

On the three-dimensional visibility skeleton: implementation and analysis

Linqiao Zhang

School of Computer Science

McGill University, Montreal

September 2009

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of Ph.D.in Computer Science

©2009 - Linqiao Zhang

Contents

Title Page	i
Table of Contents	iii
Abstract	vii
Résumé	ix
Declaration	xi
Acknowledgments	xiii
List of Figures	xv
List of Tables	xxi
1 Introduction	1
2 Background and Related Work	11
2.1 The Visibility Complex	11
2.1.1 The 2D Visibility Complex	12
2.1.2 The 3D Visibility Complex	20
2.2 The Visibility Skeleton	25
2.2.1 The 2D Visibility Skeleton	26
2.2.2 The 3D Visibility Skeleton	28
2.2.3 The Size Complexity of the Visibility Skeleton	38
2.3 Overview of the Sweep Algorithm	40
3 Experimental Study of the Size of the 2D Visibility Complex	45
3.1 Models	46
3.2 Experiments	48
3.2.1 Software	48
3.2.2 Setting	48
3.2.3 Experimental Results and Interpretation	50
3.3 Summary and Bibliographic Notes	59
4 An Implementation of the Sweep Algorithm	61
4.1 The Input	61

4.2	The Output	62
4.3	Description of the Implementation	63
4.3.1	Preliminaries: The CGAL Library and Number Types	66
4.3.2	The 2D Visibility Skeleton	68
4.3.3	Computing Events	70
4.3.4	The Event List	74
4.3.5	Updating the 2D Visibility Skeleton and the Event List	75
4.3.6	Computing the Ordering of Bitangents	80
4.3.7	Computing the 3D Visibility Skeleton Vertices	82
4.4	Complexity of the Implementation	86
4.5	Software Validation	87
4.5.1	Visualization	88
4.5.2	Experimental Verification	92
4.6	Performance	93
4.6.1	Running Time in Terms of n and k	93
4.6.2	Running Time in Terms of the Number of Polygons	97
4.6.3	Running Time in Terms of Number Types	101
4.7	Conclusion and Bibliographic Notes	103
5	The Algebraic Degree of the Predicates	105
5.1	Computing Lines through Four Lines	107
5.2	Predicates	114
5.2.1	Preliminaries	114
5.2.2	Transversals to Four Lines	115
5.2.3	Transversals to Four Segments	117
5.2.4	Ordering Planes through Two Fixed Points, Each Containing a Third (Rational) Point or a Line Transversal	125
5.3	Experiments	129
5.4	Discussion	132
5.5	Bibliographic Notes	132
6	Experimental Study of the Size of the 3D Visibility Skeleton	133
6.1	The Visibility Skeleton of a Set of Polytopes	135
6.2	Setting of the Experiments	136
6.2.1	The Model	136
6.2.2	The Experiments	137
6.2.3	Number Type and Machine Characteristics	139
6.3	Experimental Results and Analysis	140
6.3.1	Number of Skeleton Vertices in Terms of \mathbf{n}	140
6.3.2	Number of Skeleton Vertices in Terms of \mathbf{n} and \mathbf{k}	141
6.4	Double versus Filtered_exact	146
6.5	Summary and Bibliographic Notes	148

7	Computing the 3D Visibility Skeleton	151
7.1	Preliminaries	154
7.2	Computational Relations among the Visibility Skeleton Vertices . . .	155
7.3	Recovery of the Full Skeleton	160
7.4	Tightness of the Succinct Skeleton	174
7.5	Discussion	176
8	Conclusion and Future Work	177
	Bibliography	183

Thesis advisors

Author

Sue Whitesides, Sylvain Lazard

Linqiao Zhang

On the three-dimensional visibility skeleton: implementation and analysis

Abstract

The visibility skeleton is a data structure that encodes global visibility information of a given scene in either 2D or 3D. While this data structure is in principle very useful in answering global visibility queries, its high order worst-case complexity, especially in 3D scene, appears to be prohibitive. However, previous theoretical research has indicated that the expected size of this data structure can be linear under some restricted conditions. This thesis advances the study of the size of the visibility skeleton, namely, using an experimental approach.

We first show that, both theoretically and experimentally, the expected size of the visibility skeleton in 2D is linear, and present a linear asymptote that facilitates estimation of the size of the 2D visibility skeleton.

We then study the 3D visibility skeleton defined by visual events, which is a subset of the full skeleton defined by Durand *et al.*. We first present an implementation to compute the vertices of that skeleton for convex disjoint polytopes in general position. This implementation makes it possible to carry on our empirical study in 3D. We consider input scenes that consist of disjoint convex polytopes that approximate randomly distributed unit spheres. We found that, in our setting, the size of the 3D visibility skeleton is quadratically related to the number of the input polytopes and linearly related to the expected silhouette size of the input polytopes. This

estimate is much lower than the worst-case complexity, but higher than the expected linear complexity that we had initially hoped for. We also provide arguments that could explain the obtained complexity. We finally prove that, using the 3D visibility skeleton defined by visual events, we can compute the remaining vertices of the full skeleton in almost linear time in the size of their output.

Directeur de thèse

Auteur

Sue Whitesides, Sylvain Lazard

Linqiao Zhang

Squelette de visibilité en trois dimensions: implémentation et analyse

Résumé

Le squelette de visibilité est une structure de donnée qui encode l'information de visibilité globale pour une scène donnée en 2D ou 3D. Cette structure de donnée est en principe très utile pour répondre à des requêtes de visibilité globale, mais elle est, en particulier en 3D, d'une complexité de haut degré dans le pire des cas qui semble prohibitive. Cependant, les recherches théoriques précédentes ont indiqué que l'espérance de la taille de cette structure de donnée peut être linéaire sous certaines conditions restreintes. Cette thèse approfondit l'étude de la taille du squelette de visibilité, au moyen d'une approche expérimentale.

Nous montrons d'abord qu'aussi bien théoriquement qu'empiriquement, l'espérance de la taille du squelette de visibilité en 2D est linéaire, et présentons une asymptote affine qui facilite l'estimation de la taille du squelette de visibilité en 2D.

Nous étudions ensuite le squelette de visibilité 3D défini par événement visuels, qui est un sous-ensemble du squelette complet défini par Durand *et al.*. Nous présentons tout d'abord une implémentation calculant les sommets de ce squelette pour des polytopes convexes disjoints en position générale. Cette implémentation nous permet de continuer notre étude empirique en 3D. Nous considérons des scènes données consistant en des polytopes convexes disjoints qui sont une approximation de sphères unités distribuées aléatoirement. Nous avons découvert que, dans ces conditions, la taille

du squelette de visibilité 3D a une relation quadratique en le nombre de polytopes donnés, et linéaire en l'espérance de la taille de la silhouette des polytopes donnés. Cette estimation est bien plus basse que la complexité dans le pire des cas, mais plus haute que la complexité linéaire que nous espérions initialement. Nous présentons aussi des arguments qui pourraient expliquer la complexité obtenue. Nous prouvons finalement qu'en utilisant le squelette de visibilité 3D défini par événement visuels, nous pouvons calculer les sommets restants du squelette complet en temps presque linéaire en la taille du résultat.

Declaration

This thesis contains no material which has been accepted in whole, or in part, for any other degree or diploma. Chapters 3 to Chapter 7 present new contributions to knowledge. Some of the results have already appeared in conferences or journals. My contribution to the research described in each of the chapters is given in detail as follows.

- Chapter 3 is based on joint research with Hazel Everett, Sylvain Lazard and Sylvain Petitjean. I contributed significantly to the research, and conducted all the experiments except those in the Section 3.2.3 "Analysis for low densities", which were conducted by Hazel Everett. I included this section here for completeness.
- Chapter 4 is my own work.
- Chapter 5 is based on joint work with Hazel Everett, Sylvain Lazard and Bill Lenhart. I contributed significantly to the research, and also I conducted the experiments.
- Chapter 6 is based on joint work with Hazel Everett, Sylvain Lazard, Christophe Weibel and Sue Whitesides. I designed and conducted all the experiments, and contributed significantly to the discussions of the results and their interpretation.
- Chapter 7 is based on joint work with Sylvain Lazard, Christophe Weibel, and Sue Whitesides. I initiated the research ideas, and equally contributed to the research investigations.

Part of the material in this thesis has been published in conferences or journals [40, 41, 106, 107] as annotated in the bibliographic notes at the end of each chapter.¹ In writing the chapters of my thesis, I have heavily relied on the existing papers. I produced the first drafts of all those papers except the paper [40] described in Chapter 5 for which I produced only part of the first draft.

¹Papers [40, 41, 107] were quoted, after publication, in the habilitation mémoire of Sylvain Lazard.

Acknowledgments

I would like to thank my supervisors, Sylvain Lazard and Sue Whitesides for all the help they gave me during my thesis preparation. I particularly appreciated the time passed with Sue in various cafe bars in Montreal, where we savored many computational geometry subjects accompanied by the scent of coffee. And I value all the debates with Sylvain, which give hard time, but shape good work.

I also thank my Ph.D. committee, professor David Avis and Gregory Dudek, for their helpful comments to my thesis. I thank Professor Gert Vegter for explaining the 2D visibility complex to me; Doctor Laurent Alonso for providing a helpful hand to solve some engineering issues at the starting point of my Ph.D; and Professor Hazel Everett for providing much help in general.

Being a joint Ph.D. candidate, I passed my Ph.D. study in both Loria, France and McGill, and I remember fondly the interesting environment and friends in both places.

Last but not least, I thank my loving husband Christophe Weibel for his humor, encouragement, and enlightenment, and my lovely son Tony who has brought unlimited joy to my life.

List of Figures

2.1	Some of the (a) vertices, (b) edges, and (c) faces of the 2D visibility complex of discs B , R , and G	15
2.2	Two directed, rotating tangent lines of object R in (a) Cartesian space, and (b) their dual representation in polar coordinates.	16
2.3	(a) and (c) Objects B , R , G in Cartesian space and their bitangents. (b) and (d) Dual representation of directed rotating bitangents. . . .	17
2.4	Some of the faces of the 2D visibility complex of objects B , R , G (shown as colored regions) and their corresponding dual representation. (a) Two faces that arise from line segments that have common occluders B , G , and their dual representations in (c); (b) one face that arises from line segments that have common occluders B , R and its dual representation in (d), and another face that arises from line segments that have common occluders R , G and its dual representation in (e).	18
2.5	The maximal free line segment corresponding to a 3D visibility complex vertex is tangent to (a) three or (b) four objects.	22
2.6	The set of maximal free line segments corresponding to a 3D visibility complex edge are tangent to (a) two or (b) three objects.	22
2.7	Dual representation of the 3D visibility complex of two spheres L and R (image credits: Fredo Durand [34]).	23
2.8	(a) The 2D visibility skeleton vertices and arcs, computed from objects A and B . Four vertices, labeled 1 , 2 , 3 , 4 , are shown as four blue line segments, and an arc, incident to vertices 1 and 2 , is shown as a set of dashed lines. (b) The corresponding 2D visibility skeleton graph of (a); the circular cycle corresponds to the vertices whose corresponding maximal free line segments are tangent to object A in clockwise order; and the other cycle is similarly defined, based on object B	27
2.9	The eight types of vertices of the 3D visibility skeleton. (a) <i>EEEE</i> , (b) <i>VEE</i> , (c) <i>FEE</i> , (d) <i>VV</i> , (e) <i>FF</i> , (f) <i>FvE</i> , (g) <i>FE</i> and (h) <i>FVV</i>	30
2.10	The degenerate case of type <i>FvE</i> vertex.	30

2.11	The four types of arcs of the 3D visibility skeleton. (a) EEE , (b) VE , (c) FE and (d) FVE	31
2.12	The arcs incident to a vertex of type (a) $EEEE$, (b), (c) VEE , (d) VV , and (e), (f) FvE	32
2.13	(a) T-event, (b) V-event, and (c) F-event.	41
3.1	Scenes of random disjoint unit discs with density as (a) 0.0025, (b) 0.1, and (c) 0.55.	47
3.2	Plots of the number of oriented bitangents, memory usage, and running time in terms of the number of unit discs, when scene density is equal to (a) 0.0025, (b) 0.005, (c) 0.025, and (d) 0.55. The unit of the memory usage is kB, that of the running time is 10^{-4} seconds.	50
3.3	The (a) slope and (b) y -intercept, in terms of μ , of the linear asymptote of the number of oriented bitangents (in terms of the number of discs): experimental data points and interpolations (of the square points) by (a) $\frac{17.49}{\mu} + 5.67 - 19.17\mu$ and (b) $-\frac{4.182}{\mu} + 19,255 - 23,789\mu$. The dashed curves are the theoretical upper bounds $(8(\mu + \frac{4\pi^2}{\mu})(n - 1))$ [41] times two since the bitangents are here oriented.	52
3.4	Onset of linearity in terms of the density μ : experimental data points and their fitting by $\frac{16.77}{\mu} + 47.55$	54
3.5	Number of non-oriented bitangents for density 0.0025, and an estimate of Eq. (3.1) for $n > 6,755$, with, in (b), the number $4\binom{n}{2}$ of possibly obstructed bitangents and the theoretical upper bound, $8(\mu + \frac{4\pi^2}{\mu})(n - 1)$ [41] (in dashed).	56
3.6	Hexagonal scene model (\mathcal{G}_4).	58
4.1	Organization of the implementation.	64
4.2	Computing potential future events (marked in purple) arising from bitangent t : (a) 4 pairs of potential T-events. (b) 4 potential V-events. (c) 4 potential F-events.	71
4.3	A T-event: (a) two bitangents (b) become collinear, and (c) a third bitangent appears; (d), (e), and (f): the 2D visibility skeleton corresponding to (a), (b), and (c).	76
4.4	(a) before, (b) during, and (c) after a V-start-event, when a set of new bitangents appears.	77
4.5	(a) before, (b) during, and (c) after a V-middle-event, a bitangent changed its supporting edge.	78
4.6	(a) before, (b) during, and (c) after a F-event, a bitangent changed its supporting edge.	78
4.7	Bitangents that are tangent to a polygon at the same or different vertices. 80	80

4.8	(a) Dropping z -coordinate results in different orientations of the two 2D discs: I_1 and I_2 . (b) Keeping the disc orientation consistent by using the sign of $(v_0 \cdot z) \times (v_2 \cdot z)$	81
4.9	The 3D visibility skeleton vertices of type (a) $EEEE$, (b) VEE and (c) FEE that are computed from T-, V-, and F-events. Note that the maximal free line segment lies in l_t , but its extent is not shown in the figure, <i>i.e.</i> if it does not extend to infinity, it is blocked beyond the figure.	83
4.10	Computation of a VEE vertex, in two cases different from Figure 4.9 (b).	84
4.11	(a) Case (i) and (b) Case (ii) of computing an FEE vertex.	86
4.12	Sample scenes of (a) <i>Scene I</i> , (b) <i>Scene II</i> , (c) <i>Scene III</i>	88
4.13	(a) One position of the sweep plane. (b) The view inside the sweep plane. (c) The eventlist, and (d) the 2D visibility skeleton for polygons in (b).	89
4.14	Snapshots of visualizing the computational steps of the implementation.	90
4.15	(a) An input of ten polytopes, and the output of (b) 6 $EEEE$, (c) 438 VEE and (d) 85 FEE type vertices.	91
4.16	Running time (in seconds; 2.88×10^4 seconds = 8 hours) in terms of n , the number of edges in the scene in Suite I ($\mu = 0.3$).	95
4.17	Running time (in seconds) in terms of $n^{1.5} k \log k$: for density (a) $\mu = 0.3$, (b) $\mu = 0.05$ and (c) $\mu = 0.01$, where the polytopes have constant complexity (n/k edges) (Suite I); and (d) for density $\mu = 0.3$, where the polytope complexity varies in the range of $[4 - 24]$, $[4 - 34]$, and $[4 - 44]$ (Suite III).	96
4.18	Running time ratio of number type <code>double</code> to <code>filtered_exact</code> , tested on the experiments in Suite I.	102
5.1	(a): Transversal ℓ intersects segment pq only if $(\ell \odot op) (\ell \odot oq) \leq 0$. (b-c): An illustration for the proof of Lemma 10.	118
5.2	Planes P_1 and P_2 such that $P_1 < P_2$	126
6.1	Three sample scenes of $k = 50$ polytopes where n/k , approximately the number of edges on each polytope, is equal to (a) 6, (b) 42, and (c) 84. The scene density $\mu = 0.3$ in all cases.	137
6.2	Mean and standard deviation of the number of computed skeleton vertices on ten scenes for each type of scene.	139
6.3	Suite I ($\mu = 0.3$): total number of skeleton vertices in terms of n , the number of edges in the scene.	140
6.4	Total number of skeleton vertices in terms of $k^2 \sqrt{n/k}$ when (a) the polytopes have a constant (n/k) number of edges (Suite I), and (b) the number k of polytopes is constant (Suite II).	142

6.5	Total number of skeleton vertices in terms of $k^2\sqrt{n/k}$ for polytope complexity (n/k edges) varying in the range of [4 - 24], [4 - 34], and [4 - 44] (Suite III).	143
6.6	The number of vertices in terms of $k^2\sqrt{n/k}$ as tested on Suite I (k polytopes having a constant, n/k , number of edges; $\mu = 0.3$).	143
6.7	Number of (a) <i>VEE</i> , (b) <i>FEE</i> vertices in terms of number of <i>EEEE</i> vertices (Suite I, $\mu = 0.3$).	146
6.8	Number of (a) <i>VEE</i> , (B) <i>FEE</i> vertices in terms of $\sqrt{n/k}$ (Suite II).	147
6.9	Error percentage of computed skeleton vertices when using number type <code>double</code> versus <code>filtered_exact</code> (Suite I).	147
7.1	Scene representing a shelf in a room with a fluorescent light on the ceiling. The black and white regions represent the umbra and full light regions. The union of the light and dark grey regions corresponds to the penumbra. The dark gray shape represents a portion of the penumbra limited by the trace of <i>FE</i> arcs. In this region, the visible portion of the light source does not exceed about 40%. The schema on the right represents a section through the middle of the scene. The points a are at the boundary of umbra, and the points c are at the boundary of the penumbra. The points b are on the maximal free line segments corresponding to an arc <i>FE</i> involving a face of the blocker. From a to b, the percentage of the light source that is visible increases linearly from 0% to about 40%, and from b to c, it increases linearly from about 40% to 100%. Since the light grey region can be made arbitrarily large by moving the light source closer to the blocker, the trace of the <i>FE</i> arcs on the floor corresponds to a discontinuity of the derivative in the percentage of visible area of the light source.	153
7.2	The possible computational relations among the types of 3D visibility skeleton vertices.	157
7.3	The value of each of the free line segments is defined by a linear function on its intersection with the plane <i>H</i>	161
7.4	The intersections with <i>H</i> of free line segments on the two <i>VE</i> arcs on each side of a degenerate <i>FvE</i> vertex move in opposite directions.	162
7.5	The intersections with <i>H</i> of free line segments on the two <i>FE</i> arcs on each side of a non-degenerate <i>FvE</i> vertex move in opposite directions.	162
7.6	The silhouette of the polytope from \mathbf{v} projected on <i>H</i> is inside the cone of the projected constraint edges. If the constraint edges are in the half-plane $u' \cdot x \geq 0$, so is the polytope.	164
7.7	(a) Bird's eye view and (b) 3D view of degenerate <i>FvE</i> vertices whose supports are polytope edge \mathbf{e} and two sequences of faces incident to \mathbf{v} , starting from \mathbf{e}_1 and \mathbf{e}_2 , which create <i>VE</i> arcs with \mathbf{v}'	168

-
- 7.8 When the polytope edge \mathbf{e} intersects with the polyhedral cone of the faces incident to \mathbf{v} , the two sequences of faces that are supports of degenerate FvE vertices turn in opposite directions until they meet, which indicates a non-degenerate FvE vertex. 169
- 7.9 In some configurations, a sequence of VE arcs may contain degenerate FvE vertices only, with a non-degenerate FvE vertex at each end. 169
- 7.10 (a) Three polytopes admit vertices of type VEE but not vertices of FEE . (b) Three polytopes admit vertices of type FEE but not vertices of type VEE . (c) A cross section of (b) as indicated by the red line segment. 175

List of Tables

2.1	Number of each type of skeleton arc incident to each type of skeleton vertex.	33
4.1	Experimental results reported in [34, 36], on a 195Mhz R10000 SGI Onyx2 (taken from [34]).	98
4.2	Experimental results for three pairs of input scenes with the number of polytope faces approximately 450, 750, and 1000.	99
4.3	Running time (in seconds) in terms of number type: <code>double</code> , <code>filtered_exact</code> , and <code>CORE</code> , as well as the ratio of <code>filtered_exact</code> and <code>CORE</code> to <code>double</code> , for four input scenes.	101
5.1	Percentages of failure of the degree 168 and degree 3 predicates using double-precision floating-point interval-arithmetic, for ϵ varying from 10^{-12} to 10^{-2}	131

Chapter 1

Introduction

Visibility problems arise commonly in areas such as computer graphics, computer vision, and robotics. In computer graphics, visibility problems have been studied for about four decades. Since the earliest problems such as visible surface determination [98], or occlusion culling [21], visibility has been always an important problem in computer graphics. In computer vision, visibility is involved in problems such as object reconstruction [86, 97], or sensor placement [99]. In robotics, it is involved in problems such as motion planning [68], or robots self-localization [31].

Given a set of objects in the Euclidean space, two points in the space are mutually visible if the line segment connecting them is not blocked by any objects. Visibility problems typically address queries on whether the points or objects of interest are visible to each other. The nature of the problem implies that the study of the visibility problem is essentially the study of the sets of lines that are related to the queried objects.

Based on the set of lines that are involved in the visibility queries, the visibility

problem can be classified as visibility from a point, a line segment, a polygon, a region, or global visibility [11]. Global visibility addresses the visibility between any pair of given objects.

Visibility queries from a point are relatively easy to handle and well understood. Typical problems related to these types of queries are ray shooting [7], visible surface determination [24, 30, 42, 98], and computing shadows cast by a point light source [102, 104]. Efficient and practical algorithms and data structures, such as ray tracing [102], Z-buffer [17, 18], binary space partitioning (BSP) tree [10, 43] have been developed to solve these problems, and some of them even have hardware implementations [17, 18].

However, when the visibility queries do not involve points, as in the problem of global illumination [51, 58], little is known. In particular there exists no solution for determining exactly and efficiently, in a 3D polygonal scene where polygons represent faces of objects, whether two given triangles see each other, or for determining the umbra cast by a polygonal light source. This situation suggests that the problem of global visibility is a hard one. Data structures such as the visibility complex [34, 85] and the visibility skeleton [36] have been proposed to deal with global visibility.

The *visibility complex* is a data structure that is designed to encode global visibility information. Roughly speaking, this data structure partitions the space of maximal free line segments into connected components of segments that touch the same objects. This data structure was initially proposed in 2D by Pocchiola and Vegter [85]. This 2D version has been extensively studied [6, 54, 84, 91], and further applied in graphics rendering [20, 77]. Later on, Durand *et al.* [34, 38] extended the study of the 2D

visibility complex to 3D. They introduced the 3D visibility skeleton data structure, which is a simplified version of the 3D visibility complex, that includes only partial information, *i.e.* the zero- and one-dimensional cells of the visibility complex.

Durand *et al.* applied the 3D visibility skeleton data structure to global illumination computation [34, 36, 37]. In their application, the input scene and the light source are modeled as 3D polygons lying on the surfaces of the objects. They compute the 3D visibility skeleton data structure through systematic brute force examination of combinations of the vertices and edges of the input (see Section 2.2.2 for details). In addition, they use various heuristics to speed up the computation. This application produces images with high quality shadow boundaries.

Despite their positive results, the work of Durand *et al.* also shows some drawbacks. First, their algorithm is not efficient because it is based on a brute force enumeration, and thus has worst-case time complexity $\Theta(n^5)$, where n is the total complexity of all the input polygons. Although the observed running time complexity, improved by the heuristics, is $\Theta(n^{2.5})$, it is still relatively high for practical use. Second, the worst-case size complexity of the 3D visibility skeleton is $\Theta(n^4)$ in the model used by Durand *et al.*, which can easily reach the memory limit of present day machines, and thus restricts the use of this data structure to small input scenes. Third, their implementation is not robust and its use requires a great deal of time-consuming human intervention to remove degeneracies from realistic scenes. As a result, the largest test scene they report contains less than 1500 polygons [34]. The 3D visibility skeleton data structure has since been often stated as impractical to use given its large size, high order complexity and robustness issues [23, 67, 70, 74, 92]. In consequence,

this data structure has not gained wide use in practical applications.

On the other hand, the empirical work [34, 36] conducted by Durand *et al.* reported that the $\Theta(n^4)$ worst-case bound is pessimistic, except for some unrealistically contrived scenes. On the basis of their preliminary experiments, the observed growth of the 3D visibility skeleton appears to be quadratic in the size of the input scene.

Motivated by the preliminary results of Durand *et al.*, more theoretical research has recently been done to study the size of the 3D visibility skeleton in various aspects. Bronnimann *et al.* [14] studied the dependence of the size of the 3D visibility skeleton on the number of polytopes rather than on the total number of edges alone. They found that, when considering the inputs as k polytopes with n edges in total, the worst-case size complexity is $\Theta(n^2k^2)$. Glisse [48] took into account the worst-case average silhouette size of the polytopes. He obtained a slightly better bound of $O(nk^3h)$, where $h \in O(n/k)$ is the maximum size of the silhouettes of each of the polytopes, which could be assumed to be in $O(\sqrt{n/k})$ under some reasonable assumptions. Devillers *et al.* [27] studied the expected size of the 3D visibility skeleton. When considering a simple case, *e.g.*, the input consists of unit balls that are randomly distributed inside a great sphere, they show that the expected size is linear; and when extending the results to convex disjoint polytopes with bounded aspect ratio and constant complexity, they show that the expected size is linear for polytopes that are sufficiently inside the great sphere, and quadratic for polytopes that are near the boundary of the great sphere.

Although much research has been done on the theoretical aspects of the size of the 3D visibility skeleton, the problem of estimating its actual size in practice with

reasonably large input scenes has remained open. The main reason for this has been the lack of a robust and efficient implementation for conducting the research.

One of the two main goals of this thesis is to provide a robust and efficient implementation to enable empirical studies of the 3D visibility skeleton. The second goal is then to use the implementation to investigate when the 3D visibility skeleton data structure can be of practical interest. For this reason, we study the expected size experimentally, and determine whether the theoretically proven expected linear bound for scenes consisting of spheres also holds for polytopal scenes.

Contributions of this thesis

We first provide a systematic experimental study of the expected size of the 2D visibility skeleton. This is a simpler case than 3D, and there exists software to conduct our experiments. More importantly, analogous to the theoretical result in 3D [27], the expected size of the 2D visibility skeleton on the input of unit discs is known to be linear [41]. Thus, observing a linear behavior in 2D experiments would validate the motivation for our research in 3D. Our experimental results not only confirm the asymptotic linear behavior of the 2D visibility skeleton as a function of the number of unit discs in the input, but also provide an estimation, for a range of different fixed scene densities of discs, of the slope and y-intercept of the linear asymptote. We also estimate the onset, in terms of the number of discs, of the linear behavior.

In the 3D case, we focus on studying a 3D visibility skeleton defined by visual event surfaces, and name it as a *succinct 3D visibility skeleton*. A skeleton thus defined is a subset of the skeleton defined by Durand *et al.* [34, 36], and its size is about 50% to 75% smaller [25, 26]. The reason that we study the succinct visibility

skeleton is because it is the main interest of our research. As a recent result shows, this smaller size data structure can be used to compute direct shadow boundaries cast by polytope light source [25, 26]. Furthermore, as we will show in Chapter 7, we can compute the remaining vertices of the full skeleton from this succinct one efficiently.

We start with an implementation of a sweep algorithm which was initially introduced by Goaoc [50]. This algorithm takes as input a set of disjoint convex polytopes in arbitrary positions, and outputs the vertices of the 3D visibility skeleton. The running time of this algorithm is $O(n^2k^2 \log n)$, where k is the number of input polytopes and n is the number of polytope edges. We recall the brute force algorithm has running time $\Theta(n^5)$ where n is the total number of edges of the input [34].

Slightly different from the sweep algorithm, our implementation takes as input any set of convex polytopes and either outputs the skeleton vertices, or reports that the polytopes are not in general position. By polytopes in general position, it is meant that, for example, no four polytope vertices are coplanar, and no two polytope edges are parallel. To the best of our knowledge, there exists no implementation of the 3D visibility skeleton that handles degeneracies. Our implementation represents an improvement in the sense that we systematically detect and report all degeneracies although the code to handle them remains unwritten. Moreover, our implementation computes the skeleton vertices but does not build the 3D visibility skeleton itself, as the focus of this thesis is on analyzing the size of this data structure. On the other hand, we will propose, in Chapter 7, another method for computing the skeleton from a subset of the vertices.

Our implementation put a lot effort into the design of predicates to gain some com-

putational efficiency. A *predicate* is a function that returns a value from a discrete set; typically a geometric predicate returns answers such as "inside", "outside", or "on the boundary of" a geometric object, and it is typically determined by the evaluation of the sign ("positive", "negative" or "equal 0") of an expression. Evaluating a predicate is often more efficient than computing the exact numerical result of the function that represents the predicate. In our implementation, most of the computational procedures involve the evaluation of a sequence of predicates such as *orientation*, which determines the orientation of four ordered 3D points, or *compare_xy*, which compares the lexicographical order of two 3D points.

Our implementation addresses robustness issues by the choice of number type. We implemented all predicates using the *CGAL Filtered_exact* number type templated with CGAL interval arithmetic (based on `double` number type) and the CORE library [22]. Using *Filtered_exact* number type allows evaluation of the predicates by first using interval arithmetic, and only if this fails, using the CORE exact number type. This ensures that all the predicates are evaluated correctly, and relatively efficiently.

Several predicates that are required by the algorithm have quite high algebraic degrees; this is the case, for example, of those that determine whether four segments admit a line transversal, or those that compare two positions of a sweep plane as it rotates about a line. Since high algebraic degrees may cause an implementation to be prone to errors when using fixed-precision floating-point arithmetic, and may require more memory space and computation time when using exact representation, it is important to study the degree of the predicates. We show that, in the current

implementation, the algebraic degree of the predicate that is used to compare the positions of two sweep planes can be as high as 168. We also show that the degree of these predicates can be decreased to 144 by modifying the current implementation. Finally, we offer some experimental results in this study to show the actual angular separation of two sweep planes that causes the failure of the algebraic degree 168 predicate when using fixed-precision interval-arithmetic.

We use our implementation to conduct experiments on k disjoint polytopes of size n/k on average, with vertices on unit spheres randomly distributed with fixed densities in a given (spherical) universe. We perform these experiments for (i) up to 230 polytopes with up to 1 700 edges and (ii) up to 130 polytopes with up to 9 000 edges. These experiments show that the number of vertices of the succinct visibility skeleton is roughly $C k \sqrt{nk}$, where the observed constant C varies with scene density but remains small (less than 5 in our setting).

This is the first experimentally determined asymptotic estimate of the size of the succinct 3D visibility skeleton for reasonably large n and expressed in terms of both n and k . The results show that the size of the succinct 3D visibility skeleton may be sub-quadratic; in particular, they show a sub-linear growth in n and a sub-quadratic growth in k . Assuming that the size of the silhouette of a polytope on n/k vertices is $O(\sqrt{n/k})$, our results suggest that we may express the size of the succinct visibility skeleton as a function that is linear in the size of the silhouette and quadratic in the number of polytopes; that is, the number of polytope vertices in the scene impacts the size of the succinct visibility skeleton only insofar as it increases the size of the silhouettes. Finally, our results indicate that there is no large constant hidden in the

big-O notation expressions for the size of the succinct 3D visibility skeleton.

Finally, we prove that the knowledge of the succinct 3D visibility skeleton (*i.e.*, the visibility skeleton defined by visual events) is necessary and sufficient to compute the full skeleton (defined by Durand *et al.*), in part or in whole, in almost linear time in the number of vertices computed.

As we discussed before, the visibility skeleton data structure has been used in graphics rendering in both 2D [20, 77] and 3D [33, 34, 36, 37]. While its size may appear to be an impediment to further applications, the detailed experimental studies of its size that we present in this thesis, together with the theoretical results, can provide a good reference for those who wish to use this data structure in their own applications.

In the rest of this thesis, we will present our detailed studies and results. We first provide some background and introduce related work in Chapter 2. We describe our experimental study of the size of the 2D visibility skeleton in Chapter 3. For the study of the 3D visibility skeleton, we present the details of our implementation in Chapter 4, the study of algebraic degree of the predicates that are involved in our implementation in Chapter 5, and the experimental study of the size of the skeleton defined by visual event surfaces in Chapter 6. Chapter 7 presents a method to efficiently compute the full 3D visibility skeleton from the one defined by visual event surfaces. Chapter 8 concludes with a summary of the main results and a discussion of possible future work.

Chapter 2

Background and Related Work

In this chapter, we provide the background material needed for the rest of the thesis, and also, we review the relevant literature. The chapter introduces the concept of the visibility complex (in Section 2.1) and the visibility skeleton (in Section 2.2), and describes the sweep algorithm (in Section 2.3) that is the basis of our implementation. The relevant literature is reviewed in each of these sections.

2.1 The Visibility Complex

Visibility computations are central in applications of computer graphics, robotics, and motion planning. Methods of reducing the expenses of these computations have been actively studied. The visibility complex, a data structure that encodes the visibility information, has been proposed to meet this need. Roughly speaking, this data structure is a partition of the space of maximal free line segments into connected components of segments that touch (*i.e.*, are tangent to, or are blocked by) the same

objects. In comparison with previously defined similar data structures, the visibility graph for example [46], this data structure has certain advantages in that it encodes global visibility information.

The visibility complex data structure was initially proposed by Pocchiola and Vegter as a data structure encoding visibility information of a scene in two dimensions [85]. In 2D, the visibility complex has been extensively studied [84, 91, 54, 6], including some of its application for rendering [77, 20]. Durand *et al.* [34, 38] initiated the study of the visibility complex in three dimensions. Various algorithms have been studied to compute this data structure [35, 62]; however, due to its size and time complexity, applications are so far limited to the use of a simplified version, called the 3D visibility skeleton, which we will describe in Section 2.2.

We introduce the 2D and 3D visibility complex in the following two subsections.

2.1.1 The 2D Visibility Complex

Introduction

As we noted before, the 2D visibility complex was initially introduced by Pocchiola and Vegter [85]. We review this data structure based on their work [85]. The description of the visibility complex we give here is more intuitive though less formal than in [85].

As in the spirit of Pocchiola and Vegter [85], we limit the 2D objects to convex disjoint open sets in general position. Additionally, there is a large circle at infinity enclosing all other objects.¹ The *free space* is thus defined as the complementary

¹Note that this large circle is not part of the input objects. Its functionality is to ensure that each extremity of any maximal free line segment is on some object.

space, with respect to the enclosing circle, of the union of all the objects. A *maximal free line segment* is a line segment that is maximal with respect to the inclusion in this free space. A set of maximal free line segments is *connected* if, roughly speaking, each segment can move continuously, within the set, to the others. More formally, a connected set of maximal free line segments consists of either bounded line segments, half-lines, or lines. In the case of bounded line segments, each line segment is defined by two points in \mathbb{R}^2 , and can be parameterized by a point in \mathbb{R}^4 . Such a set of maximal free line segments is *connected* if the set of corresponding points in \mathbb{R}^4 is connected. Similarly, a half-line can be parameterized by a point and a direction, and a line can be parameterized by a point in Plücker space.

The 2D visibility complex thus defined is a cell complex [85].² All the maximal free line segments that are grouped into the same cell (component) agree on which objects they touch. Furthermore, the line segments in the same cell have 0-, 1- or 2-degrees of freedom, and hence these cells are called vertices, edges, and faces respectively.

- *vertices*. Each vertex of the 2D visibility complex corresponds to a maximal free line segment that is tangent to two objects. Such a line segment has 0-degrees of freedom with respect to that property. Figure 2.1 (a) illustrates two such vertices.
- *edges*. Each edge of the 2D visibility complex corresponds to a maximal connected set of maximal free line segments that are all tangent to exactly one and the same object. This implies that the endpoints of these segments all lie on

²A cell complex, or CW-complex, is, at first sight, a partition of the space into cells of various dimensions that are homeomorphic to open balls such that the boundary of any cell (defined as the image, through the homeomorphism, of the boundary of the corresponding ball) is included in the union of a collection of cells of smaller dimensions. See also [56, 101]) for more precise definition.

the same pair of objects or at infinity. We note that the edge thus defined does not contain their endpoints. Such a maximal free line segment has 1-degree of freedom with respect to these properties. Each edge of the complex is incident to two vertices of the complex. Here we say that an edge is incident to a vertex if the line segment corresponding to a vertex contains a line segment that is a limit of line segments that give rise to the edge. For example, Figure 2.1 (b) shows a component of line segments that are tangent to \mathbf{B} and blocked by \mathbf{G} . These line segments have 1-degree of freedom with respect to these properties, and thus give rise to an edge of the complex. That edge is incident to the vertex arising from a line segment tangent to \mathbf{B} and \mathbf{R} and blocked by \mathbf{G} , and another vertex arising from a line segment tangent to \mathbf{B} and \mathbf{G} . Note that in Figure 2.1 (b), the solid line segment tangent to \mathbf{B} and \mathbf{G} represents a limit of the segments defining the edge. This line segment, together with its extension, illustrated as dotted, is a line that represents the vertex that the edge is incident to. Note also that the other edges and vertices arising from the objects \mathbf{B} , \mathbf{R} and \mathbf{G} are not illustrated in Figure 2.1 (b). Finally, we note that in the work of Pocchiola and Vegter [85], the visibility complex is formally defined in a quotient space, such that a line segment corresponding to the limit of an edge is identified to a line segment corresponding to a vertex.

- *faces*. Each face corresponds to a connected set of maximal free line segments that are blocked by the same two common objects; thus each such line segment has 2-degrees of freedom with respect to this property. A face is incident to an edge (or a vertex) if the line segments in the component that gives rise to

the face can be continuously moved within that component to the line segments that give rise to the edge (or the vertex). For example, Figure 2.1 (c) shows a component of line segments that are blocked by **B** and **G**, and that have 2-degrees of freedom with respect to this property; thus they give rise to a face of the complex. That face is incident to vertices that are arising from line segments tangent to **B** and **G**, **B** and **R**, **R** and **G**; and edges that are arising from line segments tangent to **B** and blocked by **G**, tangent to **G** and blocked by **B**, tangent to **R** and blocked by **B** and **G**. The vertices and edges incident to a face form a cycle.

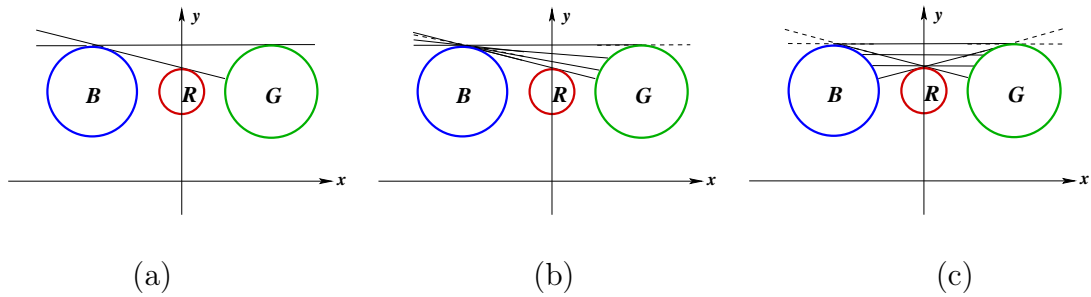


Figure 2.1. Some of the (a) vertices, (b) edges, and (c) faces of the 2D visibility complex of discs **B**, **R**, and **G**.

In the visibility complex, the dimension of a cell corresponds to the number of degrees of freedom of the maximal free line segment giving rise to the cell. As previously noted, in the 2D visibility complex, a vertex is a 0D cell, an edge is a 1D cell, and a face is a 2D cell.

The 2D visibility complex can be better viewed in a dual representation that is expressed in the polar coordinates of a directed line. Here a directed line $y \cos(\theta) - x \sin(\theta) = u$ is expressed by polar coordinates (θ, u) , where θ is the angle the line

forms with the x-axis measured in the counterclockwise sense, and u is the signed distance of the line to the origin. Given an object, for example, disc R in Figure 2.2 (a), and an angle θ , there are two directed lines tangent to the object. As θ varies from 0 to 2π , the motion of the two directed lines that are tangent to disc R appear as two sinusoidal waves in the (θ, u) dual representation, as in Figure 2.2 (b).

These two sinusoidal waves partition the line space into three cells, namely, the space in between the two sinusoidal waves, called *cell I*, the space that is above the two sinusoidal waves, called *cell II*, and the space that is below the two sinusoidal waves, called *cell III*. Any (oriented) line that belongs to *cell I* intersects disc R , whereas any (oriented) line that belongs to *cell II* (respectively *cell III*) leaves disc R to its left (respectively right), and any line on the cell boundary is tangent to disc R .

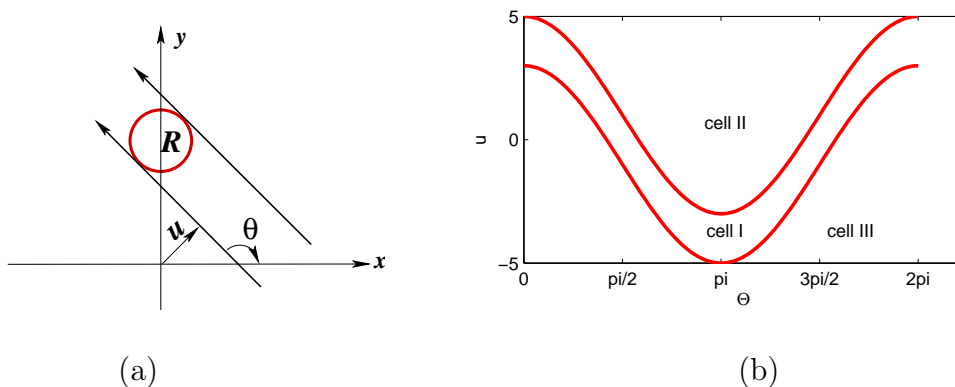


Figure 2.2. Two directed, rotating tangent lines of object \mathbf{R} in (a) Cartesian space, and (b) their dual representation in polar coordinates.

When there are several (more than one) objects in the scene, their tangents in the dual representation appear as several pairs of sinusoidal waves (Figure 2.3). These sinusoidal waves partition the dual space into cells, such that the vertices of the visibility complex map to the intersections of two sinusoidal waves, the edges of

the visibility complex map to the curved segments on the sinusoidal waves that are delimited by two consecutive vertices, and the faces of the visibility complex map to the cells that are delimited by a chain of vertices and edges.

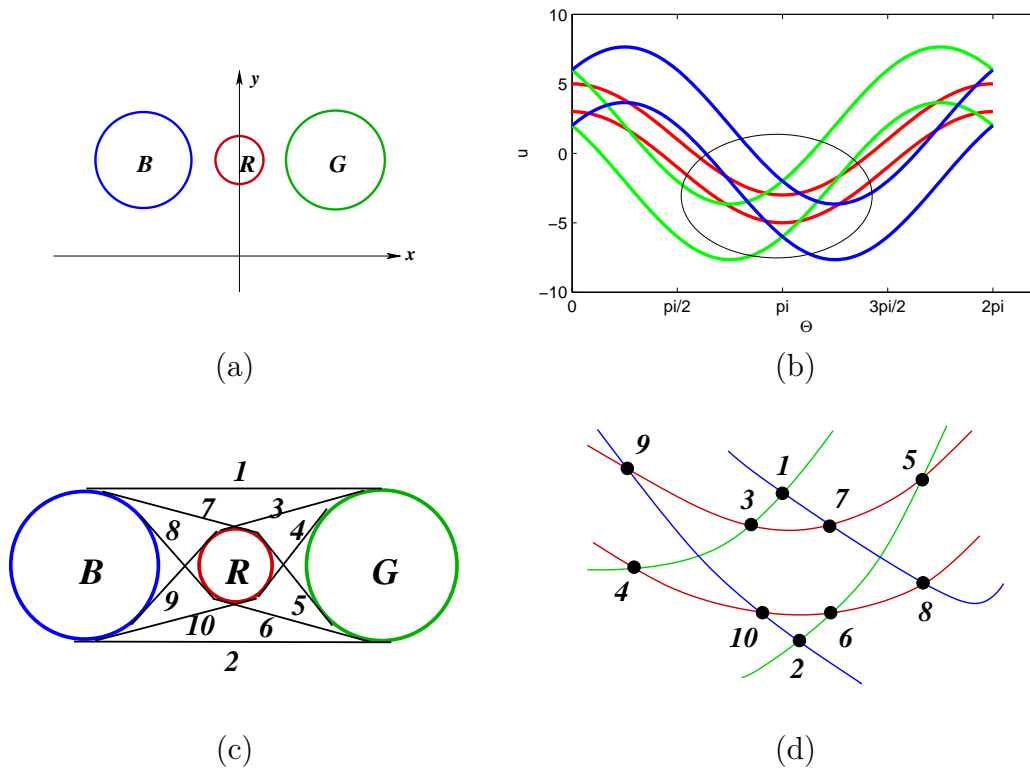


Figure 2.3. (a) and (c) Objects *B*, *R*, *G* in Cartesian space and their bitangents. (b) and (d) Dual representation of directed rotating bitangents.

Figure 2.3 (d) details the circled region in Figure 2.3 (b). Each numbered vertex in Figure 2.3 (d) has its corresponding representation in Cartesian space as shown in Figure 2.3 (c). Figure 2.4 illustrates some of the faces of this example scene in Cartesian space and in the corresponding dual representation.

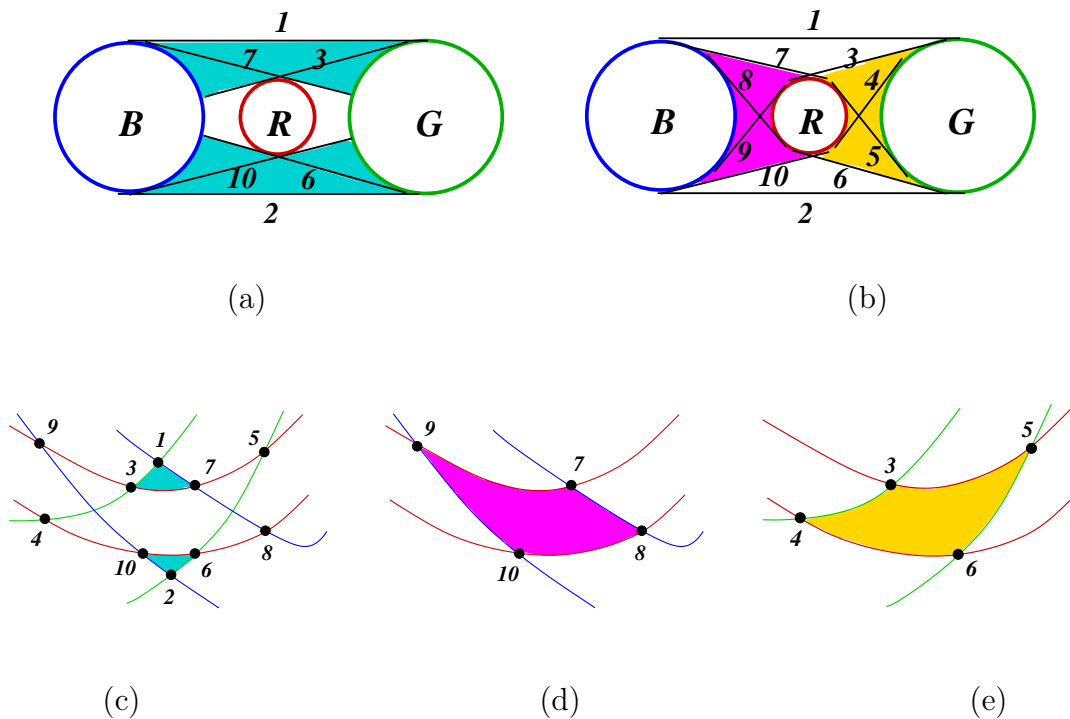


Figure 2.4. Some of the faces of the 2D visibility complex of objects **B**, **R**, **G** (shown as colored regions) and their corresponding dual representation. (a) Two faces that arise from line segments that have common occluders **B**, **G**, and their dual representations in (c); (b) one face that arises from line segments that have common occluders **B**, **R** and its dual representation in (d), and another face that arises from line segments that have common occluders **R**, **G** and its dual representation in (e).

Algorithms and Implementations

Pocchiola and Vegter presented a greedy flip algorithm to compute the 2D visibility complex data structure of an input set of n disjoint (or touching) convex objects of constant complexity. The algorithm runs in $O(n \log n + k)$ time, where n is the number of objects and k is the complexity of the 2D visibility complex [84, 85]. The space complexity for the computation is linear in k in [85], and improved to be linear in n in [84]. The size of k is $\Theta(n^2)$ in the worst case and $\Omega(n)$ in all cases. Briefly,

the algorithm constructs a pseudo-triangulation of the input objects using their free bitangents. Starting from a unit vector at angle 0, they then rotate the unit vector up to angle π . The bitangents that have the same angle as the rotating vector are flipped, and the pseudo-triangulation is updated accordingly.

Angelier and Pocchiola later on implemented the greedy flip algorithm [5]. In particular, they improved the flip operation in the algorithm to constant computation time per flip by means of a “sum of squares” theorem [6]. The implementation takes convex disjoint discs as input, as allowed by the algorithm [84, 85], and it handles objects such as points, segments, and convex polygons, by applying symbolic perturbation to ensure these objects satisfy a smooth boundary condition.

Before Angelier and Pocchiola, Rivière implemented a sweep algorithm to compute the visibility complex of convex disjoint polygons [90]. While this implementation has $O(n \log n + k)$ running time and $O(n)$ space, where n is the total complexity of input polygons and k is the size of the computed 2D visibility complex. Like the greedy flip algorithm, the author shows through experimental results that his implementation is efficient in practice, as the constant of the big Oh notation is small.

Rivière also gave a method for updating the visibility complex in a dynamic polygonal scene in $O(\log n)$ time at each step, after $O(k \log n)$ time precomputation of this data structure [91].

Applications

The 2D visibility complex has various applications.

In [76], Orti *et al.* apply the visibility complex in radiosity computation. The

vertices of the visibility complex are used to compute the discontinuity mesh, and the faces of the visibility complex are used to compute the form factor. Computing radiosity this way can avoid much redundant computation compared to the traditional method.

In [20], Cho and Forsyth use the 2D visibility complex in ray tracing. Two basic properties that they use to render images efficiently are: 1) within the same cell of the visibility complex, the set of rays encounter the same set of objects; 2) radially sweeping a ray in primary, Cartesian space is equivalent to walking along a line segment in the dual space. Their experimental results show that their ray tracer, which uses the 2D visibility complex, is about 3.5 times faster than the conventional ray tracer.

Moreover, Rivière [91] and Hall-Holt [54] make use of the visibility complex data structure to design algorithms for maintaining views in scenes in which the view point moves, but objects are fixed.

2.1.2 The 3D Visibility Complex

Introduction

We introduce the 3D visibility complex based on the work of Durand et al. [34, 38] as it was initially presented.

The 3D visibility complex is an extension of the concept of the 2D visibility complex. However the 3D visibility complex is not a cell complex since some of the cells are not contractible to a point. Moreover a line in 3D has four degrees of freedom, and therefore the cells that partition the 3D line space can have higher dimension

than the cells in the 2D case.

In what follows, we define the cells of the 3D visibility complex. We note that the concept of each cell is based on smooth objects that are in general position. We say 3D objects are in general position when free line segments tangent to any n of them have $4 - n$ degrees of freedom with respect to that property, for $0 \leq n \leq 4$.

- a *vertex* corresponds to a maximal free line segment that is tangent to three or four objects that are in general position (Figure 2.5). In the case of three objects, the maximal free line segment lies on a plane that is tangent to two objects. Such line segment has 0-degrees of freedom. A vertex is a 0D cell.
- an *edge* corresponds to a set of maximal free line segments that are tangent to two or three objects that are in general position, and form one connected component (Figure 2.6). In the case of two objects, the set of maximal free line segments lie on a set of planes that are tangent to the two objects. In such a set, each line segment has 1 degree of freedom. An edge is a 1D cell.
- a *bitangency (respectively tangency) face* corresponds to a set of maximal free line segments that are tangent to two (respectively one) objects that are in general position, and form one connected component. In such a set, each line segment has 2 (respectively 3) degree(s) of freedom. A bitangency (respectively tangency) face is a 2D (respectively 3D) cell.
- a *face* corresponds to a set of line segments that are being occluded by the same two objects, form one connected component. Each of the line segments in such a set has 4-degrees of freedom. A face is a 4D cell.

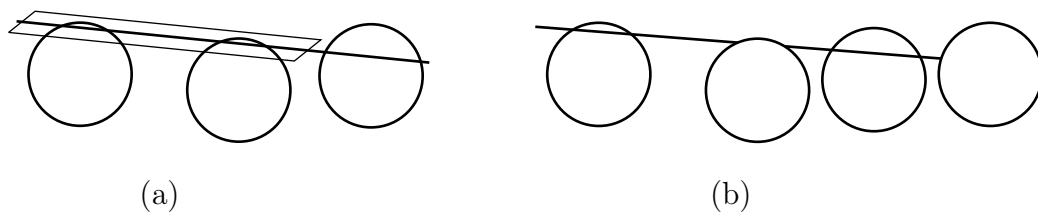


Figure 2.5. The maximal free line segment corresponding to a 3D visibility complex vertex is tangent to (a) three or (b) four objects.

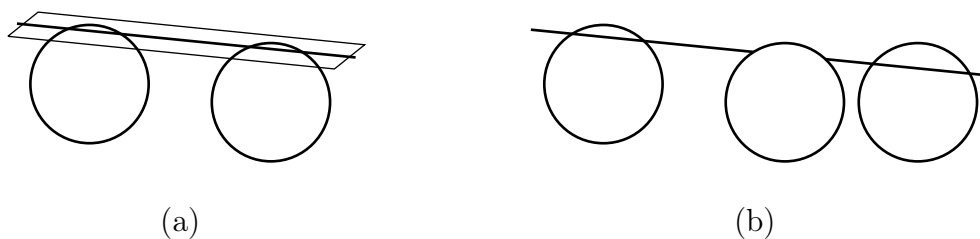


Figure 2.6. The set of maximal free line segments corresponding to a 3D visibility complex edge are tangent to (a) two or (b) three objects.

In the dual representation, a line is expressed by the polar coordinates (θ, φ, u, v) where θ is the azimuth and φ is the elevation. In order to define u and v , we consider the plane perpendicular to the line and passing through the origin. We chose an orthogonal coordinate system u, v on this plane such that the u axis is perpendicular to the y axis. The values u and v in the dual representation are then the coordinates in this system of the intersection of the line with the plane. This definition does not allow representation of a line parallel to the y axis, but the definition is sufficient for many purposes.

Given a sphere \mathbf{L} , as in Figure 2.7, and a direction (θ, φ) , a set of lines that are tangent to sphere \mathbf{L} appear as a circle in the dual representation. When varying θ from 0 to π , the tangency face of \mathbf{L} appears as a curvy tube \mathbf{L} in the dual. Furthermore,

if considering the fourth dimension φ as similar to a time t , then varying φ from 0 to π can be described as the morphing motion of the curvy tube. As in Figure 2.7, this morphing curvy tube \mathbf{L} is the dual representation of the tangency face of sphere \mathbf{L} . It partitions the line space into intersecting, tangent, and non-intersecting lines to sphere \mathbf{L} , depending on whether a given line in its dual representation is inside, on, or outside the curvy tube.

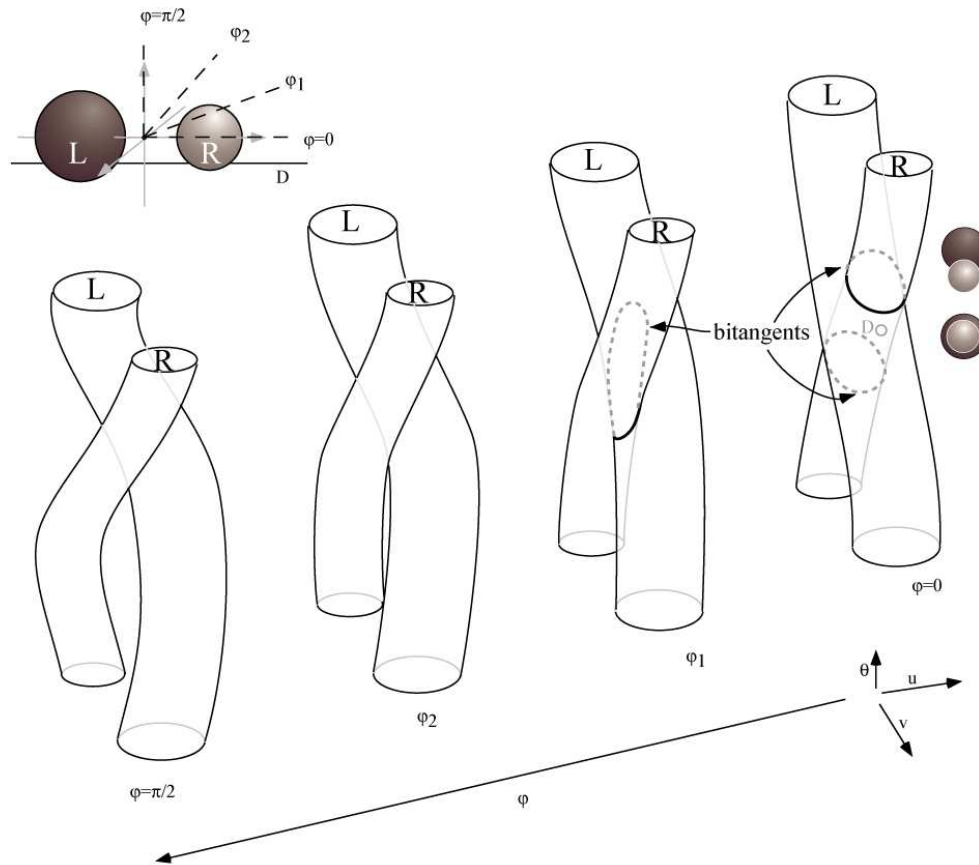


Figure 2.7. Dual representation of the 3D visibility complex of two spheres \mathbf{L} and \mathbf{R} (image credits: Fredo Durand [34]).

Similarly, the tangency face of another sphere \mathbf{R} , as in Figure 2.7, appears as

another morphing curvy tube \mathbf{R} . The intersection volume of these two curvy tubes represents a set of lines that intersect both spheres. In particular, when the surfaces of these two tubes intersect, it results in a 3D curve in the (θ, u, v) coordinate system for a given φ cross-section, and represents a bitangency face. For convenience we call this curve a *bitangency curve*. When two such bitangency curves intersect, their intersection represents an edge. We call this intersection an *edge curve*. This edge curve appears as one point for a given φ in (θ, u, v) coordinates, and appears as a curve when φ varies. Note that the extremities of the bitangency curve, at which the value of θ is minimal or maximal, also result in an edge curve. Such an edge curve corresponds to a set of lines that lie on a set of planes that are tangent to two spheres. When two edge curves intersect at some φ , this intersection defines a unique 4-tuple (θ, φ, u, v) of coordinates, which corresponds to a line tangent to four objects in 3D Cartesian space, and represents a vertex of the 3D visibility complex.

Algorithms and Implementations

Durand *et al.* presented a double sweep algorithm to compute the 3D visibility complex of an input scene consisting of 3D polygons or smooth objects [35]. The running time of this algorithm is $O((v + n^3) \log n)$ where n is the complexity of the input, and v is the number of the vertices of the 3D visibility complex, which is $\Omega(n)$ and $O(n^4)$. Moreover, the authors emphasized that v is much less than $O(n^4)$ in experimental results. Hence the running time of the algorithm in practice appears to be less than the worst case theoretical running time of $O(n^4 \log n)$. Nevertheless, due to the complicated double sweep nature of the algorithm, no implementation appears

to have been done.

Goaoc presented a sweep algorithm to compute the 0D and 1D cells of the visibility complex (see Section 2.3 for detail) [50]. Based on Goaoc’s sweep algorithm, Hornus presented another sweep algorithm to compute the 3D visibility complex of an input scene consisting of disjoint convex polytopes [62]. The running time of this algorithm is $O(n^2k^2 \log n)$, where k is the number of polytopes and n is the number of edges. The author claims that this is the first seemingly implementable algorithm for computing the 3D visibility complex, although there is no implementation available yet.

Applications

The 3D visibility complex was first proposed for applications such as global illumination, kinetic visibility, etc. However, it has so far not been used, due to its complicated data structure.

Instead, a simplified version of the 3D visibility complex, that is, the 3D visibility skeleton, was used for global illumination and has attracted more research attention.

In the next subsection, we introduce the visibility skeleton data structure.

2.2 The Visibility Skeleton

The visibility skeleton data structure is a simplified version of the visibility complex that consists of only the 0D and 1D cells of the visibility complex. This data structure was first introduced in 3D by Durand *et al.* [34, 36] and used in shadow boundary computation [32, 34, 36]. The application of this data structure was successful but limited to only small input scenes, since the practical running time com-

plexity of the algorithm that is used to compute it is relatively high ($O(n^{2.5})$), and the worst-case size complexity of this data structure is large ($O(n^4)$). These apparent limitations have motivated researchers to study its average size and to design efficient algorithms to compute it. The ongoing research of others in this area will be introduced later in this chapter, as indeed, the main part of this thesis also studies the 3D visibility skeleton.

The 2D visibility skeleton data structure was later introduced by Goaoc [50] when designing a sweep algorithm to compute the 3D visibility skeleton.

In the visibility skeleton data structure, a 0D cell is called a vertex and a 1D cell is called an arc. Recall that, in the visibility complex data structure, the 0D cell corresponds to a maximal free line segment that has 0-degrees of freedom; and the 1D cell corresponds to a connected set of maximal free line segments that have 1-degree of freedom. This concept holds in the visibility skeleton data structure as well. Moreover, the incidence relation of vertices and arcs is encoded into a graph, namely, the visibility skeleton graph. We will introduce this data structure in 2D and 3D in detail in the next two sections.

2.2.1 The 2D Visibility Skeleton

We introduce this data structure based on the work of Goaoc [50], but limit ourselves to the case where objects are in general position. By general position, we mean that no three vertices are collinear.

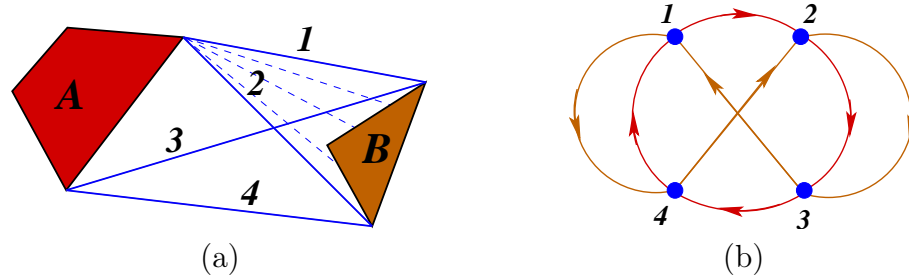


Figure 2.8. (a) The 2D visibility skeleton vertices and arcs, computed from objects **A** and **B**. Four vertices, labeled **1**, **2**, **3**, **4**, are shown as four blue line segments, and an arc, incident to vertices **1** and **2**, is shown as a set of dashed lines. (b) The corresponding 2D visibility skeleton graph of (a); the circular cycle corresponds to the vertices whose corresponding maximal free line segments are tangent to object **A** in clockwise order; and the other cycle is similarly defined, based on object **B**.

2D Visibility Skeleton Vertices and Arcs

In 2D, there is only one type of vertex and one type of arc. Each vertex corresponds to a maximal free bitangent that is tangent to two objects. Each arc corresponds to a set of connected maximal free line segments that are tangent to one given object, and possibly blocked by 0, 1, or 2 other objects. A graphical illustration of the 2D visibility skeleton vertices and arcs is shown in Figure 2.8 (a).

The 2D Visibility Skeleton Graph

In the 2D visibility skeleton graph, each vertex is incident to four arcs, and each arc is incident to two vertices. In particular, the corresponding maximal free line segments of each arc, and its two incident vertices, are tangent to a common object. An example graph is shown in Figure 2.8 (b), which is computed from the two polygons **A** and **B**, as shown in Figure 2.8 (a).

The 2D visibility skeleton graph is a directed graph whose arcs are oriented as

follows. Two adjacent vertices of the graph correspond to two maximal free line segments tangent to a common object. The free line segments are ordered clockwise, and the arc incident to both vertices is oriented accordingly. As in Figure 2.8, the circular cycle of directed arcs gives the ordering of the 4 bitangents around polygon **A**; the cycle of the remaining directed arcs gives the ordering of the 4 bitangents around polygon **B**.

Algorithms, Implementations, and Applications

As we mentioned at the beginning of this section, the 2D visibility skeleton concept was arose during the design of a sweep algorithm to compute the 3D visibility skeleton [50]. In [50], Goaoc uses the same algorithm [84] (presented by Pocchiola and Vegter) and implementation [5] (implemented by Angelier and Pocchiola) of the 2D visibility complex (see Section 2.1 for detail), but discards the information about the faces (2D cells).

2.2.2 The 3D Visibility Skeleton

Unlike the 2D visibility skeleton, the nature of the 3D visibility skeleton varies with the nature of the input objects, *i.e.* whether the contour of the object is smooth or not. Moreover, over the literature history, people have studied the 3D visibility skeleton according to their own research interests, and defined this data structure differently.

In this section, we first introduce this data structure using the 0D and 1D cells of the visibility complex for an input consisting of 3D polytopes, based on the work

of Durand *et al.* [34, 36]. Then we introduce an alternative definition that is based on visual event surfaces, based on the work of Demouth *et al.* [25]. Then finally, we introduce this data structure based on smooth objects.

The 3D Visibility Skeleton of a Set of Polytopes

We first make some preliminary definitions in order to explain the types of vertices and arcs of the 3D visibility skeleton of a set of polytopes.

A *support vertex* of a line is a polytope vertex that lies on the line. A *support edge* of a line is a polytope edge that intersects the line but has no endpoint on it (a support edge intersects the line at only one point of its relative interior). A *support* of a line is one of its support vertices or support edges. The supports of a segment are defined to be the supports of the interior of the segment; thus if a maximal free segment ends at a vertex of a polytope, this vertex is not a support. A *support polytope* of a line is the polytope that the support of the line lies on.

3D Visibility Skeleton Vertices. There are eight types of skeleton vertices. We define them based on [34, 36], and show graphical illustrations in Figure 2.9. Note that unless stated otherwise, no two supports will come from the same polytope. A skeleton vertex has type: *EEEE* if its set of supports consists of four edges; *VEE* if its set of supports consists of a vertex and two edges; *FEE* if its set of supports consists of two edges on one face, and two additional edges; *VV* if its set of supports consists of two vertices; *FF* if its set of supports consists of two edges on one face, and two edges on another face; *FvE* if its set of supports consists of a vertex and an edge on one face, and an edge; *FE* if its set of supports consists of two adjacent vertices of the

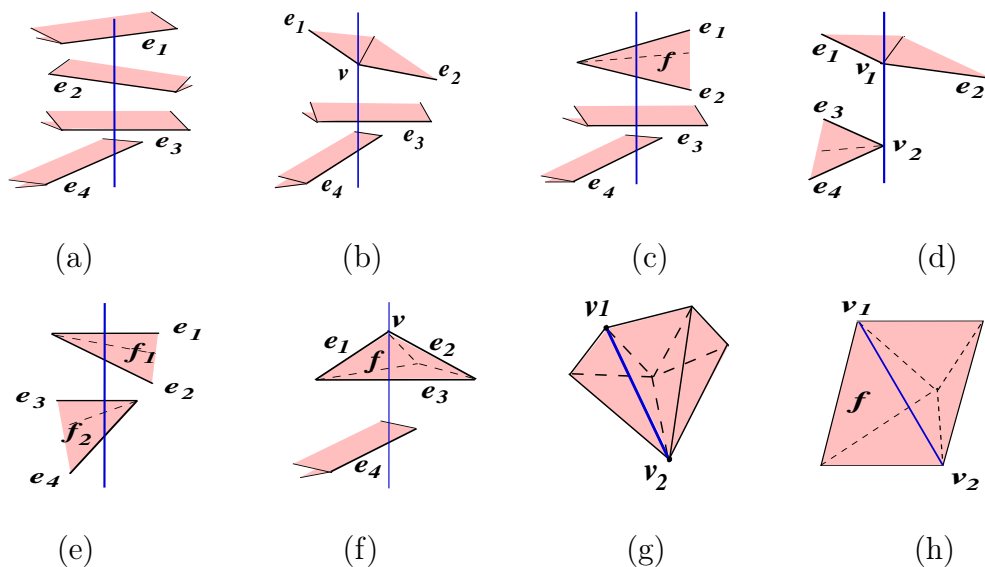


Figure 2.9. The eight types of vertices of the 3D visibility skeleton. (a) $EEEE$, (b) VEE , (c) FEE , (d) VV , (e) FF , (f) FvE , (g) FE and (h) FVV .

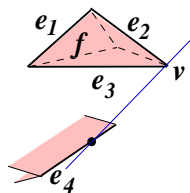


Figure 2.10. The degenerate case of type FvE vertex.

same polytope; and FVV if its set of supports consists of two non-adjacent vertices on the same face of a polytope.

We note a degenerate case of the type FvE vertex, as shown in Figure 2.10. Its set of supports consists of a vertex, a face that the vertex lies on, and an additional edge. Note that in contrast to a non-degenerate FvE vertex, a degenerate FvE vertex has only one edge support. Note also that our definition differs from the discussion in [48], in which this degenerate case of an FvE vertex is not considered to generate a skeleton vertex.

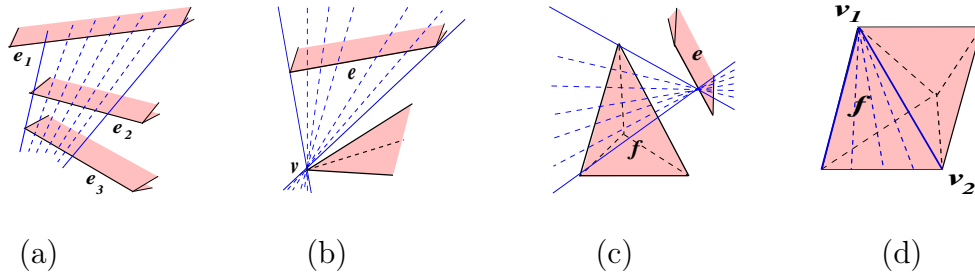


Figure 2.11. The four types of arcs of the 3D visibility skeleton. (a) EEE , (b) VE , (c) FE and (d) FVE .

It should be stressed that the maximal free line segment corresponding to a skeleton vertex is tangent to all its support polytopes.

3D Visibility Skeleton Arcs. There are four types of skeleton arcs. We define them, based on [34, 36], as follows, and show graphical illustrations in Figure 2.11. Note that unless stated otherwise, no two supports will come from the same polytope. An arc has type: EEE if its set of supports consists of three edges; VE if its set of supports consists of a vertex and an edge; FE if its set of supports consists of two edges on one face, and one additional edge; Fv if its set of supports consists of one vertex and one edge on the same face which is not incident to it.

Again, we emphasize that the maximal free line segments corresponding to a skeleton arc are tangent to all their support polytopes.

The 3D Visibility Skeleton Graph. When the inputs are convex disjoint polytopes in general position, each skeleton vertex is incident to three to six skeleton arcs, according to the type of the skeleton vertex; and each skeleton arc is incident to two skeleton vertices. We note that the visibility skeleton graph is not necessarily a connected graph.

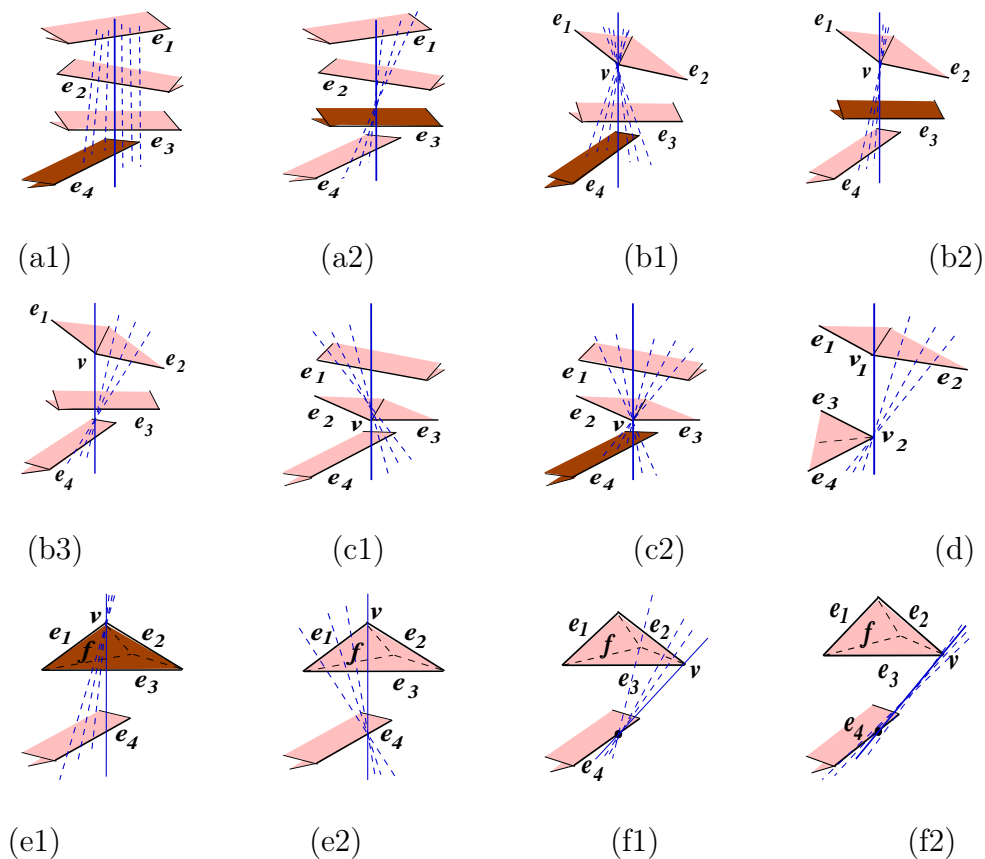


Figure 2.12. The arcs incident to a vertex of type (a) $EEEE$, (b), (c) VEE , (d) VV , and (e), (f) FvE .

An $EEEE$ skeleton vertex has six EEE skeleton arcs incident to it. As in Figure 2.12 (a), two are supported by edges e_1, e_2, e_3 (Figure 2.12 (a1)), two by edges e_2, e_3, e_4 ; and one by edges e_1, e_2, e_4 (Figure 2.12 (a2)), another one by edges e_1, e_3, e_4 . A VEE skeleton vertex has five incident arcs when the polytope vertex is to one side of the two polytope edges; and six incident arcs when the polytope vertex is in between the two polytope edges. In the former case, as in Figure 2.12 (b), two VE arcs have supports v, e_3 (Figure 2.12 (b1)), one has supports v, e_4 (Figure 2.12 (b2)); one EEE arc has supports e_1, e_3, e_4 (Figure 2.12 (b3)), and another one has supports e_2, e_3, e_4 .

		EEE	VE	FE	FVE
$EEEE$		6			
VEE		2	3-4		
FEE		2		3-4	
VV			4		
FvE	non-dg.		1	2	2
	dg.		2	1	
FF				4	
FE					4
FVV					4

Table 2.1. Number of each type of skeleton arc incident to each type of skeleton vertex.

In the latter case, as in Figure 2.12 (c), one EEE arc has supports e_1, e_3, e_4 (Figure 2.12 (c1)), one has supports e_1, e_2, e_4 ; two VE arcs have supports v, e_1 (Figure 2.12 (c2)), and another two are supported by v, e_4 . An FEE skeleton vertex has five or six incident arcs, in a configuration similar to that of a VEE vertex, but its incident arcs are of type FE and EEE . A VV skeleton vertex has four incident arcs of type VE . As in Figure 2.12 (d), one has supports v_2, e_2 , and the other three have supports v_2, e_1, v_1, e_1 , and v_1, e_2 respectively. An FF skeleton vertex has four FE arcs incident to it, with similar configuration as for a VV vertex. A FvE skeleton vertex in its generic case has five incident arcs. As in Figure 2.12 (e), one VE arc has supports v, e_4 (Figure 2.12 (e1)), one FE arc has supports e_1, e_3, e_4 (Figure 2.12 (e2)), another one has supports e_2, e_3, e_4 ; and two FVE arcs have supports v, e_3 on each side of the FvE vertex. The degenerate case of an FvE vertex has three incident arcs: as in Figure 2.12 (f), one FE arc, and two VE arcs. Finally, both FE and FVV skeleton vertices have four incident FVE arcs. We omit their graphical illustrations because of their simplicity.

The preceding paragraph is summarized in Table 2.1.

We finally note that the 3D visibility skeleton thus defined also applies to 3D polygonal input.

The 3D Visibility Skeleton Defined by Visual Event Surfaces

Later on, in their study of shadow boundaries, based on disjoint convex polytopes, Demouth *et al.* [25, 26] take a different approach. They study local changes in the view, *i.e.* surfaces in space such that when crossed by a viewpoint, a new polytope comes into view or a previously seen polytope disappears; in particular, they do not consider the appearance or disappearance of a polytope feature as a change in the view.

For pairwise disjoint convex objects with smooth algebraic surfaces, it is well known that the visual event surfaces for these objects are generated by the maximal free line segments of arcs that are tangent to three objects (Figure 2.6 (b)) or that are tangent to two objects in planes tangent to the two objects (Figure 2.6 (a)) [83]. Demouth *et al.* prove that this also holds for the visual event surfaces of pairwise disjoint polytopes [25], which was previously only a conjecture.

That is, they prove that local changes in the view happen when crossing the surfaces generated by any arc of type EEE , whose set of supports consists of three edges, or any arc of type VE whose set of supports consists of an edge and a vertex that define a plane tangent to both their respective polytopes (excluding arcs of type VE that define a plane not tangent to both support polytopes).

They consequently define a skeleton consisting only of these arcs, and of the incident skeleton vertices. The skeleton vertices include those of type $EEEE$, whose

set of supports consists of four edges; of type VEE , whose set of supports consists of a vertex and two edges; and the vertices of type VV whose set of supports consists of two vertices contained in a plane tangent to both polytope. In this definition, two of the edges supporting a vertex of type $EEEE$ can also be from the same polytope, which corresponds to a vertex of type FEE as defined by Durand *et al.*.

We note that the VE arcs thus defined form a subset of the VE arcs as defined by Durand *et al.*; similarly for the VV vertices.

The resulting visibility skeleton graph is a subset of the visibility skeleton defined by Durand *et al.*, with the following incidence properties. An $EEEE$ vertex has either two or six EEE arcs incident to it. A VEE vertex has two EEE arcs, and between zero and four VE arcs incident to it, and a VV vertex has two VE arcs incident to it.

The 3D Visibility Skeleton of a Set of Smooth Disjoint Convex Objects

In the visibility literature, the vertices and arcs of the 3D visibility skeleton of smooth objects have been studied in various ways, *e.g.* views [88, 89] and visual events [25, 26, 27, 80, 81, 83]. In what follows, we briefly summarize the 0D and 1D cells of the 3D visibility complex introduced in Section 2.1.2 (based on the work of Durand *et al.* [34, 38]), and thus define the 3D visibility skeleton of a set of smooth convex objects.

For a given set of smooth convex objects that are in general position,³ the arcs of the 3D visibility skeleton can be defined by either two or three objects. In the case of two objects, the corresponding connected set of maximal free line segments of an

³Here general position means that the sets of maximal free line segments defining the arcs of the skeletons are one-dimensional (in the space of maximal free line segments), defining the vertices of the skeletons are zero-dimensional.

arc lie on a set of planes that are tangent to the two objects (*e.g.* Figure 2.6 (a)). This type of arc is called $\mathbf{T} + +\mathbf{T}$. In the case of three objects, the corresponding connected set of maximal free line segments of an arc are tangent to the three objects (*e.g.* Figure 2.6 (b)). This type of arc is called $\mathbf{T} + \mathbf{T} + \mathbf{T}$.

If the maximal free line segments of a $\mathbf{T} + +\mathbf{T}$ arc can be continuously moved within the arc to become tangent to a third object, then this gives rise to a vertex of the 3D visibility skeleton of type $\mathbf{T} + +\mathbf{T} + \mathbf{T}$ (*e.g.* Figure 2.5 (a)), to which the $\mathbf{T} + +\mathbf{T}$ arc is incident. Similarly, if the maximal free line segments of a $\mathbf{T} + \mathbf{T} + \mathbf{T}$ arc can be continuously moved within the arc to become tangent to a fourth object, then this gives rise to a vertex of type $\mathbf{T} + \mathbf{T} + \mathbf{T} + \mathbf{T}$ (*e.g.* Figure 2.5 (b)), to which the $\mathbf{T} + \mathbf{T} + \mathbf{T}$ arc is incident.

The two types of arcs and vertices, together with their incidence relations, define the 3D visibility skeleton of a set of smooth convex objects.

Algorithms, Implementations, and Applications

Several brute force implementations have been presented in the visibility skeleton literature.

Durand *et al.* first computed the 3D visibility skeleton data structure using a brute force algorithm, and applied it to shadow boundary computation [34, 36]. In this application, they modeled the 3D scene as 3D polygons (representing the surfaces of the 3D objects) that are in general position. In the case of degenerate input, they modified the inputs by hand to remove the degeneracies. The worst case time complexity of this implementation is $O(n^5)$, and its practical running time is reported

as $O(n^{2.5})$, due to the heuristics they used to speed up the computation. Nevertheless, the application of Durand *et al.* was limited to small input scenes (less than 1500 polygons that are used to describe the surfaces of objects).

Duguet *et al.* used a subset of the vertices of the 3D visibility skeleton to compute shadow boundaries [32]; all the vertices were computed by a brute force algorithm. This application shows an improvement in comparison to the work of Durand *et al.* in the sense that it applies an epsilon parameter to merge the details of the shadow boundaries, which reduces the total computation; however, this method is limited to point light sources only.

Two other brute force implementations have been done, one by Schröder, which represents the input scene as 3D polytopes [94], and one by Graves, which represents the input scene as a set of 3D triangles [47]. The work of Schröder is an extension of Durand *et al.* in terms of the input scenes. The work of Graves is to study the size of the 3D visibility skeleton, but her results are limited to small input size (less than 800 polygons).

Goac presented a sweep algorithm to compute the 3D visibility skeleton when the input scene consists of disjoint 3D convex polytopes [50] (see Section 2.3 for details). This algorithm has complexity $O(n^2 k^2 \log n)$ where k is the number of polytopes and n is the number of edges. The implementation of this algorithm is described later in this thesis (see Chapter 4 for details).

Brönnimann *et al.* also presented a sweep algorithm to compute the *EEEE* vertices of the 3D visibility skeleton when the input consists of possibly intersecting 3D polytopes [14]. This algorithm has the same complexity as the previous sweep

algorithm.

Very recently, using the implementation of this thesis, Demouth *et al.* [25, 26] computed the 3D visibility skeleton defined by visual event surfaces, and applied it to the computation of direct shadows cast by convex polyhedra. The preliminary results have shown that the size of the visibility skeleton defined by visual event surfaces is much smaller than the classical one; hence it may be useful in computer graphics rendering.

2.2.3 The Size Complexity of the Visibility Skeleton

Although the visibility skeleton data structure is simpler and smaller than the visibility complex data structure, its size, especially in 3D, is typically large, which appears to be a limitation for its practical use. This has motivated previous research [14, 27] to study the size of the visibility skeleton data structure.

Size of the 2D Visibility Skeleton

Given n pairwise disjoint objects in 2D, the worst-case size complexity of the 2D visibility complex is $\Theta(n^2)$ [85]. This bound applies to the 2D visibility skeleton as well. Moreover, experimental results [20] on scattered triangle scenes suggested that the actual size of the 2D visibility skeleton can be linear. Everett *et al.* prove that the expected size complexity of the 2D visibility skeleton is linear in the number of the input objects [41].

Size of the 3D Visibility Skeleton

When a 3D scene is modeled by n 3D polygons, the worst-case size complexity of the 3D visibility skeleton is $\Theta(n^4)$ [36]. However, worst-case analysis is often pessimistic. It was pointed out in [14, 27] that the scenes that exhibit the worst-case size complexity are artificial and rarely exist in reality.

Devillers *et al.* have modeled the input scenes as k randomly distributed unit balls, and have shown that the expected size of the 3D visibility skeleton of such scene models is linear in k [27]. Moreover, they also extended their results to input scenes that consist of 3D polygons, or 3D polytopes. When the input scene \mathcal{U} is modeled as a great sphere, and the input objects, by polygons or polytopes that have constant complexity and bounded aspect ratio, and that are uniformly distributed in \mathcal{U} , their result indicates that the expected size the 3D visibility skeleton is linear for those objects that are "sufficiently" inside \mathcal{U} , and is $O(k^2)$ for those "near" the boundary of \mathcal{U} .

Brönnimann *et al.* [14] modeled the 3D scenes as k convex polytopes in arbitrary position, and with n edges in total, and they reported that the worst-case size complexity of the 3D visibility skeleton is $\Theta(n^2k^2)$. Moreover, using the same scene model, and when considering the worst-case silhouette size of the polytopes, Glisse [48] showed a slightly better bound of $O(nk^2\sqrt{nk})$.

We note that the research articles [14, 33, 34, 36, 37, 48], on either worst-case or expected size study of the size of the 3D visibility skeleton, are all based on analyses of type *EEEE* vertices. We recall that type *EEEE* vertices have four polytope supports. The other types of vertices have no more than three polytope supports; thus type

$EEEE$ vertices exhibits the highest theoretical worst-case size complexity and have been the focus of previous size complexity studies. However, in Chapter 6, through experimental study, we show that, in our setting, the size of the 3D visibility skeleton is mainly dominated by type VEE or VV vertices, instead of type $EEEE$ vertices.

2.3 Overview of the Sweep Algorithm

Based on their size complexity study, Goaoc [50] and Brönnimann *et al.* [14] both proposed sweep algorithms to compute certain types of vertices of the 3D visibility skeleton of k convex polytopes with n edges in total, and possibly lying in degenerate position. In particular, Goaoc [50] assumes the polytopes are pairwise disjoint, and the proposed algorithm computes a global visibility skeleton data structure that consists of type $EEEE$, VEE , FEE and VV vertices.⁴ Brönnimann *et al.* [14] extend the input scene to polytopes that are possibly intersecting, and propose a similar algorithm to compute the $EEEE$ vertices. Both algorithms have running time $\Theta(n^2k^2 \log n)$.

We briefly overview the sweep algorithm based on the work of Goaoc [50].

Given k convex disjoint polytopes that have n edges in total, the algorithm sweeps a plane about each edge e of each polytope in turn. The sweep plane is initially coplanar with one face incident to edge e and rotates about edge e until it becomes coplanar with the other face incident to e .

Initially, the sweep plane intersects the input polytopes in a set of polygons, and

⁴Efrat *et al.* [39] presented a similar algorithm for computing not necessarily free isolated transversals in the same setting.

the 2D visibility skeleton of these polygons is computed. This involves computing all the *bitangents*, *i.e.*, the maximal free line segments tangent to two polygons. Generically, a bitangent is tangent to two polygons in the sweep plane at two vertices. Each of these vertices lies on an edge of the input polytopes; these edges are called as the *support edges* of the bitangent. All the bitangents that are tangent to a polygon are sorted in clockwise order.

During the sweep, an event occurs whenever a bitangent appears or disappears, the support edges of a bitangent change, or when there is a change in the order of the bitangents around a polygon vertex. The 2D visibility skeleton of the polygons intersected by the sweep plane is updated according to each of these events. Moreover, a sorted set of *special bitangents*, defined as maximal free line segments going through an endpoint of edge e and tangent to a polygon in the sweep plane, are maintained and updated as well.

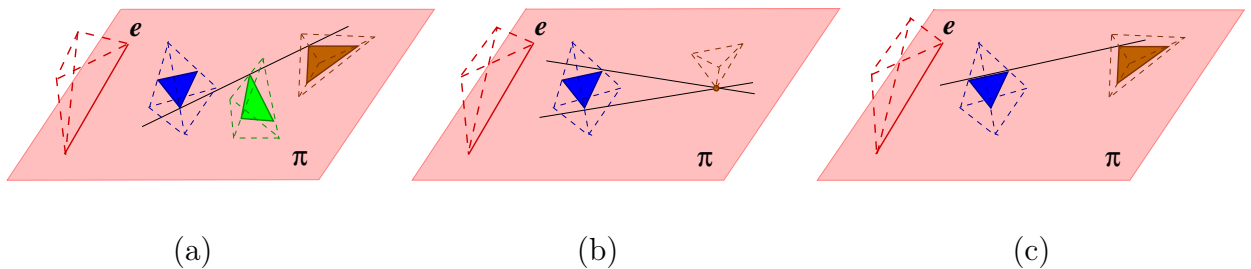


Figure 2.13. (a) T-event, (b) V-event, and (c) F-event.

There are three types of events: the T, V, and F-events (see Figure 2.13). A T-event occurs when two or three bitangents become aligned. At such an event, a new bitangent may be created, or a bitangent may be deleted, and there is a change in the order of the bitangents around a polygon vertex. A V-event occurs when the sweep plane goes through an endpoint of a support edge. If the sweep plane encounters

a polytope for the first (resp. last) time, a set of new bitangents is created (resp. deleted); otherwise, a support edge of a bitangent changes. It should be stressed that there is no V-event when the sweep plane goes through a vertex of a polytope that is not a support of a bitangent in the sweep plane. An F-event occurs when a bitangent becomes coplanar with a face incident to one of its support edges. The bitangent then contains an edge of one of the polygons to which it is tangent. A support edge of the bitangent changes.

All events are computed on-line during the sweep except for the $O(k)$ V-events that correspond to positions the sweep plane that encounter a polytope for the first time. In particular, the T-events are computed as follows. Whenever two bitangents are consecutive around a polygon vertex it is possible that they will eventually become aligned at a T-event. Such a pair of bitangents share a support edge, and thus the pair has three support edges in total, denoted e_1, e_2 and e_3 . Recall that the sweep plane contains an edge e , by definition, and thus any line in a sweep plane intersects (possibly at infinity) the line ℓ_e containing edge e . Hence, a T-event corresponds to a position of the sweep plane such that it contains a line transversal to line ℓ_e and to the three support edges e_1, e_2 and e_3 . Each T-event can thus be computed in constant time by computing the line transversals to these four supports.

The vertices of the 3D visibility skeleton are also computed on-line during the sweep. The *EEEE* vertices, corresponding to segments tangent to four polytopes, are computed as follows. When a T-event is computed, we test whether the corresponding maximal free bitangent intersects edge e between e_1 and e_2 (or e_1 and e_3). If so, a *EEEE* vertex is reported. The *VEE* vertices are simply obtained as T-events involving

a special bitangent. The VV vertices correspond to V -events involving a special bitangent. Finally, the FEE vertices are obtained in the initial or final sweep plane as (maximal free) bitangents intersecting edge e .

The running time of this algorithm is $O(n^2k^2 \log n)$. There are, in the worst case, $\Theta(n^2k^2)$ events in total [14]. They can be computed in $\Theta(n^2k^2 \log n)$ time because each of the events is computed in constant time except for the V -events that correspond to positions the sweep plane that encounter a polytope for the first time; these events can be computed in $O(n)$ per sweep. Now, the insertion and deletion of events in the event queue takes $O(\log k)$ time per event because the event queue has size $O(k)$ at any time. Indeed, there are $O(k)$ disjoint polygons in any instance of a sweep plane and thus $O(k^2)$ bitangents to these polygons; furthermore, each bitangent induces a constant number of possible future events, except for the $O(k)$ V -events that correspond to positions the sweep plane that encounter a polytope for the first time. The $\Theta(n^2k^2)$ events may result in $\Theta(n^2k^2)$ vertices of the 3D visibility skeleton, which are kept in a list. Inserting each computed skeleton vertex in the 3D visibility skeleton costs $O(\log n)$ time. In total, the running time of the sweep algorithm is $O(n^2k^2 \log n)$.

We note that when only estimating the number of skeleton vertices but not constructing the 3D visibility skeleton, the above sweep algorithm [50] has $O(n^2k^2 \log k)$ running time. We also note that, although the algorithm in [14] only computes the type $EEEE$ vertices, it has $O(n^2k^2 \log n)$ running time. This is because the input polytopes in [14] can be non-disjoint, and in the worst case, there can be $O(n^2)$ bitangents on the sweep plane.

Chapter 3

Experimental Study of the Size of the 2D Visibility Complex

In this chapter, we study experimentally the size of the 2D visibility complex of discs and disc-like objects. While the worst-case complexity of the 2D visibility complex is quadratic, experimental results on scenes consisting of scattered triangles strongly suggest that the size of the visibility complex is linear [20]. In addition, theoretical results prove that the expected number of *free bitangents*, *i.e.*, of maximal non-occluded line segments tangent to two discs, among n uniformly distributed, possibly intersecting, unit discs in \mathbb{R}^2 , is linear [41]. Here we carry on a detailed experimental study of the constants in the asymptotic linear behavior of the expected number of free bitangents. We provide experimental estimates on the slope and y -intercept of the asymptote in terms of the density of discs. We also estimate the onset of the linear behavior in terms of the density.

Note that although we study in this chapter the size of 2D visibility complex,

the obtained results apply to the 2D visibility skeleton as well, since the size of both structures is dominated by their 0-D cells, *i.e.*, the non-occluded bitangents, which are the same.

The rest of this chapter is organized as follows. Section 3.1 describes the models of distributions of unit discs we consider in this chapter. We present in Section 3.2 our experiments and the interpolation of the number of free bitangents among random pairwise disjoint unit discs and summarize in Section 3.3.

3.1 Models

We consider *pairwise disjoint* discs for our experiments. The reason for this is because the experimental assessment uses the only known released implementation of the 2D visibility complex that is time efficient (*i.e.*, the one due to Angelier and Pocchiola [5]) and this implementation requires disjoint discs.

In what follows, let $n \in \mathbb{N}$, D_1, \dots, D_n be n unit discs and call p_i the center of D_i . Let also \mathcal{U} (resp. \mathcal{U}^+) be the disc of radius $R > 0$ (resp. $R + 1$) centered at the origin O .

A sample scene of our model is constructed by choosing the n centers of discs one at a time from the uniform distribution over \mathcal{U} with the constraint that each newly generated center is at distance larger than 2 from all the centers already generated. Since we are interested in asymptotic behavior as n increases, we set μ to a constant value and define the radius R of the universe \mathcal{U} to be

$$R^2 = \frac{n}{\mu}.$$

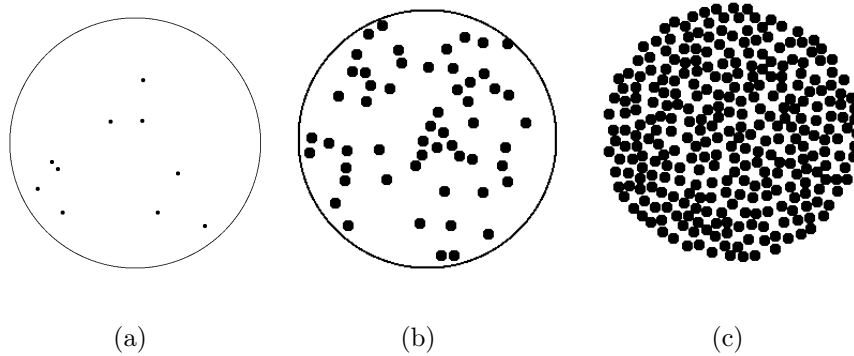


Figure 3.1. Scenes of random disjoint unit discs with density as (a) 0.0025, (b) 0.1, and (c) 0.55.

In this model, the density of discs inside \mathcal{U}^+ (defined as the ratio of area covered by discs to the total area) is

$$\frac{n}{(R+1)^2} \sim \mu \quad \text{when } n \rightarrow \infty.$$

Figure 3.1 shows examples of random scenes for various densities.

Note that we generate random points over a disc of radius R using two uniformly distributed variables $r \in [0, R^2]$ and $\theta \in [0, 2\pi)$ and then taking

$$\begin{cases} x = \sqrt{r} \cos \theta, \\ y = \sqrt{r} \sin \theta. \end{cases}$$

Note also that this distribution is different from the uniform distribution of disjoint discs which would be achieved by generating sets of n centers independently from the uniform distribution over \mathcal{U} until a set is generated in which all the corresponding discs are pairwise disjoint (such a distribution is clearly impractical for generating large and dense scenes).

3.2 Experiments

We first describe the software we used in our experimental study in Section 3.2.1. We then describe our experiments in Section 3.2.2 and finally present our experimental results and their interpretation in Section 3.2.3.

3.2.1 Software

2D Visibility Complex Package. We use the existing software due to Angelier and Pocchiola [5] to compute the 2D visibility complex. This software implemented the greedy flip algorithm [6, 84]. The inputs of this software can be pairwise disjoint bounded convex polygons, discs, and line segments. Its output is the 2D visibility complex. The implemented algorithm has complexity $O(m + n \log n)$, where n is the size of the input, m is the size of the output.

In this experimental study, we use the `Simple_cartesian` kernel and floating point (double) number type of CGAL [19] to compute the 2D visibility complex.

ExpLab. ExpLab is software that facilitates setting up and running experiments, as well as analyzing experimental data. We use this software to manage our experiments.

3.2.2 Setting

With the model defined as in Section 3.1, we measure, for various densities, the number of bitangents in the scene. We also measure the memory usage and the running-time costs of computing these free bitangents.

We run experiments on scenes with up to 4,500 unit discs and density ranging

from 0.0025 to 0.55. We increment the density by 0.0025 for $\mu < 0.025$ and by 0.025 for $\mu \geq 0.025$. We increment the number of discs by 40 up to 1,200 and by 100 after. For small and medium densities, i.e. $\mu \leq 0.01$ and $\mu \in [0.0125, 0.0225]$, we compute the visibility complex for only up to 1,200 and 2,000 discs, respectively, because of memory limitations in the software implementation (see Section 3.3 for further discussion on this issue).

We do not consider densities μ larger than 0.55 because our scene generation scheme fails for such large densities. As Figure 3.1 shows, density 0.55 already implies a fairly dense scene. (Note that Thue proved in 1890 that the best packing of unit discs in the infinite plane is the regular hexagonal tiling – each disc being tangent to six others – and has density $\frac{\pi}{\sqrt{12}}$; thus $\frac{\pi}{\sqrt{12}} \approx 0.91$ is an upper bound for the density of our scenes.)

For each density value and number of discs we consider, we run 10 experiments and report the means of the measures. The standard deviations are very small and we do not report them. We report the number of oriented bitangents, the memory usage in units of kB and the running time in units of 10^{-4} seconds (so that running time, number of bitangents and memory usage can be drawn on the same figure).

Note that the visibility complex package outputs *oriented* bitangents: for each maximal free non-oriented line segment tangent to two discs, the visibility complex implementation outputs two oriented bitangents. Since it is more intuitive to count non-oriented bitangents, we make the distinction between the two in what follows.

All the experiments were done on a i686 machine with AMD Athlon 1.73 GHz CPU running Linux and 1 GB of main memory. We use the *getrusage()* command to

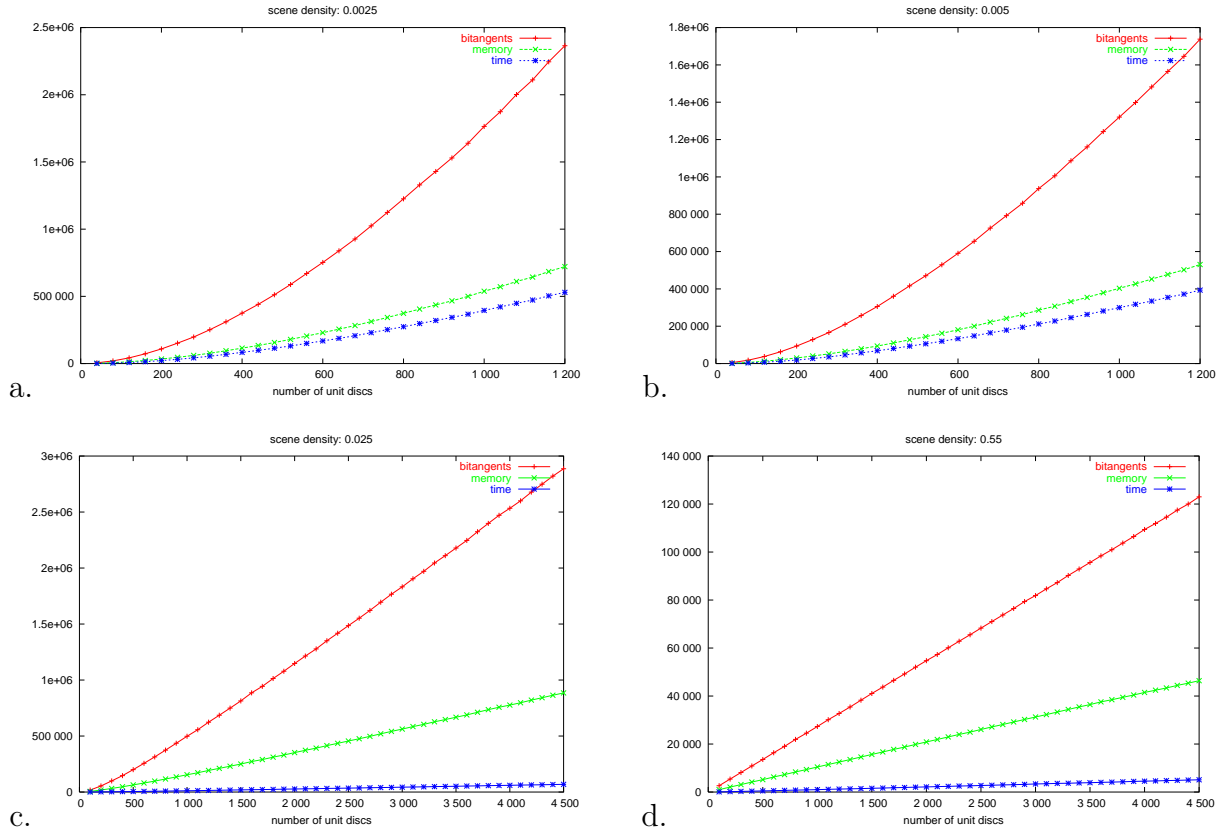


Figure 3.2. Plots of the number of oriented bitangents, memory usage, and running time in terms of the number of unit discs, when scene density is equal to (a) 0.0025, (b) 0.005, (c) 0.025, and (d) 0.55. The unit of the memory usage is kBs, that of the running time is 10^{-4} seconds.

measure user time and *mallinfo()* function to measure memory usage.

3.2.3 Experimental Results and Interpretation

We present here our experimental results. We display in Figure 3.2 the output of our experiments for four representative values of the density (equal to 0.0025, 0.005, 0.025 and 0.55). Figure 3.2 shows quite clearly that the number of oriented bitangents, the memory usage, and the running time have a linear asymptotic behavior in terms of

the number of discs.¹ We note that the slopes of the asymptotes are different for each density μ and are decreasing functions in terms of μ . We also observe that the number of discs at which the linear behavior appears to start is a decreasing function of μ .

In the rest of this subsection, we use least-squares fitting to estimate, in terms of scene density μ and number of discs n , the linear asymptote of the number of oriented bitangents and the onset of this linear behavior. For linear least-squares fitting on a set of p data points (x_i, y_i) , recall that the *correlation coefficient* r , which measures the quality of fit, is defined as

$$r = \frac{p \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{(p \sum x_i^2 - (\sum x_i)^2)(p \sum y_i^2 - (\sum y_i)^2)}}.$$

The closer r is to 1, the better the fit is.

Asymptotic Properties of the Number of Bitangents

For each experimental density value $\mu \in [0.0025, 0.55]$, we estimate the asymptote of the number of oriented bitangents (in terms of the number of discs) using a least-squares fitting on a subset of all the data points, as follows. We compute a least-squares fitting, first using all data points, and then recursively after removing the point corresponding to the smallest number of discs, until the correlation coefficient of the fit of the remaining set of points is larger than some threshold.

We choose the threshold for the correlation coefficient with care. Indeed, a threshold too small would imply that all the data points are always used for the least-squares fitting, which would not be satisfactory for small densities (see for instance

¹Note that the linear asymptotic behavior of the time complexity is only apparent since the time complexity of the Greedy Flip Algorithm is in $\Theta(n \log n + m)$ where m is the size of the output.

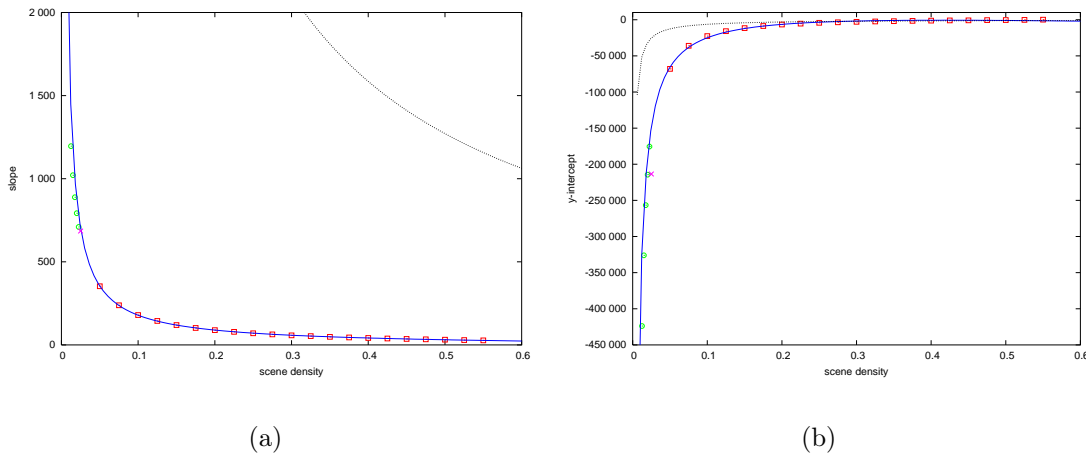


Figure 3.3. The (a) slope and (b) y -intercept, in terms of μ , of the linear asymptote of the number of oriented bitangents (in terms of the number of discs): experimental data points and interpolations (of the square points) by (a) $\frac{17.49}{\mu} + 5.67 - 19.17\mu$ and (b) $-\frac{4,182}{\mu} + 19,255 - 23,789\mu$. The dashed curves are the theoretical upper bounds $(8(\mu + \frac{4\pi^2}{\mu})(n-1))$ [41] times two since the bitangents are here oriented.

Figure 3.2.a). A threshold too large would imply that only two data points are kept for the fitting which is also not satisfactory. In practice, we have a small window for a threshold that is neither too small nor too large. We choose the square of the threshold for the correlation coefficient to be equal to 0.99969.

Figure 3.3 shows the estimated slopes and y -intercepts of the linear asymptotes for the scene densities that are larger or equal to 0.0125 in our experiments. We do not consider the asymptotes for smaller densities because they are not significant; indeed these asymptotes are only estimated by two points because of our choice of the correlation coefficient threshold.

We observe that the extracted slopes and y -intercepts appear intimately related to the inverse of μ . Moreover, the slopes and y -intercepts are bounded theoretically (in a slightly different model where the discs may intersect) by functions of the type

$\frac{a}{\mu} + b\mu$ [41]. We thus try to fit functions of the form $\frac{a}{\mu} + b\mu + c$ to the data points. However, we only interpolate the data points corresponding to densities strictly larger than 0.025 because we are only confident of the quality of the interpolated asymptotes for these densities. The reason for this is that when the density gets strictly smaller than 0.025, the number of points used for estimating the asymptotes drops by more than half because the maximum number of discs used for the experiments drops from four thousand to two thousand, and the minimum number of discs used for interpolating the asymptotes increases to over 800 (see Figure 3.4); hence, for densities in $[0.0125, 0.0225]$, the slopes and y -intercepts are thus estimated with fewer data points (namely between eight and twelve points). We also do not use the points of density 0.025 (the cross in Figure 3.3) because the y -intercept data point seems inaccurate. Note that although they are not used for interpolation, the estimated slopes and y -intercepts for $\mu \leq 0.025$ are used for asserting the quality of the fits.

Using least-squares fitting, we obtain the interpolating functions $\frac{17.49}{\mu} + 5.67 - 19.17\mu$ and $-\frac{4.182}{\mu} + 19,255 - 23,789\mu$ for the slopes and y -intercepts respectively. As Figure 3.3 shows, the data points lie very close to the fitting curves. Moreover, the points corresponding to densities $\mu < 0.025$ lie also quite close to the fitted curves, which is a good hint that our interpolations are satisfactory.

An interesting issue is to determine, as a function of μ , the value n_0 of the number of discs at which the linear asymptotic behavior starts. We choose n_0 to be the smallest value of n used for estimating the asymptote. Figure 3.4 shows the value of n_0 for densities in $[0.0125, 0.125]$; note that we substantially refined the increment of the density for these experiments. We restricted ourselves to these densities because

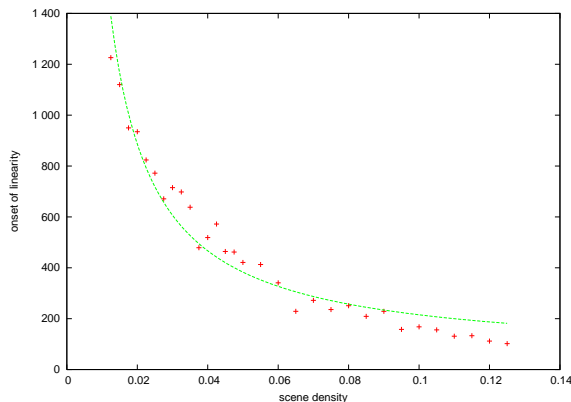


Figure 3.4. Onset of linearity in terms of the density μ : experimental data points and their fitting by $\frac{16.77}{\mu} + 47.55$.

our data is only meaningful in that range in view of our choice of the correlation coefficient threshold. Indeed, outside of it, either only two points or all points are kept for estimating the asymptote.

Fitting these data points by a function of the form $\frac{a}{\mu} + b$, we obtain the function $\frac{16.77}{\mu} + 47.55$. As Figure 3.4 shows, this interpolation is not nearly as good as for the slope and y -intercept of the asymptote. One of the reasons for this is that, for a fixed value of the density μ , the number of bitangents has not been computed for every value of n : there is an increment δn between consecutive data points ($\delta n = 40$ for $n < 1,200$). So the onset n_0 is only accurate up to δn . This impacts the goodness of fit since least-squares fitting is known to be sensitive to outliers. Better results are obtained by linearly interpolating the correlation coefficient between consecutive data points and picking the value of n corresponding to the threshold.

Results. Summarizing, we showed that the number of free non-oriented bitangents (which is exactly half the number of oriented bitangents) in a scene consisting of n

randomly distributed disjoint unit discs is approximated by

$$\left(\frac{8.74}{\mu} + 2.84 - 9.59\mu \right) n - \frac{2,091}{\mu} + 9,628 - 11,895\mu \quad \text{for } n > \frac{16.77}{\mu} + 47.55 \quad (3.1)$$

where μ denotes the density of the scene.

The approximation is good in the sense that, in our experiments, for all the densities and all numbers of discs greater than $\frac{16.77}{\mu} + 47.55$, the error between the observed and estimated number of bitangents is small. More precisely, this error does not exceed 2% for densities in the range $[0.05, 0.55]$. For smaller densities, the error increases to roughly 10% for $\mu = 0.025$ and 30% for $\mu = 0.0125$. For densities less than or equal to 0.01, the number of discs in our experiments is 1,200 which is less than the estimated linear onset and we thus do not have a measurement of the error.

Note that even though the y -intercept of Equation (3.1) is not always smaller than the y -intercept of the theoretical upper bound, that is, $8(\mu + \frac{4\pi^2}{\mu})(n - 1)$ [41] (as hinted in Figure 3.3.b), a straightforward computation yields that the estimated number of free bitangents (Eq. (3.1)) is always less than the theoretical upper bound of for $n \geq 1$. (Indeed, if $F(n)$ denotes the upper bound minus the estimated number of bitangents, as a function of n , both $F(1)$ and the slope of F are positive for all densities $\mu > 0$.)

Analysis for Low Densities

To evaluate the quality of our interpolation for low densities, we ran some specific experiments for density 0.0025 (see Figure 3.1). We implemented a brute force algorithm for computing the number of bitangents which, compared to Angelier's implementation, is extremely slow but, since it merely counts the bitangents without

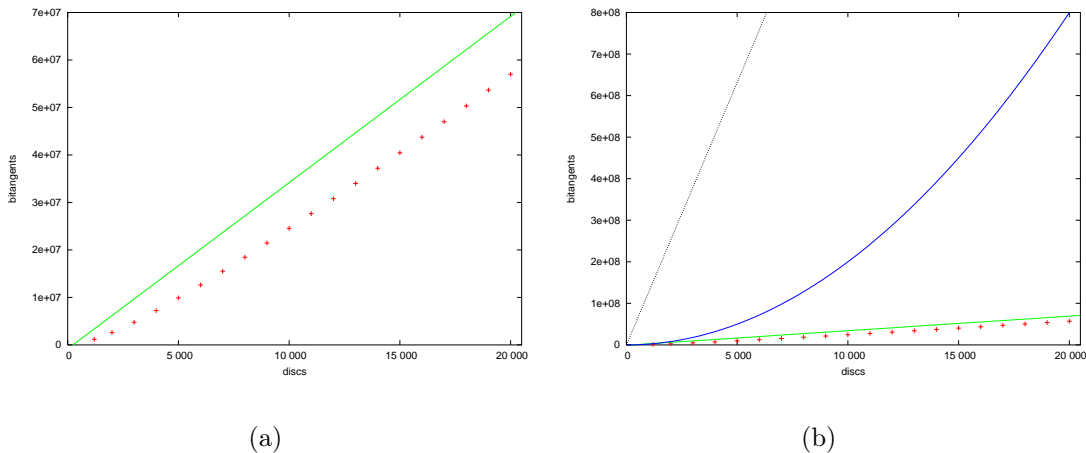


Figure 3.5. Number of non-oriented bitangents for density 0.0025, and an estimate of Eq. (3.1) for $n > 6,755$, with, in (b), the number $4\binom{n}{2}$ of possibly obstructed bitangents and the theoretical upper bound, $8(\mu + \frac{4\pi^2}{\mu}(n-1))$ [41] (in dashed).

storing them, uses no memory and therefore allowed us to compute the number of bitangents for rather large numbers of discs. We ran that experiment on random test scenes from 1,000 to 20,000 discs with an increment by one thousand. The entire set of experiments took over 14 days to compute. Figure 3.5 shows the results of these experiments as well as the interpolated number of bitangents obtained from Equation (3.1): $3,501n - 826,846$ for $n > 6,755$. As Figure 3.5.a shows, the slope of the asymptote of the number of bitangents seems well estimated by Eq. (3.1) but the error on the y -intercept is substantial, leading to an error for the number of bitangents decreasing (strictly) from 34.4% to 17.6% for n ranging from 7,000 to 20,000. However, as Figure 3.5.b shows, the estimate is rather accurate when compared to the theoretical upper bound, $8(\mu + \frac{4\pi^2}{\mu}(n-1))$ [41], or to the number, $4\binom{n}{2}$, of possibly obstructed bitangents.

Analysis for High Densities

The above experimental study focuses on scenes whose density ranges in $[0.0025, 0.55]$. Within this density range, we estimated the asymptotic properties of the number of bitangents in terms of the number of discs. We show here that this estimation is likely to be reasonable even for very large densities.

We consider a hexagonal grid as follows (see Figure 3.6). For any integer $i \geq 1$, the grid \mathcal{G}_i consists of one central hexagon and i rings of hexagons. We set the distance between the centers of adjacent hexagons to be equal to $2(1 + \varepsilon)$. We place one unit disc in each hexagon of the grid and we choose $\varepsilon > 0$ small enough so that any pair of discs that are not on the boundary of the grid admit no free outer bitangent. All the centers of the discs in grid \mathcal{G}_i are contained in a disc of radius $R_i = (1 + 2i)(1 + \varepsilon) - 1$. Let $m_i = 6i$ be the number of hexagons in ring i . The grid \mathcal{G}_i contains $n_i = 1 + \sum_{j=1}^i m_j = 1 + 3i(i + 1)$ hexagons, thus the density of centers in the disc of radius R_i is $\mu_i = \frac{n_i}{R_i^2}$, a decreasing function of i which tends to $\frac{3}{4(1+\varepsilon)^2}$.

The number of *non-oriented* bitangents in \mathcal{G}_i is as follows. Every disc admits 2 inner bitangents with each of its neighboring discs and with no other disc (for ε sufficiently small); furthermore, all discs have 6 neighboring discs except for $6(i - 1)$ discs on the boundary of the grid which have 4 neighbors and 6 discs on the boundary of the grid which have 3 neighbors. Summing, and taking into account that each inner bitangent is counted twice, we get that the number of inner non-oriented bitangents in \mathcal{G}_i is $n_{i-1} \cdot 6 + 6(i - 1) \cdot 4 + 6 \cdot 3 = 6i(3i + 1)$. The discs on the boundary of the grid also admit outer bitangents: the number of outer bitangents between the $i + 1$ discs on one of the six sides of the hexagonal ring is between i (if the discs are in

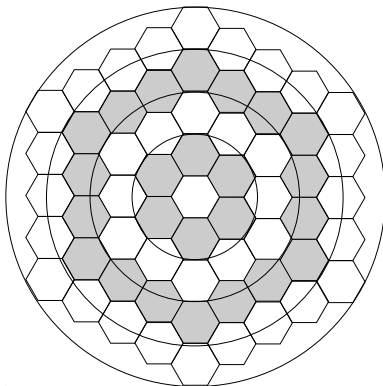


Figure 3.6. Hexagonal scene model (\mathcal{G}_4).

“convex position”) and $\frac{i(i+1)}{2}$ (if the discs are in “non-convex position”). Hence, the total number τ_i of non-oriented bitangents in \mathcal{G}_i is between $6i(3i+2)$ and $3i(7i+3)$.

As can be seen, when i is greater than 25, n_i is larger than 1,951, the density μ_i lies in $\left(\frac{0.75}{(1+\varepsilon)^2}, \frac{0.78}{(1+\varepsilon)^2}\right)$ and the ratio τ_i/n_i lies in $(5.92, 7)$.

For ε sufficiently small, it is reasonable to believe that any scene of n_i unit discs in a disc of radius $R_i + 1$ has roughly the same number of bitangents because the density is high enough that it seems unlikely that scenes may have substantially different combinatorial characteristics.² If this assumption is correct, then the slope of the number of non-oriented bitangents estimated for random scenes should apply. For a density of 0.75, Equation (3.1) gives an estimated slope of 7.3 instead of some value in $(5.92, 7)$ in our analysis. Hence, the estimated slope in Equation (3.1) is reasonably close to the expected slope of the number of bitangents.

²Note that not much is known on optimal disc packing inside a disc; see [53].

3.3 Summary and Bibliographic Notes

We made an experimental assessment of the size of the visibility complex for disjoint random unit discs. Our experiments give a good idea of the asymptotic behavior of the number of bitangents while the theoretical bound is very rough (see Figure 3.5.b). Furthermore, the fact that the estimated asymptotic rate of growth is reasonably small in our random setting indicates that the size of the visibility complex might be tractable in practical, real-world applications. As an example, for a reasonable density of $\mu = 0.1$ (see Figure 3.1) and for $n > 215$ we can expect $90n - 12,500$ bitangents.

It should be noticed that the visibility complex package [5] we used for our experiments is extremely fast (see Figure 3.2) especially compared to a brute force algorithm (see Section 3.2.3). However, unlike the brute force approach, the package uses a substantial amount of memory, which prevented us from running experiments for very low density and very large numbers of discs. These memory limitations are however reasonable since the memory consumption per bitangent (roughly 300 bytes) is substantial but not abnormal considering that the package has not been optimized for memory space. This situation can nonetheless be improved by using the *antichain* feature of the package which, using only $O(n)$ storage, reports the list of free bitangents without storing them in main memory. We unfortunately discovered this feature after all the experiments were completed and did not redo all the experiments. Nonetheless, we observed that this feature allows us to compute, with 1 GB of memory, the number of bitangents among up to 3,500 discs in a scene of density 0.0025 compared to 1,500 discs without using the *antichain* feature.

The work presented in this chapter has first appeared in the International Journal of Computational Geometry and Applications [41].

Chapter 4

An Implementation of the Sweep Algorithm

In this chapter, we describe in detail our implementation of the sweep algorithm [50] for computing the 3D visibility skeleton (see Section 2.3 for details), along with its validation and performance analysis. We will discuss the input and the output of the implementation in Section 4.1 and Section 4.2 respectively; details of this implementation in Section 4.3; software validation in Section 4.5; and finally the performance of this implementation in Section 4.6.

4.1 The Input

While the sweep algorithm [50] discussed in Section 2.3 handles convex disjoint polytopes that can possibly lie in degenerate position, our implementation handles only polytopes in general position. Here general position means that no two sweep

events occur at the same sweep position of a rotating sweep plane (see Section 2.3 for details). This also implies more familiar assumptions such as no two parallel polytope edges, no four coplanar polytope vertices, and no four line segments belonging to a common hyperboloid (see [15] for details). In case of degeneracy, the software will report the type of degeneracy and abort. Extending the implementation to handle degeneracies remains for future work, based on the theoretical analysis of [14, 50].

We note that to our best knowledge, there exists no implementation of the 3D visibility skeleton that handles degeneracies, including the implementation of Durand in which degeneracies were avoided by perturbing the input scenes by hand [34]. Duguet [33] proposed a method for handling degeneracies, but only for computing a section of the visibility skeleton, that is, a set of maximal free line segments that are supported by concurrent lines. Our implementation represents an improvement in the sense that we systematically detect all degeneracies although the code to handle them remains unwritten.

At the current stage of the implementation, a possible way to handle the degeneracies is to avoid them by perturbing the input.

4.2 The Output

Of the eight types of 3D visibility skeleton vertices for inputs consisting of polytopes [34, 36], our implementation computes and outputs type $EEEE$, VEE , and FEE vertices. Moreover, we systematically enumerate VV vertices. The remaining vertices can be computed from these four types of vertices (see Chapter 7 for details), or from the input.

We note that the reason we compute VV vertices through systematic enumeration is because the actual computation, according to our experimental observations, is fairly fast, although the worst-case running time complexity is $\Theta(n^3)$ (n is the total number of edges of the input polytopes). In addition, the implementation becomes much simpler. Of course, computing vertices of type VV using the sweep algorithm is always an option.

We note also that the algorithm presented in [50] computes $EEEE$, VEE , FEE , and VV vertices, whereas the algorithm presented in [14] computes only $EEEE$ vertices.

4.3 Description of the Implementation

As shown in Figure 4.1, this implementation consists of four major components: **Scene Generator**, **Base Classes**, **Sweep**, and **Visualization**. The **Scene Generator** generates input scenes. **Base Classes** defines all the fundamental objects (as in Figure 4.1). The major computations of the sweep algorithm happen in the **Sweep** component, which is the core part of this implementation. Finally, the **Visualization** component displays the algorithm at work, and also serves as a debugging tool.

In terms of software engineering details, this implementation is based on the *CGAL* library [19] (see Section 4.3.1). It also uses the 2D visibility complex package [5, 6], and software which computes line transversals to four lines [87]. This implementation is written in **C++** (since the *CGAL* library is written in **C++**). Roughly speaking, its four major components, **Scene Generator**, **Base Classes**, **Sweep**, and **Visualization**, consist of about 400; 4,000; 6,000; and 800 lines of code respectively. Apart from the other assisting functions, this implementation consists of about 12,000

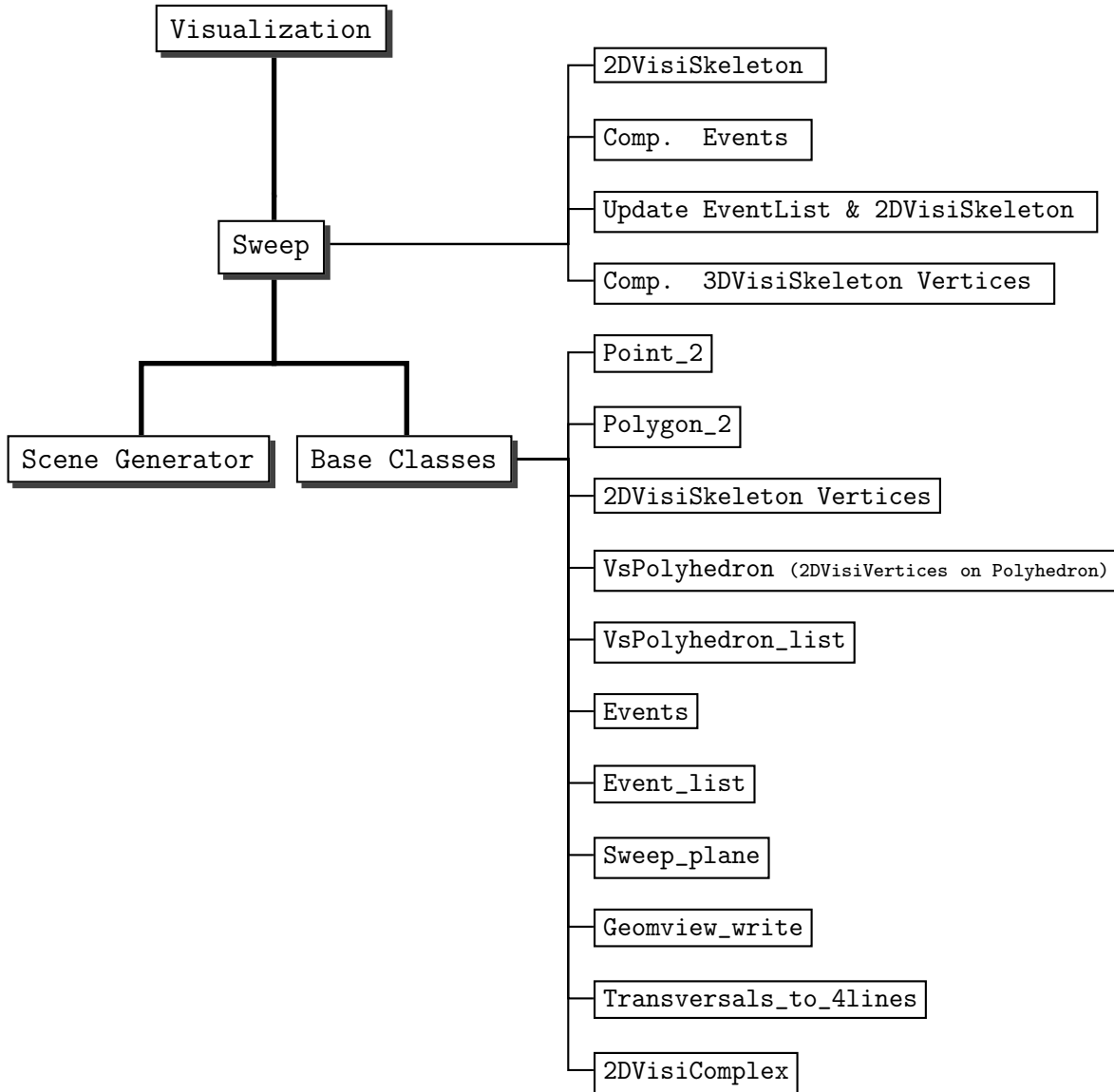


Figure 4.1. Organization of the implementation.

lines of code.

In the rest of this section, we mainly provide a detailed description of the `Sweep` component, since it is the most difficult part of this implementation. We classify this component into four main building blocks as follows: 2D visibility skeleton (de-

scribed in Section 4.3.2), **Computing Events** (described in Section 4.3.3), **Updating Event List and 2D Visibility Skeleton** (described in Section 4.3.5), and **Computing the 3D Visibility Skeleton Vertices** (described in Section 4.3.7). Section 4.3.6 describes the computation of the ordering of the bitangents, which is not a building block of the `sweep` component, but which is related to computing and updating the 2D visibility skeleton. Section 4.3.1 introduces the *CGAL* library and its number types, which is the basis of the four building blocks.

For the **Base Classes**, we provide a description of the `Event_list` in Section 4.3.4. The classes `2D Visibility Skeleton Vertices`, `Polygon_2`, and `Point_2` are briefly described in Section 4.3.2 (**The 2D Visibility Skeleton**); and the `Events` class is briefly described in Section 4.3.3 (**Computing Events**). We omit the descriptions of the rest of the classes since their implementation is straightforward. We however note that the `Transversals_to_4lines` and `2DVisiComplex` classes are the wrap up classes of the 2D visibility complex package [5, 6] and the software for computing line transversals to four lines [87].

We also omit the description of the **Scene Generator** and the **Visualization** components, since their implementation mainly uses the functions provided by *CGAL*.

We finally note that this section is meant to provide a bridge from the theoretical result [14, 50] to the software engineering work. It is primarily focused on the difficult or tricky parts in terms of implementation of the theory, so as to provide some handy references apart from the available software. For this reason, the content of this section is not a translation of the functions or classes of the written software into text description. For further information, the code is available at:

<http://www.cs.mcgill.ca/~lzhang15/webpage/software/software.html>¹

Before we start, to assist in the description, we define the *in-event polytope* as the polytope that supports the event, and the *in-event bitangent* as the bitangent that supports the event.

4.3.1 Preliminaries: The CGAL Library and Number Types

CGAL [19] is an open source library that is implemented in C++. It provides geometric primitive objects, data structures, and many implemented fundamental computational geometry algorithms. Its implementations emphasize efficiency and reliability. Moreover, its extensible kernel allows users to extend its geometric objects and predicates as they need.

One of the reasons we chose to use the *CGAL* library for our implementation is that it provides geometric primitive objects, *e.g.* point, segment, line, polygon, polyhedron, and operations on these objects. We adopted its polyhedron and polygon data structures directly. We also intensively used its existing predicates such as `orientation` of four 3D points, `intersection` of lines (or line segments) with a plane, and so forth. Moreover, we used its existing kernel geometric objects such as 3D and 2D points, lines, segments, vectors, etc. We also extended its 2D point kernel [59]. We note that the *CGAL* geometric objects are parametrized by number types. This allows us to change easily the number types we use in our implementation to conduct performance analysis.

Another reason, and also the most important reason for using the *CGAL* library, is

¹Note that the code uses the version of *CGAL*-3.2.1.

that several number types are available, together with a filtering technique [13] that is based on these number types. The *CGAL* library provides its own number types, such as `MP_Float` for representing multi-precision floating point values, and `Interval_nt` for doing interval arithmetic. In addition, it also incorporates the `C++` built-in number types, and the number types that are defined in the *GMP* [52], *LEDA* [69] and *CORE* [22] libraries.

The classical `C++` built-in floating point number types, such as `double`, have limited precision and therefore, round off the number whenever the number of bits exceeds the precision. Such number types, when directly used in a computational procedure, may cause computation failure, although they are efficient to use. The *CORE* library, on the other hand, provides exact number type representation, that is, it represents the number in a tree that records the entire arithmetic procedure, instead of evaluating the number at each intermediate step. This number type representation can guarantee exact comparison results. However, exact computation is inherently inefficient, especially when division, square root operations, and high algebraic degrees are involved.

We have tested our implementation on `double` and *CORE* number types. The `double` number type is very efficient to use, but causes frequent computational failures with large inputs (more than 50 polytopes, for example). The *CORE* number type, although it provides exact computational results, is so inefficient in our context that it is virtually impossible to use.

The filtering technique [13], which is realized in *CGAL* as the `Filtered_exact` number type, takes advantage of both the floating point and exact number type

representations, and provides exact and efficient computation. It is based on interval arithmetic, which consists of executing relatively fast but imprecise computations (using limited precision number types), which only guarantees that the result is inside some interval. When answering predicates, it is only necessary to know whether a value is positive or negative; so when the whole interval is either positive or negative, this is enough to provide an accurate answer to the predicate in a certified manner. If this method fails, the computation is performed again using the exact number type.

To ensure exact and efficient computation, we chose to use the `Filtered_exact` number type of *CGAL*, templated with the *CGAL* interval arithmetic (based on `double` number type) and the *CORE* library. Using filtered exact computation ensures exact computational results; however, it cannot be as efficient as a number type such as `double`. On random inputs, in our setting, the computation is roughly three times slower than when simply using the `double` number type. But, this is still much faster than using *CORE* exact number type, which is 70 times slower than `double`.²

4.3.2 The 2D Visibility Skeleton

Computation. The initial sweep plane may intersect some input polytopes, resulting in a set of 3D polygons (*i.e.*, their vertices have 3D coordinates) on the sweep plane. We convert the 3D polygons into 2D polygons by dropping one non-trivial coordinate.³ Specifically, for a given 3D polygon, each vertex, which is a 3D point, is converted into a 2D point. We then compute a counterclockwise sequence of ex-

²Note that these estimations are based on the performance of our implementation only; see Section 4.6.3 for details.

³We drop the z -coordinate if the sweep plane is not perpendicular to the $x - y$ plane, and otherwise, we drop the y -, or the x -coordinate accordingly.

treme points from this set of 2D points, using the function `ch_graham_andrew` (based on the Graham scan algorithm [4], whose implementation follows the description of Mehlhorn [72]) provided by *CGAL*, and compute a 2D polygon from this sequence of points.

The 2D visibility complex is computed from this set of 2D polygons by using existing software [5, 6] (see Section 3.2.1 for detail). We furthermore extract the 2D visibility skeleton from the computed 2D visibility complex, and discard the 2D visibility complex. The reasons for doing this are:

1. the tangents (corresponding to the skeleton vertices) in the output of the software we use for the 2D visibility complex are oriented, whereas the orientation information is unnecessary and indeed cumbersome in the visibility skeleton structure;
2. the visibility skeleton structure is more concise and compact, so we can save memory space, which is important because memory space is one of the bottlenecks in our implementation.

This extraction mainly uses the *C++ map* data structure to keep track of the oriented tangents and merge them into the non-oriented ones.

We note that the 2D visibility skeleton is computed only once on the initial sweep plane. When the sweep plane rotates, the 2D visibility skeleton is updated locally according to the sweep events, to save computation time (see details in Section 4.3.5).

Structure. Each of the 2D visibility skeleton vertices corresponds to a bitangent, and encodes the information about the supports of the bitangent, that is, the two

polytopes, the two polytope edges, one on each of the polytope, and the two 3D points that are the intersection of the two polytope edges with the sweep plane. This 3D information is used to compute the sweep events.

For the computation of the 2D visibility skeleton, each of the polygon vertices is a 3D point that is projected into 2D. In order to include within the 2D points the 3D information about the polytope edge supporting the bitangent in the sweep plane, we extended the `Point_2` geometry kernel (this could be done using the `Extensible Kernel` feature of *CGAL* [59]).

Each 2D skeleton vertex has four pointers pointing to its four incident vertices in the graph, distinguished by the two polygons to which it is tangent, as well as the clockwise and counterclockwise orientation of the vertices (see details in Section 2.2.1). These pointers build the incidence relations of the visibility skeleton graph.

Finally, the 2D visibility skeleton uses the `list` data structure to maintain all the skeleton vertices.

4.3.3 Computing Events

For each vertex (corresponding to a bitangent) of the 2D visibility skeleton that is either newly computed or being updated, we compute its potential T-, V-, and F-events. We note that the updates of a 2D skeleton vertex can record the changes of its supports, or the changes of its incident skeleton vertices. In particular, on the initial sweep plane, this computation is done for each of the vertices of the 2D visibility skeleton, since they are all considered as new.

The structure of the events is defined differently according to their type. First,

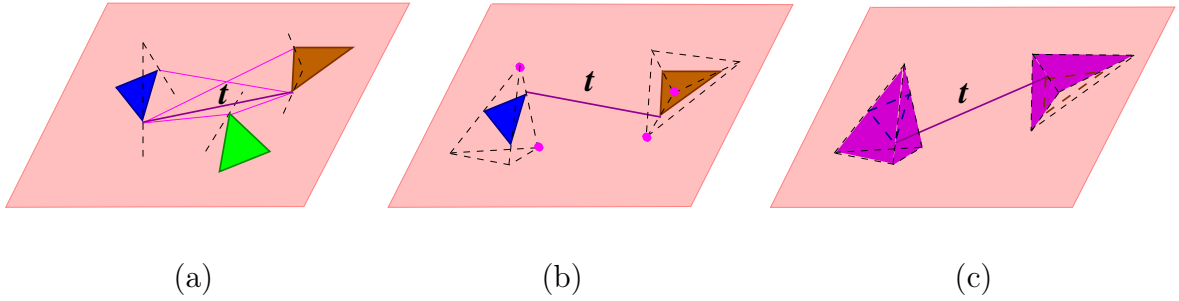


Figure 4.2. Computing potential future events (marked in purple) arising from bitangent t : (a) 4 pairs of potential T-events. (b) 4 potential V-events. (c) 4 potential F-events.

they all include the information of the position of the sweep plane and the type of the event. In addition, the T-event includes the two or three bitangents that are involved in this T-event. The V-event includes the polytope and the polytope vertex at which the V-event happens, as well as the type of the V-event, *i.e.* V-start-, V-middle-, or V-end-event (see details in Section 4.3.5). And the F-event includes the bitangent that is involved in this F-event, and the two polytope edges on which the F-event happens.

In what follows, we describe how each type of event is computed. We first define the *sweep range* to be the range through which the sweep plane rotates, as it rotates about an edge until it hits a face of the polytope to which it is tangent.

Computing potential T-events. Recall that the 2D visibility skeleton is a directed graph. Each bitangent t in a sweep plane Π has a clockwise and counterclockwise neighboring bitangent that is tangent in Π to one of two supporting polytopes of t ; this gives rise to four neighboring bitangents. The bitangent, together with each of its four neighbors, generates four pairs of neighboring bitangents. For each pair of the neighboring bitangents, we compute potential T-events if 1) the two bitangents are

tangent to one common polytope edge, and 2) they are tangent to two other distinct polytopes.

Computing T-events makes use of the three polytope edges to which the pair of neighboring bitangents are tangent, in addition to the polytope edge about which the sweep plane is rotating. The four polytope edges may admit at most two transversals.⁴ Any of the computed transversals corresponds to one possible future T-event. We add the T-event to the event list if: case 1) only one transversal is inside the sweep range; or case 2) the two transversals are both inside the sweep range, in which case we only add the one that is encountered first by the sweep plane, since, when a T-event occurs, the neighboring bitangents will change, and thus the second T-event will not be valid at that stage.

Note that on the initial sweep plane, each of the bitangents is considered as new. To avoid redundant computation, for each bitangent, we consider only its clockwise neighboring bitangents.

Note also that a T-event occurs when either two or three bitangents become colinear. In the latter case, the same T-event may be computed more than once. To avoid redundant computation, before computing each T-event, we check the event list to see if there is any T-event that is computed from the same four polytope edges.

We use the software developed by Redburn [87] to compute transversals to four lines. We omit the computation details here but emphasize that the computational method implemented in this software involves high algebraic degrees (see Chapter 5 for details). This fact directly affects our choice of number type and raises robustness

⁴Note that it is well known that four line segments in general position admit up to two transversals [60] (p.164). In the case of the degeneracies, there might be up to four, or infinitely many transversals to four line segments [15]; this implementation reports any degeneracies and aborts.

issues, as described in Section 4.3.1.

Computing potential V-events. As in Figure 4.2 (b), each of the two polytope edges to which a bitangent is tangent has two incident polytope vertices. Any of these four polytope vertices, if it is inside the sweep range, admits one V-event. Note however that for each polytope edge, if both of its incident vertices are inside the sweep range, only the V-event associated with the vertex that is swept first by the sweep plane is added to the event list, because the other V-event will be computed again later on.

For those polytopes that are not intersected by the initial sweep plane, we compute a V-start-event on each of them. The V-start-event corresponds to the first vertex of the polytope hit by the sweep plane. We add this event to the event list if it is inside the sweep range.

Computing potential F-events. As in Figure 4.2 (c), each of the two polytope edges to which a bitangent is tangent has two incident polytope faces. Any of the four polytope faces, together with the other polytope edge to which the bitangent is tangent, and the polytope edge about which the sweep plane rotating, may admit a transversal. In total, there are possibly four such transversals. Any of the transversals, if inside the sweep range, corresponds to a possible F-event. However, for each polytope edge, if each of its two incident faces admits one F-event, only the one swept first by the sweep plane is added to the event list, because the other F-event will be deleted when updating the first F-event (since the polytope edge is no longer the support of the bitangent).

Note that there are two edges on a polytope face that support the F-event. The old supporting edge is known already, but the new supporting edge needs to be computed. We compute it by enumeration, that is, we enumerate the edges that are incident to the polytope face one after the other until we find the new edge.

4.3.4 The Event List

The computed events are kept in an event list. The actual data structure of the event list is a `skip list`, which ensures $O(\log(x))$ time for search, insertion and deletion operations, where x is the size of the event list.

This event list is a sorted list in which the events are kept in the order in which they would occur during the sweep. We use an `orientation predicate` to sort the events. The `orientation` predicate takes four input 3D points from which it constructs a 4×4 matrix, and returns the sign of the 4×4 determinant of the four points in homogeneous coordinates with the last coordinate equal to 1. The first three input points define a plane, and the sign of the determinant of the matrix tells on which side of the plane the fourth point lies. If we denote by e_1, e_2 the two endpoints of the polytope edge e that supports the current sweep, given two events that are defined by two points v_1, v_2 in \mathbb{R}^3 , the `orientation` predicate is essentially defined as the sign of the determinant $Det(e_1, e_2, v_1, v_2)$. The actual definition is more complicated and requires consideration of a point on the polytope of edge e but not on the edge e ; see Section 5.2.4 for details.

Each of the two 3D points v_1, v_2 that correspond to two events can either be a polytope vertex, an intersection point of a polytope edge with a plane, or a point on

a transversal to four lines. In the latter case, because of the computational procedure, the 3D point inherits high algebraic degree. Thus, designing and analyzing the predicate of sorting the events requires a substantial study. We present the details of this study in Chapter 5.

Moreover, each skeleton vertex has pointers pointing to all the events that are computed from it. Each event also points to all the skeleton vertices that it relates to. These pointers ensure constant access to skeleton vertices or events during updating.

4.3.5 Updating the 2D Visibility Skeleton and the Event List

Initially, the event list contains all the V-start-events, and all the events that are computed from the initial 2D visibility skeleton. When the sweep plane starts rotating, it stops at the event position that is the first in the event list. The 2D visibility skeleton is updated according to the event. Consequently, the event list is updated as well, according to the updates of the 2D visibility skeleton. In particular, the first event in the event list is deleted. The updates of the event list and the 2D visibility skeleton happen interactively and repeatedly until the event list is empty.

We describe the details of the updates as follows.

Updating the 2D visibility skeleton according to the event type.

- T-event: Either two bitangents become colinear and a third bitangent appears at and after the event (Figure 4.3, from (a) to (c)); or three bitangents become colinear, and one bitangent disappears after the event (Figure 4.3, from (c) to (a)). In the 2D visibility skeleton, this corresponds to a skeleton vertex

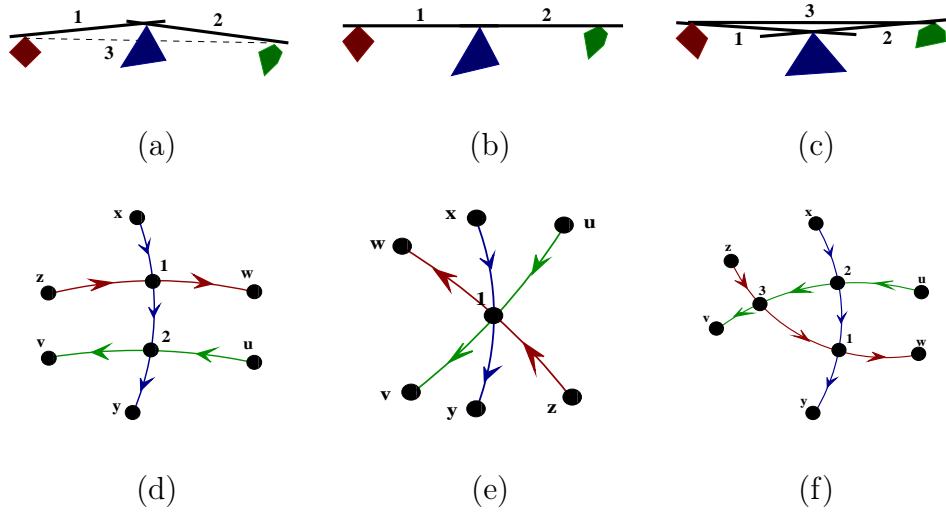


Figure 4.3. A T-event: (a) two bitangents (b) become collinear, and (c) a third bitangent appears; (d), (e), and (f): the 2D visibility skeleton corresponding to (a), (b), and (c).

appearing or disappearing, respectively. We add or delete the skeleton vertex, and update the adjacencies of the three skeleton vertices (Figure 4.3, (d), (e), and (f)).

- V-event: There are three cases related to this event: (i) a V-start-event: the vertex of the in-event polytope is the first vertex of the polytope to be hit by the sweep plane, that is, the sweep plane starts intersecting the polytope at this vertex; (ii) a V-end-event: the vertex of the in-event polytope is the last vertex of that polytope hit by the sweep plane, that is, the sweep plane finishes intersecting the polytope at this vertex; and (iii) a V-middle-event: the sweep plane intersects a vertex of the in-event polytope, such that part of the polytope is above the sweep plane, and part of the polytope is below the sweep plane. We update the 2D visibility skeleton according to these V-start-, V-middle-, and V-end-events as follows:

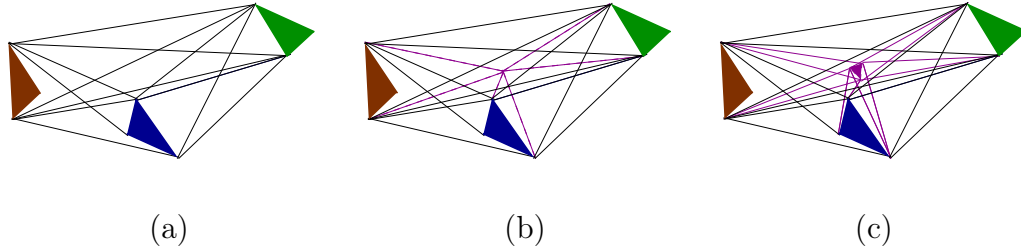


Figure 4.4. (a) before, (b) during, and (c) after a V-start-event, when a set of new bitangents appears.

- (i) V-start-event: a set of new bitangents may appear involving the vertex of the in-event polytope (Figure 4.4). We first compute the set of polygons on the current sweep plane, and then compute the set of new bitangents that are tangent to the vertex and the set of polygons.⁵ These bitangents correspond to the newly appearing 2D visibility skeleton vertices, and we add them to the 2D visibility skeleton. Moreover, for each new skeleton vertex, we compute its four adjacent skeleton vertices, using the method presented in Section 4.3.6.
- (ii) V-middle-event: the visibility skeleton itself undergoes no change; however, a set of bitangents that are tangent to the vertex of the in-event polytope change their supporting edges (Figure 4.5). We update the supporting edges of the in-event bitangents.
- (iii) V-end-event: a set of bitangents that are tangent to the vertex of an in-event polytope disappears. We delete their corresponding 2D visibility skeleton vertices, and update the adjacencies of the remaining vertices.

⁵In the current implementation, we compute the new bitangents using a brute force method, for simplicity. An algorithm for computing them with improved theoretical running time is available [2, 95]. We leave the implementation of this algorithm as future work.

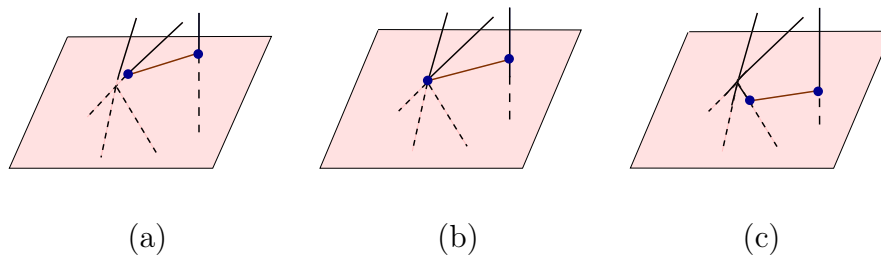


Figure 4.5. (a) before, (b) during, and (c) after a V-middle-event, a bitangent changed its supporting edge.

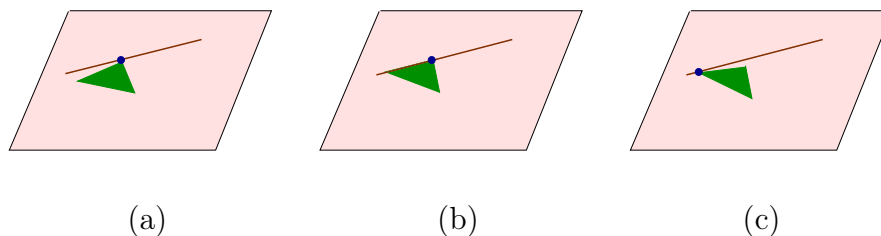


Figure 4.6. (a) before, (b) during, and (c) after a F-event, a bitangent changed its supporting edge.

The V-end-event can be regarded as the reverse of the V-start-event, as shown in Figure 4.4 from (c) to (a).

- F-event: the visibility skeleton itself undergoes no change, but the in-event bitangent changes one of its supporting edges (Figure 4.6), and is updated accordingly.

Updating the event list according to the event type.

- T-event: we first delete all the T-events related to the in-event bitangents, since for each skeleton vertex corresponding to the in-event bitangent, its four incident skeleton vertices may change (see Figure 4.3); therefore, the previously computed T-events may not be valid anymore (see Section 4.3.3 on the computation of potential T-events).

Then, we recompute the T-events according to the updated arcs. Moreover, if a new bitangent appears, we compute the potential F-events involving the new bitangent, and insert the computed new events. Note that we do not compute potential V-events for this particular new bitangent: since its two supporting polytope edges are also the supporting edges of the other two bitangents that are involved in this T-event updating, this means the V-events that are related to these two supporting polytope edges are already computed. However, we need to set the pointers between the new bitangent and its related V-events. If an existing bitangent disappears, we delete all the events that are related to the bitangent from the event list.

- V-event: for a V-start-event, for each of the new bitangents, we compute 4 potential T-events, 2 potential V-events and 2 potential F-events, and insert the computed new events. Also, for each existing old bitangent, if the arcs incident to its corresponding vertex in the 2D visibility skeleton are being updated, we delete all the related T-events from the event list. For a V-middle-event, we delete all the T-, V-, F-events related to the old supporting edge from the event list, and compute new T-, V-, F-events related to the new supporting edge, and insert the newly computed events. For a V-end-event, we delete from the event list all the events related to the disappearing bitangents.
- F-event: similar to a V-middle-event, we delete all the events related to the old supporting edge from the event list, and compute the new events related to the new supporting edge and insert the newly computed events.

In summary, based on the first event in the event list, we rotate the sweep plane

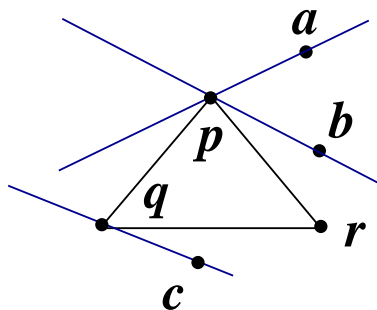


Figure 4.7. Bitangents that are tangent to a polygon at the same or different vertices.

to its next position. At each event position, we update the 2D visibility skeleton and the event list according to the current event type. The sweep ends when there are no more events in the list.

4.3.6 Computing the Ordering of Bitangents

The 2D visibility skeleton is a directed graph (see 2.2.1 for details), with the computed bitangents (2D skeleton vertices) appearing in clockwise order around the polygon to which they are tangent (see Figure 2.8). The ordering information of the bitangents is needed when computing the sweep events, or when updating the 2D visibility skeleton.

A set of bitangents may be tangent to a polygon at the same vertex, *e.g.* bitangents pa and pb in Figure 4.7, or on different vertices, *e.g.* bitangents pa and qc in Figure 4.7. In the former case, we order these bitangents using the same predicate as when ordering the events. In the latter case, we order these bitangents based on the ordering of the polygon vertices.

To decide the ordering of the polygon vertices, in our implementation, we choose to use the predicate `is_clockwise_oriented`, which is an operation provided by the

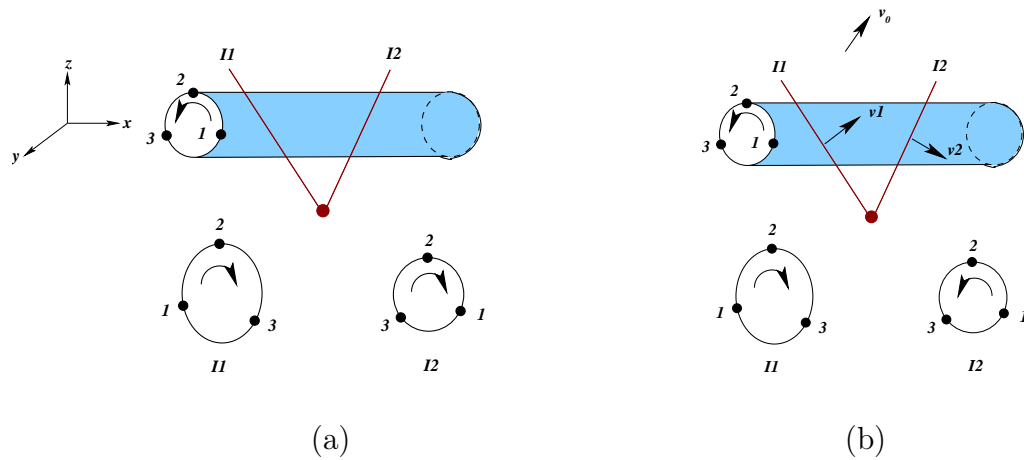


Figure 4.8. (a) Dropping z -coordinate results in different orientations of the two 2D discs: $I1$ and $I2$. (b) Keeping the disc orientation consistent by using the sign of $(v_0 \cdot z) \times (v_2 \cdot z)$.

polygon_2 object. However, the 2D polygons are converted from the 3D polygons by dropping a non-trivial coordinate (in practice, the z -coordinate), and the 3D polygons are computed from the intersection of the sweep plane with the polytopes. It may happen that, for a given polytope, when it intersects a sweep plane at two different rotational positions, the resulting two 2D polygons have different orientations. For example, in Figure 4.8 (a), the two discs $I1$ and $I2$ that result from the intersection of the cylinder with the sweep plane at positions $I1$ and $I2$ have different orientations after dropping the z -coordinate.

During a sweep, the 2D visibility skeleton is updated at each sweep position. This implies that new bitangents may appear, and thus may be added into the 2D visibility skeleton, using the predicate `is_clockwise_oriented`. The different orientations of the 2D polygons that are computed from the same polytope can cause the wrong ordering of the skeleton vertices.

To keep the consistency of the polygon orientations with respect to the polytope

they are computed from, we choose a direction of view, v_0 , which can be the normal vector of the initial sweep plane (Figure 4.8 (b)). During the sweep, we compute the normal vector v_i of the sweep plane at the current sweep position. We use the sign of $(v_0 \cdot z) \times (v_i \cdot z)$ to keep the orientation of the 2D polygons v_i consistent with the initial view direction. For example, in Figure 4.8 (b), the sign of $(v_0 \cdot z) \times (v_2 \cdot z)$ is negative, and we therefore reverse the orientation of disc $I2$, which results in the consistent orientation of discs $I1$, $I2$ with respect to the cylinder from which they are computed.

4.3.7 Computing the 3D Visibility Skeleton Vertices

When a V- or F-event happens, if the supporting line l_t of the in-event bitangent t intersects edge e (about which the sweep plane rotates), we denote that intersection point by $p = l_t \cap e$ (Figure 4.9 (b) and (c)). Similarly, when a T-event happens, if the supporting line l_t of the two collinear in-event bitangents t_1 and t_2 intersects edge e , we denote that intersection point by $p = l_t \cap e$ (Figure 4.9 (a)). Point p may lie in t (or in t_1 or t_2), in which case t itself (or t_1 , t_2 together) is the minimal free line segment associated with the vertex of the 3D visibility skeleton. If, however, p does not lie in t (or t_1 and t_2), as in Figure 4.9, and the line segment between p and the near endpoint of t (or t_1 and t_2) is not blocked, then the line segment between p and the far endpoint of t (or t_1 and t_2) is the minimal free line segment. To define the maximal free line segment, we can extend the minimal free line segment in both directions, stopping if and only if it becomes blocked, or extend it to infinity otherwise. This determines up to two blockers on l_t .

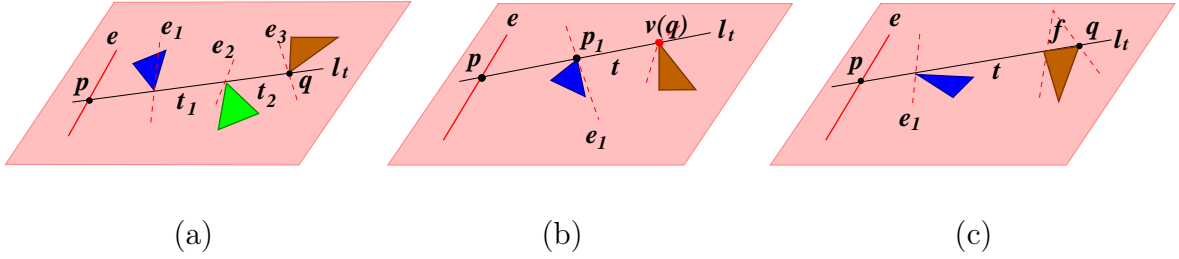


Figure 4.9. The 3D visibility skeleton vertices of type (a) $EEEE$, (b) VEE and (c) FEE that are computed from T-, V-, and F-events. Note that the maximal free line segment lies in l_t , but its extent is not shown in the figure, *i.e.* if it does not extend to infinity, it is blocked beyond the figure.

In the current implementation, we compute only minimal free line segments,⁶ since the main purpose of this thesis is to investigate the size of the 3D visibility skeleton. However, for the application to global illumination, the maximal free line segments are required. These can be computed from the minimal free line segments, with increased computational complexity. From now on, we often refer to the minimal free line segments.

Recall that each minimal, or maximal free line segment corresponds to a vertex of the 3D Visibility Skeleton. A minimal or maximal free line segment computed from a T-event corresponds to a type $EEEE$ vertex; if it is computed from a V-, or F-event, it corresponds to a vertex of type VEE , or FEE respectively (Figure 4.9).

Given that the sweep algorithm rotates a sweep plane around each polytope edge in turn, a minimal free line segment may appear in more than one sweep and thus may be computed more than once. For example, in Figure 4.9 (a), the minimal free line segment pq appears in those sweeps which the sweep plane rotates about polytope edge e , e_1 , e_2 , or e_3 . In what follows, we describe how we compute the $EEEE$, VEE ,

⁶Precisely, we only compute the line segment defined by t (or t_1 and t_2) without computing its blockers. As a consequence, we have no knowledge about how far the minimal free line segment can extend.

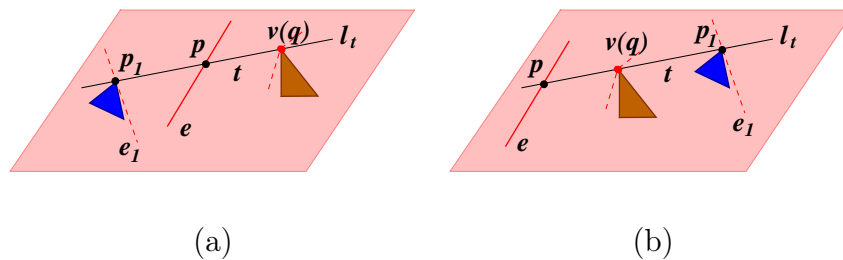


Figure 4.10. Computation of a *VEE* vertex, in two cases different from Figure 4.9 (b).

and *FEE* vertices, and how we avoid redundant computation.

EEEE vertices. Consider four edges e_1, e_2, e_3, e_4 that support a T-event; see Figure 4.9 (a). As the algorithm sweeps a plane about edges $e = e_1, e_2, e_3, e_4$ in turn, there will be two sweeps such that $p = l_t \cap e$ is an internal point of the minimal free line segment, and two sweeps for which it is an endpoint of the minimal free line segment. To avoid redundancy, we output an *EEEE* vertex if and only if i) $p = l_t \cap e$ is an internal point p_i on the minimal free line segment; and ii) compared to the other internal point $p_j = l_t \cap e_j$, edge e_i contains the polytope vertex that is lexicographically first among the four polytope vertices on e_i and e_j .

VEE vertices. In this case, the bitangent t has a polytope vertex v as one endpoint and its other endpoint lies at a point p_1 on some edge e_1 . Let $p = l_t \cap e$ where e is the edge about which the sweep plane rotates (see Figure 4.9 (b), also Figure 4.10).

If v is extremal in the ordering of p, p_1 , and v on l_t , and if p lies between p_1 and v (Figure 4.10 (a)), then the bitangent t is a minimal free line segment associated with a *VEE* vertex and we output this vertex. If p_1 lies between p and v (Figure 4.9 (b)), then to avoid redundancy, we do nothing (the *VEE* vertex is output when the sweep plane rotates about e_1).

If v is not extremal and v lies between p and p_1 on l_t (Figure 4.10 (b)), then pp_1 is a minimal free line segment associated with a V-event if and only if pp_1 is not blocked. We check in a brute force manner whether pp_1 is blocked. To avoid redundancy, we check only once whether line segment vp is blocked, that is, when e contains the polytope vertex that occurs first in lexicographic order among the four endpoints of edge, e and e_1 .

FEE vertices. Consider the minimal free line segment associated with an *FEE* vertex, and let l_t denote its supporting line. To describe the computation of the *FEE* vertex, let us view the minimal free line segment associated with the *FEE* vertex and l_t in the plane π_f of the face f associated with the *FEE* vertex. Let e_1 and e_2 denote the edges of f intersected by l_t , and let e_3 and e_4 denote the remaining two support edges of the minimal free line segment associated with the *FEE* vertex. Let $p_i = l_t \cap e_i$, $1 \leq i \leq 4$. As shown in Figure 4.11, either (i) $l_t \cap f$ does not lie between p_3 and p_4 on segment p_3p_4 , or (ii) $l_t \cap f$ lies between p_3 and p_4 . In Case (i), we output the *FEE* vertex as the sweep plane rotates about the edge e_3 . This has the advantage that the minimal free line segment p_1p_4 appears as a bitangent during the sweep, so that there is no need to further test if the minimal line segment is free. Computationally, this is equivalent to testing, when the F-event arising from face f occurs during a sweep, if the associated bitangent t intersects the edge e about which the sweep plane rotates. In Case (ii), we output the *FEE* vertex when the sweep plane begins or ends its rotation about the edges e_1 or e_2 of face f , whichever contains the polytope vertex that is lexicographically first among the four endpoints of e_1 and e_2 . This avoids redundancy and having to check for blockers, since the minimal free line

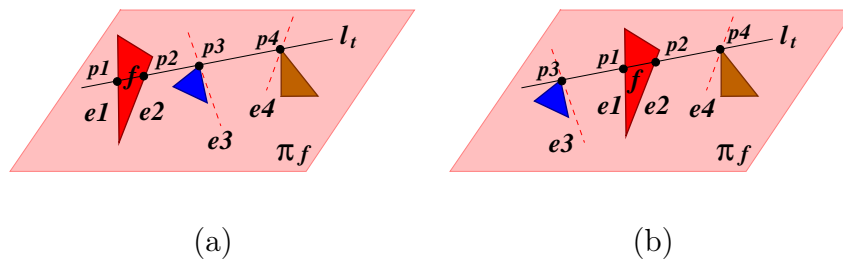


Figure 4.11. (a) Case (i) and (b) Case (ii) of computing an *FEE* vertex.

segment p_3p_4 appears as a bitangent.

4.4 Complexity of the Implementation

As analyzed in Section 2.3, for k given polytopes with n edges in total, the worst-case complexity of the sweep algorithm is $O(n^2k^2 \log k)$, which is dominated by computing and processing T-events. To simplify the implementation of the algorithm, the following two operations are implemented using a brute force method. We analyze their complexity as follows.

Computing the free bitangents at each V-start-event. The worst-case complexity of this operation is $O(n^3k)$. Indeed, at each V-start-event, there are $O(k)$ polygons with total complexity $O(n)$ on the sweep plane. From a single point, the operation of computing all the tangents to the $O(k)$ polygons costs $O(n)$ time, and testing non-occlusion of each computed bitangent costs $O(n)$ time. There are $O(k)$ V-start-events for each sweep and there are n sweeps in total.

Computing *VEE* vertices when, for a given V-event, the polytope vertex V lies between the two polytope edges. As discussed in Section 4.3.7 (the case

of a *VEE* vertex) and illustrated in Figure 4.10 (b), we check in a brute force manner, for this particular case, to see whether a V-event implies a *VEE* vertex. Again, the worst-case complexity of this operation is $O(n^3k)$. There are $O(k)$ polygons with total complexity $O(n)$ on the sweep plane, the operation of testing non-occlusion of the corresponding minimal segment of a *VEE* vertex is $O(n)$, and there are $O(n^2k)$ *VEE* vertices.

Note that, through our experimental testing, we observed that the proportion of the V-events in this particular case is very low in comparison with the rest of the V-events. As a result, we observed that the brute force implementation for this particular case has almost no effect on the overall actual running time of our implementation.

Total running time complexity. In total, the running time complexity of this implementation, in the worst case, is $O(n^3k) + O(n^2k^2 \log k)$, which is equivalent to $O(n^2k(k \log k + n))$.

4.5 Software Validation

We validated our software through visualization and experimental verification, which are explained in detail in Section 4.5.1 and 4.5.2.

The input scenes that were used to test our software were generated artificially. Briefly, we generated the input scene by first generating k uniformly distributed disjoint unit spheres in a spherical universe with density μ . For each sphere, we then uniformly generated a set of vertices inside the sphere and computed their convex hull. We show three sample scenes that are generated from this scene generating

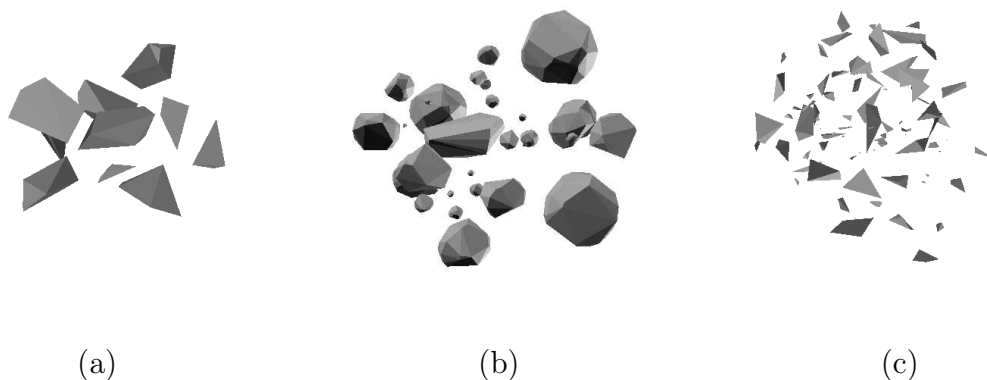


Figure 4.12. Sample scenes of (a) *Scene I*, (b) *Scene II*, (c) *Scene III* .

schema in Figure 4.12. *Scene I, II, III* has 10, 30, 90 polytopes, and 129, 2964, 660 total edges respectively.

4.5.1 Visualization

The input to our implementation is a set of polytopes, and the output is the vertices of types $EEEE$, VEE and FEE of the 3D visibility skeleton. At each intermediate step, i.e. at each event position, the computed 2D visibility skeleton undergoes certain updates. We used geomview [45] to display the input, the output, and the updates of each intermediate step to verify the correctness of the program through visualization.

This method is limited to a small number of input polytopes. When the number of input polytopes is large, at each intermediate step, the computed 2D visibility skeleton contains a large number of vertices, and as a result, the output tends to have a large number of vertices. In both the 2D and the 3D visibility skeleton displays, the vertices appear on the display window as a large number of corresponding line segments,

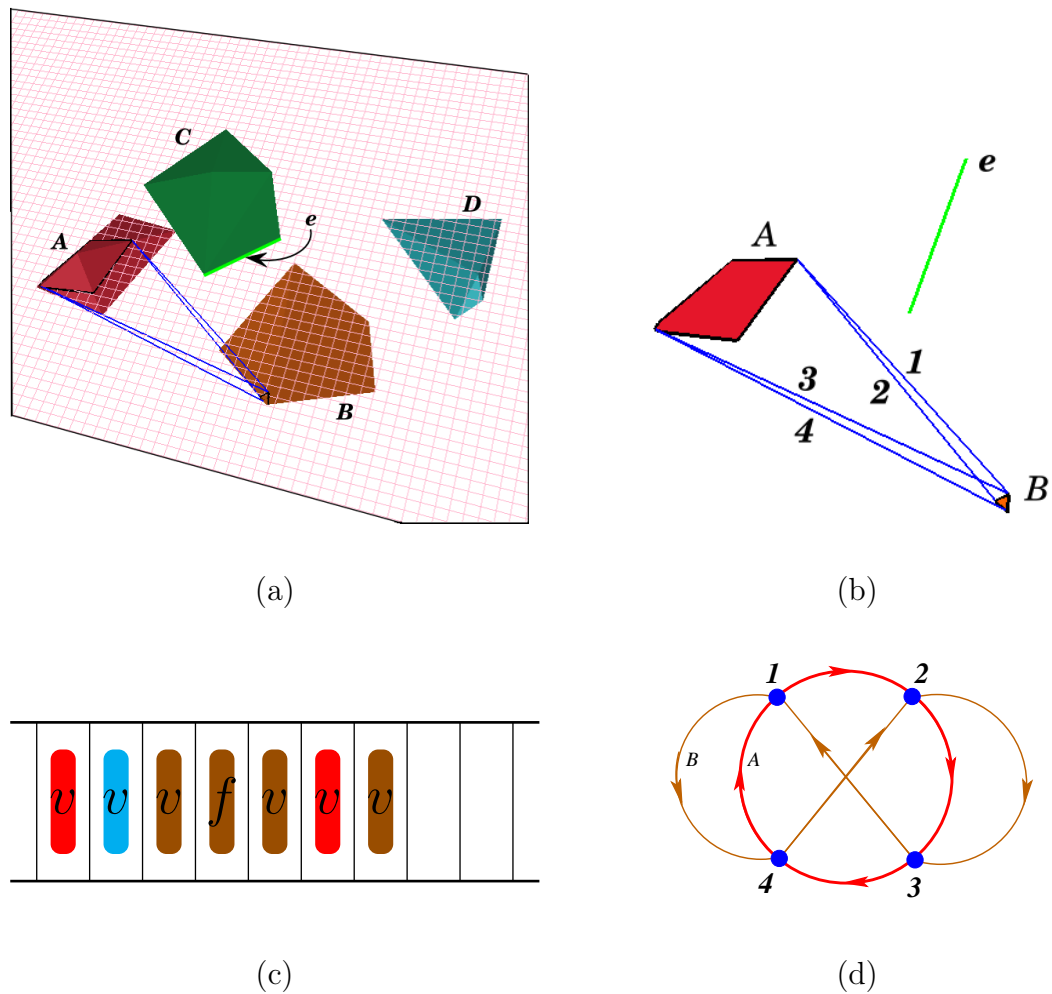


Figure 4.13. (a) One position of the sweep plane. (b) The view inside the sweep plane. (c) The eventlist, and (d) the 2D visibility skeleton for polygons in (b).

which makes the verification difficult. However, when the number of input polytopes is small, this visualization method can serve as partial evidence for verification.

An example of visualizing the computational step and the computational results with 4 input polytopes was made into a video [107], which is available at:

<http://www.computational-geometry.org/SoCG-videos/socg07video/Visibility.mov>

The frames of the video were mostly based on the snapshots of the output of our implementation at its various steps. When displaying the intermediate computational

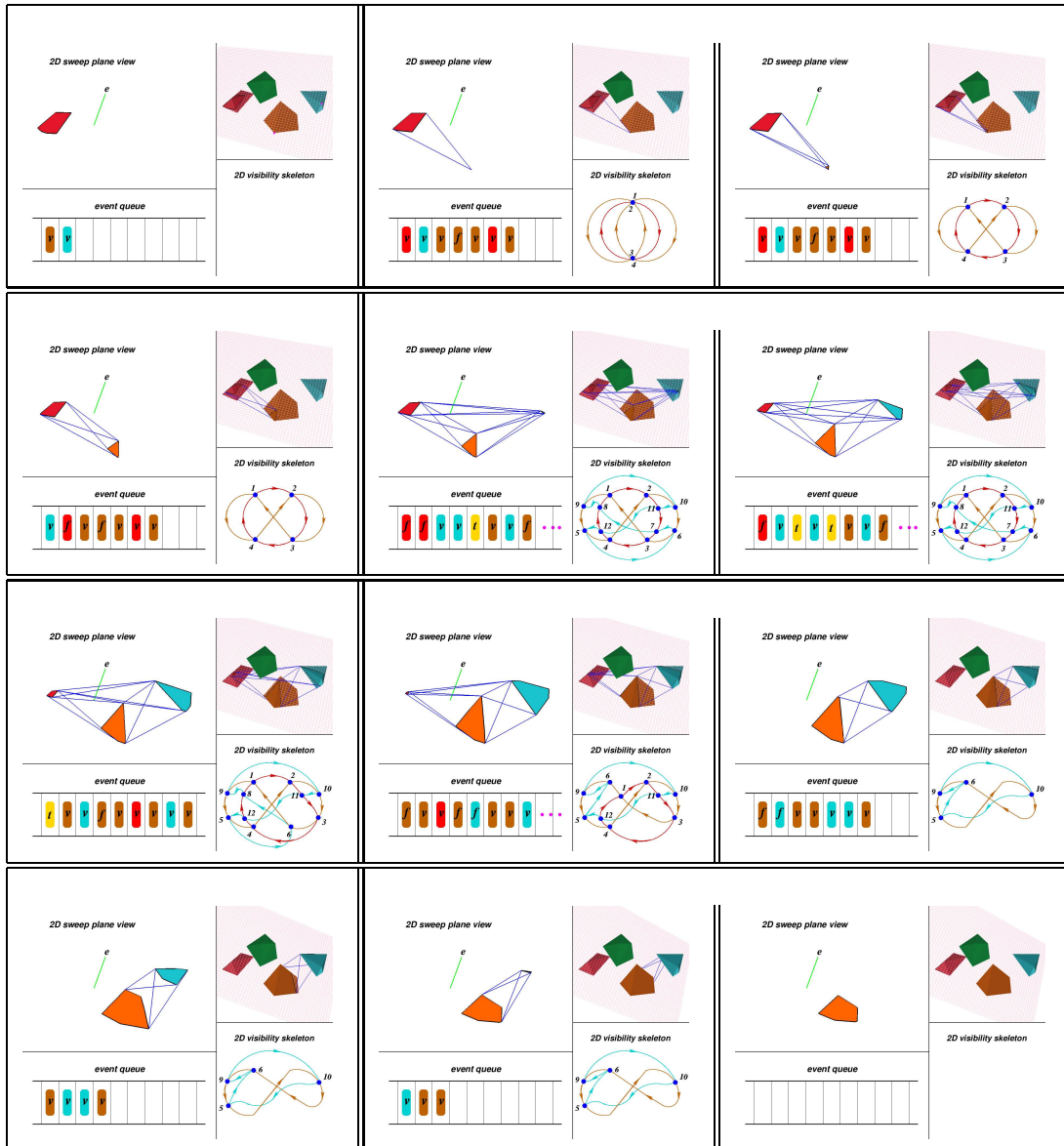


Figure 4.14. Snapshots of visualizing the computational steps of the implementation.

steps, we showed, in both 3D and 2D views (Figure 4.13 (a) and (b)), the action of the rotating sweep plane intersecting the input polytopes, as well as the concurrent states of the eventlist and the 2D visibility skeleton graph (Figure 4.13 (c), (d)). Note that in Figure 4.13 (c) and Figure 4.14, the V- and F-events are illustrated in the same color as the corresponding polytope on which the event occurs, and the T-events

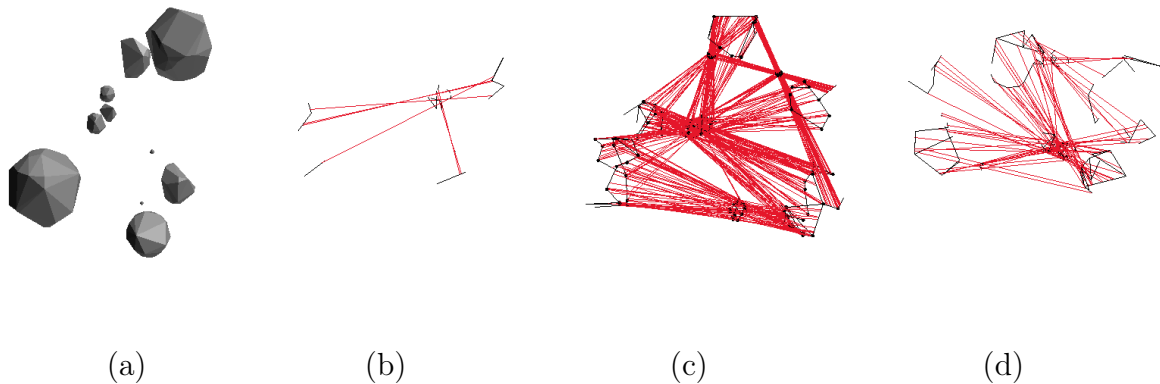


Figure 4.15. (a) An input of ten polytopes, and the output of (b) 6 *EEEE*, (c) 438 *VEE* and (d) 85 *FEE* type vertices.

are illustrated in yellow. In Figure 4.13 (d), each of the directed cycles indicates the clockwise ordering of the bitangents around a polytope which has the same color as the cycle. Finally, a few frames of the intermediate computational steps are shown in Figure 4.14.

We briefly discuss the technical details of making this video. The graphical output was produced using the *Geomview* software [45] through the interface provided by the *CGAL* library. We took snapshots of the *Geomview* window display, while rotating the viewpoint to provide a 3D view of the objects in the display window. Finally, we used *iMovie* [63] to assemble all the snapshots together into the final video. We used the *Audacity* [8] software for the audio.

We display the output for an input scene in Figure 4.15. Figure 4.15 (a) shows 10 input polytopes, which is a subset of *Scene II* (Figure 4.12 (b)). Figure 4.15 (b), (c), (d) illustrate the *EEEE*, *VEE*, *FEE* vertices, respectively. Note that each visibility skeleton vertex corresponds to a free line segment that is tangent to its supporting polytope edges (or vertices). In Figure 4.15 (b), (c), (d), the vertices are displayed in

red, and the supporting polytope edges and polytope vertices are displayed in black. There are 6 *EEEE* vertices, 438 *VEE* vertices, and 85 *FEE* vertices in the scene in Figure 4.15 (a).

4.5.2 Experimental Verification

We verified the correctness of our implementation by comparing its output, that is, the type *EEEE*, *VEE* and *FEE* vertices, with that of an implementation of the brute force algorithm. Since the brute force algorithm implementation was straightforward, having only about a thousand lines of code, the output is less error prone, and credible to use to verify the implemented software.

Given a set of input polytopes, we verified that the two sets of outputs from the implementations were equal. That is, we checked that each computed 3D visibility skeleton vertex from one output had an identical vertex from the other output, and vice-versa. Here we define a pair of identical vertices as two vertices having the same vertex type, with their corresponding transversals having the same geometric position, and having the same supporting polytope edges (or vertices).

The example scenes *I* and *III* in Figure 4.12 were used for the verification. Additionally, we tested another 18 scenes with number of polytopes varying from 5 to 130, and the number of polytope edges up to 1000. We note that the number of edges in our tested scene is relatively small, because the brute force algorithm (with complexity $O(n^5)$) tends to be very time consuming.

All the twenty tests used number type `filtered_exact`. On all twenty input scenes, experimental results showed that both implementations admit equal sets of

output. Naturally, such experiments do not guarantee that our implementation is error free, but they can at least testify the relative stability of our implementation, so that we can trust and rely on the experimental data from this implementation to analyze the software performance and carry on experimental studies.

4.6 Performance

In this section, we study the performance of our implementation in various aspects. We first study the running time in terms of parameter n and k , for given k input polytopes with n edges in total. We then study the running time in terms of the number of polygons, such that we can compare our implementation with the existing one of Durand *et al.* [34, 36]. Finally, we study the running time of our implementation in terms of the number types `double`, `filtered_exact`, and `CORE`.

All the experiments were done on a *i686* machine with *Pentium* 2.80 GHz CPU running Linux, with 2 GB of main memory. Running time was measured with the `getrusage()` command and the `ru_utime` attribute.

4.6.1 Running Time in Terms of n and k

We first briefly describe the experiments that were used in our experimental study. We then analyze the results obtained in terms of parameter n , which is the parameter used in many other studies on the same subject [34, 36, 47, 94], and in terms of parameter $n \sqrt{n} k \log k$, which suits the running time of our implementation better.

The Experiments. To study the performance of our implementation, we used the same experiments that were carried out for studying the size of 3D visibility skeleton (see Chapter 6), since both results were obtained during the same experimental testing. We briefly summarize our experiments here and delay the detailed description to Section 6.2.2, since the experimental setting corresponds more to the study of the size of 3D visibility skeleton.

We generated three suites of experiments according to three parameters: μ , the scene density; k , the number of polytopes; and n/k , the number of edges per polytope. We recall that n is the total number of edges of the input. In Suite I, we chose $n/k \approx 6, 42, \text{ and } 84$, $\mu = 0.3, 0.5 \text{ and } 0.01$, and varied k from 10 up to 230 in each input scene. In Suite III, we chose $\mu = 0.3$, varied n/k in the range of $[4, 24]$, $[4, 34]$, and $[4, 44]$ uniformly in each input scene for k varying from 10 to 150. We omit the description of Suite II, since this suite of experiments did not contribute useful information to our running time study.

Running Time in Terms of n . We present, in terms of the total number n of edges in the scene, the running time of our implementation (in seconds) of Suite I of our experiments, for $n/k \approx 6, 42, \text{ and } 84$, respectively. As shown in Figure 4.16, for a given value of n/k , the running time appears to be in $\Theta(n^{2.5} \log n)$ (see Figures 4.16 and 4.17). Furthermore, we notice that, for a fixed n , the running time drops drastically when the number k of polytopes gets smaller. These observations are consistent with the theoretical bounds, that the time complexity of the algorithm is in $O(n^2 k^2 \log k)$.

In what follows we analyze the running time in terms of both n and k .

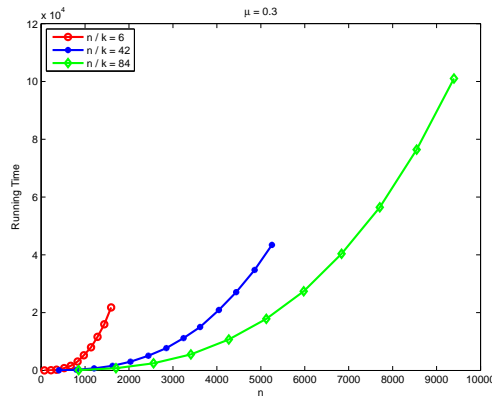


Figure 4.16. Running time (in seconds; 2.88×10^4 seconds = 8 hours) in terms of n , the number of edges in the scene in Suite I ($\mu = 0.3$).

Running Time in Terms of $n \sqrt{n} k \log k$. We first study the running time of our implementation in terms of $n \sqrt{n} k \log k$ for experiments in Suite I, and show the results in Figure 4.17(a), 4.17(b) and 4.17(c) for scene density $\mu = 0.3$, 0.05 and 0.01 respectively. We observe that for a fixed polytope complexity n/k , the running time seems linear in $n \sqrt{n} k \log k$. More precisely, we observe a running time of $C'_\mu n \sqrt{n} k \log k$ seconds with C'_μ no more than $3 \cdot 10^{-4}$ for the considered densities (roughly equal to $2.1 \cdot 10^{-4}$, $1.9 \cdot 10^{-4}$, and $1.5 \cdot 10^{-4}$ for μ equal to 0.3, 0.05, and 0.01, respectively). Note, however, that for density 0.3, the data we obtained from groups $n/k \approx 42$ and 84 fit the estimated time complexity well, whereas the data from the group $n/k \approx 6$ is a constant factor away.

We furthermore study the time complexity in terms of $n \sqrt{n} k \log k$ on Suite III of our experiments. As shown in Figure 4.17(d), Suite III admits the same observation as in Suite I, that the running time of our implementation is linear in terms of $n \sqrt{n} k \log k$.

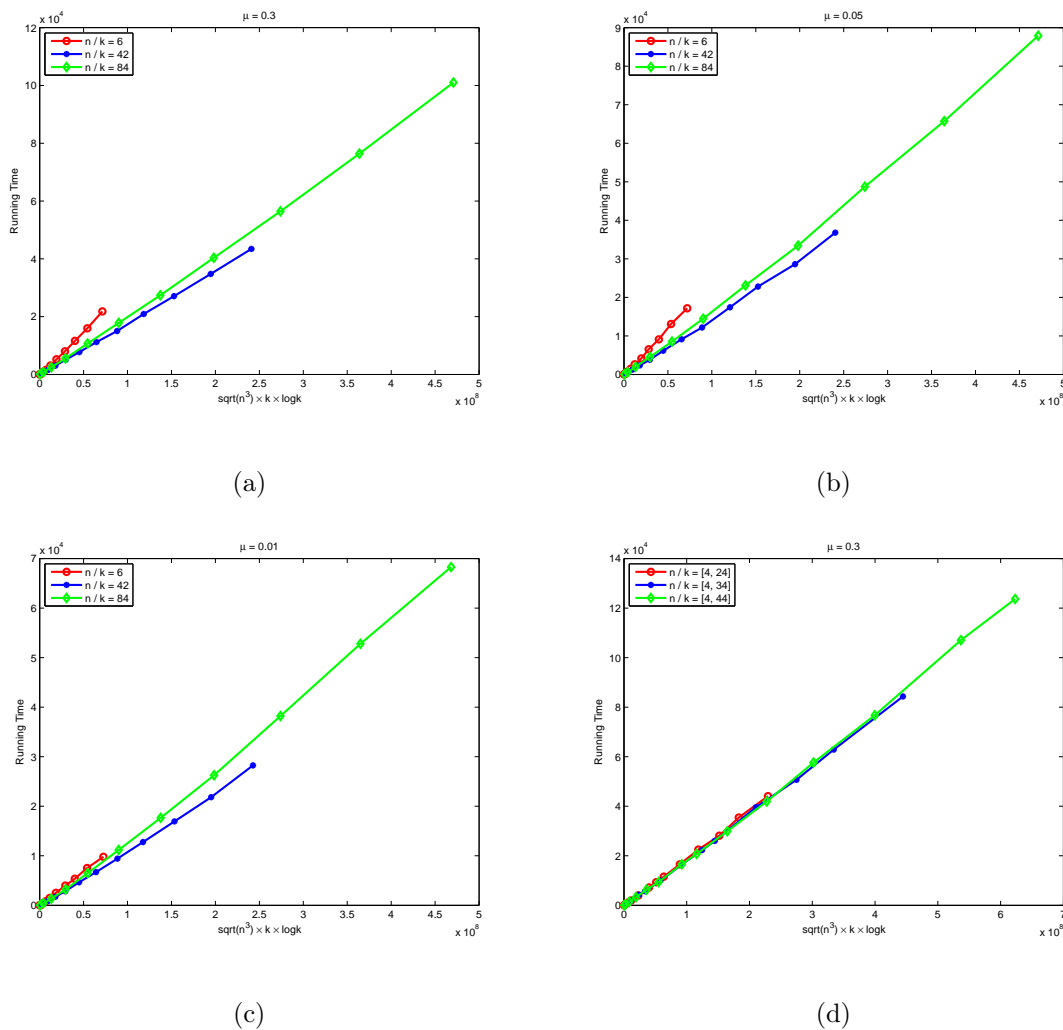


Figure 4.17. Running time (in seconds) in terms of $n^{1.5} k \log k$: for density (a) $\mu = 0.3$, (b) $\mu = 0.05$ and (c) $\mu = 0.01$, where the polytopes have constant complexity (n/k edges) (Suite I); and (d) for density $\mu = 0.3$, where the polytope complexity varies in the range of $[4 - 24]$, $[4 - 34]$, and $[4 - 44]$ (Suite III).

The observed running time can be intuitively explained as follows. Note first that $n \sqrt{n} k \log k$ is equal to $n \sqrt{n/k} k \sqrt{k} \log k$. We dissect this expression as follows. First, n is the number of sweeps performed by the algorithm. We observe that the factor $k \sqrt{k}$ is linearly related to the average over all sweeps of the maximal

number of bitangents encountered in the sweep plane during a sweep (we omit here the presentation of these experiments). This is reasonable since the number of bitangents in the sweep plane varies from $\Theta(k^2)$, the trivial worst-case bound, to $\Theta(k)$, the expected bound in the right setting [41]. Furthermore, the factor $\sqrt{n/k}$ naturally relates to the number of updates caused by each bitangent during the sweep; indeed, following a bitangent during a sweep, the bitangent will encounter vertices on each of the two polytopes supporting it; the number of these vertices on each polytope is related and, intuitively, is less than the worst-case size of the silhouettes of the polytopes, which is in $O(\sqrt{n/k})$ in our setting.⁷ Finally, $\log k$ is the complexity of each update of the event list.

We recall that the worst-case running time complexity of this implementation is $O(n^2k(k \log k + n))$, which has much higher order than our observed running time.

4.6.2 Running Time in Terms of the Number of Polygons

We compare the performance of our implementation of the sweep plane algorithm with the brute force implementation by Durand *et al.* [34, 36], since it is the most well-known one among the three existing brute force implementations. The other two are by Schröder [94] and Glaves [47].

Recall that Durand *et al.* [34, 36] model the input scene as 3D polygons. For example, a cube would be modeled as 6 independent polygons (its faces), whereas

⁷It is shown that for any polyhedron of size $\Theta(m)$, the size of its silhouette viewed from a random point is $O(\sqrt{m})$ under some reasonable hypotheses [49] (see also [65] for the special case of polyhedra that approximate spheres). Thus, in our setting, it is reasonable to assume that the size of the silhouette does not depend much on the choice of the viewpoint, that is, for any polytope with n/k edges we consider, it is reasonable to assume that its silhouette has size $O(\sqrt{n/k})$ from any viewpoint.

our implementation models the input scene as one 3D polytope. The running time of the implementation of Durand *et al.* depends on the number n of polygons of constant complexity. The theoretical worst-case running time of their implementation is $O(n^5)$, and the practical running time, due to the heuristics they used, is about $O(n^{2.5})$. We show the experimental results of Durand *et al.*, which are reported in [34, 36], in Table 4.1.

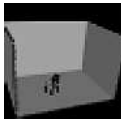
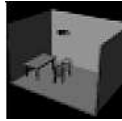
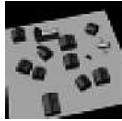




	a	b	c	d	e	f	g
Scene							
Polygons	84	168	312	432	756	1056	1488
Nodes ($\cdot 10^3$)	7	37	69	199	445	753	1266
Arcs ($\cdot 10^3$)	16	91	165	476	1074	1836	3087
Construction	1 s 71	12 s 74	37 s 07	1 min 39 s	5 min 36 s	14 min 36 s	31 min 59 s
Mem. (Mb)	1.8	9	21	55	135	242	416

Table 4.1. Experimental results reported in [34, 36], on a 195Mhz R10000 SGI Onyx2 (taken from [34]).

From Section 4.6.1, our experimental results suggest that the practical running time of our implementation is $O(n\sqrt{nk} \log k)$, which depends on the total number of polytope edges n and the number of polytopes k . The practical running time of the implementation of Durand *et al.* depends only on n . We therefore compared the two implementations by considering input scenes that have similar numbers of polygonal faces, that is, for each number of polygons, for our implementation, we distinguished between $n \sim k$ and $n \gg k$. Specifically, we considered 3 pairs of scenes. Each pair of scenes contains two types: (i): $n \sim k$; (ii): $n \gg k$, and they have roughly the same number of polygons. As shown in Table 4.2, we considered the approximate

numbers of polygons as 450, 750, and 1000, which are similar to column d , e and f in Table 4.1. We compare the running time, and the number of computed $EEEE$, VEE , and FEE vertices of these three pairs of input scenes with the experimental results in [34, 36] (Table 4.1). All the tests used number type *double*, which is the same as used in [34, 36]. The computational results are listed in Table 4.2.




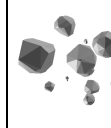

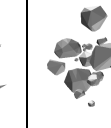
	pair 1		pair 2		pair 3	
Polygons	approx. 450		approx. 750		approx. 1000	
Polytopes	$n \sim k$	$n \gg k$	$n \sim k$	$n \gg k$	$n \sim k$	$n \gg k$
Scene						
k	90	9	150	15	200	20
n	660	636	1146	1143	1530	1479
t (<i>min.</i>)	5.30	0.12	29.5	0.91	66.3	2.18
$EEEE$	843	0	5872	12	11296	64
VEE	22401	199	89804	1571	157771	3597
FEE	1296	41	4738	339	8034	767

Table 4.2. Experimental results for three pairs of input scenes with the number of polytope faces approximately 450, 750, and 1000.

From Table 4.2, we can observe that, within each of the 3 pairs of input scenes, the number of polytope faces (i.e. polygons) are similar, but the running times are very different. Roughly speaking, the running times of scenes that feature $n \sim k$ are about 30 to 40 times slower than the running times of scenes that feature $n \gg k$.

Comparing the results in Table 4.2 with those in Table 4.1, for the scenes that have similar number of polygons, we also observe that the running time reported in Table 4.1 is faster than the scene in Table 4.2 that features $n \sim k$, but slower than the scene in Table 4.2 that features $n \gg k$. This should however be moderated by

the fact that the machine used for the experiments of Table 4.1 is substantially slower than the one used in the experiments of Table 4.2. These facts suggest that, for a given number of polygons, when compared to the brute force implementation, our implementation appears inefficient for scenes with $n \sim k$, but competitive for scenes with $n \gg k$. Therefore, to compare the running time of the two implementations, it is more accurate to use both parameters n and k , rather than the number of polygons alone.

Indeed, the reason for these results becomes very clear when we look at the practical running time of the two implementations, that is, $O(n\sqrt{nk} \log k)$ versus $O(n^{2.5})$. For a given scene, when $n \sim k$, the running time of our implementation is about $O(n^{2.5} \log n)$, which is larger than the practical complexity $O(n^{2.5})$ of the brute force implementation; whereas when $n \gg k$, the running time of our implementation is about $O(n^{1.5})$, which is asymptotically less than the brute force implementation.

In summary, we have compared our implementation with a brute force implementation that used heuristics for speeding up the computation. Due to the algorithm we used in our implementation, the same heuristics that are used in the brute force one are not applicable. As a result, our implementation is only competitive with the brute force one for certain input scenes, described as $n \gg k$. One possibility for future work arising from this thesis is to design some heuristics that are applicable to the sweep plane algorithm, and thus to try to improve the practical efficiency of our implementation.

	double (sec.)	filtered_exact (sec.)	$\frac{\text{filtered_exact}}{\text{double}}$	CORE (sec.)	$\frac{\text{CORE}}{\text{double}}$
scene <i>I</i>	1	3.6	3.7	44	45
scene <i>II</i>	117	406	3.5	5,487	47
scene <i>III</i>	229	742	3.2	15,100	66
scene <i>IV</i>	1290	3807	3.0	94,110	73

Table 4.3. Running time (in seconds) in terms of number type: `double`, `filtered_exact`, and `CORE`, as well as the ratio of `filtered_exact` and `CORE` to `double`, for four input scenes.

4.6.3 Running Time in Terms of Number Types

We compared the running time of our implementation using number type `double`, `filtered_exact`, and `CORE`. The tests were made on 4 input scenes. Scene *I*, *II* and *III* are shown in Figure 4.12. We recall that these scenes have 10, 30, 90 polytopes, and 129, 2964, 660 edges respectively. Scene *IV* has 150 polytopes and 1146 edges. Our implementation gives the same outputs with all three number types on each of the four input scenes. The obtained data is listed in Table 4.3. Note that here, we tested the running time of the implementation without computing and outputting the 3D visibility skeleton vertices. Therefore, the computation mainly focused on updating the 2D visibility skeleton and the event list, which mostly involves computation of predicates, and thus reflects the performance of the three number types.

From Table 4.3, we observe that using `filtered_exact` number type is about 3 to 4 times slower than using `double`, whereas the `CORE` number type is much slower than `double`, and varies from 44 to 73 times from scene *I* to scene *IV*.

Based on all our experiments that used number type `CORE`, we have observed that, when the input scenes have a large number of polytopes, the `CORE` number type tends to be much slower. This may explain the fact that, in Table 4.3, `CORE` has larger

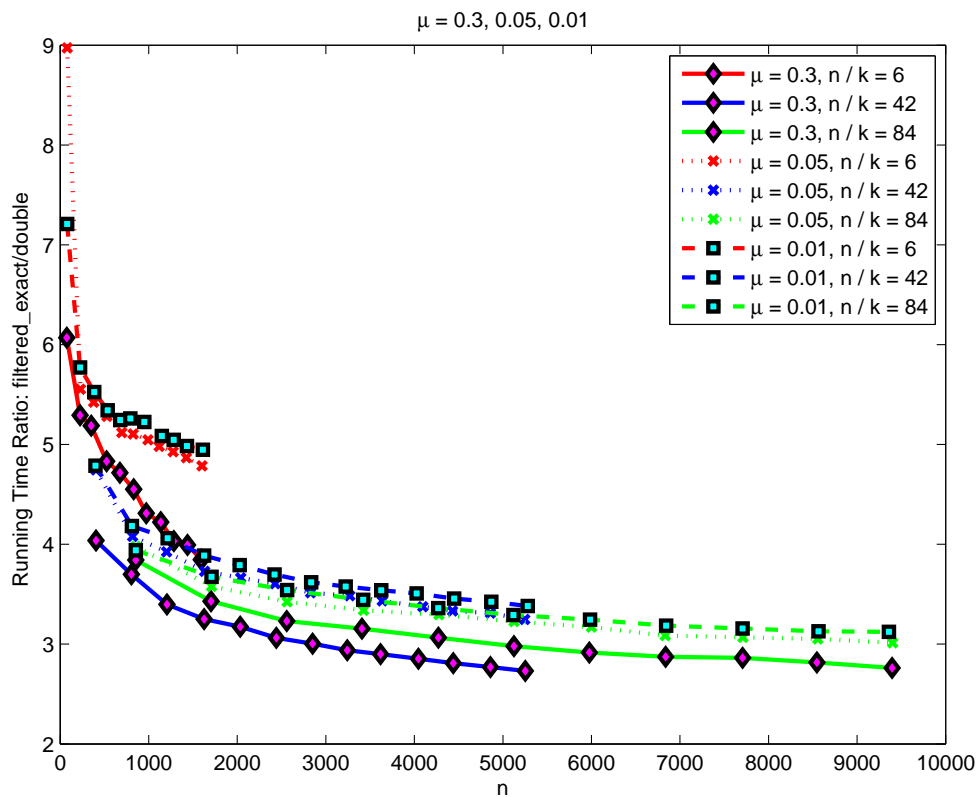


Figure 4.18. Running time ratio of number type double to `filtered_exact`, tested on the experiments in Suite I.

running time variation when compared to `double`.

We further studied the running time ratio of number type `filtered_exact` to `double` using the experiments of Suite I (briefly described in Section 4.6.1, and in more detail in Section 6.2.2), and show the results in Figure 4.18.

The observation we can make from Figure 4.18 is very similar to that from Table 4.3, that is, the running time ratio may be high (as high as nine times at the highest) when n is very small, but converges to about three when n gets larger. The cause may be that, when n is small, the running time is mainly spent on applying the number

type `filtered_exact` itself.

4.7 Conclusion and Bibliographic Notes

We have presented in this chapter an implementation of the sweep-plane algorithm to compute the vertices of the 3D visibility skeleton of disjoint polytopes. The performance analysis indicates that the skeleton vertices can be computed exactly in a reasonable amount of time. Further work includes improving the performance of this implementation and completing the implementation for degenerate situations. A major challenge is to extend the sweep algorithm to handle general polyhedra.

A terse discussion of this chapter has appeared in the Proceedings of 16th Annual European Symposium on Algorithms [106], and some of the results in Section 4.5.1 have appeared in the session of video and multimedia presentations in the 23rd Annual ACM Symposium on Computational Geometry [107].

Chapter 5

The Algebraic Degree of the Predicates

In this chapter, we study various predicates and their degrees appearing in the implementation of the sweep algorithm. Note that computing these predicates can often be an important operation in solving 3D visibility problems arising in computer graphics [14, 33, 36, 38, 39, 82].

A predicate is a function that returns a value from a discrete set. Typically, geometric predicates answer questions of the type “Is a point inside, outside or on the boundary of a set?” and return a value from a discrete set such as “yes”, “no”, “undetermined”. We consider predicates that are evaluated by boolean functions of more elementary predicates, the latter being functions that return the sign ($-$, 0 or $+$) of a multivariate polynomial whose arguments are a subset of the input parameters of the problem instance (see, for instance, [12]). By *degree* of a procedure for evaluating a predicate, we mean the maximum degree in the input parameters

among all polynomials used in the evaluation of the predicate by the procedure. In what follows we informally refer to this measure as the degree of the predicate. We are interested in the degree because it provides a measure of the number of bits required for an exact evaluation of our predicates when the input parameters are integers or floating-point numbers; the number of bits required is then roughly the product of the degree with the number of bits used in representing each input value.

In this chapter, we first study the degree of standard procedures for determining the number of line transversals to four lines or four segments in 3D. Recall that four lines in \mathbb{R}^3 admit 0, 1, 2 or an infinite number of line transversals [60] (p.164) and that four segments admit up to 4 or an infinite number of line transversals [15]. These predicates are not only used to compute the T-events, but also are ubiquitous in 3D visibility problems. Finally, we study the predicate for ordering planes through two fixed points, each plane containing a third rational point or a line transversal to four segments or lines. This predicate is used to sort the event list.

Our study shows that standard procedures for solving these predicates have high degrees. We study, in particular, procedures that involve computing the Plücker coordinates [96] of the line transversals involved in the predicates. Throughout the chapter, the points defining input geometric primitives (which can be lines, segments, and triangles) are, by assumption, given by their Cartesian coordinates, and the degrees of the procedures for evaluating predicates are expressed in these coordinates. We show that, for determining the number of transversals to four lines or four segments, such standard methods lead to procedures of degree 22 or 36, respectively. Also, in a rotational sweep about a line, for ordering two planes, each defined by a line transver-

sal to four lines, such methods lead to a procedure of degree 144. Furthermore, in some implementations, the Plücker coordinates of the relevant line transversals are computed in a way that the degrees of these procedures are even higher. For instance, the procedure for evaluating the latter predicate for ordering planes then becomes of degree 168 instead of degree 144. These very high degrees may help explain why using fixed-precision floating-point arithmetic in implementations for solving 3D visibility problems are prone to errors when given real-world data (see, for instance, [47]).

The degrees we present are tight, that is, they correspond to the maximum degree of the polynomials to be evaluated, in the worst case, in the procedures we consider. It should be stressed that these degrees refer to polynomials used in specific evaluation procedures, and we make no claim as to the optimality of these procedures.

In the next section we describe a standard method used for computing the line transversals to four lines, which is common to all our predicates. In Section 5.2 we describe the predicates and their degrees. Some experimental results are presented in Section 5.3.

5.1 Computing Lines through Four Lines

We describe here a method for computing the line transversals to four lines in real projective space $\mathbb{P}^3(\mathbb{R})$. This method is a variant, suggested by Devillers and Hall-Holt, and also described in Redburn [87], of that by Hohmeyer and Teller [61]. Note that, for evaluating predicates, the latter method is not appropriate because it uses singular value decomposition for which we only know of numerical methods and thus the line transversals cannot be computed exactly, when needed.

Each line can be described using Plücker coordinates (see [96], for example, for a review of Plücker coordinates). If a line ℓ in \mathbb{R}^3 is represented by a direction vector \vec{u} and a point p in \mathbb{R}^3 then ℓ can be represented by the six-tuple $(\vec{u}, \vec{u} \times \overrightarrow{Op})$ in real projective space $\mathbb{P}^5(\mathbb{R})$, where O is any arbitrarily, fixed, origin and \times denote the cross product. The side product \odot of any two six-tuples $\ell = (a_1, a_2, a_3, a_4, a_5, a_6)$ and $k = (x_1, x_2, x_3, x_4, x_5, x_6)$ is $\ell \odot k = a_4x_1 + a_5x_2 + a_6x_3 + a_1x_4 + a_2x_5 + a_3x_6$. The fundamental importance of the side product lies in the fact that a six-tuple $k \in \mathbb{P}^5(\mathbb{R})$ represents a line in 3D if and only if $k \odot k = 0$; this defines a quadric in $\mathbb{P}^5(\mathbb{R})$ called the Plücker quadric. More generally, recall that two lines intersect in real projective space $\mathbb{P}^3(\mathbb{R})$ if and only if the side product of their Plücker coordinates is zero. Notice that this implies that there is a predicate for determining whether two lines intersect in $\mathbb{P}^3(\mathbb{R})$ which is of degree two in the Plücker coordinates of the lines and, if the lines are each defined by two points, of degree three in the Cartesian coordinates of these points.

Oriented lines in \mathbb{R}^3 , with direction vector \vec{u} and through a point p , can be represented similarly by a six-tuple $(\vec{u}, \vec{u} \times \overrightarrow{Op})$ in real oriented projective space (*i.e.*, the quotient of $\mathbb{R}^6 \setminus \{0\}$ by the equivalence relation induced by positive scaling). The sign (positive or negative) of the side operator of two oriented lines ℓ and k then determines on which “side” of line ℓ , k lies; for instance, if op and oq are two lines oriented from o to p and from o to q and ℓ is an arbitrarily oriented line such that ℓ , p , q , and o are not coplanar, then $(\ell \odot op)(\ell \odot oq) \leq 0$ if and only if ℓ intersects segment pq (see Figure 5.1(a)).

Given four lines ℓ_1, \dots, ℓ_4 , our problem here is to compute all lines

$k = (x_1, x_2, x_3, x_4, x_5, x_6) \in \mathbb{P}^5(\mathbb{R})$ such that $k \odot \ell_i = 0$, for $1 \leq i \leq 4$, which can be written in the following form:

$$\begin{pmatrix} a_4 & a_5 & a_6 & a_1 & a_2 & a_3 \\ b_4 & b_5 & b_6 & b_1 & b_2 & b_3 \\ c_4 & c_5 & c_6 & c_1 & c_2 & c_3 \\ d_4 & d_5 & d_6 & d_1 & d_2 & d_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (5.1)$$

where the rows of the 4×6 matrix contain the Plücker coordinates of the four lines.

This can be rewritten as

$$\begin{pmatrix} a_6 & a_1 & a_2 & a_3 \\ b_6 & b_1 & b_2 & b_3 \\ c_6 & c_1 & c_2 & c_3 \\ d_6 & d_1 & d_2 & d_3 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} + \begin{pmatrix} a_4x_1 + a_5x_2 \\ b_4x_1 + b_5x_2 \\ c_4x_1 + c_5x_2 \\ d_4x_1 + d_5x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (5.2)$$

Let δ denote the determinant of the above 4×4 matrix. Assuming $\delta \neq 0$, we can solve the system for x_3, x_4, x_5 , and x_6 in terms of x_1 and x_2 . Applying Cramer's rule, we get

$$\begin{cases} x_3 = -(\alpha_1x_1 + \beta_1x_2)/\delta \\ x_4 = -(\alpha_2x_1 + \beta_2x_2)/\delta \\ x_5 = -(\alpha_3x_1 + \beta_3x_2)/\delta \\ x_6 = -(\alpha_4x_1 + \beta_4x_2)/\delta \end{cases}$$

where α_i (respectively β_i) is the determinant δ with the i^{th} column replaced by

$(a_4, b_4, c_4, d_4)^T$ (respectively $(a_5, b_5, c_5, d_5)^T$). We rewrite this system as

$$\begin{cases} x_1 = -u\delta \\ x_2 = -v\delta \\ x_3 = \alpha_1 u + \beta_1 v \\ x_4 = \alpha_2 u + \beta_2 v \\ x_5 = \alpha_3 u + \beta_3 v \\ x_6 = \alpha_4 u + \beta_4 v \end{cases} \quad (5.3)$$

with $(u, v) \in \mathbb{P}^1(\mathbb{R})$. Since k is a line, we have $k \odot k = 0$, which implies

$$x_1 x_4 + x_2 x_5 + x_3 x_6 = 0.$$

Substituting in the expressions for $x_1 \dots x_6$, we get

$$Au^2 + Buv + Cv^2 = 0 \quad (5.4)$$

where

$$\begin{aligned} A &= \alpha_1 \alpha_4 - \alpha_2 \delta, \\ B &= \alpha_1 \beta_4 + \beta_1 \alpha_4 - \beta_2 \delta - \alpha_3 \delta, \\ C &= \beta_1 \beta_4 - \beta_3 \delta. \end{aligned}$$

Solving this degree-two equation in (u, v) and replacing in (5.3), we get (assuming that $A \neq 0$) that the Plücker coordinates of the transversal lines k are:

$$\begin{cases} x_1 = B\delta \mp \delta \sqrt{B^2 - 4AC} \\ x_2 = -2A\delta \\ x_3 = -B\alpha_1 + 2A\beta_1 \pm \alpha_1 \sqrt{B^2 - 4AC} \\ x_4 = -B\alpha_2 + 2A\beta_2 \pm \alpha_2 \sqrt{B^2 - 4AC} \\ x_5 = -B\alpha_3 + 2A\beta_3 \pm \alpha_3 \sqrt{B^2 - 4AC} \\ x_6 = -B\alpha_4 + 2A\beta_4 \pm \alpha_4 \sqrt{B^2 - 4AC}. \end{cases} \quad (5.5)$$

Lemma 1. *Consider four lines, given by the Cartesian coordinates of pairs of points, that admit finitely many line transversals in $\mathbb{P}^3(\mathbb{R})$. If the four lines are not parallel to a common plane, the Plücker coordinates of their transversals in $\mathbb{P}^3(\mathbb{R})$ can be written as $\phi_i + \varphi_i\sqrt{\Delta}$, $i = 1, \dots, 6$, where ϕ_i , φ_i , and Δ are polynomials of degree at most 17, 6, and 22, respectively, in the coordinates of the input points. Otherwise, the Plücker coordinates of the transversals can be written as polynomials of degree at most 19. Moreover, these bounds are, in the worst case, reached for three of the coordinates.*

Proof. The assumption that the four lines admit finitely many transversals in $\mathbb{P}^3(\mathbb{R})$ ensures that the 4×6 matrix of Plücker coordinates (in (5.1)) has rank 4. Consider first the case where the four input lines are not all parallel to a common plane. Then, the 4×3 matrix of the direction vectors of the four lines has rank 3. By the basis extension theorem, this matrix can be complemented by one of the other columns of the matrix of Plücker coordinates (of (5.1)) in order to get a 4×4 matrix of rank 4. We can thus assume, without loss of generality, that the 4×4 matrix of (5.2) has rank 4.

Since, by assumption, the four lines admit finitely many transversals in $\mathbb{P}^3(\mathbb{R})$, A, B , and C in (5.4) are not all zero. We compute the degree, in the coordinates of the input points, of the various polynomial terms in (5.5). For each input line ℓ_i , the first three and last three coordinates of its Plücker representation have degree 1 and 2, respectively. Hence δ , α_1 , and β_1 have degree 5 and α_i and β_i have degree 6 for $i = 2, 3, 4$. Hence, A, B , and C have degree 11 and the bounds on the degrees of ϕ_i, φ_i , and Δ follow. Note, in particular, that, if $A \neq 0$, these bounds are reached for $i = 4, 5, 6$.

Consider now the case where the four input lines are parallel to a common plane. Since the four lines admit finitely many transversals in $\mathbb{P}^3(\mathbb{R})$, they are not parallel. It follows that the 4×3 matrix of the direction vectors of the four lines has rank 2. Two vectors, say (a_i, b_i, c_i, d_i) for $i = 1, 2$, are thus linearly independent and, by the basis extension theorem, the corresponding 4×2 matrix can be complemented by two other columns (say, (a_i, b_i, c_i, d_i) for $i = 4, 5$) of the matrix of Plücker coordinates (of (5.1)) in order to define a 4×4 matrix of rank 4. As above, a straightforward computation gives the Plücker coordinates of the line transversal. We get

$$x_1 = \alpha_1 u, \quad x_2 = \alpha_2 u, \quad x_3 = -u \delta, \quad x_4 = \alpha_3 u + \beta_3 v, \quad x_5 = \alpha_4 u + \beta_4 v, \quad x_6 = -v \delta$$

where $(u, v) \in \mathbb{P}^1(\mathbb{R})$ is a solution of the equation

$$A' u^2 + B' u v = 0 \quad \text{where} \quad A' = \alpha_1 \alpha_3 + \alpha_2 \alpha_4 \quad \text{and} \quad B' = \alpha_1 \beta_3 + \alpha_2 \beta_4 + \delta^2; \quad (5.6)$$

$\delta, \alpha_1, \alpha_2, \beta_3, \beta_4$ have degree 6 and α_3, α_4 have degree 7 (and $\beta_1 = \beta_2 = 0$), and thus A' and B' have degree 13 and 12, respectively. Note that A' and B' are not both zero since there are finitely many transversals. The Plücker coordinates of the transversals can thus be written as polynomials of degree at most 19 and, for one of the transversals (the one not in the plane at infinity), this bound is reached for three coordinates (namely, x_4, x_5, x_6). \square

Lemma 2. *Consider four lines, given by the Cartesian coordinates of pairs of points, that admit finitely many line transversals in $\mathbb{P}^3(\mathbb{R})$. If the four lines are not parallel to a common plane, we can compute on each transversal two points whose homogeneous coordinates have the form $\phi_i + \varphi_i \sqrt{\Delta}$, $i = 1, \dots, 4$, where ϕ_i, φ_i , and Δ are polynomials of degree at most 17, 6, and 22, respectively, in the coordinates of the*

input points. Otherwise, we can compute on each transversal two points whose homogeneous coordinates are polynomials of degree at most 19. Moreover, these bounds are reached, in the worst case, for some coordinates.

Proof. Denote by w_1 (respectively w_2) the vector of the first (respectively last) three coordinates of (x_1, \dots, x_6) , the Plücker coordinates of a line k , and let n denote any vector of \mathbb{R}^3 . Then, if the four-tuple $(w_2 \times n, w_1 \cdot n)$ is not equal to $(0, 0, 0, 0)$, it is a point (in homogeneous coordinates) on the line k (by Lagrange's triple product expansion formula). By considering the axis unit vectors for n , we get that the four-tuples $(0, x_6, -x_5, x_1)$, $(-x_6, 0, x_4, x_2)$, $(x_5, -x_4, 0, x_3)$ that are non-zero are points on the transversal lines k . Either five of the six Plücker coordinates of k are zero or at least two of these four-tuples are non-zero and thus are points on k . In the latter case, the result follows from Lemma 1. In the former case, two points with coordinates 0 or 1 can easily be computed on line k since the line is then one of the axes or a line at infinity defined by the directions orthogonal to one of the axes. \square

Remark 3. In some implementations (for instance, the one of [87]), the 4×4 submatrix of the matrix of Plücker coordinates (see (5.1)) used for computing the line transversals is chosen, by default, as the leftmost submatrix whose determinant has degree 7 in the coordinates of the input points. In this case, the Plücker coordinates of the line transversals are written as $\phi_i + \varphi_i \sqrt{\Delta}$, $i = 1, \dots, 6$, where ϕ_i , φ_i , and Δ are polynomials of degree at most 20, 7, and 26, respectively, in the coordinates of the input points (and these bounds are reached). A similar statement holds for the homogeneous coordinates of two points on the transversals.

5.2 Predicates

5.2.1 Preliminaries

We start with two straightforward lemmas on the degree of predicates for determining the sign of simple algebraic numbers. If x is a polynomial expression in some variables, we denote by $\deg(x)$ the degree of x in these variables. This first lemma is trivial and its proof is omitted.

Lemma 4. *If a, b , and c are polynomial expressions of (input) rational numbers, the sign of $a + b\sqrt{c}$ can be determined by a predicate of degree $\max\{2 \deg(a), 2 \deg(b) + \deg(c)\}$.*

Lemma 5. *If $\alpha_i, \beta_i, \delta, \mu$, $i = 1, 2$, are polynomial expressions of (input) rational numbers, the sign of $\alpha_1 + \beta_1 \sqrt{\delta} + (\alpha_2 + \beta_2 \sqrt{\delta}) \sqrt{\mu}$ can be obtained by a predicate of degree*

$$\begin{aligned} & \max\{4 \deg(\alpha_1), \quad 4 \deg(\beta_1) + 2 \deg(\delta), \quad 4 \deg(\alpha_2) + 2 \deg(\mu), \\ & 4 \deg(\beta_2) + 2 \deg(\delta) + 2 \deg(\mu), \quad 2 \deg(\alpha_1) + 2 \deg(\beta_1) + \deg(\delta), \\ & 2 \deg(\alpha_2) + 2 \deg(\beta_2) + 2 \deg(\mu) + \deg(\delta)\}. \end{aligned}$$

Proof. The predicate is to evaluate the sign of an expression of the form $a + b\sqrt{\mu}$, where $a = \alpha_1 + \beta_1 \sqrt{\delta}$, $b = \alpha_2 + \beta_2 \sqrt{\delta}$, and $\alpha_i, \beta_i, \mu, \delta$ are rational. This can be done by evaluating the signs of a , b , and $a^2 - b^2\mu$. The first two signs can be obtained by directly applying Lemma 4. On the other hand, $a^2 - b^2\mu$ is equal to $A + B\sqrt{\delta}$ with $A = \alpha_1^2 + \beta_1^2\delta - \alpha_2^2\mu - \beta_2^2\mu\delta$ and $B = 2\alpha_1\beta_1 - 2\alpha_2\beta_2\mu$. The sign of $A + B\sqrt{\delta}$ can be determined by another application of Lemma 4, which gives the result. \square

5.2.2 Transversals to Four Lines

We consider first the predicate for determining whether four lines admit 0, 1, 2, or infinitely many line transversals in $\mathbb{P}^3(\mathbb{R})$ (that is lines in $\mathbb{P}^3(\mathbb{R})$ that intersect, in $\mathbb{P}^3(\mathbb{R})$, the four input lines). An evaluation of this predicate directly follows from the algorithm described in Section 5.1 for computing the line transversals. In what follows, all input points are, by assumption, given by their Cartesian coordinates.

Theorem 6. *Given four lines defined by pairs of points, there is a predicate of degree 22 in the coordinates of these points to determine whether the four lines admit 0, 1, 2, or infinitely many line transversals in $\mathbb{P}^3(\mathbb{R})$.*

Proof. We consider three cases. First, if the four lines are parallel, which can easily be determined by a predicate of degree 3, then they admit infinitely many line transversals in $\mathbb{P}^3(\mathbb{R})$. Second, if the four lines are not parallel but parallel to a common plane, which can easily be determined by a predicate of degree 3, then the four lines admit infinitely many transversals if Equation (5.6) is identically zero and, otherwise, 2 line transversals in $\mathbb{P}^3(\mathbb{R})$. This can be determined with a predicate of degree 13 (see the proof of Lemma 1). Finally, if the four lines are not parallel to a common plane, they admit infinitely many transversals if Equation 5.4 is identically zero and otherwise, 0, 1, or 2 transversals depending on the sign of Δ (in Lemma 1), which is of degree 22 in the coordinates of the points defining the lines. \square

Note that if the leftmost (instead of the rightmost) 4×4 submatrix of the matrix of Plücker coordinates (in (5.1)) is used for computing line transversals (see Remark 3) then the procedure described in the above proof has degree 26 instead of 22.

All line transversals are defined in \mathbb{R}^3 except when the four input lines are parallel to a common plane, in which case the intersection of this plane with the plane at infinity is a line transversal at infinity. Note also that determining whether a line transversal in $\mathbb{P}^3(\mathbb{R})$ is transversal in \mathbb{R}^3 amounts to determining whether the transversal is parallel to one of the four input lines ℓ_i , that is, if their direction vectors are collinear. This can be done, by Lemmas 1 and 4, by a predicate of degree 36 in the Cartesian coordinates of the points defining the input lines.

Note, however, that if the points defining the ℓ_i have rational coordinates and if the transversal is parallel to one of the ℓ_i , the Plücker coordinates of the transversal are rational; indeed, the multiplicative factor of the direction vectors is rational (since one of the coordinates of the direction vector of the transversal is rational, *e.g.*, x_2 in (5.5)) and thus all the coordinates of this direction vector are rational, which implies that Δ is a square in (5.5). Hence, deciding whether a transversal is parallel to one of the input lines ℓ_i can be done by first determining whether Δ is a square and, if so, testing whether the direction vectors are collinear. It thus follows from Lemma 1 that determining whether a transversal is parallel to one of the input lines ℓ_i can be done with fixed-precision floating-point arithmetic using a number of bits roughly equal to 22 times the number of bits used in representing each input value. This should be compared to the degree 36 of the above procedure.

In this chapter we have restricted our attention to evaluation procedures for predicates that consist entirely of determining the signs of polynomial expressions in the input parameters. We see here an example of a predicate which may be more efficiently evaluated by a procedure which permits other operations, in this case, determining

whether a rational number is a square. This provides an interesting example of a geometric predicate whose algebraic degree does not seem to be an entirely adequate measure of the number of bits needed for the computation.

5.2.3 Transversals to Four Segments

We consider here the predicate of determining how many transversals four segments of \mathbb{R}^3 admit. Recall that four segments may admit up to 4 or infinitely many line transversals [15]. In this section, we prove the following theorem.

Theorem 7. *Given four line segments, there is a predicate of degree 36 in the coordinates of their endpoints to determine whether those segments admit 0, 1, 2, 3, 4, or infinitely many line transversals.*

Note that if the leftmost (instead of the rightmost) 4×4 submatrix of the matrix of Plücker coordinates (in (5.1)) is used for computing line transversals (see Remark 3), then the procedure described below for the predicate of Theorem 7 has degree 42 instead of 36.

We consider, in the following, the supporting lines of the four segments, that is, the lines containing the segments. In the case where one (or several) segment is reduced to a point, we regard as a supporting line any line through this point and parallel to at least another supporting line. We first consider the case where the four supporting lines admit finitely many transversals in $\mathbb{P}^3(\mathbb{R})$; this can be determined by a predicate of degree 22, by Theorem 6.

Lemma 8. *Given four segments in \mathbb{R}^3 whose supporting lines admit finitely many line*

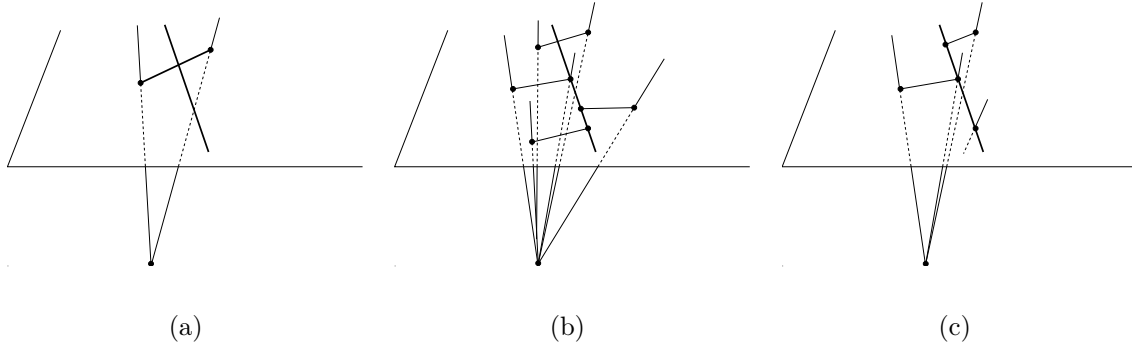


Figure 5.1. (a): Transversal ℓ intersects segment pq only if $(\ell \odot op) (\ell \odot oq) \leq 0$. (b-c): An illustration for the proof of Lemma 10.

transversals in $\mathbb{P}^3(\mathbb{R})$, determining the number of transversals to the four segments can be done with a predicate of degree 36 in the coordinates of their endpoints.

Proof. Let ℓ denote an (arbitrarily) oriented line, as well as its Plücker coordinates, that is a transversal to the four lines; ℓ can be computed as described in Section 5.1. We consider the predicate for determining whether ℓ intersects each of the four segments, in turn. Let p and q denote the endpoints of one of these segments. For any two distinct points r and s , denote by rs the Plücker coordinates of the line rs oriented from r to s ; depending on the context, rs also denotes the line through r and s or the segment from r to s .

If a point o does not lie in the plane containing line ℓ and segment pq (see Figure 5.1(a)), then line ℓ intersects segment pq if and only if the oriented line ℓ is on the opposite side of the two oriented lines from o to p and from o to q , that is, if $(\ell \odot op) (\ell \odot oq) \leq 0$ (recall that \odot denotes the side operator – see Section 5.1).

On the other hand, point o lies in a plane containing line ℓ and segment pq if and only if ℓ intersects (in $\mathbb{P}^3(\mathbb{R})$) both lines op and oq , that is both side operators $\ell \odot op$

and $\ell \odot oq$ are zero. By choosing point o to be, for instance, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$, or $(1, 1, 1)$, we ensure that one of these points will not be coplanar with ℓ and segment pq unless segment pq lies on ℓ .

Hence the predicate follows from the sign of the side operators of the line transversal and of a line defined by two points, one of which has coordinates equal to 0 or 1. The degree of the Plücker coordinates of the line through these two points is thus 1 (in the coordinates of the input points). Hence, by Lemma 1, the predicate can be computed by determining the sign of polynomials of degree at most 20 if the input lines are parallel to a common plane and, otherwise, by determining the sign of expressions of the form $a + b\sqrt{c}$ where a , b and c have degree at most 18, 7, and 22, respectively; moreover, these bounds are reached. By Lemma 4, the predicate thus has degree 36, which concludes the proof. \square

We now consider the case where the four lines admit infinitely many transversals. Recall that, in $\mathbb{P}^3(\mathbb{R})$, four lines or line segments admit infinitely many transversals only if [15]:

1. they lie in one ruling of a hyperbolic paraboloid or a hyperboloid of one sheet,
2. they are all concurrent, or
3. they all lie in a plane, with the possible exception of a group of one or more that all meet that plane at the same point.

We treat the cases independently.

Lemma 9. *Given four segments in \mathbb{R}^3 whose supporting lines are pairwise skew and admit infinitely many line transversals, determining the number of their line transversals can be done with a predicate of degree at most 36 in the coordinates of their*

endpoints.

Proof. When four lines are pairwise skew, their common transversals can be parameterized by their points of intersection with one of the lines; moreover, the set of common transversals to the four segments corresponds (through this parameterization) to up to four intervals on that line, and the transversals that correspond to the endpoints of these intervals contain (at least) one endpoint of the segments [15]. We can compute and order all these interval endpoints and determine whether there exists a transversal (to the four segments) through each midpoint of two consecutive distinct interval endpoints. By construction and by [15], the four segments admit such a transversal if and only if they admit infinitely many transversals.

The set of interval endpoints, on, say, segment s_1 is a subset of the endpoints of s_1 and of the intersection points of s_1 with the planes containing s_2 and an endpoint of s_3 or s_4 and of the intersection points of s_1 with the planes containing s_3 and an endpoint of s_2 . The coordinates of these points can be trivially computed as rational expressions of degree 4 in the coordinates of the segment endpoints. The coordinates of the midpoints are thus rational expressions of degree at most 8.

The transversal to the four lines through (any) one of these midpoints intersects line ℓ_2 and lies in the plane containing line ℓ_3 and the considered midpoint; the coordinates of the intersection point between this plane and ℓ_2 are rational expressions of degree at most 19. Finally, determining whether a transversal (to the four lines) through two points whose coordinates are rational expressions of degree 8 and 19 is a transversal to each of the four segments can be done, as in the proof of Lemma 8, using side operators. Hence, we can decide whether the four segments admit infinitely

many transversals with a predicate of degree at most 36 since the Plücker coordinates of the line transversal are of degree at most 35.

Now, if the four segments admit finitely many transversals, we can determine the number of transversals as follows. As mentioned above, the set of transversals can be parameterized by intervals on a line and the interval endpoints correspond to transversals that go through a segment endpoint. A transversal is isolated if and only if it corresponds to an interval that is reduced to a point. Thus, a transversal is isolated only if it goes through two distinct segment endpoints (the segments necessarily have distinct endpoints since, by assumption, their supporting lines are pairwise skew and thus no segment is reduced to a point). Determining whether the lines through two distinct endpoints intersect the other segments can easily be done, as described in the proof of Lemma 8, by computing the sign of side operators which are here of degree 3 in the coordinates of the segment endpoints. \square

Lemma 10. *Given four segments in \mathbb{R}^3 whose supporting lines are not pairwise skew and admit infinitely many line transversals, determining the number of their line transversals can be done with a predicate of degree 7 in the coordinates of their endpoints.*

Proof. First, note that testing whether two segments intersect can be done using side operators with a predicate of degree 3. The four lines containing the segments are not pairwise skew and they admit infinitely many line transversals. Thus, they are all concurrent or they all lie in a plane H , with the possible exception of a group of one or more that all meet that plane at the same point [15]. Four cases may occur:

- (i) all four lines lie in a plane H ,

- (ii) three lines lie in a plane H and the fourth line intersects H in exactly one point,
- (iii) two lines lie in a plane H and two other lines intersect H in exactly one and the same point,
- (iv) three lines are concurrent but not coplanar.

Differentiating between these cases can be done by determining whether sets of four segment endpoints are coplanar (which is a predicate of degree 3). We study each case in turn.

Case (i). The four segments are coplanar. Any component of transversals contains a line through two distinct segment endpoints. Hence the four segments have finitely many transversals if and only if any line through two distinct endpoints that is a transversal to the four segments is an isolated transversal. This only occurs¹ (see Figure 5.1(b)) when the transversal goes through the endpoints of three segments such that the segment, whose endpoint is in between the two others, lies (in H) on the opposite side of the transversal than the two other segments. This can be tested by computing the sign of scalar products and side operators between the transversal and the lines through a point o not in H and the segment endpoints (see Figure 5.1(b)). This leads to a predicate of degree 4.

Case (ii). Three lines lie in a plane H . Testing whether the fourth segment intersects the plane H can easily be done by computing the point of intersection between H and the line containing the fourth segment, leading to a predicate of degree 3. If the fourth segment does not intersect plane H , the four segments have no transversal

¹For simplicity, we do not discuss here the case where the line transversal contains one of the four segments.

unless the first three segments are concurrent, in which case the four segments have one or infinitely many transversals depending on whether the four lines supporting the segments are concurrent. Otherwise, let p denote the point of intersection. We assume that the three segments in H are not concurrent; otherwise the four segments have infinitely many transversals. Thus, any component of transversals contains a line through p and through a segment endpoint. Hence the four segments have finitely many transversals if and only if any line through p and a segment endpoint that is a transversal to the four segments is an isolated transversal. Testing whether such a line is a transversal to all segments can be done, as in the proof of Lemma 8, by computing the sign of side operators of the line transversal and of lines through a segment endpoint and a point o not in H ; the coordinates of point p are rational expressions of degree 4; thus the Plücker coordinates of the transversal have degree at most 6, which leads to a predicate of degree 7. Such a line transversal is isolated (see Figure 5.1(c)) if and only if² the transversal goes through two endpoints of two distinct segments that lie on the same side (in plane H) of the transversal or not, depending whether p is in between the two endpoints or not. This test can be done by computing the sign of scalar products and side operators between the transversal and the lines through a point o not in H and the segment endpoints (see Figure 5.1(c)). This test also leads to a predicate of degree 7. We can thus determine the number of isolated transversals with a predicate of degree 7.

Case (iii). Two lines lie in a plane H and two other lines intersect H in exactly one and the same point. (Note that there may be two instances of plane H for a given

²We assume here for simplicity that the line transversal contains no segment.

configuration.) This case is similar to Case (ii).

Case (iv). Three lines are concurrent but not coplanar. If none of the three corresponding segments intersect, they have no common transversal. If only two segments intersect, the three segments have exactly one transversal; checking whether that transversal intersects the fourth segment can easily be done with a predicate of degree 3. Now, if the three segments intersect, then the four segments have infinitely many transversals if they are concurrent or if their supporting lines are not concurrent. Otherwise, if the four segments are not concurrent but their supporting lines are, the four segments then have a unique transversal. This can also be checked with a predicate of degree 3. □

We can now conclude the proof of Theorem 7. By Theorem 6, we can determine with a predicate of degree 22 whether the four lines containing the four segments admit finitely many transversals in $\mathbb{P}^3(\mathbb{R})$. If the four lines admit finitely many transversals, then, by Lemma 8, determining the number of transversals to the four segments can be done with a predicate of degree 36. Assume now that the four lines admit infinitely many transversals. Note that determining whether the input lines are pairwise skew can easily be done with a predicate of degree 3. Thus, by Lemmas 9 and 10, determining whether the four segments admit 0, 1, 2, 3, 4, or infinitely many line transversals can be done by a predicate of degree at most 36. Hence, we can determine the number of transversals to four segments with a predicate of degree 36. □

5.2.4 Ordering Planes through Two Fixed Points, Each Containing a Third (Rational) Point or a Line Transversal

Let ℓ be a line defined by two points v_1 and v_2 , and let $\vec{\ell}$ be the line ℓ oriented in the direction $\overrightarrow{v_1v_2}$.

We define an ordering of all the planes containing ℓ with respect to the oriented line $\vec{\ell}$ and a reference point O (not on ℓ). Let P_0 be the plane containing O and ℓ , and let P_1 and P_2 be two planes containing ℓ . We say that $P_1 < P_2$ if and only if P_1 is encountered strictly before P_2 when rotating counterclockwise about $\vec{\ell}$ a plane from P_0 (see Figure 5.2a).

Let p_i be any point on plane P_i but not on ℓ , for $i = 1, 2$, and let $D(p, q)$ denote the determinant of the four points (v_1, v_2, p, q) given in homogeneous coordinates.

Lemma 11. *With $\chi = D(O, p_1) \cdot D(O, p_2) \cdot D(p_1, p_2)$, we have:*

- (a) *If $\chi > 0$ then $P_1 > P_2$.*
- (b) *If $\chi < 0$ then $P_1 < P_2$.*
- (c) *If $\chi = 0$ then*
 - (i) *if $D(p_1, p_2) = 0$, then $P_1 = P_2$,*
 - (ii) *else if $D(O, p_1) = 0$, then $P_1 < P_2$,*
 - (iii) *else $P_1 > P_2$.*

Proof. Assume first that $D(O, p_1) \cdot D(O, p_2) > 0$, that is, that p_1 and p_2 lie strictly on the same side of the plane P_0 (see Figure 5.2b). Then the order of P_1 and P_2 is determined by the orientation of the four points (v_1, v_2, p_1, p_2) , that is, by the sign of $D(p_1, p_2)$. It is then straightforward to notice that $P_1 > P_2$ if and only if $D(p_1, p_2) > 0$.

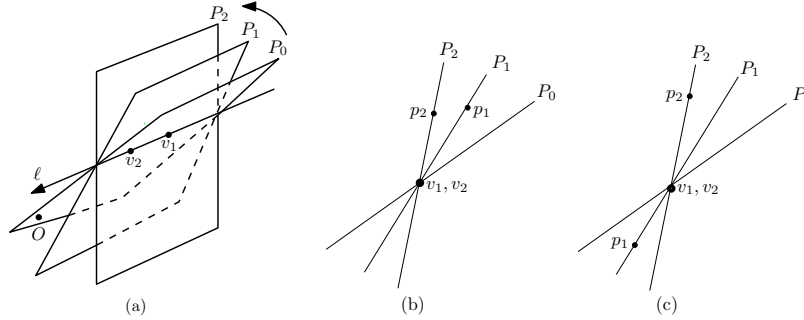


Figure 5.2. Planes P_1 and P_2 such that $P_1 < P_2$

Hence, if $\chi > 0$, then $P_1 > P_2$ and, if $\chi < 0$, then $P_1 < P_2$.

Suppose now that $D(O, p_1) \cdot D(O, p_2) < 0$, that is, that p_1 and p_2 lie strictly on opposite sides of the plane P_0 (see Figure 5.2c). The order of P_1 and P_2 is then still determined by the sign of $D(p_1, p_2)$. However, $P_1 > P_2$ if and only if $D(p_1, p_2) < 0$. Hence, we have in all cases that, if $\chi > 0$, then $P_1 > P_2$ and, if $\chi < 0$, then $P_1 < P_2$.

Suppose finally that $\chi = 0$. If $D(p_1, p_2) = 0$, then p_1 and p_2 are coplanar and $P_1 = P_2$. Otherwise, if $D(O, p_1) = 0$, then $P_0 = P_1$; thus P_1 is smaller than all other planes (containing $\vec{\ell}$), and in particular $P_1 \leq P_2$. Furthermore, since $D(p_1, p_2) \neq 0$, $P_1 \neq P_2$ and thus $P_1 < P_2$. Otherwise, $D(O, p_2) = 0$ and we get similarly that $P_2 < P_1$. \square

Computing a point on a plane defined by ℓ and a line transversal. We want to order planes P_i that are defined by line ℓ and either a rational point not on ℓ , or by a line transversal to ℓ and three other lines. In the latter case, we consider a point on the line transversal (which is non-rational, in general; see Lemma 2). The following lemma tells us that, in general, such a plane P_i contains no rational points

outside of ℓ , and that in the cases where it does contain such a rational point, the line transversal is then rational. Hence, if the points computed on the line transversal, as described in Lemma 2, are not rational, there is no need to search for simpler points on the plane (but not on ℓ).

Lemma 12. *The plane P containing a rational line ℓ and a line transversal to ℓ and three other segments, each determined by two rational points, contains in general no rational points except on ℓ . Furthermore, if plane P contains a rational point not on ℓ then the line transversal is rational.*

Proof. Suppose that the plane P contains a rational point p not on ℓ . Then the plane contains three (non-collinear) rational points, p and two points on ℓ , and thus P is a rational plane. This plane intersects the three other segments in three points, all of which are rational and lie on the transversal. So the transversal is a rational line, which implies that the discriminant $B^2 - 4AC$ in Equation (5.5) is a square, which is not the case in general. \square

Comparing two planes. We want to order planes P_i that are defined by either line ℓ and another (input rational) point not on ℓ , or by line ℓ and a line transversal to ℓ and three other lines.

By Lemma 11, ordering such planes about ℓ amounts to computing the sign of determinants of four points (in homogeneous coordinates). Two of these points are input (affine rational) points on ℓ (v_1 and v_2) and each of the two other points is either an input (affine rational) point r_i , $i = 1, 2$, or is, by Lemma 2 (and Lemma 12), a point u_i whose homogeneous coordinates have degree at most 19 (in the coordinates of the

input points) or a point of the form $p_i + q_i \sqrt{\Delta_i}$, $i = 1, 2$, where the Δ_i have degree 22 and where the p_i and q_i are points with homogeneous coordinates of degree at most 17 and 6, respectively. If the four points are all input points, then the determinant of the four points has degree 3 in their coordinates.

If only three of the four points are input points, then the determinant of the four points is either a polynomial of degree 22 or it has the form $D(p_1, r_1) + D(q_1, r_1) \sqrt{\Delta_1}$ where the degrees of the $D()$ are 20 and 9, respectively, in the coordinates of the input points. Hence, by Lemma 4, the sign of this expression can be determined with a predicate of degree 40.

Finally, if only two of the four points are input points, then the determinant has one of the following forms (depending on whether the quadruples of lines defining the transversals are parallel to a common plane); the degrees are given in terms of the coordinates of the input points:

- (i) $D(u_1, u_2)$, which is of degree 40.
- (ii) $D(u_1, p_1) + D(u_1, q_2) \sqrt{\Delta_1}$, where the $D()$ have degree 38 and 27, respectively.
- (iii) $D(p_1, p_2) + D(q_1, p_2) \sqrt{\Delta_1} + (D(p_1, q_2) + D(q_1, q_2) \sqrt{\Delta_1}) \sqrt{\Delta_2}$, where the $D()$ have degree 36, 25, 25, and 14, respectively.

Hence, by Lemma 5, the sign of these expressions can be determined with a predicate of degree at most 144 (and the bound is reached in the worst case). We thus get the following result.

Theorem 13. *Let ℓ be an oriented line defined by two points, let p_0 be a point not on ℓ , and let P_0 be the plane determined by ℓ and p_0 . Given two planes P_1, P_2 containing ℓ there is a predicate that determines the relative order of P_1 and P_2 about ℓ with*

respect to P_0 having the following degree in the coordinates of the input points:

- (i) degree 3 if $P_i, i = 1, 2$ are each specified by a (input) point p_i ;
- (ii) degree 40 if P_1 is specified by a point p_1 and P_2 is determined by a line transversal to ℓ and three other lines ℓ_1, ℓ_2, ℓ_3 , each specified by two (input) points;
- (iii) degree 144 if $P_i, i = 1, 2$ are each determined by a line transversal to ℓ and three other lines $\ell_{i,1}, \ell_{i,2}, \ell_{i,3}$, each specified by two (input) points.

Remark 14. Note that, if the leftmost (instead of the rightmost) 4×4 submatrix of the matrix of Plücker coordinates (in (5.1)) is used for computing line transversals (see Remark 3) then the predicates of Theorem 13 have degree 3, 46, and 168.

5.3 Experiments

In this section, we report the results of experiments that analyze the behavior of the predicate for ordering, in a rotational sweep about a line, two planes each defined by a line transversal to four lines, which is the predicate related to Theorem 13(iii). The degree of the procedure we use for evaluating this predicate is 168 because we use for computing line transversals to four lines the code of Redburn [87], which, as noted in Remarks 3 and 14, leads to degree 168 instead of 144 as in Theorem 13(iii).

The standard approach to comparing two such planes is to first evaluate the predicate using fixed-precision interval-arithmetic. This is very efficient but may fail when the sign of an expression cannot be successfully determined because the result of the evaluation of the expression is an interval that contains zero. If this happens, the answer to the predicate is then obtained by either evaluating exactly the expression (and thus its sign) using exact arithmetic or by increasing the precision

of the interval arithmetic until either the result of the evaluation of the expression is an interval that does not contain zero or the separation bound is attained (see for instance [16, 73, 93, 105]); in both approaches the computation is much slower than when using fixed-precision interval-arithmetic. We are thus interested in determining how often the fixed-precision interval-arithmetic evaluation of our predicate fails.

To test our predicate, we generate pairs of planes, each defined by two lines, one chosen at random and common to the two planes, and the other defined as a transversal to the common line and to three other random lines. We are interested in evaluating our predicate in the case where the two planes are very close together, that is, when there is significant risk of producing an error when using finite-precision floating-point arithmetic.

We generate two sets of four lines. Each line of the first set is determined by two points, all of whose coordinates are double-precision floating-point numbers chosen uniformly at random from the interval $[-5000, 5000]$. The second set of lines is obtained by perturbing the points defining three of the lines of the first set; the fourth line is not perturbed and is thus common to the two sets. To perturb a point p , we translate it to a point chosen uniformly at random in a sphere centered at p , with radius ϵ .

We compute, for each of these two sets of four lines, a line transversal. If either set of four lines does not admit a transversal (which happens roughly 24% of the time), we throw out that data and start again. Otherwise, we choose a transversal in a consistent way for the two sets of four lines, that is, such that one transversal converges to the other when ϵ tends to zero. Each transversal, together with the

predicates \ ϵ	10^{-12}	10^{-10}	10^{-8}	10^{-6}	10^{-4}	10^{-2}
degree 168	99.6%	50.4%	7.6%	0.8%	0.08%	0.008%
degree 3	99.5%	8.2%	0.08%	0.001%		

Table 5.1. Percentages of failure of the degree 168 and degree 3 predicates using double-precision floating-point interval-arithmetic, for ϵ varying from 10^{-12} to 10^{-2} .

common line, defines a plane.

For various values of ϵ , varying from 10^{-2} to 10^{-10} , we evaluate the predicate using double-precision floating-point interval arithmetic until we obtain 1000 pairs of planes for which the computation of the predicate fails. We measure the percentage of time that the computation fails. The results of these experiments are shown in Table 5.1.

We observe, as expected, that when ϵ is sufficiently small (10^{-10}), that is, when the two planes are often close enough to each other, the fixed-precision interval-arithmetic predicate fails with high probability and that this probability decreases as ϵ increases. When $\epsilon = 10^{-2}$, the probability of failure is close to zero. Finally, we have also observed that the predicate fails when the angle between the two planes is less than roughly 10^{-8} radians, which is, of course, independent of ϵ .

Note that the percentage of failure of the degree 168 predicate using fixed-precision interval-arithmetic is, as expected, high compared to lower-degree predicates. Table 5.1 shows the failure rate for the degree 3 predicate related to Theorem 13(i). We use the same experimental scheme as above, that is, we choose at random three points that define a plane and perturb one of these points by at most ϵ .

All the experiments were done on a i686 machine with AMD Athlon 1.73 GHz CPU and 1 GB of main memory using the CGAL interval number type with double-

precision floating-point numbers [19].

5.4 Discussion

Although the algebraic degrees of the predicates computed with the standard method is high, we used it in our implementation nevertheless, because, at the time when we implemented these predicates, this was the only known method, and also because there was existing software, developed by Redburn [87], which allowed us to provide a first prototype of our implementation.

A theoretical result presented by Devillers *et al.* has since shown that the same predicates can be computed in much lower algebraic degrees [28]. Developing alternative software that uses the result of Devillers *et al.* could be future work.

5.5 Bibliographic Notes

The work presented in this chapter first appeared in the Journal of Computational Geometry: Theory and Application [40], which was a special issue of invited papers from the 18th Canadian Conference on Computational Geometry in 2006.

Chapter 6

Experimental Study of the Size of the 3D Visibility Skeleton

We address in this chapter the problem of computing and estimating the size of the visibility skeleton of k disjoint polytopes of total complexity n in generic position. In fact, the visibility skeleton we studied is defined by the visual event surfaces, and we measure the size of the skeleton as the number of its vertices (since vertices have constant degree under general position assumptions). A precise definition of the visibility skeleton defined by the visual event surfaces is given in Section 2.2.2, and we briefly recall it in Section 6.1. We then present experiments on k disjoint polytopes of size n/k , with vertices on congruent spheres randomly distributed with fixed densities in a given (spherical) universe. We perform these experiments for (i) up to 230 polytopes with up to 1 700 edges and (ii) up to 130 polytopes with up to 9 000 edges. These experiments show that the number of vertices of the visibility skeleton is roughly $C k\sqrt{nk}$, where the observed constant C varies with scene density but

remains small (≤ 5 in our setting). Our experiments also indicate that the average running time of our implementation is $O(n^{3/2}k \log k)$. By contrast, the theoretical worst-case running time of the algorithm in our setting is $O(n^2k^2 \log k)$.

These results are significant for three reasons. First, this is the first experimentally determined asymptotic estimate of the number of vertices of the 3D visibility skeleton that takes into account not only the total number n of edges, but also the number k of polytopes in the scene. The results show that the size of the visibility skeleton may be sub-quadratic; in particular, they show a sub-linear growth in n and a sub-quadratic growth in k . Second, assuming that the size of the silhouette of a polytope on n/k vertices is $O(\sqrt{n/k})$, our results show that we may express the size of the visibility skeleton as a function that is linear in the size of the silhouette and quadratic in the number of polytopes; that is, the number of vertices in the scene impacts the size of the visibility skeleton only insofar as it increases the size of the silhouettes. Finally, our results indicate that there is no large constant hidden in the big-Oh notation.

We have conducted experiments only for scenes consisting of disjoint polytopes. However, this is less of a limitation than it may at first appear; it is reasonable to expect that our bounds will also roughly apply to the case of, for instance, arbitrary polyhedra decomposed into k convex tessellated surface patches of total complexity n .

In the next section, we briefly recall the definition of the visibility skeleton that we study in this chapter. We discuss our experimental setting in Section 6.2, present our experimental results in Section 6.3, compare the experimental results of using number type `double` to `filtered_exact` in Section 6.4, and finally summarize and discuss our results in Section 6.5.

6.1 The Visibility Skeleton of a Set of Polytopes

In this chapter, we study not the full one-skeleton [34, 36] but rather the skeleton that is defined by the visual event surfaces [25, 26]. The skeleton thus defined contains only those arcs that correspond to local changes in the view, *i.e.*, arcs such that, when a viewpoint crosses the surface generated by the set of segments corresponding to the arc, a new polytope comes into view or a previously seen polytope disappears; in particular, we do not consider the appearance or disappearance of a polytope feature as a change in the view. More precisely, those are the arcs of type EEE , and type VE if only its set of supports consists of an edge and a vertex that define a plane tangent to their respective polytopes. The 3D visibility skeleton thus defined consists only of vertices that are incident to those arcs, that is, the vertices of type $EEEE$, VEE , FEE , and VV if only its set of supports consists of two vertices that lie on a plane that is tangent to their two respective polytopes.

We study the number of vertices of the visibility skeleton thus defined and refer to it, with abuse of notation, as the size of the visibility skeleton. Since, under our general position assumptions, the degree of each skeleton vertex is bounded by a small constant, the actual size of the skeleton, including the arcs, will be a small constant factor away from what we measure.

6.2 Setting of the Experiments

6.2.1 The Model

The input scenes are generated in two phases. First, we generate a set of disjoint spheres and, in phase two, we generate one convex polytope in each sphere.

In phase one, for a given number of spheres k and scene density μ , we generate k unit spheres in a spherical universe of center O and radius R , where $R = \sqrt[3]{k/\mu}$ (that is, $\mu = k \frac{4}{3}\pi / \frac{4}{3}\pi R^3$). The centers of the spheres are chosen, one by one, uniformly from the ball centered at O of radius $R-1$. When a newly generated sphere intersects any existing one, the new sphere is discarded. Note that the spheres are not uniformly distributed since the new sphere is not chosen independently of the previous ones.

In phase two, for each sphere, we generate a set of vertices using the hypercube rejection method [66] (p.131-132). That is, we first generate the vertices uniformly in a cube which is circumscribed to the sphere. We reject those that fall out of the sphere, and project the remaining vertices on the surface of the sphere. This results in a uniform distribution of the vertices on the surface of sphere. We then compute the convex hull. This defines one polytope for each sphere and guarantees that all the polytopes are disjoint. We note that the density of the polytopes in the scene is somewhat less than the density μ of the spheres of phase one, however it can serve as an indication of the pairwise distance of the input polytopes.

Here we emphasize that our objective in this chapter is not to study uniformly distributed disjoint polytopes approximating spheres. We have used this scene model because it provides a simple way to generate large scenes containing disjoint polytopes.

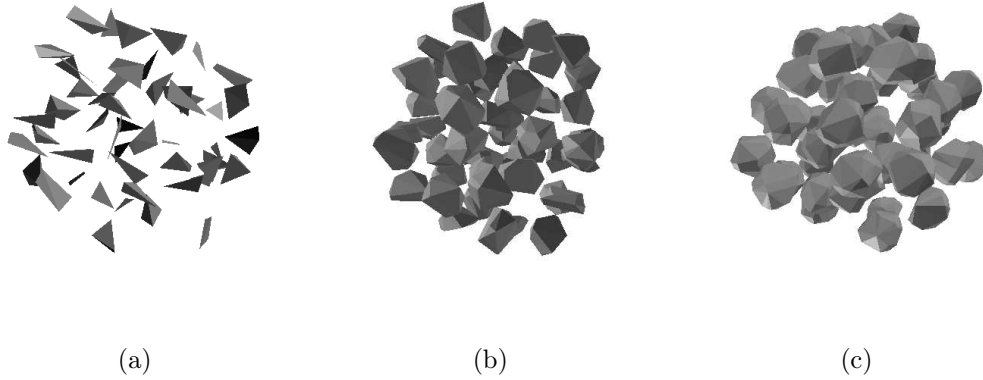


Figure 6.1. Three sample scenes of $k = 50$ polytopes where n/k , approximately the number of edges on each polytope, is equal to (a) 6, (b) 42, and (c) 84. The scene density $\mu = 0.3$ in all cases.

Furthermore, it allows us to compare with previous theoretical results [27].

6.2.2 The Experiments

We consider scenes of polytopes, as defined above, depending on three parameters, the number k of polytopes, the total number n of polytope edges, and the scene density μ . We perform three suites of experiments in which we measure the number of visibility skeleton vertices.

In Suite I, we fix the scene density μ and the number n/k of edges per polytope. For different values of k , we generate scenes of k polytopes each having n/k edges. We perform experiments for $\mu = 0.3, 0.05$ and 0.01 and for $n/k \approx 6, 42$ and 84 .¹ A sample scene with $k = 50$ is shown in Figure 6.1. For each value of n/k , we vary the number k of polytopes as follows: (a) when $n/k \approx 6$, we vary k from 10 to 190 (giving $n \in [75, 1\,425]$), (b) when $n/k \approx 42$, we vary k from 10 to 130 (giving $n \in [400, 5\,200]$),

¹In fact, we generate polytopes whose numbers of vertices range in $[4, 6]$, $[15, 17]$ and $[30, 32]$, respectively. The number of edges per polytope is thus not actually fixed but varies slightly; the polytopes we generated have, on average, 7.5, 40, and 85 edges, respectively.

and (c) when $n/k \approx 84$, k varies from 10 to 110 (giving $n \in [850, 9\,400]$). As we will see, the number of visibility skeleton vertices appears to be roughly $C_\mu k\sqrt{nk}$ in these experiments where C_μ is a constant that depends on the density.

In Suite II of our experiments, we also fix the scene density μ to 0.3 and vary the number n/k of edges per polytope for fixed numbers of polytopes. Namely, we consider $k = 30, 60$, and 90 and vary n/k from 6 to 102. As we will see, these experiments confirm that when n/k varies (in the given range), the complexity observed in the first set of experiments holds.

In Suite III of our experiments, we again fix the scene density μ to 0.3 and vary n a lot for each scene. Specifically, we vary the number of polytopes k from 10 to 150 in step 10. And for each k , we test on three scenes whose number of polytope edges are randomly generated in range between $[4, 24]$, $[4, 34]$, and $[4, 44]$. We note the difference of this Suite of experiments from Suite I and II is that the number of polytope edges varies a lot within each scene. Again, we will see in the next section that the previously observed complexity holds as well in this Suite of experiments.

We remark that, for each type of scene, that is, for each chosen scene density μ , polytope complexity n/k , and number of polytopes k , we randomly generate one scene for our experimental study. Since, on small sets of scenes, we have generated ten scenes for each type, the observed standard deviation of the computed number of skeleton vertices on the 10 test scenes is small to the mean. In Figure 6.2, we show the mean and standard deviation of the obtained results on ten scenes of each type, where the types of scenes vary as: when $\mu = 0.3$, $n/k = 12$, k varies from 10 to 150 in step 20; and when $\mu = 0.01$, $n/k = 36$, k varies from 10 to 80 in step 10. We observe

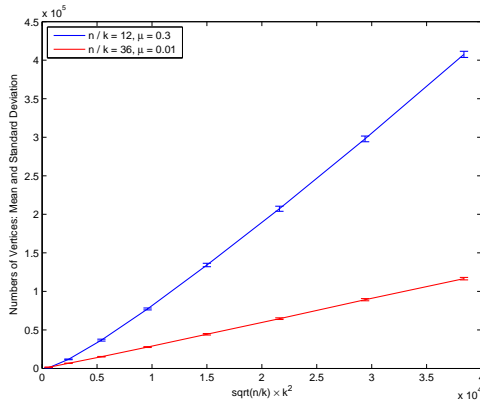


Figure 6.2. Mean and standard deviation of the number of computed skeleton vertices on ten scenes for each type of scene.

the standard deviation is within 5% of the mean, and when plotting the data with our obtained parameter, as shown in Figure 6.2, the displayed error does not affect our data analysis.

We finally note that a scene with density $\mu = 0.3$ is very dense (see Figure 6.1 and recall Kepler’s Theorem that the density of any sphere packing in 3D space is at most $\pi/3\sqrt{2} \approx 0.74$). Density $\mu = 0.3$ is close to the highest density we can reach in a reasonable amount of time with our scene generation scheme.

6.2.3 Number Type and Machine Characteristics

All the experiments use `filtered_exact` number type. They were all done on an *i686* machine with *Pentium* 2.80 GHz CPU running Linux, with 2 GB of main memory. Running time was measured with the `getrusage()` command and the `ru_utime` attribute.

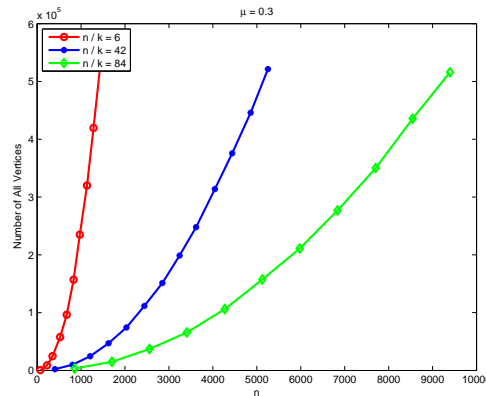


Figure 6.3. Suite I ($\mu = 0.3$): total number of skeleton vertices in terms of n , the number of edges in the scene.

6.3 Experimental Results and Analysis

6.3.1 Number of Skeleton Vertices in Terms of n

We present, in terms of the total number n of edges in the scene, the total number of visibility skeleton vertices of Suite I of our experiments in Figure 6.3, for $n/k \approx 6$, 42, and 84, respectively.

For a given value of n/k , the number of skeleton vertices appears to be quadratic in n (see Figures 6.3 and 6.4 (a)). Notice also that for a fixed n , the size of the output drops drastically when the number k of polytopes gets smaller (see Figure 6.3). These observations are consistent with the theoretical bounds, that is, the worst-case output size is in $\Theta(n^2k^2)$ [14].

The rest of this section analyses the output size in terms of n and k .

6.3.2 Number of Skeleton Vertices in Terms of n and k

We present, in Figures 6.4 and 6.5, the number of skeleton vertices in terms of $k\sqrt{nk} = k^2\sqrt{n/k}$.² In all the figures, the number of these skeleton vertices appears to be linear in this parameter with a constant that depends on the scene density μ . For $\mu = 0.3, 0.05$, and 0.01 , the constant is roughly 5, 4, and 3. The constant appears to decrease in terms of μ which is consistent with the intuition that the constant goes to zero as the scene density goes to zero (since the probability that there exists a line transversal to three polytopes goes to zero as the density goes to zero and that the number of vertices of type VV is asymptotically negligible).³

Note that, for any fixed density μ and any given value of $k^2\sqrt{n/k}$, the number of these skeleton vertices varies very little in terms of the polytope complexity n/k (Figure 6.4(a)) and in terms of the number of polytopes (Figure 6.4(b)). Even when the input scenes that consist of polytopes (with vertices on spheres) whose complexities vary a lot, and when considering n/k as the average complexity of the input polytopes, the observed results are still very well fit by the function $C_\mu k^2\sqrt{n/k}$ (Figure 6.5). This suggests that $C_\mu k^2\sqrt{n/k}$ is a good predictor of the number of these skeleton vertices regardless of the polytope complexity, at least for the scene densities μ and the ranges of n/k used here.

Our experiments thus indicate that, in our setting, the number of skeleton vertices

²For experiment of Suite III, we choose n/k as the average number of polytope edges in each scene and show results in figure 6.5.

³Note that this observation is also consistent with a related experimental study on the impact of the density on the constant of the asymptotic linear complexity of the 2D visibility skeleton of randomly distributed unit discs [41] (also in Chapter 3).

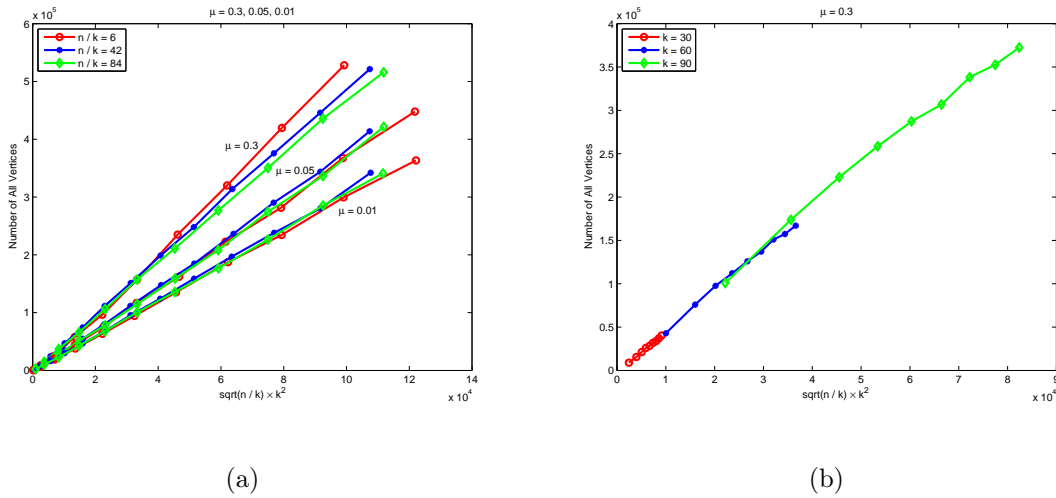


Figure 6.4. Total number of skeleton vertices in terms of $k^2 \sqrt{n/k}$ when (a) the polytopes have a constant (n/k) number of edges (Suite I), and (b) the number k of polytopes is constant (Suite II).

is roughly

$$C_\mu k^2 \sqrt{n/k},$$

where C_μ is a constant that depends on the density μ of the scene. The experiments hint that this constant is small and is a decreasing function of μ .

This observed complexity is, as expected, much smaller than worst-case bounds. Recall that, for k polytopes with n edges in total, the worst-case number of skeleton vertices is $\Theta(n^2 k^2)$ [14]. Also, if the silhouettes of the polytopes have size $\sqrt{n/k}$ in the worst case, the worst-case number of skeleton vertices is $O(nk^3 \sqrt{n/k})$ [48, §6.7]. These worst-case bounds are much larger than our observed size (by a factor $n\sqrt{nk}$ and nk).

We analyze below the observed complexity of $C_\mu k^2 \sqrt{n/k}$ in terms of (i) k when the complexity of the polytopes is constant, and (ii) the silhouette size of the polytopes when the number k of polytopes is constant.

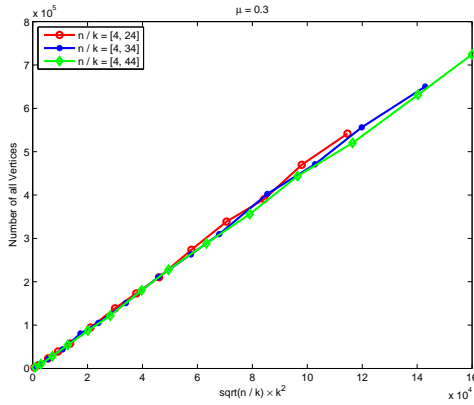


Figure 6.5. Total number of skeleton vertices in terms of $k^2 \sqrt{n/k}$ for polytope complexity (n/k edges) varying in the range of [4 - 24], [4 - 34], and [4 - 44] (Suite III).

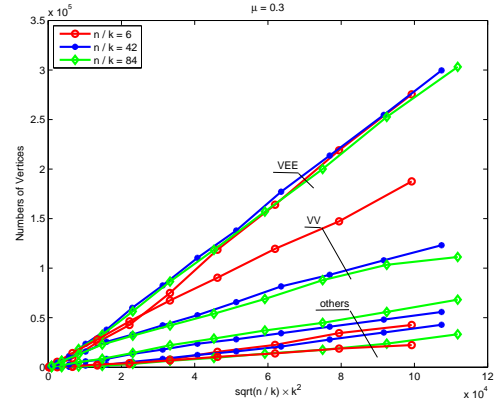


Figure 6.6. The number of vertices in terms of $k^2 \sqrt{n/k}$ as tested on Suite I (k polytopes having a constant, n/k , number of edges; $\mu = 0.3$).

Analysis of the number of skeleton vertices in terms of k . If each polytope has constant complexity (*i.e.*, n/k in $\Theta(1)$), our experiments exhibit a quadratic growth (in terms of k) of the number of skeleton vertices. This is consistent with previous experiments [36, 47] in which the scenes consist of polygons of constant complexity and is also consistent with the best known theoretical *expected* upper bound of $O(k^2)$ [27] corresponding to our setting. However, this contradicts the intuitive linear bound of $\Theta(k)$ when n/k is constant. Recall that in [27], for k randomly distributed congruent spheres, the expected number of visibility events is linear and that, for constant-size polytopes of bounded aspect ratio inside such spheres, the expected number of visibility events is linear for events that occur sufficiently inside the universe, but it is upper bounded by $O(k^2)$ for events near the boundary of the universe. It is possible that the expected upper bound of $O(k^2)$ is tight but it is also possible that our experiments did not reach the asymptotic behavior of the complex-

ity. If this is the case, it is then reasonable to believe that our experimental estimate of the complexity is an overestimate.

Analysis of the number of skeleton vertices in terms of the silhouette size of the input polytopes. If we fix the number k of polytopes and vary the total number n of edges, our experiments show that the number of skeleton vertices depends linearly on $\sqrt{n/k}$. We argue below that this means that, in our setting, when k is fixed, the number of skeleton vertices depends linearly on the silhouette size of the input polytopes and suggest an explanation for this.

Recall that, for any polyhedron of size $\Theta(m)$, the size of its silhouette viewed from a random point is $O(\sqrt{m})$ under some reasonable hypotheses [49] (see also [65] for the special case of polyhedra that approximate spheres). Since the vertices of the polytopes we consider are randomly distributed on a sphere, it is reasonable to assume that the size of the silhouette does not depend much on the choice of the viewpoint. In other words, for any polytope with n/k edges we consider, it is reasonable to assume that its silhouette has size $O(\sqrt{n/k})$ from any viewpoint. Hence, when k is fixed, the number of skeleton vertices depends linearly on the silhouette size of the input polytopes.

We offer the following intuitive explanation for this observation. Consider the arcs of type EEE of the skeleton. The endpoints of these arcs are vertices of type VEE , FEE or $EEEE$. When the number k of polytopes is fixed and the number n of edges tends to infinity, the polytopes tend to spheres and the segments corresponding to vertices of type $EEEE$ converge to segments that are tangent to four spheres; hence, in our setting, the number of $EEEE$ vertices converges to a constant. A similarly

remark holds for those VEE vertices that correspond to intersections of two arcs of types VE and EEE (thus corresponding to segments tangent to three polytopes while lying in planes that are tangent to two of them). Moreover, in the successive refinements of polytopes as n increases, each EEE arc incident to an $EEEE$ vertex or one of the above VEE vertices will become a sequence of EEE arcs joined at VEE vertices (that is, subdivision vertices such that the sets of supports are invariant along the subdivided arcs). For such a sequence of arcs, the number of these VEE vertices is the number of polytope vertices encountered by a maximal free line segment as it slides from the segment corresponding to one end of the sequence to the other, while remaining tangent to the three polytopes (nearly spheres) involved. The number of such polytope vertices is, intuitively, at most the worst-case size of the silhouette of each polytope, which we have assumed to be in $O(\sqrt{n/k})$. As a polytope gets more complex and tends to a sphere, the subset of lines in the line space that are tangent to the polytope on its vertices tends toward the subset of lines that are tangent to the sphere. This is also the case for lines tangent to the polytope on its faces. For this reason, the number of FEE vertices is asymptotically the same as that of VEE vertices.

Thus, intuitively, we can expect that, for fixed k , (i) the number of vertices of type $EEEE$ converges to a constant as n goes to infinity, (ii) the number of vertices of type VEE or FEE is in $O(\sqrt{n/k})$ times the number of $EEEE$ vertices, and thus that (iii) the number of VEE and FEE vertices is in $O(\sqrt{n})$ (for k fixed). In our experiments, we have observed (ii) and (iii) (shown in Figure 6.7 and 6.8 respectively), but not (i). In Figure 6.6, the number of VEE vertices is much larger than the number of $EEEE$

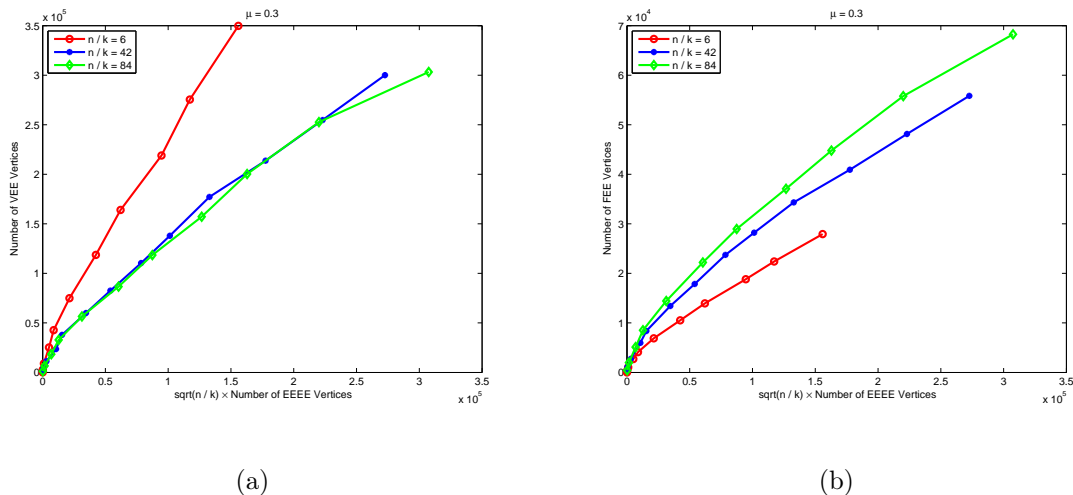


Figure 6.7. Number of (a) *VEE*, (b) *FEE* vertices in terms of number of *EEEE* vertices (Suite I, $\mu = 0.3$).

vertices, which is consistent with the previous discussion, while the convergence of *EEEE* is not seen in our experimental range.

Finally, the number of *VV* vertices is, intuitively, bounded by the product of the number of pairs of polytopes that are mutually visible and the size of the polytope silhouettes. In our experiments (Figure 6.6) we observe a complexity of roughly $\Theta(k^2 \sqrt{n/k})$.

6.4 Double versus Filtered_exact

We also ran all the experiments of Suite I using number type `double` and compared the computed results with those we obtained by using number type `filtered_exact`. We show the error percentage of the computed skeleton vertices, computed as $(\# \text{filtered_exact} - \# \text{double}) * 100 / \# \text{filtered_exact}$, in Figure 6.9.

From Figure 6.9, we observe that the error percentages, in the range of our ex-

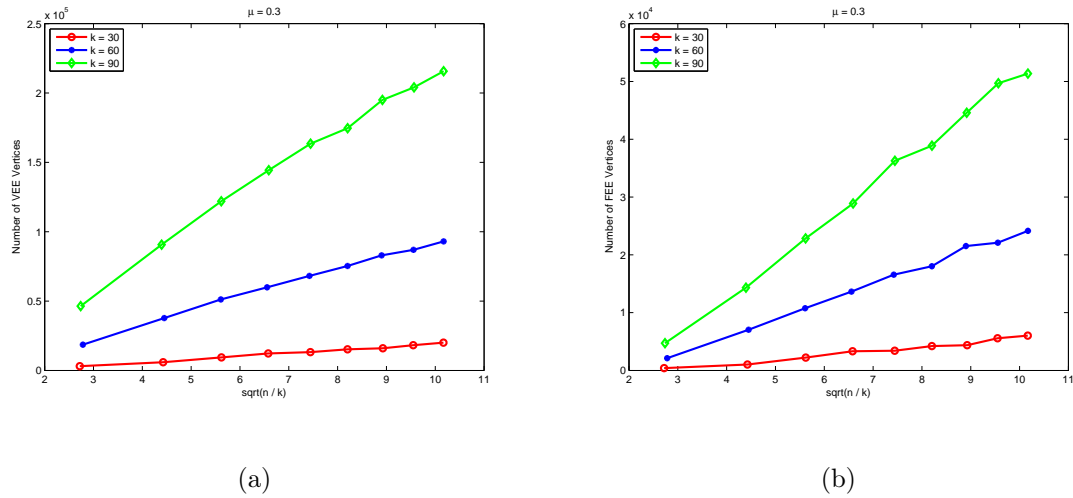


Figure 6.8. Number of (a) VEE, (B) FEE vertices in terms of $\sqrt{n/k}$ (Suite II).

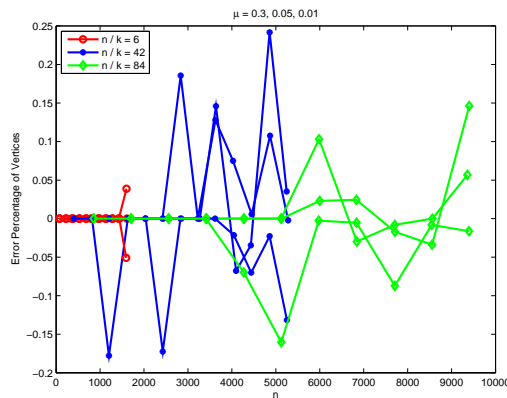


Figure 6.9. Error percentage of computed skeleton vertices when using number type `double` versus `filtered_exact` (Suite I).

periments, are less than 0.25%. In particular, there is no error when the number of polytope edges n is small (< 800 in our experiments).

We notice that the values shown in Figure 6.9 appear both positive and negative. This suggests that using number type `double` causes different types of computation error. Some errors may cause the program to abort. In this case, the number of

skeleton vertices will be undercounted. Note that, from our experimental observation, the abortion rate of our software, when using number type `double`, is roughly about 0.1%. Some other errors may be to simply overcount a skeleton vertex which does not exist, or vice-versa. For example, computing the skeleton vertex of type *VEE* or *FEE* uses predicate `line_segment_intersect`, which tests whether a line and a segment intersect. The computation error of this predicate may overcount, or undercount, a *VEE* or a *FEE* vertex.

In terms of choosing which number type to use, our experimental study indicates that the number type `filtered_exact` is not too slow when one needs to perform exact computation. Compared to `double`, it is about 3 to 4 times slower (see Figure 4.18). However, when one can afford occasional failure in a computation, the number type `double` can be a good choice. Especially when dealing with small input sizes ($n < 800$ in our experimental range), one can even expect exact results.

6.5 Summary and Bibliographic Notes

Our experiments suggest that, in our setting, the number of vertices of the 3D visibility skeleton is $C_\mu k\sqrt{nk}$. The constant C_μ , which depends on the scene density, is no more than 5 for n and k in our experimental range and for the various densities that we studied.

This is the first prediction of the actual size of the 3D visibility skeleton, for reasonably large n , that is expressed in terms of both n and k . Assuming that the size of the silhouette of a polytope with n/k edges is $O(\sqrt{n/k})$, our results show that the size of the visibility skeleton is linear in the size of the silhouette and quadratic in

the number of polytopes. Surprisingly, the constant C_μ is rather small; this indicates that there is no large constant hidden in the big-Oh notation.

The experiments also suggest that the expected running time of our implementation of the sweep plane algorithm is $C'_\mu n \sqrt{n} k \log k$ seconds, where C'_μ depends on the scene density but is, on our machine, no more than $3 \cdot 10^{-4}$ for the considered densities.

Our results indicate that the visibility skeleton is of reasonable size and can be computed exactly in a reasonable length of time.

A succinct version of this chapter, except Section 6.4, has appeared in the Proceedings of the 16th Annual European Symposium on Algorithms [106].

Chapter 7

Computing the 3D Visibility Skeleton

Recall that in Section 2.2.2, we introduced a 3D visibility skeleton defined by visual event surfaces [25, 26]. When considering input consisting of a set of convex disjoint polytopes, a visibility skeleton thus defined consists of only arcs of type EEE , and a subset of type VE , and vertices of types $EEEE$, VEE , FEE , and a subset of type VV . This data structure is a subset of the 3D visibility skeleton defined by Durand *et al.* [34, 36], which consists of the 0D and 1D cells of the 3D visibility complex. For convenience, in this chapter, we refer to the former definition as a *succinct 3D visibility skeleton*, and to the latter one as a *full 3D visibility skeleton*. Moreover, we define the vertices contained in the succinct skeleton as *primary vertices*, and the remaining vertices of the full skeleton as *secondary vertices*.

From the study of Demouth *et al.*, the size of the succinct 3D visibility skeleton is only about 25% to 50% of the full one [25, 26]. However, the skeleton vertices and arcs it contains are sufficient to compute the direct shadow boundaries cast by polytopes. While compact and useful on its own, the succinct 3D visibility skele-

ton does not always contain the necessary visibility information for answering global visibility queries. For example, in global illumination computation, the form factor can be approximated by point-to-area calculations [9] computed at the vertices of the inputs [71, 100]. To compute the point-to-area form factor for vertices, we need all the VE arcs, including those not encoded in the succinct visibility skeleton [34]. Also, when generating high quality shadows, the typical approach of linearly interpolating light intensity within the penumbrae [3, 55, 78] is not always sufficient to express the subtlety of penumbrae cast by polytope features, as shown in Figure 7.1; in particular, arcs FE may be needed even though they do not correspond to visual events. Apart from global illumination, other problems such as visibility culling [103], architectural acoustics [44], or endoscopy [57] also need global visibility information.

The full visibility skeleton, on the other hand, contains all the necessary information for most visibility queries. However its large size has been seen as an impediment for its practical use [34, 36]. In this chapter, we study in detail the 3D visibility skeletons computed from a set of convex disjoint polytopes in general position. We prove that knowing the succinct 3D visibility skeleton is sufficient to compute efficiently the secondary vertices; in particular, these computations can be local, that is, only the vertices and arcs of interest need to be computed. Furthermore, the full skeleton can be computed if necessary.

In terms of computing the full skeleton, we prove that, given k polytopes with n edges in total, the full visibility skeleton can be computed from the succinct one in $O(p \log p + m \log m)$ time, where p is the number of the primary vertices minus the $EEEE$ vertices, and m is the number of secondary vertices. The worst-case size com-

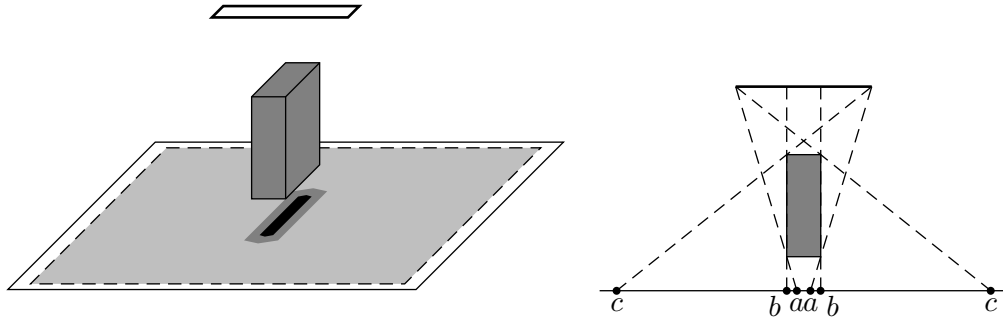


Figure 7.1. Scene representing a shelf in a room with a fluorescent light on the ceiling. The black and white regions represent the umbra and full light regions. The union of the light and dark grey regions corresponds to the penumbra. The dark grey shape represents a portion of the penumbra limited by the trace of FE arcs. In this region, the visible portion of the light source does not exceed about 40%. The schema on the right represents a section through the middle of the scene. The points a are at the boundary of umbra, and the points c are at the boundary of the penumbra. The points b are on the maximal free line segments corresponding to an arc FE involving a face of the blocker. From a to b , the percentage of the light source that is visible increases linearly from 0% to about 40%, and from b to c , it increases linearly from about 40% to 100%. Since the light grey region can be made arbitrarily large by moving the light source closer to the blocker, the trace of the FE arcs on the floor corresponds to a discontinuity of the derivative in the percentage of visible area of the light source.

plexity of the primary vertices of interest to us, $EEEE$ vertices excluded, is $\Theta(n^2k)$, and the worst-case size complexity of the secondary vertices is $\Theta(n^2)$. Thus, in the worst case, $O(p \log p + m \log m)$ is equivalent to $O(n^2k \log n)$.

There exist various algorithms for computing the secondary vertices. For instance one can use the sweep algorithm described in Chapter 2 (also in [14, 50]), or one can also compute, in a brute force way, the possibly occluded candidate secondary vertices and perform ray shooting to check for occlusion [1, 29, 79]. To our knowledge, the best worst-case running time is $O(n^2k \log n)$, obtained by computing $O(n^2)$ candidate

secondary vertices in a brute force way and checking for occlusion using the Dobkin-Kirkpatrick hierarchical representation [29, 75], which leads to performing $O(n^2)$ ray shooting queries on each of the k polytopes in $O(\log n)$ time each. Comparatively, the method we propose has the same complexity in the worst case. However, our method is output-dependent, and thus, it can be much more efficient than previously existing algorithms. In addition, our method takes as input the primary vertices, whose observed size is, in a random setting, $Ck\sqrt{nk}$ for a small constant C (see Chapter 6), which is much smaller than the worst-case size, that is $\Theta(n^2k)$.

The rest of this chapter is organized as follows. We provide necessary definitions in the next section. We then introduce the computational relations among the types of visibility skeleton vertices in Section 7.2. We prove that we can recover the full skeleton from the succinct one in $O(p \log p + m \log m)$ time in Section 7.3. By a series of examples, we show in Section 7.4 that none of the primary vertex types can be omitted while maintaining the validity of this result. We discuss our results in Section 7.5.

We note that the study in this chapter heavily depends on the definition of the 3D visibility skeleton that we gave in Section 2.2.2.

7.1 Preliminaries

We first define precisely the *primary vertices* as the skeleton vertices of types $EEEE$, VEE , FEE , together with the vertices of type VV that lie in a plane tangent to both polytopes; and the *secondary vertices* as the remaining vertices of type VV , and the vertices of types FF , FvE , FE , and FVV .

Next, we define the concept of *constraint*. Recall that for any skeleton vertex, a support vertex is a polytope vertex that lies on the relative interior of the free line segment, and a support edge is a polytope edge that intersects the free line segment in its relative interior. For any skeleton vertex or arc, we define its *constraint edges* as the edges incident to a support vertex, such that a plane containing the free line segment and one of these incident edges is tangent to the support polytope. The *constraints* of a skeleton vertex or arc are defined as its support edges and constraint edges. It is easy to see that any skeleton vertex has, in general position, four constraints, and any skeleton arc has three.

We finally define a *master arc* as a set of arcs that share the same set of supports.¹ On a master arc, the arcs and their incident vertices are sorted according to the position of their intersections with a chosen support edge of the master arc, in some chosen direction. We note that the master arc is only a facilitating data structure, which allows quick access to its associated arcs.

7.2 Computational Relations among the Visibility Skeleton Vertices

Recall from Section 2.2.2 that a 3D visibility skeleton vertex corresponds to a maximal free line segment that has 0-degrees of freedom, and the degrees of freedom are restricted by its supports. Alternatively, the degrees of freedom are restricted by its constraints.

¹The master arc concept was initially presented in [94].

Next, we specify that the knowledge of a skeleton vertex includes its corresponding maximal free line segment and its supports and constraint edges, that is, any support polytope edges, any support polytope vertices with incident constraint edges, and all support polytopes. Computing a skeleton vertex includes computing the maximal free line segment and all these supports and constraint edges.

Note that for any free line segment that corresponds to a skeleton vertex, we determine the four polytope edges to which it is tangent (the constraints). Since any incident skeleton arc is tangent to three of those, we can determine all incident skeleton arcs by relaxing one degree of freedom, removing in turn each edge from the list of edges to which the free line segment is tangent. The computation time is constant.

Given a skeleton arc, together with the knowledge of its constraints and support polytopes, any incident skeleton vertex can be computed from the skeleton arc, provided that the skeleton vertex has the same set of support polytopes as the skeleton arc. The computation involves enumerating incident edges of a polytope vertex to find constraint edges, or edges on a polytope face to find the support edges of the incident skeleton vertex. This requires $O(\delta)$ time computation, where δ is the maximum degree of a polytope vertex or number of edges on a polytope face. For example, given an EEE arc, one can compute the vertices of types VEE and FEE that are incident to the EEE arc in $O(\delta)$ time. But, if an EEE arc is incident to an $EEEE$ vertex, then this $EEEE$ vertex can not be computed directly from the EEE arc, since the $EEEE$ vertex has an additional, unknown support polytope.

The possible computational relations among skeleton vertex types are summarized

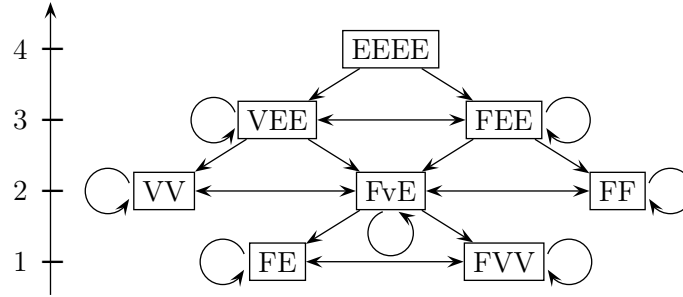


Figure 7.2. The possible computational relations among the types of 3D visibility skeleton vertices.

in the diagram in Figure 7.2. The edges in this diagram give all possible pairs of vertex types that can be connected by an arc of the full visibility skeleton. Furthermore, an arrow oriented from one vertex type to another indicates that the set of support polytopes of a vertex of the former type contains that of a vertex of the latter type. This means that vertices of the latter type can be computed from adjacent vertices of the former type in $O(\delta)$ time. Note that skeleton vertices of types appearing in rows 1 (bottom), ..., 4 (top) of the diagram in Figure 7.2 are supported by 1, ..., 4 polytopes, respectively.

We now prove the correctness of the computational relations illustrated in the diagram of Figure 7.2 in Lemma 15 and 16.

Lemma 15. *Let \mathcal{X} and \mathcal{Y} denote vertex types in the full skeleton graph, such that in Figure 7.2, \mathcal{X} has more support polytopes than \mathcal{Y} (so \mathcal{X} appears in a higher row than \mathcal{Y}), and there is an edge directed from \mathcal{X} to \mathcal{Y} . Then we cannot, from a skeleton vertex of type \mathcal{Y} , determine an adjacent skeleton vertex of type \mathcal{X} in time independent of the number of input polytopes.*

Proof. Since any vertex of type \mathcal{X} has more support polytopes than any vertex of

type \mathcal{V} , when computing an incident vertex from the arc that is created by a vertex of type \mathcal{V} , the additional support polytope, and furthermore the additional support edge (or support vertex) on the additional support polytope, is unknown. This requires a search for the additional polytope and the additional support edge (or support vertex) on the polytope. In general, this search cannot be done directly from a vertex of type \mathcal{V} (given only the knowledge of its supports). \square

Lemma 16. *From any skeleton vertex, it is possible to compute each adjacent skeleton vertex in $O(\delta)$ time, where δ is the maximum degree of a polytope vertex or number of edges on a polytope face, under the condition that all support polytopes of the adjacent skeleton vertex and of the connecting skeleton arc are also support polytopes of the starting skeleton vertex.*

Proof. By our definition of the 3D visibility skeleton vertex, we already know the support polytopes of the adjacent skeleton vertex.

From the starting skeleton vertex, we need to find incident skeleton arcs. Since we know the supports of the skeleton vertex and any constraint edges, we know four polytope edges tangent to the corresponding free line segment of the skeleton vertex (the constraints), and thus we can find its incident skeleton arcs in constant time by removing in turn each of these constraints.

We now consider how to deal with each type of arc incident to a given vertex. From an *EEE* arc, it is possible to find unknown incident skeleton vertices of type *VEE* or *FEE* by checking all polytope faces and vertices incident to the polytope edges, and computing the corresponding candidates. Since their number is constant, we can find the incident vertices in constant time by enumerating them. Similarly,

from a VE arc, we find unknown incident VV vertices by checking polytope vertices incident to the polytope edge. From an FE arc, we find unknown FF vertices by checking polytope faces incident to the polytope edge.

Moreover, from a VE arc, we find unknown incident FvE vertices by checking the four polytope faces that are incident to the two constraint edges incident to the polytope vertex. From an FE arc, we find unknown FvE vertices by checking the four polytope vertices that are incident to the two polytope edges lying on the polytope face.

Thus we can find the free line segment that corresponds to an adjacent skeleton vertex in constant time, along with any of its support vertices or faces.

According to our definition, we also need to compute the edges incident to any support vertex or support face that defines an incident skeleton arc. For a VEE or VV vertex, these are constraint edges, which can be computed by checking through all the polytope edges incident to the support polytope vertices. This can be done in $O(\delta)$ time. For an FEE or FF vertex, these are support edges, which can be computed by checking all the polytope edges on the support polytope faces. Again, this can be done in $O(\delta)$ time.

Similarly, one can prove the computational relations for skeleton vertices of type FE and FVV . □

In summary, for any directed edge on Figure 7.2 from type \mathcal{X} to type \mathcal{Y} , Lemma 16 shows that it is possible to compute a vertex of type \mathcal{Y} from an adjacent vertex of type \mathcal{X} ; Lemma 15 shows that it is impossible to compute a vertex of type \mathcal{X} from a vertex of type \mathcal{Y} efficiently, that is, in time independent of the number of polytopes.

7.3 Recovery of the Full Skeleton

Recall that the five types of secondary vertices are FF , FvE , FVV , FE , and a subset of VV . The vertices of type FVV and FE are easy to find on their own, and will be computed separately. In this section, we mainly show how to compute vertices of type FvE , FF , and the subset of VV that belongs to the secondary vertices. For this, we explore the subgraph of the visibility skeleton consisting of VE and FE arcs and their incident vertices, that is, the VEE , FEE , VV , FvE and FF vertices. We call this subgraph the *partial graph*.

We first prove that all connected components of the partial graph contain at least a primary vertex of type VV , VEE or FEE . This allows us to find all FvE , FF and the remaining VV vertices by simple graph exploration, examining vertices adjacent to those we have already computed.

To prove that all connected components contain a primary vertex of type VV , VEE or FEE , we divide the partial graph into the subgraphs of vertices and arcs that are supported by every pair of polytopes, and prove that they all contain such a vertex. The proof uses an optimization concept, that is, we define an objective function, and prove the local minimum of each connected component is a primary vertex of type VV , VEE , or FEE .

Rather than defining a single objective function for the whole partial graph, we define a collection of objective functions, one for each pair of polytopes that support at least one skeleton vertex. Each objective function will be used for the enumeration of the vertices and arcs supported by the corresponding pair of polytopes. In other words, the vertices and arcs supported by each pair of polytopes form subgraphs of

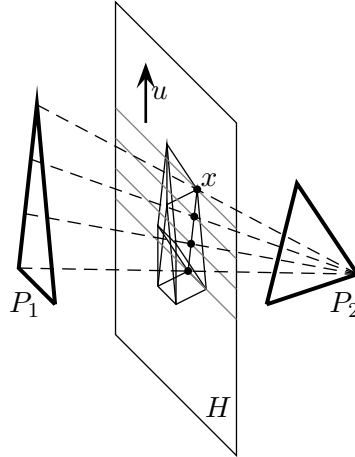


Figure 7.3. The value of each of the free line segments is defined by a linear function on its intersection with the plane H .

the partial graph, which are enumerated separately, starting from the primary vertices of type VEE , FEE , and VV .

For each pair of polytopes P_1 and P_2 that support at least one vertex, we define the following objective function. Let H be a plane separating P_1 and P_2 ,² and let u be a vector that is in generic direction. For any maximal free line segment l tangent to P_1 and P_2 that represents a skeleton vertex or a point of a skeleton arc, we define the *value* of l as $f(l) = u \cdot x$, where x is a vector representing the intersection of H with the supporting line of l .³ Note that l always intersects H , being tangent to P_1 and P_2 , which are separated by H (See Figure 7.3).

Lemma 17. *Any vertex v of the visibility skeleton of type FF or FvE that is supported by P_1 and P_2 has an adjacent vertex v' , connected by a VE or FE arc, such that the*

²This separating plane can be computed when computing the primary vertices using the sweep algorithm. Specifically, it corresponds to the V-event that places the two polytopes, one supporting the sweep plane and the other supporting the V-event, on different sides of the sweep plane.

³We can define the generic direction of u as $u = (1, \varepsilon, \varepsilon^2)$, so that when ε goes to 0, $f(l_1) \neq f(l_2)$, for any free line segments l_1 and l_2 representing adjacent skeleton vertices.

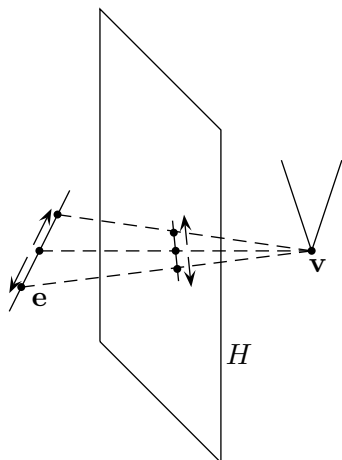


Figure 7.4. The intersections with H of free line segments on the two VE arcs on each side of a degenerate FvE vertex move in opposite directions.

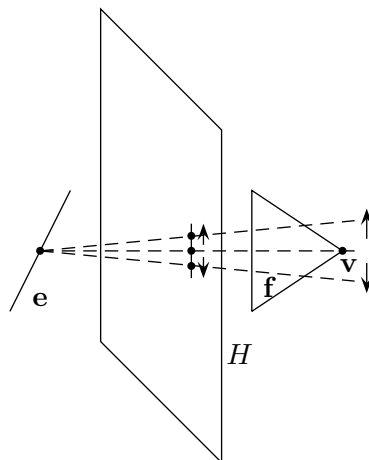


Figure 7.5. The intersections with H of free line segments on the two FE arcs on each side of a non-degenerate FvE vertex move in opposite directions.

value of f along the arc continuously decreases from v to v' .

Proof. If free line segments are tangent to a common vertex and edge, their intersection with plane H is on a straight line. If we parametrize these free line segments by $l(t)$, $t \in [0, 1]$, such that $l(0)$ and $l(1)$ correspond to the free line segments tangent to the extremities of the edge, $f(l(t))$ is an affine function, which is not constant since u is in generic position, so $f(l(t))$ has a minimum at 0 or 1.

In particular, the set of free line segments representing a VE or FE arc of the visibility skeleton share a 3D point and are tangent to a common polytope edge e , and so the value of the free line segments increases or decreases monotonically between two skeleton vertices connected by arcs of these types.

Recall that degenerate FvE vertices are tangent to a polytope edge and a vertex of a different polytope, and are in the supporting plane of a face incident to the vertex,

without intersecting that face except on the vertex (Figure 7.4). A degenerate FvE vertex always has two incident VE arcs corresponding to the same polytope vertex and polytope edge, in opposite directions. Therefore, the free line segment corresponding to a degenerate FvE vertex is in the middle of a set of free line segments defined by a vertex and an edge, which corresponds to two incident skeleton arcs. Since it is in the middle, its value is not minimum, so the value is decreasing along one of the arcs.

Non-degenerate FvE vertices are tangent to a polytope face and a polytope vertex of that face, and to an edge of a different polytope (Figure 7.5). If a non-degenerate FvE vertex is defined by an edge \mathbf{e} of polytope P , a vertex \mathbf{v} and a face \mathbf{f} of polytope P' , then it has two incident FE arcs corresponding to the same intersection point on \mathbf{e} and polytope edge (incident to \mathbf{f} , but not \mathbf{v}), in opposite directions. Again, the corresponding free line segment is in the middle of a set of free line segments defined by a vertex and an edge, which corresponds to two incident skeleton arcs, and the value is decreasing along one of the arcs.

Vertices of type FF have four incident FE arcs. We define as p and p' the intersection points of the free line segment corresponding to an FF vertex with the two support edges on one of the two polytopes. Then the two FE arcs, obtained by rotating the maximal free line segment corresponding to the FF vertex around p and p' , will move in opposite directions on a line in H . Therefore, one of them is decreasing. □

Lemma 18. *Let v be a vertex of the visibility skeleton of type VV . If all adjacent vertices of v have a higher value than v , then v lies in a plane tangent to both polytopes, i.e., v is a primary vertex.*

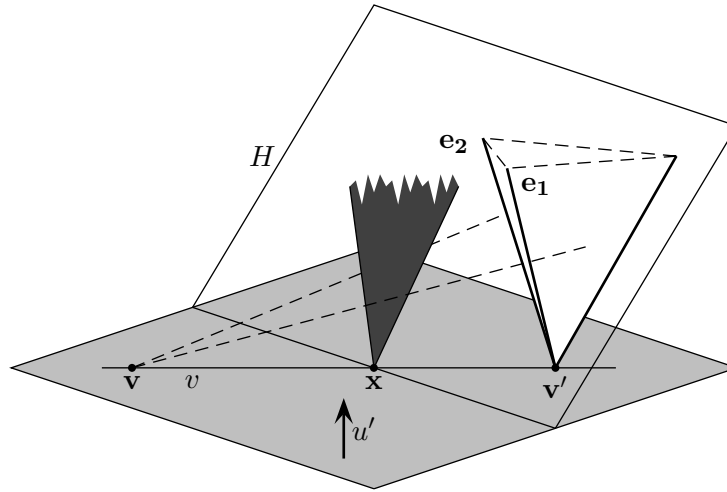


Figure 7.6. The silhouette of the polytope from \mathbf{v} projected on H is inside the cone of the projected constraint edges. If the constraint edges are in the half-plane $u' \cdot x \geq 0$, so is the polytope.

Proof. Let v be a VV vertex supported by the polytope vertices \mathbf{v} and \mathbf{v}' in the polytopes P and P' , respectively, such that adjacent skeleton vertices have a higher value. Let \mathbf{x} be the intersection of v with H . Without loss of generality, we assume \mathbf{x} to be the origin. Since H contains the origin, its equation is $H = \{x : a \cdot x = 0\}$ for some vector a . We modify u into u' , such that u' is perpendicular to v , but the value function does not change. This can be done by setting

$$u' = u - \frac{u \cdot d}{a \cdot d} a,$$

where d is a vector from \mathbf{v} to \mathbf{v}' . It is easy to check that $u' \cdot d = 0$ and $u \cdot x = u' \cdot x$ for any x in H .

Vertices of type VV have four incident VE arcs, two supported by each polytope vertex. Let \mathbf{e}_1 and \mathbf{e}_2 be the two constraint edges of v incident to \mathbf{v}' ; then \mathbf{v} and \mathbf{e}_1 generate a VE arc incident to v , as well as \mathbf{v} and \mathbf{e}_2 . Viewed from \mathbf{v} , the silhouette of P projected on H is contained in the cone between \mathbf{e}_1 and \mathbf{e}_2 (see Figure 7.6). By

assumption, the value of free line segments is increasing along these arcs, so P is in the half-space $u' \cdot x \geq 0$.

Similarly, P' is in the half-space $u' \cdot x \geq 0$, and so v lies in the plane $u' \cdot x = 0$ which is tangent to P and P' . \square

From Lemma 17 and 18, we obtain Corollary 19.

Corollary 19. *Let P_1 and P_2 be a pair of polytopes, with the related objective function f defined on the subgraph of the partial graph of the visibility skeleton containing vertices and arcs whose corresponding maximal free line segments are tangent to P_1 and P_2 , and possibly other polytopes. Any local minimum of the objective function is a primary vertex of type VV , VEE , or FEE .*

Since any non-empty subgraph related to two polytopes is finite, it has a minimum which has to be a primary vertex.

We now show how to explore the partial graph, in order to compute the secondary vertices of type VV , FvE and FF from primary vertices of type VEE , FEE and VV . To explore the partial graph efficiently, we compute the secondary vertices of type VV , FvE or FF along a sequence of VE or FE arcs. In what follows, we will show that this can be done in time $O(p' \log p + s')$, where p' and s' are the number of primary and respectively secondary vertices in the sequence, and p is the total number of primary vertices in the succinct visibility skeleton.

We will examine separately VE arcs and FE arcs. First we define a *sequence of VE arcs* as a maximal set of connected VE arcs that share the same support polytope edge and vertex.

Lemma 20. *Any sequence of VE arcs has a VV vertex, a VEE vertex, or a non-degenerate FvE vertex at each extremity, and arcs in the sequence are separated by degenerate FvE vertices or VEE vertices.*

Proof. Any non-degenerate FvE vertex has a single incident VE arc, and any VV vertex has four of them, but none are supported by the same polytope vertex and edge. They are therefore extremities of the sequence of VE arcs. Any degenerate FvE vertex has two incident VE arcs supported by the same polytope vertex and edge, so it is in the middle of a sequence. A VEE vertex has three or four incident VE arcs, which are supported by two different polytope edges. When two arcs are supported by the same edge, the vertex is in the middle of the sequence; when only one arc is supported by an edge, the vertex is an extremity of the sequence. \square

We define a *sequence of FE arcs* as a maximal set of connected FE arcs that share the same support polytope face, and are tangent to the same other polytope. We note that, according to our definition, a sequence of FE arcs is not necessarily always supported by the same polytope edge.

Lemma 21. *Any sequence of FE arcs has a degenerate FvE vertex or FEE vertex at each extremity, and arcs in the sequence are separated by non-degenerate FvE vertices, FEE vertices or FF vertices.*

Proof. Any degenerate FvE vertex has a single incident FE arc. It is therefore an extremity of the sequence of FE arcs. Any non-degenerate FvE vertex has two incident FE arcs supported by the same polytope face and polytope edge. It is in the middle of a sequence. And any FF vertex has four incident FE arcs, each polytope face

supporting two of them, which are tangent to different edges on the other polytope. They are therefore in the middle of a sequence. An *FEE* vertex has three or four incident *FE* arcs, which are supported by two different polytope edges. When two arcs are supported by the same edge, the vertex is in the middle of the sequence; when only one arc is supported by an edge, the vertex is an extremity of the sequence. \square

Lemma 22. *Any sequence of VE arcs can be computed in $O(p' \log p + s')$ time if we know its primary vertices, or, when it contains no primary vertices, if we know one secondary vertex; here p' and s' are the number of primary and respectively, secondary vertices in the sequence, and p is the total number of primary vertices in the succinct visibility skeleton.*

Proof. First, we find the primary vertices in the sequence from the total list of primary vertices in the succinct visibility skeleton. If the list is sorted by supports, this can be done in $O(p' \log p)$. We then show that secondary vertices can be computed in linear time in their number.

All skeleton vertices in a sequence of *VE* arcs are supported by a vertex \mathbf{v} on polytope P and an edge \mathbf{e} . Suppose \mathbf{x} is a point moving on \mathbf{e} , and let us consider the free line segment l containing \mathbf{x} and \mathbf{v} . The constraints of l are \mathbf{e} and two constraint edges incident to \mathbf{v} , denoted as \mathbf{e}_1 and \mathbf{e}_2 . Let p_1, p_2 be the two planes containing \mathbf{x} and $\mathbf{e}_1, \mathbf{e}_2$ respectively. Then plane p_1 (respectively p_2) is tangent to P and contains \mathbf{x}, \mathbf{v} and \mathbf{e}_1 (respectively \mathbf{e}_2). As \mathbf{x} moves along \mathbf{e} , planes p_1 and p_2 roll around the faces and edges incident to \mathbf{v} . Let C be the polyhedral cone created by faces incident to \mathbf{v} , and let C' be the centrally symmetric cone also having its apex at \mathbf{v} . If the supporting line of \mathbf{e} does not intersect C or C' , the two planes roll in the

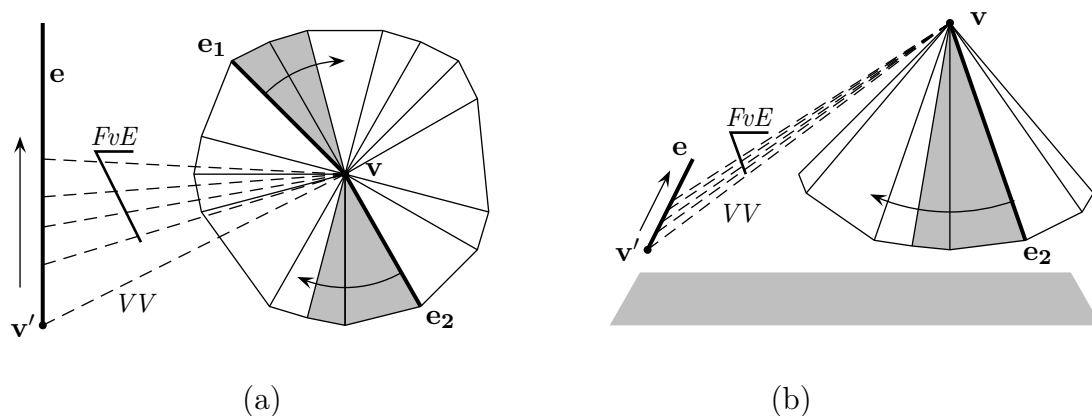


Figure 7.7. (a) Bird's eye view and (b) 3D view of degenerate FvE vertices whose supports are polytope edge e and two sequences of faces incident to v , starting from e_1 and e_2 , which create VE arcs with v' .

same direction around v (Figure 7.7). Otherwise, they roll in opposite directions (Figures 7.8 and 7.9).

Suppose we know one of the extremities of the sequence of VE arcs, which can be either VV , VEE or non-degenerate FvE vertices. We examine each of the cases in turn as follows.

Case i): *The extremity is of type VV (supported by v and v'):* Then the planes tangent to P contain its constraint edges, e_1 and e_2 ; the supporting planes of the polytope faces incident to e_1 or e_2 intersect the supporting line of the polytope edge e on either side of v' (see Figures 7.7 and 7.8). From those two edges (e_1 and e_2), circling around the polytope vertex v , we enumerate the two sequences of faces incident to v . Their supporting planes intersect the polytope edge e , and each of the intersections indicates a degenerate FvE vertex.

If there is a VEE vertex in the sequence of arcs, then we already know the vertex, as well as the point where the free line segment corresponding to the VEE vertex in-

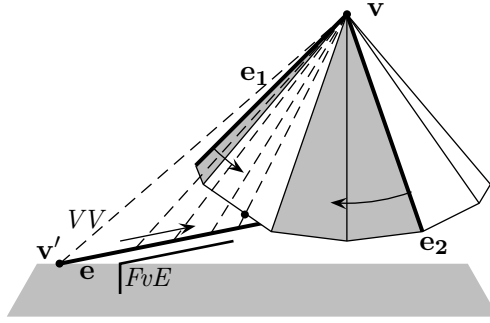


Figure 7.8. When the polytope edge e intersects with the polyhedral cone of the faces incident to v , the two sequences of faces that are supports of degenerate FvE vertices turn in opposite directions until they meet, which indicates a non-degenerate FvE vertex.

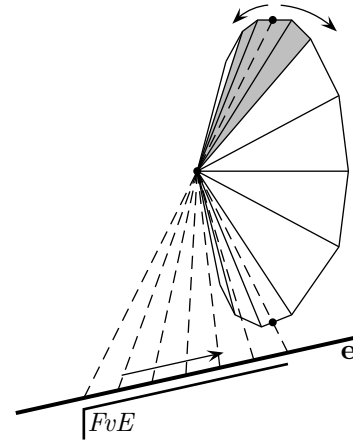


Figure 7.9. In some configurations, a sequence of VE arcs may contain degenerate FvE vertices only, with a non-degenerate FvE vertex at each end.

tersects the edge e . Thus we can insert it in the sequence when the intersections reach that point. The VEE vertex can end the sequence; otherwise we keep enumerating the faces incident to v .

If the sequence of arcs ends with a VV vertex, we stop when the intersections reach the other end of edge e .

In case the two sequences of faces are turning in opposite directions, and they turn until they meet (see Figure 7.8), this indicates a non-degenerate FvE vertex at the end of the sequence of arcs.

Case ii): *The extremity of the sequence of VE arcs formed by vertex v and edge e is a VEE vertex (supported by v , e and e'):* then let e_1 and e_2 denote the constraint edges of the VEE vertex, incident to v , and we continue as in the case that the

extremity is of type VV .

Case iii): *The extremity of the sequence of VE arcs formed by vertex \mathbf{v} and edge \mathbf{e} is a non-degenerate FvE vertex (supported by \mathbf{f} , \mathbf{v} and \mathbf{e}):* Then we enumerate from \mathbf{f} the two sequences of faces incident to \mathbf{v} and proceed as above.

In case we do not know an extremity of the sequence of arcs, but we do know a vertex in the middle, we can explore the sequence in each direction using the same method, since we know the constraints of the vertex. If it is of type VEE , then let \mathbf{e}_1 and \mathbf{e}_2 denote its constraint edges incident to \mathbf{v} , and continue as above. If it is of type degenerate FvE (supported by \mathbf{f} , \mathbf{v} and \mathbf{e}), then let \mathbf{e}_1 , \mathbf{e}'_1 and \mathbf{e}_2 be its three constraint edges incident to \mathbf{v} , with \mathbf{e}_1 and \mathbf{e}'_1 incident to \mathbf{f} . We then enumerate faces incident to \mathbf{v} in one direction starting with \mathbf{e}_1 and \mathbf{e}_2 , and in the other starting with \mathbf{e}'_1 and \mathbf{e}_2 .

Note that if the sequence does not have a VV or VEE vertex at any extremity, then the supporting planes of *all* faces incident to the vertex \mathbf{v} intersect with edge \mathbf{e} (see Figure 7.9). All of these intersections correspond to a FvE vertex. \square

Lemma 23. *Any sequence of FE arcs can be computed in $O(p' \log p + s')$ if we know the primary vertices, or, when the sequence contains no primary vertex, if we know one secondary vertex; here p' and s' are the number of primary and, respectively, secondary vertices in the sequence, and p is the total number of primary vertices in the succinct visibility skeleton.*

Proof. In Lemma 22, first we find the primary vertices in the sequence from the total list of primary vertices in the succinct visibility skeleton in $O(p' \log p)$ time. We then show that secondary vertices can be computed in linear time in their number.

If the extremity of the sequence of FE arcs formed by face \mathbf{f} and polytope P is an FEE vertex (supported by face \mathbf{f} , edge \mathbf{e} on P and \mathbf{e}' on some other polytope), then let \mathbf{e}_f and \mathbf{e}'_f denote its two support edges on face \mathbf{f} . From \mathbf{e}_f or \mathbf{e}'_f , circling around the face \mathbf{f} , the two sequence of polytope vertices (on face \mathbf{f}) will create non-degenerate FvE vertices with \mathbf{e} .

If there is an FEE in the sequence of arcs, then we know the vertex, as well as its support edges on \mathbf{f} . When the sequences of vertices reach the FEE vertex, we insert it in the sequence of arcs. The FEE vertex can end the sequence; otherwise we keep enumerating the vertices of \mathbf{f} .

If the sequence of arcs contains an FF vertex, it corresponds to face \mathbf{f} and a face \mathbf{f}' that is incident to edge \mathbf{e} . These are inserted in the sequence of arcs when the sequences of vertices cross the supporting plane of \mathbf{f}' .

In case we do not know an extremity of the sequence of arcs, then they are the type of degenerate FvE . In this case, all vertices of face \mathbf{f} create a degenerate or non-degenerate FvE vertex with the other polytope. So starting from any skeleton vertex we know, we can enumerate polytope vertices on face \mathbf{f} , adding vertices of type FF and FEE along the way as above. \square

We remark that secondary vertices in sequences of arcs are computed in time linear in their number. However, for each extremity of a sequence that is a secondary vertex, we need to enumerate the edges incident to a polytope vertex or to a face in order to compute the constraints of the skeleton vertex, and this is done in $O(\delta)$ time.

The method for finding the secondary vertices is to explore the partial graph with a simple search. That is, we first examine each primary vertex of type VV , VEE or

FEE , and find all secondary vertices of type VV , FvE and FF on adjacent sequences of arcs. We keep a list of discovered secondary vertices, and check before adding any new one whether it is already there. We then examine each vertex in that list, looking again for secondary vertices on adjacent sequences of arcs, which are added to the end of the list on the condition that they are not yet there. In this sense, we are treating the list like a queue. To search the list more efficiently, we can order it (lexicographically for example), and keep track of the queuing order by adding to each vertex a pointer to the next one to be examined. Checking whether a vertex is already in the list can then be done in logarithmic time.

Since any vertex is adjacent to a constant number of arcs, the whole search is done in $O(p \log p + m' \log m')$ time, where p is the number of primary vertices of type VV , VEE and FEE , and m' is the number of secondary vertices of type VV , FvE and FF . That is, each of the p primary vertices is found in $O(\log p)$ time by searching through the list of primary vertices, and each of the m' vertices is found in constant time and added to the list of secondary vertices in $O(\log m')$ time.

Note that for each sequence that ends with a secondary vertex, computing the constraints of this vertex is done in $O(\delta)$ time, but this is bounded by $O(m'\delta)$, which is negligible in comparison to $O(m' \log m')$.

Remark 24. *The FE vertices correspond to edges of polytopes and can be computed by simple enumeration. Furthermore, FVV vertices correspond to diagonals of faces of polytopes, and can also be found by simple enumeration.*

Theorem 25. *Given the succinct visibility skeleton, one can compute the full visibility skeleton from the succinct one in $O(p \log p + m \log m)$ time, where p is the number*

of primary vertices of type VV , VEE and FEE , and m is the number of secondary vertices of type VV , FvE , FF , FE , and FVV .

Proof. We start with the knowledge of all the primary vertices, and find the secondary vertices. In order to do that, we explore the partial subgraph of the visibility skeleton containing all VE and FE skeleton arcs and the skeleton vertices at their extremities. We find in this way all vertices of type VV , FvE and FF .

Corollary 19 shows that we know at least one vertex in each connected component of this partial subgraph. Lemmas 22 and 23 show that from any vertex in a sequence of VE or FE arcs, we can find all the secondary vertices in the sequence in time linear in their number. As any unknown vertex is connected to a known vertex through a series of sequences of arcs, we can find all vertices.

We have seen that using our special exploration procedure, a graph of p known vertices and m' unknown vertices can be explored in $O(p \log p + m' \log m')$ time. Vertices of type FE and FVV are computed separately in linear time in their number m'' . Thus the complete enumeration is done in $O(p \log p + m' \log m' + m'')$ time, where $m' + m'' = m$. Since $O(p + m' \log m' + m'') \in O(p \log p + m \log m)$, the theorem follows. \square

We finally note that the graph exploration method we explained above can be applied to only part of the input polytopes. In this case, we can first find all the primary vertices that are related to the polytopes of interest, and apply the graph exploration on only these primary vertices.

7.4 Tightness of the Succinct Skeleton

In this section, we show, mostly by examples, that Theorem 25 is tight in the sense that if any type of primary vertices, $EEEE$, VEE , FEE , or VV , is regarded instead as a type of secondary vertices, and thus excluded from the succinct skeleton, then the statement of the theorem no longer holds.

Type $EEEE$. Any vertex of type $EEEE$ requires four support polytopes, and there are no skeleton arcs that have four support polytopes, by assumption. Hence, by Lemma 15, vertices of type $EEEE$ must be regarded as primary.

Types VEE and FEE . When three input polytopes are not the support polytopes of any $EEEE$ vertex, then the vertices of types VEE and FEE that have supports on the three polytopes cannot be computed from any $EEEE$ vertex.

Moreover, some scenes may generate vertices of type VEE but no vertices of type FEE , as shown in Figure 7.10 (a). Hence, by Lemma 15, vertices of type VEE cannot be dropped.

Furthermore, some scenes may generate vertices of type FEE but no vertices of type VEE , as shown in Figure 7.10 (b) and explained below.

The scene in Figure 7.10 (b) consists of a prism that approximates a cylinder, positioned between two truncated pyramids that approximate truncated cones, where the full cones would barely intersect.

A supporting plane of a face of the prism intersects the two truncated pyramids in two polygonal arcs that approximate hyperbolas (Figure 7.10 (c)). These two polygonal arcs admit two bitangents that lie in the supporting plane of the face of

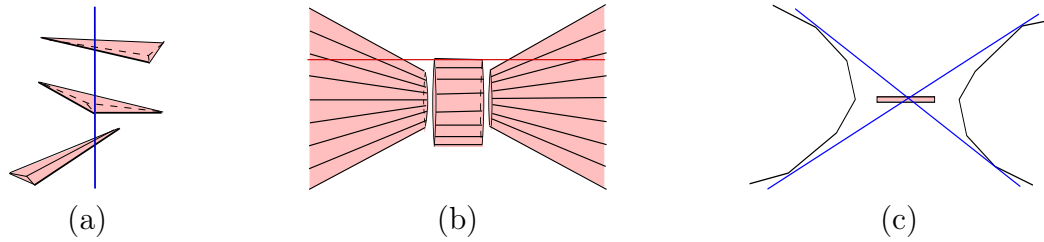


Figure 7.10. (a) Three polytopes admit vertices of type VEE but not vertices of FEE . (b) Three polytopes admit vertices of type FEE but not vertices of type VEE . (c) A cross section of (b) as indicated by the red line segment.

the prism and that cross the face, generating two FEE vertices.

As the supporting plane rolls around the prism, either one of the two bitangents will cross a pyramid face, or both bitangents will cross a face of the prism. Thus for this scene, each arc of type EEE is incident to two FEE vertices. Since there are no vertices of type VEE , the vertices of type FEE cannot be computed from vertices of type VEE . Hence, by Lemma 15, vertices of type FEE cannot be dropped.

Type VV . When two input polytopes are not the support polytopes of any $EEEE$, VEE or FEE vertex, then the vertices of type VV that have supports on the two polytopes cannot be computed from any $EEEE$, VEE or FEE vertex. Moreover, when two polytopes resemble two flat tetrahedra and face each other, then the only primary vertices they admit are of type VV . Therefore, the primary vertices of type VV must be computed.

7.5 Discussion

We have presented a method to recover the full visibility skeleton, either partial or complete, from a succinct one. The full visibility skeleton is the 0D and 1D cells of the 3D visibility complex [34, 36], whereas the succinct one is defined by visual event surfaces, and is a subset of the full one [25, 26]. Recovering the full skeleton mainly consists of computing the secondary vertices of type FvE , FF , and VV (whose supports do not lie on a plane that is tangent to both support polytopes), from the primary vertices of type VEE , FEE , and VV (whose supports lie on a plane that is tangent to both support polytopes).

The running time of our method, in the worst case, is $O(p \log p + m \log m)$, where p is the number of primary vertices, and m is the number of secondary vertices. When considering the worst-case size of p and m , which is $O(n^2k)$ and $O(n^2)$ respectively (for input consisting of k polytopes with n edges in total), $O(p \log p + m \log m)$ is equivalent to $O(n^2k \log n)$. This has the same worst-case complexity as the best running time of the previous existing algorithms, that is, using the Dobkin-Kirkpatrick Hierarchical representation to test whether the line segments corresponding to secondary vertices are free [29]. However, our method can be much more efficient when considering the expected size of primary vertices, as well as having running time that is output-dependant.

We finally note that the study of this chapter is at present limited to input consisting of disjoint convex polytopes in general position. Generalizing it to handle other input types (e.g. polytopes in degenerate position) would be an interesting subject for future research.

Chapter 8

Conclusion and Future Work

One of the main subjects of this thesis has been the study of the size of the visibility skeleton. We first studied the size of the 2D skeleton. Through our experimental study, we provided a linear function that estimates the size of the 2D visibility skeleton, in a random setting, in terms of the number of input discs and the scene density. This experimental result not only shows that the estimated size of the 2D visibility skeleton is much smaller than the theoretical worst-case sizes, but it is also more specific than theoretical expected sizes in the sense that we estimated the constants, y-intercept, and the linear onset in the function that estimates the skeleton size. Thus, when the size of this data structure is a concern, our experimentally determined size estimate gives more specific information than a linear function in big Oh notation.

The remainder of the thesis studied the visibility skeleton in 3D. When considering the input as a set of disjoint convex polytopes in general position, we only studied the size of the skeleton limited to the skeleton vertices that are related to the visual events surface, which is a subset of the (classically defined) full visibility skeleton. We

provided a systematic experimental study to show that the size of the 3D visibility skeleton thus defined is not too big. Specifically, in our setting, its size is quadratically related to the number of input polytopes, and linearly related to the average silhouette size of the polytopes. This estimate is higher than the expected linear complexity that we had initially hoped for, but much lower than the worst case complexity. We furthermore proved that, using this subset of the 3D visibility skeleton, we can compute the remaining vertices of the full skeleton efficiently, that is, essentially, in almost linear time in the size of the output.

A limitation of our experimental study is the model of the inputs. Recall that the input scenes consist of randomly distributed convex disjoint polytopes. Compared to real scenes, this is a very restricted and not so realistic setting. Although our experimental results can provide some indication of the size of the 3D visibility skeleton in a realistic scene model, estimating its size on real scenes remains to be done.

This thesis also provided an implementation for computing the vertices of the 3D visibility skeleton. The input of this implementation is a set of randomly distributed convex disjoint polytopes in general position. To our best knowledge, this is the first non-brute force implementation that is related to computing the 3D visibility skeleton, although the current implementation can only handle inputs that are in general position.

We used our implementation to analyze experimentally the size of the 3D visibility skeleton defined by visual event surfaces. Our implementation has also been used, by Demouth and Goaoc, to compute shadow boundaries [25, 26]. We made the implementation a free software available online [64], so as to provide assistance to those

who are interested in studying or experimenting with the 3D visibility skeleton. As we mentioned in Chapter 1, researchers in various fields, including global illumination, visibility culling, architectural acoustics, and endoscopy [23, 44, 57, 67, 70, 74, 92, 103], have stated that the 3D visibility skeleton data structure is impractical to use, though the global visibility information it encodes would be of interest. We hope our implementation, in addition to our experimental results, will encourage those researchers to reconsider this data structure.

For the purpose of providing a robust implementation, we have carefully studied the algebraic degrees of the predicates that are involved in our implementation. This study gave some insight on the nature of inputs that may cause the failure of predicates when using fixed-precision floating-point arithmetic. Also, the observed high algebraic degree of the predicates was an incentive for finding alternative procedures to compute them. This led to a research result that computes the same predicates with algebraic degree 36 instead of 168 [28].

We aimed to provide an efficient and robust implementation to compute the 3D visibility skeleton. Our implementation is efficient when the inputs consist of polytopes with a large number of edges, due to the nature of the algorithm. However, when the input consists of simple polytopes, it is not as efficient as the previous brute force implementation once it is improved by heuristics. One possible continuation of this thesis would be to apply heuristics to improve the overall performance of our implementation. In terms of robustness, our current implementation can handle inputs that are in general position, and can detect the inputs that are in degenerate position. Extending this implementation to handle degenerate inputs remains to be

done.

Finally, the research contained in this thesis suggests the following future work.

Implementation. Our implementation can be extended and improved in various directions.

- *Input.* As discussed above (see also Chapter 4), our current implementation only handles input in general position, and if the input is not in general position, our implementation recognizes it. There has been theoretical research on enumerating degenerate situations [14]. To design and write code to handle them is a difficult task that remains to be done.

Furthermore, this implementation only considers inputs that consist of convex disjoint polytopes, which rarely represent realistic scenes. Designing algorithms and providing an implementation that handles non-convex or non-disjoint polyhedra is a future research direction.

- *Algebraic degrees of the predicates.* As discussed in Chapter 5, in the current implementation, the answer to the predicate “number of transversals to four lines” is computed with a procedure of algebraic degree 27, which results in the predicate of “ordering two sweep planes” having algebraic degree 168. Providing an implementation of the predicate proposed in [28] would decrease the algebraic degree from 168 to 36. When following the exact computation paradigm (which is the setting of our implementation), those predicates will require less memory space and possibly less computation time; hence studying the performance of these lower degree predicates would be an possible future direction.

- *3D visibility skeleton.* The current implementation only computes the vertices of the 3D visibility skeleton that is defined by visual event surfaces [25, 26]. Another step would be to provide an implementation computing the full visibility skeleton graph, using the technique introduced in Chapter 7.

Applications. The main motivation of studying the 3D visibility skeleton is to apply this data structure in global illumination and shadow boundary computation. Indeed, the results of this thesis have already been used by Demouth [25, 26] for this purpose.

Furthermore, the visibility information encoded in the 3D visibility skeleton can be used for testing the visibility between objects or object features, which is of potential interest independent of global illumination and shadow computation. Examples can be found in visibility culling [103], architectural acoustics [44], and endoscopy [57]. However, the use of this data structure in any particular application may raise specific issues, for example, simplification of the data structure, algorithm design, and complexity analysis when using this data structure. Considering how little is known about using the 3D visibility skeleton, these research directions provide a rich ground for further investigation.

Bibliography

- [1] P. K. Agarwal and M. Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM Journal on Computing*, 25:100–116, 1996.
- [2] P. K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. North-Holland, 1998.
- [3] T. Akenine-Möller and U. Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In Proceedings of the *13th Eurographics Workshop on Rendering (EGRW'02)*, pages 297–306, Aire-la-Ville, Switzerland, 2002. Eurographics Association.
- [4] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.
- [5] P. Angelier and M. Pocchiola. CGAL-based implementation of visibility complexes. Technical Report ECG-TR-241207-01, Effective Computational Geometry for Curves and Surfaces (ECG), 2003.
- [6] P. Angelier and M. Pocchiola. A sum of squares theorem for visibility complexes and applications. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Discrete and Computational Geometry*, volume 25 of *Algorithms and Combinatorics*, pages 79–139. Springer-Verlag, 2003.
- [7] J. Arvo and D. Kirk. *A survey of ray tracing acceleration techniques*. Academic Press Ltd., London, UK, 1989.
- [8] Audacity: The Free, Cross-Platform Sound Editor. <http://audacity.sourceforge.net>.
- [9] D. R. Baum, H. E. Rushmeier, and J. M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. In Proceedings of the *16th annual conference on Computer graphics and interactive techniques (SIGGRAPH'89)*, pages 325–334, New York, NY, USA, 1989. ACM.

-
- [10] J. Bittner. Efficient construction of visibility maps using approximate occlusion sweep. In *Proceedings of the 18th spring conference on Computer graphics (SCCG'02)*, pages 167–175, New York, NY, USA, 2002. ACM.
- [11] J. Bittner and P. Wonka. Visibility in computer graphics. *Journal of Environmental Planning*, 30:729–756, 2003.
- [12] J.-D. Boissonnat and F. Preparata. Robust plane sweep for intersecting segments. *SIAM Journal on Computing*, 29(5):1401–1421, 2000.
- [13] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. In *Proceedings of the 14th ACM Annual Symposium on Computational Geometry (SoCG'98)*, pages 165–174, Minneapolis, Minnesota, 1998.
- [14] H. Brönnimann, O. Devillers, V. Dujmović, H. Everett, M. Glisse, X. Goaoc, S. Lazard, H.-S. Na, and S. Whitesides. Lines and free line segments tangent to arbitrary three-dimensional convex polyhedra. *SIAM Journal on Computing*, 37(2):522–551, 2007.
- [15] H. Brönnimann, H. Everett, S. Lazard, F. Sottile, and S. Whitesides. Transversals to line segments in three-dimensional space. *Discrete and Computational Geometry*, 34(3):381–390, 2005.
- [16] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA'01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 254–265, Aarhus, Denmark, 2001. Springer-Verlag.
- [17] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of CS, University of Utah, December 1974.
- [18] E. E. Catmull. Computer display of curved surfaces. In *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pages 11–17, May 1975.
- [19] CGAL: Computational Geometry Algorithms Library. <http://www.cgal.org>.
- [20] F. Cho and D. Forsyth. Interactive ray tracing with the visibility complex. *Computers and Graphics*, 23(5):703–717, 1999. Special issue on Visibility - Techniques and Applications.
- [21] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.
- [22] The CORE library. <http://cs.nyu.edu/exact/>.

-
- [23] K. Daubert, J. Kautz, H.-P. Seidel, W. Heidrich, and J.-M. Dischler. Efficient light transport using precomputed visibility. *Computer Graphics and Applications, IEEE*, 23(3):28–37, 2003.
- [24] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
- [25] J. Demouth. *Événements visuels et limites d’ombres*. PhD thesis, Université Nancy 2, Nov. 2008.
- [26] J. Demouth and X. Goaoc. Computing direct shadows cast by convex polyhedra. In Proceedings of the *25th European Workshop on Computational Geometry*, March 2009.
- [27] O. Devillers, V. Dujmović, H. Everett, X. Goaoc, S. Lazard, H.-S. Na, and S. Petitjean. The expected number of 3D visibility events is linear. *SIAM Journal on Computing*, 32(6):1586–1620, 2003.
- [28] O. Devillers, M. Glisse, and S. Lazard. Predicates for line transversals to lines and line segments in three-dimensional space. In Proceedings of the *24th ACM Annual Symposium on Computational Geometry (SoCG’08)*, Maryland, USA, 2008.
- [29] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In Proceedings of the *17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 400–413. Springer, 1990.
- [30] S. E. Dorward. A survey of object space hidden surface removal. *International Journal of Computational Geometry and Applications*, 4:325–362, 1994.
- [31] M. Drumheller. Mobile robot localization using sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2):325–332, 1987.
- [32] F. Duguet. Implémentation robuste du squelette de visibilité. Master’s thesis, INRIA Sophia-Antipolis, 2001.
- [33] F. Duguet and G. Drettakis. Robust epsilon visibility. In Proceedings of the *29th annual conference on Computer graphics and interactive techniques (SIGGRAPH’02)*, pages 567–575, New York, NY, USA, 2002. ACM.
- [34] F. Durand. *Visibilité tridimensionnelle : étude analytique et applications*. PhD thesis, Université Joseph Fourier - Grenoble I, 1999.

- [35] F. Durand, G. Drettakis, and C. Puech. The 3D visibility complex: a unified data-structure for global visibility of scenes of polygons and smooth objects. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG'97)*, pages 153–158, Kingston, Canada, 1997.
- [36] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH'97)*, pages 89–100, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
- [37] F. Durand, G. Drettakis, and C. Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, 18(2):128–170, 1999.
- [38] F. Durand, G. Drettakis, and C. Puech. The 3D visibility complex. *ACM Transactions on Graphics*, 21(2):176–206, 2002.
- [39] A. Efrat, L. Guibas, O. Hall-Holt, and L. Zhang. On incremental rendering of silhouette maps of a polyhedral scene. *Computational Geometry: Theory and Applications*, 38(3):129–138, 2007.
- [40] H. Everett, S. Lazard, B. Lenhart, and L. Zhang. On the degree of standard geometric predicates for line transversals in 3D. *Computational Geometry: Theory and Applications*, 42(5):484–494, 2009.
- [41] H. Everett, S. Lazard, S. Petitjean, and L. Zhang. On the expected size of the 2D visibility complex. *International Journal of Computational Geometry and Applications*, 17(4):361–382, 2007.
- [42] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer graphics: principles and practice*. Addison-Wesley, 1995.
- [43] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques (SIGGRAPH'80)*, pages 124–133, New York, NY, USA, 1980. ACM.
- [44] T. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, M. Sondhi, J. E. West, G. Pingali, P. Min, and A. Ngan. A beam tracing method for interactive architectural acoustics. *The Journal of the Acoustical Society of America*, 115(2):739–756, 2004.
- [45] Geomview. <http://www.geomview.org>.
- [46] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20:888–910, 1991.

- [47] L. Graves. An exploration of the 3D visibility complex. Master's thesis, Polytechnic University, Brooklyn, NY., 2007.
- [48] M. Glisse. *Combinatoire des droites et segments pour la visibilité 3D*. Thèse d'université, Université Nancy 2, Oct 2007.
- [49] M. Glisse and S. Lazard. An upper bound on the average size of silhouettes. *Discrete and Computational Geometry*, 40(2):241–257, 2008.
- [50] X. Goaoc. *Structures de visibilité globales : tailles, calculs et dégénérescences*. Thèse d'université, Université Nancy 2, May 2004.
- [51] C. Goral, K. Torrance, D. Greenberg, and B. Battaile. Modelling the interaction of light between diffuse surfaces. *Computer Graphics Proceedings, Annual Conference Series*, 18(3):212–222, 1984. Proceedings of SIGGRAPH'84.
- [52] GMP: The GNU multiple precision arithmetic library. <http://gmplib.org>.
- [53] R. L. Graham, B. D. Luboachevsky, K. J. Nurmela, and P. R. J. Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181:139–154, 1998.
- [54] O. Hall-Holt. *Kinetic Visibility*. PhD thesis, Stanford University, 2002.
- [55] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, Dec. 2003.
- [56] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.
- [57] T. He, L. Hong, D. Chen, and Z. Liang. Reliable path for virtual endoscopy: ensuring complete examination of human organs. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):333–342, Oct.-Dec. 2001.
- [58] P. Heckbert. Discontinuity meshing for radiosity. In Proceedings of the *3rd Eurographics Workshop on Rendering (EGRW'92)*, pages 203–216, 1992.
- [59] S. Hert, M. Hoffmann, L. Kettner, S. Pion, and M. Seel. An adaptable and extensible geometry kernel. In Proceedings of the *5th Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 76–91, Århus, Denmark, Aug 2001. Springer.
- [60] D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea Publishing Company, 1952.
- [61] M. Hohmeyer and S. Teller. Determining the lines through four lines. *Journal of Graphics Tools*, 4(3):11–22, 1999.

- [62] S. Hornus. *Maintenance de la visibilité depuis un point mobile, et applications*. PhD thesis, Université Grenoble I – Joseph Fourier, 2006.
- [63] iMovie, Apple Computer Inc.
- [64] Implementation of the sweep algorithm: computing the vertices of the 3D visibility skeleton. <http://www.cs.mcgill.ca/~lzhang15/webpage/software/software.html>.
- [65] L. Kettner and E. Welzl. Contour edge analysis for polyhedron projections. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 379–394. Springer, 1997.
- [66] D. E. Knuth. *The Art of Computer Programming, Vol. II: Seminumerical Algorithms*. Addison-Wesley, 1973.
- [67] S. Laine, T. Aila, U. Assarsson, J. Lehtinen, and T. Akenine-Möller. Soft shadow volumes for ray tracing. *ACM Transactions on Graphics*, 24(3):1156–1165, 2005.
- [68] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, USA, 1991.
- [69] LEDA: Library of Efficient Data Types and Algorithms. <http://www.algorithmic-solutions.com/>.
- [70] J. Lehtinen, S. Laine, and T. Aila. An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum*, 25(3):303–312, 2006.
- [71] D. Lischinski, B. Smits, and D. P. Greenberg. Bounds and error estimates for radiosity. In Proceedings of the *21st annual conference on Computer graphics and interactive techniques (SIGGRAPH'94)*, pages 67–74, New York, NY, USA, 1994. ACM.
- [72] K. Mehlhorn. *Data structures and algorithms 3: multi-dimensional searching and computational geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.
- [73] M. Mignotte. Identification of algebraic numbers. *Journal of Algorithms*, 3(3):197–204, 1982.
- [74] S. Nirenstein, E. Blake, and J. Gain. Exact from-region visibility culling. In Proceedings of the *13th Eurographics Workshop on Rendering (EGRW'02)*, pages 191–202, Aire-la-Ville, Switzerland, 2002. Eurographics Association.

-
- [75] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.
- [76] R. Orti, F. Durand, S. Rivière, and C. Puech. Using the visibility complex for radiosity computation. In *Proceedings of the ACM Workshop on Applied Computational Geometry*, Philadelphia, May 1996.
- [77] R. Orti, S. Rivière, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. In *Computer Graphics Forum: Proceedings Eurographics '96*, volume 15, pages 237–248, Poitiers, 1996.
- [78] S. Parker, P. Shirley, and B. Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah,, 1998.
- [79] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.
- [80] S. Petitjean. Computing exact aspect graphs of curved objects bounded by smooth algebraic surfaces. Technical report, University of Illinois, June 1992. Master's Thesis.
- [81] S. Petitjean, J. Ponce, and D. J. Kriegman. Computing exact aspect graphs of curved objects: algebraic surfaces. *International Journal of Computer Vision*, 9(3):231–255, 1992.
- [82] H. Plantinga and C. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.
- [83] O.A. Platonova. Singularities of the mutual disposition of a surface and a line. *Russian Mathematical Surveys*, 36:248–249, 1981.
- [84] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete and Computational Geometry*, 16(4):419–453, 1996. Proceedings of the 11th ACM Annual Symposium on Computational Geometry (SoCG'95).
- [85] M. Pocchiola and G. Vegter. The visibility complex. *International Journal of Computational Geometry and Applications*, 6(3):279–308, 1996. Proceedings of the 9th ACM Annual Symposium on Computational Geometry (SoCG'93).
- [86] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, 1987.

- [87] J. Redburn. Robust computation of the non-obstructed line segments tangent to four amongst n triangles. B.A. Thesis, Williams College, Massachusetts, 2003.
- [88] J. H. Rieger. On the classification of views of piecewise-smooth objects. *Image and Vision Computing*, 5:91–97, 1987.
- [89] J. H. Rieger. The geometry of view space of opaque objects bounded by smooth surfaces. *Artificial Intelligence*, 44(1-2):1–40, July 1990.
- [90] S. Rivière. Topologically sweeping the visibility complex of polygonal scenes. In Proceedings of the *11th ACM Annual Symposium on Computational Geometry (SoCG'95)*, pages C36–C37, Vancouver, June 1995.
- [91] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In Proceedings of the *13th ACM Annual Symposium on Computational Geometry (SoCG'97)*, pages 421–423, Nice, June 1997.
- [92] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion. Conservative volumetric visibility with occluder fusion. In Proceedings of the *27th annual conference on Computer graphics and interactive techniques (SIGGRAPH'00)*, pages 229–238, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
- [93] E. R. Scheinerman. When close enough is close enough. *American Mathematical Monthly*, 107:489–499, 2000.
- [94] A. Schröder. *Globale Sichtbarkeitsalgorithmen*. PhD thesis, Philipps-Universität Marburg, June 2003.
- [95] M. Sharir. Algorithmic motion planning. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, CRC Press, 1997, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL, 1997.
- [96] K. Shoemake. Plücker coordinate tutorial. *Ray Tracing News*, 11(1), 1998.
- [97] P. Srinivasan, P. Liang, and S. Hackwood. Computational geometric methods in volumetric intersection for 3d reconstruction. *Pattern Recognition*, 23(8):843–857, 1990.
- [98] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*, 6(1):1–55, 1974.
- [99] K. A. Tarabanis, P. K. Allen, and R. Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on Robotics and Automation*, 11(1):86–104, 1995.

-
- [100] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics Proceedings, Annual Conference Series*, 23(3):315–324, 1989. Proceedings of SIGGRAPH’89.
- [101] J. H. C. Whitehead. Combinatorial homotopy I, II. *Bulletin of the American Mathematical Society*, 55:213–245 and 453–496, 1949.
- [102] T. Whitted. An improved illumination model for shaded display. *Computer Graphics Proceedings, Annual Conference Series*, 13(2):14, 1979. Proceedings of SIGGRAPH’79.
- [103] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In Proceedings of the *Eurographics Workshop on Rendering (EGRW’00)*, pages 71–82, London, UK, 2000. Springer-Verlag.
- [104] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, 1990.
- [105] C. K. Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7(1-2):3–23, 1997.
- [106] L. Zhang, H. Everett, S. Lazard, C. Weibel, and S. Whitesides. On the size of the 3D visibility skeleton: experimental results. In Proceedings of the *16th Annual European Symposium on Algorithms (ESA’08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 805–816, Karlsruhe, Germany, Sept. 2008. Springer.
- [107] L. Zhang, H. Everett, S. Lazard, and S. Whitesides. Towards an implementation of the 3D visibility skeleton. In Proceedings of the *23rd ACM Annual Symposium on Computational Geometry (SoCG’07)*, S. Korea, 2007. Video.