National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La quaiité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Computational Geometry in Two and a Half Dimensions

Binhai Zhu

School of Computer Science
McGill University
Montreal, Canada

April 1994

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

ISBN 0-612-00150-4

Canada

Abstract

In this thesis, we study computational geometry in two and a half dimensions. These so-called polyhedral terrains have many applications in practice: computer graphics, navigation and motion planning, CAD/CAM, military surveillance, forest fire monitoring, etc.

We investigate a series of fundamental problems regarding polyhedral terrains and present efficient algorithms to solve them. We propose an $O(n)$ time algorithm to decide whether or not a geometric object is a terrain and an $O(n \log n)$ time algorithm to compute the shortest watchtower of a polyhedral terrain. We study the terrain guarding problem, obtain tight bounds for vertex and edge guards and $O(n)$ algorithms to place these guards. We study the tetrahedralization of certain simple and non-simple polyhedra (which include some special classes of solid terrains) and present efficient algorithms to tetrahedralize them. We also investigate the problem of computing the α-hull of a terrain. Finally, we present efficient algorithms for the intersection detection and computation of Manhattan terrains.

Résumé

Dans cette thèse, nous étudions la géométrie algorithmique en 2.5 dimensions. Ces terrains polyhédraux ont plusieurs applications en pratique: le graphisme assisté par ordinateur, ia navigation et la planification de mouvement, CAD/CAM, la surveillance militaire, la monitorisation des feux de forêt, etc. Nous investigons une série de problèmes fondamentaux concernant les terrains polyhédraux et nous présentons des algorithmes efficaces pour les résoudre. Nous proposons un algorithme de temps O(n) pour décider si un objet géométrique est un terrain et un algorithme de temps O(n log n) pour calculer la tour de guêt la plus courte pour un terrain polyhédral. Nous étudions le problème de garder un terrain, obtenons des bornes justes pour les gardes de sommets et d'arêtes et des algorithmes de temps O(n) pour placer ces gardes. Nous étudions la tétrahédralization de certains polyhèdres simples et non-simples (qui incluent certaines classes spéciales de terrains solides) et nous présentons des algorithmes efficaces pour les tétrahédralizer. Nous investigons également le problème de calculer la coquille-α d'un terrain. Finalement, nous présentons des algorithmes efficaces pour détecter et calculer les intersections de terrains de type Manhattan.

Statement of Originality

All the results, except for basic definitions and the review of certain previous results (which will be indicated in the text), should be considered as original contributions to knowledge. Section 2.1 is a joint work with Zhenyu Li; Section 2.2 is a joint work with Boudewjin Asberg, Gregoria Blanco, Prosenjit Bose, Jesus Garcia-Lopez, Mark Overmars, Godfried Toussaint and Gordon Wilfong; Chapter 4 is joint work with Godfried Toussaint, Clark Verbrugge and Caoan Wang; Section 6.2 is a joint work with Prosenjit Bose, Thomas Shermer and Godfried Toussaint. All the other results are independently obtained by the author.

Acknowledgments

First of all, I wish to express my gratitude to my supervisor, Godfried Toussaint, for his advice, encouragement, criticism, support, and most of all, for introducing me to this topic. I also wish to thank all the other professors in the computational geometry group: David Avis, Luc Devroye, Hossam ElGindy and Sue Whitesides, who have taught me a lot during the last three years.

I would also like to thank all my officemates for numerous conversations while the thesis was in progress: Jit Bose, Xiaowen Chang, David Eu, Eric Guevremont, and Elsa Omana-Pulido. I have also benefited from talking with Marc van Kreveld, which triggered the work of Chapter 7. Thank you, Marc! Special thanks are due to Eric Guevremont for helping me to translate the abstract into French and to Paul Kruszewski for his careful reading and comments of an early version of this thesis.

Many other friends outside of McGill should receive some special thanks: Zhenyu Li, Jim Ruppert, Tom Shermer, T.S. Tan, Takeshi Tokuyama and Caoan Wang. I would also like to thank my supervisor at York University, Andy Mirzaian, who introduced me to the field of computational geometry and taught me a lot about how to do research seriously and elegantly.

I thank all my other friends for making my three-year stay in Montreal an exciting, unforgetable one. Finally I would like to thank my parents and my uncle for their constant love and encouragement.

Contents

List of Figures

Chapter 1

Introduction

After more than two decades of development, computational geometry has reached a new, sophisticated level. Whereas most of the basic problems in *two dimensions* (2D) have already found satisfactory solutions, many of the problems in *three dimensions* (3D) remain to be solved efficiently. Progress on solving 3D problems is relatively slow, mainly because these problems are typically much harder both computationally and combinatorially than their 2D counterparts. For example, although the planar point location problem has optimal solutions with $O(\log n)$ query time after $O(n)$ time and space preprocessing [Kir83, EGS86, Col86, ST86], the best result known for the 3D counterpart has only $O(\log^2 n)$ query time after $O(n \log^2 n)$ time and space preprocessing [PT92].

In this thesis, we will study computational geometry in *two and a half dimensions* (2.5D), i.e., geometric problems regarding the so-called *polyhedral terrains*. First, since a polyhedral terrain is a special geometric object in 3D, efficient solutions on polyhedral terrains might shed some light on the difficult problems in 3D and the computational difficulty (intractability) of geometric problems regarding polyhedral terrains implies the computational difficulty of the corresponding problems in 3D. Second, polyhedral terrains themselves have many applications in practice: computer graphics, navigation and motion planning, military surveillance, forest fire monitoring, locations of radio transmission stations, etc. They are the main objects studied in the fields of *geographical information systems* (GIS) [Bur86] and spatial databases

[GS91, AO93]. Unlike GIS and spatial database researchers, who are mainly concerned with the practical performance of their algorithms, as geometers we study polyhedral terrains in a more formal manner by giving efficient algorithms and/or proving the computational difficulty (intractability) for the terrain-related problems. Of course, the ultimate objective is to give better practical algorithms rather than to obtain only theoretical results. We show that most of the algorithms proposed in this thesis are practical enough to be implemented efficiently.

We begin by giving the definitions for the geometric objects to be studied in this thesis.

A *polyhedral terrain* T (*terrain* for short) with n vertices is a connected 3D polyhedral surface such that for each point $v = (x, y, z)$ on the surface, $z = \mathcal{F}(x, y)$ for some linear function $\mathcal{F}$ [1]. In other words, any vertical line intersects a terrain at most once and the orthogonal projection of a terrain on the XY-plane is a (bounded) planar subdivision. In general, we say a polyhedral surface S is a terrain along a direction $\vec{\mathbf{d}}$ if the intersection of S and any line parallel to $\vec{\mathbf{d}}$ in 3D is either empty or a point. A plane H with norm $\vec{\mathbf{d}}$ is called a projection plane of S. A line segment in 3D is called *rectilinear* if it is perpendicular to either the XY-, XZ- or the YZ-plane. A polyhedral object A is called *rectilinear* if and only if every edge of A is rectilinear. A *Manhattan terrain* M with n vertices is a connected 3D rectilinear polyhedral surface such that the intersection of any vertical line with M is either empty, a point, or a vertical line segment. Since a polyhedral terrain (or a Manhattan terrain) is essentially a planar graph, it can be represented using the *Double Connected Edge List* (DCEL) [PS85] data structure. Throughout this thesis, we assume that once a polyhedral terrain (or a Manhattan terrain) is given, the corresponding DCEL representation is also given.

A *solid terrain* T [2] is a simple polyhedron such that there exists a face f of T and the intersection of T with any line perpendicular to f is either empty, a point, or a line segment with one endpoint lying on f. A *solid Manhattan terrain* M is a simple rectilinear polyhedron such that there exists a face f of M and the intersection of M with any line perpendicular to f is either empty, or a line segment with one endpoint

[1] Hence why many people tend to call a polyhedral terrain a 2.5D geometric object.

[2] We will use T to denote either a terrain or a solid terrain and we will use M to denote either a Manhattan terrain or a solid Manhattan terrain in this thesis.

lying on f. We call f a *valid base* of T (and M).

Since this thesis mostly deals with the complexity of solving geometric problems it is essential to define the *models of computation* and to specify the primitive operations allowed to be executed. We adopt a *random access machine* (RAM) similar to that described in [AHU74], but each storage location can hold a single real number. The primitive operations include: (1) the arithmetic operations $(+, -, \times, /)$; (2) comparisons between two real numbers $(<, \leq, =, \neq, \geq, >)$; (3) indirect addressing of memory; and (4) k-th toot, trigonometric functions, EXP and LOG. In [PS85] this model is referred to as the *real RAM*.

Throughout this thesis, we assume the reader is familiar with the following basic notations: O, Ω, o and Θ. We also assume the reader is familiar with some basic concepts in computational geometry (for example, convex hull, Voronoi Diagram). For more information regarding these definitions, the readers may refer to [PS85] (particularly Chapter 1).

In Chapter 2 we consider two elementary problems regarding polyhedral terrains. First, all previous polyhedral terrain research assumes that a polyhedral terrain is already given. Thus we consider the elementary problem: given a polyhedral surface in 3D, how fast can one test whether or not it is a polyhedral terrain? We show that this can be answered in linear time via linear programming. Second, by a non-trivial extension of the above result we present a linear time algorithm to test whether or not a given simple polyhedron is a solid terrain. It turns out that a solid terrain is a geometric object which can be manufactured by *stereolithography*. Earlier versions of the above results are partially presented in [LZ93] and [ABB+93] respectively.

In Chapter 3 we investigate the problem of computing the α-hull, a generalization of the convex hull of a solid terrain, which is closely related to the problem of manufacturing a solid terrain by *NC-machining*. It turns out that by computing the *upper envelope* of a set of surface patches in 3D the α-hull of a solid terrain can be computed deterministically in $O(n^3)$ time, and it can be computed using a randomized algorithm with expected running time of $O(n^{2+\epsilon})$ (for any $\epsilon > 0$). Since in practice we are not able to produce the α-hull of a terrain with a NC-machine we obtain an algorithm to compute an approximate α-hull. The complexity of our algorithm is

inversely related to the error, defined as the *Hausdorff* distance between the exact and approximate α-hulls.

In Chapter 4 we consider the problem of tetrahedralizing certain special classes of solid terrains. The general problem of deciding whether or not a simple polyhedron in 3D can be tetrahedralized is known to be NP-complete [RS92]. Nevertheless, some special classes of simple and non-simple polyhedra can be tetrahedralized efficiently [GP88, Ber93]. We show that an arbitrary solid terrain does not always admit a tetrahedralization. We extend the classes of tetrahedralizable polyhedra, which include several classes of solid terrains. In particular we show that the following classes of simple and non-simple polyhedra can always be tetrahedralized: simple slab (with or without holes), subdivision slab, a box with fixed depth rectilinear holes and a box with linearly ordered rectangular holes. The first two classes can be considered as generalized solid terrains, while the latter two classes belong to the class of solid Manhattan terrains [TVWZ93]. We also discuss the problem of tetrahedralizing the class of polyhedra which are the union of k convex polyhedra. We show that for k=2 and 3 the resulting polyhedra can be efficiently tetrahedralized; however, for $k \geq 4$, the polyhedra do not always admit a tetrahedralization.

In Chapter 5 we consider the shortest watchtower and related problems for a polyhedral terrain. The shortest watchtower is defined to be the shortest vertical line segment erected on the terrain such that the top of the tower can see the whole terrain. Sharir gave an $O(n \log^2 n)$ time algorithm for solving this problem and posed as an open problem the computation of such a watchtower in $O(n \log n)$ time [Sha88]. By extending the *hierarchical representation* of a convex polyhedron of Dobkin and Kirkpatrick [DK85], we solve the above open problem in $O(n \log n)$ time. Such an extension on the hierarchical representation also extends the previous result regarding the intersection detection between a line (or line segment) e and a preprocessed convex polyhedron P. If line e does not intersect P then we can report in logarithmic time the shortest distance between them along a given direction. In other words, we can report the shortest distance along the direction at which e and P will collide if either of them is moved along the fixed direction. Again with this extension, we show that the shortest vertical distance between two convex polyhedra can be computed in

linear time. Furthermore, such a data structure also solves the *intersection detection problem* between two convex polyhedra in $O(\log^2 n)$ time after $O(n)$ time and space preprocessing [Zhu92, Zhu93]. This achieves the same complexity as the best known algorithms [Col86, DK90].

In Chapter 6 we consider the problem of guarding polyhedral terrains, i.e., placing a set of *guards* on a terrain such that the whole terrain is collectively covered by these guards. Cole and Sharir showed that computing the minimum number of vertex guards to cover the surface of a terrain is NP-complete [CS89]. By modifying their proof we first show that the problem of finding the minimum number of edge guards to cover a terrain is also NP-complete. We show that $\lfloor \frac{n}{2} \rfloor$ vertex guards are always sufficient and sometimes necessary to guard an n-vertex terrain. We also present a linear time algorithm for placing $\lfloor \frac{3n}{5} \rfloor$ vertex guards to cover a terrain. With respect to edge guards, Everett and Rivera-Campo showed that $\lfloor \frac{n}{3} \rfloor$ edge guards are always sufficient [ERC92]. We show that $\lfloor \frac{(4n-4)}{13} \rfloor$ edge guards are sometimes necessary to guard the surface of an n-vertex terrain. Finally, we present a linear time algorithm for placing $\lfloor \frac{2n}{5} \rfloor$ edge guards to cover a polyhedral terrain [BSTZ92, BSTZ93].

In Chapter 7 we study the problems of intersection detection and computation of Manhattan terrains. These problems arise very offten in practice since the surfaces of many modern buildings can be thought of as Manhattan terrains. We show that after $O(n \log n)$ time and space preprocessing, the intersection of a rectilinear line segment (ray or line) with a Manhattan terrain can be detected in $O(\log n)$ time. Furthermore, if no intersection occurs, we can then report the shortest vertical distance between the rectilinear line segment and the Manhattan terrain within the same time bound. We achieve this by building a two-layer *hybrid segment tree* with the second layer as *symmetric order heaps* [HT84] and applying the *fractional cascading* technique [CG86]. With these results, we show that given two Manhattan terrains with a total of $O(n)$ vertices, we can either compute the shortest vertical distance between them or report their intersection in $O(n \log n)$ time. The generalized version of this problem, computing the shortest vertical distance between two non-intersecting polyhedral terrains, is more difficult and has been studied recently. A randomized algorithm with time complexity $O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$) is obtained by Chazelle et al. [CEGS89].

Finally, we show that given two Manhattan terrains with a total of $O(n)$ vertices, we can compute their intersection (upper envelope) in $O(n \log n + K)$ time, where K is the combinatorial complexity of the envelope. The generalized problem of computing the intersection of two polyhedral terrains is solved with a randomized algorithm with time $O(n^{4/3+\epsilon} + K^{1/3}n^{1+\epsilon} + K \log^2 n)$ (for any $\epsilon > 0$), where K is the size of output [Pel93].

In Chapter 8 we conclude the thesis by summarizing the main results and posing a related set of open problems.

Chapter 2

Testing if a polyhedral object is a terrain

Since we are studying computational geometry in 2.5D, it is not unreasonable to assume that the input for these problems consists of polyhedral terrains. However, in many applications we may not know in advance whether or not a given polyhedral surface is a terrain. Therefore, a question which arises naturally is how fast one can decide if a given polyhedral surface is a polyhedral terrain. Similarly, given a simple polyhedron how fast can one decide if it is a solid terrain? In this chapter we show that we can answer both of these two questions in linear time. In Section 2.1 we present a linear time algorithm by linear programming to test whether or not a polyhedral surface is a terrain. By a non-trivial extension of this algorithm in Section 2.1, in Section 2.2 we present a linear time algorithm to test if a simple polyhedron is a solid terrain.

2.1 Testing if a polyhedral surface is a polyhedral terrain

In recent years there has been much research conducted involving polyhedral terrains in the areas of geographical information systems (GIS), spatial databases, computer

graphics and computational geometry [dFP+86, Sha88, RS88, CS89, CEGS89, Lee91], [PV92, AS93]. Common to all of this research is the assumption that a polyhedral terrain is given as input. We consider a different problem that asks given a connected 3D polyhedral surface S if there is a projection plane N with norm $\vec{n} =< x_1, x_2, x_3 >$ such that the orthogonal projection of S on N is a planar subdivision. In other words, does there exist a direction $\vec{n} =< x_1, x_2, x_3 >$ such that S is a terrain along $\vec{n}$? In 2D a related problem of testing whether or not a simple polygon is monotone is solved by Preparata and Supowit [PS81].

We show how this problem can be formulated as a linear programming problem. Let the outer norm of the plane containing the face f_i of S be $\vec{n}_i =< a_i, b_i, c_i >$. We start with the following result.

Lemma 2.1: A connected polyhedral surface S is a terrain if and only if there exists $\vec{n} =< x_1, x_2, x_3 >$ such that all the dot products of $\vec{n}_i \bullet \vec{n}$ $(= a_i x_1 + b_i x_2 + c_i x_3)$ are greater than zero.

Proof: If S is a terrain then there exists a direction $\vec{n} =< x_1, x_2, x_3 >$ such that any directed line l along this direction intersects S at most once. This immediately implies that the angles between l and the norms of the faces of S are less than $\pi/2$, i.e., all $\vec{n}_i \bullet \vec{n}$ are greater than zero.

If there exists $\vec{n} =< x_1, x_2, x_3 >$ such that for all i, $\vec{n}_i \bullet \vec{n} > 0$ then we show that S must be a terrain along $\vec{n}$. If this is not the case, then there exists a directed line l along $\vec{n}$ such that l intersects S at least twice. Without loss of generality, assume l intersects S consecutively at faces with norms $\vec{n}_j, \vec{n}_k$. Since S is connected, one of the angles between $\vec{n}$ and $\vec{n}_j$, $\vec{n}$ and $\vec{n}_k$ must be greater than $\pi/2$. This immediately implies that either $\vec{n}_j \bullet \vec{n} < 0$ or $\vec{n}_k \bullet \vec{n} < 0$. □

With the above lemma, we can formulate the problem as follows.

Minimize $x_1 + x_2 + x_3$ (or any objective function of x_1, x_2, x_3)
subject to
$a_i x_1 + b_i x_2 + c_i x_3 > 0$
for $i = 1, ..., m$ (m is the number of faces of S and $m \leq 2n - 4$).

By applying Meggido's algorithm [Meg84], which solves a linear programming problem in linear time when the dimension is fixed, we can decide in linear time whether or not a feasible solution exists. A feasible solution gives us a projection plane. This algorithm can actually be extended to determining whether there exists a d-hyperplane ($d \geq 1$) for which a d-dimensional surface is monotone. Again by Meggido's algorithm, this extension still runs in linear time for fixed d. Therefore, we have the following theorem.

Theorem 2.2: Given a connected polyhedral surface S with n vertices in 3D, one can decide in linear time whether it is a terrain. If the answer is YES, one can also give a projection plane for which S is a terrain.

It should be noted that the above algorithm solves in linear time the problem of determining if there is a *hemisphere* which contains a given set of points on a sphere [DR80]. We can apply the above algorithm to test whether or not an n-vertex simple polyhedron is a solid terrain in $O(n^2)$ time. The test can be performed as follows. Each time we fix a face and test if the remaining polyhedral surface is a terrain with respect to this face in linear time. It is easy to see that this can be done in $O(n^2)$ time. Nevertheless, we show in the next section that this can actually be done in $O(n)$ time.

2.2 Testing if a simple polyhedron is a solid terrain

Before presenting a linear time algorithm to test if a simple polyhedron is a solid terrain, we discuss briefly the relationship between a solid terrain and a CAD/CAM system developed and patented by 3D Systems of Sylmar, CA that employs a manufacturing process called *stereolithography* (see Figure 2.1).

Stereolithography employs a vat of liquid photocurable plastic, a computer controlled table T on a stand S that can be moved up and down in the vat and a laser L above the vat that can shine on the surface of the liquid plastic and can move in a horizontal plane. The system works as follows. At the first step the table is just below

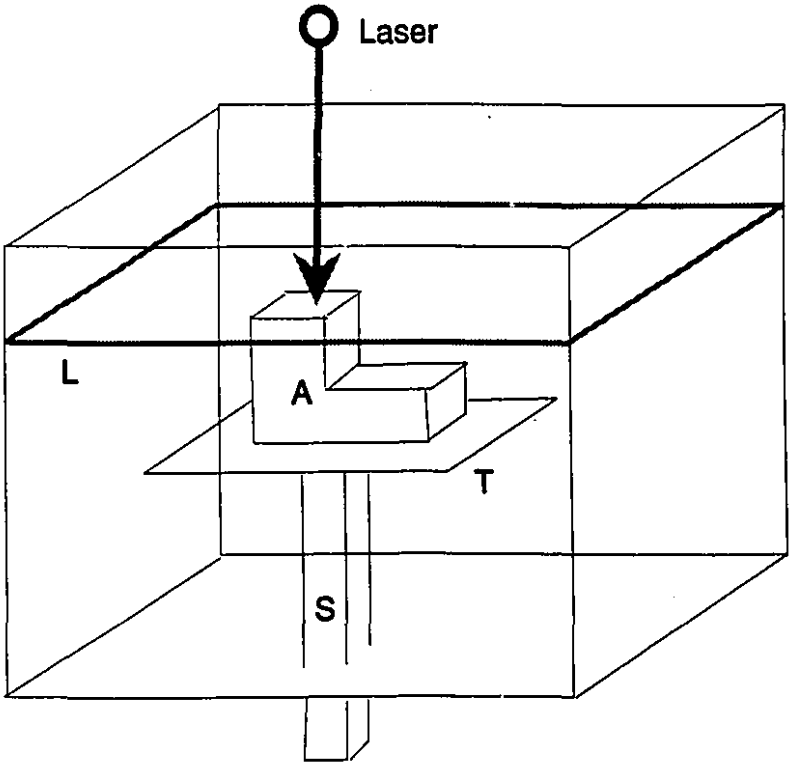


Figure 2.1: Stereolithography.

the surface of the plastic and the laser is controlled to move about so that the light shines on the surface of the plastic and draws the bottom-most cross-section of the object A being built. When the laser light contacts the plastic, the plastic solidifies and so the first cross-section of the object is formed and rests on the table. At the next step the table is lowered a small amount to allow liquid to cover the hardened layer and the laser then draws the next cross-section of the object. The light from the laser penetrates the liquid just deep enough so that this cross-section is welded to the lower cross-section produced at the previous step. This process is repreated until the entire object is formed. The direction given by a normal to the table pointing to the laser is called the *direction of formation* for the object.

There are some objects that can be formed only if the direction of formation is

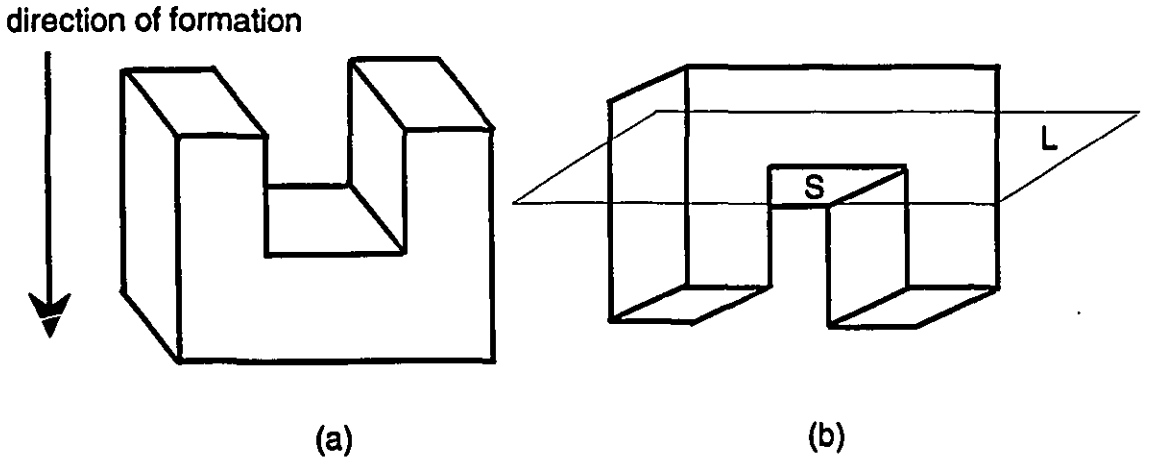


Figure 2.2: Objects can and can not be manufactured by stereolithography.

chosen correctly. For example, in Figure 2.2, the object (a) can be formed in the position shown. However, if the object is formed in the opposite direction as in Figure 2.2 object (b) then stereolithography fails. Consider what occurs when the cross-section is reached where the surface S lies. The surface S is not supported below and so as it is formed it sinks to the level of the table. Naturally, there are some objects that can not be formed using stereolithography regardless of the direction of formation chosen.

From the above description of stereolithography, it is clear that an object (modeled as a polyhedron) can be constructed using stereolithography if and only if it is a solid terrain. Furthermore, the *direction of formation* corresponds to the direction for the object to be a solid terrain. Consequently, a linear time algorithm to decide whether a simple polyhedron is a solid terrain would imply a linear time algorithm to decide whether or not an object can be constructed using stereolithography.

Now we give the necessary detail to decide if a simple polyhedron is a solid terrain. Suppose we are given a simple polyhedron A in 3D and we want to either find a face f of A that is a valid base or determine that A is not a solid terrain.

In this section we use the following notation. Let A be a polyhedron with n

vertices. For each face f of A, let $f(1), f(2), \ldots, f(k_f)$ be the faces of A that share at least one edge with f. Let $\vec{d}(f), \vec{d}(1), \vec{d}(2), \ldots, \vec{d}(k_f)$ be the corresponding unit norms of these faces. Let A_f be the plane containing the face f. Let $\theta_i(f)$ be the angle interior to A between the plane A_f and the plane $P(i)$ containing $f(i)$ about the line of intersection of A_f and $P(i)$. If $\theta_i(f) \leq \pi/2$ for all i, $1 \leq i \leq k_f$, then f is called *acute*. If f is acute and for some i, $\theta_i(f) < \pi/2$, then f is said to be *strictly acute*. We show several properties regarding A that will give rise to a linear time feasibility testing algorithm. Without loss of generality, let $\vec{d}(f) = < 0, 0, -1 >$. We have the following observation.

Observation 2.3: If face f is a valid base for polyhedron A then f is acute. Furthermore, if A is a convex polyhedron then face f is a valid base if and only if f is acute.

Thus if A is known to be convex we can decide in linear time whether or not A is a solid terrain. We now turn our attention to polyhedral objects that are not necessarily convex. We show that if the polyhedral object A has a strictly acute face f, then either f or one of its adjacent faces must be a valid base if A is a solid terrain.

Lemma 2.4: If polyhedron A is a solid terrain and f is strictly acute then $f, f(1), \ldots,$ or $f(k_f)$ is a valid base.

Proof: If f is a valid face then the lemma is proved. Suppose f is not a valid face and the valid face V does not belong to $f(1), \ldots, f(k_f)$. Let $\vec{d}(V)$ be the unit norm of V. Since f is strictly acute, the angle between $-\vec{d}(f), \vec{d}(i) (1 \leq i \leq k_f)$ is less than $\pi/2$ and consequently all $\vec{d}(i) (1 \leq i \leq k_f)$ are above the plane $Z = 0$.

Let us now compute the convex hull of $\vec{d}(i)$ $(1 \leq i \leq k_f)$, which is an infinite cone C starting at the origin. C contains $-\vec{d}(f)$, since f is a simple polygon (see Figure 2.3). If $-\vec{d}(V)$ lies above the plane $Z = 0$ then the angle between $\vec{d}(f)$ and $-\vec{d}(V)$ is greater than $\pi/2$ which implies that V can not be a valid base. If $-\vec{d}(V)$ lies below the plane $Z = 0$ then the angle between $-\vec{d}(f)$ and $-\vec{d}(V)$ is greater than $\pi/2$. Since $-\vec{d}(f)$ is contained in C at least one angle between $\vec{d}(i), -\vec{d}(V)$ is greater than $\pi/2$ and again V can not be a valid base. □

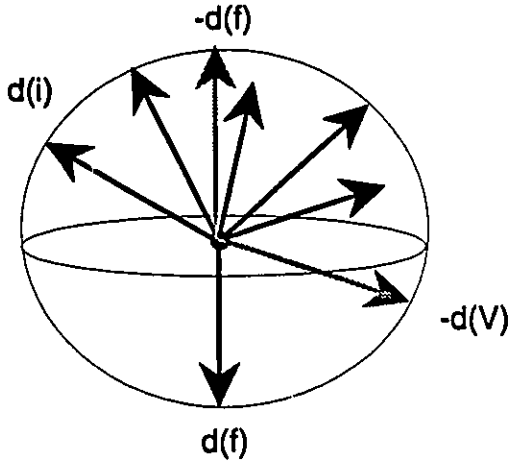


Figure 2.3: Only a strictly acute face and its adjacent faces can be valid bases.

Assume that f is not strictly acute, we define f_{top} as follows. If there is more than one face with norm $-\vec{d}(f)$, then f_{top} is empty (there does not exist a valid base with norm $-\vec{d}(f)$); otherwise, we use f_{top} to denote that face.

Lemma 2.5: If polyhedron A is a solid terrain and f is an acute (but not strictly acute) face then $f, f(1), \ldots, f(k_f)$ or f_{top} is a valid base.
Proof: Follows from Lemma 2.4 and the definition of f_{top}. □

Unfortunately, in 3D an acute face f may have $O(n)$ adjacent faces. Therefore even if we already have an acute face f we still need to test $O(n)$ candidate bases, which will imply an $O(n^2)$ time algorithm. Nevertheless, we have the following theorem.

Theorem 2.6: Polyhedron A is a solid terrain if and only if A contains a acute face f and at least one of f, f_{top} (if it exists) and at most four faces adjacent to f is a valid base of A.
Proof: The "if" part is trivial. We need only show the "only if" part. Following from Lemmas 2.4 and 2.5, if A is a solid terrain and A contains an acute face f then either $f, f(1), \ldots, f(k_f)$ or f_{top} is a valid base. Now we only need to show that among $f(1), \ldots, f(k_f)$ there are at most four faces which can be valid bases of A. Let $e_i = \overline{v_i v_{i+1}}$ be the edge of f which is adjacent with $f(i)$. We show that if $f(i)$ is a

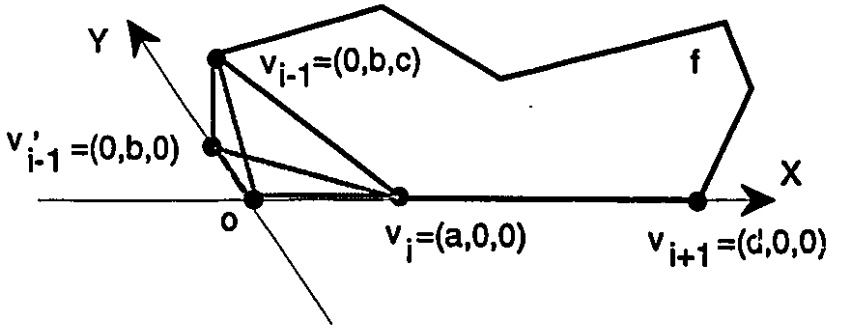


Figure 2.4: A terrain has at most six valid bases.

valid base of A then e_i has the following properties:

(1) e_i is on the convex hull f, and

(2) the two inner angles of v_i and v_{i+1} within f are at most $\pi/2$.

It is clear that the first property has to be correct. If e_i is not on the convex hull of f then the line through e_i intersects the interior of f and therefore the plane through $f(i)$ intersects the interior of f. This contradicts the assumption that $f(i)$ is a valid base. Now we prove the second property. Without loss of generality, let the plane containing $f(i)$ be the plane $Z = 0$ and let $e_i = \overline{v_i v_{i+1}}$ be on the X-axis such that $v_i = (a, 0, 0)$ and $v_{i+1} = (d, 0, 0)$ $(d > a > 0)$. Furthermore, let $v_{i-1} = (0, b, c)(b, c > 0)$ be on the plane $X = 0$ and v'_{i-1} be the vertical projection of v_{i-1} on $Z = 0$. As Figure 2.4 shows, the interior angle of v_i within f is greater than $\pi/2$. It turns out that the plane through the triangle $\Delta v_i v_{i-1} v'_{i-1}$ is $bX + a(Y - b) = 0$ while the plane through f (i.e., through $\Delta v_i v_{i-1} o$) is $-cY + bZ = 0$. The norms of the two planes are $\vec{n}_1 =< b, a, 0 >$ and $\vec{n}_2 =< 0, -c, b >$ respectively. Consequently the angle between the two planes is greater than $\pi/2$ since $\vec{n}_1 \bullet \vec{n}_2 = -ac < 0$. Since f is acute, the angle between f and $f(j)$ ($f(j)$ shares $\overline{v_{i-1} v_i}$ with f) is less than or equal to $\pi/2$. Consequently $f(j)$ lies below f, which implies that $f(i)$ can not be a valid base. We can show similarly that if the inner angle of v_{i+1} is greater than $\pi/2$ then $f(i)$ can not be a valid base.

Now assume we have already computed in linear time the convex hull of f, $CH(f)$.

We show that there are at most four edges of $CH(f)$ satisfying property (2).

Given a convex polygon C with K edges, if an edge e_j $(1 \leq j \leq K)$ has the property that $\theta(e_{j-1}, e_j) \leq \pi/2, \theta(e_j, e_{j+1}) \leq \pi/2$ then e_j is called a valid base of C. Suppose there are L angles less than or equal to $\pi/2$ in C. Since

$$\sum_{i=0}^{K-1} \theta(e_i, e_{i+1}) = (K-2)\pi,$$

we have

$$\frac{L}{2}\pi + (K-L)\pi \geq (K-2)\pi,$$

which implies $L \leq 4$.

Therefore, there are at most four edges of $CH(f)$ which satisfy this property. This implies that there can be at most four faces among $f(i)$ $(1 \leq i \leq k_f)$ which can be a valid base of A. □

Thus in $O(n)$ time, where n is the number of vertices in polyhedron A, the number of possible valid bases can be reduced to at most six. By Theorem 2.2, deciding if a face f of polyhedron A is a valid base can be done in $O(n)$ time, where n is the number of vertices of A. Therefore we can test whether A is a solid terrain and find a valid base (if it is) in $O(n)$ time. Consequently, we have the following theorem.

Theorem 2.7: Given a simple polyhedron with n vertices one can test in $O(n)$ time whether or not it is a solid terrain and if it is a solid terrain one can identify all valid bases with the same time complexity.

We have thus obtained an optimal $O(n)$ time algorithm to decide if a polyhedral surface is a terrain and an optimal $O(n)$ time algorithm to test if a given polyhedron is a solid terrain. A further question would be decomposing an arbitrary polyhedral surface into minimum number of terrains. The related 2D problem of decomposing the boundary of a simple polygon into minimum number of monotone chains has been solved with optimal linear time algorithms [CRS92, RR92, LZ93]. However this 3D problem seems much more difficult. We feel strongly that it is NP-complete, but no progress has been made in this direction. Thus we pose this as an open problem to conclude this chapter:

Open Problem 1: What is the complexity of decomposing a 3D polyhedral surface S into the minimum number of terrains (along different directions) if S is not a terrain?

Chapter 3

The α-hull and related problems of a terrain

In Chapter 2 we presented linear time algorithms to test if a polyhedral surface and a simple polyhedron is a terrain. This in hand, we may start to consider the convex hull problem, whose role in computational geometry is known to be that of sorting in algorithms. Although there are several optimal $\Theta(n \log n)$ ($\Theta(n)$) time algorithms known for computing the convex hull of a set of points (a simple polygon) in 2D [PS85], the only known optimal algorithm for computing the convex hull of a set of points in 3D is the divide-and-conquer algorithm by Preparata and Hong [PH77]. With this result we can compute the convex hull of a polyhedral terrain in $O(n \log n)$ time.

The α-hull is a generalization of convex hull of a set of points in 2D and 3D and was proposed a decade ago [EKS83]. It was originally proposed to extract the shape of a set of points. Recently it was found that α-hulls can be applied to NC-machining [WCC+93]. In this chapter we consider the problem of computing the α-hull of a terrain. It turns out that the α-hull of a terrain can be computed in deterministic $O(n^3)$ time by first computing the *offsets* of the vertices, edges and faces of the terrain, which are a set of $O(n)$ simple surface patches in 3D. Then we obtain the α-hull by applying the trivial cubic algorithm for computing the *upper envelope* of n simple surface patches in 3D. With the recent results of computing the upper envelope of

n surface patches in 3D [HS93, Sha93a, Sha93b], it can be computed in randomized $O(n^{2+\epsilon})$ time (for any $\epsilon > 0$). Since in practice we are not able to manufacture the α-hull of a terrain with a NC-machine we use *discrete approximation* to obtain an $O((N+n)^2)$ time algorithm to compute an approximate α-hull of a solid terrain T, where N is the number of sample points chosen to approximate the surface of T. We also show that N is inversely proportional to the *error*, which is defined as the *Hausdorff* distance, between the exact and the approximate α-hulls of T. Some simple, elegant and practical ideas are implemented in the algorithm. We hope the algorithm and the related ideas will be useful in the NC-machining industry.

3.1 Preliminary

We just mention that the convex hull of a polyhedral terrain can be computed in $O(n \log n)$ time. A natural generalization of the convex hull problem is to compute the α-hull of a (solid) terrain. It turns out that in theory the α-hull of a solid terrain T is the resulting object manufactured by NC-machining [Bez72, PW79, FP79, BFK84, DJSH89], with T being the target.

We first give a brief introduction of the definition and related results regarding α-hulls [EKS83]. Given a real α, a *generalized disc of radius* $1/\alpha$ is defined as follows:

(i) if $\alpha > 0$, it is a (standard) disc of radius $1/\alpha$;

(ii) if $\alpha < 0$, it is the complement of a disc of radius $1/\alpha$; and

(iii) if $\alpha = 0$, it is a half plane.

The α-hull of a set of n points S in the plane is defined to be the intersection of all generalized discs of radius $1/\alpha$ that contain all of the points in S. Figure 3.1 shows the α-hull of a set of points in the plane where $\alpha < 0$. The convex hull of S is precisely the α-hull of S when α is equal to zero. Edelsbrunner et al. [EKS83] developed optimal algorithms to compute the α-hull of a set of points S by first computing the *nearest and furthest point* Voronoi Diagrams of S.

The definition of α-hull and the corresponding algorithms can be generalized to a set of n points in 3D. However, because the Voronoi Diagram of a set of n points in 3D has quadratic size, the time complexity of computing the α-hull is $O(n^2)$. The

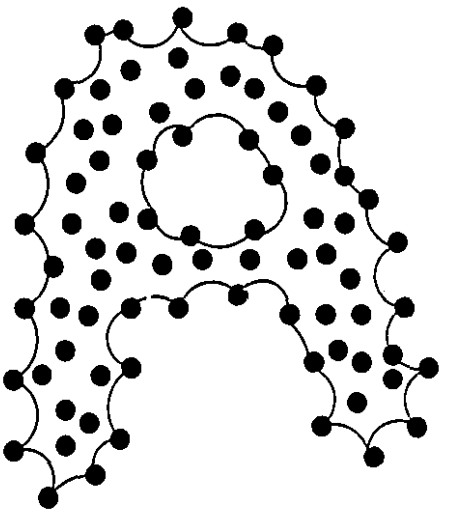


Figure 3.1: The α-hull of a set of points in the plane.

following theorem summarizes the main results of [EKS83, EM94].

Theorem 3.1 [EKS83, EM94]: The α-hull of a set S of n points in the plane can be computed in time $\Theta(n \log n)$ using $O(n)$ space. The α-hull of a set S of n points in 3D can be computed in time $O(n^2)$ using $O(n^2)$ space.

Next we give a brief introduction of NC-machining, i.e., machining a polyhedral object on a numerically controlled (NC) machine. The machining of sculptured surfaces with numerical control (NC) is a common practice in industry. Sculptured surfaces arise in design and manufacture of automobile bodies, ship hulls, aircraft, etc. Often when a computer is used to control the NC machine, polyhedral approximations of these objects are usually used instead. A three-axis NC machine (X,Y,Z Cartesian movements, see Figure 3.2) with a ball-end cutter is the most common machine used in practice. Two types of cutting errors can occur: gouging and excess material. Decreasing the size of the cutter radius often solves these problems, however a smaller cutter will cut more slowly, break more often and wear faster. Determining the optimal tool size as well as simulation, detection and elimination of the errors in NC machining have been studied for quite a long time and many approximation

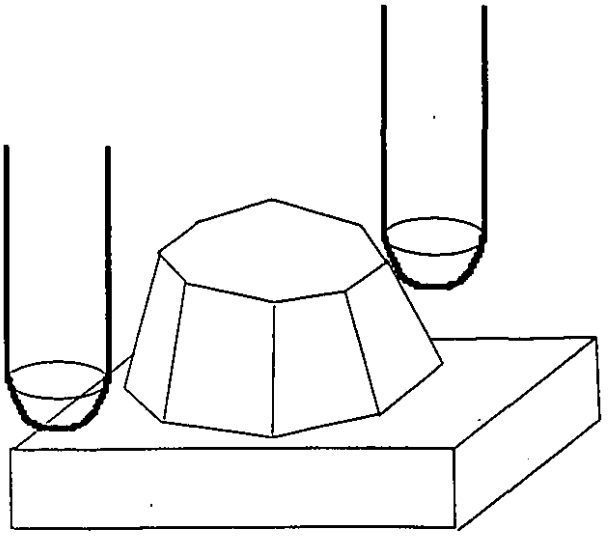


Figure 3.2: An example of three-axis NC-machining.

algorithms, simulation methods and practical systems have been proposed.

We see that in theory the α-hull of a solid terrain T is exactly the resulting object produced by a NC-machine with a cutter of radius $1/\alpha$, with T being the target. We now give the formal definition of the α-hull of a solid terrain. Given a real α, a *generalized ball of radius* $1/\alpha$ is defined as follows:

(i) if $\alpha > 0$, it is a (standard) ball of radius $1/\alpha$;

(ii) if $\alpha < 0$, it is the complement of a ball of radius $1/\alpha$; and

(iii) if $\alpha = 0$, it is a half space in 3D.

The α-hull of a solid terrain T with n vertices is defined to be the intersection of all generalized balls of radius $1/\alpha$ that contain all of the points on the surface of T. The convex hull of T is precisely the α-hull of T when $\alpha = 0$. When $\alpha > 0$ the α-hull of T is of no practical interest to us although it can be computed exactly in $O(n^2)$ time with the result of [EKS83, EM94]. We only consider the case when $\alpha < 0$. It should be noted that when $\alpha < 0$ the α-hull of T is different from the α-hull of the vertices of T.

3.2 Computing the exact and approximate α-hulls of a terrain

In this section we discuss the problem of computing the exact and approximate α-hulls of a terrain. For the corresponding 2D problem, Woo et al. propose a linear time algorithm to compute the α-hull of a monotone chain [WCC+93]. In addition, two trivial $O(n \log n)$ time algorithms are mentioned to compute the α-hull of a monotone chain: the *Constrained Voronoi Diagram* method [LD81, Yap87, Che89], and the *upper envelope* method [Her89]. Correspondingly, we might generalize these methods to 3D to compute the α-hull of T. However, as for the Constrained Voronoi Diagram in 3D, it is only known that if the polyhedral obstacles (i.e., points, line segments and faces) satisfy certain axioms then the Constrained Voronoi Diagram exists and no algorithm is known to compute such a Constrained Voronoi Diagram in 3D [Sti91]. It turns out that the *upper envelope* method can be generalized to 3D to compute the α-hull of a terrain.

The *upper envelope* of a set of n simple surface patches is defined as a polyhedral surface S such that the Z-coordinates of every point of S is the pointwise maximum of the n surface patches. By first computing the *offsets* of the vertices, edges and faces of T, which are all simple surface patches (see Figure 3.3)[1], then computing the upper envelope of these offsets we can obtain the α-hull of T. The upper envelope of n simple surface patches in 3D can be computed in $O(n^3)$ time since there could be $\Omega(n^3)$ intersections between these n patches. There are a series of results for computing the upper envelope of n simple surface patches in 3D [HS93, Sha93a, Sha93b]. It turns out that the upper envelope of n surface patches can be computed in randomized $O(n^{2+\epsilon})$ time (for any $\epsilon > 0$). Consequently the α-hull of a solid terrain can be computed within the same time.

Observation 3.2: Given a solid terrain T with n vertices, the α-hull of T can be computed deterministically in $O(n^3)$ time and with a randomized algorithm in

[1] **In Figure 3.3, we only show an illustration of the offset of an edge and a face. The offset of a vertex can be computed by computing the intersection of the surface of the ball centered at that vertex with all the offsets of its adjacent edges and faces.**

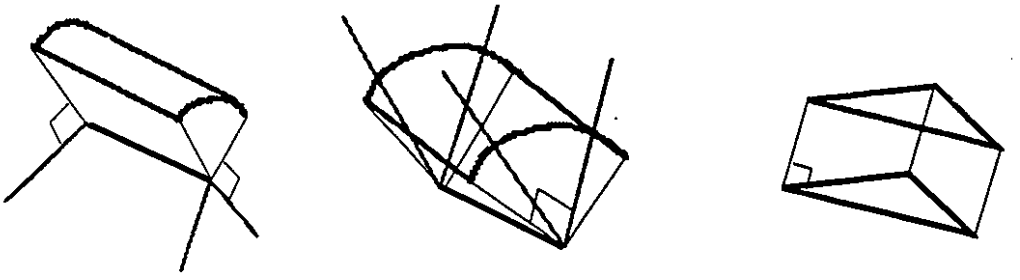


Figure 3.3: The offsets of edges and faces.

$O(n^{2+\epsilon})$ expected time (for any $\epsilon > 0$).

However, in practice we are not able to produce the exact α-hull of a terrain with a NC-machine even if the terrain is a prism! Whenever we translate the ball-end cutter on the top triangle the corresponding trajectory on the top triangle is a line segment (which is of area zero). Therefore, to produce the exact α-hull of a terrain with a NC-machine we might have to move the cutter infinitely. With this consideration, we use *discrete approximation* to obtain an $O((N+n)^2)$ time algorithm to compute an approximate α-hull of a solid terrain T. The basic idea is to replace the exact representation of a solid terrain by carefully chosen sample points in space. This idea is not new in NC-machining and there are several works in NC-machining using such a method [Cha83, Hoo86, WW86, DJSH89]. Intuitively, the more sample points chosen, the more accurate the simulation will be. However, in practice a huge amount of sample points would make the computation infeasible. Therefore an acceptable amount of sample points must be chosen carefully so that the errors of simulation are bounded. We show that an approximate α-hull of a solid terrain T can be computed in $O((N+n)^2)$ time, where N denotes the number of sample points chosen to approximate the surface of T and is closely related to the accuracy between the approximate and the exact α-hulls of T.

First of all, we define a function to characterize the quality of an approximation. Let S_1 be the set of points on the α-hull of T and S_2 be the set of points on an approximate α-hull of T. Let $\delta_H(S_1, S_2)$ (the *Hausdorff* distance) be defined as

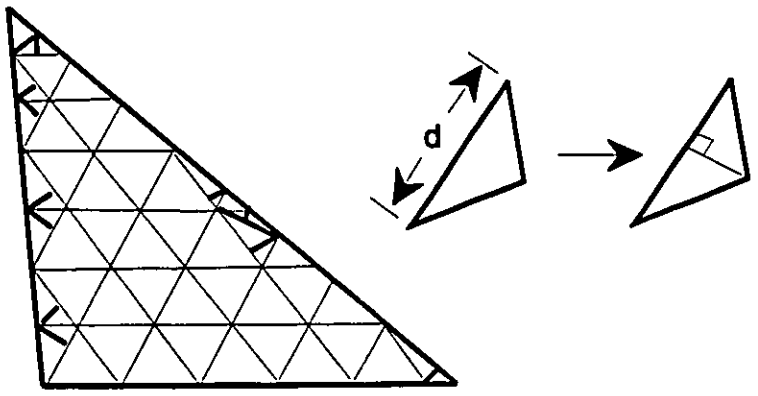


Figure 3.4: Computing the mesh of a triangular face.

$$\delta_H(S_1, S_2) = \max\{\max_{a \in S_1} \min_{b \in S_2} d(a,b), \max_{b \in S_2} \min_{a \in S_1} d(a,b)\},$$

where $d(a,b)$ is the Euclidean distance between a and b. We also say that $\delta_H(S_1, S_2)$ is the *error* between the exact and approximate α-hulls of T.

We now start to present the details of our algorithm. First we obtain a triangulation of each face F of T by choosing a set of sample points on F and then triangulating F with these sample points together the vertices of F. The resulting triangulation is usually called the *mesh generation* of T. The criteria here are (1) each edge in the mesh generation of T is not greater than a given value d, and (2) no obtuse triangles exist in the mesh generation. To make things easy we just generate the mesh by equilateral triangles with edge length d. As shown in Figure 3.4 there can be obtuse triangles near the boundary of F. We simply transform each obtuse triangle into two acute triangles such that the edge length of each acute triangle is no more than d. The complexity of computing such a mesh generation of T is proportional to the number of sample points used, since such a mesh generation is a planar straight line graph.

There are two kinds of errors induced when we use a mesh of triangles to approximate T and compute the corresponding α-hull. The first error arises when a ball protrudes through a supporting plane, and is characterized by the following lemma.

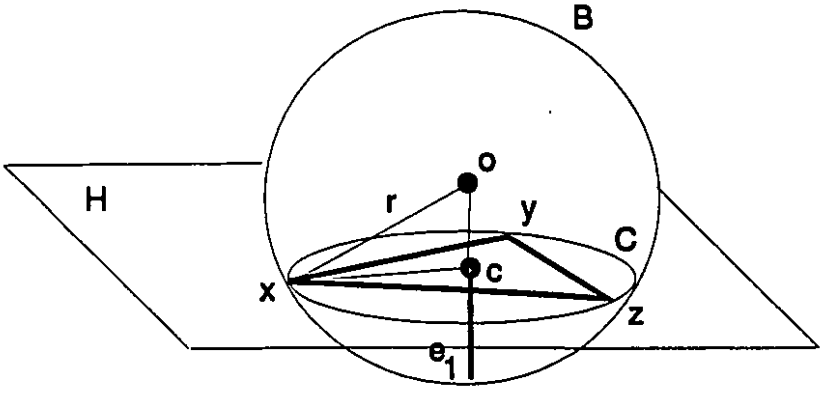


Figure 3.5: Illustration for the proof of Lemma 3.7.

Lemma 3.3: Let B be a ball with radius r and center o, $\triangle xyz$ be a regular triangle (with edge length d) inscribed in B, C be the circumcircle of $\triangle xyz$ with center c, H be the plane through C, and θ be the angle between $\overline{ox}$ and $\overline{oc}$. Then the Hausdorff distance between H and the spherical cap cut off by H is bounded by

$$e_1 = r - r\cos\theta,$$

where $\sin\theta = \frac{d}{\sqrt{3}r}$.

Proof: (Refer to Figure 3.5) Following the Pythagorean theorem, we have

$$e_1 = r - |oc| = r - r\cos\theta,$$

where $\sin\theta = \frac{d}{\sqrt{3}r}$. By Taylor's theorem,

$$e_1 = r - r\cos\theta = r - r(1 - \frac{\theta^2}{2!} + o(\theta^4)) \sim \frac{r\theta^2}{2}, as\theta \to 0.$$

It should be noted that the above bound regarding this type of error is in fact the worst case. The reason is that if the triangle is an acute triangle with edge length at most d then the radius of the largest circumcircle of the triangle is at most $\frac{d}{\sqrt{3}}$. In other words, if $\triangle xyz$ is not a regular triangle then $\sin\theta < \frac{d}{\sqrt{3}r}$. □

The underlying idea behind this lemma is that if we take H as a face of T and we choose x, y, z as the sample points then the error between the exact and approximate α-hulls regarding H is bounded by e_1. In practice, to manufacture an object efficiently (by cutting off excessive materials) with a NC-machine the angle θ can not be too small. At the same time, e_1 has to be bounded. The above lemma shows that these objectives can be achieved within certain error range. For example, if $\theta = \frac{\pi}{12} = 15°$ then $e_1 \approx 0.04r$, which is usually satisfactory.

As for the second kind of error, the careful reader might have already noticed that there could be another kind of protrusion, that is, the ball is supported by the two endpoints of the nearest edge together with two vertices which can or can not be the vertices of a triangle in the mesh generation.

Lemma 3.4: Let B be a ball with radius r and center o, $\overline{xy}$ be the closest edge to o, $\overline{oz}$ be the distance between o and $\overline{xy}$ (w is the midpoint of $\overline{xy}$), H be the plane through $\overline{xy}$ such that $\overline{ow}$ is also the distance between o and H, a, b be the two other vertices supporting B and β be the angle between $\overline{ox}$ and $\overline{ow}$. The maximum distance between H and the spherical cap cut off by H is

$$D = r - r\cos\beta,$$

where $\sin\beta \leq \frac{d}{2r}$. Consequently, the Hausdorff distance between B and the exact α-hull of T is bounded by

$$e_2 < 3D.$$

Proof: (Refer to Figure 3.6) Following the Pythagorean theorem, we have

$$D = r - |ow| = r - r\cos\beta,$$

and since the length of $\overline{xy}$ is at most d, $\sin\beta \leq \frac{(\frac{d}{2})}{r} = \frac{d}{2r}$ (note that $e_1 > D$).

As for the Hausdorff distance between B and the exact α-hull of T, we could see that B may also protrude the faces adjacent to a and b by an amount of at most D. This implies that the center of B is translated three times to reach the current position; consequently, the Hausdorff distance between B and the exact α-hull of T is bounded by

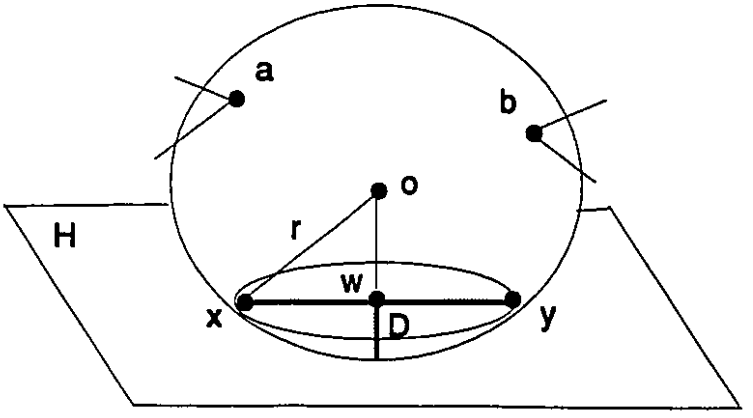


Figure 3.6: Illustration for the proof of Lemma 3.8.

$$e_2 < 3D.$$

□

Our algorithm is essentially based on the above two lemmas. Suppose a solid terrain T, α, and the allowed approximation error $\mathcal{E}$ are already given. Since

$$\mathcal{E} \geq \max(e_1, e_2) = \max \begin{cases} \frac{1}{\alpha}(1-\cos\theta) & \text{where } \sin\theta \leq \frac{d\alpha}{\sqrt{3}} \\ \frac{3}{\alpha}(1-\cos\beta) & \text{where } \sin\beta \leq \frac{d\alpha}{2}, \end{cases}$$

we have

$$d \leq \max \begin{cases} \sqrt{\frac{6\mathcal{E}}{\alpha} - 3\mathcal{E}^2} \\ \sqrt{\frac{8\mathcal{E}}{3\alpha} - \frac{4\mathcal{E}^2}{9}} \end{cases}.$$

If $\mathcal{E} \leq \frac{30}{23\alpha}$ then the maximal length d of the equilateral triangles in the mesh generation is determined by

$$d = \max \begin{cases} \sqrt{\frac{6\mathcal{E}}{\alpha} - 3\mathcal{E}^2} \\ \sqrt{\frac{8\mathcal{E}}{3\alpha} - \frac{4\mathcal{E}^2}{9}} \end{cases} = \sqrt{\frac{6\mathcal{E}}{\alpha} - 3\mathcal{E}^2} \leq \sqrt{\frac{6\mathcal{E}}{\alpha}}.$$

Therefore we have the following theorem.

Theorem 3.5: Let T be a solid terrain and B be a ball with radius $1/\alpha$. The approximate α-hull of T can be computed in $O((N+n)^2)$ time, where N is the

number of vertices added in the mesh and is inversely proportional to the accuracy of the approximation.

Proof: We first compute the maximum edge length in the mesh generation given α and the allowable error $\mathcal{E}$. Then we obtain a mesh generation with the maximum edge length d. This creates N new vertices and we can run the algorithm of Edelsbrunner et al. [EKS83, EM94] in $O((N+n)^2)$ to obtain the approximate α-hull with error at most $\mathcal{E}$. N is related not only to the area of the surface of T but also to the shape of the faces of T. In practice most of the objects to be machined are well-shaped, therefore we could give a bound on N, i.e., $N = O(\frac{\text{area}(T)}{\text{area}\Delta_{xyz}}) = O(\frac{\text{area}(T)}{d^2})$. When $\mathcal{E}$ is small, $N = O(\frac{\text{area}(T)}{\mathcal{E}})$. □

It should be noted that the α-hull of T can never protrude any face of T. With this property, we can *smooth* some part of some faces of T so that there is no protrusion below this part. For example, if a face f of T is large and flat then we can smooth some part of f. Although this procedure does not decrease the error between the approximate and exact α-hull, it does so for that specific part. It is conceivable that this procedure is useful in practice since many products of NC-machining are not general terrains and usually have many flat faces.

One of the main open questions in this regard is to improve the $O(n^3)$ upper bound for computing the α-hull of a terrain.

Open Problem 2: Is it possible to improve the $O(n^3)$ upper bound for computing the α-hull of a terrain?

Chapter 4

Tetrahedralizing special classes of solid terrains

Decomposing a geometric object into simpler parts is one of the most fundamental problems in computational geometry [CD85, Kei85]. This decomposition is employed in such applications as graphics, pattern recognition, solid modeling and mesh generation for finite element methods.

It is well known that any simple polygon can be triangulated [Len11]. The problem of triangulating a simple polygon in linear time has been one of the main research problems in computational geometry for nearly two decades. Recently Chazelle showed that a simple polygon can be triangulated in linear time [Cha91]. The problem of tetrahedralizing a simple polyhedron in 3D without adding new vertices (Steiner points) is significantly more difficult than its 2D counterpart. Schoenhardt gave a counterexample (see the 6-vertex twisted prism in Figure 4.1) which shows that it is not always possible to tetrahedralize a simple polyhedron [Sch28]. Bagemihl extended Schoenhardt's result by showing that there exist n-vertex simple polyhedra which can not be tetrahedralized [Bag48] (Figure 4.2). Seidel gave a counterexample to show that not all simple rectilinear (isothetic) polyhedra can be tetrahedralized (Chapter 10 of [O'R87]). Recently Ruppert and Seidel showed that it is NP-complete to decide if a simple polyhedron can be tetrahedralized [RS92]. Nevertheless Chazelle and Palios showed that if $O(r^2)$ Steiner points are allowed, a simple polyhedron can

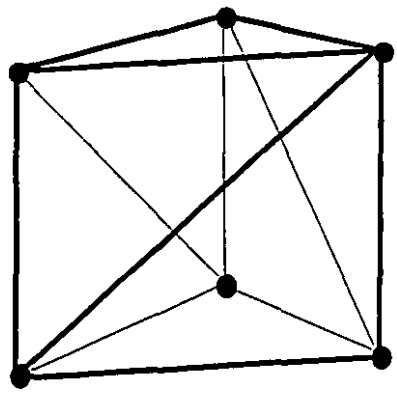


Figure 4.1: Schoenhardt's counterexample.

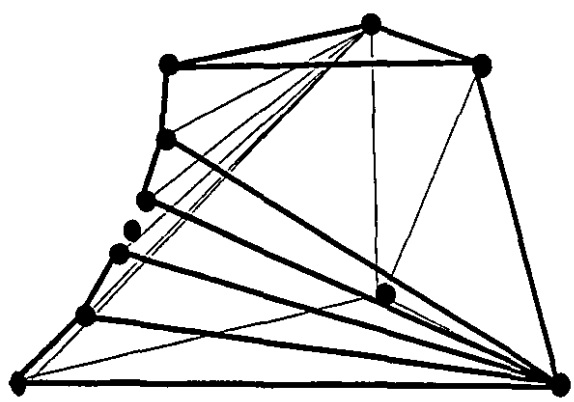


Figure 4.2: Bagemihl's counterexample.

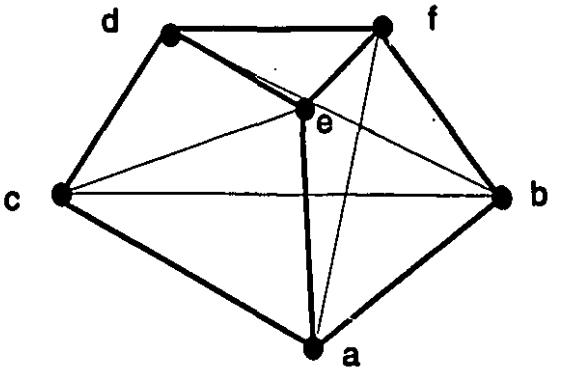


Figure 4.3: A solid terrain can not be tetrahedralized.

be tetrahedralized into a linear number of tetrahedra in $O(n \log n + r)$ time, where r is the number of reflex edges of the given polyhedron [CP90]. (Given an arbitrary polyhedron A in 3D, we assume that every edge of A is determined by two of its faces. An edge of A is called a *reflex* edge if the outer dihedral angle of the two faces defining that edge is less than π, it is called a *flat (coplanar)* edge if the angle is equal to π, otherwise it is called a *convex* edge. A vertex is called *reflex* if it is adjacent to at least one reflex edge.)

We first give a counterexample to show that a solid terrain is not always tetrahedralizable. The example (see Figure 4.3) is essentially based on the Schoenhardt polyhedron. The coordinates of the vertices on the bottom face are $a = (0, -\frac{4\sqrt{3}}{3}, 0)$, $b = (2, \frac{2\sqrt{3}}{3}, 0)$ and $c = (-2, \frac{2\sqrt{3}}{3}, 0)$. Initially we have a regular frustum such that the coordinates of the vertices on the top faces are $(0, -\frac{2\sqrt{3}}{3}, 2)$, $(1, \frac{\sqrt{3}}{3}, 2)$ and $(-1, \frac{\sqrt{3}}{3}, 2)$. We rotate the upper face, a regular triangle, in counterclockwise order around its center by a small angle θ. We have the coordinates of the vertices of the new top face as follows: $d = (-\frac{2\sqrt{3}}{3}\cos(\frac{\pi}{6} - \theta), \frac{2\sqrt{3}}{3}\sin(\frac{\pi}{6} - \theta), 2)$, $e = (\frac{2\sqrt{3}}{3}\sin(\theta), -\frac{2\sqrt{3}}{3}\cos(\theta), 2)$ and $f = (\frac{2\sqrt{3}}{3}\cos(\frac{\pi}{6} + \theta), \frac{2\sqrt{3}}{3}\sin(\frac{\pi}{6} + \theta), 2)$ (see Figure 4.4).

The outer norm of the face Δace is

$$\vec{n}_1 = < -4\sqrt{3}, -4, 4\sin\theta - \tfrac{4\sqrt{3}}{3}\cos\theta + \tfrac{8\sqrt{3}}{3} >,$$

while the outer norm of the face Δcde is

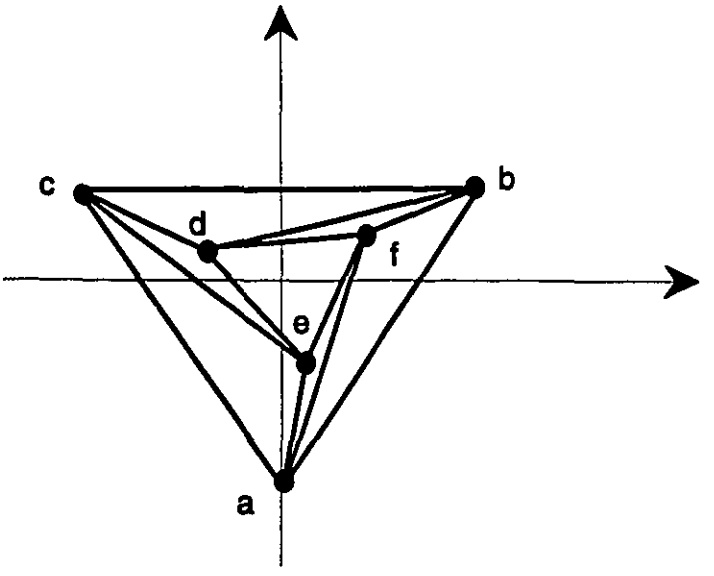


Figure 4.4: A solid terrain can not be tetrahedralized (top view).

$\vec{n}_2 =< -\frac{4\sqrt{3}}{3}(\cos\theta + \sin(\frac{\pi}{6} - \theta)),$
$-\frac{4\sqrt{3}}{3}(\sin\theta + \cos(\frac{\pi}{6} - \theta)),$
$\frac{4}{3}[\sin\theta\sin(\frac{\pi}{6} - \theta) - \sin\theta + \sqrt{3}\sin(\frac{\pi}{6} - \theta) - \cos\theta\cos(\frac{\pi}{6} - \theta) - \cos(\frac{\pi}{6} - \theta) + \sqrt{3}\cos\theta] > .$

If θ is very small, the polyhedron we obtain is clearly not tetrahedralizable since all the vertices are reflex (edges $\overline{ce}$, $\overline{af}$ and $\overline{bd}$ are all reflex) (compare Figures 4.3 and 4.4). We only need to show that when θ is very small, there exists $\vec{n}=< 0, 0, 1 >$ such that $\vec{n} \bullet \vec{n}_i > 0$ for all the faces (with norm $\vec{n}_i$) except Δabc. By symmetry it suffices to show that for $i = 1, 2$.

In fact we have

$$\lim_{\theta \to 0} \vec{n} \bullet \vec{n}_1 = \frac{4\sqrt{3}}{3} > 0 \text{ and}$$
$$\lim_{\theta \to 0} \vec{n} \bullet \vec{n}_2 = \frac{2\sqrt{3}}{3} > 0.$$

Consequently the polyhedron formed by a, b, c, d, e, f is a solid terrain which does not admit a tetrahedralization. Therefore, we have the following theorem.

Theorem 4.1: A solid terrain does not always admit a tetrahedralization.

Although it seems that tetrahedralizing a general simple polyhedron in 3D is an intractable problem, there are some results known regarding tetrahedralizing special classes of simple and non-simple polyhedra. It is well known that a convex polyhedron in 3D can always be tetrahedralized in linear time. Recently Goodman and Pach [GP88] proved that the class of simple polyhedra defined by $CH(P \cup Q) - P - Q$ ($CH(P \cup Q)$ is the convex hull of P and Q), such that P and Q are both convex polyhedra and $P \cap Q = \emptyset$, can always be tetrahedralized. They also showed that the class of non-simple polyhedra defined by $P - Q$, such that P and Q are both convex polyhedra and $Q \subset P$ (we call such a polyhedron a *convex annulus* henceforth), can always be tetrahedralized. Both of these two algorithms have a time complexity of $O(n^2)$. Bern showed that the first algorithm is optimal and he proposed an $O(n \log n)$ time algorithm to improve the second one [Ber93].

Although we have just presented a counterexample to show that not all solid terrains can be tetrahedralized, we show in this chapter that the following classes of simple and non-simple polyhedra can always be tetrahedralized: simple slab (with or without holes), subdivision slab, a box with fixed-depth rectilinear holes (a type-1 box henceforth) and a box with linearly ordered rectangular holes (a type-2 box henceforth). The first two classes of polyhedra can be considered as generalized solid terrains, while the latter two classes of boxes belong to the class of solid Manhattan terrains. We also discuss the problem of tetrahedralizing the class of polyhedra which are the union of a collection of convex polyhedra.

4.1 Tetrahedralizing simple and non-simple slabs

We begin by giving some elementary definitions. A simple slab S is a simple polyhedron defined by translating a simple polygon F with m vertices until it reaches another identical polygonal face F' such that both F and F' are faces of S and all other faces defined by $\overline{ab} \in F, \overline{a'b'} \in F'$ are parallelograms. If F is a triangle, S is called a *prism*. If F is a simple polygon with k holes we call the resulting non-simple polyhedron a slab with k holes. In a prism we say that two diagonals are *coincident* if

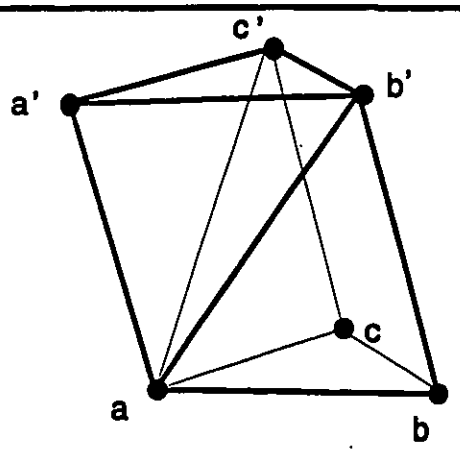


Figure 4.5: Two coincident diagonals in a prism.

they meet at a vertex (see Figure 4.5). Note that if a prism has two coincident diagonals then we can tetrahedralize the prism in two different ways: we can triangulate the untriangulated face (free face henceforth) by inserting a diagonal arbitrarily. The dual of a triangulated simple polygon is defined as the graph such that the vertices of the graph correspond to the triangles in the triangulated polygon and there is an edge between two vertices in the graph if and only if the two corresponding triangles in the triangulated polygon share a diagonal. It is known that the dual of a simple polygon with n vertices is a tree with $n-2$ vertices and $n-3$ edges; furthermore, once the polygon is triangulated its dual can be computed in linear time [O'R87]. We propose the following algorithm to show that an n-vertex slab with k holes can be tetrahedralized.

Algorithm 4.1

BEGIN

(1) Triangulate the two polygons with holes F, F' such that they have the same triangulation.

(2) Construct a graph G with $k+1$ vertices such that each vertex corresponds to a hole or the outermost polygon. Any diagonal which connects two holes or connects a hole with the outermost polygon defines the edge between the two corresponding vertices in G, if such an edge is not defined yet. This yields

a connected graph G. Compute a spanning tree of G. Label the k diagonals which correspond to the k edges of this spanning tree cutting diagonals and double these cutting diagonals. This gives us a triangulated polygon (with no holes).

(3) Visit every prism whose top has one cutting diagonal and insert coincident diagonals on the faces of the prism opposite to the face whose top edge is the cutting diagonal (this makes the face a free face). These prisms are labelled constrained prisms because if one of the inserted diagonals is later flipped then so is its neighbor in order to preserve coincidence and the resulting freedom for its cutting diagonal face.

(4) Visit every prism whose top has two cutting diagonals and insert coincident diagonals on the faces of the prism containing the cutting diagonals. If more than one triangle each containing two cutting diagonals have these cutting diagonals anchored on the same vertex then all inserted diagonals are made coincident on this vertex. If a set of triangles each containing two cutting diagonals share cutting diagonals alternatively then all inserted diagonals are made alternatively coincident to the vertices of F and F' on which two connecting two cutting diagonals are anchored (refer to Figure 4.6). These prisms are labelled frozen prisms because these diagonals will never be changed.

(5) Compute the dual tree of the triangulation obtained at Step (2).

(6) Starting with any leaf perform a depth-first search of the dual tree to visit all prisms. Triangulate all prisms except frozen ones. If a prism is ordinary it can be triangulated arbitrarily. If the prism is constrained then coincidence should be respected.

END

Proof of Correctness:

As shown in Figure 4.7, the possible conflicts arise when we run the depth-first search and triangulate the relevant prisms; furthermore, conflicts can only arise at prisms which contain cutting diagonals where we must make sure that the prisms on both sides of the cutting diagonals contain matching diagonals. We know that there can not be any face in the triangulation of the upper face with three cutting diagonals,

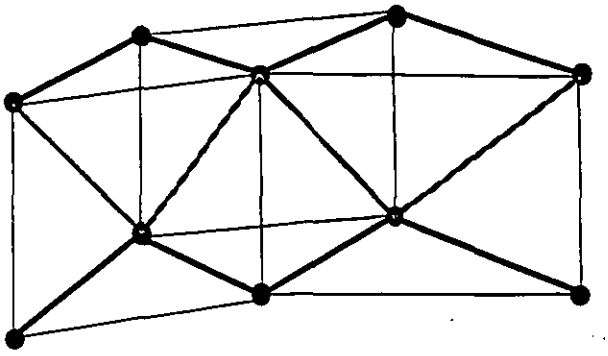


Figure 4.6: Three type-3 prisms share cutting diagonals alternatively.

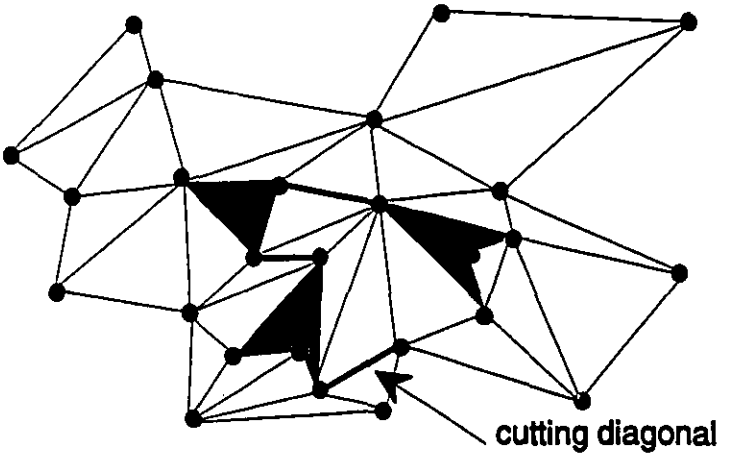


Figure 4.7: Tetrahedralizing a slab (top view).

since the existence of such a face implies that the spanning tree we obtain in Step (2) has a cycle. Consequently, a prism can contain at most two cutting diagonals. We call a prism corresponding to a leaf in the dual tree whose upper face contains one cutting diagonal a *type-1* prism; a prism corresponding to a degree two node in the dual tree whose upper face contains one cutting diagonal a *type-2* prism and a prism corresponding to a leaf in the dual tree whose upper face contains two cutting diagonals a *type-3* prism.

Now we consider the following cases:

(I) A type-1 prism or a type-2 prism shares the free face with another type-1 prism or another type-2 prism. According to Step (3), the top edge of the free face is a cutting diagonal and the free face can be triangulated arbitrarily. We just need to choose one triangulation to make the tetrahedralization of the two prisms consistent.

(II) A type-1 prism or a type-2 prism shares its free face with a type-3 prism. According to Step (4), this face is a face of the type-3 prism and is already triangulated by inserting a coincident diagonal. This gives us a triangulation of the free face (hence a tetrahedralization of the two prisms).

(III) A type-3 prism shares a cutting diagonal face with another type-3 prism. Again according to Step (4), the face of a type-3 prism whose top edge is not a cutting diagonal is a free face. If the two (or more) prisms have their cutting diagonals anchored on the same vertex then all inserted diagonals are incident to this vertex therefore the face separating the two prisms has a consistent triangulation; otherwise we insert coincident diagonals on the non-free faces of one prism, then according to the triangulation of the face separating the two prisms we insert coincident diagonals on the non-free faces of the other prism (see Figure 4.6). For both of these two subcases any triangulation of the free faces gives us a tetrahedralization of the two prisms. □

Complexity Analysis of Algorithm 4.1:

Step (1) takes $O(n \log n)$ time and space [AAP86]. Each of the remaining steps takes $O(n)$ time and space. Since a triangulated polygon with n vertices has $O(n)$ edges and faces, the graph G we build at Step (2) has at most $O(n)$ edges. Consequently, a spanning tree of G can be computed in $O(n)$ time [AHU83]. With a similar

argument, we can show that each of Steps (3)-(6) takes $O(n)$ time. Therefore the algorithm runs in $O(n \log n)$ time and linear space. Furthermore this is optimal since any tetrahedralization of a polyhedron must triangulate all the faces of the polyhedron and triangulating a polygon with holes with a total of $O(n)$ vertices (i.e., F, F') has a lower bound of $\Omega(n \log n)$ in the algebraic computation tree model [AAP86]. If F is a simple polygon with no holes, then Step (1) takes $O(n)$ time and so does the whole algorithm. We therefore have the following theorem.

Theorem 4.2: An n-vertex polygonal slab with holes can be tetrahedralized in optimal $O(n \log n)$ time and linear space and a simple n-vertex slab can be tetrahedralized in optimal $O(n)$ time.

Now we generalize Algorithm 4.1 to obtain an algorithm for tetrahedralizing a (bounded) *subdivision slab* with n vertices. A (bounded) subdivision slab is built by translating a planar subdivision to a different plane (without loss of generality, assume the two planes are parallel to the XY-plane) such that every face which is not parallel to the XY-plane is a parallelogram.

The correctness proof is sufficiently similar to that for Algorithm 4.1 to be omitted. We only show how to change the problem such that we can apply Algorithm 4.1 directly.

Algorithm 4.2:

BEGIN

(1) Triangulate the upper and lower faces of the subdivision slab.

(2) Compute a spanning tree of the dual graph of the triangulation. If an edge of the dual graph is not an edge of the spanning tree, then double the edge in the triangulation which corresponds to that dual graph edge. This yields a triangulated polygon. To make the terminology consistent with what we used in Algorithm 4.1, we call these edges which are doubled cutting diagonals.

(3) Run Steps (3)-(6) of Algorithm 4.1.

END

Complexity Analysis of Algorithm 4.2:

Step (1) has a time complexity of $O(n \log n)$ since the face of a planar subdivision is

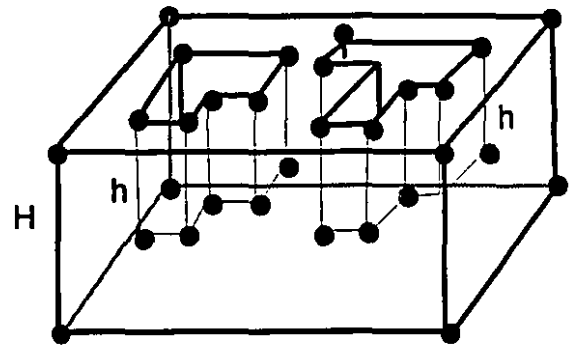


Figure 4.8: A type-1 box.

a polygon with holes and triangulating a polygon with holes with $O(n)$ vertices takes $\Theta(n \log n)$ time [AAP86]. Step (2) takes $O(n)$ time since the size of the triangulated subdivision is $O(n)$ (so is its dual) and a spanning tree of a graph with $O(n)$ edges can be computed in linear time [AHU83]. The remaining steps take a total of $O(n)$ time following the analysis of Algorithm 4.1. Therefore Algorithm 4.2 runs in $O(n \log n)$ time and linear space. We have consequently established the following theorem.

Theorem 4.3: An n-vertex subdivision slab can be tetrahedralized in $O(n \log n)$ time and linear space.

4.2 Tetrahedralizing special classes of solid Manhattan terrains

We mention early in this chapter that it is not always possible to tetrahedralize a simple rectilinear polyhedron. However we strongly believe that any solid Manhattan terrain admits a tetrahedralization, although we have not found a proof yet. In this section we show that certain special classes of solid Manhattan terrains can always be tetrahedralized.

We first give a definition of the two special classes of solid Manhattan terrains to be studied in the following context. A *type-1* box is defined as follows (see Figure

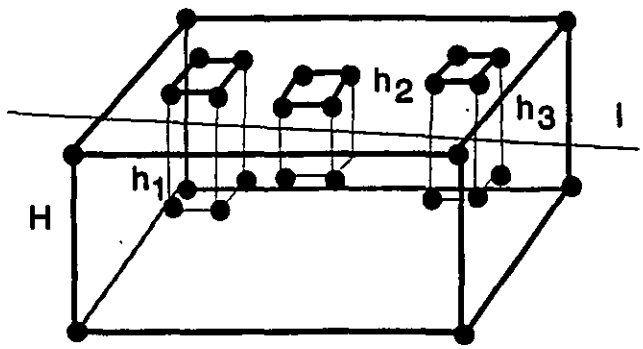


Figure 4.9: A type-2 box.

4.8): given a 3D rectilinear box with height H, dig a set of *non-intersecting* rectilinear holes at the top of the box such that each of these holes has a unique depth $h < H$, we call the resulting simple rectilinear polyhedron a type-1 box. A *type-2* box is defined similarly (see Figure 4.9): given a 3D rectilinear box with height H, dig a set of nonintersecting rectangular holes on the top of the surface of the box such that the heights of the holes $h_i < H$ (for all i), and the holes are linearly ordered along a direction l on F (i.e., the orthogonal projections of the holes onto l form a set of disjoint line segments on l and the sorted order of these line segments along l is already known).

Since the algorithms in this section rely heavily on the algorithms by Goodman and Pach [GP88] and Bern [Ber93], we give below a detailed description of these algorithms.

Goodman and Pach's algorithms

Assume P, Q are two convex polytopes in d-dimension, in [GP88] it was shown that the polytopes $CH(P \cup Q) - P - Q$ where $P \cap Q = \emptyset$ and $P - Q$ where $Q \subset P$ all admit a cell decomposition whose maximal cells are openly disjoint convex polytopes. Goodman and Pach first proved the following theorem as a prelude.

Theorem 4.4 [GP88]: Let H be a hyperplane crossing a d-dimensional convex polytope P. Assume that H does not contain any vertex of P, and let H^+ and H^- be the two halfspaces determined by H. Then P admits a cell decomposition whose maximal cells are openly disjoint convex polytopes $Q_1, Q_2, \ldots$ such that each $Q_i = CH(F_i \cup G_i)$, where (a) $F_i \subset H^+$, $G_i \subset H^-$ are proper faces of P and (b) $\dim F_i + \dim G_i = d-1$.

We only give a sketch of the proof. The proof can be thought of as "bending a polytope about a hyperplane". It consists essentially of projecting the vertices of the polytope P onto a wedge in $(d+1)$-dimension formed by two d-dimensional halfspaces meeting along H, then computing the convex hull P' of the new set of vertices in $(d+1)$−dimension, and finally projecting the facets of the "upper" boundary of P' back into the original d-space (see Figure 4.10). A face Q' of P' is called an "upper" facet if Q' can be "seen" from the $+\infty$ of the $(d+1)$st coordinate axis.[1] This results in a cell decomposition of P which is transverse to H.

In Figure 4.10 we present a 2D convex polytope P with seven vertices and a hyperplane H crossing P but does not contain any vertex of P. Following Goodman and Pach's proof we project the vertices of P onto the 3D wedge defined by H, then we compute the convex hull P' of the projected vertices in 3D and finally we project the upper hull of P' back to 2D. This results in a triangulation of P such that the vertices of each internal diagonal do not lie on the same side of H.

In 3D the above theorem implies that if P is a convex polyhedron, H is a plane cutting through P and H does not contain any vertex of P then there exists a tetrahedralization of P such that each tetrahedron has the property that the vertices of the tetrahedron do not all belong to one of the two halfspaces H^+, H^-.

By modifying the proof for the above theorem slightly, Goodman and Pach showed the following two theorems relevant to this thesis. As the above proof, these proofs are all constructive; consequently, Goodman and Pach have implicitly proposed the algorithms to compute the cell decompositions of the two classes of polytopes $CH(P \cup Q) - P - Q$ and $P - Q$. Furthermore, since the core of their algorithms is to compute

[1] Mathematically, this means that there exist $(s,t) \in R^{d+1}$ $(s \in R^d, t > 0)$ and a real α such that $aff\, Q' = \{(x,y) \in R^{d+1} : s \bullet x + ty = \alpha\}$ and $P' \subset \{(x,y) \in R^{d+1} : s \bullet x + ty \leq \alpha\}$. Here $aff\, Q'$ denotes the affine subspace of R^{d+1} spanned by the points of Q'.

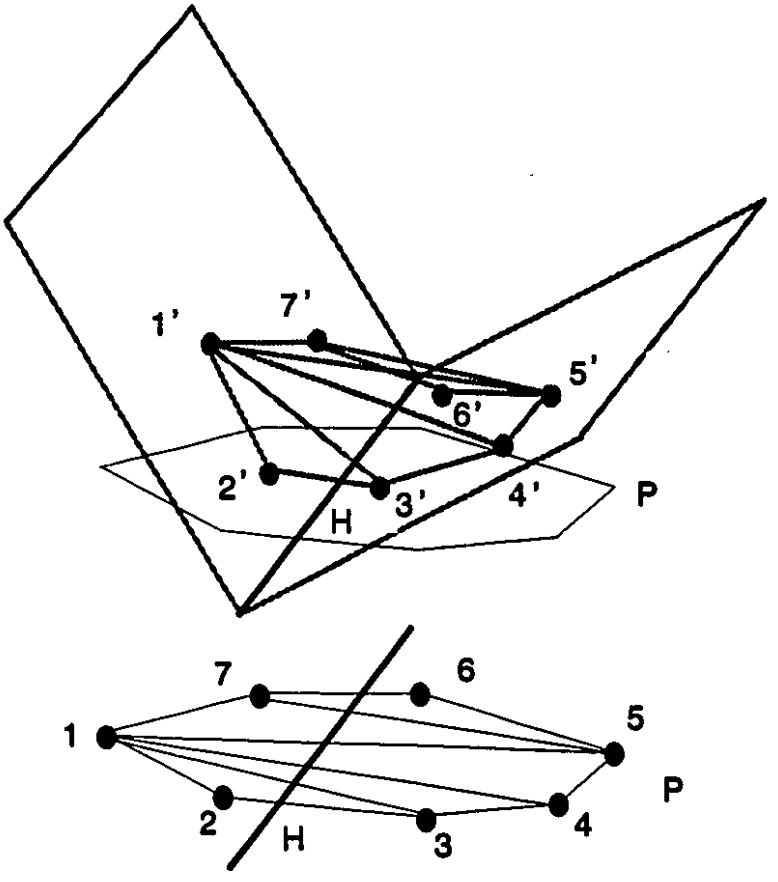


Figure 4.10: Illustration for the proof of Theorem 4.4.

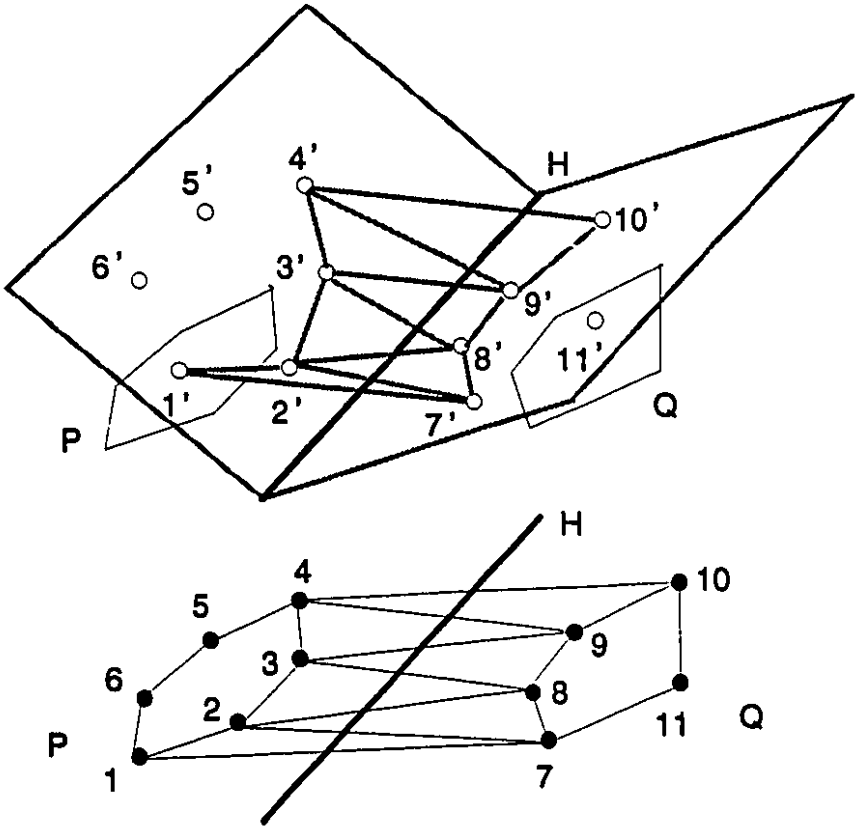


Figure 4.11: Illustration for the proof of Theorem 4.5.

the convex hull of n points in $(d+1)$−dimensions, both of these two algorithms takes $O(n^{\lfloor \frac{d+1}{2} \rfloor})$ time, with the recent optimal convex hull algorithm by Chazelle [Cha93]. Clearly both algorithms run in $O(n^2)$ time when $d = 3$. Now we introduce the following theorem for compute the cell decomposition of the polytope $CH(P \cup Q) - P - Q$ where $P \cap Q = \emptyset$.

Theorem 4.5 [GP88]: Let P and Q be two disjoint d-dimensional convex polytopes. Then $CH(P \cup Q) - P - Q$ admits a cell decomposition whose maximal cells are openly disjoint convex polytopes Q_1, Q_2,... such that each $Q_i = CH(F_i \cup G_i)$, where F_i and G_i are proper faces of P and Q, respectively, and dim F_i + dim $G_i = d - 1$.

Again we only give a sketch of the proof. Interested reader is encouraged to refer to [GP88] for the details of the proof. Let H be a hyperplane strictly separating P from Q. Goodman and Pach's proof consists essentially of projecting the vertices of the polytopes P, Q onto a wedge in $(d+1)$−dimension formed by two d−dimensional halfspaces meeting along H, then computing the convex hull P' of the new set of vertices in $(d+1)$-dimension, and finally projecting the facets of the "lower" boundary of P' back into the original d-space (see Figure 4.11). A face S' of P' is called "lower" facet if S' can be "seen" from the $-\infty$ of the $(d+1)$st coordinate axis.[2] This results in a cell decomposition of P which is transverse to H.

In Figure 4.11 we show two disjoint convex polytopes P and Q and a hyperplane H strictly separating P from Q. Following their proof we project the vertices of P onto the 3D wedge defined by H, then we compute the convex hull P' of the projected vertices in 3D and finally we project the lower hull of P' back to 2D. This results in a triangulation of the simple polygon $CH(P \cup Q) - P - Q$ such that the vertices of each internal diagonal of $CH(P \cup Q) - P - Q$ do not lie on the same side of H.

In 3D the above theorem implies that if P and Q are two disjoint convex polyhedra then the polyhedron $CH(P \cup Q) - P - Q$ admits a tetrahedralization. Finally we introduce the following theorem for compute the cell decomposition of the polytope $P - Q$ $(Q \subset P)$.

Theorem 4.6 [GP88]: Let P and Q be two d-dimensional convex polytopes and $Q \subset P$. Then $P - Q$ admits a cell decomposition whose maximal cells are openly disjoint convex polytopes Q_1, Q_2,... such that each $Q_i = CH(F_i \cup G_i)$, where F_i and G_i are proper faces of P and Q, respectively, and dim F_i + dim $G_i = d - 1$.

The proof of this theorem is made again by modifying that of Theorem 4.4. In this case H is a d-hyperplane at infinity and the wedge is defined by a pair of parallel halfspaces in $(d+1)$−dimensions. We first project the vertices of P, Q onto the two parallel halfspaces respectively, then we compute the convex hulls P', Q' of the projected vertices and finally we project the convex hulls of P', Q' back to d−space. This results in a cell decomposition of $P - Q$ which is transverse to H. In 3D Theorem

[2] Mathematically, this means that there exist $(s,t) \in R^{d+1}$ $(s \in R^d, t < 0)$ and a real α such that aff $S' = \{(x,y) \in R^{d+1} : s \bullet x + ty = \alpha\}$ and $P' \subset \{(x,y) \in R^{d+1} : s \bullet x + ty \leq \alpha\}$.

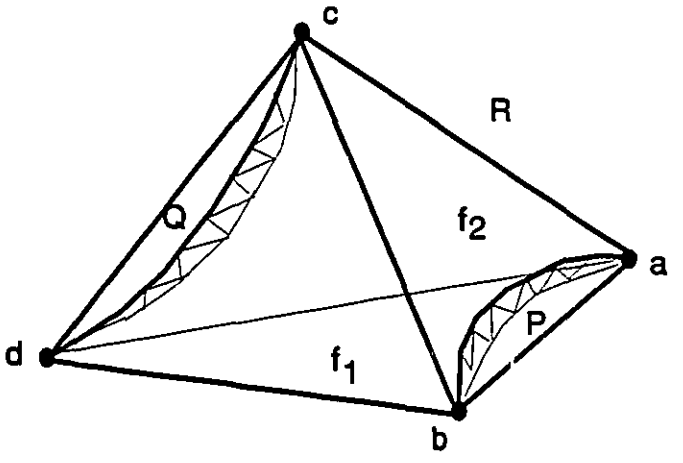


Figure 4.12: A tetrahedralization of $CH(P \cup Q) - P - Q$ can have $\Omega(n^2)$ tetrahedra.

4.6 implies that if P, Q are two convex polyhedra such that $Q \subset P$ then $P - Q$ admits a tetrahedralization. Recently Bern have proposed an $O(n \log n)$ time algorithm to tetrahedralize $P - Q$, which will be presented in the following paragraphs.

Bern's algorithm

Bern [Ber93] showed that algorithm of Goodman and Pach for tetrahedralizing the class of polyhedra $CH(P \cup Q) - P - Q$ is optimal [3] and proposed an $O(n \log n)$ time algorithm to improve their algorithm for tetrahedralizing the class of polyhedra $P - Q$.

Theorem 4.7 [Ber93]: Let P, Q be two disjoint convex polyhedra in 3D. $\Omega(n^2)$ tetrahedra are necessary to tetrahedralize $CH(P \cup Q) - P - Q$.

[3] However, Bern left out some crucial details in the abstract and from what he presented in that abstract his proof was not convincing although his idea was correct. He constructed two bands of triangles, one oriented vertically and the other horizontally. He claimed that each tetrahedron has volume $O(1)$ and if the volume of the polyhedron is $O(n^2)$ then any tetrahedralization must have $\Omega(n^2)$ tetrahedra. But it is not convincing that the polyhedron constructed by Bern has volume $O(n^2)$, even if the two original bands are parallel. The reason is that there is no exact formula to compute the volume bounded by two parallel convex polygons unless the sectional area function of the resulting polyhedron is given explicitly (See Chapter 8 of [KB38]).

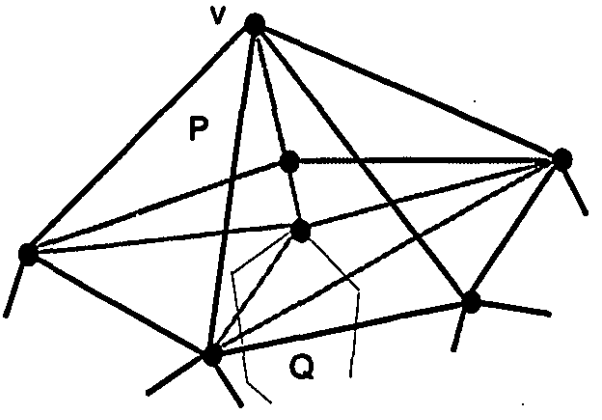


Figure 4.13: The cap of a $P \ominus Q$ vertex.

Proof [4]: We construct a regular tetrahedron R such that the length of each edge of the tetrahedron is n. It is clear that the volume of the tetrahedron is $O(n^3)$. We choose two non-adjacent edges, $\overline{ab}$ and $\overline{cd}$, of the tetrahedron R and we plot two convex polygonal curves with size $O(n)$ on the two faces f_1, f_2 incident to $\overline{ab}$. These two convex polygonal curves, together with $\overline{ab}$, define a convex polyhedron P. We can choose these two convex polygonal curves such that they are very close to $\overline{ab}$; furthermore, we can also plot $O(n)$ vertices on the surfaces of P bounded by f_1, f_2 such that all the edges of P, except $\overline{ab}$, are of length of $O(1)$ (see Figure 4.12). We construct Q similarly based on $\overline{cd}$. The tetrahedra that lie in $CH(P \cup Q) - P - Q$ use either a small triangular face from one convex polygon and a vertex from the other, or two edges from the two different convex polyhedra. In either case, the volume of such a tetrahedron is $O(n)$. The volume of $CH(P \cup Q) - P - Q$ is very close to the original tetrahedron R hence is also $O(n^3)$; therefore, any tetrahedralization of $CH(P \cup Q) - P - Q$ has at least $\Omega(n^2)$ tetrahedra. □

Given a simple polyhedron A and a convex vertex u of A, let $cap(u)$ be the polyhedron that is the closure of the difference of A and the convex hull of all the

[4]This proof, with full details, is a modification to the original proof by Bern [Ber93].

vertices of of A except u. It is known that $cap(u)$ is a star-shaped polyhedron and u lies in its kernel; consequently, $cap(u)$ can be tetrahedralized in time linear of its size [CP90].

The basic idea of Bern's algorithm for tetrahedraling $P-Q$ is to remove iteratively a $cap(v)$ of $P-Q$. Each vertex of $cap(v)$ is either a vertex of P adjacent to v, or is a vertex of Q lying outside $CH(P-v)$. After the removal of $cap(v)$, P shrink to a smaller convex polyhedron, with vertex set equal to all the remaining vertices of P and some vertices of Q. We say these vertices of Q *pop up*, i.e., first appear on P after the removal of $cap(v)$ (Figure 4.13). We call a vertex u a $P \ominus Q$ vertex if u belongs to P but not Q. If time complexity is not of concern, we can keep on deleting the cap of a $P \ominus Q$ vertex u until all the vertices of Q have popped up. Computing $cap(u)$ takes $O(n \log n)$ time since computing the convex hull of $O(n)$ points in 3D takes $\Theta(n \log n)$ time [PS85]. Therefore by deleting (and tetrahedralizing) the caps of $P \ominus Q$ vertices consecutively it takes $O(n^2 \log n)$ time to tetrahedralize $P-Q$. Therefore this tetrahedralization produces $O(n^2)$ tetrahedra.

To improve the complexity of the above procedure, we delete an independent set of $P \ominus Q$ vertices (and their caps). Such an independent set I can be computed in $O(n)$ time and enables the removal of $O(n)$ tetrahedra. The total number of iterations of such removals is $O(\log n)$ [Kir83], [DK85] (see also Chapter 4 about the construction of independent sets). The harder part is to compute the caps of vertices in I. To accomplish this, we first compute P', the convex hull of the vertices of $P-I$. This can be done in linear time by filling the "holes" resulting from the deletions of independent vertices, using a brute-force method (since each hole has $O(1)$ size). We then compute the convex hull of P' and Q in linear time. Computing the convex hull of the union of two convex polyhedra can be solved using geometric duality: transform the vertices of the polyhedra into faces of dual polyhedra and then compute the intersection of the dual polyhedra. Points interior to both dual polyhedra dualize to planes exterior to both old polyhedra. Therefore $CH(P' \cup Q)$ can be computed in $O(n)$ time using Chazelle's linear time algorithm for intersecting two convex polyhedra [Cha92]. Since there are $O(\log n)$ iterations and as we have just shown, each iteration can be accomplished in $O(n)$ time, the total time and space

complexity for tetrahedralizing $P - Q$ is $O(n \log n)$. Consequently Bern has shown the following theorem.

Theorem 4.8 [Ber93]: Let P, Q be two convex polyhedra in 3D such that $Q \subset P$. The (non-simple) polyhedron $P - Q$ can be tetrahedralized in $O(n \log n)$ time and space.

We now present an algorithm to tetrahedralize a type-1 box. The underlying idea is to partition a type-1 box into disjoint parts such that each part can be tetrahedralized with a known algorithm.

Algorithm 4.3

BEGIN

(1) Compute the convex hull of the rectilinear holes. Since the holes have the same depth, we obtain a slab with holes and height h.

(2) Run Algorithm 4.1 to tetrahedralize the slab obtained in Step (1). Remove all tetrahedra and triangulate the bottoms of those holes. We are left with a box with a hole E such that the bottom of E is a planar triangulation.

(3) Run the algorithm by Bern to tetrahedralize the polyhedron we obtain in Step (2).

END

Proof of Correctness:

According the algorithm, after Step (1) and (2) we have a rectilinear box with a hole such that there are coplanar edges on the hole. We can think of this as a special convex annulus such that the outer polyhedron is a 8-vertex rectilinear polyhedron and the inner polyhedron is a convex polyhedron (with coplanar edges). We will show that by running a simplied version of Bern's algorithm, this special convex annulus can be tetrahedralized in $O(n \log n)$ time.

From the above discussion on Bern's algorithm, we can notice that in the whole process of deleting caps, no edge is ever added between two vertices of Q. Therefore as long as there is no vertex of P coplanar with a face of Q, the above algorithm works for the case when Q has coplanar vertices and edges.

The convex annulus we have is also special in the sense that P has only 8 vertices.

Therefore we only need to delete and tetrahedralize 8 caps, i.e., we do not have to compute and delete the independent vertices of P as in [Ber93]. Thus this step takes $O(n \log n)$ time following the above discussion. □

Complexity Analysis of Algorithm 4.3:

Step (1) takes $O(n \log n)$ time since it involves computing the convex hull of a set of rectilinear polygons. Step (2) takes $O(n \log n)$ time following Theorem 4.2. We have just showed that Step (3) also takes $O(n \log n)$ time. Therefore Algorithm 4.3 runs in $O(n \log n)$ time and linear space. We have therefore obtained the following theorem.

Theorem 4.9: An n-vertex type-1 box can be tetrahedralized in $O(n \log n)$ time and linear space.

Now we propose a quadratic time algorithm to tetrahedralize an n-vertex type-2 box with k linearly ordered holes: $p(1), p(2), ..., , p(k)$. Although the holes have a linear order along l, if we tetrahedralize them incrementally we simply obtain an $O(n^3)$ time algorithm. This is because that an incremental algorithm takes $O(n)$ step and each step involves tetrahedralizing a $CH(P \cup Q) - P - Q$ *alike* polyhedron, which takes $\Omega(n^2)$ time following Theorem 4.7. We use a divide-and-conquer method to obtain an $O(n^2)$ time algorithm.

Algorithm 4.4

BEGIN

(1) Tetrahedralize the region bounded by the holes by a divide-and-conquer method, i.e., recursively tetrahedralize the regions formed by the first $k/2$ holes and by the last $k/2$ holes, then combine the result by running Goodman and Pach's algorithm.

(2) Run the algorithm of Bern to tetrahedralize the convex annulus we obtain in Step (1).

END

Proof of Correctness:

The only thing we need to show is that Step (1) always works. The crucial point is that the holes are linearly ordered. The first iteration will tetrahedralize

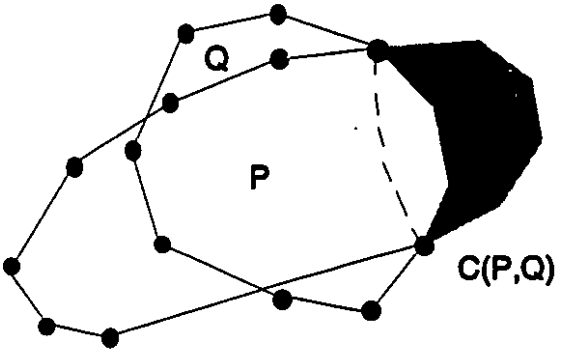


Figure 4.14: A U(2) polyhedron and the illustration for a convex cap.

the regions formed by $CH(p(1),p(2)) - p(1) - p(2)$; $CH(p(3),p(4)) - p(3) - p(4)$; ...; $CH(p(k-1),p(k)) - p(k-1) - p(k)$ (assume k is even). After the first iteration we have $k/2$ convex holes which are still linearly ordered along l. This property holds after any iteration. Therefore we can continue this recursive procedure until we end up with only one convex hole. Step (2) is correct following the correctness proof of Algorithm 4.3. □

Complexity Analysis of Algorithm 4.4:

The analysis of Algorithm 4.4 is as follows: Step (2) runs in $O(n \log n)$ time following Theorem 4.8. The complexity of Step (1) can be measured by the following recurrence relation:

$$T(n) = 2T(n/2) + O(n^2) = O(n^2).$$

Therefore the time complexity of Algorithm 4.4 is $O(n^2)$ and we have the following theorem.

Theorem 4.10: An n-vertex type-2 box can be tetrahedralized in $O(n^2)$ time.

4.3 Tetrahedralizing the union of convex polyhedra

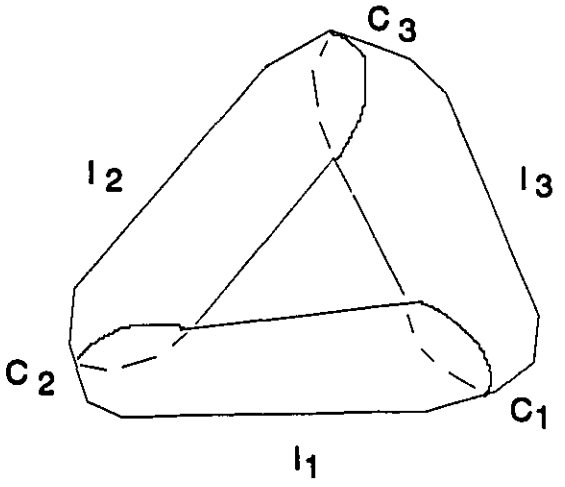


Figure 4.15: An interlocked U(3) polyhedron.

In this section we investigate the tetrahedralization of a class of polyhedra which are the union of a set of convex polyhedra. A polyhedron is called a $U(k)$ polyhedron ($U(k)$ for short) if it is the union of k convex polyhedra. Therefore trivially, $U(1)$ can always be tetrahedralized. We will show that a $U(2), U(3)$ polyhedron can always be tetrahedralized, but a $U(k)$ polyhedron (for $k \geq 4$) can not always be tetrahedralized.

Let P and Q be two convex polyhedra such that $P \cap Q$ is not empty. In order to simplify the analysis of our algorithms in this section, we will not consider the cases when P and Q do not have proper intersections, i.e., when $P \cap Q$ is only a point, a line segment or a polygonal face. Let $\partial P, \partial Q$ be the surfaces of P, Q respectively. Let C, which will be called a *crown* henceforth, be a simple closed polygonal chain of $\partial P \cap \partial Q$. C separates ∂P into two parts $\partial P_1, \partial P_2$ such that $C = \partial P_1 \cap \partial P_2$. Let S_1, S_2 denote all the vertices on $\partial P_1, \partial P_2$ respectively. We call any one of $CH(S_1) - CH(C), CH(S_2) - CH(C)$ a *convex cap* (with respect to Q) and denote it by $C(P, Q)$ (see Figure 4.14). We first show that a $U(2)$ polyhedron can be tetrahedralized in linear time.

Theorem 4.11: An n-vertex $U(2)$ polyhedron can be tetrahedralized in linear time.

Proof: Suppose we are given a $U(2)$ polyhedron which is the union of two convex polyhedra (The two convex polyhedra P, Q do not have to be given explicitly). If the given $U(2)$ polyhedron is convex, then it is clear that it can be tetrahedralized. Otherwise we know that there is at least one reflex edge. Choose a reflex vertex v of the $U(2)$ polyhedron. Since P, Q are all convex v has to be a vertex of a crown of P, Q, i.e., $v \in \partial P, v \in \partial Q$. Therefore v can see the whole interior of P and Q, hence the $U(2)$ polyhedron given is a star-shaped polyhedron with v in its kernel. Consequently, the $U(2)$ polyhedron can be tetrahedralized in linear time. □

Now we start to show that a $U(3)$ polyhedron can always be tetrahedralized. Unlike for a $U(2)$ polyhedron, we assume that the three convex polyhedra are also given. The reason for attaching this condition is that our algorithm needs to identify all the crowns of the given $U(3)$ polyhedron. It is easy to see that a $U(3)$ polyhedron does not have to be simple. A $U(3)$ polyhedron $P \cup Q \cup R$ is called *interlocked* if and only $P \cap Q \cap R = \emptyset$ while $P \cap Q \neq \emptyset$, $P \cap R \neq \emptyset$ and $Q \cap R \neq \emptyset$ (see Figure 4.15). For the following two lemmas, we assume that all the crowns have already been identified.

Lemma 4.12: An n-vertex convex cap can be tetrahedralized in $O(n \log n)$ time.
Proof: (Refer to Figure 4.14) By the definition, a crown C separates the surface of P into two parts $\partial P_1, \partial P_2$ such that $C = \partial P_1 \cap \partial P_2$. Therefore $C \subset \partial P_1$. Consequently $CH(C) \subset CH(S_1)$, which immediately implies that $CH(S_1) - CH(C)$ defines a convex annulus. We know already that a convex annulus can be tetrahedralized in $O(n \log n)$ time [Ber93]. Therefore this lemma is proved. □

Lemma 4.13: An n-vertex interlocked $U(3)$ polyhedron can be tetrahedralized in $O(n \log n)$ time.
Proof: (Refer to Figure 4.15) First of all, following Lemma 4.12 we can tetrahedralize and remove all the convex caps (if any) in a total of $O(n \log n)$ time. Let I denote an interlocked $U(3)$ polyhedron with no convex cap, and let C_1, C_2 and C_3 be its three crowns. Then $CH(C_1), CH(C_2)$ and $CH(C_3)$ separate I into three disjoint parts, I_1, I_2 and I_3. Each of $CH(I_1) \cup CH(C_2)$, $CH(I_2) \cup CH(C_3)$ and $CH(I_3) \cup CH(C_1)$ is a convex cap which can be tetrahedralized in $O(n \log n)$ time following Lemma 4.12. □

We now present the following algorithm to tetrahedralize a $U(3)$ polyhedron. There are several cases of a $U(3)$ polyhedron resulting from three convex polyhedra. The following tetrahedralization algorithm deals with each of these cases separately.

Algorithm 4.5

BEGIN

(1) Test whether the given $U(3)$ polyhedron is convex by traversing all the edges of the polyhedron. If it is convex, tetrahedralize it directly and exit.

(2) Test whether there is a reflex vertex which is adjacent with three reflex edges. If there is such a vertex v then tetrahedralize $U(3)$ by adding diagonals (and consequently faces) from v to all other vertices which are not adjacent with v and then exit.

(3) Compute all the crowns as follows. Start with a reflex edge $e = \overline{v_1 v_2}$ and identify the two faces f_1, f_2 adjacent with e such that $f_1 \in P$, $f_2 \in Q$.
An edge of a crown is always determined by a face $f_1 \in P$ and a face $f_2 \in Q$, therefore always keep $f_1 \in P$ and $f_2 \in Q$ that determine the current crown edge. If $e_1 \in P$ is adjacent with f_1 and $v_1 = e_1 \cap f_2$, then keep the face of P which shares e_1 with f_1 as the new f_1. If $e_2 \in Q$ is adjacent with f_2 and $v_1 = e_2 \cap f_1$, then keep the face of Q which shares e_2 with f_2 as the new f_2 (see Figure 4.16).

Repeat this operation until v_2 is reached, thus one crown has been computed. Continue the above procedure until all the reflex edges are traversed. Consequently all the crowns have been computed.

(4) Remove and tetrahedralize all the convex caps determined by the crowns computed in Step (3) (using the results of Lemma 4.12).

(5) If the remainder is a convex polyhedron, then tetrahedralize it directly. Otherwise, the remainder is an interlocked $U(3)$ polyhedron. Use the method in Lemma 4.13 to tetrahedralize it.

END

Proof of Correctness:

If a $U(3)$ polyhedron is convex, then Step (1) identifies it and tetrahedralizes it directly in linear time. If there exists a reflex vertex v of the $U(3)$ polyhedron $P \cup Q \cup R$

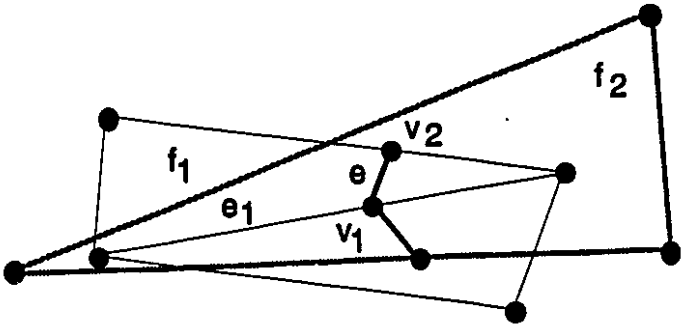


Figure 4.16: Illustration for the computation of a crown.

such that $v \in P$, $v \in Q$, and $v \in R$, then it is clear that the given U_3 polyhedron is star-shaped hence can be tetrahedralized in linear time (see Figure 4.17). We know that the union of any two convex polyhedra can only result in vertices which are adjacent to at most two reflex edges. Therefore, if there exists a reflex vertex which is adjacent with three reflex edges then such a vertex must belong to $\partial P \cap \partial Q \cap \partial R$. Step (2) identifies this case and tetrahedralize it in linear time.

After Step (2), if the algorithm does not stop then we know that every vertex of the $U(3)$ polyhedron belongs to at most two original polyhedra. In this case no two crowns intersect each other. Step (3) identifies all the crowns. Following Lemma 4.12, all the convex caps of P, Q, R can be tetrahedralized in $O(n \log n)$ time. In Step (4), all the convex caps (if any) are removed and tetrahedralized. The remainder is either a convex polyhedron (case (a)), which can be tetrahedralized easily; or an interlocked $U(3)$ polyhedron with no convex cap (case (b)), which can be tetrahedralized in $O(n \log n)$ time following Lemma 4.13. □

Therefore we have established the following theorem.

Theorem 4.14: An n-vertex $U(3)$ polyhedron can be tetrahedralized in $O(n \log n)$ time.

Finally we give a counterexample to show that a $U(4)$ polyhedron does not always

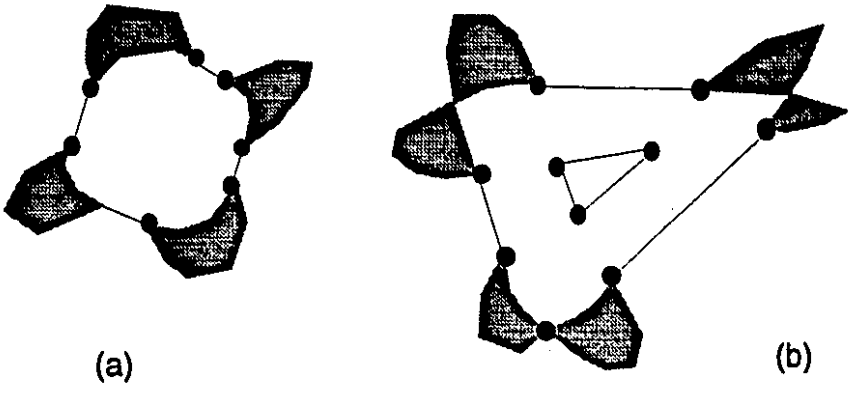


Figure 4.17: Illustration for the proof of Theorem 4.14.

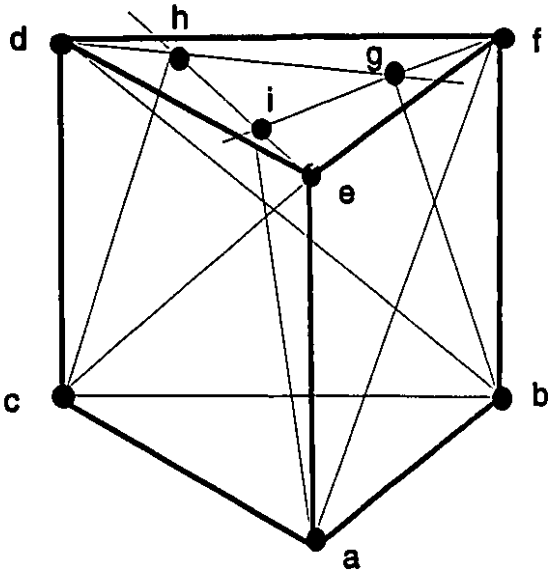


Figure 4.18: The Schoenhardt polyhedron can be decomposed into 4 disjoint convex polyhedra.

admit a tetrahedralization. The example is essentially based on the Schoenhardt polyhedron, which can be decomposed into four disjoint convex polyhedra: $P(c,e,d,h)$, $P(a,e,f,i)$, $P(b,d,g,f)$ and $P(a,b,c,g,h,i)$ (no two convex polyhedra intersect except on their surface) (see Figure 4.18).

The detail of this example is given as follows. The coordinates of the vertices on the bottom face are $a = (0, -\frac{2\sqrt{3}}{3}, 0)$, $b = (1, \frac{\sqrt{3}}{3}, 0)$ and $c = (-1, \frac{\sqrt{3}}{3}, 0)$. Initially we have a regular triangular prism such that the coordinates of the vertices on the top faces are $(0, -\frac{2\sqrt{3}}{3}, 2)$, $(1, \frac{\sqrt{3}}{3}, 2)$ and $(-1, \frac{\sqrt{3}}{3}, 2)$. The Schoenhardt polyhedron is obtained by rotating the upper face, which is a regular triangle, around its center by a small angle θ (see Figure 4.19). We have the coordinates of the vertices of the new face as follows:

$$d = (-\tfrac{2\sqrt{3}}{3}\cos(\tfrac{\pi}{6}-\theta), \tfrac{2\sqrt{3}}{3}\sin(\tfrac{\pi}{6}-\theta), 2),$$
$$e = (\tfrac{2\sqrt{3}}{3}\sin(\theta), -\tfrac{2\sqrt{3}}{3}\cos(\theta), 2) \text{ and}$$
$$f = (\tfrac{2\sqrt{3}}{3}\cos(\tfrac{\pi}{6}+\theta), \tfrac{2\sqrt{3}}{3}\sin(\tfrac{\pi}{6}+\theta), 2).$$

We extend the plane H_1 through a, c, e, the plane H_2 through a, b, f and the plane H_3 through b, c, d. We need to show that the common intersection of the half-pianes H_1^-, H_2^+ and H_3^-, together with plane $Z \geq 0$ and $Z \leq 2$, gives us a convex polyhedron $P(a,b,c,g,h,i)$. In fact we only need to show that the intersections of H_1^- with $Z = 2$, H_2^+ with $Z = 2$ and H_3^- with $Z = 2$ are not empty.

Suppose that H_1 intersects with $Z = 2$ at line $\overline{eh'}$ such that h' is a point on $\overline{df}$, H_2 intersects with $Z = 2$ at line $\overline{fi'}$ such that i' is a point on $\overline{de}$ and H_3 intersects with $Z = 2$ at line $\overline{dg'}$ such that g' is a point on $\overline{ef}$. We can show that the coordinate of h' is

$$\begin{aligned}
x = {} & [2\sqrt{3}\sin(\theta)(\cos(\tfrac{\pi}{6}+\theta)+\cos(\tfrac{\pi}{6}-\theta)) - 2\sin(\tfrac{\pi}{6}+\theta)\cos(\tfrac{\pi}{6}-\theta) \\
& -2\cos(\theta)(\cos(\tfrac{\pi}{6}+\theta)+\cos(\tfrac{\pi}{6}-\theta)) - 2\cos(\tfrac{\pi}{6}+\theta)\sin(\tfrac{\pi}{6}+\theta)] \\
& /[3\cos(\tfrac{\pi}{6}-\theta)+3\cos(\tfrac{\pi}{6}+\theta) - \sqrt{3}\sin(\tfrac{\pi}{6}-\theta)+\sqrt{3}\sin(\tfrac{\pi}{6}+\theta)], \\
y = {} & [2\sqrt{3}\sin(\theta)(\sin(\tfrac{\pi}{6}+\theta)-\sin(\tfrac{\pi}{6}-\theta)) - 2\sqrt{3}\sin(\tfrac{\pi}{6}+\theta)\cos(\tfrac{\pi}{6}-\theta) \\
& -2\cos(\theta)(\sin(\tfrac{\pi}{6}+\theta)-\sin(\tfrac{\pi}{6}-\theta)) + 2\sqrt{3}\cos(\tfrac{\pi}{6}+\theta)\sin(\tfrac{\pi}{6}-\theta)] \\
& /[3\cos(\tfrac{\pi}{6}-\theta)+3\cos(\tfrac{\pi}{6}+\theta) - \sqrt{3}\sin(\tfrac{\pi}{6}-\theta)+\sqrt{3}\sin(\tfrac{\pi}{6}+\theta)].
\end{aligned}$$

Furthermore when $\theta < \pi/6$, h' always lies to the left of the perpendicular bisector of $\overline{df}$. Symmetrically, this holds for g' and i'. Therefore when we rotate the

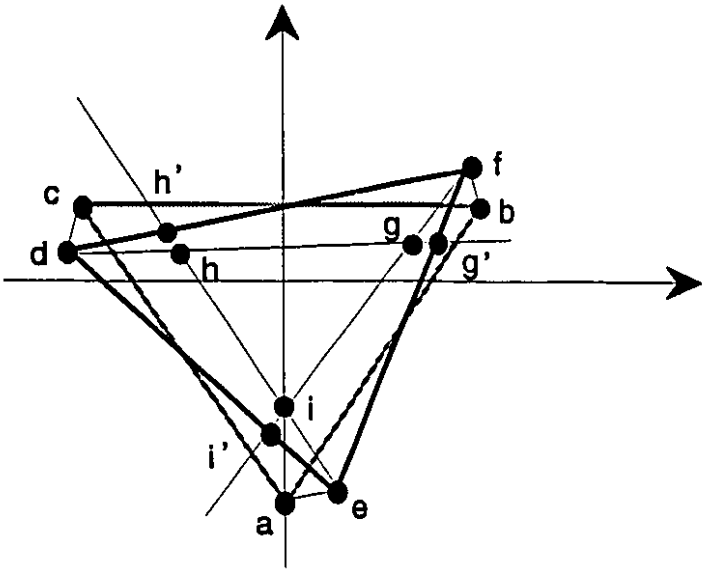


Figure 4.19: The Schoenhardt polyhedron can be decomposed into 4 disjoint convex polyhedra (top view).

upper face by an angle no greater than $\pi/6$ the intersection of H_1^- with $Z = 2$, H_2^+ with $Z = 2$ and H_3^- with $Z = 2$ is not empty; furthermore, this intersection defines exactly the convex polyhedron $P(a, b, c, g, h, i)$. Similarly we can show that $P(c, e, d, h), P(a, e, f, i)$ and $P(b, d, g, f)$ are the intersections of certain half-planes hence are all convex polyhedra (Figure 4.18).

We have thus shown that the Schoenhardt polyhedron is the union of four convex polyhedra. Therefore, we have the following theorem.

Theorem 4.15: A $U(4)$ polyhedron does not always admit a tetrahedralization.

4.4 Some remarks

We followed in the footsteps of Goodman and Pach [GP88] as well as Bern [Ber93] by showing in this chapter that certain special classes of simple and non-simple polyhedra can always be tetrahedralized. We also provided efficient algorithms for obtaining tetrahedralizations of polyhedra belonging to these classes. For the general problem

of deciding whether a solid terrain can be tetrahedralized, we are neither able to obtain a polynomial time algorithm, nor able to prove its NP-completeness. We list it as an open problem.

Open Problem 3: What is the complexity of deciding whether a solid terrain can be tetrahedralized?

Although we have presented a counterexample to show that a solid terrain can not always be tetrahedralized, we are not able to present such an example for the class of solid Manhattan terrains. We strongly believe the following conjecture is true:

4. Conjecture: All solid Manhattan terrains can be tetrahedralized without using Steiner points.

We show in this chapter that a $U(3)$ polyhedron can always be tetrahedralized on the condition that the three convex polyhedra are also given. It would be very interesting to know whether this latter condition can be removed or not.

Open Problem 5: Can a $U(3)$ polyhedron always be tetrahedralized?

Chapter 5

The shortest watchtower and related problems

5.1 Introduction

The problem of computing the shortest watchtower of a given terrain, i.e., a shortest vertical line segment erected on the terrain such that the top of the line segment can see every point on the surface (see Figure 5.1), was posed by deFloriani, Falcidieno and Nagy [Sha88]. Sharir gave an $O(n \log^2 n)$ algorithm for solving the problem [Sha88] (n is the number of vertices of the terrain). He also posed the problem of computing the shortest watchtower in $O(n \log n)$ time and made the conjecture that either the *fractional cascading* [CG86] or the *hierarchical representation* [DK85] technique might give us the solution.

In this chapter we show that we can solve the problem in $O(n \log n)$ time by storing additional information on Dobkin-Kirkpatrick's hierarchical representation of a convex polyhedron. In Section 5.2, we give the details of our algorithm. In Section 5.3, we discuss a variation of the shortest watchtower problem: computing the shortest vertical distance between two convex, non-intersecting terrains. In Section 5.4, we pose a set of closely related open problems for future research.

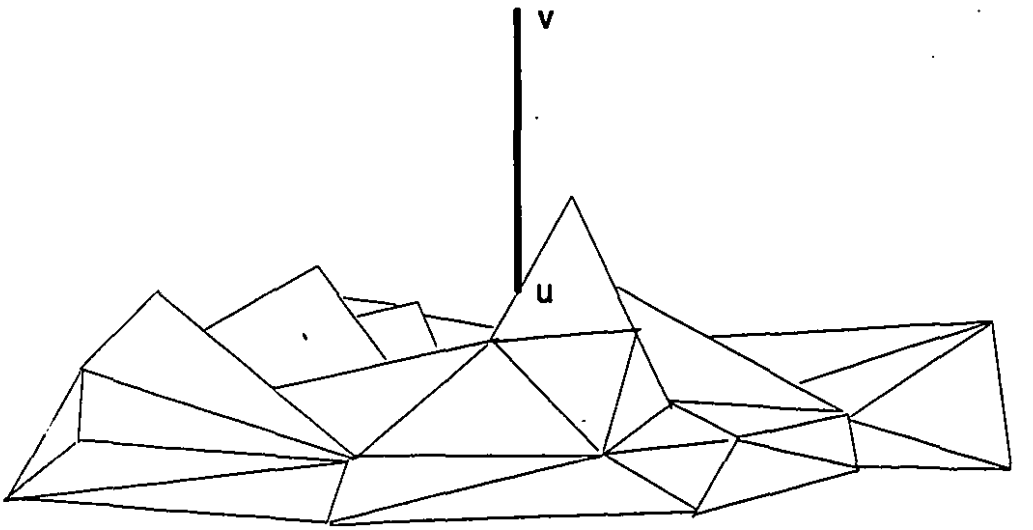


Figure 5.1: The shortest watchtower of a polyhedral terrain.

5.2 Computing the shortest watchtower of a terrain in $O(n \log n)$ time

5.2.1 Preliminary

We begin by making the following definition. Given two line segments s_1 and s_2 in 3D, if there is a vertical line l such that $s_1 \cap l \neq \emptyset$, $s_2 \cap l \neq \emptyset$, then the vertical distance between s_1 and s_2 (denoted by $d(s_1, s_2)$) is the difference between the Z-coordinates of $s_1 \cap l$ and $s_2 \cap l$. Otherwise the vertical distance between the two segments is defined to be infinity.

First we follow [Sha88] to reformulate the original problem. Let $f_1, ..., f_m$ be the planar faces of S, and let $\pi_1, ..., \pi_m$ be the planes containing these faces, a point v can see the entire surface of S if and only if it lies above every π_i (m is the number of faces of S and $m \leq 2n-4$). It turn out that the intersection of all halfspaces defined by π_i's is an unbounded convex polyhedron, and by using Muller and Preparata's algorithm [MP79], it can be computed in $O(n \log n)$ time (we denote it as L). Now the problem

is reduced to computing the shortest vertical distance between a polyhedral terrain S with $O(n)$ faces and another convex polyhedral terrain L with $O(n)$ faces lying above S. The shortest vertical line segment $\overline{uv}$, with $u \in S$, $v \in L$, must satisfy one of the following properties:

(1) v is a vertex of L;

(2) u is a vertex of S;

(3) u lies on an edge of S and v lies on an edge of L.

The first two cases can be done in a total of $O(n \log n)$ time by applying any $O(\log n)$ planar point location algorithm [Kir83, ST86]. The third case can be solved by using a nested binary search, which has a running time of $O(\log^2 n)$, to compute the shortest vertical distance between a line segment e and an arbitrary convex polyhedron P (assume that e and P does not intersect) [Sha88]. We improve this bound to $O(\log n)$, thus improving the overall bound to $O(n \log n)$.

Throughout this chapter, we use $P(q)$ to denote a point on P (q is the vertical projection of $P(q)$ on the plane $Z = 0$). We use e to denote a line (line segment) on the plane $Z = 0$ and its vertical projection on P is denoted by $P(e)$. (Similarly we use $e(q)$ to denote a point on a line e.) Assume P is an n-vertex convex polyhedron and its vertical projection is R, which is a bounded planar subdivision. Let $e = \overline{ab}$ be an edge of S, $e(t)$ be any point on e and $P(t)$ is the point (on the lower hull) of P lying directly above $e(t)$. Let

$$F_e^P(t) = d(P(t), e(t)).$$

We have the following observation:

Observation 5.1: $F_e^P(t)$ is a piecewise linear convex function.

5.2.2 The hierarchical representation of a convex polyhedron and its extension

Now we give a brief description of Dobkin-Kirkpatrick's hierarchical representation of a convex polyhedron [DK85]. Let P be a polyhedron in 3D with vertex set $V(P)$,

edge set $E(P)$ ($|V(P)|, |E(P)| \in O(n)$). A sequence of polyhedra, $H(P) = P_1, ..., P_k$, is said to be a *hierarchical representation* of P if

(i) $P_1 = P$ and P_k is a 3-simplex (i.e., a convex polyhedron whose size is constant);

(ii) $P_{i+1} \subset P_i$, for $1 \leq i < k$;

(iii) $V(P_{i+1}) \subset V(P_i)$; and

(iv) the vertices of $V(P_i) - V(P_{i+1})$ form an independent set (i.e., are non-adjacent) in P_i.

Furthermore, as shown in [DK85], there exists a constant $c = 11$ such that for a convex polyhedron P in 3D there exists a hierarchical representation of degree at most c, $O(\log n)$ height, and $O(n)$ size and such a hierarchical representation can be constructed in $O(n)$ time. We show briefly in the following paragraphs the procedure to construct such a hierarchical representation. Further details can be found in [Kir83, DK85].

Suppose the faces of P have already be triangulated. Since the surface of P can be represented as a planar graph, P has at most $3n-6$ edges (following Euler's formula). The idea is as follows, each time we delete a constant factor of the number of vertices in the current convex polyhedron to obtain a *coarser* one. We obtain $k = O(\log n)$ convex polyhedra $(P_0, P_1, ..., P_k)$, such that,

(1) P_0 is the given convex polyhedron P.

(2) P_{i+1} is obtained from P_i such that the number of vertices in P_{i+1} is at most a constant fraction of that in P_i (we show later that this constant is 23/24).

(3) Each face of P_{i+1} intersects at most a constant c of faces in P_i.

We use the *independent set* idea to achieve our goal. A set of vertices is independent if no two vertices of the set are adjacent. We show how to find a suitable independent set of P as follows (one which has at least $n/24$ vertices). Since P has at most $3n - 6$ edges the sum of all the vertex degrees is at most $6n - 12$. Then the average vertex degree has to be less than 6. By the pigeon-hole principle, there are at least $n/2$ vertices whose degree is no more than 11. We lump these vertices into a set called I_{11}. We pick a vertex v_1 in the set I_{11}, by the definition of independent set, all of its neighbors (which is at most 11) can not be in the independent set. This means that for every 12 vertices in I_{11}, we can obtain at least 1 vertex that is not

adjacent to another vertex in I_{11}. Therefore, we can find an independent set I with size $1/12$ that of I_{11} or at least $n/24$ in size.

By eliminating the $n/24$ vertices of I from the current convex polyhedron (and all the edges incident at those vertices) and computing the convex hulls of the "holes" resulting from the deletion of these independent vertices, we can reduce the total number of vertices by a constant factor of $1/24$ to have the next *coarser* convex polyhedron. The independent set of $n/24$ vertices from P can be found in $O(n)$ time; furthermore, computing the convex hull of the "holes" resulting from the deletion of the independent vertices takes $O(n)$ time (since each convex hull has a constant number of faces).

The total time and space complexity of constructing a hierarchical representation of P is therefore:

$$O\left(n + \tfrac{23}{24}n + \left(\tfrac{23}{24}\right)^2 n + \left(\tfrac{23}{24}\right)^3 n + ...\right) = O\left(\tfrac{n}{1-\frac{23}{24}}\right) = O(24n) = O(n).$$

Because each time we extract a constant factor of vertices out of the previous convex polyhedron, the number of levels of this hierarchical structure is $O(\log n)$. From this construction we have the following observation.

Observation 5.2: There are at most 11 edges of P_i intersecting any supporting plane of P_{i+1}.

Suppose we already have the (reverse) hierarchical representation $P_k, ..., P_{i+1}$, $P_i, ..., P_1$ ($P_1 = P$) for P. The shortest vertical distance between P_i and an arbitrary line segment $e = \overline{ab}$ (denoted by $d_{min}(\overline{ab}, P_i)$) is defined as the minimum vertical distance between $\overline{ab}$ and $\overline{rs}$ over all $\overline{rs} \in P_i$.

Suppose P_{i+1} and an edge $\overline{xy}$ of P_{i+1} are given such that $d(\overline{ab}, \overline{xy})$ is a breakpoint (i.e., a vertex) of the distance function $F_e^{P_{i+1}}$, then two neighboring breakpoints of $d(\overline{ab}, \overline{xy})$ must be $d(e_1, \overline{ab})$ and $d(e_2, \overline{ab})$ where e_1 is an edge of one triangular face along $\overline{xy}$ and e_2 is an edge of the other triangular face along $\overline{xy}$. In Figure 5.2, $e_1 = \overline{xw}, e_2 = \overline{yz}$. We say that such a pair (e_1, e_2) is a *local pair* of $\overline{xy}$. We first show the following lemma.

Lemma 5.3: Assume $(\overline{xw}, \overline{yz})$ is a local pair of $\overline{xy}$ in P_{i+1}. Then it is not a local

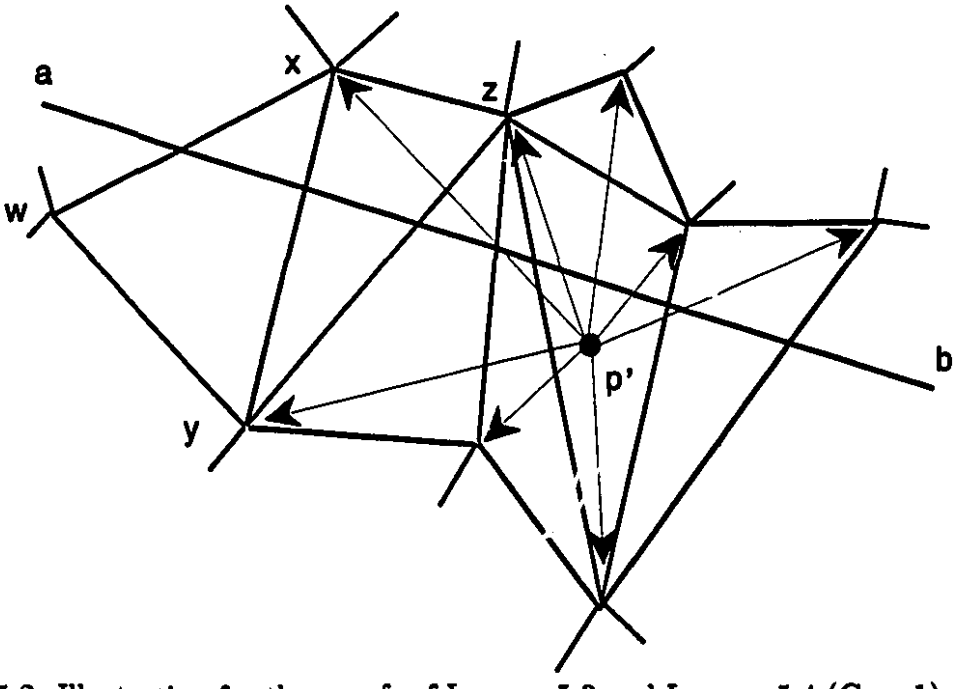


Figure 5.2: Illustration for the proofs of Lemma 5.3 and Lemma 5.4 (Case 1).

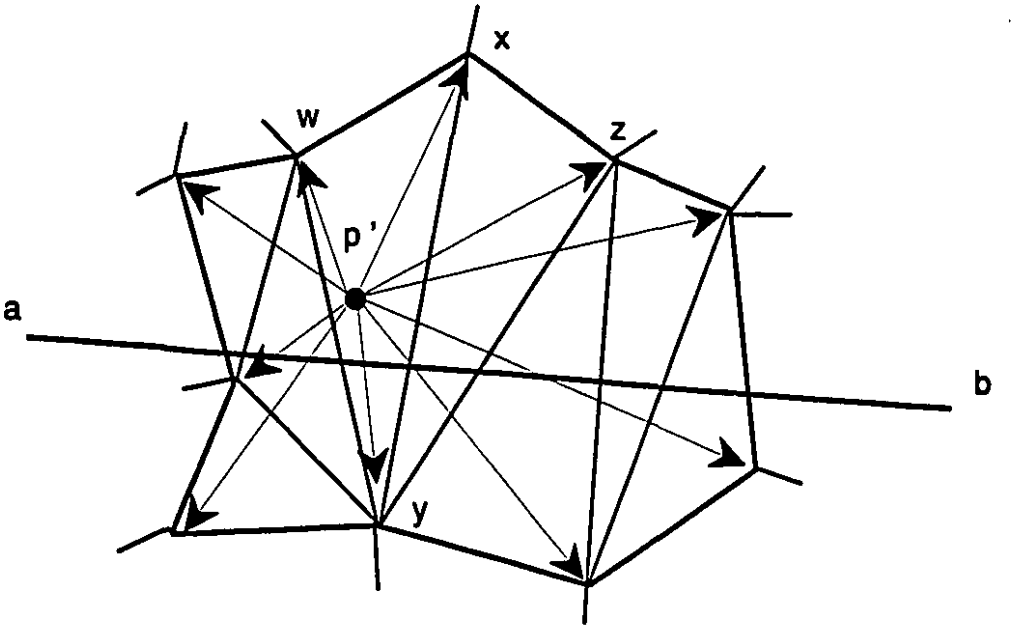


Figure 5.3: Illustration for the proofs of Lemma 5.3 and Lemma 5.4 (Case 2).

pair of $\overline{xy}$ in P_i if and only if there is a $p' \in V(P_i) - V(P_{i+1})$ such that both $\overline{p'x}$ and $\overline{p'y}$ belong to $E(P_i)$.

Proof: When we construct the hierarchical representation of P_{i+1} from P_i we keep deleting independent vertices with degree at most 11. The deletion of such a vertex r gives us a *hole* and we then compute the convex hull of the hole (actually a part of the convex hull which conforms to the convexity of P_i, we will call it *lower hull* of r and denote it by $H(r)$ henceforth). Now we are actually reversing this order.

If there is a $p' \in V(P_i) - V(P_{i+1})$ such that both $\overline{p'x}$ and $\overline{p'y}$ belong to $E(P_i)$ then we have the following two cases (without loss of generality we refer to Figure 5.2 and suppose we only add p' back to P_{i+1}):

(1) $\overline{xy}$ is a boundary edge of $H(p')$.

(2) $\overline{xy}$ is not a boundary edge of $H(p')$.

In the first case z has to be on the boundary as well, therefore the face Δxyz can not be a face of P_i. This implies that neither $\overline{xz}$ nor $\overline{yz}$ can be a neighboring breakpoint

of $\overline{xy}$ for the distance function between e and P_i. In the second case if we add p' to P_{i+1} then both of the faces Δxyz and Δxyw do not belong to P_i. Hence $\overline{xy}$ is not an edge of P_i and the result trivially holds (see Figure 5.3).

If there is no such $p' \in V(P_i) - V(P_{i+1})$ such that both $\overline{p'x}$ and $\overline{p'y}$ belong to $E(P_i)$ then both of the faces Δxyz and Δxyw are faces of P_i. Hence $< \overline{xw}, \overline{yz} >$ remains a local pair of $\overline{xy}$ in P_i. □

With this result we proceed to prove the following lemma.

Lemma 5.4: Assume the function $F_e^{P_{i+1}}(t)$ achieves its minimum at edge $\overline{xy}$ of P_{i+1}, then $d_{min}(\overline{ab}, P_i)$ is either equal to $d_{min}(\overline{ab}, P_{i+1})$ or the shortest vertical distance between $\overline{ab}$ and $\overline{pq}$ such that $p \in V(P_i) - V(P_{i+1})$, $q \in V(P_{i+1})$, $\overline{pq} \in E(P_i) - E(P_{i+1})$ and $\overline{px}, \overline{py} \in E(P_i) - E(P_{i+1})$, if such p, q exist.

Proof: Note that we have two important properties here:

(1) the distance function $F_e^{P_{i+1}}(t)$ is convex, and

(2) if we add a vertex $p' \in V(P_i) - V(P_{i+1})$ (along with its adjacent edges and faces) to P_{i+1}, the resulting polyhedron is still convex. (By adding all these vertices we finally obtain P_i.)

We prove Lemma 5.4 by contradiction. It is obvious that if $d_{min}(\overline{ab}, P_i)$ is not equal to $d_{min}(\overline{ab}, P_{i+1})$ then $d_{min}(\overline{ab}, P_i) < d_{min}(\overline{ab}, P_{i+1})$ and $d_{min}(\overline{ab}, P_i)$ must be the vertical distance between $\overline{ab}$ and a line segment $\overline{pq}$ such that $p \in V(P_i) - V(P_{i+1})$, $q \in V(P_{i+1})$ and $\overline{pq} \in E(P_i) - E(P_{i+1})$. The crucial point is that if such p, q exist then both $\overline{px}$ and $\overline{py}$ belong to $E(P_i) - E(P_{i+1})$.

Suppose our claim is false, i.e., $d_{min}(\overline{ab}, P_i) < d_{min}(\overline{ab}, P_{i+1})$ and it achieves the minimum at $\overline{p'q'} \in E(P_i) - E(P_{i+1})$ such that $p' \in V(P_i) - V(P_{i+1})$, $q' \in V(P_{i+1})$ and at least one of $\overline{p'x}$ and $\overline{p'y} \notin E(P_i) - E(P_{i+1})$.

Assume we add only p' (and the corresponding edges adjacent to it) to P_{i+1}. According to the above discussion the resulting polyhedron P' is still convex. Since we only add p' to P_{i+1} and at most one of $\overline{p'x}$ and $\overline{p'y}$ belongs to $E(P_i) - E(P_{i+1})$, at most one of $\overline{p'x}$ and $\overline{p'y}$ belongs to $E(P')$. By Lemma 5.3 the two faces Δxyz and Δxyw along the edge $\overline{xy}$ are faces of P_i (hence are faces of P') (Figure 5.3). Now let us consider the distance function between $e = \overline{ab}$ and P'. It has two local

minima: $d(\overline{ab}, \overline{xy})$ (this is equal to $d_{min}(\overline{ab}, P_{i+1})$, since Δxyz and Δxyw are faces of P', $d(\overline{ab}, \overline{xy})$ is a local minimum), and $d(\overline{ab}, \overline{p'q'})$ (this is the global minimum by assumption). However this contradicts Observation 5.1. □

Lemma 5.4 enables us to check only a small number of edges of P_i when computing $d_{min}(\overline{ab}, P_i)$. We show in the following lemma that this number is actually a constant.

Lemma 5.5: If $d_{min}(\overline{ab}, P_{i+1})$ is known, then to compute $d_{min}(\overline{ab}, P_i)$ we need only check at most 22 edges in $E(P_i) - E(P_{i+1})$ in constant time.

Proof: Following Lemma 5.4, in order to compute $d_{min}(\overline{ab}, P_i)$ we need to find all $p \in V(P_i) - V(P_{i+1})$ such that both $\overline{px}$ and $\overline{py}$ belong to $E(P_i) - E(P_{i+1})$. It turns out that there are at most two such p's. We have two cases: (1) $\overline{xy} \in P_i$ and (2) $\overline{xy} \notin P_i$.

In Case (1) we might have such a p' to the right of the line $\overline{xy}$ and a p'' to the left of the line $\overline{xy}$ such that all edges incident upon p' in $E(P_i)$ intersect the supporting plane Δxyz and all the edges incident upon p'' in $E(P_i)$ intersect the supporting plane Δxyw. By Observation 5.2, there are at most 11 edges incident to p' (p''). Therefore in total we have at most 22 candidate edges. To compute $d_{min}(\overline{ab}, P_i)$ we simply compute the minimum vertical distance between $\overline{ab}$ and these 22 edges and choose the smaller of this minimum and $d_{min}(\overline{ab}, P_{i+1})$. This gives us $d_{min}(\overline{ab}, P_i)$. It is clear that $d_{min}(\overline{ab}, P_i)$ is computed in $O(1)$ time.

In Case (2) we have fewer candidates, since there is only one such p. All of the edges incident to p (including $\overline{px}, \overline{py}$) belong to $E(P_i)$ and they all intersect with any supporting plane through $\overline{xy}$. Again by Observation 5.2, there are at most 11 candidate edges. Consequently we can compute $d_{min}(\overline{ab}, P_i)$ by computing the minimum vertical distance between $\overline{ab}$ and these 11 edges in $O(1)$ time.

Although we know that at most two such p's exist from the above discussion, in order to design an efficient algorithm, we need to find out $p \in V(P_i) - V(P_{i+1})$ in constant time. We can clarify this by storing additional information in the hierarchical representation of P without increasing the overall time and space bound when computing the hierarchical representation of P. In the process of deleting $r \in V(P_i)$ to obtain P_{i+1}, we compute the $H(r)$. For each edge of $H(r)$ we assign a parent node

(r, i) to it. In this process, no edge can have more than two level-i parent nodes (a boundary edge of the hull can have two such parent node since it can be a boundary edge of two lower hulls). Therefore given $\overline{xy} \in E(P_{i+1})$ we can retrieve its level-i parent nodes in $V(P_i) - V(P_{i+1})$ in $O(1)$ time. From these parent nodes (at most 2), we can list the edges (at most 22) incident to them in $E(P_i)$ in constant time. □

Lemma 5.4 and Lemma 5.5 enable us to design an algorithm for an arbitrary edge e and a convex polyhedron P such that the algorithm returns the shortest vertical distance between e and P in $O(\log n)$ time. We first give a revised algorithm for computing the hierarchical representation of a convex polyhedron by storing i-level parents for all edges in P_{i+1}.

Algorithm *Revised-Hierarchical-Representation(P)*;
BEGIN
FOR $i = 1$ to k **UNTIL** $|P_{k+1}| = 3$ **DO**
(1) Compute the hierarchical representation from P_i to P_{i+1} using the *independent vertex deletion* of Kirkpatrick [Kir83] and Dobkin-Kirkpatrick [DK85].
(2) For each vertex r being deleted in $V(P_i)$, compute $H(r)$.
(3) Associate r with every edge of $H(r)$, these are the level-i parent vertex for these edges in $E(P_{i+1})$.
END

Suppose we already have a revised version of the hierarchical representation of a convex polyhedron P. The following algorithm computes the shortest vertical distance between P and an arbitrary line segment (or line) e in $O(\log n)$ time. For simplicity we assume that e does not intersect with P and the vertical distance between e and P is not infinity at this stage. It is known that whether a line or a line segment intersects with a convex polyhedron P or not can be detected in $O(\log n)$ time [DK90]. And we can test whether the vertical distance between e and P is infinity in $O(\log n)$ time by detecting the intersection of the planar projection of P, which is a convex polygon, with the planar projection of e. In fact we can generalize the definition of the distance between two line segments in 3D such that the following algorithm can also detect the intersection of e and P, i.e., we do not have to make the assumption that P is

above e. The generalization is given after the following algorithm.

Algorithm *Shortest-Vertical-Distance(e, P)*;
BEGIN
(1) Set $i = k$.
(2) Compute $d_{min}(e, P_{i+1})$, suppose it is equal to $d(e, \overline{xy})$ for some $\overline{xy} \in P_{i+1}$.
(3) List all the edges incident to the level-i parent nodes of $\overline{xy}$. These are the candidate edges of P_i for computing $d_{min}(e, P_i)$. Compute the minimum vertical distance between e and all of these edges. Compare it with $d_{min}(e, P_{i+1}) = d(e, \overline{xy})$, the smaller of the two is $d_{min}(e, P_i)$. Record the new $\overline{xy}$.
(4) Continue (3) until $d_{min}(e, P_1)$ is obtained, output $d_{min}(e, P_1)$.
END

The correctness of the above algorithm directly follows from Lemma 5.4 and Lemma 5.5. We now show how to generalize the distance between two line segments in 3D and how to modify the above algorithm to make it work for the case when e and P intersect. Given two line segments s and e in 3D, if there is a vertical line l such that $s \cap l$ is higher than $e \cap l \neq \emptyset$, then the signed vertical distance between s and e is defined as $D(s, e) = d(s, e)$, otherwise $D(s, e) = -d(s, e)$. With this new definition, even if e and P intersects Observation 5.1 still holds (without loss of generality we only consider the case when e intersects with the lower convex hull of P or it lies below the lower convex hull of P). With Observation 5.1, we can use Lemmas 5.4 and 5.5 to obtain a witness of the intersection of e and P. If the shortest vertical distance between e and P_{i+1} is positive and at the next step the shortest vertical distance between e and P_i becomes non-positive, then we stop and report the point of P_i (which achieves the shortest vertical distance between e and P_i) as a witness of the intersection of e and P. Consequently we have the following theorem.

Theorem 5.6: After an $O(n)$ time and space preprocessing on P, we can either report the intersection of P with a line segment (or a line) e by giving a witness, or report that the vertical distance between e and P is infinity, or compute the shortest vertical distance between e and P in $O(\log n)$ time.

We can further generalize the above algorithm to compute the shortest distance between a convex polyhedron and a line segment (or a line) along a given direction $\vec{\mathbf{d}} =< \alpha, \beta, \gamma >$ where $\alpha^2 + \beta^2 + \gamma^2 = 1$. In this sense the shortest vertical distance is the special case when $\vec{\mathbf{d}} =< 0, 0, 1 >$. We need only perform an orthogonal transformation to obtain a new coordinate system $< X', Y', Z' >$ such that in the new system $\vec{\mathbf{d}}' =< 0, 0, 1 >$. This is done as follows, we have

$$A\vec{\mathbf{x}} = \vec{\mathbf{x}}',$$

and we know that $\vec{\mathbf{x}}_1 =< 0, 0, 0 >$, $\vec{\mathbf{x}}_1' =< 0, 0, 0 >$ and $\vec{\mathbf{x}}_2 =< \alpha, \beta, \gamma >$, $\vec{\mathbf{x}}_2' =< 0, 0, 1 >$ satisfy the above equation. But since A is a 3×3 orthogonal matrix containing nine variables, with these information we have an infinite number of such orthogonal matrices A. To simplify things we set $\vec{\mathbf{e}} =< -\beta, \alpha, 0 >$, which is perpendicular to $\vec{\mathbf{d}}$, as the unit vertor along X'. Then $\vec{\mathbf{x}}_3 =< -\beta, \alpha, 0 >$, $\vec{\mathbf{x}}_3' =< 1, 0, 0 >$ also satisfy the above equation. With this information, we have a unique orthogonal matrix A.

Now we obtain a representation of the convex polyhedron P and the line segment e in the new coordinate system by performing the orthogonal transformation A. We represent them as P_A and e_A respectively in the new system. Then we run the *Shortest-Vertical-Distance* algorithm to compute the shortest vertical distance between P_A and e_A. This gives us the shortest distance of P and e along the direction $\vec{\mathbf{d}} =< \alpha, \beta, \gamma >$. Therefore we have the following theorem.

Theorem 5.7: After $O(n)$ time and space preprocessing on P, we can either report the intersection of P with a line segment (or a line) e by giving a witness, or report that the distance between e and P along a given direction $\vec{\mathbf{d}} =< \alpha, \beta, \gamma >$ is infinity, or compute the shortest distance between e and P along $\vec{\mathbf{d}}$ in $O(\log n)$ time.

Finally we give an $O(n \log n)$ time algorithm for solving the shortest watchtower problem.

Algorithm *Shortest-Watchtower(S);*
BEGIN

(1) Compute the intersection of the halfspaces defined by each face of the terrain by using Muller and Preparata's algorithm [MP79]. This gives us the unbounded convex polyhedron L.

(2) Compute the revised hierarchical representation of L.

(3) For every vertex v of S, perform a point location to compute all the vertical distances $\overline{uv}$ such that v is a vertex of S. Compute the minimum (with the location) among all of these distances.

(4) For every vertex u of L, perform a point location to compute all the vertical distances $\overline{uv}$ such that u is a vertex of L. Compute the minimum (with the location) among all of these distances.

(5) For every edge e in S, run *Compute-Shortest-Distance(e,L)*; compute the minimum (with the location) among all of these distances.

(6) Among the three minima obtained in Steps (3), (4) and (5) choose the minimum (with the corresponding location). Output it as the shortest watchtower of S.

END

Therefore we have the main result of this chapter.

Theorem 5.8: Algorithm *Shortest-Watchtower(S)* computes the shortest watchtower of a polyhedral terrain S with n vertices in $O(n \log n)$ time.

5.3 Computing the shortest vertical distance between two convex terrains

A variation of the shortest watchtower problem, pointed out by Sharir, is to compute the shortest vertical distance between two arbitrary, non-intersecting polyhedral terrains. Using a technique called *generalized point location*, Chazelle and Sharir obtained an $O(n^{1.999878})$ algorithm for the problem, which beat the trivial $O(n^2)$ time bound [CS90]. Chazelle et al. [CEGS89] also gave a randomized algorithm with time

$O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$) for solving this problem. It is very interesting whether we can obtain a faster, deterministic algorithm.

The second problem we are interested in is the problem of computing the shortest vertical vertical distance between two non-intersecting convex polyhedra (or two convex terrains which are convex in opposite direction). We show that if the *revised hierarchical representation* of the two convex polyhedra is known, we can compute the shortest vertical distance between the two convex polyhedra in $O(\log^2 n)$ time. By generalizing the distance function, we can detect the intersection of two convex polyhedra in $O(\log^2 n)$ time (after $O(n)$ time and space preprocessing). This achieves the same bound as Cole's *similar list* method [Col86], and Dobkin and Kirkpatrick's *hierarchical representation* method [DK85, DK90].

The underlying idea of our method is to apply the revised hierarchical representation of a convex polyhedron. It is known that the vertical distance function between two convex polyhedra is convex [Roc70]. With this property we show that when the shortest vertical distance between P_i and Q_i is known, we need only check the shortest vertical distance between at most 22 edges $\in E(P_{i-1}) - E(P_i)$ and Q_i, which can be done in $O(\log n)$ time by Algorithm *Shortest-Vertical-Distance(e, P)*. Consequently, we have shown that the shortest vertical distance between two preprocessed convex polyhedra P, Q can be computed in $O(\log^2 n)$ time. By generalizing the definition of the vertical distance function, we also solve the version when P, Q intersect.

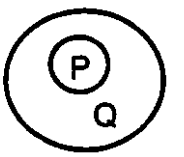
We first list a topological relationship between two given convex polyhedra P, Q in 3D (Figure 5.4). Cases (1) and (2) can be detected easily and are thus ignored. Without loss of generality, we only consider case (3) (hence (4) and (5)). For simplicity, we only consider the lower hull of P and the upper hull of Q and still use P and Q to represent them henceforth.

We first follow Section 5.1 and define certain distance functions between a convex polyhedron and a given convex polyhedron. Let $P(t)$ be any point on the surface of P and $Q(t)$ be the (vertical) projection of $P(t)$ on the surface Q. We define

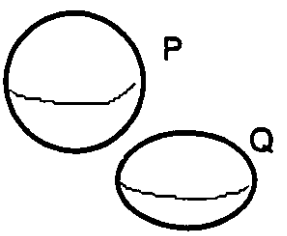
$$F_Q^P(t) = d(P(t), Q(t)),$$

as the distance function between P and Q. By a well-known result in convex analysis

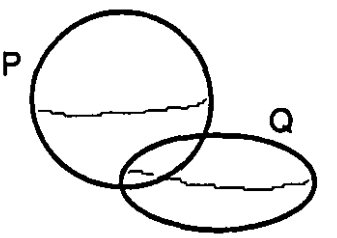
Case (1)



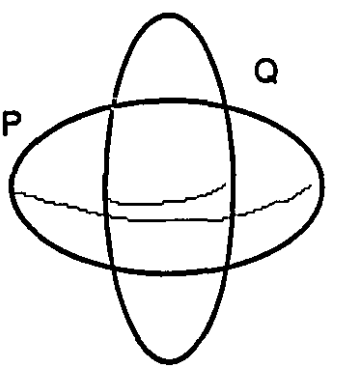
Case (2)



Case (3) The lower hull of P is above the upper hull of Q.



Case (4) The lower hull of P intersects the upper hull of Q.



Case (5) The lower hull of P is below the upper hull of Q.

Figure 5.4: The topological relationship between two convex polyhedra.

[Roc70] we have the following lemma.

Lemma 5.9 [Roc70]: $F_Q^P(t)$ is a piecewise linear convex function over t.

Suppose we already have the (reverse) hierarchical representation $P_k, ..., P_{i+1}$, $P_i, ..., P_1$ ($P_1 = P$) for P and the (reverse) hierarchical representation $Q_m, ..., Q_{j+1}$, $Q_j, ..., Q_1$ ($Q_1 = Q$) for Q. Without loss of generality we also assume that $k \leq m$. The minimum vertical distance between P_i and Q_j (denoted by $d(P_i, Q_j)$) is defined as the minimum vertical distance between

(1) $\overline{ab} \in P_i$ and $\overline{rs} \in Q_j$ over all $\overline{ab} \in P_i$ and $\overline{rs} \in Q_j$; or

(2) a vertex $v_i \in P_i$ and a face $f_j \in Q_j$; or

(3) a vertex $v_j \in Q_j$ and a face $f_i \in P_i$.

The idea of our algorithm is to essentially start with $i = k$ and compute the shortest vertical distance between P_i and Q_i. Then we reach the step of P_{i-1} and Q_{i-1} in two substeps: $< P_i, Q_i > \Rightarrow < P_{i-1}, Q_i > \Rightarrow < P_{i-1}, Q_{i-1} >$ (i.e., alternatively decreasing the indices P_i and Q_i). We show that each of these two substeps takes $O(\log n)$ time. Without loss of generality, we only give a detailed description of the first substep. Given P_i and Q_i, suppose $d(\overline{ab}, \overline{xy})$ is the shortest distance between P_i and Q_i such that $\overline{ab} \in P_i$ and $\overline{xy} \in Q_i$. Furthermore, assume $P_i(o) \in \overline{ab}$ and $Q_i(o) \in \overline{xy}$ realize the shortest vertical distance between P_i and Q_i, the two vertices adjacent to $\overline{ab}$ is c, d and the two vertices adjacent to $\overline{xy}$ is z, w.

Lemma 5.10: Suppose $d(\overline{ab}, \overline{xy})$ is the shortest vertical distance between P_i and Q_i such that $\overline{ab} \in P_i$ and $\overline{xy} \in Q_j$. Then the shortest vertical distance between P_{i-1} and Q_i is either equal to $d(\overline{ab}, \overline{xy})$ or the shortest vertical distance between Q_i and $\overline{pq} \in E(P_{i-1}) - E(P_i)$ such that $p \in V(P_{i-1}) - V(P_i)$, $q \in V(P_i)$, and $\overline{pa}, \overline{pb} \in E(P_{i-1}) - E(P_i)$, if such p, q exist.

Proof: Note that we have two important properties here:

(1) the distance function $F_{Q_i}^{P_i}(t)$ is convex; and

(2) if we add a vertex $p' \in V(P_{i-1}) - V(P_i)$ (along with its adjacent edges and faces) to P_i, the resulting polyhedron is still convex. (By adding all these vertices we finally obtain P_{i-1}.)

Now we prove Lemma 5.10 by contradiction.

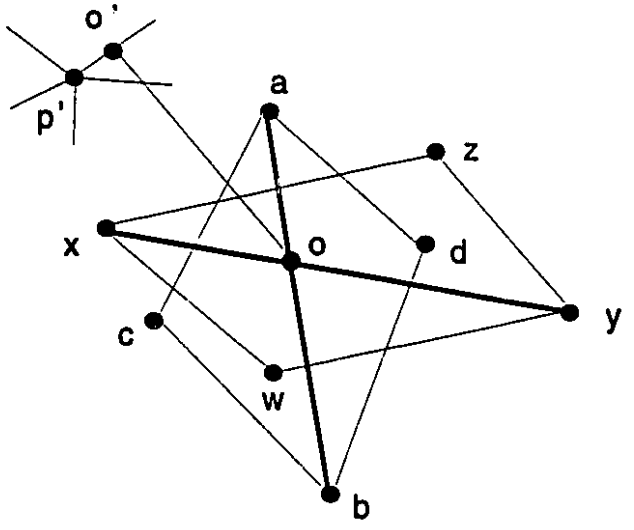


Figure 5.5: Illustration for the proof of Lemma 5.10.

It is obvious that if $d(P_{i-1}, Q_i)$ is not equal to $d(P_i, Q_i)$ then $d(P_{i-1}, Q_i) < d(P_i, Q_i)$ and $d(P_{i-1}, Q_i)$ must be the vertical distance between an edge of Q_i and an edge $\overline{pq} \in E(P_{i-1}) - E(P_i)$ such that $p \in V(P_{i-1}) - V(P_i)$, $q \in V(P_i)$. The crucial point is that if such p, q exist then both $\overline{pa}$ and $\overline{pb}$ belong to $E(P_{i-1}) - E(P_i)$.

Suppose our claim is false, i.e., $d(P_{i-1}, Q_i) < d(P_i, Q_i)$ and it achieves the minimum at $\overline{p'q'} \in E(P_{i-1}) - E(P_i)$ such that $p' \in V(P_{i-1}) - V(P_i)$, $q' \in V(P_i)$ and at least one of $\overline{p'a}$ and $\overline{p'b} \notin E(P_{i-1}) - E(P_i)$.

Assume we add only p' (and the corresponding edges adjacent to it) to P_i. According to the above discussion the resulting polyhedron P' is still convex. Since we only add p' to P_i and at most one of $\overline{p'a}$ and $\overline{p'b}$ belongs to $E(P_{i-1}) - E(P_i)$, at most one of $\overline{p'a}$ and $\overline{p'b}$ belongs to $E(P')$. Because of that the faces of P_i incident to $\overline{ab}$, $\triangle abd$ and $\triangle abc$, are faces of P' (see Figure 5.5). Assume $P'(o') \in P'$ and $Q_i(o') \in Q_i$ realize the shortest vertical distance between P' and Q_i.

Now let us consider the distance function between $P'(\overline{oo'})$ and Q_i. It has two local minima: $d(\overline{ab}, \overline{xy})$ (this is equal to $d(P_i, Q_i)$, since $\triangle abc$ and $\triangle abd$ are faces of P', $d(\overline{ab}, \overline{xy})$ is a local minimum), and $d(P'(o'), Q_i(o'))$ (this is the global minimum by assumption). But this contradicts the fact that the shortest vertical distance function

between two convex polyhedra is always convex. □

Lemma 5.11: Given $d(P_i, Q_i)$, to compute $d(P_{i-1}, Q_i)$ it is sufficient to check the shortest vertical distance between at most 22 edges in $E(P_{i-1}) - E(P_i)$ and Q_i. Furthermore, this can be done in $O(\log n)$ time.

Proof: In Lemma 5.10 we clarify the (relatively hard) case when $d(P_i, Q_i)$ is realized by a pair of edges of P_i and Q_i. If $d(P_i, Q_i)$ is realized by a vertex $v_i \in P_i$ and a face $f_i \in Q_i$, then in $O(\log n)$ time we can find the polyhedron (with at most 11 edges of $E(P_{i-1}) - E(P_i)$) which is below f_i^*, where f_i^* is the supporting plane of P_i which is parallel to f_i [DK90]. Thus $d(P_{i-1}, Q_i)$ is equal to either $d(v_i, f_i)$ or the shortest vertical distance between the 11 edges of $E(P_{i-1}) - E(P_i)$ below f_i^* (if exist) and Q_i. The case when $d(P_i, Q_i)$ is realized by a vertex $v_i \in Q_i$ and a face $f_i \in P_i$ can be clarified symmetrically. Following Lemma 5.5 and Theorem 5.6, $O(\log n)$ time suffices to compute $d(P_{i-1}, Q_i)$. □

Lemmas 5.10 and 5.11 enable us to design an algorithm for computing the shortest vertical distance between two preprocessed convex polyhedra P and Q in $O(\log^2 n)$ time. We simply start from P_k and Q_k until we reach P_1 and Q_1, using $O(\log n)$ steps, each taking $O(\log n)$ time. Similar to what we have done in Section 5.2, we can adopt the generalized vertical distance function (which is also convex) and apply Lemmas 5.10 and 5.11. Therefore we can also deal with the case when P, Q intersects. Consequently we have the following theorem.

Theorem 5.12: After $O(n)$ time and space preprocessing on P, Q, one can either report the intersection of P with Q by giving a witness, or report that the vertical distance between P and Q is infinity, or report the shortest vertical distance between P and Q in $O(\log^2 n)$ time.

We can generalize the above algorithm further to compute the shortest distance between two convex polyhedra along a given direction. The details, being identical to those of Theorem 5.7, are therefore omitted.

Corollary 5.13: After an $O(n)$ time and space preprocessing on P, Q, one can either report the intersection of P with Q by giving a witness, or report that the distance

between P and Q along a given direction $\vec{d} = < \alpha, \beta, \gamma >$ is infinity, or report the shortest distance between P and Q along $\vec{d}$ in $O(\log^2 n)$ time. □

5.4 Some remarks

Although we have successfully solved the shortest watchtower problem in $O(n \log n)$ time, there are nonetheless some related problems remaining.

Open Problem 6: What is the lower bound for computing the shortest watchtower of a polyhedral terrain? Does the information that L is a special convex polyhedron help improving the $O(n \log n)$ upper bound? We strongly believe that $\Omega(n \log n)$ is the lower bound since in our algorithm there are two steps with an $\Omega(n \log n)$ lower bound (i.e., computing the intersection of n half spaces, locating n points in a planar triangulation). Proving the $\Omega(n \log n)$ lower bound or giving an $o(n \log n)$ time algorithm remains to be an open problem. On the other hand, another related problem of computing the *lowest watchtower* of a polyhedral terrain, i.e., a line segment erected on the terrain such that the top of the watchtower can see the whole terrain and the Z-coordinate of the top is minimized, can be formulated as a linear programming problem and can thus be solved in linear time.

Open Problem 7: For the problem of computing the shortest vertical distance between two non-intersecting terrains, is it possible to obtain a faster, deterministic algorithm?

Open Problem 8: For the problem of intersection detection between two convex polyhedra, is it possible to obtain an $o(\log^2 n)$ query bound (even at the cost of increasing preprocessing time and space)? This is one of the fundamental problems in intersection detection. It is very interesting that no $o(\log^2 n)$ query bound has been achieved although three different methods have been tried. Is $\Omega(\log^2 n)$ a lower bound for this problem?

Chapter 6

Guarding polyhedral terrains

Victor Klee posed the problem of determining the minimum number of guards sufficient to cover the interior of an n-sided art gallery (polygon) in 1973. Chvátal showed that $\lfloor \frac{n}{3} \rfloor$ guards are sufficient and sometimes necessary to cover the interior of an n-sided art gallery using a lengthy combinatorial argument [Chv75]. Subsequently Fisk [Fis78] gave a concise and elegant proof using the fact that the vertices of a triangulated polygon may be three-colored. Avis and Toussaint [AT81] used Fisk's proof to design an $O(n \log n)$ algorithm for placing the guards. Recently, Kooshesh and Moret [KM92] showed that the guards can be placed in linear time. Although many similar problems have been studied in computational geometry [O'R87, She89, She92], little is known about guarding an object in three dimensions. In this chapter we present some results on guarding the surface of a polyhedral terrain.

The problem of guarding a polyhedral terrain was first investigated by deFloriani, et al. [dFP+86]. They showed that finding the minimum number of vertex guards could be done using a set covering algorithm. Cole and Sharir [CS89] showed that the problem was NP-complete.

In this chapter, we first show that the problem of finding the minimum number of edge guards for a polyhedral terrain is NP-complete. Then we show that $\lfloor \frac{n}{2} \rfloor$ vertex guards are always sufficient and sometimes necessary to guard an n-vertex terrain. We also present a linear time algorithm for placing $\lfloor \frac{3n}{5} \rfloor$ vertex guards to cover a terrain. With respect to edge guards, we establish that $\lfloor \frac{(4n-4)}{13} \rfloor$ edge guards are sometimes

necessary to guard the surface of an n-vertex terrain. The sufficiency result of $\lfloor \frac{n}{3} \rfloor$ edge guards is proved by Everett and Rivera-Campo [ERC92]. Finally, we present a linear time algorithm for placing $\lfloor \frac{2n}{5} \rfloor$ edge guards to cover a polyhedral terrain. Reducing the gap between sufficiency and neccssity for edge guards and finding efficient, practical algorithms to achieve the known bounds remain open problems.

We begin by reviewing some of the terminology used throughout this chapter.

Recall that a *polyhedral terrain* (or *terrain*) T is a connected polyhedral surface in R^3 with n vertices such that the intersection of T with every vertical line is either a point or it is empty. This immediately implies that there exists a planar graph T' associated with the terrain whose embedding is the orthogonal projection of T on the XY-plane. Consequently, every component x (a vertex, edge or a face) of T has a corresponding component x' in T'. Two points a, b on or above T are said to be *visible* if the line segment $\overline{ab}$ does not intersect any point strictly below T.

Throughout this chapter, we only consider problems concerning vertex and edge guards. A *vertex guard* is a guard that is only allowed to be placed at the vertices of T. An *edge guard* is a guard that is only allowed to be placed on the edges of T. A point x on T is said to be visible to an edge if there exists a point y on the edge such that x and y are visible.

6.1 Minimum edge guarding a polyhedral terrain is NP-complete

In this section we show that the problem of deciding whether or not a set of edge guards (including two endpoints) can cover the surface of a terrain is NP-complete. The NP-hard proof is achieved by modifying the proof of Cole and Sharir for proving the NP-completeness of the minimum vertex guarding a polyhedral terrain problem, which uses a reduction from SATISFIABILITY, a famous NP-complete problem [Coo71].

SATISFIABILITY

INSTANCE: A set of variables $x_1, ..., x_n$ and a CNF formula F with clauses $C_1, ..., C_m$ over $x_1, ..., x_n$.

QUESTION: Is there a truth assignment of F?

Essentially we first follow [CS89] to construct a polyhedral terrain. Then we modify the construction of *pits* to prove our result. Start with a horizontal plane, erect $n-1$ *walls* and dig some *pits*. The $n-1$ parallel walls separate the plane into n rows, where each row corresponds a variable. The pits all lie within rows. There are $2m$ pits per row and they are arranged in a circular fashion (see Figure 6.1 (a), which displays the view of a row from the above). The upper rims of the pits are all the same, and the rims of each pair of adjacent pits in the same row have a common vertex, which we call a peak vertex. The modification is to first construct the pits differently such that the rims of each pair of adjacent pits in the same row have a common edge, which we call a *peak edge* (*peak* for short) (see Figure 6.1 (b)). We create m columns, perpendicular to the rows, one per clause. Each column cuts through all of the walls. The idea is that only from a peak edge are the interiors of the two adjacent pits completely visible, and only from the boundary or interior of a pit can we see the whole pit. This can be achieved by making the pit very deep relative to the height of the walls (see Figure 6.2). The rest of the construction is exactly the same as that of Cole and Sharir [CS89].

In order to see the $2m$ bottoms of all the pits in a row, at least m edge guards will have to be needed. Moreover, let $e_1, ..., e_{2m}$ be the peak edges in some row. Then to be able to view all the pits in this row with exactly $2m$ edges, they will have to be placed at every other peak edge (i.e., either at the odd-numbered peaks or at the even-numbered peaks of that row). Assume row r corresponds to variable x_r, the choice of even peak edges for the guarding edges in r will correspond to setting $x_r = true$, while the choice of odd peak edges will correspond to setting $x_r = false$. If clause C_j of formula F does not contain x_r or $\bar{x}_r$, then the peaks p_{2j-1}, p_{2j} in row r are placed outside of any column; if C_j contains x_r (resp. $\bar{x}_r$), then peak edge p_{2j} (resp. p_{2j-1}) is placed inside the jth column while p_{2j-1} (resp. p_{2j}) is placed outside of any column. We can arrange this by varying the lengths of the pits in the rth row.

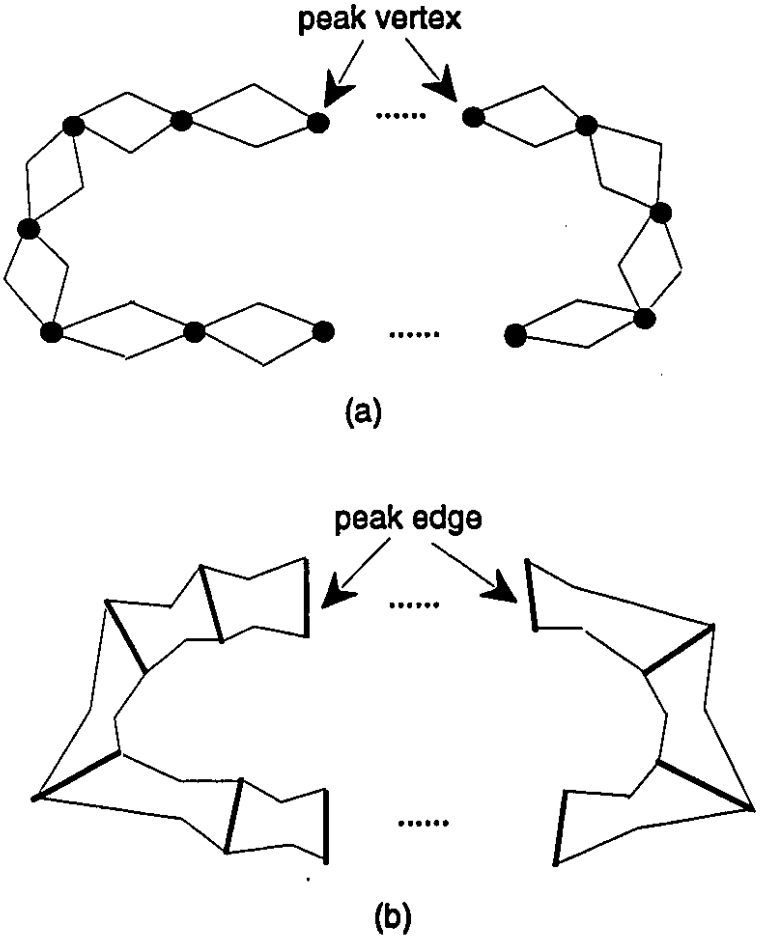


Figure 6.1: A view of the pits in a row.

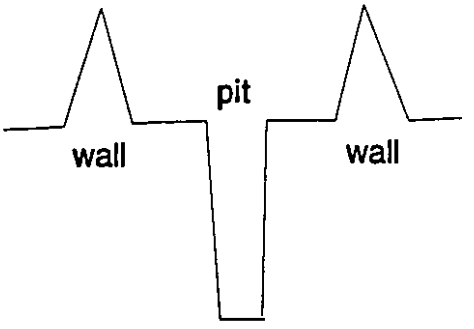


Figure 6.2: The pit can only be guarded by the edges on its rim (or inside it).

A wall has a triangular cross-section (parallel to columns). Thus a side of a wall can be seen entirely from any peak edge in either of the adjacent rows. Therefore, no additional edge guards are required for viewing the sides of the walls parallel to the rows. We show in Figure 6.3 a terrain for the following formula

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4).$$

Thus the reduction results in a polyhedral terrain δ with $O(mn)$ faces which requires at least mn edge guards to see all of it. Moreover, if exactly this number of edge guards is used, all of them must be placed at peaks in the manner described above. Furthermore, we must have at least one edge guard per column. This occurs if and only if formula F is satisfiable. We have thus shown that SATISFIABILITY is reducible to the problem of determining whether a polyhedral terrain can be guarded from a given number of edge guards in polynomial time. Therefore the latter problem is NP-hard.

However, unlike the problem of minimum vertex guarding a terrain, the problem of minimum edge guarding a terrain is not clearly in NP. We present below a polynomial time algorithm to decide whether a polyhedral terrain can be guarded by a set of k edges. Without loss of generality we triangulate all the faces of the terrain so that all the faces are triangular.

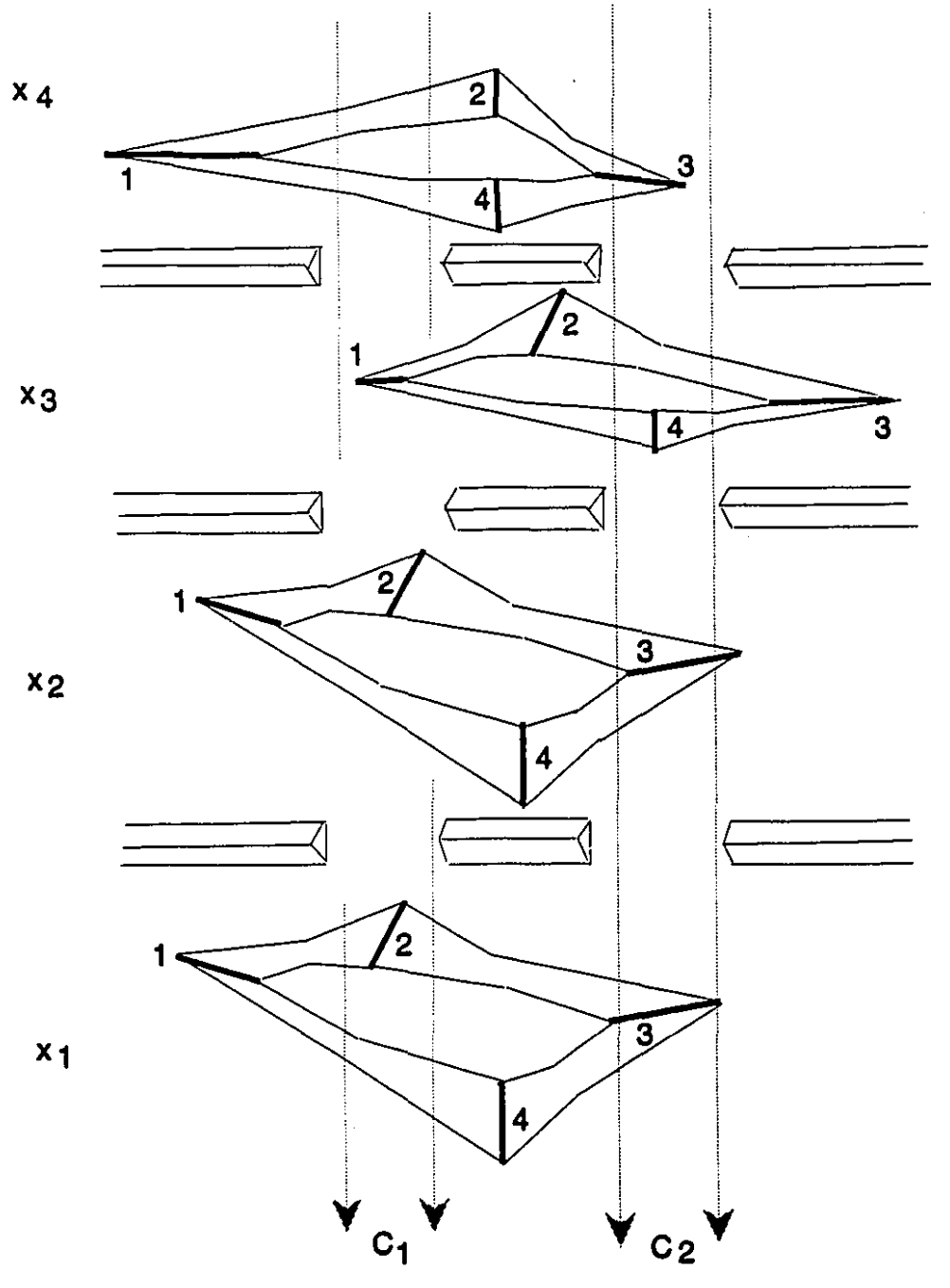


Figure 6.3: A terrain for the formula $F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4)$

Algorithm 6.1

BEGIN

(1) Remove all of the faces which are adjacent to the endpoints of the k edges. These faces can be guarded by the edge guards. If no face is left then report **YES** and exit.

(2) For each face left decide if it can be strongly guarded by an edge guard. If the answer is positive then report **YES** and exit; otherwise, decide if these faces can be guarded collectively by the k edge guards, if the answer is positive then report **YES** and exit, otherwise report **NO** and exit.

END

Notice that we have left out some details in Step (2). Testing if a face can be *strongly guarded* by an edge, i.e., the face can be guarded by every point on that edge, can be done by deciding whether or not the convex hull of the face and the edge is empty, which can be accomplished in linear time. Since there could be $O(n)$ faces left after Step (1) and for each face we need to test if it can be strongly guarded by one of the k edge guards the total complexity of this procedure is $O(n^2k) = O(n^3)$ time. The problem of testing whether or not a face F can be collectively guarded by the k edge guards is much more difficult. We must apply the results of [BDEG94] to compute the part of F visible from an edge in $O(n^5)$ time and $O(n^4)$ space.[1] Therefore, decide whether or not F can be guarded collectively by the k edge guards can be accomplished in $O(n^7)$ time and $O(n^6)$ space. Again, since there could be $O(n)$ faces left after Step (1) the total time complexity for this procedure is $O(n^8)$ time.

Therefore, the problem of minimum edge guarding a polyhedral terrain is in NP. Consequently we have shown the following theorem.

Theorem 6.1: It is NP-complete to determine, for a given polyhedral terrain with $O(n)$ faces and a given integer k, whether there exist k edge guards on the terrain which collectively see the entire surface of the terrain.

[1]This visible part is the union of the arrangement of "hyper"-planes, defined by $O(n^2)$ parabola and lines, thus has combinatorial complexity of $O(n^4)$. More details can be found in [BDEG94].

6.2 Guarding polyhedral terrains

A set of guards covers a terrain if every point on the terrain is visible from at least one guard in the set. The vertex guarding problem we study is the following: what is the number of vertex guards which is always sufficient and sometimes necessary to cover any polyhedral terrain? Similarly, the edge guarding problem is to determine the number of edge guards which is always sufficient and sometimes necessary to cover any polyhedral terrain.

The combinatorial counterparts of these terrain guarding problems can be expressed as guarding problems on the planar triangulated graph derived from the terrain. A vertex guard on the graph can only guard the faces adjacent to that vertex, and an edge guard on the graph can only guard the faces adjacent to the endpoints of the edge. The following theorem shows that we can restrict our investigation to guarding problems on the derived planar triangulated graph.

Theorem 6.2: In the worst case, the guarding problem on a polyhedral terrain is equivalent to the combinatorial guarding problem on the planar triangulated graph derived from the terrain.

Proof: Suppose a vertex on a polyhedral terrain could not guard one of the faces adjacent to it. This would imply that some other face was obstructing its vision, but that would violate the property that the intersection with a vertical line must be a single point. Thus a guarding of the planar triangulated graph implies a guarding of the polyhedral terrain.

On the other hand, given any planar triangulated graph embedded in the plane T', we can construct a terrain T based on T' by projecting it on the upper half of a sphere containing the planar triangulated graph. In this case, a vertex (edge) guard x of T can only see the faces adjacent to it. □

6.2.1 Guards on a terrain

In this section we show that $\lfloor \frac{n}{2} \rfloor$ vertex guards are always sufficient and sometimes necessary to guard a polyhedral terrain. We also show that $\lfloor \frac{(4n-4)}{13} \rfloor$ edge guards are

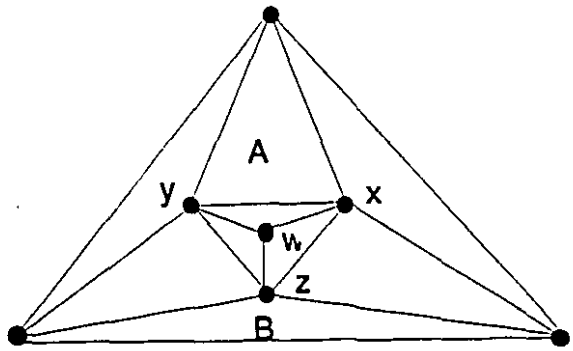


Figure 6.4: A seven-vertex terrain.

sometimes necessary to guard a polyhedral terrain.

Vertex guards

Lemma 6.3: The seven-vertex graph shown in Figure 6.4 needs at least three vertex guards. Furthermore, if three vertex guards are used to cover it, then at most one of the three guards can be an exterior vertex.

Proof: Suppose that two vertices suffice. One of the inner four vertices must be chosen to cover the inner triangles. If the central vertex is chosen, then the remaining unguarded (outer layer) triangles can not be covered by one guard, as the triangles A and B do not share a vertex. Therefore, one of the three middle vertices must be chosen. Without loss of generality, suppose vertex z is chosen. Then, the unguarded triangles (A and the three triangles adjacent to A) are not coverable by one vertex guard.

Now we show that at most one vertex guard can be an exterior vertex. If all three were exterior vertices, then the middle three triangles would be unguarded. Suppose that at least two of the vertex guards are exterior vertices. Without loss of generality, let them be the bottom two. We now have A and the three central triangles (directly below A) unguarded. These triangles can not be guarded with one additional guard. □

From the graph in Figure 6.4, we construct a series of planar subdivisions $S_1, \ldots, S_k$, where S_1 is the graph of Figure 6.4 and S_{k+1} is obtained from S_k in the following manner: let S_{k+1} be the graph of Figure 6.4 with one of the central triangles replaced by a copy of S_k (without loss of generality, suppose it is the one below face A). We show the following property about S_k:

Lemma 6.4: S_k is triangulated, has $n_k = 4k - 1$ vertices and needs $g_k = 2k - 1$ guards. If S_k is covered by exactly $2k - 1$ guards, then at most one guard is on the exterior face.

Proof: By induction on k.

Basis: $k = 1$. Follows from Lemma 6.3.

Inductive Hypothesis: For all $k \leq t$, $t \geq 1$, S_k is triangulated, has $n_k = 4k - 1$ vertices and needs $g_k = 2k - 1$ guards. Furthermore, if it is covered by exactly $2k - 1$ guards, then at most one guard is on the exterior face.

Inductive Step: $k = t + 1$. S_{t+1} is triangulated by construction. It has $n_t + 4 = (4t - 1) + 4 = 4(t + 1) - 1$ vertices. We now only need to show that it requires $2(t+1)-1 = 2t+1$ guards, and that if it uses that few, then only one exterior vertex is a guard.

In S_{t+1}, there is a copy of S_t. By induction, this copy of S_t must use at least $2t-1$ guards. We consider cases based on how many guards this copy of S_t uses as follows.

Case 1: The copy of S_t uses exactly $2t - 1$ guards. Then the copy of S_t has at most one guard on one of its exterior vertices. There are 4 subcases: no guard is placed on the exterior of S_t, left vertex (y) is a guard, right vertex (z) is a guard, and the lower vertex (w) is a guard.

1.1: No guard is placed on the exterior of S_t. Since S_t is already covered, two guards suffice to cover the remainder of S_{t+1}. We have that $g_{t+1} = (2t - 1) + 2 = 2(t + 1) - 1$. If exactly 2 guards are used, then at most one of them can be on the exterior of S_{t+1}.

1.2: A guard is placed at y. This configuration requires at least two guards. If covered with exactly two guards ($(2t - 1) + 2 = 2t + 1$ guards total), then at most one is on the exterior face.

1.3: A guard is placed at z. This subcase is symmetric to subcase 1.2.

1.4: A guard is placed at w. There is a ring of six triangles that requires two guards and at most one of these guards is on the exterior face.

Case 2: The copy of S_t uses exactly 2t guards. Then the copy of S_t may have guards on all three of its exterior vertices (i.e., x, y, z). However, this still leaves one face (B) uncovered, so one more guard is required. If only one more guard (2t+1 total) is used, then only that guard may be on the exterior face.

Case 3: The copy of S_t uses more than 2t guards. Then the induction hypothesis is true. □

Therefore we have the following theorems.

Theorem 6.5: There exists a terrain on n vertices, for any $n \equiv 3 \bmod 4$ that requires $\lfloor n/2 \rfloor$ vertex guards.

Proof: This follows directly from Lemma 6.4. For that terrain, we have: $g_k = 2k - 1$ and $n_k = 4k - 1$, therefore

$$g_k = 2((n_k + 1)/4) - 1 = (n_k + 1)/2 - 1 = (n_k - 1)/2 = \lfloor n_k/2 \rfloor. \square$$

Theorem 6.6: $\lfloor n/2 \rfloor$ vertex guards are always sufficient and sometimes necessary to guard the surface of an arbitrary terrain T with n vertices.

Proof: First 4-color the vertices of T'. This can always be done since T' is a planar graph [AH77]. By the pigeon hole principle, among the 4 colors there must be 2 colors such that no more than $\lfloor n/2 \rfloor$ vertices are colored by these two colors. Furthermore, these $\lfloor n/2 \rfloor$ vertices are sufficient to guard all of the faces of T' (because every triangle must have at least one vertex colored with one of these 2 colors). Necessity follows from Theorem 6.5. □

Edge guards

We now commence our investigation on edge guards.

Lemma 6.7: The terrain in Figure 6.5 needs at least two edge guards. Furthermore, if a mixture of edge guards and vertex guards are allowed, then one edge guard and one vertex guard suffice.

Proof: Suppose one edge guard suffices. We then have the following cases.

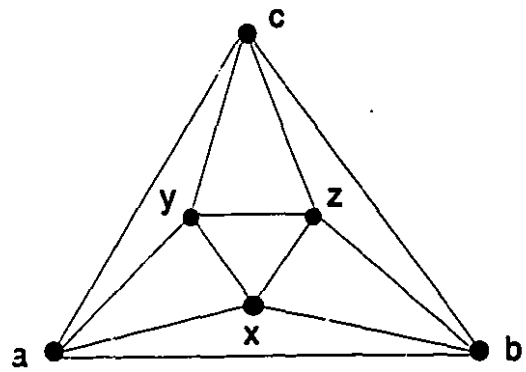


Figure 6.5: A six-vertex terrain which needs two edge guards.

Case 1. $\overline{ab}, \overline{ac}, \overline{bc}$ do not cover triangle(x, y, z).

Case 2. $\overline{ay}, \overline{ax}$ do not cover triangle(b, c, z).

Case 3. $\overline{xz}$ does not cover triangle(a, y, c).

All other cases follow by symmetry. Therefore we need at least two edge guards for the terrain in Figure 6.5 (edges $\overline{ab}$ and $\overline{yz}$ suffice). For all the cases above the unguarded faces can be covered by one vertex guard. □

Theorem 6.8: There is a planar triangulation that needs at least $(4n - 4)/13$ edge guards.

Proof: Such a planar triangulation is derived from an arbitrary triangulated convex polygon P with v vertices and $v - 2$ internal triangular faces.

We put a copy of Figure 6.5 in each face of P and along each edge of the boundary of P. Then we triangulate the untriangulated faces. (In total we add $v + (v - 2) = 2v - 2$ such copies to P.) Suppose the triangulation we obtain is P^* and it needs g_e edge guards. Because guards can not be shared between any copies of Figure 6.5, P^* requires at least $g_e = 2(2v - 2) = 4v - 4$ edge guards and has $v_{P^*} = v + 6(2v - 2) = 13v - 12$ vertices. Substituting v_{P^*} by n, we have: $g_e = (4n - 4)/13$. □

6.2.2 Algorithms for placing terrain guards

In this section, we present some practical, efficient algorithms for placing the vertex and edge guards. Since establishing the number of vertex guards and the number of edge guards sufficient to cover a terrain required the use of the four color theorem, finding a practical efficient algorithm to place the guards seems unlikely unless a deeper understanding of the problem is achieved. To this end, we present practical algorithms for guard placement which approximate the upper bounds.

Placing vertex guards

Observation 6.9: Given a five coloring of the vertices of any terrain, any set of three color classes provides a vertex guarding of the terrain since every face of the terrain is a triangle except possibly the outer face (which need not be guarded).

Based on this observation, a simple linear time algorithm follows:

Algorithm 6.2:

BEGIN

(1) Five-color the vertices of the planar triangulation graph.

(2) Among the five colors, choose three colors which are minimally used.

END

By [CNS81], Step (1) takes $O(n)$ time. Clearly, $O(n)$ time also suffices for Step (2). Therefore, the complexity of Algorithm 6.2 is $O(n)$.

Edge guard placement

We extend some of the elegant ideas of Everett and Rivera-Campo [ERC92] in order to develop a linear time algorithm for placing $\frac{2n}{5}$ edge guards to cover a polyhedral terrain. We use the following lemma.

Lemma 6.10: Given a finite collection of R real numbers there exists an element of R that must be less than or equal to average.

Proof: Let k be the average of the collection R. Suppose that there were no elements of R that were less than or equal to k. This implies that all of the elements are greater than k. But then k could not be the average. □

Our edge guard algorithm proceeds as follows. The first step in the algorithm is to five color the vertices. Let the five colors be: 1, 2, 3, 4, 5.

Let Matching(a, b, c) denotes a *maximal* matching (which is not necessarily a *maximum* matching) on the graph induced by the vertices in the three color classes a, b and c. Although Matching(a, b, c) does not provide a set of edges that guards the whole terrain, if we take all the edges in Matching(a, b, c) as well as one edge from each of the remaining unmatched vertices of color a, b, and c then we guard the whole terrain by Observation 6.9. Let Guard(a, b, c) represent the size of a set of edge guards obtained in this way. Also, let Size(a, b, c) represent the number of vertices of the three color classes a, b, and c. We have the following relation: Guard(a, b, c) = Size(a, b, c) - Matching(a, b, c). This relation holds because for each edge of the matching, we reduce the number of unmatched vertices by 2 which results in a reduction of the size of Guard by 1.

There are 10 possible combinations of three color classes resulting from the five coloring of the graph. We list them here in lexicographical order for reference: 123, 124, 125, 134, 135, 145, 234, 235, 245, 345. Let c_i represent the i^{th} combination in lexicographical order. Notice that each color class appears in 6 combinations. Thus,

$$\sum_{i=1}^{10} \text{Size}(c_i) = 6n.$$

Therefore we have the following lemma.

Lemma 6.11 If $\sum_{i=1}^{10} \text{Matching}(c_i) \geq 2n$, then there exists a guarding of size $\leq \lfloor \frac{2n}{5} \rfloor$

Proof: The average size of Guard =

$$\frac{\sum_{i=1}^{10} \text{Size}(c_i) - \sum_{i=1}^{10} \text{Matching}(c_i)}{10} \leq \frac{6n - 2n}{10} = \frac{2n}{5}$$

Therefore, one of the combinations provides a Guarding of size $\leq \frac{2n}{5}$ by Lemma 6.10. □

When $\sum_{i=1}^{10} \text{Matching}(c_i) \leq 2n$, we have the following lemma.

Lemma 6.12 One of the following pairs of Matchings provides a set of edges that guards the whole terrain: Matching$(1, 2, 3)$ and Matching$(1, 4, 5)$, Matching$(1, 2, 5)$

and Matching$(2,3,4)$, Matching$(1,2,4)$ and Matching$(3,4,5)$, Matching$(1,3,4)$ and Matching$(2,3,5)$, Matching$(1,3,5)$ and Matching$(2,4,5)$.

Proof: Let us first consider the first pair of matchings. Suppose there is a triangle which is not guarded. This means that all three vertices of the triangle must be unmatched. Clearly, the triangle can not contain edges whose endpoints have colors : $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{1, 5\}$, $\{2, 3\}$, $\{4, 5\}$, because if it did, we could add an extra edge to one of the Matchings contradicting the fact that it is maximal. So it must contain one of: $\{2, 4\}$, $\{2, 5\}$, $\{3, 4\}$, $\{3, 5\}$. Suppose it contained $\{2, 4\}$. Well the third vertex must have color: 1, 3, or 5. Thus, the triangle contains an edge which must be guarded. If it did not we could add an extra edge to one of the two matchings contradicting the fact that they are maximal. The argument is similar for the other three $\{2, 5\}$, $\{3, 4\}$, $\{3, 5\}$. The argument for the other four matching pairs is also similar. □

The average size of a matching pair $= \sum_{i=1}^{10} \frac{\text{Matching}(c_i)}{5} \leq \frac{2n}{5}$ (note that the average is taken over five since there are five matching pairs). Thus, one of the pairs of matchings provides a guarding with the desired size by Lemma 6.10.

Computing a maximal matching on a graph induced by the three chosen colors can be done in linear time in the number of edges in the graph. Thus $O(n)$ time suffices to compute all of the matchings induced by all 10 combinations of three color classes. Once all of the matchings are computed, Lemmas 6.10 and 6.11 guarantee that either a guarding or a pair of matchings will have size less than or equal to $\frac{2n}{5}$. Since there are only 10 different guardings and 5 pairs of matchings, the appropriate set can be found in only linear time. Therefore, we have the following theorem.

Theorem 6.13: Given a polyhedral terrain on n vertices, $O(n)$ time is sufficient to find a set S of edges to guard the terrain, where $\|S\| \leq \lfloor \frac{2n}{5} \rfloor$.

6.2.3 Conclusions

The following table summarizes the results of guarding polyhedral terrains.

	Sufficiency	*Necessity*	*Algorithmic Bounds*
Vertex Guards	$\lfloor n/2 \rfloor$	$\lfloor n/2 \rfloor$	$\lfloor 3n/5 \rfloor$
Edge Guards	$\lfloor n/3 \rfloor$	$\lfloor (4n-4)/13 \rfloor$	$\lfloor 2n/5 \rfloor$

There are some open problems related to this chapter and we summarize them as follows.

Open Problem 9: Is it possible to reduce the gap between sufficiency and necessity for edge guards? Are there practical efficient algorithms that match the known bounds?

Note: We have tried to use computer to find whether there is a 9-vertex planar triangulation which needs 3 edge guards. Recently with the help of David Avis and Komei Fukuda (they showed [and generated] that there are 78 non-isomorphic triangulations for a set of 9 planar points), we showed that there is no such triangulation which needs 3 edges (by checking all the 78 triangulations). (If there were one, we could immediately improve the lower bound on edge guards to $(6n-6)/19$.)

Chapter 7

Intersection detection and computation for Manhattan terrains

Intersection detection and computation is one of the fundamental problems in computational geometry. This problem finds applications in motion planning, collision detection and avoidance, computer graphics, CAD and VLSI [PS85]. Typical previous known results regarding intersection detection are as follows: detecting the intersection of a line or a line segment with a convex polygon [CD87], with a simple polygon [CG89]; detecting the intersection between two simple polygons [Mou92, Ama93]; detecting the intersection of a line, a plane, or a convex polyhedron with a convex polyhedron [DK83, CD87, DK90]. For the general intersection detection problem regarding non-convex polyhedra in 3D, a few results are only known very recently.

With the results of ray-shooting [dB92], de Berg shows that the intersection detection query of a line (a ray or a line segment) with an arbitrary polyhedron can be answered in $O(\log n)$ time and the data structure can be constructed in $O(n^{4+\epsilon})$ (for any $\epsilon > 0$) time and space. Furthermore, the intersection of a rectilinear line (a ray or a line segment) with an axis-parallel polyhedron can be detected in $O(\log n)$ time with $O(n^{1+\epsilon})$ (for any $\epsilon > 0$) time and space preprocessing. If the space complexity is of more concern, then he shows that the query can be answered in $O(\log n(\log\log n)^2)$

time and the data structure can be constructed in $O(n \log n)$ time and space. Since a Manhattan terrain is a special axis-parallel polyhedron which arises frequently in practice we would like to ask: if the object is a Manhattan terrain instead of an arbitrary axis-parallel polyhedron, can the rectilinear ray shooting be performed more efficiently? We show in this chapter that this question can be answered positively.

There are many more results regarding intersection computation and we are unable to list all of them. Most of these results before 1988 can be found in Chapter 7 of [PS85] (the second edition). Two of the most famous results of intersection computation after 1988 are the optimal linear time algorithm for computing the intersection of two convex polyhedra by Chazelle [Cha92] and the optimal $O(n \log n + K)$ time algorithm for computing the intersection of n line segments [CE92].

The known results regarding the intersection detection and computation of polyhedral terrains are as follows. The problem of computing the shortest vertical distance between two non-intersecting polyhedral terrains (two sets of lines or line segments in 3D) has been solved with a randomized algorithm with time $O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$); consequently, the problem of detection the intersection between two polyhedral terrains can be solved with the same bound [CEGS89]. The problem of computing the longest vertical distance between two polyhedral terrains (two sets of lines or line segments in 3D) has also been solved with a randomized with time $O(n^{4/3+\epsilon})$ (for any $\epsilon > 0$) [GP92]. (**Note:** The corresponding problem of computing the shortest or longest distance between two sets of rectilinear lines in 3D can be reduced to computing the red/blue closest pair and furthest pair between two sets of reals and can be solved easily in $\Theta(n \log n)$ and $\Theta(n)$ time respectively. However, the corresponding problem of computing the shortest or longest distance between two sets of rectilinear line segments in 3D can not be solved in this way.) The problem of computing the intersection (upper envelope) of two polyhedral terrains one of which is convex has been solved in optimal $O(n \log n + K)$ time [Sha88]. The problem of computing the intersection of two polyhedral terrains has been solved with a randomized algorithm with time $O(n^{4/3+\epsilon} + K^{1/3}n^{1+\epsilon} + K \log^2 n)$ (for any $\epsilon > 0$), where K is the size of output [Pel93].

In this chapter we consider the intersection detection and computation problems

for Manhattan terrains (solid Manhattan terrains). We show that after $O(n \log n)$ time and space preprocessing, the intersection of a rectilinear line segment (ray, or line) with a Manhattan terrain can be detected in $O(\log n)$ time. For the dynamic version of this problem, we show that there exists a dynamic data structure with a query and update of $O(\log^2 n)$ time and $O(n \log n)$ space. With these results, we are able to show that:

(1) Given two Manhattan terrains with a total of $O(n)$ vertices, we can either compute the shortest vertical distance between them or report their intersection in $O(n \log n)$ time. Equivalently, given two sets of rectilinear line segments, the shortest distance between them can be computed in $O(n \log n)$ time.

(2) Given two Manhattan terrains (or two sets of rectilinear line segments) with a total of $O(n)$ vertices, the longest vertical distance between them can be computed in $O(n \log n)$ time.

(3) Given two Manhattan terrains with a total of $O(n)$ vertices, we can compute their intersection (upper envelope) in $O(n \log n + K)$ time, where K is the combinatorial complexity of the envelope.

The techniques and data structures we use include: *multi-layer tree*, *segment tree* [Ben77, VW82], *symmetric order heap* [HT84], *fractional cascading* [CG86] and any one of the techniques and data structures supporting planar point location queries in $O(\log n)$ time with $O(n)$ time and space preprocessing [Kir83, EGS86, ST86]. In order to support the dynamic version of the above problems, we also use the 2D dynamic point location algorithm of Preparata and Tamassia [PT89].

7.1 Preliminary

We begin by recalling some elementary definitions. A Manhattan terrain $\mathcal{M}$ with n vertices is a connected 3D rectilinear polyhedral surface such that the intersection of any vertical line with $\mathcal{M}$ is either empty, a point, or a vertical line segment. A solid Manhattan terrain $\mathcal{M}$ is a simple rectilinear polyhedron such that there exists a face f of $\mathcal{M}$ and the intersection of $\mathcal{M}$ with any line perpendicular to f is either empty, or a line segment with one endpoint lying on f.

A segment tree is a data structure that is used to store a set of intervals on the real line. Segment trees are introduced by Bentley in 1977 [Ben77]. Since then, they have found many applications, especially for axis-parallel and c-oriented geometric objects [PS85].

Let S be a set of n possibly overlapping half-open intervals I_i's on the real line, i.e., $S = \{I_1, ..., I_n\}$. Let $I_i = [x_i, x_i')$. The $m \leq 2n$ different endpoints of I_i's partition the real line into $m+1$ half-open *elementary intervals*. The segment tree that stores S is a balanced binary tree with $m+1$ leaves, which correspond to the $m+1$ elementary intervals. Each internal node v of T has an interval associated with it that is the union of the intervals associated with its two children. In other words, each node v of T has an interval associated with it that is the union of all elementary interval leaves of $T(v)$. We denote this interval by I_v. An interval $I_j \in S$ is stored at those nodes v such that $I_v \subseteq I_j$, but $I_{parent(v)} \not\subseteq I_j$. One can check that I_j is precisely the disjoint union of all the intervals I_v over all node v where I_j is stored.

We give a procedure for building a segment tree by inserting intervals one after another [VW82]. Assume we have already built a tree T based on the $m+1$ elementary intervals (the root r of T corresponds to the real line), we want to store a line segment I in T.

Algorithm Insertion(T(r),I);

BEGIN

(1) If $I_r \subseteq I$, then add I to the node list of r.

(2) If $I_r \not\subseteq I$, then at least one of the following hold.

(2.1) If $I_{left(r)} \cap I \neq \emptyset$, then **Insertion(T(left(r)),I);**

(2.2) If $I_{right(r)} \cap I \neq \emptyset$, then **Insertion(T(right(r)),I).**

END

From the above insertion procedure we can see that I can be stored with at most two nodes at each level of T. Therefore an interval is stored at most $O(\log n)$ times in T, which implies that the space for storing T is $O(n \log n)$. We denote the subset of intervals that are stored at some node v by $L(v)$, and call $L(v)$ the *associated list of* v. Consequently we have the following theorem [VW82].

Theorem 7.1 [VW82]: A segment tree T for a set S of n intervals can be constructed

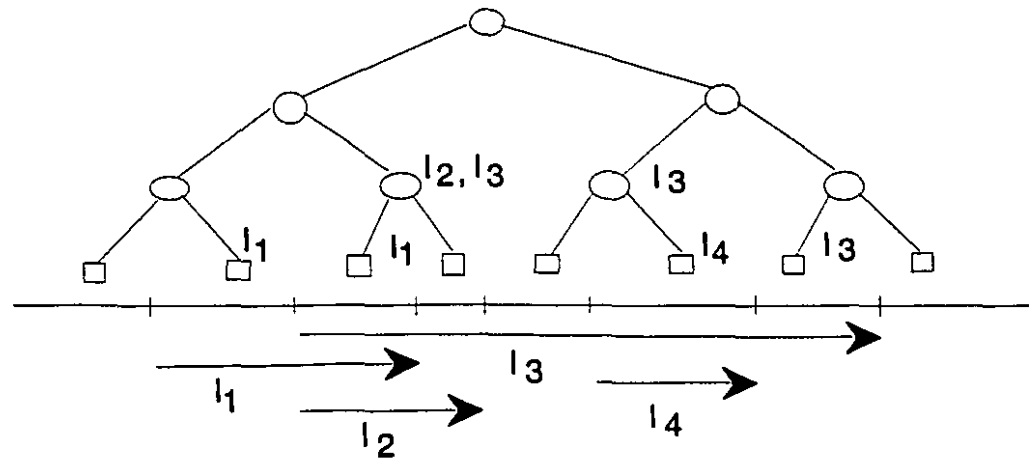


Figure 7.1: A segment tree of four line segments.

in $O(n \log n)$ time and space. The query of reporting all intervals that contain a query point x can be answered in $O(\log n + k)$ time, where k is the number of intervals reported. Furthermore, the set of intervals that contain x is the union of all associated lists $L(v)$ of all v on the search path of x in T.

In Figure 7.1, we show a segment tree storing four segments. In practice, to solve more complex problems we might need to store each associated list as some other data structure instead of a linked list. This gives us *multi-layer* data structures. Willard and Leuker [WL85] showed the following result:

Theorem 7.2 [WL85]: Suppose that the associated lists $L(v)$ in a segment tree are stored in data structures that can be built in $O(N(|L(v)|))$ time and space with $O(U(|L(v)|))$ query and update time. Then the total time and space for constructing this multi-layer date structure is $O(N(|L(v)|) \log n)$, the query and update time is $O(U(|L(v)|) \log n)$.

The intersection detection problem for Manhattan terrains is to design data structures to support efficient queries about the intersection between a preprocessed Manhattan terrain $\mathcal{M}$ and an arbitrary rectilinear line segment or another Manhattan

terrain. We also consider the dynamic version of the above problems, i.e., when only a constant number of changes occurred on the vertices, edges and faces of $\mathcal{M}$. (In the worst case, adding or deleting a new face to $\mathcal{M}$ can cause linear number of changes on vertices, edges and faces of $\mathcal{M}$.)

7.2 Detecting the intersection of two Manhattan terrains

In this section we show how to detect the intersection between two Manhattan terrains in $O(n \log n)$ time. We first show how to report the intersection between a Manhattan terrain $\mathcal{M}$ and a query rectilinear line segment in $O(\log n)$ time after $O(n \log n)$ time and space preprocessing. Recall that a line segment is *rectilinear* if it is perpendicular to either the XY-, or YZ- or XZ-planes.

Distance Definition 1: Given two rectilinear line segments t_1 and t_2 in 3-D, if there is a vertical line l such that $t_1 \cap l \neq \emptyset$, $t_2 \cap l \neq \emptyset$, then the *vertical distance* between t_1 and t_2 (denoted by $d(t_1, t_2)$) is the difference between the Z-coordinate of $t_1 \cap l$ and $t_2 \cap l$. Otherwise the vertical distance between the two rectilinear segments is infinity.

Unless specified otherwise, we use the above definition of distance. We first show how to compute the shortest vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear line segment after preprocessing $\mathcal{M}$. We then show how to generalize this procedure to answer the intersection detection query.

We first give an $O(\log^2 n)$ solution to compute the shortest vertical distance between a rectilinear line segment $\overline{ab}$ and a Manhattan terrain $\mathcal{M}$. The shortest vertical line segment $\overline{uv}$, with $v \in \overline{ab}$, $u \in \mathcal{M}$, must satisfy one of the following properties:

(1) v is either a or b;

(2) u lies on an edge of $\mathcal{M}$ and v lies on $\overline{ab}$.

The first case can be dealt with using planar point location (for this reason, we will ignore the case when $\overline{ab}$ is a vertical line segment, i.e., it is perpendicular to the XY-plane). For the second case we just consider the case when $\overline{ab}$ is perpendicular to the YZ-plane. Without loss of generality, assume $a = (x_a, y), b = (x_b, y)$.

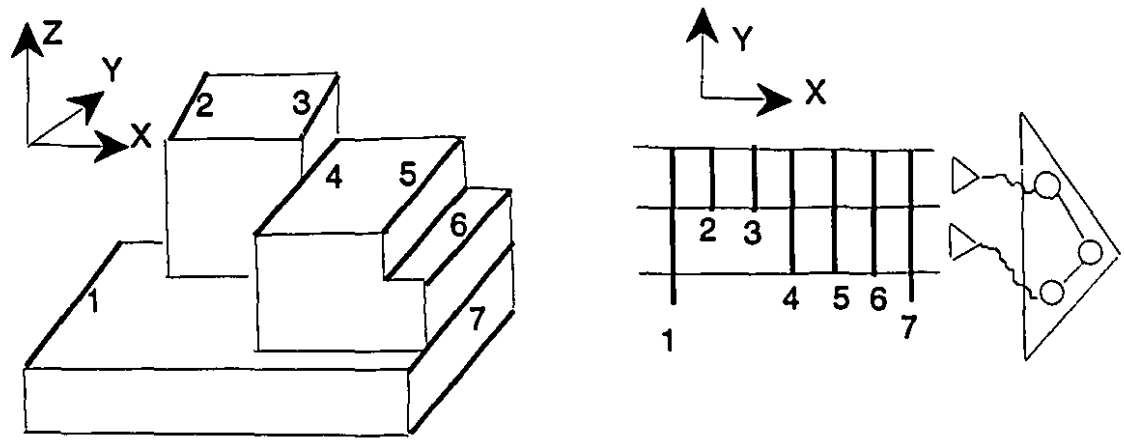


Figure 7.2: A Manhattan terrain and its two-layer segment tree.

The idea is to construct a *two-layer tree* such that the first layer is a segment tree and the second layer is a balanced binary search tree. We first build a segment tree T for all of the edges of $\mathcal{M}$ which are vertical to the XZ-plane (Figure 7.2). A node v in T has a certain y-interval I_v associated with it. Node v can be considered to represent the horizontal slab $[-\infty, +\infty] \times I_v$. When considering a vertical line segment inside the horizontal slab corresponding to v, we always restrict our attention to the part of the vertical line segment inside the slab corresponding to v. A search with y in T gives us $O(\log n)$ associated lists $L(v)$ such that the shortest vertical distance between $\overline{ab}$ and $\mathcal{M}$ is equal to the shortest vertical distance between $\overline{ab}$ and one of the elements in these lists.

We fix a plane $\mathcal{P}$ parallel to the XY-plane which is above $\mathcal{M}$. We store each associated list $L(v)$ in a balanced search tree T_v according to the X-coordinates of these $O(n)$ line segments such that a leaf corresponds to a line segment in $L(v)$ and each leaf store its distance to the plane $\mathcal{P}$; furthermore, the parent of two nodes is the one whose distance to $\mathcal{P}$ is smaller. With such a T_v, we can compute the shortest vertical distance between $\mathcal{P}$ and those segments between $X = x_a, X = x_b$, which is equal to the distance between $\mathcal{P}$ and the *nearest-common-ancestor* of x_a

and x_b (denoted by $nca(x_a, x_b)$), in the time proportional to the height of T_v (which is $O(\log n)$). Then the shortest vertical distance between $\overline{ab}$ and these segments can be computed in an extra $O(1)$ time: it is equal to $d(\mathcal{P}, nca(x_a, x_b)) - d(\mathcal{P}, \overline{ab})$, if $\overline{ab}$ is below $\mathcal{P}$; otherwise, it is equal to $d(\mathcal{P}, nca(x_a, x_b)) + d(\mathcal{P}, \overline{ab})$. In total, we have $O(\log n)$ associated lists for an elementary interval (which are the associated lists stored at the nodes on the path from the root to the leaf representing that elementary interval). For each of these $L(v)$ (and the corresponding T_v) it takes $O(\log n)$ time to compute the shortest vertical distance between $\overline{ab}$ and $nca(x_a, x_b)$ in T_v. Therefore the complexity for answering such a query is $O(\log^2 n)$ and the preprocessing time and space is $O(n \log n)$. Consequently we have the following theorem.

Theorem 7.3: After $O(n \log n)$ time and space preprocessing, the shortest vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be answered in $O(\log^2 n)$ time.

Note: Throughout this chapter, all the results regarding computing the shortest vertical distance can be generalized to computing the longest vertical distance. What we need to do is to construct a T'_v corresponding to $L(v)$ such that the leaves of T'_v correspond to the line segments in $L(v)$ and each leaf store its distance to the plane $\mathcal{P}$; furthermore, the parent of two nodes is the one whose distance to $\mathcal{P}$ is larger. With such a data structure we search the longest distance between $\mathcal{M}$ and a rectilinear line segment in the multi-layer tree with the same time complexity.

Moreover, the above data structure also solves the dynamic version of the problem. The point is that the second layer is a balanced binary search tree which supports an $O(\log n)$ time update [Tar83]. Combining this with the results of Preparata and Tamassia [PT89] (which supports $O(\log^2 n)$ time for a query or an update after $O(n)$ time and space preprocessing), we have the following corollary.

Corollary 7.4: After $O(n \log n)$ time and space preprocessing, the shortest (longest) vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be solved in $O(\log^2 n)$ time. A Manhattan terrain of constant size can be inserted into or deleted from the structure representing $\mathcal{M}$ in $O(\log^2 n)$ time.

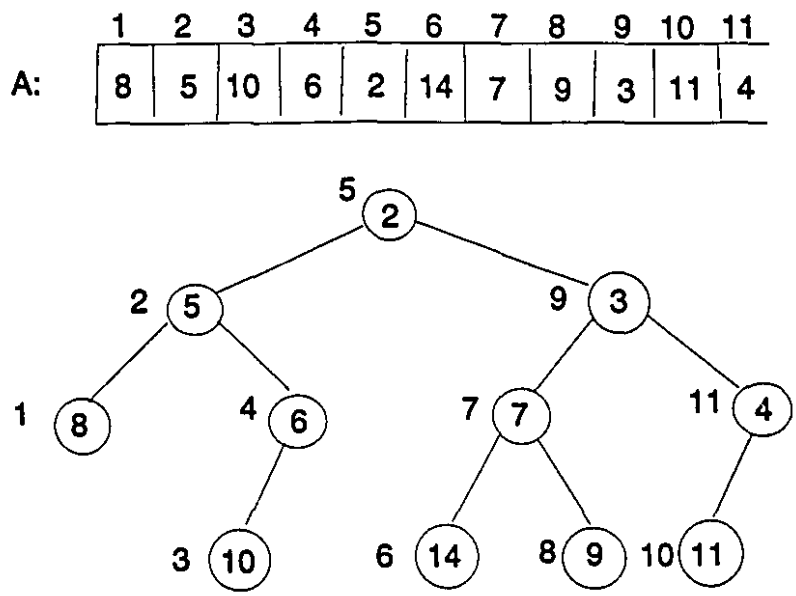


Figure 7.3: An array A and its Symmetric Order Heap.

We show that for the static version of the problem, we can improve the query time to $O(\log n)$ without increasing the time and space for preprocessing. The crucial point is that instead of storing a balanced binary tree for $L(v)$, we can store a special data structure called *symmetric order heap* [HT84], such that the *nearest-common-ancestor* of two nodes in the heap can be answered in $O(1)$ time. We call the resulting data structure a two-layer *hybrid segment tree.*

Let A[1..n] be an array of n real numbers. A *Symmetric Order Heap* (*SH*) is a binary tree that holds the entries of A[1..n]. *SH* has n nodes and each A[i] appears in only one node of *SH*. If we denote the node that holds A[i] by w_i, then *SH* has the following properties:

1. *SH* is a heap, i.e., if w_i is the parent of w_j then A[i] $\leq$ A[j].

2. The symmetric order (inorder) traversal of nodes of *SH* is $w_1, w_2, ..., w_n$, i.e., nodes in symmetric order contain A[1], A[2],...,A[n].

The following theorem is established in [HT84]:

Theorem 7.5 [HT84]: A symmetric order heap can be constructed in linear time and after an additional linear time preprocessing on the heap we can answer the nearest-common-ancestor query in $O(1)$ time.

Let $val(i) = d(\mathcal{P}, w_i)$ be the shortest vertical distance between w_i and $\mathcal{P}$, where $w_i \in L(v)$. We can use the $val(i)$ information for each w_i to construct a *SH* for every associated list. Then by Theorem 7.5, we can answer the nearest-common-ancestor query between any w_k and w_j in $O(1)$ time such that w_k is the leftmost interval intersecting with (the XY-projection of) $\overline{ab}$ and w_j is the rightmost interval intersecting with (the XY-projection of) $\overline{ab}$. However, there are $O(\log n)$ associated lists for an elementary interval and for each associated list $L(v)$ finding the leftmost (rightmost) interval intersecting with $\overline{ab}$ by binary search takes $O(\log n)$ time. Consequently the total time complexity could be $O(\log^2 n)$, which is no better than the binary tree implementation. Nevertheless we can apply the *fractional cascading* technique of Chazelle and Guibas [CG86] to improve the bound to $O(\log n)$.

Suppose $L(v)$ is the associated list of interval v in the segment tree T, and let $L(left(v))$ be the associated list of interval $left(v)$ in T. The basic observation is that the position of x_a in $L(v)$ will give us information about its position in the associated lists of its two children (for simplicity we only discuss $L(left(v))$). What we do is to first add two new pointers from each element in $L(v)$ to the smallest (largest) element which is at least as large (small) in $L(left(v))$. With these pointers, for the example shown in Figure7.4 we can find in $O(1)$ time the leftmost (rightmost) interval in $L(left(v))$ intersecting the query line segment $\overline{ab}$ once we know the the leftmost (rightmost) interval in $L(v)$ intersecting $\overline{ab}$. However, we are rarely in this fortunate situation since in general there could be $O(n)$ intervals in $L(left(v))$ between interval 1 and 2 (see Figure7.4). This problem can be overcome by copying certain elements of $L(left(v))$ into $L(v)$ and vice versa [CG86]. It turns out that the copying can be done in such a way that the search in $L(left(v))$ can be done in constant time if we already know the position the query value in $L(v)$; furthermore, the asymptotic preprocessing time and space is not affected. We summarize the result of [CG86] as follows.

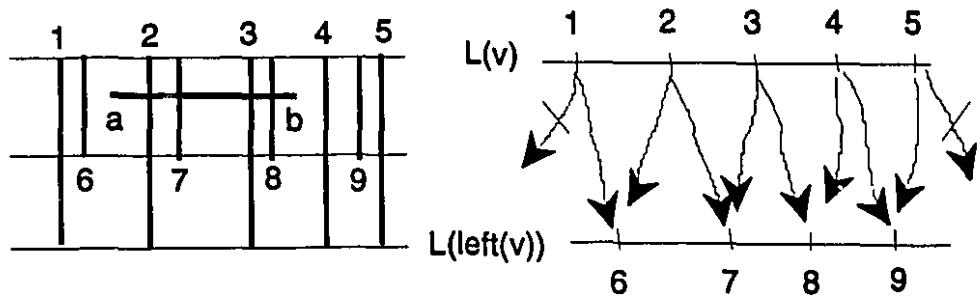


Figure 7.4: The pointers used for fractional cascading.

Theorem 7.6 [CG86]: Let T be a binary tree with $O(n)$ nodes, and suppose that each node v in T stores an ordered list $L(v)$. Fractional cascading allows us to compute the position of a query value in $L(left(v))$ in $O(1)$ time if we know the position of the query value in $L(v)$. The time needed to set up the fractional cascading structure is $O(n + \sum_{v \in T} |L(v)|)$.

Consequently we perform a binary search in the associated list which is stored at the root of the tree T, and then we can do the searches in the lists stored at the remaining nodes on the search path in $O(1)$ time per list. (We can also say that the amortized time for this search is $O(1)$ per list). Therefore we have the following theorem.

Theorem 7.7: After $O(n \log n)$ time and space preprocessing, the shortest vertical distance between a Manhattan terrain $\mathcal{M}$ and a rectilinear query line segment can be computed in $O(\log n)$ time.

Proof: The query is essentially the same as the one we perform for Theorem 7.3. The difference is that we can find the shortest vertical distance between $\overline{ab}$ and the elements in an associated list in $O(1)$ time with the result of Theorems 7.5 and 7.6. An elementary interval have $O(\log n)$ associated subsets, therefore in total we can find the shortest vertical distance between $\overline{ab}$ and all the elements in all these subsets in $O(\log n)O(1) = O(\log n)$ time. The time and space for preprocessing is still $O(n \log n)$. □

With this result we can proceed to obtain the following theorem.

Theorem 7.8: The shortest vertical distance problem between two non-intersecting Manhattan terrains with a total of $O(n)$ vertices can be computed in $O(n \log n)$ time.
Proof: As we have discussed above the shortest distance between $\overline{ab} \in \mathcal{M}_1$ and $\mathcal{M}_2$ can either be the shortest distance between a (b) and $\mathcal{M}_2$ or the shortest distance between $\overline{ab}$ and $\mathcal{M}_2$. The former can be solved using point location in $O(\log n)$ time and the latter can be solved in $O(\log n)$ time by Theorem 7.7. For every edge in $\mathcal{M}_2$, we can perform the above procedures symmetrically. Since there are $O(n)$ edges in $\mathcal{M}_1, \mathcal{M}_2$, the total time complexity is $O(n \log n)$. □

Now we show how to generalize the above result to answer the intersection detection queries of a rectilinear line segment and a preprocessed Manhattan terrain. In fact we can simply generalize the definition of the shortest vertical distance between an edge $\overline{xy}$ of $\mathcal{M}$ and $\overline{ab}$ as follows:

Distance Definition 2: If $\overline{ab}$ is above $\overline{xy}$, then $D(\overline{ab}, \overline{xy}) = d(\overline{ab}, \overline{xy})$, otherwise $D(\overline{ab}, \overline{xy}) = -d(\overline{ab}, \overline{xy})$.

With this definition, we can see that $\overline{ab}$ intersects with $\mathcal{M}$ if and only if the shortest vertical distance between $\overline{ab}$ and $\mathcal{M}$ is negative. Then we can follow Theorem 7.7 to obtain the following result (the only difference is that we store $D(\overline{ab}, v_i)$ instead of $d(\overline{ab}, v_i)$ in $val(i)$).

Theorem 7.9: After $O(n \log n)$ time and space preprocessing, we can either report the intersection of $\overline{ab}$ (or a rectilinear line) and $\mathcal{M}$ (by giving a witness), or return the shortest vertical distance between $\overline{ab}$ and a Manhattan terrain $\mathcal{M}$, or report this distance is infinity in $O(\log n)$ time.

Consequently we have the following result which is symmetric to Theorem 7.8.

Theorem 7.10: Given two Manhattan terrains with a total of $O(n)$ vertices we can either report their intersection (by giving a witness) or return the shortest vertical distance between them in $O(n \log n)$ time.

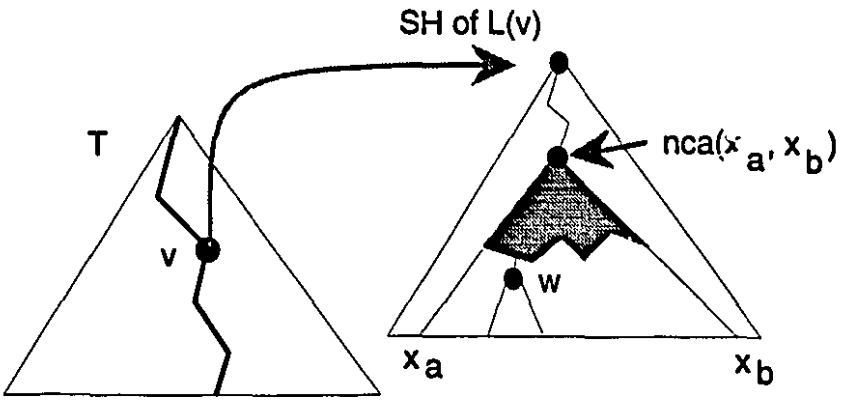


Figure 7.5: Computing the intersections of a line with a Manhattan terrain.

7.3 Computing the intersection of two Manhattan terrains

Following the second distance definition and Theorem 7.9, we can compute the intersection of a Manhattan terrain $\mathcal{M}$ and a query line segment $\overline{ab}$ in $O(\log n + k_{ab})$ time, where k_{ab} is the combinatorial complexity of the intersection of $\mathcal{M}$ and $\overline{ab}$. With this result we can compute all the edge/face intersections between the two Manhattan terrains. Consequently, the intersection (*upper envelope*) of two Manhattan terrains can be computed in $O(n \log n + K)$ time, where K is the combinatorial complexity of the envelope.

Theorem 7.11: The intersection of $\mathcal{M}$ and a query line segment $\overline{ab}$ can be computed in $O(\log n + k_{ab})$ time, where k_{ab} is the combinatorial complexity of the intersection of $\overline{ab}$ and $\mathcal{M}$.

Proof: We preprocess $\mathcal{M}$ as we have done in Theorem 7.7, that is, the second layer is a set of *SH*'s. Once we locate the position of x_a and x_b in $L(v)$ (which is stored as a symmetric order heap, each node has additional pointers to its next level for fractional cascading), we can compute the nearest-common-ancestor of them, $nca(x_a, x_b)$, in $O(1)$ time. Then we start a preorder traversal at the subtree rooted at $nca(x_a, x_b)$.

During the traversal, if a node w representing $\overline{cd}$ is traversed such that $D(\overline{ab}, \overline{cd}) > 0$ then discard the subtree rooted at w (since all these nodes rooted at w are below $\overline{ab}$ and there is no intersection between $\overline{ab}$ and all these nodes).

Suppose there are k_i intersections between $\overline{ab}$ and the edges stored in (the symmetric order heap of) $L(v)$. Since the number of leaves in a tree is at most the number of internal nodes plus one, it is clear that we visit at most $2k_i + 1$ nodes in $L(v)$ to compute all the intersections between $\overline{ab}$ and the edges stored in $L(v)$ (see Figure 7.5). Consequently it takes $\sum_{1 \le i \le \log n}(2k_i + 1) = O(\log n + 2\sum_{1 \le i \le \log n} k_i) = O(\log n + k_{ab})$ time to compute all the intersections between $\overline{ab}$ and $\mathcal{M}$. □

With this result, we can solve the following problem. Given two Manhattan terrains $\mathcal{M}_1, \mathcal{M}_2$, compute the upper envelope of $\mathcal{M}_1, \mathcal{M}_2$ (i.e., viewing $\mathcal{M}_1, \mathcal{M}_2$ as two functions $z = f_1(x, y)$ and $z = f_2(x, y)$ [1], the *upper envelope* of $\mathcal{M}_1, \mathcal{M}_2$ is the graph of the pointwise maximum of f_1, f_2). Clearly each vertex of the upper envelope is either:

(1) a vertex of $\mathcal{M}_1$ lying below $\mathcal{M}_2$, or

(2) a vertex of $\mathcal{M}_2$ lying below $\mathcal{M}_1$, or

(3) an intersection of an edge of $\mathcal{M}_1$ with a face of $\mathcal{M}_2$, or

(4) an intersection of an edge of $\mathcal{M}_2$ with a face of $\mathcal{M}_1$.

The first two types of vertices can be found in $O(n \log n)$ time using planar point location. The last two types of vertices can be found in $O(n \log n + K)$ time by Theorem 7.11. After all these vertices have been computed, we have the following theorem:

Theorem 7.12: The upper envelope of two Manhattan terrains can be computed in $O(n \log n + K)$ time, where K is the combinatorial complexity of the upper envelope.
Proof: We have just shown how to compute the vertices of the envelope in $O(n \log n + K)$ time. To list the envelope as a planar graph, we need to list the edges and faces adjacent with these vertices. This can be done without affecting the asymptotic time and space complexity. For the first two types of vertices this is straightforward as we

[1] For a point (x, y) on the XZ-plane or the YZ-plane, f_1 (f_2) is defined as the maximal length vertical line segment on $\mathcal{M}_1$ ($\mathcal{M}_2$) such that the lower endpoint of the line segment is (x, y).

already have the DCEL representations of the two Manhattan terrains. For a vertex v of the envelope such that $v = e_1 \cap f_2$ such that e_1 is an edge of $\mathcal{M}_1$ and f_2 is a face of $\mathcal{M}_2$. The faces of the envelope which are adjacent to v are exactly the parts of those faces adjacent to e_1 in $\mathcal{M}_1$ and a part of f_2. Again since we already have the DCEL representations of $\mathcal{M}_1$ and $\mathcal{M}_2$, the faces which are adjacent with the last two types of vertices can be listed in $O(n \log n + K)$ time. □

Similarly, given two solid Manhattan terrains such that their valid bases are on the same plane we can compute their intersection or union in $O(n \log n + K)$ time.

7.4 Some remarks

In this chapter, we have shown that the rectilinear ray shooting problem for a Manhattan terrain can be solved more efficiently than that for an arbitrary axis-parallel polyhedron. However, our data structure for a Manhattan terrain does not support efficient ray shooting queries for an arbitrary ray. We list this as an open problem.

Open Problem 10: Is it possible to solve the intersection detection (ray shooting) problem of a Manhattan terrain with an arbitrary line in 3D in $O(\log n)$ time with $o(n^2)$ time and space preprocessing?

Chazelle et al. [CEGS89] have an $O(\log^2 n)$ time solution to solve the ray shooting problem between a polyhedral terrain and an arbitrary line with $O(n^{2+\epsilon})$ time and space preprocessing. de Berg [dB92] has an $O(\log n)$ time solution to solve the ray shooting problem between a 3D axis-parallel polyhedron and an arbitrary line with $O(n^{2+\epsilon})$ time and space preprocessing. de Berg's result gives us a better solution to the above problem. But can we do even better? (Note that a Manhattan terrain is a special polyhedral terrain as well as a special 3D axis-parallel polyhedron.)

Chapter 8

Conclusions

In this thesis, we have studied a series of problems regarding polyhedral terrains. These problems include the problem of testing if a polyhedral object is a terrain, computing the shortest watchtower of a terrain, guarding a polyhedral terrain with a number of guards which is provably always sufficient and sometimes necessary, detecting and computing the intersection between Manhattan terrains to the problem of tetrahedralizing several classes of simple and non-simple polyhedra, which include some special classes of solid terrains. These results, all of which are both practical and implementable, have applications in computer graphics, CAD/CAM, military surveillance, forest fire monitoring, locations of radio transmission stations and the emerging areas of geographical information systems and spatial databases. In this final chapter, we summarize the most representative results of this thesis and mention some related open problems.

In Chapter 2 the problems of deciding if a polyhedral surface is a polyhedral terrain and if a simple polyhedron is a solid terrain have been investigated. It turns out that the latter problem is closely related to the problem of deciding whether or not a polyhedral object can be manufactured by stereolithography. Optimal, practical and straightforward linear time algorithms have been obtained to solve these problems.

In Chapter 3 we consider a generalization of the convex hull, i.e., the α-hull of a terrain and obtain algorithms to compute the exact and approximate α-hulls of a terrain. This problem is closely related to the problem of manufacturing a solid

terrain using NC-machining.

In Chapter 4 the problem of tetrahedralizing simple and non-simple polyhedra, which include some special classes of solid terrains, is studied. Although it is known that not all polyhedra admit a tetrahedralization and it is NP-complete to decide whether a simple polyhedron can be tetrahedralized, there are some results known about tetrahedralizing some special classes of simple and non-simple polyhedra by Goodman and Pach [GP88], and Bern [Ber93]. In Chapter 4 we extend the set of tetrahedralizable simple and non-simple polyhedra by showing that certain classes of simple and non-simple polyhedra, which include some special classes of solid terrains, admit a tetrahedralization and can be tetrahedralized efficiently. We also show that an arbitrary solid terrain does not always admit a tetrahedralization.

In Chapter 5 the problem of computing the shortest watchtower is studied. The first known $O(n \log n)$ algorithm to compute the shortest watchtower of a polyhedral terrain is proposed. This settles an open problem posed six years ago by Sharir [Sha88].

In Chapter 6 the problems of guarding polyhedral terrains with vertex and edge guards are studied. Although the problems of locating the minimum number of vertex and edge guards to guard the whole surface of a polyhedral terrain are all NP-complete, it has been shown in Chapter 6 that $\lfloor n/2 \rfloor$ vertex guards are always sufficient and sometimes necessary to guard the surface of an arbitrary polyhedral terrain and $\lfloor (4n-4)/13 \rfloor$ edge guards are sometimes necessary to guard the surface of a polyhedral terrain, which is the best known lower bound in contrast to the $\lfloor n/3 \rfloor$ upper bound by Everett and Rivera-Campo [ERC92]. Although these results are not practical due to the employment of the four-color theorem in the proof, practical linear time algorithms have been obtained to guard a polyhedral terrain.

In Chapter 7 the problems of detecting and computing the intersection of Manhattan terrains have been studied. Although the same problems regarding arbitrary polyhedral terrains have been investigated in recent years and fast randomized algorithms have been proposed, we are able to show that these algorithms can be improved significantly if the terrains are rectilinear (Manhattan terrains). A data structure, which is obtained by marrying the standard segment tree with the special

symmetric order heap, is proposed to represent a Manhattan terrain and is at the core of these algorithms.

Although we have studied a series of elementary problems involving polyhedral terrains and have obtained many new results in this thesis, there are many problems which remain to be solved efficiently, many of them have been mentioned in the thesis. Below we give a list to summarize these problems for future research.

(1) What is the complexity of decomposing a 3D polyhedral surface S into the minimum number of terrains (along different directions) if S is not a terrain?

(2) Is it possible to improve the $O(n^3)$ upper bound for computing the α-hull of a terrain?

(3) What is the complexity of deciding if a solid terrain can be tetrahedralized?

(4) **Conjecture**: All solid Manhattan terrains can be tetrahedralized without using Steiner points.

(5) Can a $U(3)$ polyhedron always be tetrahedralized?

(6) What is the lower bound for computing the shortest watchtower of a polyhedral terrain? Or, is it possible to obtain an $o(n \log n)$ time algorithm to solve it?

(7) For the problem of computing the shortest vertical distance between two non-intersecting terrains, is it possible to obtain a faster, deterministic algorithm?

(8) For the problem of intersection detection between two convex polyhedra, is it possible to obtain an $o(\log^2 n)$ query bound (even at the cost of increasing preprocessing time and space)?

(9) Is it possible to reduce the gap between sufficiency and necessity for edge guards in Chapter 6? Are there practical, efficient algorithms that match the known bounds for vertex and edge guards?

(10) Is it possible to solve the intersection detection (ray shooting) problem for a Manhattan terrain with an arbitrary line in 3D in $O(\log n)$ time with $o(n^2)$ time and space preprocessing?

Besides all these theoretical problems, a further direction of research is to apply these recently developed algorithms to the fields of geographical information systems and spatial databases to obtain better GIS and spatial database software products.

Very recently there have been several results in this field, for example, applying plane sweep to obtain efficient intersection queries in spatial databases [GS91]. It will be very interesting to see how additional geometric algorithms can be applied directly in GIS and spatial databases.

Bibliography

[AAP86] T. Asano, T. Asano, and R. Pinter. Polygon triangulation: efficiency and minimality. *J. Algorithms*, 7:221–231, 1986.

[ABB+93] B. Asberg, G. Blanco, P. Bose, J. Garcia-Lopez, M. Overmars, G. Toussaint, G. Wilfong, and B. Zhu. Feasibility of design in stereolithography. In *Proc. FSTTCS'93, India.*, volume 761 of *Lecture Notes in Computer Science*, pages 228–237. Springer-Verlag, 1993.

[AH77] K. Appel and W. Haken. Every planar map is 4-colorable. *Ill Journal of Mathematics*, 21:429 – 567, 1977.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Publishing Company, Reading, MA., 1974.

[AHU83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data structures and algorithms.* Addison-Wesley Publishing Company, Reading, MA., 1983.

[Ama93] N. M. Amato. An optimal algorithm for finding the separation of simple polygons. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 48–59. Springer-Verlag, 1993.

[AO93] D. Abel and B. C. Ooi. *Advances in Spatial Databases: Third Intl. Sympos., SSD'93*, volume 692 of *Lecture Notes in Computer Science.* Springer-Verlag, 1993.

[AS93] P. Agarwal and M. Sharir. On the number of views of polyhedral terrains. In *Proc. 5th Canadian CG Conf.*, pages 55–60, 1993.

[AT81] D. Avis and G.T. Toussaint. An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recognition*, 13(6):395–398, 1981.

[Bag48] F. Bagemihl. On indecomposable polyhedra. *American Mathematical Monthly*, pages 411–413, 1948.

[BDEG94] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. *Algorithmica*, 11:360–378, 1994.

[Ben77] J. Bentley. Algorithms for Klee's rectangle problems. unpublished manuscript, Department of Computer Science, Carnegie-Mellon University, 1977.

[Ber93] M. Bern. Compatible tetrahedralizations. In *Proc. 9th ACM Computational Geometry Conf.*, pages 281–288, 1993.

[Bez72] P. Bezier. *Numerical Control—Mathematics and applications.* John Willey and Sons, London, 1972.

[BFK84] W. Boem, G. Fagin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer-Aided Geometric Design*, 1:1–60, 1984.

[BSTZ92] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding polyhedral terrains. In *Proc. 30th Allerton Conf.*, pages 402–404, 1992.

[BSTZ93] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding polyhedral terrains. *Submitted to: Computational Geometry: Theory and Applications (also as Technical Report SOCS 92.20, McGill University)*, 1993.

[Bur86] P.A. Burrough. *Principles of geographical information systems for land resources assessment.* Clarendon Press, Oxford, UK, 1986.

[CD85] B. Chazelle and D.P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pages 63–133. North-Holland, Amsterdam, Netherlands, 1985.

[CD87] B. Chazelle and D. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, 1987.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[CEGS89] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Lines in space—combinatorics, algorithms and applications. In *Proc. 21st STOC*, pages 382–393, 1989.

[CG86] B. Chazelle and L. Guibas. Fractional cascading, Part I: A data structuring technique. *Algorithmica*, 1(3):133–162, 1986.

[CG89] B. Chazelle and L. Guibas. Visibility and intersection problems in planar geometry. *Disc. Comp. Geom.*, 4(6):551–581, 1989.

[Cha83] I. Chappel. The use of vectors to simulate material removed by numerically controlled milling. *CAD*, 15(3):156–158, 1983.

[Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Disc. Comp. Geom.*, 6(5):485–524, 1991.

[Cha92] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[Cha93] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Disc. Comp. Geom.*, 10(4):377–409, 1993.

[Che89] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[Chv75] V. Chvatal. A combinatorial theorem in plane geometry. *J. Comb. Theory Ser. B*, 18:39–41, 1975.

[CNS81] N. Chiba, T. Nishizeki, and N. Saito. A linear 5-coloring algorithm for planar graphs. *J. Algorithms*, 2:317–327, 1981.

[Col86] R. Cole. Searching and sorting similar lists. *J. Algorithms*, 7(3):202–220, 1986.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd STOC*, pages 151–158, 1971.

[CP90] B. Chazelle and L. Palios. Triangulating a nonconvex polytope. *Disc. Comp. Geom.*, 5:505–526, 1990.

[CRS92] V. Chandru, V. T. Rajan, and R. Swaminathan. Monotone pieces of chains. *ORSA Journal on Computing*, 4(4):439–446, 1992.

[CS89] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *J. Symbolic Computation*, 7:11–30, 1989.

[CS90] B. Chazelle and M. Sharir. An algorithm for generalized point location and its application. *J. Symbolic Computation*, 10(3):281–309, 1990.

[dB92] M. de Berg. *Efficient algorithms for ray shooting and hidden surface removal.* PhD thesis, Department of Computer Science, Utrecht University, 1992.

[dFP+86] L. deFloriani, B. Falcidieno, C. Pienovi, D. Allen, and G. Nagy. A visibility-based model for terrain features. In *Proc. 2nd International Symp. on Spatial Data Handling*, pages 235–250, 1986.

[DJSH89] R. L. Drysdale, III, R. B. Jerard, B. Schaudt, and K. Hauck. Discrete simulation of NC machining. *Algorithmica*, 4:33–60, 1989.

[DK83] D.P. Dobkin and D.G. Kirkpatrick. Fast detection of of polyhedral intersection. *Theoret. Comput. Sci.*, 27(5):241–253, 1983.

[DK85] D.P. Dobkin and D.G. Kirkpatrick. A linear time algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(4):381–392, 1985.

[DK90] D.P. Dobkin and D.G. Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach. In *Proc. 17th ICALP*, pages 400–413, July, 1990.

[DR80] D. P. Dobkin and S. P. Reiss. The complexity of linear programming. *Theoret. Comput. Sci.*, 11:1–18, 1980.

[EGS86] H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[EKS83] H. Edelsbrunner, D.G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory*, IT-29:551–559, 1983.

[EM94] H. Edelsbrunner and E. Muecke. Three-dimensional Alpha Shapes. *ACM Trans. on Graphics*, (to appear), 1994.

[ERC92] H. Everett and E. Rivera-Campo. Edge guarding a polyhedral terrain. unpublished manuscript, Department of Computer Science, University of Quebec at Montreal, 1992.

[Fis78] S. Fisk. A short proof of Chvatal's watchman theorem. *J. Combin. Theory Ser. B*, 24:374, 1978.

[FP79] I.D. Faux and M.J. Pratt. *Computational Geometry for design and manufacture.* Ellis Horwood, Chichester, UK, 1979.

[GP88] J. Goodman and J. Pach. Cell decomposition of polytopes by bending. *Israel J. Mathematics*, 64(2):129–138, 1988.

[GP92] L. Guibas and M. Pellegrini. New algorithmic results for lines-in-3-space problems. Technical Report TR-92-005, International Computer Science Institute, 1992.

[GS91] O. Guenther and H.J. Schek. *Advances in Spatial Databases: Second Intl. Sympos., SSD'91*, volume 525 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[Her89] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[Hoo86] T. Van Hook. Real time shaded NC milling display. *ACM SIGGRAPH*, 20(4):15–20, 1986.

[HS93] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. In *Proc. 9th ACM Symp. Comput. Geom.*, pages 11–18, 1993.

[HT84] D. Harel and R. Tarjan. Fast algorithm for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[KB38] W. Kern and J. Bland. *Solid Mensuration with proofs.* John Willey and Sons, NY, 1938.

[Kei85] J.M. Keil. Decomposing a simple polygon into simpler components. *SIAM J. Comput.*, 14:799–817, 1985.

[Kir83] D.G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[KM92] A. Kooshesh and B. Moret. Three-coloring the vertices of a triangulated simple polygon. *Pattern Recognition*, 25(4):443–444, 1992.

[LD81] D.T. Lee and R.L. Drysdale. Generalization of Voronoi Diagrams in the plane. *SIAM J. Comput.*, 10(1):73–87, 1981.

[Lee91] J. Lee. Analyses of visibility sites on topographic surfaces. *Int. J. GIS.*, 5:413–429, 1991.

[Len11] N. Lennes. Theorems on the simple finite polygon and polyhedron. *American Journal of Mathematics*, 33:37–62, 1911.

[LZ93] Z. Li and B. Zhu. On the monotonicity of polygons and polyhedral terrains. In *Proc. ICYCS'93, Beijing, China*, pages 629–632, 1993.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, 1984.

[Mou92] D. Mount. Intersection detection and separators for simple polygons. In *Proc. 9th ACM Symp. on Computational Geometry*, pages 303–311, 1992.

[MP79] D. Muller and F. Preparata. Finding the intersection of n half-spaces in time $O(n \log n)$. *Theoret. Comput. Sci.*, 8(4):45–55, 1979.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[Pel93] M. Pellegrini. On lines missing polyhedral sets in 3-space. In *Proc. 9th ACM Symp. on Computational Geometry*, pages 19–28, 1993.

[PH77] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[PS81] F.P. Preparata and K. Supowit. Testing a simple polygon for monotonicity. *Inform. Process. Lett.*, 12(4):161–164, 1981.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[PT89] F. Preparata and R. Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM J. Comput.*, 18(4):811–830, 1989.

[PT92] F. Preparata and R. Tamassia. Efficient point location in a convex spatial cell-complex. *SIAM J. Comput.*, 21(2):267–280, 1992.

[PV92] F. P. Preparata and J. S. Vitter. A simplified technique for hidden-line elimination in terrains. In *Proc. 9th Symp. Theoret. Aspects Comput. Sci.*, volume 577 of *Lecture Notes in Computer Science*, pages 135–146. Springer-Verlag, 1992.

[PW79] R. Pressman and J. Williams. *Numerical Control and computer aided manufacturing.* John Willey and Sons, NY, 1979.

[Roc70] R.T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, NJ, 1970.

[RR92] A. Rosenbloom and D. Rappaport. Moldable and castable polygons. In *Proc. 4th Canadian CG Conf.*, pages 322–327, 1992.

[RS88] J. H. Reif and S. Sen. An efficient output-sensitive hidden-surface removal algorithms and its parallelization. In *Proc. 4th ACM Symp. on Computational Geometry*, pages 193–200, 1988.

[RS92] J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional polyhedra. *Disc. Comp. Geom.*, 7:227–253, 1992.

[Sch28] E. Schoenhardt. Uber die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98:309–312, 1928.

[Sha88] M. Sharir. The shortest watchtower and related problems for polyhedral terrains. *Inform. Process. Lett.*, 29(5):265–270, 1988.

[Sha93a] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. In *Proc. 34th IEEE Symp. Found. Comput. Sci. (FOCS 93)*, pages 498–507, 1993.

[Sha93b] M. Sharir. Arrangements of surfaces in higher dimensions: Envelopes, single cells, and other recent developments. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 181–186, Waterloo, Canada, 1993.

[She89] T. Shermer. *Visibility properties of polygons.* PhD thesis, School of Computer Science, McGill University, Montreal, Canada, 1989.

[She92] T. C. Shermer. Recent results in art galleries. *Proc. IEEE*, 80(9):1384–1399, 1992.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Comm. ACM*, 29:669–679, 1986.

[Sti91] S. Stifter. An axiomatic approach to Voronoi-diagrams in 3D. *J. Comput. Syst. Sci.*, 43:361–374, 1991.

[Tar83] R.E. Tarjan. *Data Structures and Network Algorithms.* SIAM monograph, Philadelphia, PA, 1983.

[TVWZ93] G. Toussaint, C. Verbrugge, C. Wang, and B. Zhu. Tetrahedralization of simple and non-simple polyhedra. In *Proc. 5th Canadian CG Conf.*, pages 24–29, 1993.

[VW82] V. Vaishnavi and D. Wood. Rectilinear line segment intersection, layered segmentation, and dynamization. *J. Algorithms.*, 3(2):160–176, 1982.

[WCC+93] T. Woo, S-Y Chou, L-L Chen, K. Tang, and S. Y. Shin. Scallop hull and its offset for a monotone chain in linear time. manuscript, Department of Industrial and Operations Engineering, University of Michigan, 1993.

[WL85] D. Willard and G. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.

[WW86] W.P. Wang and K.K. Wang. Geometric modeling for swept volume of moving solids. *IEEE Computer Graphics and Applications*, 6(12):8–17, 1986.

[Yap87] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Disc. Comput. Geom.*, 2:365–393, 1987.

[Zhu92] B. Zhu. Improved algorithms for computing the shortest watchtower of polyhedral terrains. In *Proc. 4th Canadian CG Conf.*, pages 286–291, 1992.

[Zhu93] B. Zhu. Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time. *Computational Geometry: Theory and Applications (submitted)*, 1993.