two-sided Hausdorff error and same topology as $\mathcal{E}$.

We first address the issue of ensuring that MC-like methods preserve topology during isosurface extraction. We defer the problem of bounding the geometric error to Sec. 3.8. We can achieve our goal of topology preservation by making sure that the requirements of MC-like methods are satisfied, i.e., $\mathcal{E}$ should intersect a grid cell C in a simple manner, and should have a simple topology within C. In particular, we ensure that $\mathcal{E}_C$ is a topological disk. We present a simple condition to ensure this property. Then we show that this condition is sufficient – if the every cell in the volumetric grid satisfies this condition, then an MC-like method can be reliably applied to the grid to obtain a topologically correct approximation.

Our sampling condition consists of two geometric criteria: complex cell criterion and star-shaped criterion. For these criteria to be well defined, we require that the isosurface intersect the grid cells in a *non-degenerate* manner. We present the non-degeneracy requirement followed by the sampling criteria.

3.4.1 Non-Degeneracy Condition

We require that the isosurface should not *graze* the boundary of the cell. See Fig. 3.5. To avoid such situations, we require that the grid cells satisfy a non-degeneracy

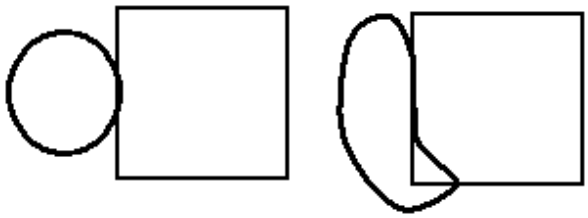


Figure 3.5: Grazing Contact: *This figure shows (in 2D) two instances of an isosurface touching the boundary of a grid cell. Our algorithm requires that all the grid cells satisfy a* non-degeneracy condition *that prohibits such contacts.*

condition.

Edelsbrunner & Shah (Edelsbrunner and Shah, 1994) presented a condition for non-degeneracy in context of Delaunay triangulation. We use their condition to define a non-degenerate intersection of $\mathcal{E}$ and a grid cell C. Let c denote a voxel, face, or edge of C. We say $\mathcal{E}$ *intersects* c *generically* if

1. $\mathcal{E} \cap c = \emptyset$ or
2. $\mathcal{E} \cap (\text{int } c) = \text{relative int}(\mathcal{E} \cap c)$ and $\mathcal{E} \cap c$ has the *right dimension* where the right dimension for a voxel, a face, and an edge is 2, 1, and 0 respectively.

DEFINITION 1 *A cell is* **non-degenerate** *if* $\mathcal{E}$ *intersects each of its voxel, faces, and edges generically.*

3.4.2 Complex Cell Criterion

We define a voxel (face) of a grid cell to be *complex* if it intersects $\mathcal{E}$ and the grid vertices belonging to the voxel (face) do not exhibit a sign change (see Figs. 3.6(a) & 3.6(b)). An edge of the grid cell is said to be *complex* if $\mathcal{E}$ intersects the edge more than once (see Fig. 3.6(c)).

MC-like methods that operate on cubical grid cells cannot handle certain sign combinations in a topologically reliable manner. There are two types of ambiguity — *face ambiguity* and *voxel ambiguity* (Wilhelms and Gelder, 1990b). When the signs at the vertices of a single face alternate during counterclockwise (or clockwise) traversal, the resulting configuration is a face ambiguity. A voxel ambiguity results when any pair of diagonally opposite vertices have one sign while the other vertices have a different sign (see Fig. 3.6(d)). We refer to both face ambiguity and voxel ambiguity as an ambiguous sign configuration.

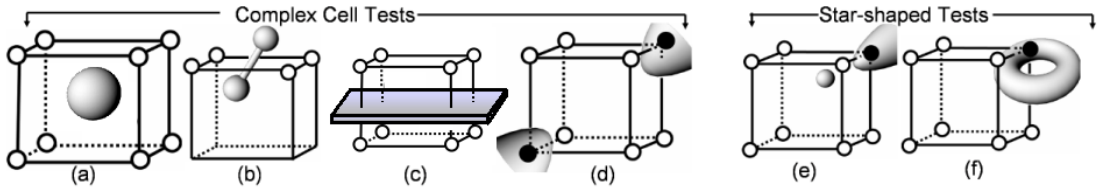


Figure 3.6: Complex cell and Star-shaped Test Cases: *This figure shows the different cases corresponding to the complex cell and star-shaped test. Figs (a), (b), (c) and (d) show cases of complex voxel, complex face, complex edge, and topological ambiguity. The white and black circles denote positive and negative grid points respectively. Fig. (e) shows the case where the isosurface is not star-shaped w.r.t a voxel. In Fig (f), the restriction of the isosurface to the right face of the cell is not star-shaped.*

DEFINITION 2 *1.* **Complex cell:** A non-degenerate cell is *complex* if it has a *complex voxel, complex face, complex edge,* or an *ambiguous sign configuration.*

2. **Complex cell criterion ($\mathcal{C}^{\square}$) :** *A non-degenerate cell C satisfies $\mathcal{C}^{\square}$ if C is not complex.*

Intuitively, the complex cell criterion ensures that $\mathcal{E}$ intersects C in a simple manner most of the times. However, this criterion by itself is not sufficient. There are cases where a C may not be complex, but $\mathcal{E}_C$ may have a complicated topology (see Figs. 3.6(e) & 3.6(f)). In such cases, $\mathcal{A}_C$ will be a poor approximation to $\mathcal{E}_C$. In Fig. 3.6(e), MC-like methods miss the surface component present in the interior of the cell [1]. Similarly in Fig. 3.6(f), MC-like methods will not reconstruct the handle present in $\mathcal{E}$. We avoid such situations by enforcing a star-shaped criterion within all the grid cells.

3.4.3 Star-shaped Criterion

We begin by defining the star-shaped property. We consider a few cases:

1. Let S be a d-dimensional nonempty subset of $\mathbb{R}^d$. The set Kernel(S) consists of all $\boldsymbol{s} \in S$ such that for any $\boldsymbol{x} \in S$, we have $\boldsymbol{sx} \subseteq S$. Set S is *star-shaped* if Kernel(S) $\neq \emptyset$. We call a point belonging to Kernel(S) a *guard* of S. Intuitively, a guard can see every point within a star-shaped primitive. See Fig. 3.7.

[1] We cannot detect the presence of the internal surface component and reconstruct it independently because we do not have an explicit representation of $\mathcal{E}$.

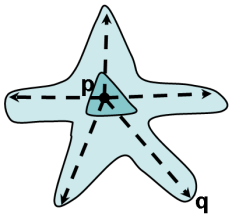


Figure 3.7: Star-shaped Primitive *The figure shows a star-shaped primitive and its kernel (shaded region in the middle). P is a guard of the kernel.*

2. Let S be a $(d-1)$-dimensional closed manifold in $\mathbb{R}^d$. The set Kernel(S) consists of all $\boldsymbol{o} \in \mathbb{R}^d$ such that for any $\boldsymbol{x} \in S$ we have $\mathbf{ox} \cap S = \{\mathbf{x}\}$. S is *star-shaped* if Kernel(S) $\neq \emptyset$. A point belonging to Kernel(S) is a *guard* of S. We assume that $\mathcal{E}$ is a 2-manifold in $\mathbb{R}^3$. Therefore we use this definition when we say $\mathcal{E}$ is star-shaped.

3. Next we define the star-shaped property for a cell. We say $\mathcal{E}$ is *star-shaped with respect to* (w.r.t) *a voxel* ϑ if there exists a point $\mathbf{o} \in \mathbb{R}^3$ such that for any $\boldsymbol{x} \in \mathcal{E}_\vartheta = \mathcal{E} \cap \vartheta$ we have $\mathbf{ox} \cap \mathcal{E}_\vartheta = \{\mathbf{x}\}$. Point $\mathbf{o}$ is a guard of $\mathcal{E}_\vartheta$. We note that it is not required to lie within the voxel. This makes the condition less restrictive. Consider a face f. We treat $\mathcal{E}_f = \mathcal{E} \cap f$ as a curve in $\mathbb{R}^2$. Let Π_f denote the plane containing f. We say $\mathcal{E}$ is *star-shaped w.r.t* f if there exists a point $\mathbf{o} \in \Pi_f$ such that for any $\boldsymbol{x} \in \mathcal{E}_f$, we have $\mathbf{ox} \cap \mathcal{E}_f = \{\mathbf{x}\}$. Point $\mathbf{o}$ is a guard of $\mathcal{E}_f$. We define $\mathcal{E}$ to be *star-shaped w.r.t a cell* if it is star-shaped w.r.t the cell's voxel, and each of its faces.

DEFINITION 3 Star-shaped criterion ($\mathcal{C}^\star$) : *A cell C satisfies $\mathcal{C}^\star$ if $\mathcal{E}$ is star-shaped w.r.t C.*

The surface is star-shaped w.r.t the cell in Figs. 3.6(a), (b), (c). On the other hand, Figs. 3.6(e) & 3.6(f) show cases where the surface is not star-shaped w.r.t a voxel or a face of the cell.

3.4.4 Topology Preserving Isosurface Extraction

DEFINITION 4 *A cell C satisfies $\mathcal{C}^{\square\star}$ if*

1. $\mathcal{E}_C = \emptyset$ *or*

2. *(a) C is non-degenerate,*

(b) C satisfies $\mathcal{C}^{\square}$, and

(c) C satisfies $\mathcal{C}^{\bigstar}$.

3. If C satisfies $\mathcal{C}^{\square\bigstar}$, we refer to it as a $\mathcal{C}^{\square\bigstar}$-cell.

We now present our main result on topology-preserving isosurface extraction: If all the grid cells are $\mathcal{C}^{\square\bigstar}$-cells, then MC-like algorithms extract an approximation $\mathcal{A}$ that has the same topology as $\mathcal{E}$. In order to prove this result, we first show that the intersection of $\mathcal{E}$ with a voxel, face, or edge is homeomorphic to a disk in the right dimension (provided the intersection is non-empty). This property is then used to establish topological equivalence. We begin by defining the properties of MC-like methods.

Properties of MC-like Reconstruction Methods

Recall that $\mathcal{A}$ is a piece-wise linear approximation of $\mathcal{E}$ obtained by performing isosurface extraction using an MC-like method. We require that the MC-like method satisfy the following properties:

Property 1: The signs of the scalar field at all the grid vertices are preserved during isosurface extraction; every grid vertex has an identical sign w.r.t both $\mathcal{E}$ and $\mathcal{A}$.

This property is satisfied by the original Marching Cubes algorithm (Lorensen and Cline, 1987) and most of its extensions (Montani et al., 1994; Kobbelt et al., 2001). However, there are a few methods such as Dual Contouring (Ju et al., 2002) that may not satisfy the property. We propose an extension to the Dual Contouring algorithm that satisfies the property (Sec. 3.9).

Property 2: Let c be an edge, face, or voxel of a cell C such that c exhibits a sign change, i.e., not all the vertices of c have the same sign. If C is a $\mathcal{C}^{\square\bigstar}$-cell then

$$\mathcal{A}_c \approx \mathbb{B}^k$$

where k is 0, 1, or 2 depending on whether c is an edge, face, or a voxel.

Consider an edge e that exhibits a sign change. MC-like algorithms output one intersection point along e. Therefore, we have $\mathcal{A}_e \approx \mathbb{B}^0$.

Let c be a face or voxel of a $\mathcal{C}^{\square\bigstar}$-cell cell. The sign configurations for which MC-like reconstruction is not a topological disk are topologically ambiguous. Because $\mathcal{C}^{\square\bigstar}$-cells, by definition, are not topologically ambiguous, the remaining sign configurations

always result in a reconstruction such that $\mathcal{A}_c \approx \mathbb{B}^k$. This is a property of MC-like reconstruction methods.

Proof of Topology Preserving Reconstruction

LEMMA 1 *Let e be an edge of a $\mathcal{C}^{\square\bigstar}$-cell such that $\mathcal{E}_e \neq \emptyset$. Then*

$$(i) \quad \mathcal{E}_e \approx \mathbb{B}^0$$
$$(ii) \quad \mathcal{A}_e \approx \mathbb{B}^0$$

Proof:
(i) Because $\mathcal{E}_e \neq \emptyset$, edge e intersects $\mathcal{E}$. There has to be exactly one intersection point: otherwise e will be complex. Therefore, $\mathcal{E}_e \approx \mathbb{B}^0$.

(ii) Because e is not complex and intersects $\mathcal{E}$, e will exhibit a sign change. By Property 2, we have $\mathcal{A}_e \approx \mathbb{B}^0$.

□

LEMMA 2 *Let f be a face of a $\mathcal{C}^{\square\bigstar}$-cell such that $\mathcal{E}_f \neq \emptyset$. Then*

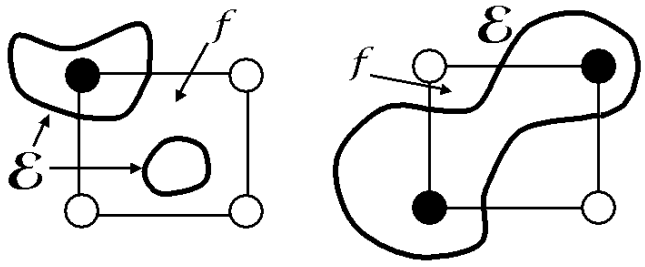
$$(i) \quad \mathcal{E}_f \approx \mathbb{B}^1$$
$$(ii) \quad \mathcal{A}_f \approx \mathbb{B}^1$$

Proof: (i)

Define a curve I embedded in a planar face f bounded by a set of edges to be a *boundary curve* if $I \approx \mathbb{B}^1$ and its boundary points belong to ∂f. We will show that $\mathcal{E}_f$ is a single boundary curve.

$\mathcal{E}_f$ can have no closed component. We prove this by contradiction. Suppose $\mathcal{E}_f$ has a closed component. Then $\mathcal{E}_f$ cannot have any other curve component because it will contradict the fact that $\mathcal{E}_f$ is star-shaped w.r.t f (see Fig. 3.8(a)). But if $\mathcal{E}_f$ is a single closed component, then f is a complex face. Therefore, $\mathcal{E}_f$ cannot have any closed components.

This means that $\mathcal{E}_f$ consists of a set of boundary curves. We will show that there exists only one boundary curve. We will prove this by contradiction. Suppose that $\mathcal{E}_f$ has multiple boundary curves. The complex edge criterion ensures that a boundary



(a) (b)

Figure 3.8: Intersection Curves *Fig. (a) supports proof of Lemma* 2. *It shows a grid cell face* f *and the intersection of* $\mathcal{E}$ *with* f. *The figure shows the intersection curve consisting of two curve components, one of which is closed. As a result, it is not star-shaped. Fig. (b) shows the case where the curve has two boundary curves. This results in a face ambiguity.*

curve intersects exactly two edges of the face f. It also ensures that two boundary curves cannot intersect the same edge. Therefore, $\mathcal{E}_f$ can have at most two boundary curves, each intersecting two edges of f. However, this results in a face ambiguity (see Fig. 3.8(b)). Hence, $\mathcal{E}_f$ can have only a single boundary curve. This means $\mathcal{E}_f \approx \mathbb{B}^1$.

(ii) f intersects $\widetilde{\mathcal{E}}$. Because f is not complex, it must exhibit a sign change. By Property 2, we have $\mathcal{A}_f \approx \mathbb{B}^1$.

□

LEMMA 3 *Let* ϑ *be a voxel of a* $\mathcal{C}^{\square\bigstar}$*-cell such that* $\mathcal{E}_\vartheta \neq \emptyset$. *Then*

$$(i) \quad \mathcal{E}_\vartheta \approx \mathbb{B}^2$$
$$(ii) \quad \mathcal{A}_\vartheta \approx \mathbb{B}^2$$

Proof: (i) We prove that $\mathcal{E}_\vartheta$ cannot contain any closed component. Similar to the proof of Lemma 2, the presence of a closed component would imply that either the primitive is not star-shaped w.r.t. ϑ, or ϑ is a complex voxel.

We now prove that $\mathcal{E}_\vartheta$ has at most one surface component with a boundary and is connected. The boundary of each surface component corresponds to a boundary curve on the faces of the cell. From the result of Lemma 2, each face contains only one

boundary curve. Furthermore, the complex face and complex edge criteria preclude the boundary curve from intersecting one or two faces. Therefore, each boundary curve intersects at least three faces. Since each face can have at most one boundary curve, $\mathcal{E}_\vartheta$ cannot have more than two surface components. The only way there can be two surface components is if two diagonally opposite cell vertices are inside the primitive while the others are outside (see Fig. 3.6(d)). This is the case of a voxel ambiguity. Therefore, $\mathcal{E}_\vartheta$ has at most one surface component with a boundary and is connected.

To show that $\mathcal{E}_\vartheta$ is a topological disk, we show that $\mathcal{E}_\vartheta$ is a simply connected surface. Suppose that $\mathcal{E}_\vartheta$ is not simply connected. This means that $\mathcal{E}_\vartheta$ has two or more boundaries. The remainder of the proof is similar to the above argument. Existence of two boundaries results in a voxel ambiguity. Therefore, $\mathcal{E}_\vartheta$ is simply connected. This proves that $\mathcal{E}_\vartheta$ is a topological disk. This concludes the proof.

(ii) ϑ intersects $\widetilde{\mathcal{E}}$. Because ϑ is not complex, it must exhibit a sign change. By Property 2, we have $\mathcal{A}_\vartheta \approx \mathbb{B}^2$.

□

THEOREM 1 *If all the grid cells are $\mathcal{C}^{\square\bigstar}$-cells, then*

$$\mathcal{E} \approx \mathcal{A}$$

Proof: Lemmas 1, 2, and 3 establish that the restrictions of $\mathcal{E}$ and $\mathcal{A}$ to the edges, faces, and voxels of the grid are homeomorphic to each other. This fact can be used to construct a homeomorphism $\mathcal{H}$ between $\mathcal{E}$ and $\mathcal{A}$ inductively. We first define $\mathcal{H}$ on the edges, then faces, and finally voxels of the grid.

Edge Case: Consider an edge e of the grid that intersects $\mathcal{E}$. According to Lemma 1, e intersects both $\mathcal{E}$ and $\mathcal{A}$ once. Let $\mathbf{p}$ and $\mathbf{q}$ be the intersection of e with $\mathcal{E}$ and $\mathcal{A}$ respectively. We define $\mathcal{H}_e(\mathbf{p}) = \mathbf{q}$. In this manner, we define homeomorphism on the edges of the grid.

Face Case: Consider a face f of the grid such that $\mathcal{E}_f \neq \emptyset$. According to Lemma 2, both $\mathcal{E}_f \approx \mathbb{B}^1$ and $\mathcal{A}_f \approx \mathbb{B}^1$. Therefore, there exists a homeomorphism between $\mathcal{E}_f$ and $\mathcal{A}_f$. However, for our purpose, any homeomorphism does not suffice. We need to define a homeomorphism is consistent with the homeomorphisms defined on the edges of face f.

The complex edge criterion ensures that $\mathcal{E}_f$ must intersect two edges bounding the face. Let e_1 and e_2 be the two edges. $\mathcal{E}$ intersects e_1 and e_2 at intersection points $\mathbf{p_1}$

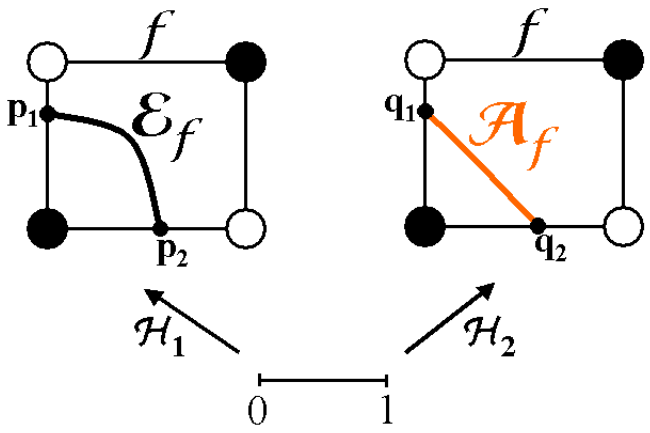


Figure 3.9: Face Homeomorphism: *This figure shows how to construct a homeomorphism between $\mathcal{E}_f$ and $\mathcal{A}_f$ on a face f. The homeomorphism is given by $\mathcal{H}_2 \circ \mathcal{H}_1^{-1}$.*

and $\mathbf{p_2}$, where $\mathbf{p_1} = \mathcal{E}_{e_1}$, and $\mathbf{p_2} = \mathcal{E}_{e_2}$. By Lemma 1, $\mathcal{A}$ also intersects e_1 and e_2 at intersection points $\mathbf{q_1}$ and $\mathbf{q_2}$, where $\mathbf{q_1} = \mathcal{A}_{e_1}$, and $\mathbf{q_2} = \mathcal{A}_{e_2}$. See Fig. 3.9.

Because $\mathcal{E}_f \approx \mathbb{B}^1$, there exists a homeomorphism $\mathcal{H}_1 : [0,1] \rightarrow \mathcal{E}_f$ such that $\mathcal{H}_1(0) = \mathbf{p_1}$ and $\mathcal{H}_1(1) = \mathbf{p_2}$. Similarly, there exists a homeomorphism $\mathcal{H}_2 : [0,1] \rightarrow \mathcal{A}_f$ such that $\mathcal{H}_2(0) = \mathbf{q_1}$ and $\mathcal{H}_2(1) = \mathbf{q_2}$. Define a *face homeomorphism* $\mathcal{H}_f : \mathcal{E}_f \rightarrow \mathcal{A}_f$ as $\mathcal{H}_f = \mathcal{H}_2 \circ \mathcal{H}_1^{-1}$ (Fig. 3.9).

Note that $\mathcal{H}_f$ is consistent with the homeomorphisms defined on the edges bounding the face. For example, we have

$$\begin{aligned} \mathcal{H}_f(\mathcal{E}_{e_1}) &= \mathcal{H}_f(\mathbf{p_1}) \\ &= \mathcal{H}_2 \circ \mathcal{H}_1^{-1}(\mathbf{p_1}) \\ &= \mathcal{H}_2(0) \\ &= \mathbf{q_1} \\ &= \mathcal{A}_{e_1} \end{aligned}$$

Voxel Case: Consider a voxel ϑ of the grid and let $\mathcal{E}_\vartheta \neq \emptyset$. Let $f_1, \ldots, f_k$ denote the cell faces that intersect $\mathcal{E}_\vartheta$. Our goal is to define a *voxel homeomorphism* $\mathcal{H}_\vartheta$ between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$ that is consistent with the face homeomorphisms $\mathcal{H}_{f_i}, i = 1, \ldots, k$.

$\mathcal{E}_\vartheta$ is bounded by a set $\mathcal{E}_{f_1}, \ldots, \mathcal{E}_{f_k}$ of boundary curves. $\mathcal{E}_\vartheta$ also intersects k edges of the cell. Let $\{\mathbf{p_1}, \ldots, \mathbf{p_k}\}$ denote the set of intersection points along the edges. According to Lemma 3, we have $\mathcal{E}_\vartheta \approx \mathbb{B}^2$. Therefore, there exists a homeomorphism

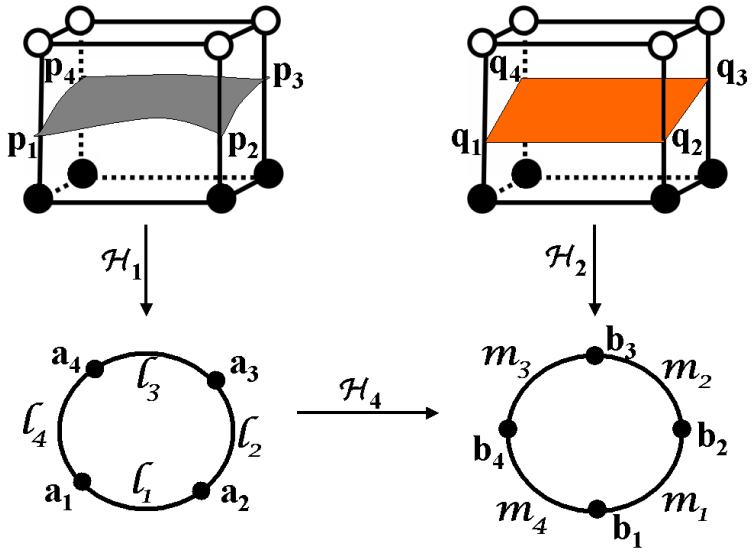


Figure 3.10: Voxel Homeomorphism: *This figure shows how to construct a homeomorphism between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$ in a voxel ϑ. The homeomorphism is given by $\mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1$.*

$\mathcal{H}_1 : \mathcal{E}_\vartheta \rightarrow \mathbb{B}^2$. Let $\mathcal{H}_1(\mathbf{p_i}) = \mathbf{a_i}$ and $\mathcal{H}_1(\mathcal{E}_{f_i}) = l_i, \ , \ i = 1, \ldots, k$. The points $\mathbf{a_1}, \ldots, \mathbf{a_k}$ belong to $\partial\mathbb{B}^2 = \mathbb{S}^1$, and partition $\mathbb{S}^1$ into k arcs $l_1, \ldots, l_k$. See Fig. 3.10.

Similar arguments hold for $\mathcal{A}_\vartheta$. $\mathcal{A}_\vartheta$ intersects the same set of cell faces and cell edges as $\mathcal{E}_\vartheta$. $\mathcal{A}_\vartheta$ is bounded by a set $\mathcal{A}_{f_1}, \ldots, \mathcal{A}_{f_k}$ of boundary curves. $\mathcal{A}_\vartheta$ intersects the cell edges giving rise to a set $\{\mathbf{q_1}, \ldots, \mathbf{q_k}\}$ of intersection points. Since $\mathcal{A}_\vartheta \approx \mathbb{B}^2$, there exists a homeomorphism $\mathcal{H}_2 : \mathcal{A}_\vartheta \rightarrow \mathbb{B}^2$. Let $\mathcal{H}_2(\mathbf{q_i}) = \mathbf{b_i}$ and $\mathcal{H}_2(\mathcal{A}_{f_i}) = m_i, \ , \ i = 1, \ldots, k$. The points $\mathbf{b_1}, \ldots, \mathbf{b_k}$ belong to $\mathbb{S}^1$, and partition $\mathbb{S}^1$ into k arcs $m_1, \ldots, m_k$ (Fig. 3.10).

It is possible to construct a homeomorphism $\mathcal{H}_3 : \mathbb{S}^1 \rightarrow \mathbb{S}^1$ that maps l_i to m_i. We can use an argument similar to the one in the **Face Case** to construct a homeomorphism $\mathcal{H}_3^i$ between l_i and m_i that maps a_i to b_i and a_{i+1} to b_{i+1}. $\mathcal{H}_3$ is then defined as an extension of $\mathcal{H}_3^i, i = 1, \ldots, k$; $\mathcal{H}_3(\mathbf{x}) = \mathcal{H}_3^i(\mathbf{x})$ if $\mathbf{x} \in l_i$

We can then extend $\mathcal{H}_3$ to define a homeomorphism $\mathcal{H}_4 : \mathbb{B}^2 \rightarrow \mathbb{B}^2$. $\mathcal{H}_4$ is defined as follows:

$$\begin{aligned} \mathcal{H}_4(0) &= 0 \\ \mathcal{H}_4(\mathbf{x}) &= \|\mathbf{x}\|_2 * \mathcal{H}_3(\mathbf{x}/\|\mathbf{x}\|_2), \mathbf{x} \in \mathbb{B}^2, \|\mathbf{x}\|_2 > 0 \end{aligned}$$

Since $\mathcal{H}_4$ is an extension of $\mathcal{H}_3$, it also maps l_i to m_i.

We are now ready to define a homeomorphism $\mathcal{H}_\vartheta$ between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$. Define

$\mathcal{H}_\vartheta = \mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1$. $\mathcal{H}_\vartheta$ is a homeomorphism between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$. See Fig. 3.10. Note that it is consistent with the homeomorphisms defined on the faces of the cell. In particular, we have

$$\begin{aligned}
\mathcal{H}_\vartheta(\mathcal{E}_{f_i}) &= \mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1(\mathcal{E}_{f_i}) \\
&= \mathcal{H}_2^{-1} \circ \mathcal{H}_4(l_i) \\
&= \mathcal{H}_2^{-1}(m_i) \\
&= \mathcal{A}_{f_i}
\end{aligned}$$

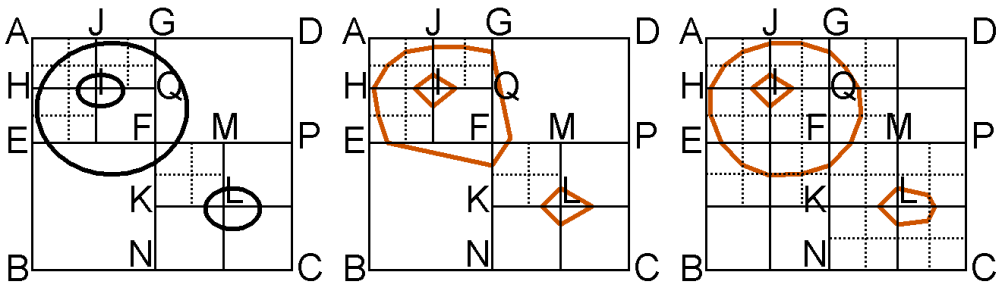
We define the homeomorphism $\mathcal{H} : \mathcal{E} \rightarrow \mathcal{A}$ in terms of the homemorphisms defined on the edges, faces, and voxels of the grid.

$$\begin{aligned}
\mathcal{H}(\mathbf{x}) &= \mathcal{H}_\vartheta(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to a voxel } \vartheta \\
&= \mathcal{H}_f(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to a face } f \\
&= \mathcal{H}_e(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to an edge } e
\end{aligned}$$

□

The above proof assumes that the reconstructed isosurface in two adjacent cells matches along the common face shared by the two cells. While the original Marching Cubes algorithm ensures this property for uniform grids, applying the algorithm to an adaptive grid violates the property, resulting in cracks in the reconstructed isosurface. Several extensions have been proposed to rectify this problem (Shekhar et al., 1996; Ju et al., 2002). We assume that one of these algorithms is used for reconstruction in our applications. We will address this issue in more detail in Sec. 3.9.

In the above proofs, we showed that $\mathcal{E}$ restricted to a cell is a topological disk in the right dimension. This topological disk property is then used to establish topological equivalence of $\mathcal{E}$ and $\mathcal{A}$. This property is similar to the *topological ball* property proposed by Edelsbrunner & Shah (Edelsbrunner and Shah, 1994). They used the topological ball property to ensure topology preservation in the context of Delaunay triangulation. While our approach shares the goal of topology preservation, it is different in that it is geared towards MC-like reconstruction methods. The novelty of our overall approach lies in the use of two simple criteria – complex cell and star-shaped criteria – to guarantee topology preserving MC-like reconstruction.



(a) 2D Example (b) Correct Topology (c) Bounded Geometric Error

Figure 3.11: Adaptive Subdivision: *This figure is a 2D illustration of our adaptive subdivision algorithm. Fig. (a) shows a volumetric grid generated by applying the sampling condition. Fig. (b) shows a topology preserving approximation obtained by applying an MC-like method to the volumetric grid. Fig. (c) shows an approximation with a bounded geometric error (see Sec. 3.8). Our algorithm performs adaptive subdivision until the isosurface within each cell is a topological disk. We ensure this condition by testing whether a cell is complex and if the isosurface is star-shaped with respect to the cell. In this figure, cell ABCD was subdivided because it corresponds to a complex voxel, cells AEFG and FNCP were subdivided because the isosurface within the cell was not star-shaped and FKLM was subdivided because of topological ambiguity. Edge IJ is complex; as a result, cells AHIJ and JIQG are subdivided.*

3.5 Topology Preserving Sampling

In this section, we provide adaptive subdivision criteria to generate a grid such that each grid cell is a $\mathcal{C}^{\square\bigstar}$-cell. Our sampling algorithm performs two tests on each grid cell. The first test checks if a cell satisfies $\mathcal{C}^{\square}$ using a *cell intersection query* (Sec. 3.5.1). The second test checks if a cell satisfies $\mathcal{C}^{\bigstar}$ using a *star-shaped query* (Sec. 3.5.2). If the grid cell fails to satisfy either of the two tests, the cell is subdivided and the two tests are recursively applied to the children cells. If the grid cell satisfies both the tests, the cell is returned as a leaf node in the octree grid. Fig. 3.11 shows a 2D illustration of the adaptive subdivision.

We first describe computational techniques for polyhedral models and their Boolean combinations, and later extend them to non-linear primitives. We also discuss details of the adaptive subdivision algorithm.

3.5.1 Cell Intersection Query

The objective of cell intersection query is to test if $\mathcal{E}$ intersects the cell. Specifically, we need to test if $\mathcal{E}$ intersects a voxel, a face, or an edge of a cell. We refer to these three tests collectively as cell intersection queries, and individually as voxel, face, and edge intersection query.

Interval Arithmetic

One technique for performing the cell intersection query is using interval arithmetic. Early work on interval arithmetic was done by Moore (Moore, 1966). Since then, it has been widely used in a large number of domains including computer graphics (Kalra and Barr, 1989; Mitchell, 1991; Snyder, 1992). Interval arithmetic is a general technique that is applicable to a wide variety of primitives. It is well suited to non-linear primitives. It also extends easily to higher dimensions. An overview of interval arithmetic is given in Appendix A.

Given a primitive $\mathcal{P}$ defined by an algebraic function $f : \mathbb{R}^3 \rightarrow \mathbb{R} = 0$, we can write an interval form $\overline{f}$ of the function. Then we can use $\overline{f}$ to answer the cell intersection query. We take advantage of the fact that each edge/face/voxel corresponds to a product of intervals.

Consider a voxel ϑ of an axis-aligned grid. ϑ corresponds to a product of intervals defined by two *diametrically opposite* vertices of the voxel. Let $v^i, i = 0, \ldots, 7$ denote the vertices of the voxel. Let $v^i = (v^i.x, v^i.y, v^i.z)$. Let $s^i = v^i.x + v^i.y + v^i.z$. Given a set $\{a_1, \ldots, a_n\}, a_i \in \mathbb{R}, i = 1, \ldots, n$, let $\arg\min_i a_i$ return k if $a_k = \min_i a_i$. $\arg\max$ is defined similarly. Let

$$\mathbf{p} = v^k \quad \text{where} \quad k = \arg \min_i s^i$$
$$\mathbf{q} = v^l \quad \text{where} \quad l = \arg \max_i s^i$$

The voxel ϑ corresponds to a product of intervals $I_\vartheta = [\mathbf{p}.x, \mathbf{q}.x] \times [\mathbf{p}.y, \mathbf{q}.y] \times [\mathbf{p}.z, \mathbf{q}.z]$.

We can test if the boundary of $\mathcal{P}$ intersects ϑ by evaluating $\overline{f}$ on I_ϑ. We use the following fact:

$$\vartheta \text{ intersects the boundary of } \mathcal{P} \quad \text{if} \quad 0 \in \overline{f}(I_\vartheta)$$

Interval arithmetic can also handle voxels that are not axis-aligned by applying a rigid transformation to I_ϑ before evaluating $\overline{f}$.

The conservativeness of interval arithmetic makes the above intersection test conser-

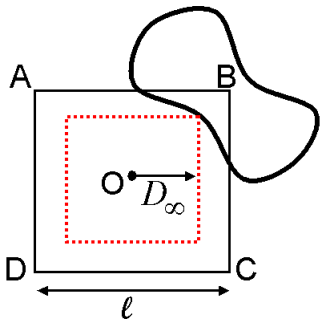


Figure 3.12: Voxel Intersection Test: *We use the l_∞ distance (indicated by the dotted red cube) to perform a voxel-intersection test. The isosurface intersects the voxel if and only if l_∞ distance between the center of the voxel ($\mathbf{o}$) and the isosurface is less than half the voxel size.*

vative: While the test is guaranteed to be satisfied by a voxel that intersects the surface, it may also be satisfied by some voxels that do not actually intersect the surface. This does not, however, affect the correctness of our algorithm.

Max-Norm Distance Computation

Another technique to answer the cell intersection query relies on max-norm distance computation (Varadhan et al., 2003a). It is efficient in practice, and is well suited to polyhedral and low degree non-linear primitives. Under the max-norm, the distance between two points $\mathbf{p}$ and $\mathbf{q}$ (in 3 dimensions) is denoted as $D_\infty(\mathbf{p}, \mathbf{q})$ and is defined as

$$D_\infty(\mathbf{p}, \mathbf{q}) = \max_i |p_i - q_i|, \qquad i = 1, 2, \ldots, 3$$

We can extend this definition for distance between a point $\mathbf{p}$ and a set $\mathcal{Q}$ in $\mathbb{R}^3$.

$$D_\infty(\mathbf{p}, \mathcal{Q}) = \min_{\mathbf{q} \in \mathcal{Q}} D_\infty(\mathbf{p}, \mathbf{q}) \tag{3.1}$$

The iso-distance ball, i.e., the set of points at a constant distance from the origin, under max-norm is a cube; so it is a natural metric for cubical cells. The above definition can be extended to cuboids by defining a suitably weighted version of the max-norm along different dimensions.

For a closed primitive, we use a signed version of the distance. Let $\mathcal{Q}$ denote the boundary of the closed primitive and let $\widetilde{\mathcal{Q}}$ be the solid bounded by $\mathcal{Q}$. We define the

signed max-norm distance $D^s_\infty(\mathbf{p}, \mathcal{Q})$ as follows:

$$D^s_\infty(\mathbf{p}, \mathcal{Q}) = sign(\mathbf{p}, \mathcal{Q}) * \min_{\mathbf{q} \in \mathcal{Q}} D_\infty(\mathbf{p}, \mathbf{q}) \tag{3.2}$$

where

$$\begin{aligned} sign(\mathbf{p}, \mathcal{Q}) &= -1 \quad if\ \mathbf{p} \in \widetilde{\mathcal{Q}} \\ &= 1 \quad otherwise \end{aligned}$$

We use max-norm distance computation to check whether $\mathcal{E}$ intersects a voxel of the cell. We use the fact that a voxel intersects the surface if and only if its unsigned three-dimensional max-norm (l_∞) distance from the center of the voxel is less than half the size of the cell. This is shown in Fig. 3.12. It is formally stated as the following lemma:

LEMMA 4 Voxel Intersection Test *Given a voxel* ϑ*,*

$$\mathcal{E}_\vartheta \neq \emptyset \quad \Leftrightarrow \quad |D^s_\infty(\mathbf{o}, \mathcal{E})| = D_\infty(\mathbf{o}, \mathcal{E}) \leq l/2$$

where $\mathbf{o}$ *and* l *are the center and width of* ϑ *respectively.*

Similarly, the face intersection test for a face f can be performed by computing two-dimensional max-norm distance between the center of f and $\mathcal{E}_f$. In this manner, we can use max-norm distance to perform the cell intersection query. We can efficiently compute max-norm distance for a wide variety of geometric primitives (Varadhan et al., 2003a).

Boolean Expression

When $\mathcal{E}$ is defined by a Boolean expression involving a number of primitives, it is difficult to compute the signed distance $D^s_\infty(\mathbf{p}, \mathcal{E})$ because we do not have an explicit boundary representation of $\mathcal{E}$. Instead, we compute an estimate $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$ of the signed max-norm distance.

The Boolean operations on the primitives define a solid whose boundary is $\mathcal{E}$. Let $\widetilde{\mathcal{E}}$ denote this solid. We perform a case analysis on $\widetilde{\mathcal{E}}$ to define $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$. We note that $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$ is an estimate of the signed max-norm distance; hence it is also signed.

1. $\widetilde{\mathcal{E}}$ is a primitive solid $\widetilde{\mathcal{P}}$. Let $\mathcal{P}$ denote the boundary of $\widetilde{\mathcal{P}}$. We have

$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = D^s_\infty(\mathbf{p}, \mathcal{P})$$

2. $\widetilde{\mathcal{E}}$ is a union of two solids: $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$.

$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = \min(\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1), \widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_2))$$

3. $\widetilde{\mathcal{E}}$ is an intersection of two solids: $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2$.

$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = \max(\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1), \widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_2))$$

4. $\widetilde{\mathcal{E}}$ is the complement of a solid: $\widetilde{\mathcal{E}} = \overline{\widetilde{\mathcal{E}}_1}$.

$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = -\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1)$$

We use $\widetilde{D}^s_\infty$ to perform the voxel intersection test. $\widetilde{D}^s_\infty$ may not be equal to the actual signed distance D^s_∞ at some points. However, its absolute value is always less than the absolute value of D^s_∞. This is stated in the following lemma.

LEMMA 5

$$|\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})| < |D^s_\infty(\mathbf{p}, \mathcal{E})|$$

We skip the proof. It follows directly from the definition of $\widetilde{D}^s_\infty$.

□

While the above property makes the voxel intersection test conservative, it preserves its correctness: If a voxel intersects $\mathcal{E}$, we take it into account. On the other hand, we may also take into account some voxels that do not intersect $\mathcal{E}$. While this may result in some unnecessary computation, it does not affect the correctness of the algorithm.

A similar technique can also be used to perform interval arithmetic on Boolean combinations of primitives. Given a voxel ϑ, we first apply interval arithmetic to each of the primitives, and then perform min/max operations on the resulting intervals. This produces an interval $I_{\mathcal{E},\vartheta}$ for $\mathcal{E}$. We then check if $0 \in I_{\mathcal{E},\vartheta}$ to test whether $\mathcal{E}$ intersects ϑ.

Edge Intersection Query

We use directed distances (Kobbelt et al., 2001) to answer the edge intersection query. The directed distance between a point $\mathbf{p}$ and a primitive $\mathcal{Q}$ along a unit vector

$\vec{\mathbf{v}}$ is the distance to the closest point on the primitive along $\vec{\mathbf{v}}$. It is denoted as $D_{\vec{v}}(\mathbf{p}, \mathcal{Q})$ and is defined as:

$$\begin{aligned} S &= \{\mathbf{q} \in \mathcal{Q} \mid \exists \lambda > 0 \text{ such that } \mathbf{q} - \mathbf{p} = \lambda \vec{\mathbf{v}}\} && (3.3) \\ D_{\vec{\mathbf{v}}}(\mathbf{p}, \mathcal{Q}) &= \min\{d(\mathbf{p}, \mathbf{q}) \mid \mathbf{q} \in S\} \quad \text{if } S \neq \emptyset && (3.4) \\ &= \infty \quad \text{otherwise} \end{aligned}$$

Our edge intersection test is based on the following property: If an edge $\mathbf{ab}$ intersects $\mathcal{E}$, then the directed distance at $\mathbf{a}$ along the direction vector $\vec{\mathbf{ab}}$ is less than the length of the vector $\vec{\mathbf{ab}}$. Based on this fact, we define an edge $\mathbf{ab}$ to be intersecting if

$$D_{\overrightarrow{\mathbf{ab}}}(\mathbf{a}, \mathcal{E}) < d(\mathbf{a}, \mathbf{b}).$$

Kobbelt et al. (Kobbelt et al., 2001) have presented computational techniques for computing directed distance for a wide variety of geometric primitives. If $\mathcal{E}$ is defined as a Boolean expression, then we can compute a conservative estimate $\widetilde{D}_{\vec{v}}$ of the directed distance in a manner similar to the max-norm distance.

3.5.2 Star-shaped Query

The computation of the exact kernel of an orientable polyhedral primitive reduces to the intersection of halfspaces determined by the tangent planes of the faces of the primitive. Using the point-hyperplane duality, this is equivalent to convex hull computation. In $\mathbb{R}^3$, for a polyhedral surface with n facets, this can be performed in $O(n \log n)$ (de Berg et al., 2000). However, to test if a primitive is star-shaped, it suffices to check if the kernel is empty or not. We refer to this test as the *star-shaped query*.

For the sake of simplicity, we first consider the star-shaped query for polyhedral primitives. We discuss extension to non-linear primitives in Section 3.5.4. For a polyhedral primitive, testing for a non-empty kernel reduces to linear programming (LP) (Schrijver, 1998). If $\boldsymbol{p}$ is a point belonging to the kernel, then each face of the polyhedron with centroid $\boldsymbol{c}$ and outward normal $\boldsymbol{n}$ defines the linear constraint $\boldsymbol{n} \cdot (\boldsymbol{c} - \boldsymbol{p}) > 0$ on $\boldsymbol{p}$. As a result, the kernel is non-empty if the set of constraints admits a feasible solution for $\boldsymbol{p}$. In fixed dimensions, LPs can be solved in linear time, and a number of efficient public domain implementations are available (GLPK, 2003; QSOPT, 2005).

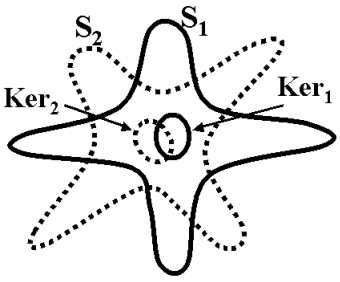


Figure 3.13: Star-shaped test for Boolean Combination: *If two star-shaped primitives $\mathcal{S}_1$ and $\mathcal{S}_2$ are star-shaped w.r.t a common point, i.e. $Kernel(\mathcal{S}_1)$ overlaps with $Kernel(\mathcal{S}_2)$, then both $\mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_1 \cap \mathcal{S}_2$ are star-shaped.*

Boolean Expression

When $\mathcal{E}$ is defined by a Boolean expression involving a number of primitives, a sufficient condition for the star-shapedness of $\mathcal{E}$ is that the intersection of all the primitive kernels is non-empty. If $\mathcal{S}_1$ and $\mathcal{S}_2$ are two star-shaped primitives with a common guard, then $\mathcal{S}_1 \odot \mathcal{S}_2$ is also star-shaped where $\odot$ denotes a Boolean operation such as union and intersection. This is because

$$Kernel(\mathcal{S}_1) \cap Kernel(\mathcal{S}_2) \subseteq Kernel(\mathcal{S}_1 \odot \mathcal{S}_2)$$

See Fig. 3.13.

For polyhedral primitives, we check for the above condition by combining the linear constraints defined by the individual primitives and testing for feasibility by solving the resulting LP. The difference operation can be rewritten as an intersection by inverting the linear constraints of the negated primitive.

We note here that the above condition is sufficient, but not necessary. We do not perform an exact test as we do not have an explicit representation of $\mathcal{E}$.

3.5.3 Adaptive Subdivision Algorithm

We start with a single grid cell that is guaranteed to bound $\mathcal{E}$. We perform two tests, *complex cell test* and *star-shaped test*, to decide whether to subdivide a grid cell. We now describe each of these tests.

Algorithm 1 `Adaptive_Subdivision`(C)
Input: Grid cell C associated with a Boolean expression $\widetilde{\mathcal{E}}_C$.
Output: An adaptive subdivision of C such that any MC-like algorithm generates topologically correct output.

if C can be disregarded by *cell culling* (Sec. 3.10.1) **then**
  return
end if
if C satisfies *complex cell* **and** *star-shaped* tests **then**
  return
end if
Subdivide C into children cells $C_i, i = 1, \ldots, k$
for $i = 1$ to k **do**
  Perform *expression simplification* w.r.t C_i: $\widetilde{\mathcal{E}}_C \xrightarrow{C_i} \widetilde{\mathcal{E}}_{C_i}$ (Sec. 3.10.2)
  Adaptive_Subdivision(C_i)
end for

Complex Cell Test

To check whether a cell is complex, we perform the following tests:

- **Complex Voxel/Face**: We use the cell intersection query to check whether $\mathcal{E}$ intersects a voxel or face of the cell. If $\mathcal{E}$ intersects the voxel (face), then we determine if the voxel (face) is complex by checking for a sign change at the cell vertices. If $\mathcal{E}$ does not intersect the voxel (face), then the voxel (face) is not considered complex.

- **Complex Edge**: We use directed distances (Kobbelt et al., 2001) to test if an edge is complex. An edge is complex if the sum of the directed distances (along the edge) from the two endpoints of the edge is less than the edge length.

- **Ambiguity**: We use the signs at the grid vertices to resolve cases corresponding to face and voxel ambiguity (see Fig. 3.8(b) and Fig. 3.6(d)).

If any of these tests results in the affirmative, the cell is complex, and we subdivide it and apply the algorithm recursively to the new cells. See Fig. 3.11. Alg. 1 shows the pseudo-code of our algorithm.

Star-shaped Test

Linear programming (LP) is used to test for the star-shapedness of a polyhedral primitive. As described earlier in this section, $\mathcal{E}$ described by a single primitive or implicitly

by a collection of primitives can be conservatively checked for the star-shaped property. We need to perform two tests on each cell C – (a) star-shaped w.r.t voxel of C, and (b) star-shaped w.r.t. each face of C.

For each polyhedral primitive, we consider only those faces of the polyhedron that intersect the voxel. This set of faces defines the constraints for an LP in $\mathbb{R}^3$, whose solution answers the test (a). For each face of the cell, we consider those faces of the polyhedron that intersect it. These faces result in a collection of piecewise linear segments on the face of the cell. Solving a similar LP defined by these linear curves in $\mathbb{R}^2$ answers test (b). When there are multiple primitives intersecting the cell, we combine the linear constraints arising from each primitive and then solve the resulting LP. Since we are only dealing with linear programs in two and three dimensions, a dual formulation of constraints and objective function is much more efficient in practice. We use such a formulation to perform the star-shaped test.

If either of these tests turns out to be negative, we subdivide the cell. Fig. 3.11 illustrates the working of the algorithm on a 2D example.

3.5.4 Star-shaped Query for Non-linear Primitives

In the previous subsections, we presented methods for performing the complex cell and star-shaped tests. We used interval arithmetic and max-norm distance computation for the complex cell test. Both of these methods are applicable to a wide class of geometric primitives. On the other hand, our method for the star-shaped query using linear programming was restricted to polyhedral primitives.

In this section, we describe a method to extend this computation to non-linear primitives. This method was proposed by Shankar Krishnan, and is described in (Varadhan et al., 2004).

It is well-known that the kernel for non-linear closed primitives like parametric and algebraic surfaces can be computed by finding the intersection of the tangent planes at points on the surface with zero Gaussian curvature (Seong et al., 2003). However, this approach involves solving a system of high degree equations and curve tracing, which is computationally intensive. Therefore, the applicability of this approach is limited.

We describe a method that avoids exact kernel computation. As previously observed, the star-shaped query reduces to testing whether the kernel is empty or not. Therefore, all we need is a point that is witness to the actual kernel. We describe a simple method to conservatively perform this test. This method proceeds by selecting

a candidate point that lies in the interior of an approximate kernel of a discretized version of the primitive. This point is computed by linear programming. We check if the candidate point belongs to the kernel of the curved primitive by using interval arithmetic. In this section, we provide the details of the candidate point selection and kernel membership test.

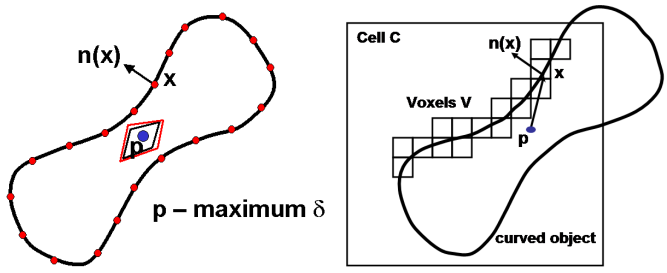
Candidate Point Selection

We start with a discretization of the curved primitive. We compute a set of sample points $\mathbf{p_i}, i = 1, \ldots, n$ on the curved surface. Let the unit outward normal at $\mathbf{p_i}$ be $\mathbf{n_i}$. These sample points define a set of linear constraints on the kernel. The constraints are of the form $\mathbf{n_i}^T\mathbf{x} \leq d_i = \mathbf{n_i}^T\mathbf{p_i}, i = 1, \ldots, n$. We add a new slack variable δ to each of the constraints - $\mathbf{n_i}^T\mathbf{x} + \delta \leq d_i$ subject to $\delta > 0$. We set the objective function to maximize δ. Intuitively, the modified constraints can be viewed as a family of parallel planes (planes moving away from the normal) defining a kernel parameterized by δ. As δ increases, the kernel shrinks. If the original constraints define a valid kernel, the maximum value of δ for the new constraints is reached at a point that is maximally interior in the kernel (see Fig. 3.14(a)). Further, the maximum δ value also gives a lower bound on the volume of the approximate kernel, and hence an indication of the existence of a non-empty exact kernel.

Kernel Membership Test using Interval Arithmetic

The candidate point computed above will lie inside the exact kernel (if non-empty) provided we computed a sufficient number of sample points ($\mathbf{p_i}$'s) on the curved surface. In general, we do not know how many such points are needed; so we choose a fixed number of points. To ensure correctness, we need to check if the candidate point is actually a witness to the exact kernel. Such a witness $\mathbf{p}$ would satisfy $\mathbf{n}(\mathbf{x})^T(\mathbf{x} - \mathbf{p}) > 0$, for all points $\mathbf{x}$ on the primitive where $\mathbf{n}(\mathbf{x})$ is the normal to the surface at $\mathbf{x}$. For an algebraic surface $f(x, y, z) = 0$, the expression becomes $\bigtriangledown f^T(\mathbf{x} - \mathbf{p})$. We can derive a similar expression for parametric surfaces.

Consider a closed algebraic surface $f(\mathbf{x}) = 0$ and a point $\mathbf{p}$. The expression defining the kernel membership test is $\Gamma(f, \mathbf{p}) \ : \ \bigtriangledown f^T(\mathbf{x} - \mathbf{p})$ subject to the condition that $f(\mathbf{x}) = 0$. Consider the case of a quadric surface, where $f()$ is given by $\mathbf{x}^T\mathbf{A}\mathbf{x}$. In this case, the expression for the gradient is $\mathbf{A}\mathbf{x}$. Therefore, $\Gamma(f, \mathbf{p}) = (\mathbf{x} - \mathbf{p})^T\mathbf{A}\mathbf{x} = -\mathbf{p}^T\mathbf{A}\mathbf{x}$, since $\mathbf{x}^T\mathbf{A}\mathbf{x} = 0$. In this special case, the expression to be tested is a linear



(a) (b)

Figure 3.14: Star-shaped test on curved primitives: *Fig. (a) shows the candidate point selection. The black curve is the curved primitive, while the red points form a discretized approximation. The same color scheme is used for the respective kernels. The linear program is set up so that if an approximate kernel exists, the objective function δ is optimized near its center. Fig. (b) shows the kernel membership test. After choosing candidate point* $\mathbf{p}$*, we use interval arithmetic to perform the test. Given a cell* C*, we compute an initial voxelization* V *of the primitive and use the intervals generated by each of the voxels inside* C *in the interval arithmetic step. Since the voxels closely approximate the primitive, the performance of the membership test is significantly improved.*

expression.

Given such an expression and an axis-aligned cell, we need to verify if it is positive inside the cell. We use interval arithmetic to perform this test reliably on the interval determined by the cell. If the expression turns out to be positive inside the cell, $\mathbf{p}$ is a witness to the exact kernel and we stop subdivision. Otherwise, the cell is subdivided and the tests are repeated on each child cell.

The above approach is conservative and may result in some unnecessary subdivision. The main benefit of this method is that it is similar in flavor to the test for polyhedral primitives, and this makes it efficient. This approach requires an explicit expression for the normal field of the surface. This is a reasonable assumption because such expressions are available for the class of surfaces representable in algebraic or rational parametric form (Manocha and Canny, 1992).

The performance of interval arithmetic depends on the degree of the expression and the tightness of the interval used. If the cells are big, interval arithmetic can return false negatives. This is primarily due to the fact that we are unable to impose the restriction that $\mathbf{x}$ should satisfy $f(\mathbf{x}) = 0$. Furthermore, if $\mathbf{p}$ lies inside the interval

box on which we perform the evaluation using interval arithmetic, $\Gamma(f, \mathbf{p})$ will never be positive. We alleviate these problems as follows:

- We do not impose the restriction that $\mathbf{p}$ be inside the grid cell. This gives us candidate points that are usually far away from the grid cell.
- Along with a discretization of the original primitive, we can also precompute a voxelization. Given a grid cell, we find voxels that intersect the cell and test for $\Gamma(f, \mathbf{p})$'s sign in each of the voxel intervals (see Fig. 3.14(b)). If all the tests return a positive sign, the point is a valid witness to the kernel. The motivation to perform the voxelization is to generate intervals that are as close to the original surface as possible. This also serves the purpose of significantly improving the performance of the interval arithmetic step.

The combination of these two steps eliminates most of the problems mentioned earlier, and avoids unnecessary subdivision.

3.6 Analysis

In this section, we analyze the behavior of our adaptive subdivision algorithm and provide a sufficient condition for its termination. We show that under certain conditions, once a cell becomes smaller than a certain size, it will eventually satisfy both complex cell and star-shaped criteria. Our analysis assumes that $\mathcal{E}$ is a smooth surface – twice-differentiable manifold. This assumption may not hold when $\mathcal{E}$ is defined using Boolean operations. We note that we make this assumption only to simplify the analysis of the algorithm; the algorithm itself does not make this assumption.

We perform the analysis in two stages. In the first stage, we analyze both complex cell and star-shaped criteria in terms of the Gauss map of $\mathcal{E}$ within the cell. The Gauss map $\mathbb{G}$ of a smooth surface $\mathcal{S}$ in $\mathbb{R}^3$ is a set-valued function from $\mathcal{S}$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p} \in \mathcal{S}$ the outward unit normal to $\mathcal{S}$ at $\mathbf{p}$. We use the Gauss map of $\mathcal{E}$ to provide sufficient conditions for when a cell will satisfy both the complex cell and star-shaped criteria. There are two separate conditions – one for complex cell criterion and another one for star-shaped criterion.

In the second stage, we relate the Gauss map conditions to the notion of *local feature size* (LFS), proposed by Amenta and Bern (Amenta and Bern, 1998). The local feature size $\mathrm{LFS}(\mathbf{p})$ at a point $\mathbf{p}$ on a surface $\mathcal{S}$ is defined as the least distance of $\mathbf{p}$ to the *medial*

axis of $\mathcal{S}$ (Fig. 3.17). The medial axis of $\mathcal{S}$ is the set of points with more than one closest point on $\mathcal{S}$. Amenta and Bern used the LFS to design a sampling condition for topology preserving reconstruction using Delaunay triangulation. They showed that it suffices to choose a set of samples on $\mathcal{S}$ such that every point $\mathbf{p} \in \mathcal{S}$ has a sample at a distance less than a fraction of LFS($\mathbf{p}$).

We extend the above definition to define the LFS of a grid cell. We then show that if a cell is smaller than a certain fraction of its LFS, then it will meet the Gauss map conditions thus satisfying both the complex cell and star-shaped criteria. This provides a bound on the size of a cell relative to its LFS. In the absence of degeneracies, the adaptive subdivision algorithm will terminate once all the cells become smaller than a fraction of their respective LFS. We use the LFS to also characterize some of the degenerate cases of our algorithm. This will be discussed in Sec. 3.7.

3.6.1 Preliminaries

We use the following notation in this section. $d(\mathbf{p}, \mathbf{q})$ denotes the Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$. $\|\vec{\mathbf{u}}\|$ denotes the length of a vector $\vec{\mathbf{u}}$. $\angle\mathbf{m}, \mathbf{n}$ denotes the angle between two vectors $\mathbf{m}$ and $\mathbf{n}$.

Let $\mathbf{o}$ be the origin of $\mathbb{R}^d$ where $d = 2, 3$. Let $\mathbf{x}_i^+, \mathbf{x}_i^-, i = 1, \ldots, d$ denote the principal directions of $\mathbb{R}^d$. $\mathbf{x}_i^+$ is a unit vector in $\mathbb{R}^d$ whose ith component is 1 and rest of the components are 0. $\mathbf{x}_i^-$ is equal to $-\mathbf{x}_i^+$. By a *principal hemisphere*, we mean a hemisphere whose axis is along one of the principal directions. For example, a principal hemisphere with $\mathbf{x}_1+$ axis in $\mathbb{R}^3$ is defined as:

$$\{\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \mid \|\mathbf{ox}\| = 1, x \geq 0\}$$

A right circular cone with an *axis* $\vec{\mathbf{u}}$ and a *half-angle* θ is defined as the set of points $\{\mathbf{x} \in \mathbb{R}^d \mid \angle\vec{\mathbf{u}}, \mathbf{ox} = \theta\}$.

Given a cell C, we define a Gauss map of restrictions of $\mathcal{E}$ to the voxel and faces of C.

- For a voxel ϑ of C, define $\mathbb{G}_\vartheta$ as a set valued function from $\mathcal{E}_\vartheta$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p} \in \mathcal{E}_\vartheta$ the outward unit normal to $\mathcal{E}_\vartheta$ at $\mathbf{p}$.

- For a face f of a cell, we consider the restriction of $\mathcal{E}$ to the plane Π_f containing f. $\mathcal{E}_f$ is treated like a curve in $\mathbb{R}^2$. We use a 2D definition of Gauss map. Define

$\mathbb{G}_f$ as a set valued function from $\mathcal{E}_f$ to the unit circle $\mathbb{S}^1$, which assigns to each point $\mathbf{p} \in \mathcal{E}_f$ the outward unit normal (defined in 2D) to $\mathcal{E}_f$ at $\mathbf{p}$.

We use the term Gauss map to also refer to the image of the Gauss map.

3.6.2 Gauss Map Condition for Complex Cell Criterion

In Sec. 3.4, we had defined a cell to be complex if it has a complex voxel, a complex face, a complex edge, or an ambiguous sign configuration. There is some redundancy in this definition: a complex voxel is allowed to have complex faces or complex edges. We now provide an equivalent definition without the redundancy. We refer to a voxel (face) as *strongly complex* if it is complex and none of its faces (edges) intersect $\mathcal{E}$. Unlike a complex voxel, a strongly complex voxel cannot have a complex face or a complex edge. A strongly complex voxel always contains a closed component of $\mathcal{E}$ in its interior. We define a cell to be complex if it has a *strongly complex* voxel, a *strongly complex* face, a complex edge, or an ambiguous sign configuration. This definition of a complex cell is equivalent to the one presented in Sec. 3.4.

We now present a Gauss map condition for when a cell satisfies the complex cell criterion. First, we introduce a definition.

DEFINITION 5 *Let c be a voxel/face of a cell. Let $\vec{\mathbf{u}}$ be a unit vector and θ be a value such that $0 < \theta \leq \pi/2$.*

1. *We say c* is normal-bounded w.r.t $(\vec{\mathbf{u}}, \theta)$ *if*

   (a) $\mathcal{E}_c = \emptyset$ *or*

   (b) $\angle \vec{\mathbf{u}}, \mathbf{n} < \theta \ \forall \mathbf{n} \in \mathbb{G}_c(\mathcal{E}_c)$

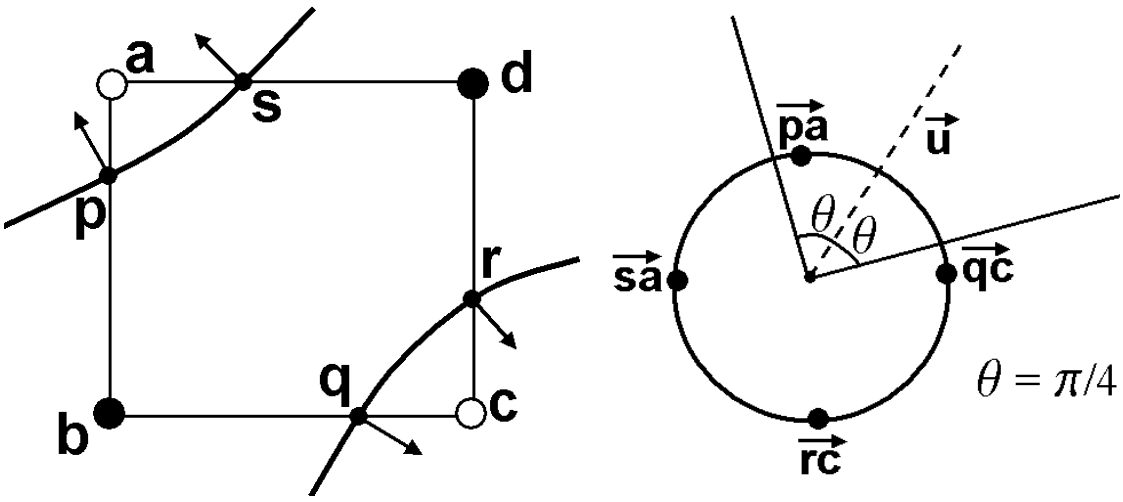
   This means that the Gauss map $\mathbb{G}_c(\mathcal{E}_c)$ lies within a right circular cone whose axis is $\vec{\mathbf{u}}$ and whose half-angle is θ.

2. *We say c* is normal-bounded by θ *if*

   (a) $\mathcal{E}_c = \emptyset$ *or*

   (b) *For any two vectors $\mathbf{n}_1, \mathbf{n}_2 \in \mathbb{G}_c(\mathcal{E}_c)$, we have $\angle \mathbf{n}_1, \mathbf{n}_2 < \theta$.*

THEOREM 2 *Consider a cell C. If the following conditions hold:*



(a) Topological Ambiguity (b) Supports Proof of Theorem 2

Figure 3.15: *This figure supports the proof of Theorem 2. Fig. (a) shows a face with ambiguity. In Fig. (b), the unit vectors along* **pa**, **sa**, **qc** *and* **rc** *have been mapped onto the the unit circle. The figure shows a cone with an axis* $\vec{\mathbf{u}}$ *and half-angle* θ*. We show that it is not possible for the Gauss map of the curve to lie within this cone.*

1. *The voxel of* C *is* normal-bounded *w.r.t* $(\vec{\mathbf{u}}, arcsin(1/\sqrt{(}3)))$ *for some unit vector* $\vec{\mathbf{u}} \in \mathbb{S}^2$.

2. *Every face of* C *is* normal-bounded *w.r.t* $(\vec{\mathbf{u}}, \pi/4)$ *for some unit vector* $\vec{\mathbf{u}} \in \mathbb{S}^1$.

3. *No edge* e *of* C *is complex.*

then C *is not complex.*

Proof: Let c be a voxel/face of C satisfying (1) or (2). We show it cannot be strongly complex and cannot have an ambiguity. Therefore, C cannot be complex.

Consider a face f satisfying (2). If f is strongly complex, then there exists a closed component of $\mathcal{E}_f$ within f. Therefore $\mathbb{G}_f(\mathcal{E}_f)$ would span the entire circle $\mathbb{S}^1$. This contradicts the fact that f is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$. Therefore, f cannot be strongly complex.

We now prove that f cannot have a face ambiguity. Let the vertices of f be **a**, **b**, **c**, and **d**, as shown in Fig. 3.15(a). Suppose f has an ambiguity. Without loss of generality, assume that **a** and **c** have a positive sign, while **b** and **d** have a negative

sign. Each of the four edges intersect $\mathcal{E}$. Let the intersection points on **ab**, **bc**, **cd**, **da** be **p**, **q**, **r**, **s** respectively. If an edge has more than one intersection point, then choose the one that is closest to the positive endpoint of the edge. Let $\mathbf{n_x}$ denote the unit normal to $\mathcal{E}_f$ at $\mathbf{x} \in \mathcal{E}_f$. Assume that the unit normal points "outwards", i.e., towards a point with a positive sign.

Since f is ambiguous, it has two boundary curves. Without loss of generality, assume **p** and **s** belong to one of the boundary curves, while **q** and **r** belong to the other. See Fig. 3.15(a).

Because **a** and **c** have a positive sign and the normals to $\mathcal{E}_f$ point outwards, the normals have to lie within a range specified by the following:

$$\begin{aligned} \angle\mathbf{n_p}, \vec{\mathbf{pa}} < \pi/2 \quad \wedge \quad \angle\mathbf{n_s}, \vec{\mathbf{sa}} < \pi/2 \\ \angle\mathbf{n_q}, \vec{\mathbf{qc}} < \pi/2 \quad \wedge \quad \angle\mathbf{n_r}, \vec{\mathbf{rc}} < \pi/2 \end{aligned} \tag{3.5}$$

where $\vec{\mathbf{pa}}, \vec{\mathbf{sa}}, \vec{\mathbf{qc}}, \vec{\mathbf{rc}}$ denote the unit vectors along $\mathbf{pa}, \mathbf{sa}, \mathbf{qc}, \mathbf{rc}$ respectively. Under the Gauss map $\mathbb{G}_f(\mathcal{E}_f)$, these unit vectors will map to *extremal points* in $\mathbb{S}^1$, i.e. points belonging to the set $\{(1,0),(0,1),(-1,0),(0,-1)\}$. See Fig. 3.15(b).

Since f is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$, $\mathbb{G}_f(\mathcal{E}_f)$ lies within a right circular cone C centered at the origin and with a half-angle $\pi/4$. The portion of $\mathbb{S}^1$ that lies within such a cone is always a subset of a principal hemisphere. Let $\mathbb{H}_1$ denote such a principal hemisphere. Then we have $\mathbb{G}_f(\mathcal{E}_f) \subseteq \mathbb{H}_1$. Furthermore, $\mathbb{H}_2 = \mathbb{S}^1 \backslash \mathbb{H}_1$ is another principal hemisphere. The apex of $\mathbb{H}_2$ is an extremal point in $\mathbb{S}^1$ ($\vec{\mathbf{rc}}$ in Fig. 3.15(b)). This extremal point corresponds to a unit vector. Let us denote it as $\vec{\mathbf{v}}$. Since $\mathbb{G}_f(\mathcal{E}_f) \subseteq \mathbb{H}_1$, the angle between $\vec{\mathbf{v}}$ and any vector in $\mathbb{G}_f(\mathcal{E}_f)$ is greater than $\pi/2$. In particular, the angle between $\vec{\mathbf{v}}$ and each of $\mathbf{n_p}, \mathbf{n_q}, \mathbf{n_r}, \mathbf{n_s}$ is greater than $\pi/2$. This contradicts Equation 3.5.

We can use a similar argument to show that the voxel ϑ of C is not strongly complex and does not have an ambiguity. We can show that if ϑ is normal-bounded w.r.t $(\vec{\mathbf{u}}, arcsin(1/\sqrt{(3)}))$, then $\mathbb{G}_\vartheta(\mathcal{E}_\vartheta)$ is always a subset of a principal hemisphere. This fact can then be used to prove the result.

□

3.6.3 Gauss Map Condition for Star-shaped Criterion

We now present a Gauss map condition for when $\mathcal{E}$ is star-shaped w.r.t a voxel. We show that a voxel ϑ will satisfy the star-shaped criterion if there exists a unit vector $\vec{\mathbf{u}}$

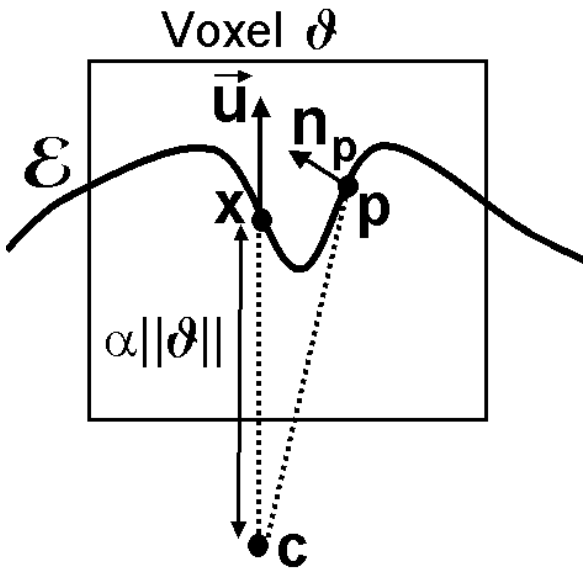


Figure 3.16: Condition for Star-shaped Criterion: *This figure supports the proof of Theorem 3. If there exists a unit vector* $\vec{\mathbf{u}}$ *such that angle between* $\vec{\mathbf{u}}$ *and the normal at any point in* $\mathcal{E}_\vartheta$ *is less than* $\pi/2$*, then* $\mathcal{E}_\vartheta$ *is star-shaped.*

such that ϑ is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/2)$. A similar 2D result holds for the faces of the cell.

Before proving the result, we introduce a definition.

DEFINITION 6 *Consider a line segment* $\mathbf{pq}$ *that intersects* $\mathcal{E}$*.*

- *We call an intersection point* $\mathbf{r} \in \mathcal{E}_{\mathbf{pq}}$ *a* **tangential intersection point** *if* $\mathbf{pq} \cdot \mathbf{n_r} = 0$*. Otherwise we say* $\mathbf{r}$ *is a* **transversal intersection point**. *We call* $\mathbf{r}$ *an* **entry point** *if* $\mathbf{pq} \cdot \mathbf{n_r} < 0$*, an* **exit point** *if* $\mathbf{pq} \cdot \mathbf{n_r} > 0$*.*
- *If all the intersection points are transversal then we say* $\mathbf{pq}$ *intersects* $\mathcal{E}$ **transversally**.

If $\mathbf{pq}$ intersects $\mathcal{E}$ transversally, then we can classify each intersection point as an entry point or an exit point. We can sort the intersection points in the order of increasing distance from $\mathbf{p}$. In the sorted order, the entry and exit points will alternate each other. This is because $\mathcal{E}$ is an oriented closed manifold.

THEOREM 3 *Let* ϑ *be a boundary voxel, i.e.* $\mathcal{E}_\vartheta \neq \emptyset$*. If there exists a unit vector* $\vec{\mathbf{u}}$ *such that* $\mathcal{E}_\vartheta$ *is* normal-bounded *w.r.t* $(\vec{\mathbf{u}}, \theta)$ *for any* $\theta,\ 0 < \theta < \pi/2$*, then*

1. *$\mathcal{E}$ is star-shaped w.r.t ϑ.*

2. *Any point belonging to the set*

$$\{(\mathbf{x} - \alpha\|\vartheta\|\vec{\mathbf{u}}) \quad | \quad \mathbf{x} \in \vartheta,\ \alpha > 1/\cos(\theta)\}$$

is a guard of $\mathcal{E}_\vartheta$.

Proof: (2) implies (1). Hence we prove (2). Choose any point $\mathbf{x} \in \vartheta$. Let $\mathbf{c} = \mathbf{x} - \alpha\|\vartheta\|\vec{\mathbf{u}}$. Consider any point $\mathbf{p} \in \mathcal{E}_\vartheta$. We first show that $\mathbf{cp} \cdot \mathbf{n_p} > 0$.

See Fig. 3.16. We have

$$\begin{aligned}
\mathbf{cp} &= \mathbf{cx} + \mathbf{xp} \\
&= \alpha\|\vartheta\|\vec{\mathbf{u}} + \mathbf{xp} \\
\mathbf{cp} \cdot \mathbf{n_p} &= \alpha\|\vartheta\|\vec{\mathbf{u}} \cdot \mathbf{n_p} + \mathbf{xp} \cdot \mathbf{n_p} \\
&> \alpha\|\vartheta\|\cos(\theta) + \mathbf{xp} \cdot \mathbf{n_p} \quad \text{because } \angle\vec{\mathbf{u}}, \mathbf{n_p} < \theta \\
&> \|\vartheta\| + \mathbf{xp} \cdot \mathbf{n_p} \\
&> 0 \quad \text{because } \|\mathbf{xp}\| < \|\vartheta\| \text{ and } \mathbf{n_p} \text{ is a unit vector}
\end{aligned}$$

We now show that $\mathbf{cp} \cap \mathcal{E} = \{\mathbf{p}\}$. We first note that $\mathbf{cp}$ intersects $\mathcal{E}$ transversally because otherwise it will contradict the fact that $\mathbf{cp} \cdot \mathbf{n_p} > 0$ for all $\mathbf{p} \in \mathcal{E}_\vartheta$.

Suppose $\mathbf{cp}$ intersects $\mathcal{E}$ at a point other than $\mathbf{p}$. Let $\mathbf{r}$ be a point belonging to $\mathcal{E} \cap (\mathbf{cp} \setminus \{\mathbf{p}\})$ that is closest to $\mathbf{p}$. Since $\mathbf{r}$ and $\mathbf{p}$ are "consecutive" intersection points, one of them is an entry point and the other is an exit point. This means either $\mathbf{cp} \cdot \mathbf{n_r} < 0$ or $\mathbf{cp} \cdot \mathbf{n_p} < 0$ which is a contradiction.

□

We note that the condition in the above theorem is sufficient, but not necessary. The above theorem can also be rephrased as follows: If the Gauss map $\mathbb{G}_\vartheta(\mathcal{E}_\vartheta)$ is a strict subset of a hemisphere of $\mathbb{S}^2$, then $\mathcal{E}_\vartheta$ is star-shaped. The unit vector $\vec{\mathbf{u}}$ will point towards the apex of such a hemisphere. Similar Gauss map conditions have been widely used in the boundary evaluation literature (Hohmeyer, 1991).

Conservative Star-shaped Test

In the case where $\mathcal{E}$ is non-linear, we use a conservative technique to answer the star-shaped query (Sec. 3.5.4). The conservative technique enumerates a set of samples

on $\mathcal{E}$ to estimate a candidate point for the guard and verifies whether $\mathcal{E}$ is actually star-shaped w.r.t the candidate point. Due to a poor estimate of the candidate point, it is possible that a voxel satisfies the condition in Theorem 3, but still fails the star-shaped criterion as per the conservative technique. However, under a more restrictive condition than the one used in Theorem 3, we can show that a voxel will satisfy the star-shaped criterion even as per the conservative technique. This restrictive condition requires that a voxel be normal-bounded by $\pi/2$.

COROLLARY 4 *Consider a voxel ϑ that is* normal-bounded *by θ, $0 < \theta < \pi/2$. Then*

1. *$\mathcal{E}$ is star-shaped w.r.t ϑ.*

2. *Any point belonging to the set*

$$\{(\mathbf{p} - \alpha\|\vartheta\|\mathbf{n_p}) \quad | \quad \mathbf{p} \in \mathcal{E}_\vartheta,\ \alpha > 1/\cos(\theta)\}$$

is a guard of $\mathcal{E}_\vartheta$.

This corollary follows directly from Theorem 3 by choosing $\vec{\mathbf{u}} = \mathbf{n_p}$ for any point $\mathbf{p} \in \mathcal{E}_\vartheta$. The corollary provides a sufficient condition when a voxel will satisfy the star-shaped criterion as per the conservative technique. It suffices to choose merely one sample (say $\mathbf{p} \in \mathcal{E}_\vartheta$) within the voxel. Then any point belonging to the set

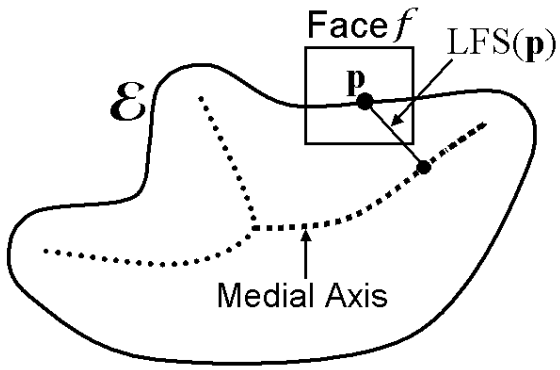
$$\{(\mathbf{p} - \alpha\|\vartheta\|\mathbf{n_p}) \mid \alpha > 1/\cos(\theta)\}$$

can be chosen as a guard. In practice, we enumerate more than one sample point to obtain a faster convergence. This enables us to verify the star-shaped criterion even in the case where the voxel is not normal-bounded by $\pi/2$.

3.6.4 Local Feature Size Condition

In this subsection, we derive a conservative lower bound on the size of the grid cells during adaptive subdivision. This provides a sufficient condition for the termination of the algorithm.

We reduce the Gauss map conditions for both the complex cell and star-shaped criteria to a common condition based on *local feature size* (LFS). We start by defining the LFS of a cell. Then we show that both the Gauss map conditions are met if the



(a)

Figure 3.17: Local Feature Size (LFS): *The LFS of a point* $\mathbf{p}$ *w.r.t* $\mathcal{E}$ *is defined as the distance between* $\mathbf{p}$ *and the medial axis of* $\mathcal{E}$*. We extend this definition to a face. The LFS of a face* f *is defined as the minimum of the LFS of all points in* $\mathcal{E}_f$*.*

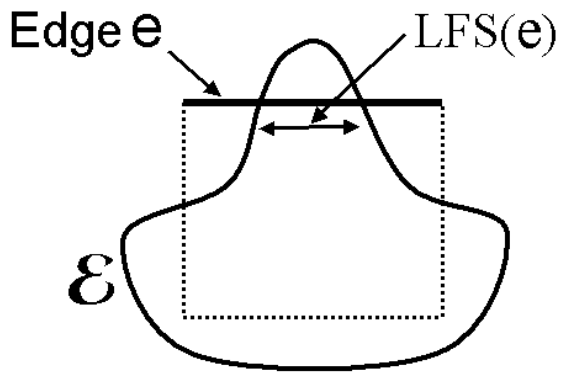
grid cells are smaller than a certain fraction of their LFS. This yields a lower bound on the cell size and in turn a sufficient condition for termination of the algorithm.

We define the LFS of a cell, which in turn is defined in terms of the LFS of its voxel, faces and edges. The LFS of a voxel is defined as the minimum of the LFS of all the points on $\mathcal{E}$ that belong to the voxel. The LFS of a face/edge is defined similarly by considering the restriction of $\mathcal{E}$ to the face/edge. See Figs. 3.17 and 3.18.

LFS of a Voxel: Let LFS : $\mathcal{E} \rightarrow \mathbb{R}$ denote the LFS of $\mathcal{E}$. Recall that LFS($\mathbf{p}$) at a point $\mathbf{p}$ on $\mathcal{E}$ is defined as the least distance of $\mathbf{p}$ to the *medial axis* of $\mathcal{E}$. Then the LFS of a voxel ϑ is defined as:

$$\begin{aligned} \text{LFS}(\vartheta) &= \min\{\text{LFS}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_\vartheta\} \quad \text{if } \mathcal{E}_\vartheta \neq \emptyset \\ &= \infty \quad \text{otherwise} \end{aligned}$$

LFS of a Face: Let Π_f be the plane containing a face f. Consider the restriction $\mathcal{E}_\Pi$ of $\mathcal{E}$ to Π. We can treat $\mathcal{E}_\Pi$ as a curve in $\mathbb{R}^2$; hence we can use a 2D definition of LFS for $\mathcal{E}_\Pi$. Let the LFS be defined by the function $\text{LFS}_{\Pi_f} : \mathcal{E}_\Pi \rightarrow \mathbb{R}$. Then the LFS of f is defined as:



(a)

Figure 3.18: LFS of an edge: *This figure shows the LFS of an edge e that is intersected by $\mathcal{E}$ at two points. In this case, the LFS of the edge is equal to the distance between the two intersection points.*

$$\begin{aligned} \mathrm{LFS}(f) &= \min\{\mathrm{LFS}_{\Pi_f}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_f\} \quad \text{if } \mathcal{E}_f \neq \emptyset \\ &= \infty \quad \text{otherwise} \end{aligned}$$

See Fig. 3.17.

LFS of an Edge: Let l_e be the line containing an edge e. Consider the restriction $\mathcal{E}_{l_e}$ of $\mathcal{E}$ to l_e. We define a one-dimensional LFS_{l_e} for points on $\mathcal{E}_{l_e}$ in terms of three cases:

1. If $\mathcal{E}_{l_e} = \emptyset$, then LFS_{l_e} is always infinity.

2. Suppose $\mathcal{E}$ intersects l_e at one point. If this intersection is tangential, then LFS_{l_e} is always zero. Otherwise it is infinity.

3. $\mathcal{E}$ intersects l_e at multiple points. Then LFS_{l_e} is defined as follows:

$$\begin{aligned} \mathrm{LFS}_{l_e}(\mathbf{p}) &= 0 \quad \text{if } \mathbf{p} \text{ is a tangential intersection point} \\ &= \inf\{d(\mathbf{p},\mathbf{q}) \mid \mathbf{q} \in \mathcal{E}_{l_e},\ \mathbf{q} \neq \mathbf{p}\} \quad \text{otherwise} \end{aligned}$$

The LFS of edge e is defined as follows:

$$\begin{aligned} \mathrm{LFS}(e) &= \min\{\mathrm{LFS}_{l_e}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_e\} \quad \text{if } \mathcal{E}_e \neq \emptyset \\ &= \infty \quad \text{otherwise} \end{aligned}$$

See Fig. 3.18.

LFS of a Cell: The LFS of a cell is defined as the minimum of the LFS of its edges, faces, and the voxel.

Intuitively, our goal is to show that a cell will satisfy the complex cell and star-shaped criteria if it is "sufficiently small". We make the previous statement precise using the following definition.

DEFINITION 7 *Let c be an edge/face/voxel of a cell C.*

1. *c is* LFS-small *if $\|c\| < \rho\,\mathrm{LFS}(c)$ for any $\rho < \beta/(1+3\beta)$ where $\beta = arcsin(1/\sqrt{(3)})$.*
2. *C is* LFS-small *if every edge/face/voxel of C is* LFS-small.

The choice of the value of ρ is determined by Theorem 5 (see below). We show that a LFS-small cell satisfies the complex cell and star-shaped criteria. Our proof relies on the following lemma presented by Amenta and Bern (Amenta and Bern, 1998). It basically states that the normals at two closeby points on the surface are close to each other. The surface $\mathcal{E}$ is assumed to be a twice-differentiable manifold.

LEMMA 6 *For any two points $\mathbf{p}$ and $\mathbf{q}$ on $\mathcal{E}$ with $d(\mathbf{p},\mathbf{q}) \leq \rho \min\{\mathrm{LFS}(p), \mathrm{LFS}(q)\}$, for any $\rho < 1/3$, the angle between the normals to $\mathcal{E}$ at $\mathbf{p}$ and $\mathbf{q}$ is at most $\rho/(1-3\rho)$ (Amenta and Bern, 1998).*

THEOREM 5 *Let C be a* LFS-small *cell. Then,*

1. *C is not complex.*
2. *$\mathcal{E}$ is star-shaped w.r.t C.*

Proof: Let c be a face/voxel of C. Because c is LFS-small, any two points in $\mathcal{E}_c$ are within distance $\|c\| < \rho$ LFS(c) where $\rho < \beta/(1+3\beta)$. Then according to Lemma 6, the angle between the normals to any two points in $\mathcal{E}_c$ is at most $\rho/(1-3\rho) = \beta = arcsin(1/\sqrt{(3)})$. Hence c is normal-bounded by β. We now use Theorem 2 and Corollary 4 to prove the result.

To apply Theorem 2, we choose $\vec{\mathbf{u}}$ to be the normal at any point in $\mathcal{E}_c$. Thus c is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$. Furthermore, by definition, an LFS-small edge is not complex. Therefore C cannot have complex edges. Theorem 2 implies that C is not complex.

Since c is normal-bounded by $\pi/4$, Corollary 4 ensures that $\mathcal{E}$ is star-shaped w.r.t both voxel as well as faces of C.

□

3.6.5 Termination

Theorem 5 provides a lower bound on the size of the grid cells relative to the LFS. During adaptive subdivision, once the size of a cell C is less than $\rho\text{LFS}(C)$, then it is LFS-small, and satisfies both the complex cell and star-shaped criteria. The algorithm will terminate provided there exist a lower bound on the LFS of every grid cell. Suppose there exists such a lower bound τ. Assuming a cell halves its size at each subdivision step, this implies a lower bound of $\rho\tau/2$ on the size of every cell. We use this fact to provide the following sufficient condition for the termination of the subdivision algorithm:

COROLLARY 6 *If there exists an $\tau > 0$, such that during adaptive subdivision, the LFS of every grid cell is greater than τ, then the subdivision algorithm will terminate and the grid cells will be of size greater than $\rho\tau/2$.*

In general, it is difficult to enforce the above condition during adaptive subdivision. During the subdivision process, as new voxels, faces and edges are created, the LFS of the newly created voxels/faces/edges change. While it is true that the LFS of a voxel of a child cell is always greater than or equal to the LFS of the voxel of its parent cell, a similar property does not hold for the faces and edges of the children cells. The LFS for the newly created faces and edges depends on how they intersect $\mathcal{E}$. If these faces and edges intersect $\mathcal{E}$ in a "near grazing" manner, then their LFS can be arbitrarily small. For example, if $\mathcal{E}$ has large flat regions that are parallel to the axis-aligned planes of the coordinate system (XY, YZ, or ZX), then the LFS of some of the faces and edges can be zero. Such problems may sometimes be alleviated by choosing a different set of co-ordinate axes.

We conclude the analysis with a few remarks. We note that LFS-small condition is sufficient, but not necessary. Furthermore, the lower bound $(\rho\tau/2)$ is an overly conservative lower bound on the cell size. In practice, we have observed that the

algorithm produces much larger cells. Finally, even though our analysis is restricted to only smooth surfaces, our algorithm is applicable to surfaces with sharp features.

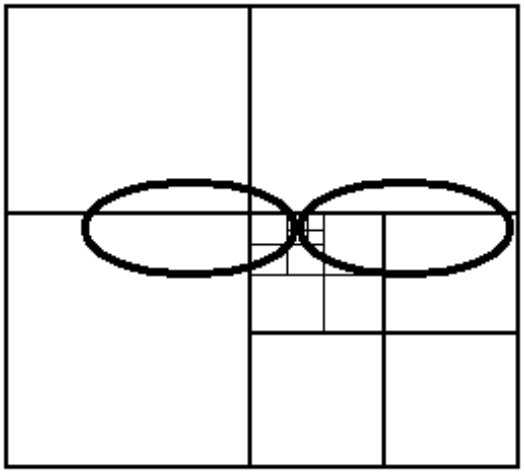


Figure 3.19: Tangential Contact: *This figure shows a case where two primitives touch each other at a point. We call such a contact a* tangential contact*. It is a degenerate case for our algorithm.*

3.7 Degeneracies

There are two types of degenerate cases for our algorithm. The first type of degeneracy occurs when $\mathcal{E}$ has a *tangential contact*. See Fig. 3.19. This can occur when two input primitives touch each other. Different types of tangential contacts are possible: the contact region may be a point, curve, or a surface. Our algorithm cannot handle such cases. No matter how much subdivision is performed in the vicinity of the tangential contact, the complex cell and star-shaped properties are never satisfied.

Another kind of degeneracy occurs when the $\mathcal{E}$ grazes an edge or a face of a cell. The contact region may be a point, curve, or a surface. We refer to these types of situations as *grazing contacts*. Fig. 3.5 shows a few examples. In the vicinity of a grazing contact, the sampling condition is never met; an edge (face) with grazing contact will not satisfy the complex edge (complex face) criterion. One way of reducing the likelihood of grazing contacts is to perform the adaptive subdivision randomly, i.e., choose random points to split the voxel, faces, and edges of the cell. For example, we can randomly select a point in the interior of the voxel, and subdivide the voxel into tetrahedral regions with the chosen point as an apex. This type of subdivision would generate a tetrahedral

grid. Isosurface extraction can then be performed using an MC-like algorithm such as Marching Tetrahedra (Payne and Toga, 1990b; Gueziec and Hummel, 1995).

Both tangential contact and grazing contact can be characterized in terms of the LFS. At a tangential contact, the LFS of the voxel containing the contact is zero. Similarly, at a grazing contact along a face or an edge, the LFS of the corresponding face or edge is zero.

Detecting the occurence of either type of degeneracy is difficult. This is because we do not have an explicit representation of $\mathcal{E}$. Note that the portion of $\mathcal{E}$ involved in the contact may belong to either a single primitive or the intersection curve between multiple primitives. While it may be possible to detect the first case, the second case is much harder as this requires intersection curve computation.

Handling degeneracies is a challenging problem that has been extensively studied in solid modeling (Seidel, 1994; Hoffmann, 2001; Ouchi and Keyser, 2004). Many approaches have been proposed to handle them. One option is to perform "special case handling": enumerating all the possible types of degeneracies and adding code to explicitly detect and resolve them. While this approach can be useful in many situations, it leads to more complexity in the underlying algorithms and representations, e.g., these algorithms need to work with non-manifold representations. Moreover, these algorithms will not be compatible with MC-like reconstruction methods: We cannot use special-purpose algorithms in cells containing degeneracies and MC-like methods in the remaining cells as this may lead to cracks in the output.

Another approach to handling degeneracies is to use perturbation methods. This approach applies a perturbation to the input to eliminate the degeneracy. The perturbation may be done either symbolically (Seidel, 1994) or numerically (Ouchi and Keyser, 2004). It may be possible to use perturbation methods to resolve grazing and tangential contacts. The input primitives defining $\mathcal{E}$ can be numerically perturbed. This defines a perturbed surface $\mathcal{E}'$. If the perturbation is chosen randomly, it is likely that $\mathcal{E}'$ is not degenerate. We can then apply our adaptive subdivision approach to the perturbed input.

The main advantage of perturbation methods is that we can apply the adaptive subdivision algorithm directly without having to explicitly handle degeneracies. However, there are a few issues with using numerical perturbation. It modifies the input data, and hence the output will be topologically equivalent to the perturbed surface $\widetilde{\mathcal{E}}$ and not the original surface $\mathcal{E}$. Moreover, adding a numerical perturbation may not necessarily eliminate the degeneracy, or even worse, it may create another. Therefore,

this method requires a robust test for detecting degeneracy. If the applied perturbation does not resolve the degeneracy, another perturbation is needed.

3.8 Geometric Error Bound

We extend our adaptive subdivision algorithm to generate $\mathcal{A}$ with a bounded geometric error. We define the geometric error as the two-sided Hausdorff distance $H(\mathcal{A}, \mathcal{E})$, which we call the *Hausdorff error*. For a definition of Hausdorff distance, see Sec. 3.1.

We describe a simple extension to the adaptive subdivision algorithm that bounds the Hausdorff error. We exploit the fact that $\mathcal{E}$ is a subset of the boundaries of a set of primitives. We bound the Hausdorff distance between $\mathcal{A}$ and the boundaries of these primitives. Assume that we are given an error tolerance $\epsilon > 0$. Let $\Gamma = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ denote the set of primitives that define $\mathcal{E}$. We augment the subdivision algorithm with the following criterion:

DEFINITION 8 Hausdorff criterion ($\mathcal{C}^\epsilon$) : *Given an $\epsilon > 0$, a cell C satisfies $\mathcal{C}^\epsilon$ if*

$$H(\mathcal{A}_C, \mathcal{P}_{i,C}) < \epsilon \quad \forall \quad \mathcal{P}_i \in \Gamma \text{ such that } \mathcal{P}_{i,C} \neq \emptyset$$

where $\mathcal{P}_{i,C} = \mathcal{P}_i \cap C$ is the restriction of $\mathcal{P}_i$ to C.

If every grid cell bounding $\mathcal{E}$ satisfies $\mathcal{C}^\epsilon$, then the Hausdorff error of $\mathcal{A}$ is also bounded by ϵ.

THEOREM 7 *If every cell $C \in \mathcal{G}$ satisfies $\mathcal{C}^\epsilon$, then*

$$H(\mathcal{A}, \mathcal{E}) < \epsilon$$

We combine $\mathcal{C}^\epsilon$ with $\mathcal{C}^\square$ and $\mathcal{C}^\bigstar$ to define the following sampling condition:

DEFINITION 9 *A cell C satisfies $\mathcal{C}^{\square\bigstar\epsilon}$ if C satisfies both $\mathcal{C}^{\square\bigstar}$ and $\mathcal{C}^\epsilon$.*

If we apply $\mathcal{C}^{\square\bigstar\epsilon}$ during adaptive subdivision, then we obtain a reconstruction $\mathcal{A}$ with a bounded two-sided Hausdorff error as well as correct topology. See Fig. 3.11(c).

We compute an upper bound δ on $H(\mathcal{A}_C, \mathcal{P}_{i,C})$, and check if δ is less than ϵ. The diameter of the cell $diam(C)$ serves as a trivial upper bound. Hence a simple subdivision criterion is to subdivide a cell if its diameter is greater than ϵ. Using $diam(C)$ as an upper bound can be overly conservative: It may result in excessive subdivision for

small values of ϵ. This is because the diameter of the cell is a loose upper bound on the Hausdorff distance between $\mathcal{A}_C$ and $\mathcal{P}_{i,C}$. It is possible to obtain a tighter upper bound in the case where all the primitives $\mathcal{P}_i$ are polyhedral. Since the output of Marching Cubes $\mathcal{A}_C$ is polygonal, the problem reduces to bounding the Hausdorff distance between two polygonal objects.

Hausdorff distance computation between polygonal objects is a well studied problem. Aspert et al. (Aspert et al., 2002) and Guthe et al. (Guthe et al., 2005) propose efficient techniques for estimating the Hausdorff distance. We provide a brief description of these techniques. Let $\mathcal{P}$ and $\mathcal{Q}$ denote two polygonal objects whose Hausdorff distance needs to be computed. We will focus on computing the forward distance $h(\mathcal{P}, \mathcal{Q})$; the backward distance can be computed similarly. For a fixed point $\mathbf{p} \in \mathcal{P}$, it is relatively easy to calculate the distance $d(\mathbf{p}, \mathcal{Q})$. However, the goal is to obtain the maximum over all points $\mathbf{p} \in \mathcal{P}$, which is difficult in practice. The above techniques resort to sampling in order to estimate $h(\mathcal{P}, \mathcal{Q})$. Each polygon of $\mathcal{P}$ is sampled, and the distance between each sample and $\mathcal{Q}$ is computed. The maximum over these distances provides an estimate of the Hausdorff distance.

We can use similar techniques to compute an upper bound δ on $h(\mathcal{P}, \mathcal{Q})$. We use the error tolerance ϵ to determine the sampling density; we recursively subdivide the polygons in P such that each polygon has a diameter less than or equal to $\epsilon/2$. The vertices of the resulting polygons provide the set S of samples. Let $\tau = \max_{\mathbf{s} \in S} d(\mathbf{s}, \mathcal{Q})$ and $\delta = \tau + \epsilon/2$. We can show that δ is an upper bound on $h(\mathcal{P}, \mathcal{Q})$. The proof is as follows.

Consider any point $\mathbf{p} \in \mathcal{P}$. Let $\mathbf{r}$ be any vertex of the polygon containing $\mathbf{p}$. Since every polygon has a diameter less than or equal to $\epsilon/2$, we have $d(\mathbf{p}, \mathbf{r}) \leq \epsilon/2$. Since $\mathbf{r}$ belongs to the set S of samples used in estimating the Hausdorff distance, we have $d(\mathbf{r}, Q) \leq \tau$. Let $\mathbf{q}$ be the point on $\mathcal{Q}$ that is closest to $\mathbf{r}$; in other words $d(\mathbf{r}, \mathcal{Q}) = d(\mathbf{r}, \mathbf{q})$. Then by triangle inequality, it follows:

$$d(\mathbf{p}, \mathbf{q}) \leq d(\mathbf{p}, \mathbf{r}) + d(\mathbf{r}, \mathbf{q}) \leq \epsilon/2 + \tau = \delta$$

This implies that $d(\mathbf{p}, \mathcal{Q}) \leq \delta$. Since this is true for any arbitrary point $\mathbf{p} \in \mathcal{P}$, it follows that $h(\mathcal{P}, \mathcal{Q}) \leq \delta$.

We can use the above technique to compute upper bounds on both forward and backward Hausdorff distances between $\mathcal{A}_C$ and $\mathcal{P}_{i,C}$. Let δ_1^i and δ_2^i denote the upper bounds on $h(\mathcal{A}_C, \mathcal{P}_{i,C})$ and $h(\mathcal{P}_{i,C}, \mathcal{A}_C)$, respectively. The maximum $\delta^i = \max(\delta_1^i, \delta_2^i)$

is an upper bound on the two-sided distance $H(\mathcal{A}_C, \mathcal{P}_{i,C})$. Let $\delta_C = \max_i \delta^i$. We can use δ_C as a threshold during adaptive subdivision: we check if $\delta_C > \epsilon$ and in that case, subdivide cell C. Using δ_C, rather than the cell diameter, can alleviate the problem of excessive subdivision.

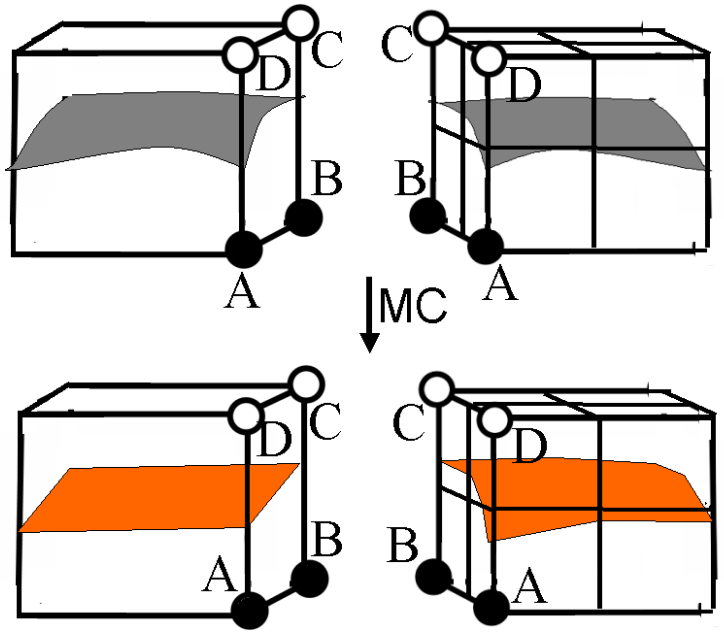
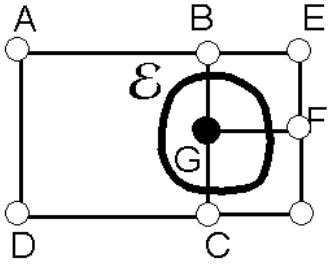


Figure 3.20: Isosurface Extraction on Adaptive Grids: *This figure shows a cell at a coarse resolution sharing a face ABCD with 4 cells at a finer resolution. Applying Marching Cubes to these grid cells results in a reconstruction that does not match along the common face ABCD.*

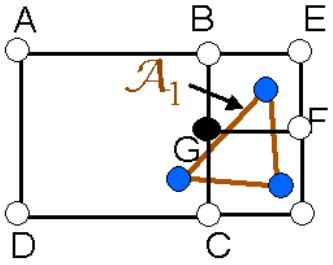
The above technique is applicable only to polygonal primitives. In case of non-linear primitives, it is harder to bound the Hausdorff distance. One possible solution relies on computing *Sleve* to the non-linear primitive (Peters, 2003). Sleve is a pair of matched triangulations that sandwiches a surface. We can exploit the fact that the two triangulations enclose the non-linear surface to compute an upper bound on the Hausdorff distance. Consider a surface $\mathcal{P}$ whose Sleve consists of triangulations $\mathcal{P}_1^s$ and $\mathcal{P}_2^s$. If both $H(\mathcal{A}, \mathcal{P}_1^s) < \epsilon$ and $H(\mathcal{A}, \mathcal{P}_2^s) < \epsilon$, then we have $H(\mathcal{A}, \mathcal{P}) < \epsilon$. We can compute the distance between $\mathcal{A}$ and the two triangulations using the techniques described for polygonal objects.

3.9 Isosurface Extraction on Adaptive Grids

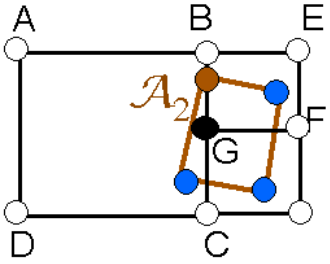
The adaptive subdivision algorithm generates an adaptive volumetric grid on which we perform isosurface extraction. Applying the original Marching Cubes algorithm (Lorensen and Cline, 1987) to an adaptive grid can result in cracks in the reconstructed isosurface. A crack occurs when the reconstructed isosurface in two adjacent cells at different resolutions does not match along a shared face (see Fig. 3.20). This becomes an issue when defining a homeomorphism between $\mathcal{E}$ and $\mathcal{A}$. One solution this problem is to employ crack patching (Shekhar et al., 1996). Crack patching modifies the extracted isosurface within the larger cell to match the extracted surface within the smaller cell. In this way, we ensure that our approximation $\mathcal{A}$ is C^0 continuous. Crack patching maintains the property that $\mathcal{A}$ restricted to the edges, faces, and voxel of the cell is a topological disk. As a result, we can still define a homeomorphism between $\mathcal{E}$ and $\mathcal{A}$, and the topological equivalence result holds.



(a) 2D Example



(b) Dual Contouring



(c) Modified Dual Contouring

Figure 3.21: 2D Modified Dual Contouring: *This figure illustrates the working of the Dual Contouring algorithm on an adaptive grid in 2D. Fig. (a) shows (in 2D) an iso-surface $\mathcal{E}$ (black curve). Fig. (b) shows the output $\mathcal{A}_1$ (brown curve) of Dual Contouring algorithm. $\mathcal{A}_1$ violates a requirement of our algorithm – it does not intersect the same set of edges as $\mathcal{E}$. For example, while $\mathcal{E}$ intersects edge BG, $\mathcal{A}_1$ does not. Therefore, we apply a modification to the Dual Contouring algorithm to enforce this requirement. The output $\mathcal{A}_2$ of this algorithm, shown in Fig. (c), satisfies the requirement.*

A better alternative is to use dual methods such as Dual Contouring (Ju et al., 2002) for isosurface extraction. An important advantage of these methods is that they can handle adaptive grids easily, and can produce a reconstructed isosurface without any cracks. We provide a brief description of the Dual Contouring algorithm. See Fig. 3.21 for a 2D example. It operates on a grid in two steps. First, for each cell that exhibits a sign change across the edges, this method examines the set of intersection points and generates a vertex (per cell) such that a quadratic error function is minimized. We refer to this vertex as the *error-minimizing* vertex. Second, for each edge that exhibits a sign change, the contouring method connects the error-minimizing vertices of the cells sharing the edge. In 2D grids, each edge is shared by two cells; hence the method outputs a line segment connecting the error-minimizing vertices of the two adjacent cells sharing the edge. In 3D uniform grids, each edge is shared by four cells; hence the method outputs a quad for each edge. In 3D adaptive grids, some of the edges in the grid may be shared by three cells; for such edges, the method outputs a triangle.

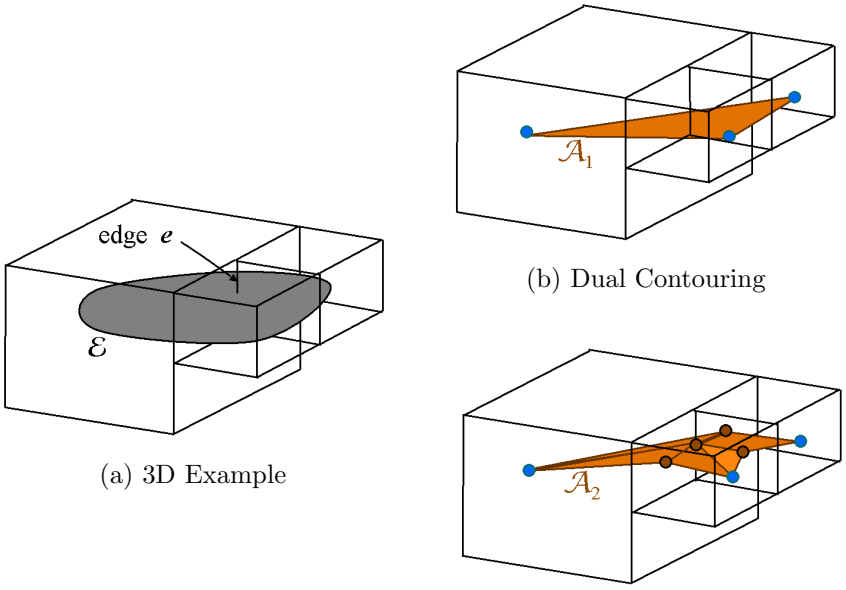
We use Dual Contouring for isosurface extraction. However, we cannot apply Dual Contouring directly – this is because the Dual Contouring does not satisfy Property 1 in Section 3.4.4. Property 1 states that the reconstructed isosurface $\mathcal{A}$ must intersect each edge, face, or voxel that exhibits a sign change. However, when applied to an adaptive grid, Dual Contouring may not satisfy this property. For example, in Fig. 3.21(b), edge BG exhibits a sign change, but does not intersect $\mathcal{A}$.

We correct this problem using a simple modification to the dual contouring method. We insert an additional intersection point along edge BG and connect it to the error-minimizing vertices of the adjacent cells. This is shown in Fig. 3.21(c). With this simple modification, both $\mathcal{E}$ and $\mathcal{A}$ exhibit identical sign configurations for every cell.

A similar modification works in 3D. See Fig. 3.22. It shows an edge e that intersects $\mathcal{E}$. The dual contouring method outputs a triangle that may not satisfy Property 1. We circumvent this problem by inserting an additional vertex v on e and generating a triangle fan around v. See Fig. 3.22(c). In order to satisfy Property 1, merely inserting v may not be enough: we also insert additional vertices on the faces that are adjacent to e. This is shown in Fig. 3.22(c).

With the above modification, we ensure that Dual Contouring preserves the sign configuration of the cell. It maintains the property that $\mathcal{A}$ restricted to the edges, faces, and voxel of the cell is a topological disk. As a result, the topological equivalence between $\mathcal{E}$ and $\mathcal{A}$ holds.

The above modification may increase the number of triangles in the reconstructed



(a) 3D Example

(b) Dual Contouring

(c) Modified Dual Contouring

Figure 3.22: 3D Modified Dual Contouring: *This figure illustrates the working of the Dual Contouring algorithm on an adaptive grid in 3D. Fig. (a) shows a portion of an isosurface $\mathcal{E}$ intersecting an edge e. Fig. (b) shows the output $\mathcal{A}_1$ of Dual Contouring algorithm. However, $\mathcal{A}_1$ violates a requirement that it should intersect the same set of edges as $\mathcal{E}$. While $\mathcal{E}$ intersects edge e, $\mathcal{A}_1$ does not. Therefore, we use a modified Dual Contouring algorithm that enforces this requirement. The output $\mathcal{A}_2$ of this algorithm, shown in Fig. (c), satisfies the requirement.*

isosurface. However, the modifications need to be applied only when Property 1 is violated. Therefore, we can first check whether Property 1 fails, and only then apply the modification. Since typically such a violation occurs only in a small number of cells, the increase in the number of triangles is not substantial.

3.10 Speedup Techniques

In this section we present two speedup techniques that improve the efficiency of the adaptive subdivision algorithm. Together, they improve the overall performance significantly.

3.10.1 Cell Culling

Since our objective is to approximate $\mathcal{E}$, it is sufficient to process only those cells that intersect $\mathcal{E}$. Based on this fact, we classify the cells in the grid into two types:

DEFINITION 10 *A cell C is a* **boundary cell** *if $\mathcal{E}_C \neq \emptyset$, and a* **non-boundary cell** *otherwise.*

We need to apply the sampling condition to only the boundary cells. While application of the sampling condition to a non-boundary cell preserves correctness of the algorithm, it is undesirable because it adds an unnecessary overhead to the algorithm. We reduce this overhead by using a technique called *cell culling*. Cell culling enables the adaptive subdivision algorithm to disregard non-boundary cells and improves the overall performance considerably.

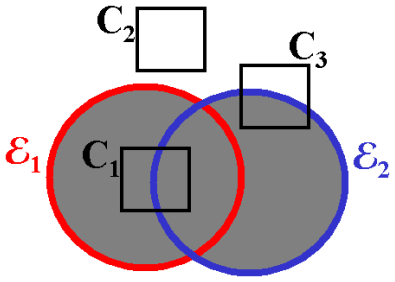


Figure 3.23: Cell Culling: *This figure shows a case where $\widetilde{\mathcal{E}}$ is defined as a union of two solids, $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$. The gray shaded region shows the union. Since C_1 is contained within $\widetilde{\mathcal{E}}_1$, it is contained within the union and is a non-boundary cell. Therefore, C_1 can be disregarded. Similarly, C_2 can also be disregarded. On the other hand, cell C_3 is a boundary cell and needs to be taken into account.*

Let $\widetilde{\mathcal{E}}$ denote the solid enclosed by $\mathcal{E}$. We disregard a cell C if it is either completely inside or completely outside $\widetilde{\mathcal{E}}$. This is the main idea behind cell culling. Fig. 3.23 shows an example of cell culling where $\widetilde{\mathcal{E}}$ is defined as a union of two solids, $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$.

Cell culling is based on the voxel intersection test (Lem. 4). We use the following property: If the signed max-norm distance to $\mathcal{E}$ at the center of a cell C is greater than half the voxel size, then we can disregard C as it is guaranteed not to intersect $\mathcal{E}$. As explained in Sec. 3.5.1, we do not compute the exact signed distance D^s_∞ to $\mathcal{E}$; it is sufficient to compute a conservative estimate $\widetilde{D}^s_\infty$ of the distance. Lemma 5 ensures that the absolute value of $\widetilde{D}^s_\infty$ is less than the absolute value of D^s_∞. Therefore, we

can use $\widetilde{D}^s_\infty$ to perform cell culling while preserving correctness. The condition for cell culling is as follows:

Cell Culling Condition: Let C be a cell with center $\mathbf{o}$ and length l.

$$\text{If} \quad |\widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E})| > l/2 \quad \text{then } C \text{ is a non-boundary cell.}$$

We show an application of this condition to cell C_1 in the union example (Fig. 3.23). Let d_1 and d_2 denote the signed max-norm distances from $\mathbf{o}$ to $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Since cell C_1 lies completely within $\widetilde{\mathcal{E}}_1$, we have $d_1 < -l/2$. This implies:

$$\begin{aligned} \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}) = \min(d_1, d_2) &< -l/2 \\ \implies |\widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E})| &> l/2 \end{aligned}$$

Therefore C_1 satisfies the condition for cell culling and we can rule out C_1 from further consideration. Similarly, we can employ cell culling for intersection and difference operations. It can also be used for a sequence of Boolean operations.

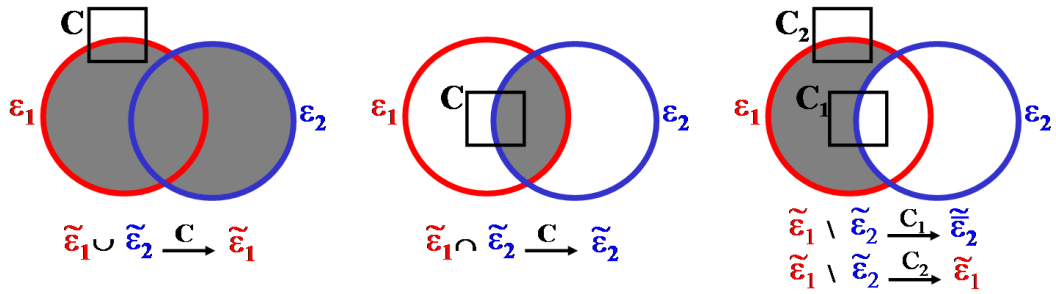
Cell culling is conservative: While every cell discarded by cell culling is a non-boundary cell, the converse is not true – it may not eliminate all the non-boundary cells. This is because we use a conservative estimate $\widetilde{D}^s_\infty$ of the signed max-norm distance. Due to conservativeness of cell culling, we may unnecessarily process some non-boundary cells; however, this does not affect the correctness of the algorithm.

3.10.2 Expression Simplification

The adaptive subdivision algorithm processes each cell in the grid independently. Within a cell C, it needs to consider only $\mathcal{E}_C$ – the portion of $\mathcal{E}$ that is contained within C. It may be possible to define $\mathcal{E}_C$ by simplifying the Boolean expression of $\mathcal{E}$. This is the main idea behind *expression simplification*.

$\widetilde{\mathcal{E}}$ denotes the solid enclosed by $\mathcal{E}$. By a slight abuse of notation, we will use $\widetilde{\mathcal{E}}$ to also refer to the Boolean expression associated with the solid $\widetilde{\mathcal{E}}$.

Consider the case of a union $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$, as shown in Fig. 3.24(a). Because cell C lies completely outside $\widetilde{\mathcal{E}}_2$, C is not influenced by $\widetilde{\mathcal{E}}_2$. Consequently, we can simplify the Boolean expression by getting rid of $\widetilde{\mathcal{E}}_2$ from the expression. This produces a simplified expression $\widetilde{\mathcal{E}}_1$ for cell C. We refer to this step as expression simplification. It can also be used for intersection and difference operations as well as a sequence of Boolean operations. See Figs. 3.24(b), 3.24(c).



(a) Union (b) Intersection (c) Difference

Figure 3.24: Expression Simplification: *Figs. (a), (b), & (c) shows examples of expression simplification for union, intersection, and difference operations respectively. In each figure, the gray shaded region shows the final solid obtained by performing the Boolean operation.*

In order to simplify an expression of $\widetilde{\mathcal{E}}$ w.r.t a cell C, we need to determine if C lies completely outside or completely inside the solids corresponding to the sub-expressions of $\widetilde{\mathcal{E}}$. For example, in the case of a union $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$, we need to test if C is completely outside $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$. We perform these tests using max-norm distance computation. Given a primitive $\mathcal{P}$ and a cell C, we use the following conditions:

$$\begin{array}{lll} \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{P}) > l/2 & \text{then} & \text{C lies completely outside } \widetilde{\mathcal{P}} \\ \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{P}) < -l/2 & \text{then} & \text{C lies completely inside } \widetilde{\mathcal{P}} \end{array}$$

where $\mathbf{o}$ and l are the center and length of cell C respectively.

Below we provide the conditions for expression simplification. We provide a condition for each Boolean operation: union, intersection, difference, and complement. $\mathcal{A} \xrightarrow{C} \mathcal{B}$ denotes a simplification of a Boolean expression $\mathcal{A}$ into an expression $\mathcal{B}$ in cell C.

1. **Union:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving union operation when C lies completely outside either $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$.

$$\begin{array}{llll} \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) > l/2 & \text{then} & \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \\ \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) > l/2 & \text{then} & \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{array}$$

2. **Intersection:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving intersection operation when C lies completely inside either $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$.

$$\begin{aligned} \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) < -l/2 \quad \text{then} \quad \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \\ \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) < -l/2 \quad \text{then} \quad \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{aligned}$$

3. **Difference:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving difference operation in either of two cases: (1) C lies completely inside $\widetilde{\mathcal{E}}_1$; (2) C lies completely outside $\widetilde{\mathcal{E}}_2$.

$$\begin{aligned} \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) < -l/2 \quad \text{then} \quad \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2 \xrightarrow{C} \overline{\widetilde{\mathcal{E}}_2} \\ \text{If} \quad \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) > l/2 \quad \text{then} \quad \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{aligned}$$

4. **Complement:** $\widetilde{\mathcal{E}} = \overline{\widetilde{\mathcal{E}}_1}$
   We can simplify a complement of an expression $\widetilde{\mathcal{E}}_1$ by simplifying $\widetilde{\mathcal{E}}_1$.

$$\text{If} \quad \widetilde{\mathcal{E}}_1 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \quad \text{then} \quad \overline{\widetilde{\mathcal{E}}_1} \xrightarrow{C} \overline{\widetilde{\mathcal{E}}_2}$$

Fig. 3.24 shows several examples of expression simplification.

We perform expression simplification in a top-down manner during adaptive subdivision. With every cell C, we maintain the corresponding Boolean expression $\widetilde{\mathcal{E}}_C$. The root cell of subdivision is associated with the original Boolean expression $\widetilde{\mathcal{E}}$. Each time we subdivide a cell C into a set of children cells $C_i, i = 1, \ldots, k$, we simplify $\widetilde{\mathcal{E}}_C$ w.r.t the children cells, i.e.,

$$\widetilde{\mathcal{E}}_C \xrightarrow{C_i} \widetilde{\mathcal{E}}_{C_i}$$

This gives us a Boolean expression $\widetilde{\mathcal{E}}_{C_i}$ for each child cell C_i.

Alg. 1 shows pseudo-code for the complete adaptive subdivison algorithm including expression simplification. Expression simplification can reduce the original Boolean expression considerably. As the adaptive subdivision progresses, the expressions corresponding to the subdivided cells become progressively simpler. Typically, the simplified expression requires only a small number of Boolean operations. This improves the performance of the overall algorithm considerably.

Consider a situation where $\widetilde{\mathcal{E}}$ is expressed as a union of a high number of primitives $\widetilde{\mathcal{E}} = \cup_i \widetilde{\mathcal{P}}_i$. For a cell C we can simplify $\widetilde{\mathcal{E}}$ by eliminating all the primitives that lie completely outside C. Typically the resulting simplified Boolean expression has only a small number of primitives. This type of union operation arises frequently in many other problems such as Minkowski sum and swept volume computation. In Chapter 4, we discuss its application to Minkowski sum computation. For our inputs, the Minkowski sum reduces to a union of several tens of thousands of primitives. Without expression simplification, computing such a large union would be infeasible. Using expression simplification, we reduce the problem to computing a union of only a small number of primitives – typically less than a hundred. We will discuss this in more detail in Chap. 4.

3.11 Performance

In this section, we analyze the performance of our algorithm. The total time taken by the algorithm is the sum of the times taken by the sampling and reconstruction steps.

- **Sampling:** This is the dominant step of our algorithm. The total time taken by this step is given by $T_S = \sum_{C \in \mathcal{G}} T(C)$ where $T(C)$ is the time spent on a single cell $C \in \mathcal{G}$. We will provide a bound on $T(C)$ below.

- **Reconstruction:** MC-like methods spend $O(1)$ time on each cell of the grid. Therefore, the time complexity of this step is $O(N)$ where N is the number of cells in the grid.

We analyze the *cell complexity*, T(C), the time taken to process a single grid cell C. $T(C)$ is the sum of the time taken by the complex cell and star-shaped tests. Below we bound the time taken by each test separately. We begin by analyzing the time complexity of the two tests on a single primitive. We consider two cases – depending on whether the primitive is polyhedral or algebraic.

Polyhedral Primitive

Consider a polyhedral primitive with n polygons. We define the size of the primitive to be n.

- **Complex Cell Test:** The complex cell test requires two types of computations:

  - **Sign query:** Determining the sign of a point takes $O(n)$ time.
  - **Cell intersection:** This requires directed distance and max-norm distance computation. Each of them takes $O(n)$ time.

- **Star-shaped test:** For a polyhedral primitive, the star-shaped test reduces to linear programming. We combine the linear constraints defined by each polygon of the primitive, and solve the resulting linear program. This step takes $O(n)$ expected time.

Algebraic Primitive

We assume an algebraic primitive with a fixed degree.

- **Complex Cell Test:**
  - **Sign query:** Computing a sign for an algebraic primitive requires evaluating a polynomial function. We consider the time taken by this step to be $O(1)$.
  - **Cell intersection:** We use directed and max-norm distance computation for low degree algebraic primitives and interval arithmetic for higher order primitives. For an algebraic primitive, both directed and max-norm distance computation reduce to solving a univariate polynomial equation. We consider the time taken by this step to be $O(1)$. Performing interval arithmetic on an algebraic primitive reduces to evaluating a polynomial function. We assume this step also takes $O(1)$ time.

- **Star-shaped test:** Our star-shaped test for an algebraic primitive uses a combination of linear programming and interval arithmetic. The linear programming step requires a discretization of the algebraic primitive. We assume that each algebraic primitive $\mathcal{P}$ has an associated discretization (see Sec. 3.5.4). Let n denote the number of points in the discretization. We define the size of the algebraic primitive to be n. The linear programming step takes an $O(n)$ expected time. We assume the interval arithmetic step takes $O(1)$ time.

Therefore, applying the complex cell and star-shaped tests to a single primitive – polyhedral or algebraic – takes $O(n)$ expected time, where n is the size of the primitive.

Our algorithm performs the complex cell and star-shaped tests on $\mathcal{E}$. Specifically, within a cell C, we perform them on $\mathcal{E}_C$, whose associated Boolean expression $\widetilde{\mathcal{E}_C}$ is obtained by performing expression simplification. Let $\Gamma_C = \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}\}$ denote the

set of primitives in the Boolean expression $\widetilde{\mathcal{E}}_C$. In order to perform the complex cell and star-shaped tests on $\mathcal{E}_C$, we need to take into account every primitive in Γ_C. For example, in order to compute the sign of a point w.r.t $\mathcal{E}_C$, we need to compute its sign w.r.t every primitive in Γ_C. Similarly, in case of star-shaped test, we combine the linear constraints derived from the all the primitives in Γ_C. Therefore, the complex cell and star-shaped tests take $\sum_{j=1}^{k} O(n_j)$ time where n_j is the size of $\mathcal{P}_{i_j}$. This means the cell complexity T(C) is given by:

$$T(C) = \sum_{j=1}^{k} O(n_j) = O(n)$$

where $n = \sum_{j=1}^{k} n_j$.

3.12 Implementation & Applications

In this section, we describe the implementation of our algorithm and highlight three different applications: Boolean operations, simplification, and remeshing.

We used C++ programming language with the GNU g++ compiler under Linux operating system. We demonstrate the performance of our algorithm on many complex models. Table 3.1 highlights the performance of our algorithm on these models. All execution times are on a 2 GHz Pentium IV PC with a GeForce 4 graphics card and 1 GB RAM.

In all our applications, we first generate an adaptive octree grid using our adaptive subdivision algorithm. We then compute a polyhedral approximation to the boundary of the final solid using a modified Dual Contouring algorithm (See 3.9). The reconstructed surface is a manifold. We used a freely available linear programming package, *QSOPT* (QSOPT, 2005), to implement the star-shaped test.

3.12.1 Boolean operations

Figure 3.25(left) shows the reconstruction of the final surface generated by our algorithm on the dragon model. The solid is defined as a union of two dragons, each with over 870K triangles. It took 95 secs to compute the approximate union. The second example is obtained by performing 5 difference operations on the Turbine Blade model (see rightmost image in Fig. 1.9). The model has more than 1.7 million triangles. The final surface has multiple components and a higher genus. Our algorithm computes

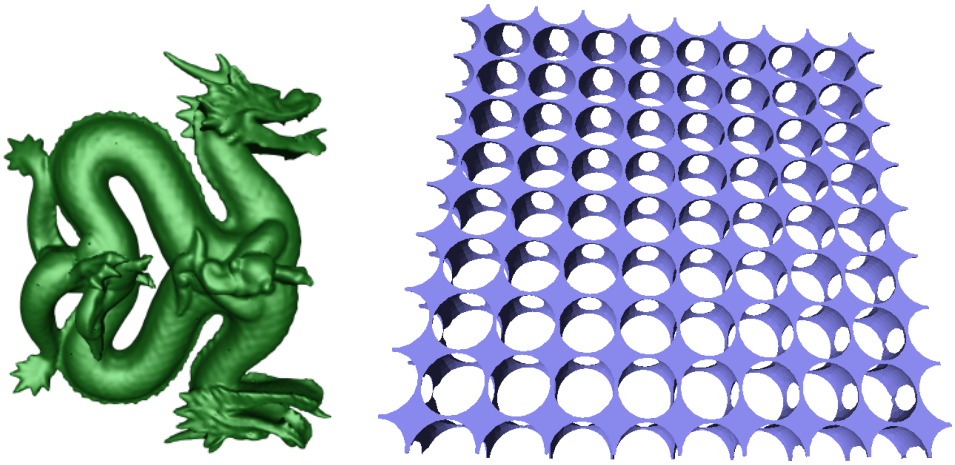
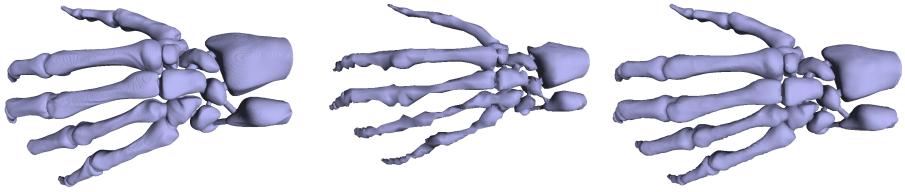


Figure 3.25: Boolean operations on complex models and curved primitives: *The left figure shows the result of the union of two dragons. Each dragon is represented using* $850K$ *triangles. Our algorithm computes a topology preserving approximation of the final boundary. It took* 95 *secs to compute an approximation with* $118K$ *triangles at a relative Hausdorff error bound of* $1/128$*. Recall that the relative Hausdorff error is defined as the ratio of the absolute Hausdorff error to the maximum length of a "tight" axis-aligned bounding box around the object. The right figure shows the result of* 100 *difference operations between a polyhedron and* 100 *ellipsoids. The resulting surface has a complex topology; it has a genus of* 208*. Our algorithm took* 16 *secs to generate an approximation with the correct topology at a relative Hausdorff error bound of* $1/64$*.*

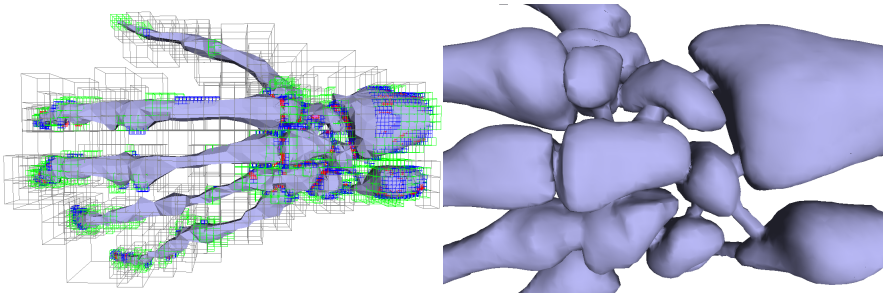
the boundary in 116 secs. Figure 3.25(right) highlights application of our algorithm to perform Boolean operations on curved primitives. It shows a challenging scenario where we perform 100 difference (Boolean) operations between a polyhedron and 100 ellipsoids. The resulting surface has a complex topology with numerous small holes; it has a genus of 208. Our algorithm took 16 secs to generate an approximation with the correct topology.

3.12.2 Topology Preserving Volumetric Simplification

Model simplification algorithms produce a lower polygon count approximation of a polygonal object that preserves the shape or appearance of the object. Simplification techniques have been used for fast display and simulation. In order to compute a drastic simplification for interactive visualization, many algorithms do not preserve the surface topology. On the other hand, preserving topology during simplification is important for applications like CAD, medical visualization, and molecular modeling. Volumetric



(a) Hand model (b) Coarse Approximation (c) Fine Approximation



(d) Adaptive Subdivision (e) Closeup

Figure 3.26: Topology-Preserving Simplification: *Figs (a),(b),(c) show the original model along with a coarse and fine approximation generated using our topology preserving simplification algorithm. The original model has* $654K$ *triangles, while the two approximations consist of* $27K$ *and* $58K$ *triangles respectively at relative Hausdorff error bounds of* 1 *and* $1/128$*. Fig (d) shows the adaptive grid generated for the coarse approximation. The colors, green, blue, and red in that order, indicate the increasing level of subdivision. Fig (e) shows a closeup view of a part of the fine approximation. It highlights the features in the original model and our algorithm is able to reconstruct all these features accurately. It took* 36 *secs to generate the approximation.*

algorithms have been proposed for model simplification (He et al., 1996; Nooruddin and Turk, 2003). These algorithms are fast in practice and are applicable to all kind of models. However, none of these algorithms give rigorous guarantees on preserving the surface topology.

We compute a discrete sampling of the distance field by applying our adaptive subdivision algorithm. Different levels of detail are generated by changing the value of the two-sided Hausdorff error tolerance. The reconstruction algorithm generates an isosurface that has the same topology as the original model. Our metric of Hausdorff error can be easily combined with other metrics such as curvature, quadric error, etc, to guide the simplification.

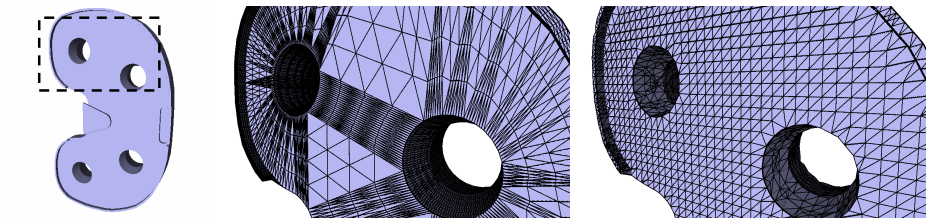


Figure 3.27: Remeshing of a CAD model: *The left image shows a CAD model, which has many "skinny" triangles with poor aspect ratios (center image). We used our algorithm to compute an improved triangulation of the model (right image). The original model had* $41K$ *triangles. Our algorithm took* 2.8 *secs to perform remeshing and generate a mesh with* $16K$ *triangles at a relative Hausdorff error of* $1/32$.

	Combi. Complexity		Performance (secs)			
Model	Input	Output	Complex	Star-shaped	Subdivision	Total
Hand Simplification (Fig. 3.26)	654,666	58,966	4.3	8.1	23	36
Turbine Simplification (Fig. 1)	1,765,388	511,182	14.1	31.3	65.8	110
Turbine Blade Boolean (Fig. 1)	1,765,388	319,074	16.8	29.1	70.4	116
Union of Dragons (Fig. 3.25)	1,714,920	118,214	10.2	21.1	63.6	95
Curved Boolean (Fig. 3.25)	-	57,286	5.4	5.1	5.5	16
Brake Hub Remeshing (Fig. 1.10)	14,208	7,056	0.38	0.55	0.90	1.85
CAD model Remeshing (Fig. 3.27)	41,152	16,524	0.58	0.81	1.41	2.8

Table 3.1: Performance: *This table highlights the complexity of our input models and performance of our algorithm. The columns on the left shows the the triangle count of the input and triangle count in our reconstruction. The columns on the right show the cumulative time taken by the complex cell test, star-shaped test and adaptive subdivision over all the grid cells. The rightmost column shows the total execution time.*

Figure 3.26 shows our simplification algorithm applied to a medical dataset, a $650K$ triangle *Hand* model. This model has a number of topological features that need to be preserved in order to maintain the anatomical structure of the hand. Figs. 3.26(b) ($27K$ triangles) and 3.26(c) ($58K$ triangles) show a coarse and a fine approximation, respectively, of the original model. The coarse approximation was computed by applying our adaptive subdivision algorithm without imposing any Hausdorff error bound. The resulting grid is shown in Fig. 3.26(d). Fig. 3.26(e) shows a closeup view of part of the finer approximation. It took 36 secs to generate the approximation. Figure 1.9 shows our simplification algorithm applied to the Turbine Blade model. This model has a high genus and many tunnels in the interior. It took 110 secs to generate a simplified model with $511K$ triangles. Note that our method preserves the complex topological features in the simplified model.

Some prior surface simplification methods can be adapted to perform topology preserving simplification (Cohen et al., 1996; Zelinka and Garland, 2002). However, one limitation of these approaches is that they need to perform global tests to avoid surface

self-intersections, which can result in considerable overhead. On the other hand, our algorithm is guaranteed not to produce self intersections.

Our subdivision criterion ensures that the isosurface is a topological disk within each grid cell by satisfying the complex cell and star-shaped criteria. In applications such as simplification and remeshing, a simpler test exists. In these applications, we have a polygonization of the original surface. We can ensure the topological disk property by computing the Euler characteristic and testing if it is equal to 1. However, in case of Boolean operations, we do not have a polygonization of the final surface (in fact, our goal is to compute a polygonization). Consequently, the test based on Euler characteristic does not work for Boolean operations.

3.12.3 Topology Preserving Remeshing

Volumetric approaches have been used for remeshing of polygonal models (Kobbelt et al., 2001; Nooruddin and Turk, 2003). In many applications, the polygonal models can have triangles with bad-aspect ratios. The goal is to compute a valid manifold representation of the underlying closed solid and ensure that the resulting triangles have a good aspect ratio. The mesh generated after remeshing can be used for multiresolution analysis or simplification.

Earlier algorithms generated a volumetric representation by sampling the distance field on a uniform grid (Kobbelt et al., 2001), or with a simplified topology (Nooruddin and Turk, 2003). However, these methods provide no guarantees on the genus or the number of connected components. We have applied our subdivision algorithm to compute a topology preserving remeshing of CAD models. Figs. 1.10 and 3.27 show some of our results. The Euler characteristic test (see Section 3.12.2) could also be used for remeshing.

3.12.4 Discussion

Table 3.1 highlights the performance of our algorithm on these models. It also provides a breakup of the total time spent in performing the complex cell test, star-shaped test, and adaptive subdivision. This corresponds to the time taken to "push" the input triangles down the octree data structure. For each octree cell C, we perform expression simplification to obtain a smaller set of primitives Γ_C that define the isosurface within the cell. We maintain a list T_C of triangles that belongs to the primitives in Γ_C. We maintain the set T_C in cell C. As we subdivide C, we partition it into 8 children C_i and

compute their triangle lists T_{C_i} similarly. For large input models, this takes substantial fraction of the total time.

3.13 Limitations

In this section, we discuss the limitations of our algorithm. As discussed earlier in Sec. 3.7, our algorithm does not terminate in certain degenerate cases. Our algorithm can only generate manifold boundaries and is not applicable to the cases where the exact boundary is non-manifold.

We do not provide a bound on the time complexity of our algorithm as a function of the combinatorial complexity of the input primitives. This is because of we are unable to give an absolute lower bound on the size of the grid cells generated during adaptive subdivision. Sec. 3.6 provides a lower bound on the cell size relative to the LFS of a cell. However, to obtain an absolute bound, the LFS needs to be expressed as a function of the combinatorial complexity of the input. This requires further analysis.

Our algorithm may perform conservative subdivision. Within a cell, we require that all the primitives should be star-shaped with respect to a common guard. This is a conservative condition. The isosurface defined by the Boolean expression over the primitives can be star-shaped within the cell even though this condition may not be satisfied. This can result in additional subdivision and lead to higher polygon counts in the approximation.

Our topology preserving simplification algorithm cannot perform drastic simplifications. This is due to the conservative subdivision and also the fact that volumetric approaches can not produce drastic simplifications (El-Sana and Varshney, 1997). Moreover, for a fixed polygon budget, approaches based on surface decimation operations like edge collapses or vertex removal (Cohen et al., 1996) will generate a higher quality simplification.

3.14 Summary

In this chapter, we have described a novel approach to compute topology preserving isosurfaces that arise in a variety of geometric processing applications. We have presented a sufficient sampling condition based on the *complex cell* and *star-shaped* criteria so that the reconstruction maintains the topology of the original isosurface. We have described a simple extension to the sampling condition to also bound the two-sided Hausdorff

error of the reconstruction. We have also described an adaptive subdivision algorithm which is efficient in practice and easy to implement. We have demonstrated the application of our algorithm to Boolean operations, topology preserving simplification, and remeshing on a number of complex examples.

Chapter 4

Minkowski Sum Approximation

The Minkowski sum of two sets P and Q is the set of points $\{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in P,\ \boldsymbol{q} \in Q\}$. Minkowski sums have been used for robot motion planning (Lozano-Pérez, 1983), penetration depth computation (Cameron, 1997; Kim et al., 2002), offset computation (Rossignac and Requicha, 1986), and mathematical morphological operations (Williams and Rossignac, 2004).

Our goal is to compute the boundary of the 3D Minkowski sum of two polyhedral models. The Minkowski sum of two convex polytopes (with n features) can have $O(n^2)$ combinatorial complexity and is relatively simple to compute. On the other hand, the Minkowski sum of non-convex polyhedra is much more challenging: its complexity can be as high as $O(n^6)$ (Halperin, 2002b). One common approach is to decompose the two non-convex polyhedra into convex pieces, compute their pairwise Minkowski sums, and finally the union of the pairwise Minkowski sums (Lozano-Pérez, 1983). The main bottleneck in implementing such an algorithm is computing the union of pairwise Minkowski sums. Given m pairwise Minkowski sums, their union can have a combinatorial complexity $O(m^3 + sm\log(m))$ where s is the number of total number of faces in the pairwise Minkowski sums (Aronov et al., 1997). In the context of Minkowski sum computation of complex non-convex polyhedral models, m is typically high – usually several thousands. Furthermore, robust computation of the boundary of the union and handling all degeneracies remains a major issue (Halperin, 2002b; Abrams and Allen, 2000). As a result, no practical algorithms are known for robust computation of exact Minkowski sum of complex non-convex polyhedral models.

Instead of computing the exact union, we approximate the union using our isosurface extraction algorithm based on $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition (Chapter 3). This yields an approximation to the exact Minkowski sum boundary. The geometric and topo-

logical guarantees of our isosurface extraction algorithm apply to the Minkowski sum approximation as well: Provided the $\mathcal{C}^{\square\bigstar\epsilon}$ condition is satisfied, the approximation is topologically equivalent to the exact Minkowski sum boundary and has a bounded two-sided Hausdorff error.

In order to speed up the computation, we employ two culling techniques that are specialized for Minkowski sum computation. The sampling algorithm performs *cell culling* to eliminate the grid cells that do not contain a part of the Minkowski sum boundary. The sampling algorithm also takes advantage of *primitive culling* and performs efficient distance and sign queries by only considering a small subset of primitives, while preserving the correctness of these queries. In practice, these culling techniques can improve the overall performance by more than two orders of magnitude on complex models.

We have used our Minkowski sum approximation algorithm for two applications.

- Robot motion planning of robots with translational degrees of freedom.
- Offsets and mathematical morphological operations.

Our algorithm can also be used for estimating tight penetration depth of polyhedral models.

Our algorithm is simple to implement. We have tested its performance on a number of polyhedral models ranging from several hundred to a few thousand triangles. The computation of Minkowski sum takes a few minutes on a 2 GHz Pentium IV processor.

The rest of the chapter is organized as follows. Section 1 presents our approximate algorithm to compute the boundary of Minkowski sum. It also presents speedup techniques that improve the overall performance considerably. Section 2 discusses application to offsets and mathematical morphological operations. Section 3 discusses application to penetration depth estimation. Section 4 highlights the performance of our algorithm on a number of complex polyhedral models. Section 5 provides a summary of the algorithm and a discussion of its limitations.

4.1 Approximate Algorithm

In this section, we present our algorithm for approximating the 3D Minkowski sum of polyhedral models.

4.1.1 Overall Approach

One common approach for computing Minkowski sum of general polyhedra is based on *convex decomposition* (Lozano-Pérez, 1983). It uses the following property of Minkowski sum. If $P = P_1 \cup P_2$, then $P \oplus Q = (P_1 \oplus Q) \cup (P_2 \oplus Q)$. The approach combines this property with convex decomposition:

1. Compute a convex decomposition for each polyhedron
2. Compute the pairwise convex Minkowski sums between all possible pairs of convex pieces in each polyhedron.
3. Compute the union of pairwise Minkowski sums.

After the second step, there are $O(mn)$ pairwise Minkowski sums where m and n are the number of convex pieces of the two polyhedra. The pairwise Minkowski sums are convex and their union can have a combinatorial complexity $O(k^3 + sk\log(s))$, where $k = mn$ and s is the number of total number of faces in the pairwise Minkowski sums (Aronov et al., 1997).

Our algorithm for Minkowski sum computation is based on the above framework. We now discuss each of the above steps in detail.

4.1.2 Convex Decomposition

The problem of computing an optimal convex decomposition of a non-convex polyhedron is known to be NP-hard (O'Rourke and Supowit, 1983). Chazelle proposed one of the earliest convex decomposition algorithms (Chazelle, 1981), which can generate $O(r^2)$ convex parts and uses $O(nr^3)$ time, where n and r are the number of polygons and notches in the original polyhedron respectively. However, no robust implementation of this algorithm is known. Most practical algorithms for convex decomposition perform surface decomposition or tetrahedral volumetric decomposition (Joe, 1991; Chazelle et al., 1997; Ehmann and Lin, 2001). Typically, these methods can generate $O(n)$ convex parts where each part has only a few faces.

We use a modification of the convex decomposition scheme available in a public collision detection library, SWIFT++ (Ehmann and Lin, 2001). This method is an implementation of the algorithm presented in (Chazelle et al., 1997). It performs surface decomposition and generates a set of convex patches c_i's of the object boundary ∂P. We compute a convex hull of each surface patch, c_i, and denote the resulting polytope

by C_i. The C_i's constitute a convex decomposition of object P. C_i's consists of two types of faces: *real faces* that belong to the original polyhedron and *virtual faces* that are artifacts of the convex hull computation. In general, the union of C_i's need not cover the entire volume of P. C_i's may create some undesirable voids in the interior of P that are bounded by the virtual faces. We disregard these voids by ignoring the virtual faces while performing the sign, distance, and star-shaped queries.

Given two polyhedra P and Q each with n triangles, the convex decomposition method divides each polyhedron typically into $O(n)$ convex parts. In practice, each convex part usually has very few polygons ($4-8$ on an average). Computing pairwise Minkowski sums between all pairs of convex pieces results in $O(n^2)$ pairwise Minkowski sums. Although this quadratic complexity may seem high, it should be viewed in context of the high complexity of the final Minkowski sum ($O(n^6)$). Even though we may need to compute the union of a large number of primitives (pairwise Minkowski sums), the primitives themselves are relatively simple and typically have a low combinatorial complexity. Our approximate algorithm is well suited to this problem.

4.1.3 Pairwise Minkowski Sum Computation

We compute the pairwise Minkowski sums between all possible pairs of convex pieces, C_i^P and C_j^Q, belonging to P and Q, respectively. Let us denote the resulting Minkowski sum as M_{ij}. We use a *convex hull algorithm* to compute M_{ij}. Its complexity is $O(n^2 \log n)$ where n is the number of vertices in C_i^P and C_j^Q. Algorithms with better time complexity bounds are known, e.g., the algorithm by Guibas and Seidel (Guibas and Seidel, 1987) can compute the Minkowski sum of two convex polyhedra in $O(n+k)$ time, where k is the number of polygons in the output. However, since C_i^P, C_j^Q and M_{ij} usually have a constant combinatorial complexity, we chose use the simpler convex hull algorithm described below.

Convex Hull Approach: It is based on the following property:

$$P \oplus Q = \mathbf{CH}(\{\mathbf{v}_i + \mathbf{v}_j | \mathbf{v}_i \in V_P, \mathbf{v}_j \in V_Q\}) \tag{4.1}$$

Here, **CH** denotes the convex hull operator, and V_P, V_Q represent the sets of vertices, respectively in polyhedra P and Q. Based on this fact, we compute the Minkowski sum as follows:

1. Compute the vector sum between all possible pairs of vertices from each polytope.
2. Compute their convex hull.

4.1.4 Union Computation

The Minkowski sum $\mathcal{M}$ is given by the union of the pairwise Minkowski sums: $\mathcal{M} = \cup_{i,j} M_{ij}$. However, computing an exact union of the pairwise Minkowski sums is not practical due to the high number of the pairwise Minkowski sums. In our input models, $\mathcal{M}$ is defined by union of tens of thousands of primitives (pairwise Minkowski sums). Exact boundary evaluation of this size is slow and prone to robustness problems.

Instead of computing the exact union, we approximate the union using our sampling-based isosurface extraction algorithm. We use the $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition to generate a grid $\mathcal{G}$ on which we perform isosurface extraction. The reconstructed isosurface is an approximation $\mathcal{A}$ to $\partial\mathcal{M}$, the boundary of the exact Minkowski sum. The geometric and topological guarantees of our isosurface extraction algorithm apply to the Minkowski sum approximation as well. Therefore, we have the following result.

THEOREM 8 *If every cell in $\mathcal{G}$ satisfies $\mathcal{C}^{\square\bigstar\epsilon}$, then*

1. **Geometric Guarantee:** $H(\mathcal{A}, \partial\mathcal{M}) < \epsilon$.

2. **Topological Guarantee:** $\mathcal{A} \approx \partial\mathcal{M}$.

Together, the geometric and topological guarantees ensure a good approximation.

4.1.5 Speedup Techniques

A naive application of the above approximate algorithm can result in poor performance. This is because in the context of Minkowski sum computation, we are dealing with a high number of primitives (M_{ij}'s). We use two culling techniques to significantly accelerate the algorithm.

Cell Culling

As explained in Sec. 3.10.1, the sampling condition needs to be applied to only *boundary cells*, i.e., cells C such that $\partial\mathcal{M} \cap C \neq \emptyset$. We perform cell culling (Sec. 3.10.1) to eliminate "non-boundary" cells, This improves the performance of our algorithm considerably.

Let Γ denote the set of all the pairwise Minkowski sum primitives (M_{ij}'s). In the context of Minkowski sum computation, the cell culling step discards a cell C if either of two conditions hold:

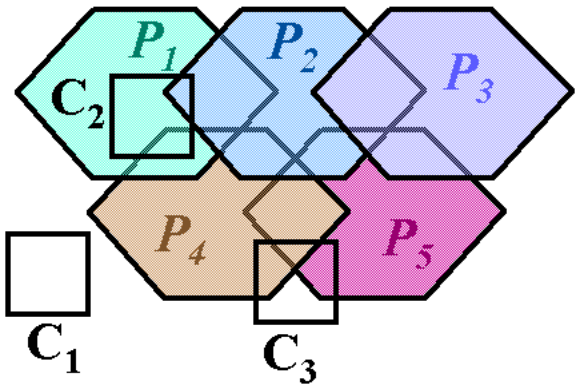


Figure 4.1: Cell and Primitive Culling: *Cell culling discards cells C_1 and C_2. Primitive culling on cell C_3 reduces the set of primitives to $\{P_4, P_5\}$.*

1. C lies outside all the primitives, $C \cap M_{ij} = \emptyset \quad \forall M_{ij} \in \Gamma$.

2. C lies inside any of the primitives, $C \subseteq M_{ij}$ for some $M_{ij} \in \Gamma$.

We check for the above conditions using max-norm distance (see Secs. 3.10.1 and 3.10.2). See Fig. 4.1.

Primitive Culling

The sampling algorithm needs to perform sign, distance, and star-shaped queries. These queries are performed several times for each grid cell and therefore significantly impact the overall performance of the algorithm. Among these queries, the sign query and the (signed) distance query are *global* in scope in that the answer to the query may depend on all the primitives. In order to perform a query on $\mathcal{M} = \cup M_{ij}$, we may have to examine each M_{ij}. Given the large number of primitives, this can slow down the overall algorithm.

Our objective is to perform *local* queries such that the answer to the query depends only on a small subset of primitives. In particular, when performing a query within a cell, we would like to inspect only those primitives that intersect the cell. Of course, we need to perform this in a manner that preserves the correctness of the query.

We achieve this by performing expression simplification (Sec. 3.10.2). We take advantage of the fact that the above queries need not be performed within cells that have been eliminated due to cell culling. In particular, we need to perform these queries within a cell C only if $C \not\subseteq M_{ij}$ for all $M_{ij} \in \Gamma$. Applying expression simplification to

$\mathcal{M} = \cup_{ij} M_{ij}$ within C results in the following simplification:

$$\cup_{ij} M_{ij} \xrightarrow{C} \cup\{M_{ij} \in \Gamma_C\}$$

where

$$\Gamma_C = \{M_{ij} \mid \partial M_{ij} \cap C \neq \emptyset\}$$

Using the above simplification, we can answer the queries by considering a much smaller number of primitives. Hence we refer to this step as *primitive culling*. See Fig. 4.1. The correctness of primitive culling follows from the following theorem:

THEOREM 9 *Let C be a cell such that $C \not\subseteq M_{ij}$ $\forall M_{ij} \in \Gamma$. We have*

$$\mathcal{M}_C = C \cap (\cup\{M_{ij} \in \Gamma_C\})$$

Proof: Consider any primitive $M_{kl} \notin \Gamma_C$. In other words, ∂M_{kl} does not intersect cell C. We show that M_{kl} will not contribute to $\mathcal{M}_C$. Since $C \not\subseteq M_{kl}$, the only possible case is $C \cap M_{kl} = \emptyset$. In this case, we have

$$\mathcal{M}_C = C \cap (\cup M_{ij}) = C \cap (\cup\{M_{ij} \in \Gamma \setminus \{M_{kl}\}\})$$

i.e., M_{kl} does not contribute to $\mathcal{M}_C$. This concludes the proof.

□

Primitive culling is important for the efficiency of Minkowski sum computation. For example, the Minkowski sum models shown in Fig. 4.4 require a union of several tens of thousands of primitives. Primitive culling reduces the number of primitives to a much smaller number – typically, less than a hundred. Primitive culling not only drastically improves the overall performance of the algorithm, but also enables the algorithm to scale up to more complex models.

We can extend the above result to perform additional culling. Let $\mathcal{P}$ be a primitive that intersects a cell C. In order to answer queries within cell C, we only need to consider $\mathcal{P}_C = \mathcal{P} \cap C$ – the portion of $\mathcal{P}$ contained within C. We use this property to disregard triangles belonging to $\mathcal{P}$ that lie outside C. Let T_{ij} be the set of triangles in M_{ij} and $T_{ij}^C \subset T_{ij}$ be the subset of triangles that intersect C. In order to answer queries within C, we only need to consider the set $T^C = \cup_{ij} T_{ij}^C$.

4.2 Offsets and Mathematical Morphological Operations

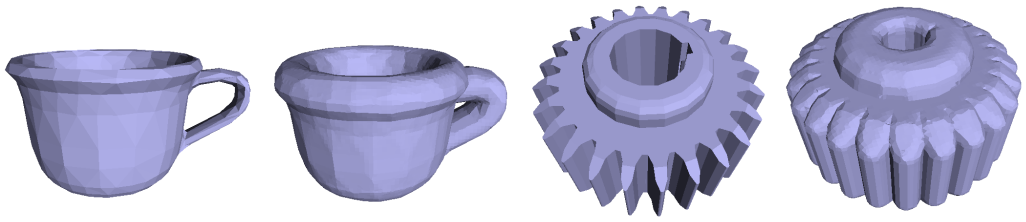
In this section, we discuss an application of our Minkowski sum approximation algorithm to mathematical morphological operations, which were introduced in Sec. 1.2.

We first consider the case of morphological operations on polyhedral objects. Both the primary morphological operations – dilation and erosion – can be expressed in terms of the Minkowski sum. Dilation of an object $\mathcal{P}$ by a structuring element $\mathcal{Q}$, is same as the Minkowski sum $\mathcal{P} \oplus \mathcal{Q}$. Erosion can be expressed in terms of a dilation of the complement of $\mathcal{P}$. Formally, we have

$$\text{Erosion}(\mathcal{P}, \mathcal{Q}) = \overline{\overline{\mathcal{P}} \oplus \mathcal{Q}'}$$

where $\overline{S}$ denotes the complement of a set S, and $\mathcal{Q}'$ denotes a copy of $\mathcal{Q}$ reflected about the origin.

Therefore, we can perform both dilation and erosion using our Minkowski sum approximation algorithm. Since our algorithm relies on a convex decomposition of the objects involved in the Minkowski sum operation, the erosion operation on an object $\mathcal{P}$ will require a convex decomposition of the complement of $\mathcal{P}$. Complementing $\mathcal{P}$ can be achieved by merely inverting the orientation of the polygons of $\mathcal{P}$. The opening and closing operations (see Sec. 1.2 for definitions) can then be implemented as a composition of dilation and erosion operations.



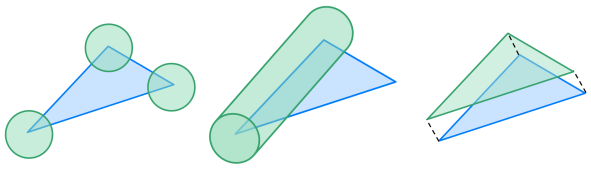
Cup (1,000 tris) Cup $\oplus$ Sphere Gear (2,382 tris) Gear $\oplus$ Sphere

Figure 4.2: Offsets: *The figures show two polygonal models, Cup and Gear, with* 1,000 *and* 2,382 *triangles respectively. Our approximation algorithm computed their offsets by computing their Minkowski sum with a sphere. It took* 33 *and* 84 *secs to compute the offsets for the two models. The approximate boundary consisted of* 14,895 *and* 22,742 *triangles.*

An interesting case of morphological operations is where the structuring element Q is a ball. In this special case, the dilation operation reduces to the offset operation. The offset of a solid is obtained by adding to the solid all the points that lie within a distance r. Mathematically it is defined as

$$\text{Offset}(\mathcal{P}) = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \exists \boldsymbol{p} \in \mathcal{P},\ \|\boldsymbol{x} - \boldsymbol{p}\| \leq r\}$$

Similarly, erosion creates an offset a distance r inwards from the original surface.



(a) Vertex (b) Edge (c) Triangle

Figure 4.3: Offset: *The offset of a triangle has three types of regions: a spherical region around each vertex, a cylindrical region around each edge and a planar region due to the displacement of the original triangle along its normal by distance r. This results in a triangular prism. We can thus express the offset of a triangle as a union of spheres, cylinders and a triangular prism.*

The offset of a polygonal object $\mathcal{P}$ consists of three types of regions:

- A spherical region around a vertex v_i of $\mathcal{P}$. This region is part of a sphere S_i of radius r centered at v_i.
- A cylindrical region around an edge e_j of $\mathcal{P}$. This region is part of a cylinder C_j of radius r and whose axis is same as e_j.
- A planar region due to a polygon p_k of $\mathcal{P}$ obtained by displacing p_k along its outward normal by a distance r. This results in a prism P_k.

See Fig. 4.2. Let $\mathcal{O} = (\cup_i S_i) \bigcup (\cup_j C_j) \bigcup (\cup_k P_k)$. The (outward) offset of $\mathcal{P}$ is given by $\mathcal{P} \bigcup \mathcal{O}$. The offset surface inwards from the original surface is given by $\mathcal{P} \setminus \mathcal{O}$. In this case the prisms P_k are obtained by displacing the polygon along the inward normal.

Thus, offset computation of polyhedral objects reduces to performing Boolean operations on polyhedra, prisms, spheres, and cylinders. We need to perform three union operations per triangle followed by a union or a difference operation with the original object. Therefore, the offset computation of an object with n triangles requires Boolean

operations on $3*n+1$ primitives. Exact computation of the offset is difficult in practice due to the presence of non-linear primitives in the boundary as well as the large number of primitives. Instead, we use our sampling-based isosurface extraction algorithm to compute a good approximation. Fig. 4.2 shows some of our results.

Note that, unlike Minkowski sum computation, the method for offset computation, does not require a convex decomposition of the object.

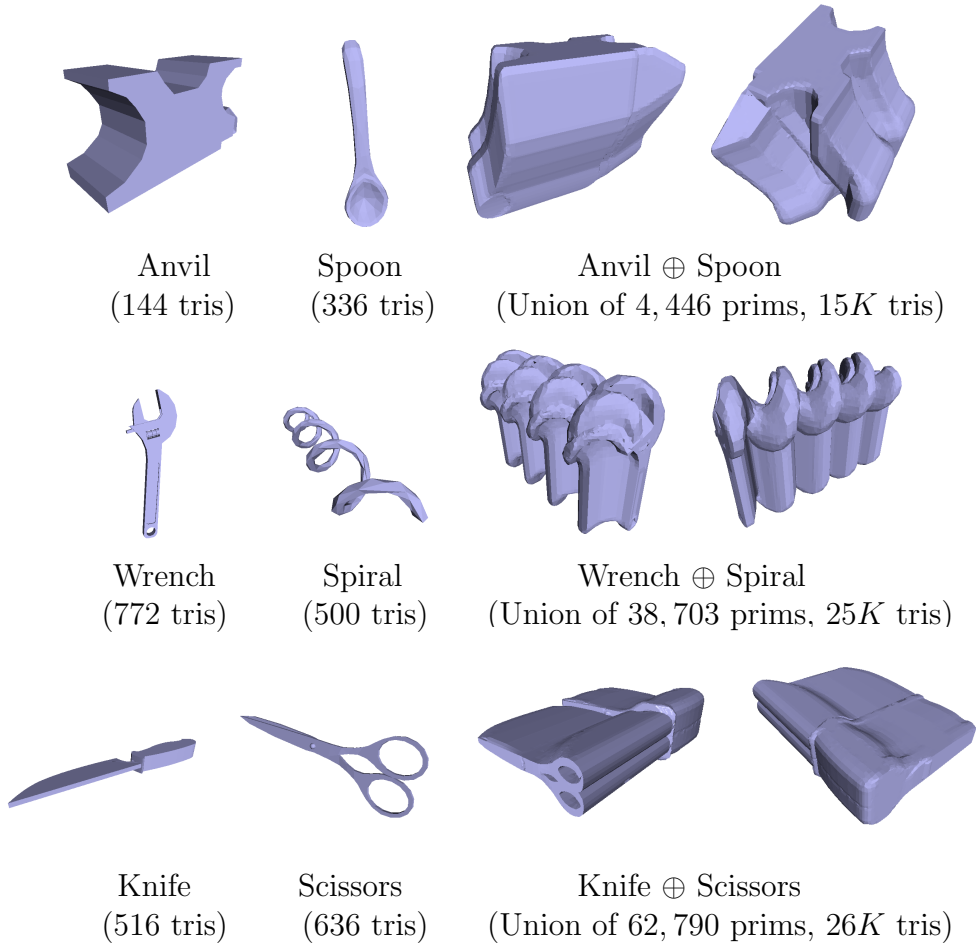
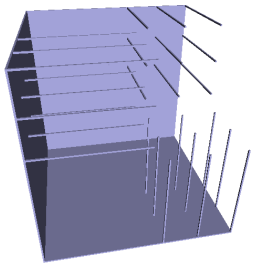
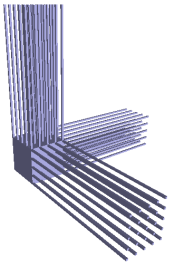


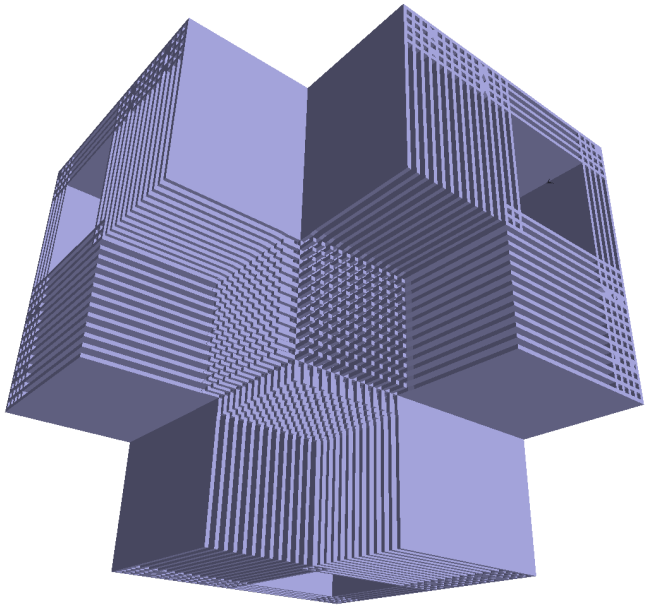
Figure 4.4: Minkowski sum of polygonal models: *The left two columns show the two primitives whose Minkowski sum is being computed. The triangle counts for the two primitives are shown in brackets. Two views of the approximation computed by our algorithm are shown in the right. Our algorithm took* 63, 316 *and* 778 *secs respectively to generate an approximation. The approximate boundary consists of* $15K$, $25K$ *and* $26K$ *triangles respectively (see Table 3.1).*



Grate 1 (444 tris)



Grate 2 (1,134 tris)



Grate 1 $\oplus$ Grate 2 (Union of 66,667 prims, 358K tris)

Figure 4.5: Complex Minkowski sum example: *The left figure show two grates with* 444 *and* 1,134 *triangles respectively. We decomposed them into* 163 *and* 409 *convex pieces respectively and computed the pairwise Minkowski sums between the convex pieces. The final Minkowski sum is given by the union of* 66,667 *pairwise Minkowski sums. Our approximation algorithm computed an approximation (shown in the right) in* 3,162 *secs (*52 *minutes). It was able to reconstruct the complex features present on the boundary.*

4.3 Penetration Depth Estimation

Our Minkowski sum approximation algorithm can be used to estimate the penetration depth between two polyhedral models. We guarantee that our estimate of penetration depth is within a user-specified tolerence ϵ of the actual value. The penetration depth of two intersecting polyhedra P and Q, $PD(P,Q)$, is the minimum translational distance that one of the polyhedra must undergo to render them disjoint. Formally, $PD(P,Q)$ is defined as:

$$\min\{\| \boldsymbol{d} \| \mid interior(P + \boldsymbol{d}) \cap Q = \emptyset\} \tag{4.2}$$

Here, $\boldsymbol{d}$ is a vector in $\mathcal{R}^3$. It is well known that one can reduce the problem of computing the PD between P and Q to a minimum distance query on the surface of their Minkowski sum, $P \oplus -Q$ (Cameron, 1997).

Our Minkowski sum approximation algorithm can be used to obtain a tight penetration depth estimate. Given any $\epsilon > 0$, we compute a geometrically close approximation $\mathcal{A}_\epsilon$, i.e., $H(\mathcal{A}_\epsilon, \partial\mathcal{M}) < \epsilon$ (Theorem 8). Our penetration depth estimate δ is given by $\delta = D(O_{Q-P}, \mathcal{A}_\epsilon)$. It is easy to prove that our estimate δ is close to the actual PD. In particular, we have $\delta - \epsilon < PD(P, Q) < \delta + \epsilon$. Thus $\delta - \epsilon$ and $\delta + \epsilon$ provide bounds on the PD. By decreasing ϵ, we can obtain arbitrarily tight bounds on the actual PD.

Our Minkowski sum approximation algorithm uses the $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition, which is designed to ensure topologically correct approximation with a bounded two-sided Hausdorff error. For the purpose of penetration depth estimation, preserving topology is not necessary: It is sufficient to compute an approximation with a bounded two-sided Hausdorff error. Hence we can relax the sampling condition; we can drop $\mathcal{C}^{\square}$ and $\mathcal{C}^{\bigstar}$ conditions, and merely use the $\mathcal{C}^{\epsilon}$ condition.

4.4 Results

In this section, we highlight the performance of our approximate algorithm on different input models. We implemented our algorithm on a 2 GHz Pentium IV PC with 1 GB main memory.

We have tested our algorithm on a number of complex models. The model complexity (Table 4.1) varied from several hundred to a few thousand triangles. Figs. 1.12 and 4.2 show some of our results on offset computation.

Figure 1.11 shows the Minkowski sum of *Brake Hub* and *Rod* models. The final Minkowski sum has a number of narrow tunnels. Our algorithm produced an approximation that preserved these features. Fig. 4.4 shows the Minkowski sum of a number of CAD models.

Fig. 4.5 shows a complex example consisting of two *Grates*. This is a very challenging scenario as the resulting Minkowski sum has a large number of complex features. It has numerous thin and needle-like features. Our algorithm is able to reconstruct these features.

Fig. 1.14 shows an application of Minkowski sum to 3D translational motion planning. Table 4.1 shows the model complexity and performance of our algorithm on these models. The culling techniques improve the performance significantly. We applied our algorithm without any culling techniques to the Anvil and Spoon model (Figure 4.2). It took more than 7 hours to generate an approximation. In comparison, using culling techniques, our algorithm was able to produce an approximation in just 63 secs.

Primitive 1			Primitive 2			Num	Performance (secs)			Output
	Tris	Pieces		Tris	Pieces	Prims	Convex	Samp	Recons	Tris
Cup	1000	338	Sphere	-	1	338	1.2	32	0.08	14,895
Gear	2,382	744	Sphere	-	1	744	3.6	81	0.09	22,742
Brake Hub	4,736	1777	Rod	24	1	1,777	4.3	135	0.04	45,753
Anvil	144	57	Spoon	336	78	4,446	3.9	59	0.02	15,638
Wrench	772	291	Spiral	500	133	38,703	27	289	0.06	25,280
Knife	516	273	Scissors	636	230	62,790	36	742	0.06	26,038
Grates 1	444	163	Grates 2	1134	409	66,667	40	3120	1.5	358,030

Table 4.1: Performance: *This table shows the performance of our algorithm on different models. The columns on the left show the statistics of the two primitives whose Minkowski sum is computed. They show the number of triangles in each primitive and the number of convex pieces generated by convex decomposition. The column, Num Prims, shows the number of convex Minkowski sums generated. The right three columns show the time taken to generate the convex Minkowski sums, sampling and isosurface reconstruction.*

4.5 Summary and Limitations

We have presented an algorithm to approximate the 3D Minkowski sum of polyhedral objects. Our algorithm provides geometric and topological guarantees on the approximation. We employ cell and primitive culling techniques to improve the performance of our algorithm. We have applied our algorithm to offset computation and motion planning of robots with translational degrees of freedom.

4.5.1 Limitations

A bottleneck in our algorithm is the convex decomposition method. Typically, it produces $O(n)$ convex pieces. Given two polyhedra each with n triangles, we usually obtain $O(n^2)$ pairwise convex Minkowski sums whose union needs to be computed. Since this set of pairwise convex Minkowski sums is an input to our approximation algorithm, its large size impacts the performance of the overall algorithm. Although our algorithm is capable of handling a high number of primitives, the large number of pairwise Minkowski sums lowers the overall performance. Using a better convex decomposition method will alleviate this problem.

Chapter 5

Free Space Approximation and Complete Motion Planning

We introduced the configuration space formulation (Lozano-Pérez, 1983) in Chapter 1. In this formulation, each position and orientation of the robot $\mathcal{R}$ maps to a point in the configuration space $\mathcal{C}$. Each obstacle $\mathcal{O}_i$, for $i = 1, \ldots, n$ maps to a region

$$\mathcal{CO}_i = \{\mathbf{q} \in \mathcal{C} : \mathcal{R}(\mathbf{q}) \cap \mathcal{O}_i \neq \emptyset\},$$

in $\mathcal{C}$, where $\mathcal{R}(\mathbf{q})$ is the subset of W occupied by $\mathcal{R}$ at the configuration $\mathbf{q}$.

The union of all the $\mathcal{CO}_i$'s, $\bigcup_{i=1}^{n} \mathcal{CO}_i$, is called *$C$-obstacle region* or *forbidden region*. The set

$$\mathcal{F} = \mathcal{C} \setminus \bigcup_{i=1}^{n} \mathcal{CO}_i.$$

is called the *free configuration space* or the *free space*.

This chapter addresses the problems of free space computation and motion planning. In free space computation, our goal is to compute $\partial\mathcal{F}$ – the boundary of the free space. We assume that $\partial\mathcal{F}$ is an orientable closed manifold. Given an initial configuration $\mathbf{q}_{init}$ and a goal configuration $\mathbf{q}_{goal}$, the objective of motion planning is to find a *collision-free path* – a continuous map $\tau : [0, 1] \rightarrow \mathcal{F}$ with $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$.

Free Space Approximation

It is well known that for a translating robot $\mathcal{R}$ and an obstacle $\mathcal{O}$, the C-obstacle can be expressed as a Minkowski sum: $\mathcal{CO} = \mathcal{O} \oplus (-\mathcal{R})$ (Lozano-Pérez, 1983). This formulation shrinks the robot to a point, and the obstacle $\mathcal{O}_i$ is transformed to the respective Minkowski sum $\mathcal{CO}_i$. The free space boundary $\partial\mathcal{F}$ can be approximated

using our Minkowski sum approximation algorithm (Chapter 4).

In this chapter, we present an algorithm for approximating the free space of robots with translational and rotational degrees of freedom (DOF). We represent the free space in terms of *contact surfaces* (C-surfaces). A C-surface of a geometric feature (vertex, edge, face) of $\mathcal{R}$ and a similar feature (vertex, edge, face) of $\mathcal{O}$ is defined as the set of points in the configuration space that represent configurations of $\mathcal{R}$ at which contact is made between these specific features. The set Γ of contact surfaces define an arrangement $\mathcal{A}(\Gamma)$. The free space $\mathcal{F}$ consists of some cells in this arrangement. Therefore, $\partial\mathcal{F}$ can be computed by computing $\mathcal{A}(\Gamma)$. However, because there are a large number of C-surfaces in $\mathcal{A}(\Gamma)$ and the the C-surfaces are non-linear, it is difficult to compute $\mathcal{A}(\Gamma)$ robustly. Therefore, instead of exact computation of $\partial\mathcal{F}$, we compute an approximation $\mathcal{A}$ to $\partial\mathcal{F}$ using our isosurface extraction algorithm based on $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition (Chapter 3). To apply the $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition, we need to perform complex cell and star-shaped tests on $\partial\mathcal{F}$. We present computational techniques to perform these tests without explicitly computing $\partial\mathcal{F}$. The geometric and topological guarantees of our isosurface extraction algorithm apply to $\partial\mathcal{F}$ as well: provided $\mathcal{C}^{\square\bigstar\epsilon}$ is satisfied, $\mathcal{A}$ is geometrically close and topologically equivalent to $\partial\mathcal{F}$. We have implemented the algorithm and applied it to compute free space approximation for planar robots with translational and rotational degrees of freedom.

Complete Motion Planning

An immediate application of the free space computation algorithm is to the problem of motion planning. We assume that the robot – a rigid or articulated object – is the only moving object in a static workspace. The obstacles are rigid and static. The robot may translate, rotate, or have different types of joints imposing sliding or rotating constraints. The geometry of both the robot and the obstacles is known.

The $\mathcal{C}^{\square\bigstar\epsilon}$ sampling condition is designed to compute a bounded-error and topologically correct approximation to the free space. However, for motion planning, this is not necessary: It suffices to capture only the connectivity of the free space. The star-shaped criterion is sufficient for this purpose. Hence we drop the $\mathcal{C}^{\square}$ and $\mathcal{C}^{\epsilon}$ criteria, and propose a relaxed sampling condition based only on $\mathcal{C}^{\bigstar}$.

We present a new deterministic sampling based algorithm for complete motion planning. Our algorithm is based on computing a *star-shaped roadmap* of the free space. The roadmap is constructed by computing a star-shaped decomposition of the free space. This produces a set of *guards* that capture the *intra-region* connectivity – the

connectivity between points belonging to the same star-shaped region. The *inter-region* connectivity is captured by computing *connectors* that connect guards of adjacent regions. The guards and connectors are combined to obtain the star-shaped roadmap.

We present an adaptive subdivision algorithm for constructing the star-shaped roadmap without explicit computation of the free space. The adaptive subdivision algorithm applies the star-shaped criterion in a recursive manner. In the absence of degeneracies in the free space, the adaptive subdivision algorithm will terminate and produce a star-shaped roadmap as the output. The resulting star-shaped roadmap captures the complete connectivity of the free space, thus enabling complete motion planning. The roadmap can be used not only to find a path, but also detect non-existence of any collision-free path.

The subdivision algorithm cannot handle degenerate cases such as *tangential contacts*, when two C-obstacles intersect each other "tangentially". The algorithm will not terminate in such cases. We analyze the behavior of the subdivision algorithm and present sufficient conditions for its termination. We also discuss the issue of degeneracies further and suggest possible ways of dealing with them.

The underlying computation in our planner is the star-shaped test. We perform this test efficiently using a conservative technique that reduces to linear programming and interval arithmetic. Unlike prior criticality-based complete algorithms (Latombe, 1991), our algorithm is able to avoid exact computation of roots of algebraic equations. Thus, it is relatively simple to implement.

We compare some features of our deterministic sampling algorithm with randomized sampling based algorithms and approximate cell-based decomposition methods. We have implemented our algorithm and applied it to compute collision free paths for low DOF robots in challenging scenarios with narrow passages and no collision-free paths.

Organization: Section 1 presents the notation used in the chapter. Section 2 describes a free space formulation in terms of C-surfaces. Appendix B provides background on C-surfaces. It describes methods for enumerating C-surfaces for two classes of robots: **2T+1R** - a planar rigid robot with 2 translational and 1 rotational dofs moving among polygonal obstacles, and **3R** - a planar articulated robot with 3 revolute joints moving among polygonal obstacles.

Section 3 presents computational techniques to answer certain queries required by the free space approximation and motion planning algorithms. Section 4 presents the free space approximation algorithm.

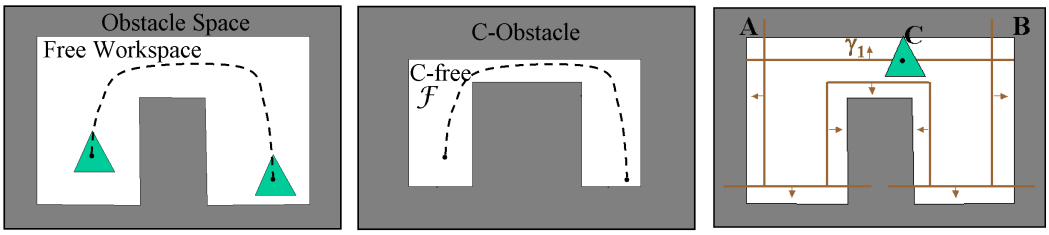
Sections 5 and 6 present the star-shaped roadmap method for motion planning; Section 5 presents the concept of star-shaped roadmaps, while Section 6 describes an adaptive subdivision algorithm for roadmap construction. Section 7 analyzes the behavior of the adaptive subdivision algorithm and presents sufficient conditions for its termination.

Section 8 describes the implementation of our free space approximation and motion planning algorithms, and demonstrates their performance on several models. Section 9 compares the star-shaped roadmap method with several prior motion planning methods. Section 10 discusses limitations of our algorithms. Section 11 concludes the chapter.

5.1 Notation and Preliminaries

$\mathcal{R}$ denotes a robot consisting of a collection of rigid subparts moving in a Euclidean space W, called the *workspace*, represented as $\mathbb{R}^d$. Let $\mathcal{O}_1, \ldots, \mathcal{O}_q$ be fixed rigid obstacles embedded in W. Assume that the geometry of $\mathcal{R}, \mathcal{O}_1, \ldots, \mathcal{O}_q$ is accurately known, and that there are no kinematic constraints to limit the motion of $\mathcal{R}$. The position and orientation of the subparts define the *configuration* of $\mathcal{R}$. The set of all configurations of $\mathcal{R}$ defines the configuration space $\mathcal{C}$. Let $\mathcal{C}$ be k-dimensional. $\mathcal{F}$ denotes the free configuration space, hereafter known as the free space. $\partial\mathcal{F}$ denotes its boundary. We assume that $\partial\mathcal{F}$ is a $(k-1)$-dimensional orientable closed manifold. $\mathcal{R}(\mathbf{q})$ denotes the subset of W occupied by $\mathcal{R}$ at a configuration $\mathbf{q}$. A region R will refer to a k-dimensional connected subset of $\mathcal{C}$. Recall the definition of the star-shaped property for a k-dimensional region R; R is said to be *star-shaped* if there exists a point $\mathbf{o} \in R$ such that $\mathbf{op} \subseteq R \; \forall \; \mathbf{p} \in R$ where $\mathbf{op}$ denotes the line segment between points $\mathbf{o}$ and $\mathbf{p}$ (including both endpoints). Point $\mathbf{o}$ is referred to as a *guard*.

Given a set S, two points $\boldsymbol{p}, \boldsymbol{q} \in S$ are connected if there exists a path between $\boldsymbol{p}$ and $\boldsymbol{q}$ that lies in S. We use the shorthand notation $\boldsymbol{p} \overset{S}{\longleftrightarrow} \boldsymbol{q}$ to mean $\boldsymbol{p}$ and $\boldsymbol{q}$ are connected in S. The connectivity relation is symmetric. Given a roadmap (an undirected graph) $\mathcal{R} = (V, E)$ and two vertices $\boldsymbol{v}, \boldsymbol{w} \in V$, $\boldsymbol{v} \overset{\mathcal{R}}{\longleftrightarrow} \boldsymbol{w}$ means that $\boldsymbol{v}$ and $\boldsymbol{w}$ are connected in $\mathcal{R}$, i.e., there exists a path between $\boldsymbol{v}$ and $\boldsymbol{w}$ consisting of a sequence of edges in E. We use $\boldsymbol{p} \overset{S}{\leftarrow\!\!/\!\!\rightarrow} \boldsymbol{q}$ to mean $\boldsymbol{p}$ and $\boldsymbol{q}$ are not connected in S. $\boldsymbol{v} \overset{\mathcal{R}}{\leftarrow\!\!/\!\!\rightarrow} \boldsymbol{w}$ is defined similarly.



(a) Workspace (b) Configuration Space (c) Contact Surfaces

Figure 5.1: Configuration Space Formulation: *Fig. (a) shows a robot (green) navigating in a 2D workspace. The robot is capable of only translation. Fig. (b) shows the two dimensional configuration space of the robot. Fig. (c) shows the contact surfaces (brown line segments) that arise from the contact between features of the robot and the obstacle. For example, γ_1 is generated as a result of a contact between the top vertex of the robot (**C**) and edge **AB** of the obstacle. We can assign an orientation to the contact surfaces (indicated by the arrows) to "point towards C-obstacle".*

5.2 Free Space Representation

The free space $\mathcal{F}$ can be described in terms of different types of contact between the robot $\mathcal{R}$ and obstacle $\mathcal{O}$. Note that $\mathcal{R}$ and $\mathcal{O}$ are in contact if

$$\mathcal{R}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset \quad \wedge \quad \text{int } \mathcal{R}(\mathbf{q}) \cap \text{int } \mathcal{O} = \emptyset$$

There can be different types of contacts depending on the type of features of the robot and the obstacle that are in contact. For example, in the case of a polygonal robot navigating among polygonal obstacles along a plane, two types of contacts are possible: a vertex of $\mathcal{R}$ may be in contact with an edge of an obstacle or an edge of $\mathcal{R}$ may be in contact with a vertex of an obstacle. If we displace $\mathcal{R}$ in such a way that the contact between the pair of features is maintained, then the robot's configuration moves along a surface in $\mathcal{C}$ called a *contact surface* (C-surface). A C-surface is generated for every pair of features of the robot and the obstacle that can be involved in contact. In the general case of a d-dimensional configuration space, a C-surface is a $(d-1)$-dimensional manifold. An overview of contact surfaces is given in Appendix B.

We note two important properties of C-surfaces:

1. **Superset property**: The set Γ of C-surfaces define an arrangement in $\mathcal{C}$. $\mathcal{F}$ is a collection of cells in this arrangement; a cell corresponding to a connected

component of $\mathcal{F}$. Furthermore, Γ is a superset of the boundary $\partial\mathcal{F}$ of free space, i.e., $\partial\mathcal{F} \subseteq \bigcup\{\gamma_i \in \Gamma\}$. See Fig. 5.1.

2. **Orientation property**: We can assign an orientation to the C-surfaces. Each C-surface γ is a subset of an algebraic surface represented as a zero-set of an algebraic function $h_\gamma : \mathcal{C} \rightarrow \mathbb{R}$. Suppose γ corresponds to a contact between a robot feature f_1 and an obstacle feature f_2. Then for a point $\mathbf{q} \in \gamma$, we have $h_\gamma(\mathbf{q}) = 0$ and $\mathbf{q}$ is a robot configuration where f_1 and f_2 are in contact. Consider a point $\mathbf{r}$ in a small neighborhood of $\mathbf{q}$. If $h(\mathbf{r}) < 0$, then $\mathbf{r}$ is a robot configuration where f_1 and f_2 overlap each other. Therefore, $\mathbf{r}$ belongs to C-obstacle. On the other hand, in the case where $h(\mathbf{r}) > 0$, $\mathbf{r}$ is a robot configuration where there is no overlap or contact between f_1 and f_2. We orient γ by defining a normal at each point $\boldsymbol{q} \in \gamma$ as follows:

$$\mathbf{n}(\mathbf{q}) = -\frac{\bigtriangledown h(\mathbf{q})}{\| \bigtriangledown h(\mathbf{q})\|}, \quad \bigtriangledown h(\mathbf{q}) \neq 0$$

The normal $\mathbf{n}(\mathbf{q})$ "points towards C-obstacle", i.e., in the direction of overlap between f_1 and f_2. See Fig. 5.1(c).

$\partial\mathcal{F}$ can be obtained by computing the arrangement of Γ. The set of $(d-1)$-dimensional cells in the arrangement provides a partition of the C-surfaces into a set of *surface components*. Given such a partition, we can combine a subset of the surface components to obtain $\partial\mathcal{F}$. However, this is not feasible in practice due to the difficulty of arrangement computation. Therefore, we avoid explicit free space computation. Instead, we perform a deterministic sampling of the free space.

5.3 Supporting Queries

Our algorithms for free space approximation and motion planning are based on the complex cell and star-shaped tests (Sec. 3.5). We generate an adaptive volumetric grid in $\mathcal{C}$ by performing the complex cell and star-shaped tests on $\partial\mathcal{F}$. These tests rely on a number of queries: sign query, star-shaped query, and cell intersection query. In Sec. 3.5, we presented computational techniques to answer these queries. Those techniques took advantage of the fact that the surface was defined as a Boolean combination over a set of primitives. We adapt these techniques to apply them to the case where the surface of interest $\partial\mathcal{F}$ is defined differently – in terms of an arrangement of a set of

C-surfaces. We answer these queries without computing an explicit representation of $\partial\mathcal{F}$.

5.3.1 Sign Query

Given a point $\mathbf{q} \in \mathcal{C}$, the sign query determines whether $\mathbf{q}$ belongs to the free space, i.e., if $\mathbf{q} \in \mathcal{F}$. The definition of the free space reduces this query to a collision check between $\mathcal{R}(\mathbf{q})$ and all the obstacles (Lin and Manocha, 2003): $\mathbf{q} \in \mathcal{F}$ if and only if $\mathcal{R}(\mathbf{q})$ does not intersect any obstacle. For a **2T+1R** robot, the query reduces to a collision check between two polygons. For a **3R** robot, it reduces to a collision check between a line segment and a polygon.

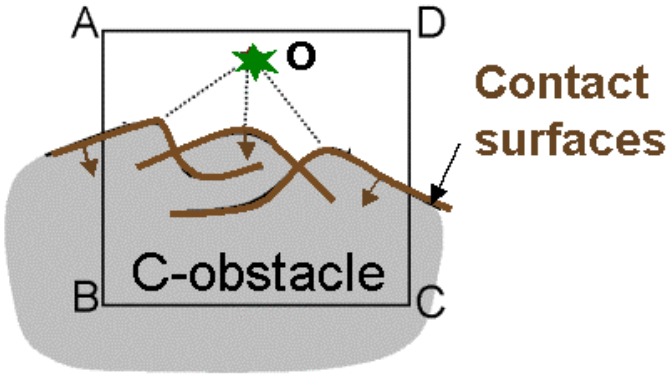


Figure 5.2: Star-shaped Test: *If in a cell C, all the C-surfaces satisfy the contact surface condition (Equation 5.1), then $\mathcal{F} \cap C$ is star-shaped w.r.t* **o**. *The arrows indicate the orientation of the C-surfaces.*

5.3.2 Star-shaped Query

Consider a grid cell C in the configuration space. Let S be a voxel or a face of C. The star-shaped query is slightly different for free space computation and motion planning.

- **Star-shaped query for free space computation:** Is $\partial\mathcal{F}$ star-shaped w.r.t S? This query is analagous to the query introduced in Sec. 3.5.2.
- **Star-shaped query for motion planning:** Is $\mathcal{F}_S = \mathcal{F} \cap S$ star-shaped w.r.t a point in $\mathcal{F}_S$? This query imposes a restriction that the guard of $\mathcal{F}_S$ belong to $\mathcal{F}_S$.

We present a common conservative test to answer the query in both cases. We exploit the fact that the C-surfaces form a superset of $\partial\mathcal{F}$ (Superset property, Sec. 5.2). This reduces the problem to performing star-shaped tests on the C-surfaces.

Contact Surface Test

Let Γ denote the set of all C-surfaces. For each C-surface $\gamma_i \in \Gamma$ that intersects S, compute the restriction $\gamma_{i,S} = \gamma_i \cap S$ to S. Let Γ_S denote the resulting set of surfaces. We can answer the star-shaped query provided S satisfies the following condition: **Contact Surface Condition:**

Is there a point $\boldsymbol{o} \in S$ such that for each $\gamma_{i,S} \in \Gamma_S$ the following holds:

$$\text{For each } \mathbf{x} \in \gamma_{i,S} \text{ with normal } \mathbf{n_x} \text{ we have } \mathbf{ox} \cdot \mathbf{n_x} > 0 \qquad (5.1)$$

See Fig. 5.2.

If S satisfies the above condition, then we can answer the query. This is formally stated as the following theorem.

THEOREM 10 Star-shaped test*: Suppose S satisfies the contact surface condition (Equation 5.1). Then*

1. *$\mathcal{F}_S \neq \emptyset \Longleftrightarrow \mathbf{o} \in \mathcal{F}$.*
2. *If $\mathbf{o} \in \mathcal{F}$, then $\mathcal{F}_S$ is star-shaped w.r.t $\mathbf{o}$.*
3. *If $\partial\mathcal{F}_S \neq \emptyset$, then $\partial\mathcal{F}_S$ is star-shaped w.r.t $\mathbf{o}$.*

Proof: We first prove 1). $\boldsymbol{o} \in \mathcal{F} \implies \boldsymbol{o} \in \mathcal{F}_S \implies \mathcal{F}_S \neq \emptyset$. We now prove the converse. Assume $\mathcal{F}_S \neq \emptyset$. We show that $\boldsymbol{o} \in \mathcal{F}$. Consider a point $\mathbf{p} \in \mathcal{F}_\mathbf{S}$. We prove that $\boldsymbol{o}$ can see $\boldsymbol{p}$, i.e., line segment $\boldsymbol{po}$ lies completely in $\mathcal{F}$. We prove this by contradiction. Suppose $\boldsymbol{o}$ does not see $\boldsymbol{p}$. In other words, the line segment $\boldsymbol{po}$ intersects the free space boundary $\partial\mathcal{F}$. The superset property of C-surfaces implies that the line segment $\boldsymbol{po}$ is intersected by a C-surface γ_k. Let $\boldsymbol{q} \in \gamma_k$ be the intersection point on the line segment $\mathbf{po}$ that is closest to $\mathbf{p}$. Since $\mathbf{q}$ lies on a C-surface and $\mathcal{F}$ is a open set, we have $\mathbf{q} \notin \mathcal{F}$. Suppose we start at point $\mathbf{p} \in \mathcal{F}$ and march along the ray $\vec{\mathbf{po}}$. Then $\mathbf{q}$ is the point closest to $\mathbf{p}$ that does not belong to $\mathcal{F}$. Hence $\mathbf{q}$ belongs to $\partial\mathcal{F}$. Moreover, The orientation property of C-surfaces ensures that the normal $\mathbf{n_q}$ at $\mathbf{q}$ "points towards C-obstacle". Since $\mathbf{p} \in \mathcal{F}$, $\mathbf{qp}$ makes an obtuse angle with $\mathbf{n_q}$. Therefore, we have $\mathbf{qp} \cdot \mathbf{n_q} < \mathbf{0}$. This implies $\mathbf{oq} \cdot \mathbf{n_q} < \mathbf{0}$ which contradicts the contact surface condition. Thus $\boldsymbol{po}$ lies completely in $\mathcal{F}$ which means $\boldsymbol{o} \in \mathcal{F}$.

Because $\mathbf{o}$ can see any point $\mathbf{p} \in \mathcal{F}$, it means that $\mathcal{F}_S$ is star-shaped w.r.t $\mathbf{o}$. Furthermore, if $\partial\mathcal{F}_S \neq \emptyset$, then for any point $\mathbf{q} \in \partial\mathcal{F}_S$, we have $\mathbf{oq} \cap \partial\mathcal{F} = \{\mathbf{q}\}$. In other words, $\partial\mathcal{F}_S$ is star-shaped w.r.t $\mathbf{o}$.

Theorem 10 reduces the star-shaped query on $\partial\mathcal{F}$ to verifying the contact surface condition for the contact surfaces. We perform these tests using a combination of linear programming and interval arithmetic (Secs. 3.5.2 3.5.4).

Conservativeness of the Star-shaped Test

There are two possible sources of conservativeness in the above test.

1. The contact surface condition is only a sufficient condition for when $\partial\mathcal{F}$ is star-shaped. It is possible for $\mathcal{F}$ to be star-shaped w.r.t C even if the contact surface condition is not satisfied. The main advantage of the test is that it does not require an explicit representation of $\mathcal{F}$.

2. When the C-surfaces are non-linear, we use a discretization technique to verify the contact surface condition (Sec. 3.5.4). This technique enumerates a set of samples on the C-surfaces to estimate a candidate point and then verifies if the contact surface condition holds for the candidate point. This test may fail due to a poor estimate of the candidate point.

Due to the conservativeness of these tests, it is possible for a cell C to fail the tests even though $\partial\mathcal{F}$ may be star-shaped w.r.t C. If C fails the above tests, then we subdivide C and repeat the tests on the subdivided cells. As a result, we preserve the correctness of the algorithm. These conservative tests may, however, result in some additional subdivision.

5.3.3 Free Space Existence Query

This query answers whether $\mathcal{F}$ intersects S, i.e., if $\mathcal{F}_S \neq \emptyset$. In general, this query is difficult without an explicit representation of $\mathcal{F}$. We answer this query in two special cases:

1. One or more of the vertices of S belongs to $\mathcal{F}$.

2. S satisfies the contact surface condition. In this case, Theorem 10. provides the following test: $\mathcal{F}_S \neq \emptyset \iff \mathbf{o} \in \mathcal{F}$.

5.3.4 Cell Intersection Query

Voxel/Face Intersection Query: This query answers whether $\partial\mathcal{F}$ intersects S, i.e., if $\partial\mathcal{F}_S \neq \emptyset$. We answer the intersection query in a special case – when S satisfies the

contact surface condition. In this case, we use a test based on the following corollary of Theorem 10.

COROLLARY 11 Cell Intersection query: *Suppose S satisfies the contact surface condition. Then*

$$\mathbf{o} \in \mathcal{F} \quad \wedge \quad \Gamma_S \neq \emptyset \quad \iff \quad \partial\mathcal{F}_S \neq \emptyset$$

Proof: Suppose $\mathbf{o} \in \mathcal{F} \quad \wedge \quad \Gamma_S \neq \emptyset$. We prove $\partial\mathcal{F}_S \neq \emptyset$ by contradiction. Suppose $\partial\mathcal{F}_S = \emptyset$. This means either $\mathcal{F}_S \subseteq \mathcal{F}$ or $\mathcal{F}_S = \emptyset$. Because S satisfies the contact surface condition and $\mathbf{o} \in \mathcal{F}$, Theorem 10 implies that $\mathcal{F}_S \neq \emptyset$. Therefore $S \subseteq \mathcal{F}$. But this means every point $\mathbf{q} \in S$ corresponds to a collision-free configuration of the robot $\mathcal{R}$. In other words, no point in S belongs to a C-surface. This contradicts the fact that $\Gamma_S \neq \emptyset$.

The proof of the converse follows from the superset property and Theorem 10.

□

To check if $\Gamma_S \neq \emptyset$, we need to test if any C-surface in Γ intersects S. This can be done using either max-norm distance computation or interval arithmetic techniques described in Sec. 3.5.1.

Edge Intersection Query: Consider an edge e with endpoints $\mathbf{a}$ and $\mathbf{b}$. This query computes the number of points at which $\partial\mathcal{F}$ intersects e. If the number of intersection points is greater than 0, then e is intersected by $\partial\mathcal{F}$.

We exploit the superset property of contact surfaces. We compute the intersection of e with all the contact surfaces. Let $I = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_k}\}$ denote the resulting set of intersection points. Since the contact surfaces are a superset of $\partial\mathcal{F}$, some of these intersection points may not belong to $\partial\mathcal{F}$. We perform a test on each $\mathbf{p_i}$ to check if it belongs to $\partial\mathcal{F}$.

Assume that the intersection points in I are sorted along the edge: Point $\mathbf{p_i}$ is closer to $\mathbf{a}$ than $\mathbf{p_{i+1}}$. Define $\mathbf{p_0} = \mathbf{a}$ and $\mathbf{p_{k+1}} = \mathbf{b}$. Compute a set of points $\{\mathbf{q_0}, \ldots, \mathbf{q_k}\}$ where $\mathbf{q_i} = (\mathbf{p_i} + \mathbf{p_{i+1}})/2$. Point $\mathbf{p_i}$ belongs to $\partial\mathcal{F}$ if either $\mathbf{q_{i-1}} \in \mathcal{F}$ or $\mathbf{q_i} \in \mathcal{F}$, which can be tested using the sign query.

5.4 Free Space Approximation Algorithm

We compute an approximation to $\partial\mathcal{F}$ using an adaptive subdivision algorithm that is almost identical to the one previously presented in Sec. 3.5. Like the previous

algorithm, the current algorithm performs both complex cell test and star-shaped test on the grid cells. There is, however, one important difference. Unlike the previous algorithm, we now impose an order in which the two tests must be applied: We require that the star-shaped test be executed before the complex cell test.

5.4.1 Star-shaped Test

The star-shaped test performs two tests on $\partial\mathcal{F}$ – (a) star-shaped w.r.t voxel, and (b) star-shaped w.r.t. each face. We use the star-shaped query presented in Sec. 5.3.2 to perform these tests.

If any of these tests results in the negative, we subdivide the cell and apply the algorithm recursively to the new cells.

5.4.2 Complex Cell Test

We perform the complex cell test on a cell C only when C has already satisfied the star-shaped test. This means C satisfies the contact surface condition (Equation 5.1). Hence we can use Corollary 11 to perform the cell intersection query on C. We take advantage of this fact while performing the complex cell test.

To check whether a cell is complex, we perform the following tests:

- **Complex Voxel/Face**: We use the cell intersection query to check whether $\partial\mathcal{F}$ intersects a voxel or face of the cell. If $\partial\mathcal{F}$ intersects the voxel (face), then we determine if the voxel (face) is complex by checking for a sign change at the cell vertices. The signs at the cell vertices are computed using the sign query. If $\partial\mathcal{F}$ does not intersect the voxel (face), then the voxel (face) is not considered complex.

- **Complex Edge**: We use the edge intersection query to test if an edge is complex. An edge is complex if the $\partial\mathcal{F}$ intersects the edge in more than one point.

- **Ambiguity**: We use the signs at the grid vertices to resolve cases corresponding to face and voxel ambiguity.

If any of these tests results in the affirmative, the cell is complex, and we subdivide it and apply the algorithm recursively to the new cells.

5.4.3 Geometric and Topological Guarantees

The adaptive subdivision algorithm generates a volumetric grid $\mathcal{G}$ on which we apply isosurface extraction. The output of isosurface extraction is an approximation $\mathcal{A}$ to $\partial\mathcal{F}$. The geometric and topological guarantees on $\mathcal{A}$ follow from Theorem 1: If every cell in $\mathcal{G}$ satisfies $\mathcal{C}^{\square\bigstar}$, then $\mathcal{A}$ is topologically equivalent to $\partial\mathcal{F}$. Furthermore, we augment the subdivision algorithm with the Hausdorff criterion (Sec. 3.8), and bound the two-sided Hausdorff error between $\mathcal{A}$ and $\partial\mathcal{F}$. Therefore, we have the following result.

THEOREM 12 *If every cell in $\mathcal{G}$ satisfies $\mathcal{C}^{\square\bigstar_\epsilon}$, then*

1. **Geometric Guarantee:** $H(\mathcal{A}, \partial\mathcal{F}) < \epsilon$.
2. **Topological Guarantee:** $\mathcal{A} \approx \partial\mathcal{F}$.

5.5 Star-shaped Roadmaps for Complete Motion Planning

In this section, we present the concept of a star-shaped roadmap, and show that it captures the connectivity of the free space, thus enabling complete motion planning. We present a deterministic algorithm for constructing a star-shaped roadmap in Sec. 5.6.

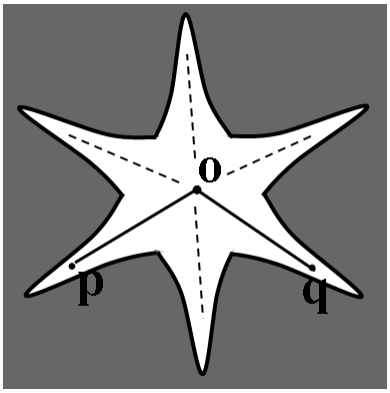
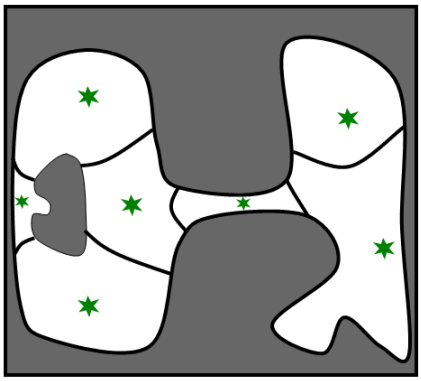


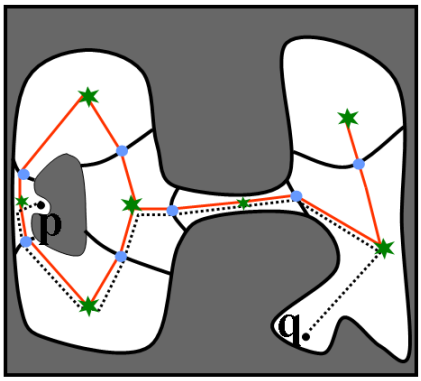
Figure 5.3: Star-shaped property: *This figure shows a star-shaped region (in white). It contains a guard $\boldsymbol{o}$ that can see every point within the region. A path between any two points $\boldsymbol{p} \in R$ and $\boldsymbol{q} \in R$ is given by $\boldsymbol{po} :: \boldsymbol{oq}$.*

5.5.1 Star-shaped Property and Motion Planning

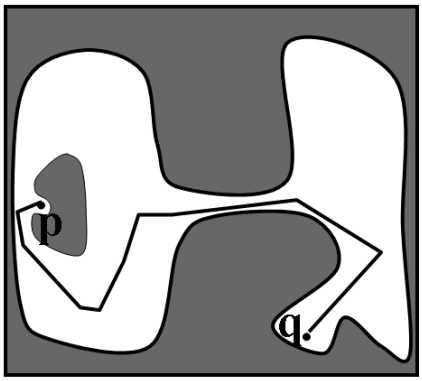
A region R is *star-shaped* if there exists a point $\boldsymbol{o} \in R$, a *guard*, that can *see* every point $\boldsymbol{p}$ in the region, i.e., the straight line segment $\boldsymbol{op} \subseteq R$. It is easy to show that a star-shaped region is always connected. Moreover, every point in the region is connected to the guard along a straight line segment. Thus, the star-shaped property is a compact way of encoding the connectivity of a region. It provides a path between every point in the region and the guard. We exploit this property for motion planning. A path between any two points $\boldsymbol{p} \in R$ and $\boldsymbol{q} \in R$ is given by $\boldsymbol{po} :: \boldsymbol{oq}$ where :: denotes path concatenation (see Fig. 5.3(a)). We extend this idea to compute a path between two arbitrary configurations in free space.



(a) Star-Shaped Decomposition



(b) Star-Shaped Roadmap



(c) Path Planning

Figure 5.4: Star-shaped Roadmap: *This figure shows how to construct a star-shaped roadmap and its application to path planning. The C-obstacle is shown in gray while the free space is shown in white. We first compute a star-shaped decomposition of the free space (Fig. (a)). Each region in the decomposition contains a guard (green star) that can see every point in the region. We connect guards of adjacent regions by computing connectors (blue circles) on the common boundary between the two regions. The guards and connectors are used to create the star-shaped roadmap as shown in Fig. (b). Fig (c) shows how a path is computed between two points $\boldsymbol{p}$ and $\boldsymbol{q}$ by connecting them to the roadmap and finding a path along the roadmap.*

5.5.2 Overall Approach

Our method relies on a *star-shaped decomposition* of the free space, i.e., a partition of $\mathcal{F}$ into a set of star-shaped regions. We show how to compute such a partition in Sec. 5.6. The guards of the star-shaped regions capture the intra-region connectivity. However, we also need to take into account the inter-region connectivity, i.e., the connectivity between points belonging to separate regions. We achieve this by computing *connectors*[1]. Our method consists of the following steps:

1. Compute a star-shaped decomposition Σ of the free space into a set of star-shaped regions $\{R_1, \ldots, R_n\}$.

2. For every pair of adjacent regions (R_i,R_j) in Σ, compute a point $\boldsymbol{c}$ on the common boundary shared by R_i and R_j. We refer to $\boldsymbol{c}$ as a *connector* – it connects the guards of R_i and R_j.

3. Construct a star-shaped roadmap $\mathcal{R}$ using the guards and connectors computed in Steps 1 and 2.

These steps are illustrated in Fig. 5.4.

5.5.3 Star-shaped Decomposition and Guard Computation

Step 1 computes a star-shaped decomposition of the free space. The resulting set of guards constitutes a sampling of the free space and we refer to it as a *star-shaped* sampling of the free space. The star-shaped sampling provides an implicit representation of the free space.

$$\boldsymbol{p} \in \mathcal{F} \Longleftrightarrow \boldsymbol{p} \text{ is visible to at least one of the guards.}$$

The concept of star-shaped decomposition is related to the famous art gallery problem (O'Rourke, 1987). The art gallery problem is concerned with finding the minimum number of guards that can cover a region. In our context, computing a smaller number of guards would be desirable, but not necessary.

5.5.4 Connector Computation

In Step 2, we capture the inter-region connectivity. It suffices to only consider paths between adjacent regions R_i and R_j that cross their common boundary R_{ij}. We com-

pute a point $\boldsymbol{c}$ belonging to R_{ij}. $\boldsymbol{c}$ is a connector. Since the regions R_i and R_j are star-shaped, $\boldsymbol{c}$ is visible to the guards of R_i and R_j. Hence, $\boldsymbol{c}$ connects the guards of two adjacent regions (see Fig. 5.4(b)).

5.5.5 Roadmap Computation

In Step 3, we combine the guards and connectors to construct a star-shaped roadmap $\mathcal{R}$ of the free space (see Fig. 5.4(b)). $\mathcal{R}$ is an undirected graph. Let V_G and V_C denote the set of guards and connectors respectively. The set of graph vertices is $V = V_G \cup V_C$. Each connector $\boldsymbol{c}$ connects two guards $\boldsymbol{g_1} \in V_G$ and $\boldsymbol{g_2} \in V_G$ of two adjacent regions. This defines two graph edges $(\boldsymbol{c}, \boldsymbol{g_1})$ and $(\boldsymbol{c}, \boldsymbol{g_2})$. Let $GUARDS(\boldsymbol{c})$ denote the set $\{\boldsymbol{g_1}, \boldsymbol{g_2}\}$. The set of graph edges E is defined as:

$$E = \{(\boldsymbol{c}, \boldsymbol{g}) \mid \boldsymbol{c} \in V_C,\ \boldsymbol{g} \in GUARDS(\boldsymbol{c})\}$$

$\mathcal{R}$ is the undirected graph (V, E, w) where the weight function $w : E \rightarrow \mathbb{R}$ is defined as a distance between the edge vertices using a suitable metric (e.g. Euclidean).

5.5.6 Complete Motion Planning

Given a star-shaped decomposition Σ and the roadmap $\mathcal{R}$, path planning becomes straightforward. Let $\boldsymbol{p}$ and $\boldsymbol{q}$ respectively denote the initial and goal configuration respectively. Assume they are connected. The star-shaped property of each region in Σ implies we can connect $\boldsymbol{p}$ and $\boldsymbol{q}$ to guards $\boldsymbol{p}^\star$ and $\boldsymbol{q}^\star$ respectively by straight line paths. In general, there may be more than one guard $\boldsymbol{p}^\star$ that may see $\boldsymbol{p}$; any one may be chosen. We compute a path between $\boldsymbol{p}^\star$ and $\boldsymbol{q}^\star$ in the roadmap $\mathcal{R}$ based on a graph search. The following theorem states that the star-shaped roadmap enables complete motion planning.

THEOREM 13 *A path exists between two points $\boldsymbol{p}$ and $\boldsymbol{q}$ in $\mathcal{F}$ if and only if $\boldsymbol{p}$ and $\boldsymbol{p}^\star$ are connected in $\mathcal{F}$, $\boldsymbol{p}^\star$ and $\boldsymbol{q}^\star$ are connected in $\mathcal{R}$, and $\boldsymbol{q}^\star$ and $\boldsymbol{q}$ are connected in $\mathcal{F}$, i.e.,*

$$\boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q} \iff \begin{array}{ll} \boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{p}^\star & \wedge \\ \boldsymbol{p}^\star \overset{\mathcal{R}}{\longleftrightarrow} \boldsymbol{q}^\star & \wedge \\ \boldsymbol{q}^\star \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q} & \end{array}$$

Proof: We prove that if $\boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q}$, then the right hand side holds. The proof of the converse is straightforward. Assume $\boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q}$. The star-shaped property implies that

$$\boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{p}^\star \qquad \boldsymbol{q}^\star \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q}$$

We prove that $\boldsymbol{p}^\star \overset{\mathcal{R}}{\longleftrightarrow} \boldsymbol{q}^\star$. The case where $\boldsymbol{p}$ and $\boldsymbol{q}$ belong to the same region is trivial because in that case $\boldsymbol{p}^\star = \boldsymbol{q}^\star$. Suppose $\mathbf{p}$ and $\mathbf{q}$ belong to two separate regions $R_{\boldsymbol{p}}$ and $R_{\boldsymbol{q}}$ respectively. Let $\mathcal{P}$ be any collision-free path between $\boldsymbol{p}$ and $\boldsymbol{q}$. Let $R_i, i = 0, \ldots n$, be the set of regions that are intersected by $\mathcal{P}$ such that $R_0 = R_{\boldsymbol{p}}$ and $R_n = R_{\boldsymbol{q}}$. Consider any two adjacent regions R_k and R_{k+1}. $\mathcal{P}$ passes from R_k to R_{k+1} through the common boundary. This means the boundary contains a connector $\boldsymbol{c}$ that is visible to both $R_k^\star$ as well as $R_{k+1}^\star$. where $R^\star$ denotes the guard of region R. This implies

$$\begin{aligned} & R_k^\star \overset{\mathcal{R}}{\longleftrightarrow} \boldsymbol{c} \quad \wedge \qquad \boldsymbol{c} \overset{\mathcal{R}}{\longleftrightarrow} R_{k+1}^\star \\ \Longrightarrow\ & R_k^\star \overset{\mathcal{R}}{\longleftrightarrow} R_{k+1}^\star \end{aligned}$$

Since this is true for every pair of adjacent regions along $\mathcal{P}$, we have $R_{\mathbf{p}}^\star \overset{\mathcal{R}}{\longleftrightarrow} R_{\mathbf{q}}^\star$. Because $R_{\mathbf{p}}^\star = p^\star$ and $R_{\mathbf{q}}^\star = q^\star$, we have $\boldsymbol{p}^\star \overset{\mathcal{R}}{\longleftrightarrow} \boldsymbol{q}^\star$. This concludes the proof.

□

An important consequence of the above theorem is that it enables us to find a collision-free path for complete motion planning.

Path Planning: if $\boldsymbol{p} \overset{\mathcal{F}}{\longleftrightarrow} \boldsymbol{q}$, then

1. There exists a straight line path α between $\boldsymbol{p}$ and $\boldsymbol{p}^\star$. Similarly, there exists a straight line path β between $\boldsymbol{q}$ and $\boldsymbol{q}^\star$.

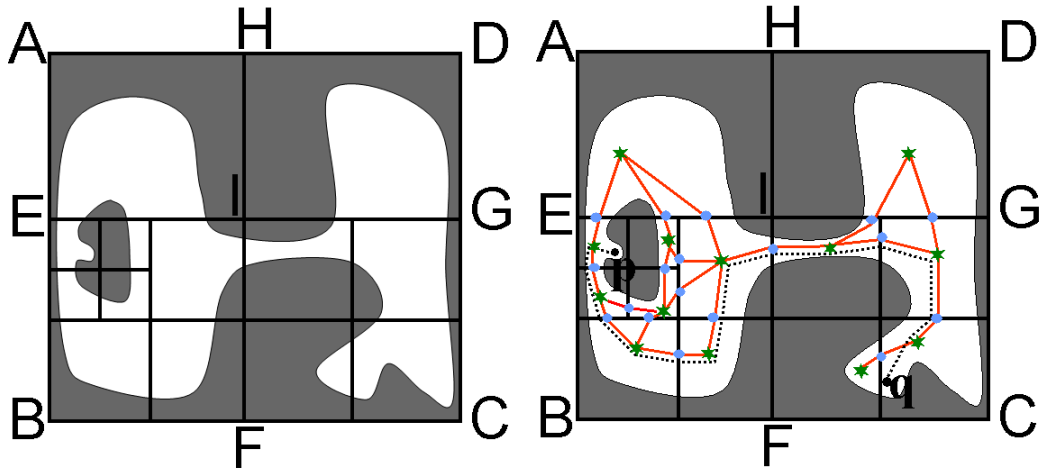
2. There exists a path δ between $\boldsymbol{p}^\star$ and $\boldsymbol{q}^\star$ in the roadmap $\mathcal{R}$.

3. A path between $\boldsymbol{p}$ and $\boldsymbol{q}$ is given by $\alpha :: \delta :: \beta$ where :: denotes path concatenation.

Theorem 13 also provides a test for non-existence of any collision-free path.

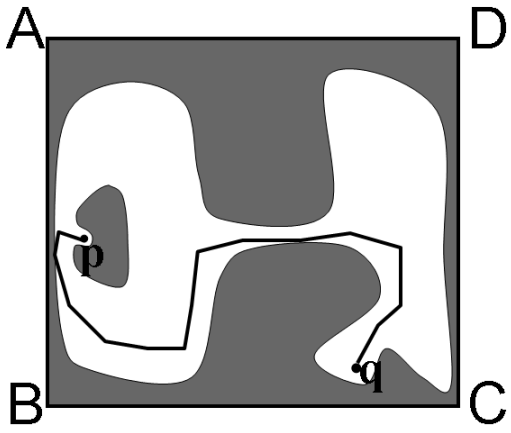
Path Non-Existence: *If there is no path between $\boldsymbol{p}^\star$ and $\boldsymbol{q}^\star$ in the roadmap $\mathcal{R}$, then there is no collision-free path between $\boldsymbol{p}$ and $\boldsymbol{q}$ in $\mathcal{F}$.*

$$\boldsymbol{p}^\star \overset{\mathcal{R}}{\leftarrow\!\!/\!\!\rightarrow} \boldsymbol{q}^\star \implies \boldsymbol{p} \overset{\mathcal{F}}{\leftarrow\!\!/\!\!\rightarrow} \boldsymbol{q}$$



(a) Adaptive Subdivision

(b) Star-Shaped Roadmap



(c) Path Planning

Figure 5.5: Star-Shaped Roadmap Construction: *This figure shows how we compute a star-shaped roadmap using adaptive subdivision of the configuration space. We subdivide the configuration space into regions R such that the free space contained within R, given by $\mathcal{F} \cap R$, is star-shaped. Fig. (b) shows the star-shaped roadmap that was obtained from the resulting subdivision. Fig. (c) shows how the roadmap is used for path computation.*

5.6 Adaptive Subdivision Algorithm for Star-shaped Roadmap Construction

In this section, we present an algorithm to compute a star-shaped roadmap by sampling the free space in a deterministic manner.

5.6.1 Configuration Space Subdivision

The algorithm presented in Sec. 5.5 relied on a star-shaped decomposition of the free space. In practice, we do not have an explicit representation of $\mathcal{F}$; hence it is not possible to compute such a decomposition explicitly. Instead, we compute a subdivision of the configuration space $\mathcal{C}$ into a collection of grid cells such that each cell C satisfies the star-shaped criterion:

$$\mathcal{F}_C = \mathcal{F} \cap C \text{ is star-shaped}$$

Such a subdivision is sufficient for computing a star-shaped roadmap of the free space.

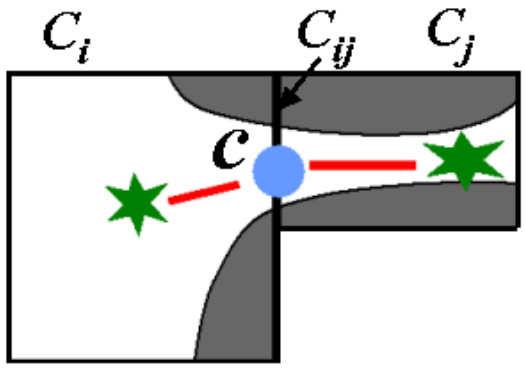


Figure 5.6: Connector: *A connector is a point that connects the free space of two adjacent cells C_i and C_j. The connector* **c** *lies along the shared boundary C_{ij} and is visible to the guards of both the regions.*

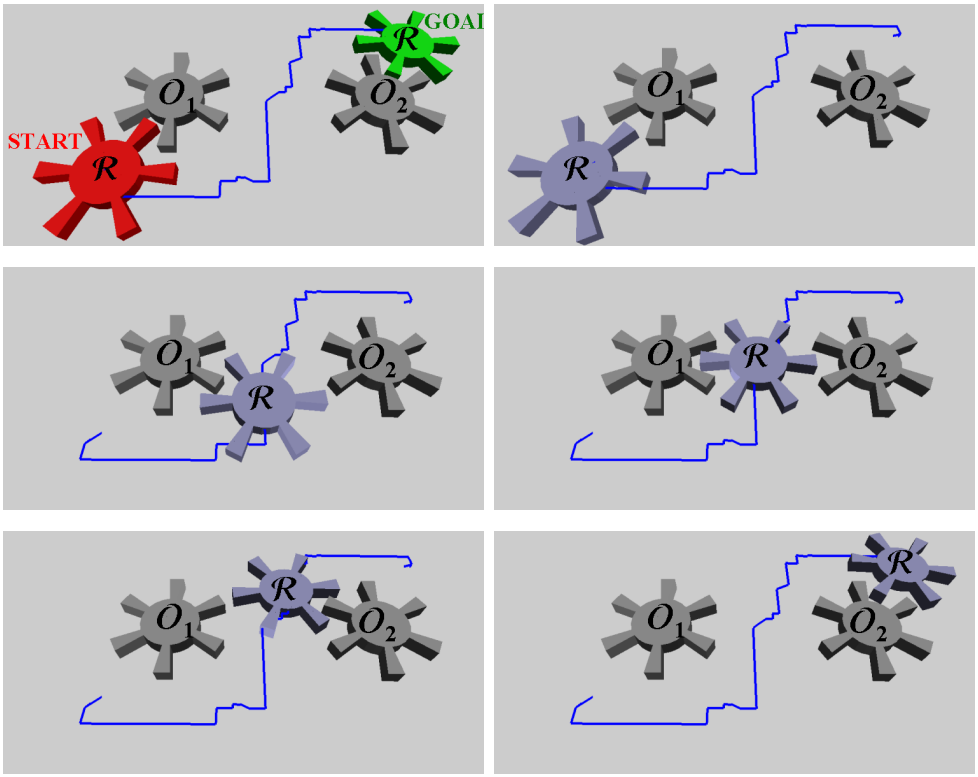
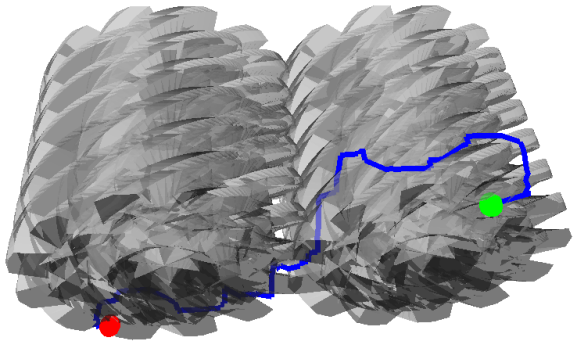
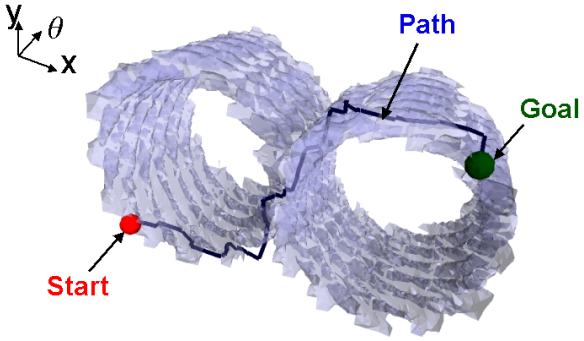


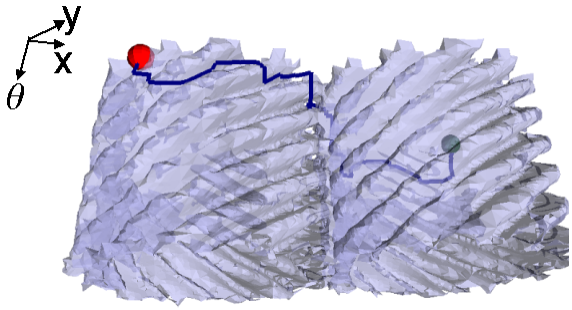
Figure 5.7: **2T+1R**: *This figure highlights application of our algorithm to planar motion planning with both translational as well as rotational dof. Fig. (a) shows a gear-shaped robot $\mathcal{R}$ navigating amongst two gear-shaped obstacles ($\mathcal{O}_1$ & $\mathcal{O}_2$) shown in gray. The start and goal locations of the robot are shown in red and green respectively. The two obstacles form a narrow passage through which the robot must pass in order to reach its goal. Moreover, it must undergo both translation as well as rotation. The figure shows a number of intermediate configurations of the robot during its motion along the path.*



(a) C-Surfaces



(b) Free Space Approximation (View 1)



(c) Free Space Approximation (View 2)

Figure 5.8: **2T+1R** Configuration Space: *This figure highlights application of our configuration space approximation algorithm to the Gears example shown in Fig. 5.7. Fig. (a) show a color-coded image of the C-surfaces. These C-surfaces form a superset of the boundary of the free space. Figs. (b) and (c) shows two views of our free space approximation (drawn translucently). The figures also show the path computed in Fig. 5.7. The path passes through a narrow passage in the configuration space.*

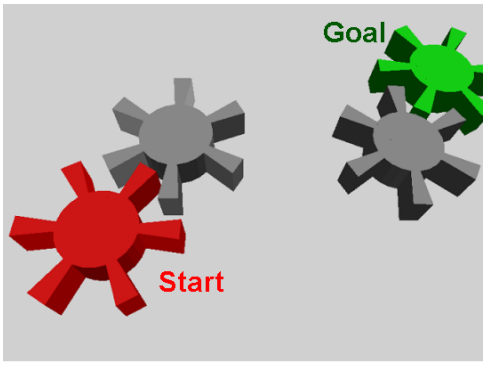


Figure 5.9: **2T+1R** Path non-existence: *If the two obstacles in Fig. 5.7 are moved closer to each other, then no collision-free path exists. Our algorithm is able to detect the non-existence of any collision-free path.*

5.6.2 Adaptive Subdivision and Guard Computation

We generate an adaptive subdivision of $\mathcal{C}$ by a recursive application of the star-shaped criterion. The star-shaped criterion is verified by invoking the star-shaped query (Sec. 5.3.2). Our algorithm starts with a cell C that corresponds to the entire configuration space $\mathcal{C}$. It invokes the star-shaped query on C and depending on the outcome, performs an adaptive subdivision of C. Our algorithm proceeds as follows:

1. Check if C satisfies the contact surface condition (Equation 5.1). If true, then
   (a) Perform the free space existence query (Sec. 5.3.3) on C: Check if $\mathbf{o} \in \mathcal{F}$.
   (b) If $\mathbf{o} \in \mathcal{F}$, set $\mathbf{o}$ as a guard. Otherwise disregard C because it does not contain any point in $\mathcal{F}$ (Theorem 10).
2. Otherwise, subdivide C into a set of children cells C_i, and recursively apply Step 1 to each C_i.

Fig. 5.5(a) illustrates the subdivision algorithm in 2D.

5.6.3 Connector Computation

The objective of connector computation is to determine if the free space of two adjacent cells, C_i and C_j, are connected through a point on their common boundary C_{ij}. In other words, we wish to test if $\mathcal{F} \cap C_{ij} \neq \emptyset$. If there exists a point $\mathbf{c} \in \mathcal{F} \cap C_{ij}$, we call $\mathbf{c}$ a connector. The star-shaped property implies that the connector is visible to the guards of C_i and C_j. See Fig. 5.6.

The problem of connector computation reduces to the free space existence query (Sec. 5.3.3). We can answer this query in two special cases: (a) if any of the vertices of C_{ij} belong to $\mathcal{F}$ or (b) C_{ij} satisfies the contact surface condition. In the second case, we have

$$\mathcal{F} \cap C_{ij} \neq \emptyset \quad \iff \quad \mathbf{o} \in \mathcal{F}$$

We exploit the above fact and compute the connector as follows:

1. If any vertex v of C_{ij} lies in $\mathcal{F}$, use v as a connector and terminate early.

2. If C_{ij} satisfies the contact surface condition and $\mathbf{o} \in \mathcal{F}$, then use $\mathbf{o}$ as the connector and terminate early.

3. If Steps 1 and 2 did not compute a connector, then subdivide C_{ij} into a set of children cells, and recursively apply Steps 1 and 2 to each of the children cells.

Since we need to compute just one point in $\mathcal{F}$ (rather than capture all of them), we terminate as soon as we find one such point. This point is classified as a connector. If C_{ij} contains no point in $\mathcal{F}$, then the subdivision process will continue until all the children cells satisfy the contact surface condition and none of the corresponding points lie in the free space. In this case, the free space of C_i and C_j belong to two separate components of the free space.

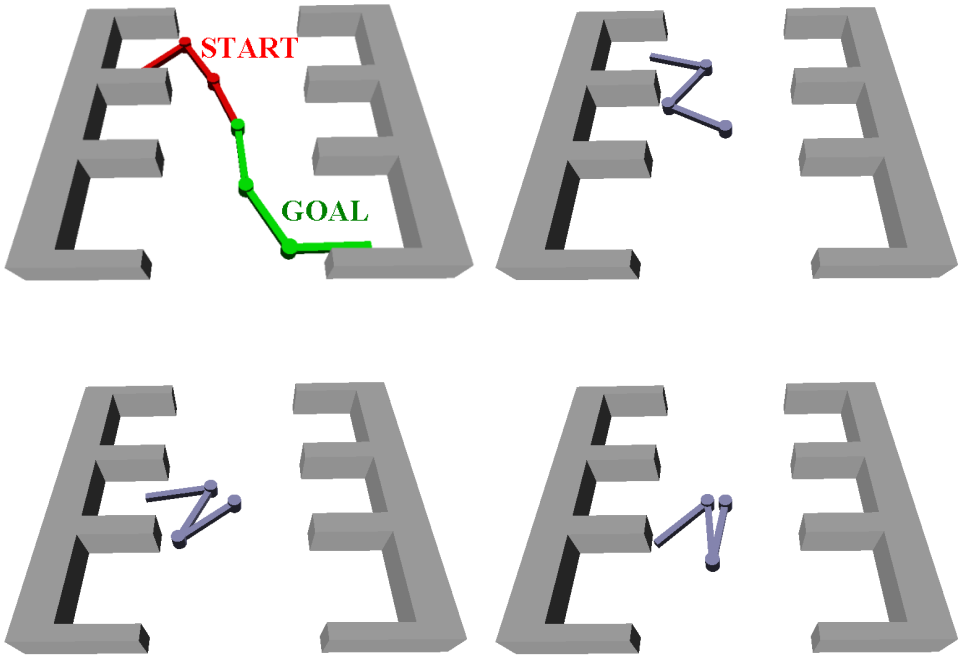
Note that the above adaptive subdivision method for connector computation resembles the subdivision method for guard computation. There are, however, two differences: (a) If d denotes the configuration space dimension, the guard and the connector are computed in dimensions d and $(d-1)$ respectively. (b) The adaptive subdivision method for guard computation continues to perform a subdivision of a cell C until a sufficient set of guards are computed that cover every point in $\mathcal{F}_C$. Whereas the adaptive subdivision method for connector computation continues to perform a subdivision of C_{ij} only until either a single connector is found or non-existence of any connector is determined.

5.6.4 Degeneracies

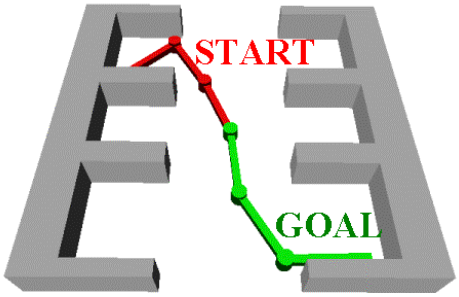
Our algorithm cannot handle tangential contacts on the boundary of free space (Sec. 3.7). A tangential contact occurs when two C-obstacles touch each other at a point thus forming a narrow passage of width zero in the free space. In such cases, the subdivision algorithm will not terminate. This is because the free space in a neighborhood of a

tangential contact is never star-shaped – for any arbitrary neighborhood of nonzero volume.

Planning a path through a tangential contact requires *motion in contact space*: The robot must touch the obstacles while passing through a tangential contact. Our algorithm does not support this type of motion. One possible solution is to isolate a set of regions potentially containing a tangential contact and use a separate contact space planner such as (Redon and Lin, 2005) to do the local planning in those regions.



(a) Path Planning



(b) Path Non-existence

Figure 5.10: **3R**: *This figure highlights application of our star-shaped roadmap algorithm to planar motion planning of an articulated robot with* 3 *revolute joints. The start and goal locations of the robot are shown in red and green respectively. The figure shows a number of intermediate configurations of the robot during its motion along the path. In Fig. (b), the obstacle is moves closer to the robot; as a result, no collision-free path exists. Our algorithm is able to detect path non-existence.*

5.7 Analysis

We had analyzed both complex cell and star-shaped criteria in Sec. 3.6 of Chapter 3. The analysis of the star-shaped criterion in Sec. 3.6 is not applicable to motion planning because the star-shaped criterion in motion planning is more restrictive: We require that the guard of a cell belong to the cell. In this section, we analyze the star-shaped criterion taking this restriction into account. Our analysis is valid only under the assumption that $\partial\mathcal{F}$ is a smooth surface – twice-differentiable manifold. We note that we make this assumption only to simplify the analysis of the algorithm; the algorithm itself does not make this assumption.

Like in Sec. 3.6, we perform the analysis in two stages. In the first stage, we analyze the star-shaped criteria using Gauss map of $\partial\mathcal{F}$ within the cell. We provide a Gauss map condition for when the star-shaped criterion is satisfied. In the second stage, we relate the Gauss map condition to another condition based on local feature size.

5.7.1 Preliminaries

We use the following notation in this section. $d(\mathbf{p}, \mathbf{q})$ denotes the Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$. $\|\vec{\mathbf{u}}\|$ denotes the length of a vector $\vec{\mathbf{u}}$. $\angle\mathbf{m}, \mathbf{n}$ denotes the angle between two vectors $\mathbf{m}$ and $\mathbf{n}$. For a point $\mathbf{p} \in \mathcal{S}$, $\mathbf{n_p}$ will denote the normal to $\mathcal{S}$ at $\mathbf{p}$. A ball in $\mathbb{R}^d$ with center $\mathbf{p} \in \mathbb{R}^d$ and radius ϵ will be denoted as $\mathbb{B}(\mathbf{p}, \epsilon)$ and is defined as $\{\mathbf{x} \mid \|\mathbf{x} - \mathbf{p}\| < \epsilon\}$.

The Gauss map $\mathbb{G}$ of a smooth surface $\mathcal{S}$ in $\mathbb{R}^3$ is a set-valued function from $\mathcal{S}$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p} \in \mathcal{S}$ the outward unit normal to $\mathcal{S}$ at $\mathbf{p}$. We use the term Gauss map to also refer to the image of the Gauss map.

5.7.2 Gauss Map Condition

Given a voxel ϑ, the star-shaped query for motion planning checks whether $\mathcal{F}_\vartheta$ is star-shaped w.r.t a point in ϑ. If ϑ is completely contained in $\mathcal{F}$, then this is trivially true. So it suffices to consider only a boundary voxel, i.e., when $\partial\mathcal{F}_\vartheta \neq \emptyset$. For a boundary voxel, $\mathcal{F}_\vartheta$ is star-shaped if $\partial\mathcal{F}_\vartheta$ is star-shaped w.r.t a point in ϑ. So in the rest of the section, we will only consider the problem of checking whether $\partial\mathcal{F}_\vartheta$ is star-shaped.

According to Corollary 4 in Sec. 3.6, if a voxel ϑ is normal-bounded by θ, $0 \leq \theta < \pi/2$ – i.e., if the angle between the normals at any two points in $\partial\mathcal{F}_\vartheta$ is less than θ – then a point $\mathbf{p} - \alpha\|\vartheta\|\mathbf{n_p}$ can be chosen as a guard for any choice of $\mathbf{p} \in \partial\mathcal{F}_\vartheta$ and

$\alpha > 1/\cos(\theta)$. However, because α is always greater than 1, this guard always lies outside the voxel. While this was sufficient for the algorithm in Chapter 3, it is not sufficient in the current context. It turns out that the requirement that $\alpha > 1/\cos(\theta)$ in Corollary 4 is overly restrictive. In the following theorem (Theorem 14), we show that it suffices if $\alpha > 2\sin(\theta/2)/\cos(\theta)$. For small values of θ, α is roughly equal to θ. Thus α can assume small values, thereby allowing the guard to lie inside the voxel.

Recall the following definitions from Sec. 3.6.

DEFINITION 11 *Consider a line segment* $\mathbf{pq}$ *that intersects* $\partial\mathcal{F}$.

- *We call an intersection point* $\mathbf{r} \in \partial\mathcal{F}_{\mathbf{pq}}$ *a* **tangential point** *if* $\mathbf{pq} \cdot \mathbf{n_r} = 0$. *Otherwise we say* $\mathbf{r}$ *is a* **transversal point**. *We call* $\mathbf{r}$ *an* **entry point** *if* $\mathbf{pq} \cdot \mathbf{n_r} < 0$, *an* **exit point** *if* $\mathbf{pq} \cdot \mathbf{n_r} > 0$.

- *If all the intersection points are transversal then we say* $\mathbf{pq}$ *intersects* $\partial\mathcal{F}$ **transversally**.

Suppose $\mathbf{pq}$ intersects $\partial\mathcal{F}$ transversally and that the intersection points are sorted in the order of increasing distance from $\mathbf{p}$. Then the intersection points will alternate between entry and exit points. This is because we assume $\partial\mathcal{F}$ is an oriented closed manifold.

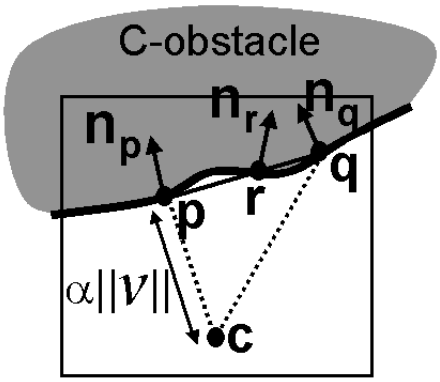


Figure 5.11: *This figure supports the proof of Theorem 14. If for any two points* $\mathbf{p}$ *and* $\mathbf{q}$ *in* $\partial\mathcal{F}_\vartheta$, *the angle between the corresponding normals* $\mathbf{n_p}$ *and* $\mathbf{n_q}$ *is less than* $\pi/2$, *then* $\partial\mathcal{F}_\vartheta$ *is star-shaped. Point* $\mathbf{c}$ *is the guard for the voxel.*

THEOREM 14 *Let* ϑ *be a boundary voxel. If* ϑ *is* normal-bounded *by* θ, $0 \leq \theta < \pi/2$, *then*

1. $\partial\mathcal{F}$ *is star-shaped w.r.t* ϑ *and*

2. *$F(\mathbf{p}, \alpha) = \mathbf{p} - \alpha\|\vartheta\|\mathbf{n_p}$ is a guard of $\partial\mathcal{F}_\vartheta$ for any choice of $\mathbf{p} \in \partial\mathcal{F}_\vartheta$ and $\alpha > 2\sin(\theta/2)/\cos(\theta)$.*

Proof: Let $\mathbf{c} = F(\mathbf{p}, \alpha)$. Consider any point $\mathbf{q} \in \partial\mathcal{F}_\vartheta$. We show that $\mathbf{cq} \cdot \mathbf{n_q} > 0$. Using this fact, we can prove that $\mathbf{cq} \cap \partial\mathcal{F} = \{\mathbf{q}\}$ in a manner similar to Theorem 3.

See Fig. 5.11. We have

$$\begin{aligned} \mathbf{cq} &= \mathbf{cp} + \mathbf{pq} \\ &= \alpha\|\vartheta\|\mathbf{n_p} + \mathbf{pq} \\ \mathbf{cq} \cdot \mathbf{n_q} &= \alpha\|\vartheta\|\mathbf{n_p} \cdot \mathbf{n_q} + \mathbf{pq} \cdot \mathbf{n_q} \end{aligned} \tag{5.2}$$

Because ϑ is normal-bounded by θ, $\angle\mathbf{n_p}, \mathbf{n_q} < \theta$. Therefore,

$$\mathbf{cq} \cdot \mathbf{n_q} > \alpha\|\vartheta\|\cos(\theta) + \mathbf{pq} \cdot \mathbf{n_q} \tag{5.3}$$

We now bound $\mathbf{pq} \cdot \mathbf{n_q}$. We can assume that $\partial\mathcal{F}$ intersects $\mathbf{pq}$ transversally at $\mathbf{q}$ because otherwise $\mathbf{pq} \cdot \mathbf{n_q} = 0 \implies \mathbf{cq} \cdot \mathbf{n_q} > 0$.

The line segment $\mathbf{pq}$ may intersect $\partial\mathcal{F}$ at points other than $\mathbf{p}$ and $\mathbf{q}$. Let $\mathbf{r}$ be a point belonging to $\partial\mathcal{F} \cap (\mathbf{pq} \setminus \{\mathbf{q}\})$ that is closest to $\mathbf{q}$. (Fig. 5.11).

$$\mathbf{pq} \cdot \mathbf{n_q} = \mathbf{rq} \cdot \mathbf{n_q} \tag{5.4}$$

For the sake of simplicity, let us first assume that $\mathbf{r}$ is a transversal intersection point. Then since $\mathbf{r}$ and $\mathbf{q}$ are "consecutive" intersection points, one of them is an entry point and the other is an exit point. This means $\mathbf{rq} \cdot \mathbf{n_r}$ and $\mathbf{rq} \cdot \mathbf{n_q}$ have opposite signs. This implies

$$|\mathbf{rq} \cdot \mathbf{n_q}| < |\mathbf{rq} \cdot (\mathbf{n_r} - \mathbf{n_q})| \tag{5.5}$$

$$\begin{aligned} &< \|\mathbf{rq}\|\|(\mathbf{n_r} - \mathbf{n_q})\| \\ &< \|\mathbf{rq}\|2\sin(\theta/2) \quad \text{because } \angle\mathbf{n_r}, \mathbf{n_q} < \theta \\ &< \|\vartheta\|2\sin(\theta/2) \quad \text{because } \|\mathbf{rq}\| < \vartheta \\ &< \alpha\|\vartheta\|\cos(\theta) \end{aligned} \tag{5.6}$$

Equations 5.3, 5.4 and 5.6 together imply $\mathbf{cq} \cdot \mathbf{n_q} > 0$.

We now consider the case where $\mathbf{r}$ is a tangential intersection point, i.e., $\mathbf{pq} \cdot \mathbf{n_r} = 0$. Even in this case Equation 5.5 holds. This is because

$$\begin{aligned} \mathbf{rq} \cdot \mathbf{n_q} &= \mathbf{rq} \cdot \mathbf{n_q} - \mathbf{pq} \cdot \mathbf{n_r} \\ &= \mathbf{rq} \cdot \mathbf{n_q} - \mathbf{rq} \cdot \mathbf{n_r} \\ &= \mathbf{rq} \cdot (\mathbf{n_q} - \mathbf{n_r}) \end{aligned}$$

□

If a voxel satisfies the condition in the above theorem, then any point from the set

$$S = \{F(\mathbf{p}, \alpha) \mid \mathbf{p} \in \partial\mathcal{F}_\vartheta,\ \alpha > 2\sin(\theta/2)/\cos(\theta)\}$$

can be chosen as a guard. S is a subset of the kernel of $\partial\mathcal{F}_\vartheta$. The distance between $\partial\mathcal{F}_\vartheta$ and the closest guard depends on θ: As θ approaches $\pi/2$, the distance approaches infinity; however, for small values of θ, the distance approaches zero, allowing the guard to lie inside the voxel. We now present a set of conditions under which a guard will belong to the voxel. First, we introduce a few definitions.

DEFINITION 12 *Consider a boundary voxel ϑ. A point $\mathbf{p} \in \partial\mathcal{F}_\vartheta$ is* ϵ-interior *if there exists a ball $\mathbb{B}(\mathbf{p}, \epsilon) \subset \vartheta$. A set S is* ϵ-interior *if every point in S is* ϵ-interior.

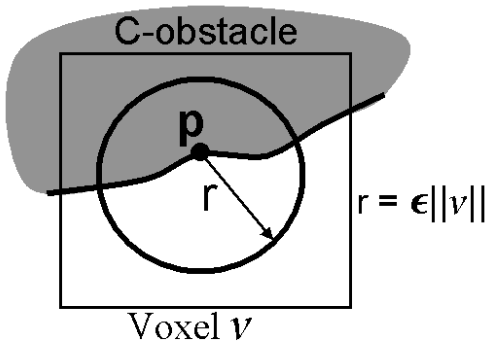


Figure 5.12: Non-degeneracy condition C2: *The figure shows a point $\mathbf{p} \in \partial\mathcal{F}_\vartheta$ in a voxel ϑ such that a ball of radius $\epsilon\|\vartheta\|$ centered at $\mathbf{p}$ lies inside ϑ. We call such a point an* $\epsilon\|\vartheta\|$-interior*point. We require that every boundary voxel have such a point.*

DEFINITION 13

$$\begin{aligned} \theta_\vartheta &= \sup\{\angle \mathbf{n}_1, \mathbf{n}_2 \mid \mathbf{n}_1, \mathbf{n}_2 \in \mathbb{G}_\vartheta(\partial\mathcal{F}_\vartheta)\} \\ \alpha_\vartheta &= 2\sin(\theta_\vartheta/2)/\cos(\theta_\vartheta) \end{aligned}$$

Intuitively, θ_ϑ and α_ϑ are a measure of the size of the Gauss map of $\partial\mathcal{F}_\vartheta$. We show that if α_ϑ is sufficiently small, then there exists a guard of $\partial\mathcal{F}_\vartheta$ that lies inside ϑ. To prove this, we need two conditions to hold.

DEFINITION 14

1. **Monotonicity condition C1:** *θ_ϑ decreases with $\|\vartheta\|$ (unless θ_ϑ is zero). Intuitively, this condition means that as a voxel ϑ becomes smaller, the Gauss map of $\partial\mathcal{F}_\vartheta$ occupies a smaller region in $\mathbb{S}^2$. This is true for smooth surfaces.*

2. **Non-degeneracy condition C2:** *There exists an $\epsilon > 0$ such that every boundary voxel ϑ contains an* $\epsilon\|\vartheta\|$-interior *point. See Fig. 5.12. This condition prevents $\partial\mathcal{F}$ from grazing along the boundary of the voxel.*

From now on, we will assume that the above two conditions hold.

As $\|\vartheta\|$ decreases, θ_ϑ decreases, which in turn causes α_ϑ to decrease. At a certain size of the voxel, α_ϑ becomes smaller than $\epsilon/2$. For such a voxel ϑ, we show that there exists a guard belonging to ϑ.

COROLLARY 15 *Suppose there exists an $\epsilon > 0$ satisfying Condition* **C2**. *If every boundary voxel ϑ satisfies $\alpha_\vartheta < \epsilon/2$, then we have*

1. *$\partial\mathcal{F}$ is star-shaped w.r.t ϑ,*

2. *$F(\mathbf{p}_\vartheta, \alpha_\vartheta + \epsilon/2)$ is a guard of $\partial\mathcal{F}_\vartheta$, and*

3. *$F(\mathbf{p}_\vartheta, \alpha_\vartheta + \epsilon/2) \in \mathcal{F}_\vartheta$.*

where $\mathbf{p}_\vartheta$ is an $\epsilon\|\vartheta\|$-interior *point in ϑ.*

Proof: Define $\mathbf{q}_\vartheta = F(\mathbf{p}_\vartheta, \alpha_\vartheta + \epsilon/2)$. Theorem 14 implies that $\mathbf{q}_\vartheta$ is a guard of $\partial\mathcal{F}_\vartheta$. We prove that $\mathbf{q}_\vartheta$ belongs to both ϑ and $\mathcal{F}$. We have

$$\begin{aligned} \mathbf{q}_\vartheta &= F(\mathbf{p}_\vartheta, \alpha_\vartheta + \epsilon/2) \\ &= \mathbf{p}_\vartheta - (\alpha_\vartheta + \epsilon/2)\|\vartheta\|\mathbf{n}_\mathbf{p} \\ &\in \mathbb{B}(\mathbf{p}_\vartheta, \epsilon\|\vartheta\|) \quad \text{because } \alpha_\vartheta < \epsilon/2 \\ &\subset \vartheta \quad \text{because } \mathbf{p}_\vartheta \text{ is } \epsilon\|\vartheta\|\text{-interior} \end{aligned} \tag{5.7}$$

Because $\mathbf{q}_\vartheta$ is a guard of $\partial\mathcal{F}_\vartheta$, the line segment between $\mathbf{p}_\vartheta$ and $\mathbf{q}_\vartheta$ does not intersect $\partial\mathcal{F}$ at any point other than $\mathbf{p}_\vartheta$. Since $\mathbf{p}_\vartheta \in \partial\mathcal{F}$ and $\mathbf{q}$ is obtained by moving along the outward normal to $\partial\mathcal{F}$, we have $\mathbf{q}_\vartheta \in \mathcal{F}$. Therefore, $\mathbf{q}_\vartheta \in \mathcal{F}_\vartheta$.

□

The above corollary ensures that as voxels become smaller, the star-shaped criterion is eventually satisfied and the resulting guard belongs to the voxel. This is true only if the monotonicity condition (**C1**) and the non-degeneracy condition (**C2**) hold. A similar result holds for the star-shaped criterion on the faces of the cell.

5.7.3 Local Feature Size Condition

In this subsection, we derive a conservative lower bound on the size of the grid cells during adaptive subdivision. This provides a sufficient condition for the termination of the algorithm.

We reduce the Gauss map condition (Corollary 15) to a condition based on local feature size (LFS). We show that the Gauss map condition is met if the grid cells are smaller than a certain fraction of their LFS. This yields a lower bound on the cell size and in turn a sufficient condition for termination of the algorithm. The following analysis closely follows to the one presented in Sec. 3.6.4.

Our goal is to show that a cell will satisfy the star-shaped criterion provided it is "sufficiently small". We make the previous statement precise using the following definition.

DEFINITION 15 *Let c be a face/voxel of a cell C. Given any $\epsilon > 0$, we have*

1. *c is* ϵ-small *if $\|c\| < \rho\mathrm{LFS}(c)$ for any $\rho < \epsilon/(4+3\epsilon)$.*
2. *C is* ϵ-small *if every face/voxel of C is* ϵ-small.

The above definition of ϵ-small resembles the definition of a LFS-small cell presented in Sec. 3.6.4. However, there are two differences. First, the value of ρ is different. Second, unlike the definition of LFS-small, ϵ-small does not place a requirement on the edges of the cell. This is because the star-shaped criterion needs to be applied only to the voxel and faces of the cell.

For an ϵ-small voxel ϑ, we have $\theta_\vartheta < \epsilon/4$. This follows from Lemma 6 by substituting $\rho = \epsilon/(4+3\epsilon)$. Furthermore, we can show that $\alpha_\vartheta = 2\sin(\theta_\vartheta/2)/\cos(\theta_\vartheta)$ is always less than $\epsilon/2$. Then Corollary 15 ensures that a ϵ-small voxel will satisfy the star-shaped criterion. This yields the following result.

THEOREM 16 *Suppose there exists an $\epsilon > 0$ satisfying Condition* **C2**. *If a voxel ϑ is* ϵ-small, *then $\partial\mathcal{F}_\vartheta$ is star-shaped w.r.t $F(\mathbf{p}_\vartheta, \alpha_\vartheta + \epsilon/2) \in \mathcal{F}_\vartheta$.*

5.7.4 Termination

The preceding analysis provides a sufficient condition for the termination of our algorithm. Our algorithm performs two types of adaptive subdivision – one for guard computation and another for connector computation. Theorem 16 is applicable to the adaptive subdivision algorithm for guard computation. It provides a lower bound on the size of the voxels relative to the LFS. During adaptive subdivision, once the size of a voxel ϑ is less than $\rho\text{LFS}(\vartheta)$, it is ϵ-small and satisfies the star-shaped criterion. The adaptive subdivision will terminate provided there exist a lower bound on the LFS of every grid cell. Suppose there exists such a lower bound $\tau > 0$. Assuming a cell halves its size at each subdivision step, this implies a lower bound of $\rho\tau/2$ on the size of every cell. We use this fact to provide the following sufficient condition for the termination of the subdivision algorithm:

COROLLARY 17 *If the following two conditions hold,*

1. *There exists an $\epsilon > 0$ satisfying Condition* **C2**.
2. *There exists a $\tau > 0$ such that during adaptive subdivision, the LFS of every voxel is greater than τ.*

then the adaptive subdivision (for guard computation) will terminate and the voxels will be of size greater than $\rho\tau/2$.

A similar argument holds for the adaptive subdivision algorithm for connector computation.

5.8 Implementation and Results

In this section, we describe the implementation of our algorithm demonstrate its performance on several models. Tables 5.1 and 5.2 highlight the performance of the free space approximation and motion planning algorithms on these models. All timings were obtained on a 2 GHz Pentium IV PC with a GeForce 4 graphics card and 1 GB RAM. We used C++ programming language with a GNU g++ compiler under Linux operating system.

Model	Complexity		Performance				
	Num of Edges/Tris		# Surf	Grid Gen	Isosurface	Grid Size	Appr Size
	Obstacle	Robot	size	(s)	(s)	size	size
Gears	36	72	3,929	212	3.2	78,384	66,389
Assembly	224	224	256	6	1.8	25,124	22,928

Table 5.1: Performance of our free space approximation algorithm: *The model complexity is provided in terms of the number of edges/triangles of the polygonal/polyhedral objects. The table shows the number of contact surfaces, the time for grid generation, the time for isosurface extraction, the size of the grid, and the size of our free space approximation.*

Model	Complexity			Performance			Statistics	
	Robot	Obstacle	# Surf	Guard (s)	Connector (s)	Planning (s)	# Guards	# Connectors
Gears A	36	72	3,929	62	49	0.22	6,764	11362
Gears B (No path)	36	72	3,929	58	32	0.18	3,412	5,348
Assembly	224	224	256	10.1	5.8	0.22	6137	15,399
3R A	3	32	140	12.3	4.9	0.43	11,349	30,566
3R B	3	32	176	12.2	4.4	0.14	10,062	25,270

Table 5.2: Performance of star-shaped roadmap method: *This table highlights the performance of our motion planning algorithm on different models. The model complexity is provided in terms of the size of the robot and the obstacle as well as the number of contact surfaces. The size of an object refers to the number of vertices for the planar examples (*Gears *and* 3R*), and the number of triangles for the 3D* Assembly *example. The performance is measured in terms of the roadmap construction time and the time to answer a single planning query. The roadmap construction time is the sum of the time taken to compute an adaptive subdivision (includes guard computation) and the time to compute the connectors. The table also provides statistics on the number of guards and connectors in the roadmap.*

Fig. 5.7 shows a **2T+1R** example. The figure shows a gear-shaped robot navigating among two gear-shaped obstacles. The robot and the obstacles are obtained by extruding planar polygons. By considering pairs of features of these planar polygons, we enumerated a set of $3,929$ contact surfaces. Our algorithm computed a free space approximation in 215 secs. The figure also shows application of our star-shaped roadmap algorithm for motion planning. Our algorithm took 111 secs to construct a star-shaped roadmap. Using this roadmap, it took only 0.22 secs to compute a path. The robot must pass through a very narrow passage to reach its goal. Furthermore, it must undergo both translation as well as rotation in order to pass through the narrow passage.

Fig. 5.7 (d) highlights the use of our algorithm to detect non-existence of any collision-free path. The two obstacles are too close to each other, and consequently, the robot cannot pass through the passage between them. Our algorithm took 90 secs to compute a roadmap for this environment and detected non-existence of any collision-free path in 0.18 secs.

Fig. 1.14 shows an application of our algorithm to assembly planning. It consists of two parts each with pegs and holes. The goal is to assemble the two parts so that the pegs of one part fit into the holes of the other. We treat one of the parts as the robot $\mathcal{R}$ and the other as the obstacle $\mathcal{O}$. The parts are allowed to translate in 3D. The free space is expressed in terms of the Minkowski sum: $\mathcal{F} = \overline{\mathcal{O} \oplus -\mathcal{R}}$. We use our Minkowski sum approximation method (described in Chapter 4) to compute the free space approximation. The free space is defined as a complement of the union of 256 pairwise Minkowski sums. Our algorithm computed an approximation in 7.8 secs. Our star-shaped roadmap algorithm took 16 secs to construct a roadmap and was able to find a path (shown in blue) in 0.22 secs. This is a challenging example because the goal configuration, wherin the pegs fit into the holes, is lodged within a very narrow passage in the configuration space.

Fig. 5.10 shows application of our algorithm to a **3R** planar articulated robot with 3 revolute joints. Our algorithm took 17 secs to construct a star-shaped roadmap. Using this roadmap, it took only 0.43 secs to compute a path. The robot must pass through a narrow passage to reach its goal.

In Fig. 5.10(b), the environment is too close to the robot and consequently, there exists no collision-free path between the initial and goal configurations. Our algorithm was able to detect non-existence of any collision-free path. It took 16 secs to compute a roadmap for this environment and detected path non-existence in 0.14 secs.

Approx. Cell Decomposition	**Star-Shaped Roadmaps**
Decomposition of $\mathcal{C}$ into *empty*, *full* and *mixed* cells	Decomposition of $\mathcal{C}$ into cells satisfying star-shaped property
Conservative approximation of $\mathcal{F}$	Complete connectivity of $\mathcal{F}$; the guards cover every point in $\mathcal{F}$
Need to subdivide mixed cells; no stopping condition	Not necessary to subdivide mixed cells that satisfy the star-shaped property
Large storage and search requirements; function of resolution parameter	Storage and search varies based on free space complexity; Lower requirements
Check for paths through empty cells and not mixed cells.	Check for paths through empty cells as well as mixed cells that satisfy the star-shaped property

Table 5.3: Comparison: *This table compares a number of aspects of our approach with approximate cell decomposition.*

5.9 Comparison with Prior Motion Planning Methods

In this section, we compare various aspects of our star-shaped roadmap method with cell decomposition methods and randomized sampling based methods.

5.9.1 Cell Decomposition Methods

Our algorithm performs adaptive subdivision similar to cell decomposition algorithms (Latombe, 1991). However, there is one major difference; unlike exact cell decomposition methods, we do not compute an explicit decomposition of the free space. Instead, we compute a subdivision of the entire configuration space, which represents the free space implicitly. The main advantage of our method is that we are able to perform the subdivision without an explicit representation of the free space.

Most practical cell decomposition algorithms are based on approximate cell decomposition approach. We perform a detailed comparison with approximate cell decomposition approach in Table 5.3. While approximate cell-decomposition algorithms are *resolution complete*, our algorithm is able to perform complete motion planning. The approximate cell decomposition methods partition the free space into *empty* and *mixed* cells. These methods find a path only through the empty cells. These are cells that lie completely in free space and form a conservative approximation of the free. They do not find a path through mixed cells, which intersect the boundary of the free space. To overcome this problem, they subdivide mixed cells. However, a drawback of these methods is that there is no stopping condition for the subdivision of mixed cells.

One important benefit of our method is that we do not always have to subdivide the mixed cells. If a mixed cell satisfies the star-shaped test, then we do not subdivide it. We can plan paths through mixed cells directly by exploiting the star-shaped property. This reduces the total number of subdivisions considerably.

5.9.2 Randomized Sampling Methods

We generate samples in the free space that are represented by guards and connectors. Table 5.4 compares our deterministic sampling approach with randomized sampling approach by showing the different steps of the two approaches. Our approach does not need to perform explicit local planning to connect nearby samples. The star-shaped property ensures that the connectors link guards belonging to adjacent regions thus

providing local planning implicitly. The main benefit of randomized sampling methods is that they easily extend to high dof robots, whereas our method has additional overhead for contact surface enumeration and conservative star-shaped tests.

Randomized Sampling	**Star-shaped Roadmaps**
Compute samples randomly	Compute guards & connectors deterministically
Check whether samples are in free space	The guards and connectors are in free space by construction
Perform local planning between nearby samples	No *explicit local planning*; star-shaped property guarantees local collision-free paths
Easily extends to high-dof robots	Storage complexity and cost of star-shaped tests increases with number of dof
May not terminate with narrow passages or no collision-free path	Guaranteed to terminate if there are no tangential contacts in free space

Table 5.4: Comparison: *This table compares the steps of our approach with those of the randomized sampling approach.*

Our approach shares some similarities with the visibility based probabilistic roadmap method (Visibility-PRM) (Simeon et al., 2000). Visibility-PRM method takes inter-sample visibility into account during the randomized sampling process. While the star-shaped property is related to visibility, it is different from the type of visibility computed by (Simeon et al., 2000). While the star-shaped property implicitly determines the visibility of an entire region, the visibility-PRM method computes the visibility of a new randomly computed sample with respect to the current set of samples. Finally, the goals of the two methods are different: the objective of the visibility-PRM method is to generate a probabilistic roadmap with fewer nodes, whereas our goal is to do complete path planning.

5.10 Limitations

Our algorithm assumes that the free space does not have any tangential contacts. Hence it cannot handle cases where the robot must touch an obstacle in order to pass through a narrow passage to get to the goal configuration.

A bottleneck in our approach is the large number of C-surfaces. Typically, our method enumerates $O(n^2)$ C-surfaces where n is the number of features in the robot and the obstacles. Many of these C-surfaces lie within C-obstacle and do not contribute to the actual boundary of the free space, thus adding an unnecessary overhead to the algorithm. We can alleviate this problem by using better *culling* techniques to eliminate such C-surfaces (Sacks, 1999).

Our motion planning algorithm is not specific to fixed number of dof. In theory, it

is applicable to robots with arbitrary dof. However, the theoretical complexity and the implementation complexity grow considerably with dof. The main bottleneck in our method is contact surface enumeration, which becomes difficult for high dof.

5.11 Conclusions

We have presented a practical algorithm for approximating the free configuration space of robots with translational and rotational degrees of freedom. Unlike previous methods, our method avoids computing an arrangement of the C-surfaces. Instead, we use a sampling-based method to compute a geometrically close and topologically correct approximation.

We have presented a sampling based algorithm for complete motion planning. It relies on computing a star-shaped roadmap of the free space. We construct this roadmap using deterministic sampling. Provided the sampling condition is met, the resulting roadmap captures the connectivity of the free space enabling us to perform complete path planning. Our algorithm is simple to implement and primarily relies on a star-shaped test which can easily be implemented. We have demonstrated the performance of our planner in complex scenarios with low dof robots.

Chapter 6

Conclusion and Future Work

In this thesis, we have addressed two classes of geometric problems. The first class includes surface extraction problems such as Boolean operations, Minkowski sum evaluation, and configuration space computation. The second class of problems is motion planning of rigid or articulated robots translating or rotating among stationary obstacles. We propose solutions to both classes of problems based on sampling. Our approach has the following advantages:

1. **Accuracy:** It provides geometric and topological guarantees on the output. Our surface extraction algorithm computes a polygonal approximation of the exact surface. The approximation is guaranteed to be topologically equivalent to the exact surface and has a bounded two-sided Hausdorff error. Our motion planning algorithm performs complete planning. It is capable of not only finding a path when one exists but also detecting non-existence of any collision-free path.

2. **Simplicity:** Our approach is relatively simple to implement. It is based on the following components:

   (a) Sign computation,

   (b) Distance computation (max-norm or directed distance),

   (c) Linear programming,

   (d) Interval arithmetic,

   (e) Isosurface extraction.

   All the above techniques are easy to implement. Moreover, there exist public domain implementations for linear programming (GLPK, 2003; QSOPT, 2005), in-

terval arithmetic (Mehlhorn and Näher, 1995), and isosurface extraction (Schroeder et al., 1997; Schaefer, 2002).

3. **Practicality:** We have demonstrated the performance of our surface extraction algorithm for the following applications: Boolean operations on complex polyhedral models and low degree algebraic primitives, model simplification and remeshing of polygonal models, Minkowski sums and offsets of complex polyhedral models, and configuration space computation for low degrees of freedom objects. We have demonstrated the performance of our motion planning algorithm on a number of challenging environments with narrow passages and no collision-free paths.

However, our overall approach suffers from the following limitations:

1. **Degeneracies:** As noted in Sec. 3.7, the adaptive subdivision step of the approach is susceptible to degeneracies. While subdividing a cell, the resulting children cells might graze the exact surface. In such cases, the algorithm does not terminate. Perturbation methods may offer a potential solution to these problems, but this requires further investigation.

2. **Robustness Problems:** Our current implementation uses floating point arithmetic which makes it prone to accuracy problems. For example, the sign query and directed distance query rely on ray shooting. A ray is shot from a query point in a certain direction and intersection points of this ray with the exact surface are computed. This computation is not robust when the query point is too close to the exact surface or when the ray passes through a vertex or edge of a polyhedral surface. Exact arithmetic may be needed to overcome the robustness problem.

3. **Conservative Tests:** As described in Sec. 3.5, our algorithm uses conservative tests to verify the complex cell and star-shaped criteria. Due to the conservativeness of these tests, a cell may be subdivided even though it satisfies both the criteria. This can result in some unnecessary subdivision and reduce the overall performance. The main advantage of these tests is that they do not require an explicit representation of the exact surface.

In the following sections, we summarize our algorithms and discuss directions for future investigation.

6.1 Surface Extraction

We have presented an algorithm for topology preserving isosurface extraction. It relies on a sufficient sampling condition based on complex cell and star-shaped criteria that ensures that the reconstructed isosurface maintains the topology of the original isosurface. We have described a simple extension to the sampling condition to bound the two-sided Hausdorff error of the reconstructed isosurface. We enforce the sampling condition by performing adaptive subdivision. This is efficient in practice and easy to implement. We have demonstrated the application of our algorithm to Boolean operations, topology preserving simplification, and remeshing on a number of complex examples.

There are many avenues for future work. Our sampling criteria – complex cell and star-shaped criteria – are geared towards Marching Cubes reconstruction. We would like to develop better reconstruction algorithms so that we could make the sampling criteria less conservative and yet preserve topology.

The star-shaped criterion ensures that the surface within every cell is star-shaped. A useful property of a star-shaped surface is that it has a spherical parametrization. The fact that every point on the surface is visible to the guard can be used to map the surface onto a portion of the unit sphere. Then a tessellation of this portion of the sphere yields a polygonization of the star-shaped surface. It would be useful to develop a reconstruction algorithm that exploits this property.

Current algorithms for kernel computation on curved primitives involve solving non-linear equations and can be slow. For the special case of rational freeform surfaces, kernel computation can be reformulated as computing the zero sets of polynomial equations (Seong et al., 2004). Solving such equations for each grid cell can be rather expensive in practice. Moreover, no good algorithms are known for kernel computation on free-form solids defined using subdivision surfaces. We would like to develop efficient algorithms for kernel computation on curved solids.

In applications such as laser scanning, the input data often contains topological noise due to inaccuracies in the scanning and merging process. We would like to investigate whether our results can be combined with the algorithms presented in (Guskov and Wood, 2001; Bischoff and Kobbelt, 2002) and used to perform topological reasoning for noise removal.

Our current implementation supports polyhedral and low order algebraic primitives. We would like to apply our algorithm to higher order NURBS and subdivision surfaces.

Finally, we plan to use our algorithm for other surface extraction problems such as swept volume computation.

6.2 Minkowski Sum Computation

We have presented an algorithm to approximate the 3D Minkowski sum of polyhedral objects. Our algorithm provides geometric and topological guarantees on the approximation. We employ cell and primitive culling techniques to improve the performance of our algorithm. We have applied our algorithm to offset computation and motion planning of robots with translational degrees of freedom.

Our algorithm relies on convex decomposition. This is the main bottleneck in the algorithm. We used a convex decomposition scheme available in a public collision detection library, SWIFT++ (Ehmann and Lin, 2001). The convex decomposition method often produces a large number of convex pieces: For a non-convex polyhedron with n triangles, it typically produces $O(n)$ convex pieces. This lowers the performance of our algorithm. A potential way of overcoming this problem is to design an algorithm that does not rely on convex decomposition. It can be shown that the Minkowski sum of two star-shaped polyhedra is a star-shaped polyhedron. We could exploit this property and design our overall approach based on star-shaped decomposition instead of convex decomposition. The main advantage of this approach is that the star-shaped decomposition of a polyhedron would typically result in fewer primitives. However, to pursue this approach, we need to develop efficient algorithms for computing Minkowski sums of star-shaped primitives. We plan to investigate this further.

6.3 Configuration Space Computation and Motion Planning

We have presented a practical algorithm for approximating the free configuration space of robots with translational and rotational degrees of freedom. Unlike many previous methods, our method avoids computing an arrangement of the C-surfaces. Instead, we use the surface extraction algorithm to compute a geometrically close and topologically correct approximation.

We have presented a sampling based algorithm for complete motion planning. It relies on computing a star-shaped roadmap of the free space. We construct this roadmap

using deterministic sampling. The resulting roadmap captures the connectivity of the free space thus guaranteeing complete path planning. Our algorithm is simple to implement and primarily relies on a star-shaped test which can easily be implemented. We have demonstrated the performance of our planner in complex scenarios with low DOF robots.

There are a range of directions to pursue for future work. We are interested in the application of our algorithm to higher DOF motion planning. Our approach uses linear programming and interval arithmetic. Both these techniques are extensible to higher dimensional spaces. However, our current algorithm relies on contact surface enumeration, which becomes difficult for high DOF. We wish to extend the algorithm so that it no longer requires contact surface enumeration. Currently we use contact surfaces to perform the star-shaped query in configuration space. One direction worth pursuing is whether this query can be performed in the workspace instead of configuration space.

We would like to combine our algorithm with randomized sampling techniques to design a hybrid algorithm for high-dof robots. Such an algorithm would first use randomized sampling techniques to compute samples in the free space and then use star-shaped sampling only in those regions where randomized sampling techniques failed to capture the local connectivity.

A bottleneck in our approach is the large number of C-surfaces. Typically, our method enumerates $O(n^2)$ C-surfaces where n is the number of features in the robot and the obstacles. Many of these C-surfaces lie within C-obstacle and do not contribute to the actual boundary of the free space, thus adding an unnecessary overhead to the algorithm. We plan to develop better *culling* techniques to eliminate such C-surfaces.

Our current algorithm does not handle scenarios where the robot is allowed to be in contact with the boundary of the obstacle. We would like to extend our algorithm to handle such cases.

Appendix A

Overview of Interval Arithmetic

We provide a brief overview of interval arithmetic. Interval arithmetic is defined on a set of intervals rather than sets of real numbers. An interval $[a, b]$ is an ordered pair $a \leq b$ representing the range of numbers $\{x \mid a \leq x \leq b\}$. Arithmetic operations on intervals are guaranteed to return an interval containing all possible solutions over the given domain. The standard arithmetic operations can be extended to intervals as follows:

$$\begin{aligned}
[a, b] + [c, d] &= [a + b, c + d] \\
[a, b] - [c, d] &= [a - b, c - d] \\
[a, b] \times [c, d] &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\
[a, b]/[c, d] &= [a, b] \times [1/d, 1/c] \text{ if } 0 \notin [c, d]
\end{aligned}$$

Given a function $f : \mathbb{R}^d \to \mathbb{R}$, we can use the above rules to write an interval form $\overline{f} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \times \mathbb{R}$ that takes as input an interval for the domain and returns an interval for the range. For example, consider the function

$$f(x, y) = x(y - 1)$$

We can define an interval form $\overline{f}$ of the function f that takes a product of intervals as an input. For example, suppose x and y belong to the intervals $[a, b]$ and $[c, d]$ respectively. Then we have

$$\begin{aligned}
\overline{f}([a, b], [c, d]) &= [a, b]([c, d] - 1) \\
&= [a, b][c - 1, d - 1] \\
&= [\min(ac - a, ad - a, bc - b, bd - b), \max(ac - a, ad - a, bc - b, bd - b)]
\end{aligned}$$

We can use $\overline{f}$ to obtain bounds on the range of f over a product of intervals. For

example, on $[0,1] \times [0,1]$, we have $\overline{f}([0,1],[0,1]) = [-1,0]$, which contains the exact range $[-1/4, 0]$.

The range returned by interval arithmetic $Q = \overline{f}([a,b],[c,d])$ on $[a,b] \times [c,d]$ is guaranteed to contain the exact range $R = \{f(x,y) \mid x \in [a,b], y \in [c,d]\}$. Interval arithmetic is conservative: in general, Q is not equal to R.

Appendix B

Contact Surfaces

This appendix provides background on contact surfaces (C-surfaces). It describes methods for enumerating C-surfaces for two classes of robots: **2T+1R** - a planar rigid robot with 2 translational and 1 rotational dofs moving among polygonal obstacles, and **3R** - a planar articulated robot with 3 revolute joints moving among polygonal obstacles.

B.1 Configuration Space of a 2T+1R Planar Rigid Object

A detailed discussion of the configuration space of a planar rigid object is provided in (Latombe, 1991).

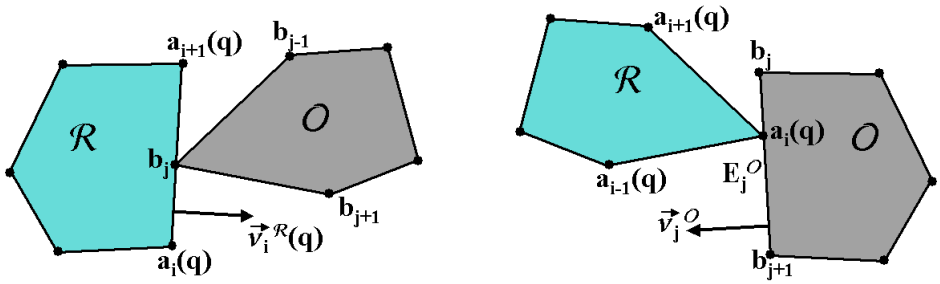
We assume that the robot $\mathcal{R}$ and the obstacle $\mathcal{O}$ are planar polygons that are defined with respect to their local coordinate frames $\mathcal{W}_\mathcal{R}$ and $\mathcal{W}_\mathcal{O}$ respectively. The configuration $\mathbf{q}$ of $\mathcal{R}$ is represented by $(x, y, \theta) \in \mathbf{R^2} \times [0, 2\pi)$, and x and y are the coordinates of the origin of $\mathcal{W}_\mathcal{R}$ with respect to the global coordinate frame $\mathcal{W}$. θ is the angle between the x-axes of $\mathcal{W}_\mathcal{R}$ and $\mathcal{W}_\mathcal{O}$. θ can "wrap around" from 2π to 0 and vice-versa (θ is mapped to θ mod 2π). Let us represent $\mathcal{R}$ with a set of vertices a_i and edges $E_i^\mathcal{R}$, and $\mathcal{O}$ with b_i and $E_i^\mathcal{O}$, each defined with respect to $\mathcal{W}_\mathcal{R}$ and $\mathcal{W}_\mathcal{O}$, respectively. Let $\vec{v}_i^\mathcal{R}(\mathbf{q})$ and $\vec{v}_j^\mathcal{O}$ be the outgoing normal vectors to $E_i^\mathcal{R}(\mathbf{q})$ and $E_j^\mathcal{O}$, respectively.

B.1.1 Contact Surfaces

We first consider the case in which both $\mathcal{R}$ and $\mathcal{O}$ are convex polygons. The non-convex case will be treated later.

Let us assume that $\mathcal{R}$ and $\mathcal{O}$ are in contact. A C-surface arises from two types of contacts between $\mathcal{R}$ and $\mathcal{O}$:

Type A Contact: A type A contact occurs when an edge $E_i^\mathcal{R}$ of $\mathcal{R}$ contains a vertex b_j of $\mathcal{O}$. See Fig. B.1(a). The constraint that the interiors of $\mathcal{R}$ and $\mathcal{O}$ do not overlap



(a) Type A contact (b) Type B contact

Figure B.1: **2T+1R:** *The figures shows the two types of contacts possible between the vertices and edges of* $\mathcal{R}$ *and* $\mathcal{O}$.

makes a type A contact between $E_i^{\mathcal{R}}$ and b_j feasible only for a subrange of orientations of $\mathcal{R}$. This subrange is determined by the following condition:

$$\text{APPL}_{i,j}^{A}(\mathbf{q}) = \quad [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j-1} - b_j) \geq 0] \quad \wedge \quad [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j+1} - b_j) \geq 0]$$

The above condition is called the *applicability condition* of type A contact between $E_i^{\mathcal{R}}$ and b_j.

If we displace $\mathcal{R}$ in such a way that the type A contact between $E_i^{\mathcal{R}}$ and b_j is maintained, then the robot's configuration moves along a surface in $\mathcal{C}$ whose equation is

$$f_{i,j}^{A}(\mathbf{q}) = 0$$

with:

$$f_{i,j}^{A}(\mathbf{q}) = \vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_j - a_i(\mathbf{q}))$$

The above equation represents a surface called a **C-surface** of type A. It is a ruled surface in $\mathcal{R}^2 \times [0, 2\pi)$. It separates $\mathcal{C}$ into two half spaces. The C-obstacle $\mathcal{CO}$ lies completely within the half space determined by $f_{i,j}^{A}(\mathbf{q}) \leq 0$.

DEFINITION 16 *The expression*

$$APPL_{i,j}^{A}(\mathbf{q}) \implies [f_{i,j}^{A}(\mathbf{q}) \leq 0]$$

is called a C-constraint of type A. It is denoted as $CONST_{i,j}^{A}(\mathbf{q})$.

Type B Contact: A type B contact occurs when a vertex a_i of $\mathcal{R}$ is contained in an edge $E_j^{\mathcal{O}}$of $\mathcal{O}$. See Fig. B.1(b). In the same manner as type A contact, a type B

contact between a_i and $E_j^{\mathcal{O}}$ is feasible only for a subrange of orientations of $\mathcal{R}$. This subrange is determined by the following condition:

$$\text{APPL}_{i,j}^{B}(\mathbf{q}) = \quad [(a_{i-1}(\mathbf{q}) - a_i(\mathbf{q})) \cdot \vec{v}_j^{\mathcal{O}} \geq 0] \wedge [(a_{i+1}(\mathbf{q}) - a_i(\mathbf{q})) \cdot \vec{v}_j^{\mathcal{O}} \geq 0]$$

The above condition is called the *applicability condition* of type B contact between a_i and $E_j^{\mathcal{O}}$.

If we displace $\mathcal{R}$ in such a way that the type A contact between a_i and $E_j^{\mathcal{O}}$ is maintained, then the robot's configuration moves along a surface whose equation is

$$f_{i,j}^{B}(\mathbf{q}) = 0$$

with:

$$f_{i,j}^{B}(\mathbf{q}) = \vec{v}_j^{\mathcal{O}} \cdot (a_i(\mathbf{q}) - b_j)$$

The above equation represents a surface called a **C-surface** of type B. Like C-surface of type A, it is a ruled surface in $\mathcal{R}^2 \times [0, 2\pi)$ and separates $\mathcal{C}$ into two half spaces. The C-obstacle $\mathcal{CO}$ lies completely within the half space determined by $f_{i,j}^{B}(\mathbf{q}) \leq 0$.

DEFINITION 17 *The expression*

$$APPL_{i,j}^{B}(\mathbf{q}) \implies [f_{i,j}^{B}(\mathbf{q}) \leq 0]$$

is called a C-constraint of type B. It is denoted as $CONST_{i,j}^{B}(\mathbf{q})$.

B.1.2 Representation of C-obstacle

The C-obstacle $\mathcal{CO}$ can be expressed in terms of the C-constraints. More precisely, we have the following theorem:

THEOREM 18 *Let* $\mathcal{R}$ *and* $\mathcal{O}$ *be two convex polygons. The C-obstacle:*

$$\mathcal{CO} = \{\mathbf{q} \in \mathcal{C} : \mathcal{R}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}$$

is such that

$$\mathbf{q} \in \mathcal{CO} \quad \iff \quad \text{CO}(\mathbf{q})$$

where

$$\text{CO}(\mathbf{q}) \equiv (\wedge_{ij} CONST_{i,j}^{A}(\mathbf{q})) \quad \wedge \quad (\wedge_{ij} CONST_{i,j}^{B}(\mathbf{q}))$$

The proof of the above theorem is given in Chapter 3 of (Latombe, 1991).

The C-obstacle is a three dimensional volume in $\mathcal{R}^2 \times [0, 2\pi)$. The boundary of C-obstacle consists of patches of type A and type B C-surfaces.

If $\mathcal{R}$ and $\mathcal{O}$ are non-convex polygons, they can be represented as finite unions of convex polygons $\mathcal{R}_k$ and $\mathcal{O}_l$, i.e., $\mathcal{R} = \bigcup_k \mathcal{R}_k$ and $\mathcal{O} = \bigcup_l \mathcal{O}_l$. By defining:

$$\mathcal{CO}_{kl} = \{\mathbf{q} \in \mathcal{C} \quad | \quad \mathcal{R}_k(\mathbf{q}) \cap \mathcal{O}_l \neq \emptyset\}$$

we get:

$$\mathcal{CO} = \bigcup_{k,l} \mathcal{CO}_{kl}$$

Therefore, even in the case of non-convex polygons, the C-obstacle is a three-dimensional volume in $\mathcal{R}^2 \times [0, 2\pi)$, bounded by patches of C-surfaces.

B.2 Configuration Space of a 3R Planar Articulated Object

Let $\mathcal{R}$ be a planar articulated robot with three revolute joints (see Fig. B.2). Let us represent $\mathcal{R}$ with a set of vertices $a_i, i = 0, \ldots, 3$ and edges $E_i^{\mathcal{R}}, i = 1, \ldots, 3$ defined with respect to a coordinate system $\mathcal{W}_{\mathcal{R}}$. The vertices a_i correspond to joints while the edges $E_i^{\mathcal{R}}$ correspond to links of the articulated robot. Each joint a_i is associated with a joint angle θ_i. The joint angle θ_1 is the angle between $E_1^{\mathcal{R}}$ and one of the axes of $\mathcal{W}_{\mathcal{R}}$. The joint angle $\theta_i, i = 2, 3$ measures the angle between $E_{i-1}^{\mathcal{R}}$ and $E_i^{\mathcal{R}}$. See Fig. B.2(a). The configuration $\mathbf{q}$ of $\mathcal{R}$ is represented by $(\theta_1, \theta_2, \theta_3) \in [0, 2\pi)^3$. Angle θ_1 can "wrap around" from 2π to 0 and vice-versa. The obstacle $\mathcal{O}$ is represented as in Sec. B.1. Let $\vec{v}_i^{\mathcal{R}}(\mathbf{q})$ and $\vec{v}_j^{\mathcal{O}}$ be the outgoing normal vectors to $E_i^{\mathcal{R}}(\mathbf{q})$ and $E_j^{\mathcal{O}}$, respectively. Here $\vec{v}_i^{\mathcal{R}}(\mathbf{q})$ represents the normal to a link in the counter-clockwise direction.

B.2.1 Contact Surfaces

We first consider the case where $\mathcal{O}$ is a convex polygon. The non-convex case will be treated later.

Type A Contact: This type of contact occurs when an edge $E_i^{\mathcal{R}}$ of $\mathcal{R}$ contains a vertex b_j of $\mathcal{O}$. See Fig. B.2(b). The constraint that the interiors of $\mathcal{R}$ and $\mathcal{O}$ do not overlap makes a contact between $E_i^{\mathcal{R}}$ and b_j feasible only for a subrange of orientations

of $\mathcal{R}$. In general, there can be two sub-ranges for which a type A contact between $E_i^{\mathcal{R}}$ and b_j occurs. These sub-ranges are determined by the following conditions:

$$\text{APPL}_{i,j}^{A1}(\mathbf{q}) = \quad [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j-1} - b_j) \geq 0] \wedge [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j+1} - b_j) \geq 0]$$

We refer to this type of contact as as a type A1 contact. Similarly, a contact can also occur for another range of orientation of the robot.

$$\text{APPL}_{i,j}^{A2}(\mathbf{q}) = \quad [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j-1} - b_j) \leq 0] \wedge [\vec{v}_i^{\mathcal{R}}(\mathbf{q}) \cdot (b_{j+1} - b_j) \leq 0]$$

We refer to this type of contact as as a type A2 contact. For any orientation of $\mathcal{R}$, atmost one of type A1 or type A2 contact is possible between $E_i^{\mathcal{R}}$ and b_j: Both can never occur simultaneously. We refer to the above conditions as the *applicability condition* of type A contact between $E_i^{\mathcal{R}}$ and b_j.

If we displace $\mathcal{R}$ in such a way that the type A contact between $E_i^{\mathcal{R}}$ and b_j is maintained, then the robot's configuration moves along a surface in $\mathcal{C}$ whose equation is

$$f_{i,j}^{\mathcal{R}}(\mathbf{q}) = 0$$

with:

$$f_{i,j}^{A}(\mathbf{q}) = \vec{v}_i^{\mathcal{R}} \cdot (b_j - a_i(\mathbf{q}))$$

The above equation represents a surface called a **C-surface** of type A. In general, the C-surface consists of two components, one corresponding to each of type A1 and type A2.

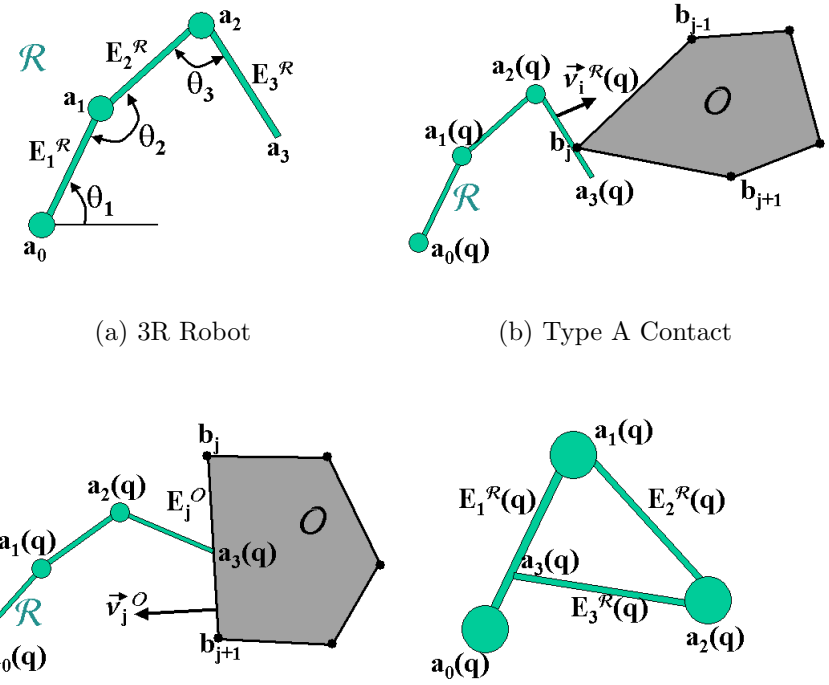
If a configuration $\mathbf{q}$ satisfies $\text{APPL}_{i,j}^{A1}$, then $\mathbf{q} \in \mathcal{CO} \implies f_{i,j}^{A}(\mathbf{q}) \leq 0$. On the other hand, if $\mathbf{q}$ satisfies $\text{APPL}_{i,j}^{A2}$, then $\mathbf{q} \in \mathcal{CO} \implies f_{i,j}^{A}(\mathbf{q}) \geq 0$.

DEFINITION 18 *The expression*

$$APPL_{i,j}^{A1}(\mathbf{q}) \implies [f_{i,j}^{A}(\mathbf{q}) \leq 0] \quad \wedge \quad APPL_{i,j}^{A2}(\mathbf{q}) \implies [f_{i,j}^{A}(\mathbf{q}) \geq 0]$$

is called a C-constraint of type A. It is defined by $CONST_{i,j}^{A}(\mathbf{q})$.

Type B Contact: A type B contact occurs when a vertex a_i of $\mathcal{R}$ is contained in an edge $E_j^{\mathcal{O}}$of $\mathcal{O}$. See Fig. B.2(c). The contact between a_i and $E_j^{\mathcal{O}}$ is feasible only for a sub-range of orientations of $\mathcal{R}$. This sub-range is determined by the following



(a) 3R Robot

(b) Type A Contact

(c) Type B Contact

(d) Type C Contact

Figure B.2: **3R:** *Fig. (a) shows a planar articulated robot with* 3 *revolute joints. Figs. (b) and (c) show the two types of contacts possible between the vertices and edges of* $\mathcal{R}$ *and* $\mathcal{O}$*. Fig. (d) shows a type C contact that arises when one link of the robot touches another link, resulting in robot self-intersection.*

condition:

$$\text{APPL}^B_{i,j}(\mathbf{q}) = \quad [(a_{i-1}(\mathbf{q}) - a_i(\mathbf{q})) \cdot \vec{v}^{\mathcal{O}}_j \geq 0] \wedge [(a_{i+1}(\mathbf{q}) - a_i(\mathbf{q})) \cdot \vec{v}^{\mathcal{O}}_j \geq 0]^1$$

The above condition is called the *applicability condition* of type B contact between a_i and $E^{\mathcal{O}}_j$.

If we displace $\mathcal{R}$ in such a way that the type B contact between a_i and $E^{\mathcal{O}}_j$ is maintained, then the robot's configuration moves along a surface whose equation is

$$f^B_{i,j}(\mathbf{q}) = 0$$

[1]We assume that $a_4 = a_3$; hence when the above condition is applied to vertex a_3, the second part of the *and* condition reduces to true.

with:

$$f^B_{i,j}(\mathbf{q}) = \vec{v}^{\mathcal{O}}_j \cdot (a_i(\mathbf{q}) - b_j)$$

The above equation epresents a surface called a **C-surface** of type B. It separates $\mathcal{C}$ into two half spaces. The C-obstacle $\mathcal{CO}$ lies completely within the half space determined by $f^B_{i,j}(\mathbf{q}) \leq 0$.

DEFINITION 19 *The expression*

$$APPL^B_{i,j}(\mathbf{q}) \implies [f^B_{i,j}(\mathbf{q}) \leq 0]$$

is called a C-constraint of type B. It is defined by $CONST^B_{i,j}(\mathbf{q})$.

The C-obstacle w.r.t the i'th link of $\mathcal{R}$ can be expressed in terms of the type A constraint derived from edge $E^{\mathcal{R}}_i$ as well as the type B constraints derived from the vertices a_{i-1} and a_i. Thus the C-obstacle of the i'th link of the robot

$$\mathcal{CO}_i = \{\mathbf{q} \in \mathcal{C} : E^{\mathcal{R}}_i(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}$$

is such that

$$\mathbf{q} \in \mathcal{CO}_i \quad \iff \quad \mathrm{CO_i}(\mathbf{q})$$

where

$$\mathrm{CO_i}(\mathbf{q}) \equiv (\wedge_j \mathrm{CONST}^A_{i-1,j}(\mathbf{q})) \quad \wedge \quad (\wedge_j \mathrm{CONST}^A_{i,j}(\mathbf{q})) \quad \wedge \quad (\wedge_j \mathrm{CONST}^B_{i,j}(\mathbf{q}))$$

The C-obstacle is given by the union of the C-obstacles of the individual links. We have the following theorem:

THEOREM 19 *Let* $\mathcal{O}$ *be a convex polygon. The C-obstacle is given by*

$$\mathcal{CO} = \bigcup_i \mathcal{CO}_i$$

If $\mathcal{O}$ is a non-convex polygon, then it can be represented as finite unions of convex polygons $\mathcal{O}_l$, i.e., $\mathcal{O} = \bigcup_l \mathcal{O}_l$. By defining

$$\mathcal{CO}(l) = \{\mathbf{q} \in \mathcal{C} \quad | \quad \mathcal{R}(\mathbf{q}) \cap \mathcal{O}_l \neq \emptyset\}$$

we get:

$$\mathcal{CO} = \bigcup_{l} \mathcal{CO}(l)$$

B.2.2 Robot Self-Intersection

In addition to preventing the robot from colliding with any obstacle, we also need to ensure that the different links of the articulated robot do not collide with each other. This means the robot cannot assume a certain set of configurations. This set is given by

$$\mathcal{CR} = \{\mathbf{q} \in \mathcal{C} \quad | \quad E_i^{\mathcal{R}}(\mathbf{q}) \cap E_j^{\mathcal{R}}(\mathbf{q}) \neq \emptyset,\ i \in \{1,2,3\},\ j \in \{1,2,3\},\ i \neq j\}$$

We now enumerate contact surfaces that arise due to contact between pairs of links of the robot. There are three cases corresponding to the pairs of links involved in contact: $(E_1^{\mathcal{R}}, E_2^{\mathcal{R}})$, $(E_2^{\mathcal{R}}, E_3^{\mathcal{R}})$, or $(E_1^{\mathcal{R}}, E_3^{\mathcal{R}})$. We can prevent link pairs $(E_1^{\mathcal{R}}, E_2^{\mathcal{R}})$ and $(E_2^{\mathcal{R}}, E_3^{\mathcal{R}})$ from touching each other by preventing the corresponding orientation angle to wrap around – more precisely, we restrict both θ_2 and θ_3 to lie within the open interval $(0, 2\pi)$. Hence we only need to consider the case where the link pair $(E_1^{\mathcal{R}}, E_3^{\mathcal{R}})$ is involved in a contact.

Two types of contacts can occur between ($E_1^{\mathcal{R}}$ and $E_3^{\mathcal{R}}$): (a) A contact between vertex a_3 and edge $E_1^{\mathcal{R}}$ and (b) A contact between vertex a_0 and edge $E_3^{\mathcal{R}}$. We refer to these contacts as type C contacts. See Fig. B.2(d).

Type C Contact: Suppose vertex a_3 lies on edge $E_1^{\mathcal{R}}$. The contact between a_3 and $E_1^{\mathcal{R}}$ is feasible only for a sub-range of orientations of $\mathcal{R}$. In general, there can be two sub-ranges, which are determined by the following conditions:

$$\begin{aligned} \text{APPL}_{3,1}^{C1}(\mathbf{q}) = &\quad [\ \theta_2 \leq \pi/2\ ] \\ \text{APPL}_{3,1}^{C2}(\mathbf{q}) = &\quad [\ \theta_2 \geq 3\pi/2\ ] \end{aligned}$$

We refer to these types of contacts as type C1 and type C2 contacts. For any orientation of $\mathcal{R}$, atmost one of them is possible between a_3 and $E_1^{\mathcal{R}}$. We refer to the above conditions as the *applicability condition* of type C contact between a_3 and $E_1^{\mathcal{R}}$. See Fig. B.2(d).

If we displace $\mathcal{R}$ in such a way that this contact is maintained, then the robot's

configuration moves along a surface whose equation is

$$f^C(\mathbf{q}) = 0$$

with:

$$f^C(\mathbf{q}) = \vec{v}_1^{\mathcal{R}} \cdot (a_3(\mathbf{q}) - a_1(\mathbf{q}))$$

The above equation represents a surface called a **C-surface** of type C. In general, the C-surface consists of two components, one corresponding to each of type C1 and type C2.

For a configuration $\mathbf{q}$ satisfies $\text{APPL}_{3,1}^{C1}$, $\mathbf{q} \in \mathcal{CR} \implies f^C(\mathbf{q}) \leq 0$. On the other hand, if $\mathbf{q}$ satisfies $\text{APPL}_{3,1}^{C2}$, , then $\mathbf{q} \in \mathcal{CR} \implies f^C(\mathbf{q}) \geq 0$.

The above types of contacts lead to the following C-constraint.

DEFINITION 20 *The expression*

$$APPL_{3,1}^{C1}(\mathbf{q}) \implies [f^C(\mathbf{q}) \leq 0] \quad \wedge \quad APPL_{3,1}^{C2}(\mathbf{q}) \implies [f^C(\mathbf{q}) \geq 0]$$

is called a C-constraint of type C. It is denoted as $CONST_{13}^{C}(\mathbf{q})$.

The C-constraint corresponding to the contact between vertex a_0 and edge $E_3^{\mathcal{R}}$ can be treated similarly. Let us denote the resulting C-constraint by $\text{CONST}_{31}^{C}(\mathbf{q})$.

$\mathcal{CR}$ can be defined as follows.

$$\mathcal{CR} = \{\mathbf{q} \in \mathcal{C} \quad | \quad E_i^{\mathcal{R}} \cap E_j^{\mathcal{R}} \neq \emptyset,\ i \in \{1,2,3\},\ j \in \{1,2,3\},\ i \neq j\}$$

is such that

$$\mathbf{q} \in \mathcal{CR} \quad \iff \quad \text{CR}(\mathbf{q})$$

where

$$\text{CR}(\mathbf{q}) \equiv (\theta_2 == 0) \quad \vee \quad (\theta_3 == 0) \quad \vee \quad (\text{CONST}_{13}^{C}(\mathbf{q}) \ \wedge \ \text{CONST}_{31}^{C}(\mathbf{q}))$$

The forbidden region of $\mathcal{R}$ is then defined as $\mathcal{CO}' = \mathcal{CO} \cup \mathcal{CR}$.

Bibliography

Abrams, S. and Allen, P. (2000). Computing swept volumes. *Journal of Visualization and Computer Animation*, 11.

Agarwal, P. K., Amenta, N., Aronov, B., and Sharir, M. (1997). Largest placements and motion planning of a convex polygon. In Laumond, J.-P. and Overmars, M. H., editors, *Algorithms for Robotic Motion and Manipulation*, pages 143–154, Wellesley, MA. A. K. Peters.

Agarwal, P. K., Flato, E., and Halperin, D. (2002). Polygon decomposition for efficient construction of minkowski sums. *Comput. Geom. Theory Appl.*, 21(1):39–61.

Agarwal, P. K., Guibas, L. J., Har-Peled, S., Rabinovitch, A., and Sharir, M. (2000). Computing the penetration depth of two convex polytopes in 3d. In *Scandinavian Workshop on Algorithm Theory*, pages 328–338.

Agarwal, P. K. and Sharir, M. (1999). Pipes, cigars, and kreplach: The union of Minkowski sums in three dimensions. In *ACM Symposium on Computational Geometry*, pages 143–153.

Agarwal, P. K. and Sharir, M. (2000). Arrangements and their applications. In Sack, J.-R. and Urrutia, J., editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam.

Amato, N., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. (1998). OBPRM: An obstacle-based PRM for 3D workspaces. In Agarwal, P. K., Kavraki, L. E., and Mason, M., editors, *Workshop on Algorithmic Foundations of Robotics*. A. K. Peters, Wellesley, MA.

Amenta, N. and Bern, M. (1998). Surface reconsruction by Voronoi filtering. In *ACM Symposium on Computational Geometry*, pages 39–48.

Aronov, B. and Sharir, M. (1994). On translational motion planning in 3-space. In *ACM Symposium on Computational Geometry*, pages 21–30.

Aronov, B. and Sharir, M. (1997). On translational motion planning of a convex polyhedron in 3-space. *SIAM Journal on Computing*, 26:1785–1803.

Aronov, B., Sharir, M., and Tagansky, B. (1997). The union of convex polyhedra in three dimensions. *SIAM Journal on Computing*, 26:1670–1688.

Artzy, E., Frieder, G., and Herman, G. T. (1981). The theory, design, implementation, and evaluation of 3-d surface detection algorithms. *Comput. Graph. Image Process.*, 15:1–24.

Aspert, N., Santa-Cruz, D., and Ebrahimi, T. (2002). Mesh: Measuring error between surfaces using the hausdorff distance,. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 705–708.

Avnaim, F. and Boissonnat, J.-D. (1989). Practical exact motion planning of a class of robots with three degrees of freedom. In *Proc. of Canadian Conference on Computational Geometry*, page 19.

Bajaj, C. and Kim, M. (1987). Generation of configuration space obstacles III: The case of moving algebraic curves. In *Proc. 4th IEEE Internat. Conf. Robot. Autom.*

Bajaj, C. L. and Kim, M.-S. (1990). Generation of configuration space obstacles. *I. J. Robotic Res.*, 9(1):92–112.

Bajaj, C. L., Pascucci, V., and Schikore, D. R. (1996). Fast isocontouring for improved interactivity. In *1996 Volume Visualization Symposium*, pages 39–46. IEEE. ISBN 0-89791-741-3.

Barraquand, J., Kavraki, L., Latombe, J.-C., Motwani, R., Li, T.-Y., and Raghavan, P. (1997). A random sampling scheme for path planning. *Int. J. Rob. Res.*, 16(6):759–774.

Basch, J., Guibas, L., Ramkumar, G., and Ramshaw, L. (1996). Polyhedral tracings and their convolutions. In *Proc. Workshop on the Algorithmic Foundations of Robotics*.

Basch, J., Guibas, L. J., Hsu, D., and Nguyen, A. T. (2001). Disconnection proofs for motion planning. In *ICRA*, pages 1765–1772.

Basu, S. (1998). On the combinatorial and topological complexity of a single cell.

Basu, S., Pollack, R., and Roy, M.-F. (1996). Computing roadmaps of semi-algebraic sets. pages 168–173.

Bekker, H. and Roerdink, J. B. T. M. (2001). An efficient algorithm to calculate the minkowski sum of convex 3d polyhedra. In *ICCS '01: Proceedings of the International Conference on Computational Sciences-Part I*, pages 619–628, London, UK. Springer-Verlag.

Bhaniramka, P., Wenger, R., and Crawfis, R. (2000). Isosurfacing in higher dimensions. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*, Washington, DC, USA. IEEE Computer Society.

Bhaniramka, P., Wenger, R., and Crawfis, R. (2004). Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):130–141.

Bischoff, S. and Kobbelt, L. (2002). Isosurface reconstruction with topology control. *Proc. of Pacific Graphics*, pages 246–255.

Blinn, J. F. (1982). A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256.

Bloomenthal, J. (1988). Polygonization of implicit surfaces. *Comput. Aided Geom. Design*, 5(4):341–355.

Bloomenthal, J., editor (1997). *Introduction to Implicit Surfaces*, volume 391. Morgan-Kaufmann.

Bloomenthal, J. and Ferguson, K. (1995). Polygonization of Non-Manifold implicit surfaces. In *SIGGRAPH 95 Conference Proceedings*, pages 309–316.

Boissonnat, J.-D., Cohen-Steiner, D., and Vegter, G. (2004). Isotopic implicit surface meshing. *STOC*.

Boissonnat, J.-D., de Lange, E., and Teillaud, M. (1997). Minkowski operations for satellite antenna layout. In *Symposium on Computational Geometry*, pages 67–76.

Boor, V., Overmars, M. H., and van der Stappen, A. F. (1999). The gaussian sampling strategy for probabilistic roadmap planners. In *ICRA*, pages 1018–1023.

Bottino, A., Nuij, W., and van Overveld, K. (1996). How to shrinkwrap through a critical point: An algorithm for the adaptive tesselation of iso-surfaces with arbitrary topology. *Implicit Surfaces*, pages 53–72.

Branicky, M., Lavalle, S., Olson, K., and Yang, L. (2001). Quasirandomized path planning.

Breen, D. and Mauch, S. (1999). Generating shaded offset surfaces with distance, closest-point and color volumes. In *Proceedings of the International Workshop on Volume Graphics*, pages 307–320.

Breen, D., Mauch, S., and Whitaker, R. (2000). 3d scan conversion of csg models into distance, closest-point and color volumes. *Proc. of Volume Graphics*, pages 135–158.

Brooks, R. A. and Lozano-Pérez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. Syst*, SMC-15:224–233.

Brost, R. (1991). *Analysis and planning of planar manipulation tasks*. PhD thesis, Carnegie Mellon University. Report CMU-CS-91-149.

Cameron, S. (1997). Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, pages 3112–3117.

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

Canny, J. (1988). *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press.

Canny, J. F. and Donald, B. (1988). Simplified voronoi diagrams. *Discrete and Computational Geometry*, 3:219–236.

Carr, H. (2004). *Topological Manipulation of Isosurfaces*. PhD thesis, The University of British Columbia.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA. ACM Press.

Chazelle, B. (1981). Convex decompositions of polyhedra. In *ACM Symposium on Theory of Computing*, pages 70–79.

Chazelle, B., Dobkin, D., Shouraboura, N., and Tal, A. (1997). Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications*, 7:327–342.

Chen, Y., Wang, H., Rosen, D., and Rossignac, J. (2005). A point-based offsetting method of polygonal meshes. In *GVU Tech Report GIT-GVU-05*.

Choset, H., Burgard, W., Hutchinson, S., Kantor, G., Kavraki, L. E., Lynch, K., and Thrun, S. (2004). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

Christiansen, H. N. and Sederberg, T. W. (1978). Conversion of complex contour line definitions into polygonal element mosaics. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 187–192, New York, NY, USA. ACM Press.

Chun-Yi, H., Patrikalakis, N. M., and Ye, X. (1996). Robust interval solid modelling part i: representations. *Computer-Aided Design*, 28(10):807–817.

Cignoni, P., Ganovelli, F., Montani, C., and Scopigno, R. (2000). Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24(3):399–418.

Cignoni, P., Montani, C., Puppo, E., and Scopigno, R. (1996). Optimal isosurface extraction from irregular volume data. In *1996 Volume Visualization Symposium*, pages 31–38. IEEE. ISBN 0-89791-741-3.

Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W. (1996). Simplification envelopes. In *Proc. of ACM Siggraph'96*, pages 119–128.

Collins, G. E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, volume 33, pages 134–183. Springer-Verlag.

Cook, L. T., III, S. J. D., Batnitzky, S., and Lee, K. R. (1983). A three-dimensional display system for diagnostic imaging applications. *IEEE Comput. Graphics Applications*, 3:13–19.

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In Rushmeier, H., editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 303–312. ACM SIGGRAPH, Addison Wesley. held in New Orleans, Louisiana, 04-09 August 1996.

Daniels, K. and Milenkovic, V. (1995). Multiple translational containment: Approximate and exact algorithms. pages 205–214.

Daniels, K. and Milenkovic, V. (1997). Multiple translational containment: Approximate and exact algorithms. In *Algorithmica 19:148-182*.

de Berg, M., van Kreveld, M., Overmars, M. H., and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition.

de Figueiredo, L. H. (1996). Surface intersection using affine arithmetic. In *GI '96: Proceedings of the conference on Graphics interface '96*, pages 168–175, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.

Dobkin, D., Hershberger, J., Kirkpatrick, D., and Suri, S. (1993). Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533.

Donald, B. R. (1984). Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab. AI-TR-791.

Donald, B. R. (1987). A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353.

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

Edelsbrunner, H. and Mücke, E. P. (1987). Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. Technical Report UIUCDCD-R-87-1393, Dept. Comput. Sci., Univ. Illinois, Urbana-Champaign, IL.

Edelsbrunner, H. and Shah, N. R. (1994). Triangulating topological spaces. In *ACM Symposium on Computational Geometry*, pages 285–292.

Ehmann, S. and Lin, M. C. (2001). Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3):500–510.

El-Sana, J. and Varshney, A. (1997). Controlled simplification of genus for polygonal models. *Proc. of IEEE Visualization*, pages 403–410.

Elber, G. and Cohen, E. (1997). Filleting and rounding using trimmed tensor product surfaces. In *SMA '97: Proceedings of the fourth ACM symposium on Solid modeling and applications*, pages 206–216, New York, NY, USA. ACM Press.

Emiris, I. and Canny, J. (1991). A general approach to removing degeneracies. pages 405–413.

Evans, R. C., Koppelman, G., and Rajan, V. T. (1987). Shaping geometric objects by cumulative translational sweeps. *IBM Journal of Research and Development*, 31:343–360.

Evans, R. C., O'Connor, M. A., and Rossignac, J. R. (1992). Construction of minkowski sums and derivatives morphological combinations of arbitrary polyhedra in cad/cam systems. *US Patent 5159512*.

Ezra, E. (2005). Almost tight bound for a single cell in an arrangement of convex polyhedra in r3. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, pages 22–31, New York, NY, USA. ACM Press.

Fang, S., Bruderlin, B., and Zhu, X. (1993). Robustness in solid modeling: a tolerance-based intuitionistic approach. *Computer-Aided Design*, 25(9):567–576.

Farouki, R. (1985). Exact offset procedures for simple solids. *Comput. Aided Geom. Design*, 2:257–279.

Farouki, R. T. and Sakkalis, T. (1990). Pythagorean hodographs. *IBM J. Res. Dev.*, 34(5):736–752.

Favre, J. M. (1997). Towards efficient visualization support for single-block and multi-block datasets. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 425–ff., Los Alamitos, CA, USA. IEEE Computer Society Press.

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

Fogel, E. and Halperin, D. (2005). Exact minkowski sums of convex polyhedra. In *Symposium on Computational Geometry*, pages 382–383.

Forsyth, M. (1995). Shelling and offsetting bodies. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pages 373–381, New York, NY, USA. ACM Press.

Fortune, S. (1997). Polyhedral modeling with multiprecision integer arithmetic. *Comput. Aided Design*, 29(2):123–133.

Fortune, S. and Van Wyk, C. J. (1993). Efficient exact arithmetic for computational geometry. In *ACM Symposium on Computational Geometry*, pages 163–172.

Frisken, S., Perry, R., Rockwood, A., and Jones, R. (2000). Adaptively sampled distance fields: A general representation of shapes for computer graphics. In *Proc. of ACM SIGGRAPH*, pages 249–254.

Fuchs, H., Kedem, Z. M., and Uselton, S. P. (1977). Optimal surface reconstruction from planar contours. *Commun. ACM*, 20(10):693–702.

Gerstner, T. and Pajarola, R. (2000). Topology preserving and controlled topology simplifying multi-resolution isosurface extraction. *Proc. of IEEE Visualization*, pages 259–266.

Ghosh, P. (1993). A unified computational framework for minkowski operations. In *Computers and Graphics, 17(4), pp.357-378.*

Ghosh, S. K. and Mount, D. M. (1987). An output sensitive algorithm for computing visibility graphs. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 11–19.

Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects. *Internat. J. Robot. Autom.*, 4(2):193–203.

GLPK (2003). Gnu linear programming kit, url:http://www.gnu.org/software/glpk/glpk.html.

Goyal, S., Ruina, A., and Papadopaulos, J. (1991). Planar sliding with dry friction i: Limit surface and moment function. In *Wear 143 (2), 307-330*.

Gueziec, A. and Hummel, R. (1995). Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328–342. ISSN 1077-2626.

Guibas, L., Ramshaw, L., and Stolfi, J. (1983). A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*

Guibas, L. and Seidel, R. (1987). Computing convolutions by reciprocal search. *Discrete Comput. Geom*, 2:175–193.

Guibas, L. J. and Hershberger, J. (1985). Computing the visibility graph of n line segments in $O(n^2)$ time. *Bull. EATCS*, 26:13–20.

Guskov, I. and Wood, Z. (2001). Topological noise removal. *Proc. of Graphics Interface.*

Guthe, M., Borodin, P., and Klein, R. (2005). Fast and accurate hausdorff distance calculation between meshes. volume 13, pages 41–48.

Hadwiger, H. (1957). Vorlesungen ber inhalt, oberflche, und isoperimetrie. *Berlin: Springer Verlag.*

Hall, M. and Warren, J. (1990). Adaptive polygonalization of implicitly defined surfaces. *IEEE Comput. Graph. Appl.*, 10(6):33–42.

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

Halperin, D. (2002a). Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232.

Halperin, D. (2002b). Robust geometric computing in motion. *International Journal of Robotics Research, 21(3).*

Halperin, D. and Sharir, M. (1995a). Almost tight upper bounds for the single cell and zone problems in three dimensions. 14:385–410.

Halperin, D. and Sharir, M. (1995b). Arrangements and their applications in robotics: Recent developments. In Goldberg, K., Halperin, D., Latombe, J.-C., and Wilson, R., editors, *Algorithmic Foundations of Robotics*, pages 495–511. A. K. Peters, Wellesley, MA.

Hartquist, E. E., Menon, J., Suresh, K., Voelcker, H. B., and Zagajac, J. (1999). A computing strategy for applications involving offsets, sweeps, and minkowski operations. *Computer-Aided Design*, 31(3):175–183.

He, T., Hong, L., Varshney, A., and Wang, S. (1996). Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184.

Heermann, P. D. (1998). Production visualization for the asci one teraflops machine. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 459–462, Los Alamitos, CA, USA. IEEE Computer Society Press.

Herman, G. and Liu, H. (1979). Three-dimensional display of human organs from computed tomograms. *Comp. Graph. and Image Processing*, 9:1–21.

Herman, G. T. and Udupa, J. K. (1983). Display of 3-D digital images: Computational foundations and medical applications. *IEEE Computer Graphics and Applications*, 3:39–46.

Hillyard, R. C. (1982). The build group of solid modellers. *IEEE Computer Graphics and Applications*, 2:43–52.

Hoffmann, C. (1989a). *Geometric and Solid Modeling*. Morgan-Kaufmann, San Mateo, CA.

Hoffmann, C. (1989b). *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California.

Hoffmann, C. (2001). Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1:143–156.

Hohmeyer, M. E. (1991). A surface intersection algorithm based on loop detection. 1(4):473–490.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78.

Hsu, D., Jiang, T., Reif, J., and Sun, Z. (2003a). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4420–4426.

Hsu, D., Jiang, T., Reif, J. H., and Sun, Z. (2003b). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *ICRA*, pages 4420–4426.

Hsu, D., Kavraki, L. E., Latombe, J.-C., Motwani, R., and Sorkin, S. (1998). On finding narrow passages with probabilistic roadmap planners. Wellesley, MA. A. K. Peters. To appear.

Hsu, D., Latombe, J.-C., and Motwani, R. (1999). Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9((4 & 5)):495–512.

Itoh, T. and Koyamada, K. (1995). Automatic isosurface propagation using an extrema graph and sorted boundart cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327. ISSN 1077-2626.

Jackson, D. (1995). Boundary representation modeling with local tolerances. *Proc. ACM Symposium on Solid Modeling and Applications*, pages 247–253.

Joe, B. (1991). Geompack. A software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, 13(5–6):325–331.

Joskowitz, L. and Sacks, E. (1995). HIPAIR: Interactive mechanism analysis and design using configuration spaces. In *ACM Symposium on Computational Geometry*, pages V5–V6.

Ju, T., Losasso, F., Schaefer, S., and Warren, J. (2002). Dual contouring of hermite data. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 21(3).

Kalra, D. and Barr, A. H. (1989). Guaranteed ray intersections with implicit surfaces. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 297–306.

Kambhampati, S. and Davis, L. S. (1986). Multiresolution path planning for mobile robots. *Internat. J. Robot. Autom.*, RA-2(3):135–145.

Kaul, A. and Farouki, R. T. (1995). Computing minkowski sums of plane curves. *Int. J. Comput. Geometry Appl.*, 5(4):413–432.

Kaul, A., O'Connor, M., and Srinivasan, V. (1991). Computing minkowski sums of regular polygons. *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 74–77.

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

Kaul, A. and Rossignac, J. (1991). Solid-interpolating deformations: Construction and animation of PIPs. In Purgathofer, W., editor, *Eurographics '91*, pages 493–505. North-Holland.

Kavraki, L., Kolountzakis, M., and Latombe, J. (1996a). Analysis of probabilistic roadmaps for path planning.

Kavraki, L. and Latombe, J. C. (1994). Randomized preprocessing of configuration space for fast path planning. *IEEE Conference on Robotics and Automation*, pages 2138–2145.

Kavraki, L., Svestka, P., Latombe, J. C., and Overmars, M. (1996b). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580.

Kedem, K. and Sharir, M. (1990). An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. *Discrete Comput. Geom.*, 5:43–75.

Keyser, J. (2000). *Exact Boundary Evaluation for Curved Solids*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill.

Keyser, J., Krishnan, S., and Manocha, D. (1997). Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. In *Proceedings of ACM Solid Modeling*. To appear.

Kim, M.-S. and Sugihara, K. (2001). Minkowski sums of axis-parallel surfaces of revolution defined by slope-monotone closed curves. In *IEICE Transactions on Information and Systems*, pages 1540–1547.

Kim, Y. J., Lin, M. C., and Manocha, D. (2002). Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics*.

Kobbelt, L., Botsch, M., Schwanecke, U., and Seidel, H. P. (2001). Feature-sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH*, pages 57–66.

Koide, A., Doi, A., and Kajioka, K. (1986). Polyhedral approximation approach to molecular orbital graphics. *J. Mol. Graph.*, 4(3):149–155.

Kriezis, G., Prakash, P., and Patrikalakis, N. (1990). Method for intersecting algebraic surfaces with rational polynomial patches. *Computer-Aided Design*, 22(10):645–654.

Krishnan, S. (1997). *Efficient and Accurate Boundary Evaluation Algorithms for Sculptured Solids*. PhD thesis, Department of Computer Science, University of N. Carolina at Chapel Hill. Available at http://www.cs.unc.edu/∼krishnas/dissertation.html.

Krishnan, S., Gopi, M., Manocha, D., and Mine, M. (1997). Interactive boundary evaluation on boolean combinations of sculptured solids. *Computer Graphics Forum*, 16(3):C67–C78. Proc. of Eurographics'97.

Lane, J. M. and Riesenfeld, R. R. (1980). A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-2(1).

Lanzagorta, M., Kral, M. V., J. Edward Swan, I., Spanos, G., Rosenberg, R., and Kuo, E. (1998). Three-dimensional visualization of microstructures. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 487–490, Los Alamitos, CA, USA. IEEE Computer Society Press.

Latombe, J. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

Latombe, J. (1999). Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, pages 1119–1128.

Laumond, J.-P. (1987). Obstacle growing in a nonpolygonal world. *Information Processing Letters*, 25:41–50.

LaValle, S. M. and Kuffner, J. J. (2000). Rapidly-exploring random trees: Progress and prospects. *Robotics: The Algorithmic Perspective (Proc. of the 4th Int'l Workshop on the Algorithmic Foundations of Robotics*.

Lee, D. T. (1978). Proximity and reachability in the plane. Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL.

Lee, I., Kim, M., and Elber, G. (1997). New approximation methods of planar offset and convolution curves. In *Geometric Modeling: Theory and Practice*.

Lee, I.-K., Kim, M.-S., and Elber, G. (1998). Polynomial/rational approximation of minkowski sum boundary curves. *Graphical Models and Image Processing*, 60(2):136–165.

Leven, D. and Sharir, M. (1985). An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. In *ACM Symposium on Computational Geometry*, pages 221–227.

Lin, M. and Manocha, D. (2003). Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.

Livnat, Y., Shen, H.-W., and Johnson, C. R. (1996). A near optimal isosurface extraction algorithm using the span space. *IEEE Trans. Visualizat. Comput. Graph.*, 2:73–84.

Lopes, A. and Brodlie, K. (2003). Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16–29.

Lorensen, W. E. (1995). Marching through the visible man. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 368, Washington, DC, USA. IEEE Computer Society.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169.

Lozano-Pérez, T. (1981). Automatic planning of manipulator transfer movements. *IEEE Trans. Syst. Man Cybern.*, SMC-11(10):681–698.

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

Lozano-Pérez, T. and Wesley, M. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570.

Lozano-Pérez, T. and Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570.

Maekawa, T. (1999). An overview of offset curves and surfaces. *Computer Aided Design, 31,165-173.*

Manocha, D. and Canny, J. F. (1992). Implicit representation of rational parametric surfaces. *J. Symb. Comput.*, 13(5):485–510.

Matheron, G. (1975). *Random Sets and Integral Geometry.* John Wiley & Sons, New York.

Mehlhorn, K. and Näher, S. (1995). LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102.

Mhlthaler, H. and Pottmann, H. (2003). Computing the minkowski sum of ruled surfaces. In *Graphical Models*, volume 65, pages 369–384.

Milenkovic, V. J. (1998). Rotational polygon overlap minimization and compaction. *Computational Geometry: Theory and Applications*, 10:305–318.

Minkowski, H. (1903). Volumen und oberflche. *Math. Ann.*, 57:447–495.

Mitchell, D. P. (1991). Three applications of interval analysis in computer graphics. In *SIGGRAPH '91 Frontiers in Rendering course notes.*

M.J.Durst (1988). Letters: Additional reference to marching cubes. *ACM Computer Graphics*, 22(4):72–73.

Monks, C. R. F., Crossno, P. J., Davidson, G., Pavlakos, C., Kupfer, A., Silva, C., and Wylie, B. (1996). Three dimensional visualization of proteins in cellular interactions. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 363–ff., Los Alamitos, CA, USA. IEEE Computer Society Press.

Montani, C., Scateni, R., and Scopigno, R. (1994). Discretized marching cubes. *Proc. of IEEE Visualization*, pages 353–355.

Moore, R. E. (1966). *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ.

Munkres, J. (1975). *Topology: A First Course.* Prentice-Hall.

Museth, K., Breen, D., Whitaker, R., and Barr, A. (2002). Level set surface editing operations. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 21(3):330–338.

Natarajan, B. K. (1994). On generating topologically consistent isosurfaces from uniform samples. *Vis. Comput.*, 11(1):52–62.

Nielson, G. M. (2004). Dual marching cubes. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 489–496, Washington, DC, USA. IEEE Computer Society.

Nielson, G. M. and Hamann, B. (1991). The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91.

Nilsson, N. (1969). A mobile automaton: An application of artificial intelligence techniques. In *Proc. IJCAI*, pages 509–520.

Ning, P. and Bloomenthal, J. (1993). An evaluation of implicit surface tilers. *IEEE Comput. Graph. Appl.*, 13(6):33–41.

Nooruddin, F. S. and Turk, G. (2003). Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. on Visualization and Computer Graphics*, 9(2):191–205.

Ó'Dúnlaing, C., Sharir, M., and Yap, C. K. (1983). Retraction: A new approach to motion-planning. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 207–220.

Ó'Dúnlaing, C., Sharir, M., and Yap, C. K. (1987). Generalized Voronoi diagrams for a ladder: II. Efficient construction of the diagram. *Algorithmica*, 2:27–59.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. (2003). Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470.

Ohtake, Y., Belyaev, A. G., and Pasko, A. (2001). Dynamic meshes for accurate polygonanization of implicit surfaces with sharp features. *Prof. of Shape Modeling International*, pages 135–158.

Okino, N., Kakazu, Y., and Kubo, H. (1973). *TIPS-1: Technical Information Processing System for Computer Aided Design and Manufacturing.* Computer Languages for Numerical Control, J. Hatvany, ed., North Holland, Amsterdam.

O'Rourke, J. (1987). *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY.

O'Rourke, J. and Supowit, K. (1983). Some np-hard polygon decomposition problems. *IEEE Transactions on Information Theory, vol. IT-29, pp.181-190*.

Ouchi, K. and Keyser, J. (2004). Handling degeneracies in exact boundary evaluation. In *Proceedings of 9th ACM Symposium on Solid Modeling and Applications*, pages 321–326.

Overmars, M. H. and Svestka, P. (1995). A probabilistic learning approach to motion planning. In *WAFR: Proceedings of the workshop on Algorithmic foundations of robotics*, pages 19–37, Natick, MA, USA. A. K. Peters, Ltd.

Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. (1995). Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446.

Patrikalakis, N. (1993). Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–95.

Payne, B. A. and Toga, A. W. (1990a). Medical imaging: Surface mapping brain function on 3d models. *IEEE Comput. Graph. Appl.*, 10(5):33–41.

Payne, B. A. and Toga, A. W. (1990b). Surface mapping brain function on 3D models. *IEEE Computer Graphics and Applications*, 10(5):33–41.

Perry, R. and Frisken, S. (2001). Kizamu: A system for sculpting digital characters. In *Proc. of ACM SIGGRAPH*, pages 47–56.

Peters, J. (2003). Efficient one-sided linearization of spline geometry. *10th IMA Conference on Mathematics of Surfaces*, pages 297–319.

Pham, B. (1992). Offset curves and surfaces: a brief survey. *Computer-Aided Design*, 24(4):223–229.

Plantinga, S. and Vegter, G. (2004). Isotopic approximation of implicit curves and surfaces. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 245–254, New York, NY, USA. ACM Press.

Pottmann, H. (1995). Rational curves and surfaces with rational offsets. *Computer Aided Geometric Design*, 12(2):175–192.

Pratt, M. (1986). Surface/surface intersection problems. In Gregory, J., editor, *The Mathematics of Surfaces II*, pages 117–142, Oxford. Claredon Press.

QSOPT (2005). Qsopt linear programming solver, url:http://www.isye.gatech.edu/ wcook/qsopt/index.html.

Raab, S. (1999). Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 163–172, New York, NY, USA. ACM Press.

Redon, S. and Lin, M. C. (2005). Practical local planning in the contact space. *Proceedings of IEEE International Conference on Robotics and Automation*.

Reif, J. (1979a). Complexity of the mover's problem and generalizations. *Proc. 20th Symp. on the Foundations of Computer Science*, pages 421–427.

Reif, J. H. (1979b). Complexity of the mover's problem and generalizations. pages 421–427.

Requicha, A. and Voelcker, H. (1985). Boolean operations in solid modeling: boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1).

Ricci, A. (1973). A constructive geometry for computer graphics. *Computer Journal*, 16(2):157–160.

Rossignac, J. and Requicha, A. (1986). Offsetting operations in solid modeling. *Comput. Aided Geom. Design*, 3:129–148.

Rossl, C., Kobbelt, L., and Seidel, H.-P. (2000). Extraction of feature lines on triangulated surfaces using morphological operators. *Proceedings of the 2000 AAAI Symposium*.

Sacks, E. (1999). Practical sliced configuration space for curved planar pairs. *International Journal of Robotics Research*, 18(1).

Sacks, E. (2001). Deterministic path planning for planar assemblies. *Proc. of IEEE Int. Conf. on Robotics and Automation*.

Samet, H. (1989). *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods*. Addison Wesley.

Schaefer, S. (2002). Dual contouring, url:http://www.subdivision.org/comp460/beasts/download/inde

Schaefer, S. and Warren, J. (2004). Dual marching cubes: Primal contouring of dual grids. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*, pages 70–76, Washington, DC, USA. IEEE Computer Society.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons.

Schroeder, W., Martin, K., and Lorensen, B. (1997). *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice-Hall Inc, New Jersey, NJ.

Schroeder, W. J., Lorensen, W. E., and Linthicum, S. (1994). Implicit modeling of swept surfaces and volumes. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 40–45, Los Alamitos, CA, USA. IEEE Computer Society Press.

Schwartz, J. T. (1981). Finding the minimum distance between two convex polygons. *Inf. Process. Lett.*, 13(4/5):168–170.

Schwartz, J. T. and Sharir, M. (1983a). On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351.

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

Sederberg, T. and Meyers, R. (1988). Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5:161–171.

Segal, M. (1990). Using tolerances to guarantee valid polyhedral modeling results. In *Proceedings of ACM Siggraph*, pages 105–114.

Seidel, R. (1994). The nature and meaning of perturbations in geometric computing. Manuscript.

Seong, J. K., Elber, G., Johnston, J., and Kim, M. S. (2004). The convex hull of freeform surfaces. *Computing*.

Seong, J.-K., Elber, G., Johnstone, J., and Kim, M.-S. (2003). The convex hull and kernel of freeform surfaces. In *UAB Technical Report (UABCIS-TR-2004-120104-02)*.

Seong, J.-K., Kim, M.-S., and Sugihara, K. (2002). The minkowski sum of two simple surfaces generated by slope-monotone closed curves. *Geometric Modeling and Processing: Theory and Applications*.

Serra, J. (1982). *Image Analysis and Mathematical Morphology*. Academic Press, London, UK.

Sharir, M. (1997). Algorithmic motion planning. In Goodman, J. E. and O'Rourke, J., editors, *Handbook of Discrete and Computational Geometry*, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL.

Shekhar, R., Fayyad, E., Yagel, R., and Cornhill, F. (1996). Octree-based decimation of marching cubes surfaces. *Proc. of IEEE Visualization*, pages 335–342.

Shewchuk, J. R. (1998). Tetrahedral mesh generation by Delaunay refinement. In *ACM Symposium on Computational Geometry*, pages 86–95.

Simeon, T., Laumond, J. P., and Nissoux, C. (2000). Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6).

Sinha, P., Klassen, E., and Wang, K. (1985). Exploiting topological and geometric properties for selective subdivision. In *ACM Symposium on Computationl Geometry*, pages 39–45.

Snyder, J. M. (1992). Interval analysis for computer graphics. In Catmull, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 121–130.

Stander, B. T. and Hart, J. C. (1997). Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proc. of ACM SIGGRAPH*, pages 279–286.

Sugihara, K. and Iri, M. (1989). A solid modelling system free from topological inconsistency. *J. Inform. Proc.*, 12(4):380–393.

Sutton, P., Hansen, C., Shen, H., and Schikore, D. (2000). A case study of isosurface extraction algorithm performance.

Thomas, U., Barrenscheen, M., and Wahl, F. (2003). Efficient assembly sequence planning using stereographical projections of c-space obstacles. *Proc. of the 5th IEEE International Symposium on Assembly and Task Planning*.

van der Stappen, A. F. and Overmars, M. H. (1994). Motion planning amidst fat obstacles. In *ACM Symposium on Computational Geometry*, pages 31–40.

Varadhan, G., Krishnan, S., Kim, Y., Diggavi, S., and Manocha, D. (2003a). Efficient max-norm computation and reliable voxelization. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Geometry Processing*, pages 116–126.

Varadhan, G., Krishnan, S., Kim, Y., and Manocha, D. (2003b). Feature-sensitive subdivision and isosurface reconstruction. *Proc. of IEEE Visualization*.

Varadhan, G., Krishnan, S., Sriram, T. V. N., and Manocha, D. (2004). Topology preserving surface extraction using adaptive subdivision. In *Eurographics Symposium on Geometry Processing*.

Velho, L. (1990). Adaptive polygonization of implicit surfaces using simplicial decomposition and boundary constraints. In Vandoni, C. E. and Duce, D. A., editors, *Eurographics '90*, pages 125–136. North-Holland.

Vleugels, J. and Overmars, M. (1997). Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222.

Voelcker, H. B. (1974). An introduction to padl: Characteristics, status, and rationale. Technical Report Research Memo. #22, University of Rochester. Production Automation Project.

Wang, S. and Kaufman, A. (1994). Volume-sampled 3d modeling. *IEEE Computer Graphics and Applications*, 14(5):26–32.

Weigle, C. and Banks, D. C. (1998). Extracting iso-valued features in 4-dimensional scalar fields. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 103–110, New York, NY, USA. ACM Press.

Westermann, R., Kobbelt, L., and Ertl, T. (1999). Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 2:100–111.

Wilhelms, J. and Gelder, A. V. (1990a). Octrees for faster isosurface generation extended abstract. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 57–62.

Wilhelms, J. and Gelder, A. V. (1990b). Topological considerations in isosurface generation extended abstract. *Computer Graphics*, 24(5):79–86.

Williams, J. and Rossignac, J. (2004). Mason: Morphological simplification. *GVU Tech. Report GIT-GVU-04-05*.

Wilmarth, S. A., Amato, N. M., and Stiller, P. F. (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, pages 1024–1031.

Wood, Z., Hoppe, H., Desbrun, M., and Schroder, P. (2002). Iso-surface topology simplification. Technical report, Microsoft Research, MSR-TR-2002-28.

Wright, T. and Humbrecht, J. (1979). Isosurf: an algorithm for plotting iso-valued surfaces of a function of three variables. In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 182–189, New York, NY, USA. ACM Press.

Wu, Y., Shah, J. J., and Davidson, J. K. (2003). Improvements to algorithms for computing the minkowski sum of 3-polytopes. *Computer-Aided Design*, 35(13):1181–1192.

Wyvill, B., McPheeters, C., and Wyvill, G. (1986). Animating soft objects. *The Visual Computer*, 2(4):235–242.

Wyvill, B. and van Overveld, K. (1996). Polygonization of Implicit Surfaces with Constructive Solid Geometry. *Journal of Shape Modelling*, 2(4):257–274.

Xavier, P. G. and LaFarge, R. A. (1997). A configuration space toolkit for automated spatial reasoning: Technical results and ldrd project final report. Technical Report SAND97-0366, Sandia Natioanl Laboratories.

Yap, C. K. (1990). Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.*, 10:349–370.

Yap, C. K. and Dubé, T. (1995). The exact computation paradigm. In Du, D.-Z. and Hwang, F. K., editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition.

Yu, J. (1992). *Exact arithmetic solid modeling*. PhD thesis, Purdue University.

Zelinka, S. and Garland, M. (2002). Permission grids: Practical, error-bounded simplification. *ACM Trans. on Graphics*.

Zhang, N., Hong, W., and Kaufman, A. (2004). Dual contouring with topology-preserving simplification using enhanced cell representation. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 505–512, Washington, DC, USA. IEEE Computer Society.

Zhu, D. and Latombe, J. (1990). Constraint reformulation in a hierarchical path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1918–1923.

Zhu, D. and Latombe, J. (1991). New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. on Robotics and Automation*, 7(1):9–20.

Zundel, A. and Sederberg, T. (1993). Using pyramidal surfaces to detect and isolate surface/surface intersections. In *SIAM Conference on Geometric Design*, Tempe, AZ.