Efficient Computation of Discrete Voronoi Diagram and Homotopy-Preserving Simplified Medial Axis of a 3D Polyhedron

by
Avneesh Sud

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

Chapel Hill
2006

ABSTRACT

AVNEESH SUD: Efficient Computation of Discrete Voronoi Diagram and Homotopy-Preserving Simplified Medial Axis of a 3D Polyhedron. (Under the direction of Dinesh Manocha.)

The Voronoi diagram is a fundamental geometric data structure and has been well studied in computational geometry and related areas. A Voronoi diagram defined using the Euclidean distance metric is also closely related to the Blum medial axis, a well known skeletal representation. Voronoi diagrams and medial axes have been shown useful for many 3D computations and operations, including proximity queries, motion planning, mesh generation, finite element analysis, and shape analysis. However, their application to complex 3D polyhedral and deformable models has been limited. This is due to the difficulty of computing exact Voronoi diagrams in an efficient and reliable manner.

In this dissertation, we bridge this gap by presenting efficient algorithms to compute discrete Voronoi diagrams and simplified medial axes of 3D polyhedral models with geometric and topological guarantees. We apply these algorithms to complex 3D models and use them to perform interactive proximity queries, motion planning and skeletal computations. We present three new results. First, we describe an algorithm to compute 3D distance fields of geometric models by using a linear factorization of Euclidean distance vectors. This formulation maps directly to the linearly interpolating graphics rasterization hardware and enables us to compute distance fields of complex 3D models at interactive rates. We also use clamping and culling algorithms based on properties of Voronoi diagrams to accelerate this computation. We introduce surface distance maps, which are a compact distance vector field representation based on a mesh parameterization of triangulated two-manifolds, and use them to perform proximity computations.

Our second main result is an adaptive sampling algorithm to compute an approximate

Voronoi diagram that is homotopy equivalent to the exact Voronoi diagram and preserves topological features. We use this algorithm to compute a homotopy-preserving simplified medial axis of complex 3D models.

Our third result is a unified approach to perform different proximity queries among multiple deformable models using second order discrete Voronoi diagrams. We introduce a new query called N-body distance query and show that different proximity queries, including collision detection, separation distance and penetration depth can be performed based on N-body distance query. We compute the second order discrete Voronoi diagram using graphics hardware and use distance bounds to overcome the sampling errors and perform conservative computations. We have applied these queries to various deformable simulations and observed up to an order of magnitude improvement over prior algorithms.

ACKNOWLEDGMENTS

When I entered the graduate program at University of North Carolina, I was unsure of my research path. To get a PhD has been a long and educational journey and I am grateful to many people along the way. I would like to thank my advisor, Prof. Dinesh Manocha, for his excellent advice and guidance - not only on my research, but on all professional aspects during my graduate career. Thanks to Prof. Ming Lin for feedback in the proximity queries project, and to Prof. Mark Foskey for his invaluable advice on problems related to topology. I would also like to thank the rest of my dissertation committee, Prof. Martin Styner and Prof. Suresh Krishnan for their suggestions and feedback.

I want to thank other colleagues who have helped me get this far. The research projects reported in this dissertation involved the efforts of several student collaborators in UNC GAMMA group. In this regard, I would like to thank Naga Govindaraju, Miguel Otaduy, Russell Gayle, Ilknur Kabul, Liangjun Zhang, Nitin Jain, Theodore Kim and Jason Sewall. Prior to the research reported in this thesis, I also had the good fortune of collaborating on several research projects in UNC Walkthrough group, and thanks Bill Baxter, Sung-Eui Yoon and Brandon Lloyd for their help. I particular I would like to thank Naga Govindaraju for his work ethic and sharing his knowledge of graphics hardware, and two colleagues who also became very close friends - Miguel Otaduy and Russell Gayle. I have enjoyed tremendously all the research discussions we have shared, at the office, at home, at coffeee shops, during late nights in the lab or even at diners over breakfast after long nights of work.

I also thank all the faculty and staff of the Department of Computer Science at the University of North Carolina at Chapel Hill, for making Sitterson Hall such a wonderful place to work at. The technical support staff of the department has been extremely helpful and responsive. I am thankful to NVIDIA Corporation for providing both software and hardware support which have been extremely useful in or research projects, especially to Mark Harris for helping clarify issues related to GPU programming.

The research projects were funded in parts by the U. S. Army Research Office, National Science Foundation, Office of Naval Research, DARPA RDECOM, and the Intel Corporation.

Getting this far involved considerable work, and I appreciate the help and support given by my friends through difficult and fun times. I would like to thank my close buddies Lindsay Stalker, Miguel Otaduy, Russell Gayle, Adrian Ilie, Ajith Mascarenhas, Udita Patel, John Chek, Heather Hanna, Matthew Briddell, and Sasa Junuzovic. I am also grateful to Ann and Stephen Aylward for their trust and having me befried three adorable dogs - Dutch, Mojo and Ginger.

My deepest thanks to my bhaiya, and my parents for their love, support and care, and in believing in me and getting me to this stage of my life. Last, but not the least, special thanks to Swamiji and God, for always being there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The Voronoi diagram is one of the most fundamental geometric data structures. The concepts behind Voronoi diagrams have been independently developed and applied in various fields of sciences, including biology, chemistry, physics, crystallography, geography and mathematics. The mathematical formulation was provided by mathematicians Voronoi and Dirichlet over a century ago. The Voronoi diagram was introduced to computational geometry by Shamos and Hoey [SH75], and has been applied to solve several problems in geometry and computer graphics.

Given a set of geometric primitives (also called 'sites') and a distance function, the *Voronoi diagram* is a subdivision of space into cells, such that all points in a cell have the same closest site according to the given distance function [VO98]. Informally, the Voronoi diagram captures the proximity structure of the collection of sites.

The *medial axis* of a geometric object is defined as the locus of centers of maximal inscribed spheres. The medial axis is a fundamental shape operation and was first proposed by Blum [Blu67] for biological shape measurement. Given a 3D solid, the medial axis is a well defined skeletal shape representation that provides complete information about the geometry and topology.

The Voronoi diagram of an object under the Euclidean distance metric is closely related to its medial axis. For a polyhedron, the medial axis can be easily constructed from the

Voronoi diagram, and vice-versa [Cul00]. Another closely related concept is that of distance fields. Given a set of geometric primitives (also called 'sites') a *distance field* is a scalar field defined at each point by the smallest distance from the point to the set of sites. The Voronoi diagram provides an implicit representation of the distance field. An example of the distance field, Voronoi diagram and medial axis of a 2D polyhedron is given in figure 1.1.

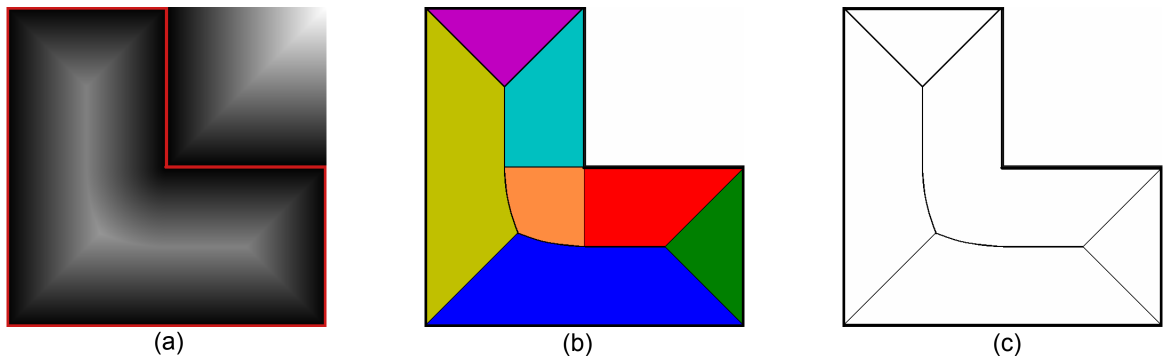


Figure 1.1: The distance field, Voronoi diagram and medial axis of a 2D polygon. *The polygon is shown in bold red. (a) The distance field in grayscale, with distance increasing from black to white (b) Voronoi diagram shown in thin black curves. Each color represents a different Voronoi region. (c) Medial axis shown in thin black curves.*

Voronoi diagrams, medial axes and distance fields have been used for a number of applications, including collision and proximity queries [LC91a, HZLM01], computer vision [PSS+03], motion planning and navigation [FGLM01, HCK+00], mesh generation and finite element analysis [SERB98, Sur03], design and interrogation [PG90, Wol92], shape analysis [BBGS99], shape simplification [TH03].

In particular, different proximity queries are required to perform collision detection, contact response and dynamic simulation. These queries are in interactive simulation systems with many applications such as surgical simulation, robotics, computer games, animation, haptics and bio-informatics. Given a general model, its Voronoi diagram provides an efficient data structure for proximity computation for any point in space. Thus the Voronoi diagram is considered as one of the most powerful data structures for proximity queries.

However, the use of Voronoi diagrams and medial axes to applications involving 3D polygonal models has been limited. This is due to the difficulty in design and implementation of reliable and efficient algorithms for computation and application of the Voronoi diagram and medial axis of 3D polygonal models.

The difficulty arises due to various reasons. Firstly, the combinatorial complexity of the Voronoi diagram is high. The upper bound on the combinatorial complexity is between $O(n^2)$ and $O(n^3 + \epsilon)$ for any positive ϵ, where n is the number of faces, edges and vertices on the polyhedron [SA95]. Secondly, the faces, edges and vertices of the Voronoi diagram of a 3D polyhedral model have algebraic degree two, four and eight, making robust computation difficult due to degeneracies and numerical errors. Finally, the medial axis exhibits instability - it is sensitive to boundary details. Small modifications in the boundary of the solid can result in large modifications of the medial axis (shown in Figure 1.7). For practical applications a stable subset of the medial axis should be computed.

1.1 Problem Definition

A geometric primitive or an object in $\mathbb{R}^d$ is called a *site*. Given a distance function, the distance between a point in $\mathbb{R}^d$ and a set of sites is the minimum distance between the point and its closest site. The *distance field* is the scalar field given by the distance from each point to the set of sites. Under the same distance function, the *first order Voronoi region* of a site k is the set of points in $\mathbb{R}^d$ closest to site k than to any other site. Then the *first order Voronoi diagram* is a partition of $\mathbb{R}^d$ into Voronoi regions of all sites.

In a standard Voronoi diagram, the sites are points. The standard Voronoi diagram is closely related to a spatial tessellation of points called the *Delaunay Tessellation*. The Delaunay tessellation is a simplicial tessellation such that the circumscribing d-sphere of each d-dimensional simplex is empty (i.e. it does not contain any points in the interior). The Voronoi diagram and Delaunay tessellation are duals of each other. Two points share a Delaunay edge iff their Voronoi regions share a $(d-1)$ dimensional face.

1.1.1 Generalized Voronoi Diagrams

The standard first order Voronoi diagram can be generalized using non-Eucidlean distance functions, higher order sites (e.g. line segments, polygons) and non-Euclidean spaces. Our goal is to compute the *generalized Voronoi diagram* of a 3D polyhedron. For a 3D polyhedron, the set of sites consists of the *boundary elements*, i.e. vertices, open edges and open faces on the boundary of the polyhedron.

The Voronoi diagram can also be generalized by computing the distances to a subset of sites. A *k-th order Voronoi region* is the set of points in $\mathbb{R}^d$ closest to a set of k sites than to any other site. The *k-th order Voronoi diagram* is the partition of $\mathbb{R}^d$ into k-th order Voronoi regions. An example of various k-th order Voronoi diagrams ($k = 1, \ldots, 5$) of a set of 5 point sites is given in figure 1.2.

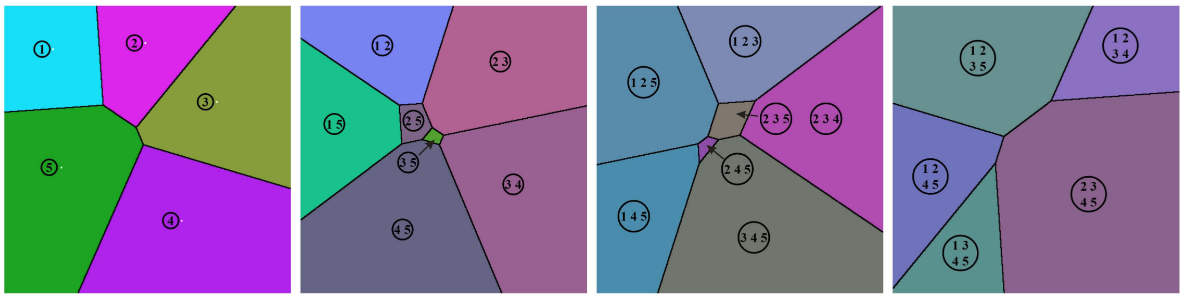


Figure 1.2: k-th order Voronoi diagrams of 5 point sites. *Left to right: First, second, third and fourth order Voronoi diagrams of* 5 *point sites. (Image courtesy [FG06]).*

As mentioned before, robust computation of exact generalized Voronoi diagram is a difficult problem[ABE04, Cul00]. Hence many approaches, sample a subset of $\mathbb{R}^d$ at a finite set of points, and compute an approximation called the *Discrete Voronoi Diagram*. The k-th order discrete Voronoi diagram (DVD) is a partition of the finite set of points into k-th order discrete Voronoi regions. The k-th order discrete Voronoi region is a finite set of points which are closest to a set of k sites than to any other site.

1.1.2 Medial Axis

For a closed polyhedron in $\mathbb{R}^d$, the *medial axis* is the locus of centers of maximal inscribed d-spheres. An inscribed sphere is *maximal* if no other inscribed sphere contains it. An alternate definition of the medial axis is the locus of points with at-least 2 closest points on the boundary of the polyhedron. The closest points on the boundary are called *footprints*. The two or more closest boundary elements (sites) are called the *governors* of a medial axis point. A generic medial axis point lies on the bisector of its two governors. In 3-D, the bisectors are quadrics. These medial axis elements are called *sheets*. The sheets meet along curves called *seams*, which in turn intersect at *junction* points. The sheets, seams and junctions for a simple polyhedron are shown in figure 1.3.

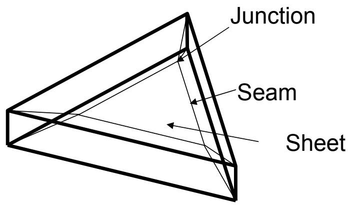


Figure 1.3: A schematic of the medial axis of a cuboid. *The sheets are in blue. Three seams are shown in orange, and the common junction is in green.*

The *medial axis transform* (MAT) is defined by associating to each axis point the radius of the maximal inscribed sphere. The original shape can be reconstructed by taking the union of all spheres. Thus the MAT is a complete shape representation, describing the shape by a lower-dimensional skeleton together with a local thickness.

For a polyhedron, the medial axis is a subset of its Voronoi diagram [ER02]. A point which lies on the Voronoi diagram but not on the medial axis lies on the bisector surface between two adjacent sites (e.g., a face and an incident edge, or an edge and incident vertex). The number of such extra surfaces is at most $O(m)$, m = number of boundary sites on the polyhedron. A 2D illustration is provided in Figure 1.4. Hence, the medial axis and

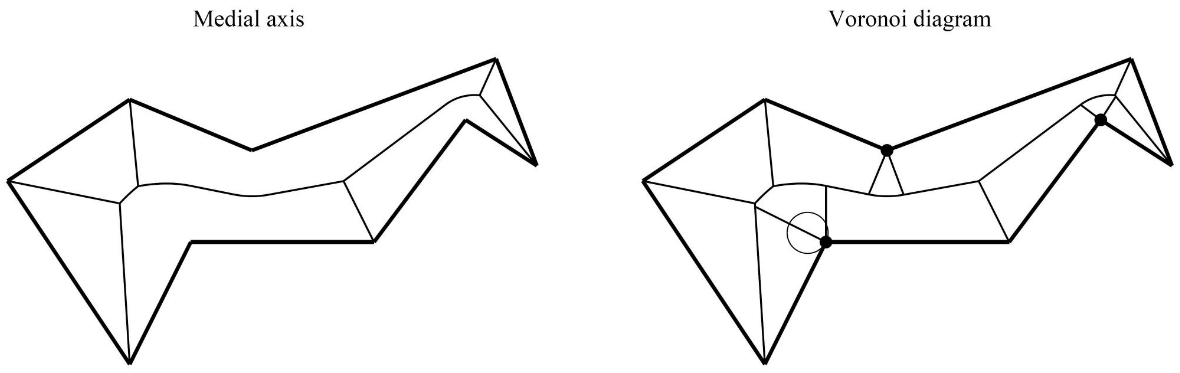


Figure 1.4: Relation between the medial axis and Voronoi diagram of a polygon. *The difference is in the Voronoi boundaries touching the* reflex vertices*. The center of the circle has* 3 *nearest sites, however the circle has only* 1 *unique point of tangency on the boundary. Hence this point belongs to the generalized Voronoi diagram, but not to the medial axis.*

Voronoi diagram can be computed from each other in linear time in the number of boudary sites [Cul00, ER02].

Homotopy equivalence: The notion of homotopy equivalence between topological sets enforces a one-to-one correspondence between connected components, holes, tunnels or cavities and also the way in which they are related. It has been shown by Lieutier [Lie03] that any bounded open subset of $\mathbb{R}^d$ is homotopy equivalent to its medial axis. Intuitively this implies that the medial axis and the shape are connected in the same way. Thus the medial axis of a polyhedron captures the topological information of the polyhedron.

Formally, two maps $f\colon \mathcal{X} \to \mathcal{Y}$ and $g\colon \mathcal{X} \to \mathcal{Y}$ are homotopic if there exists a continuous family of maps $h\colon [0,1] \times \mathcal{X} \to Y$, such that $h(0,\mathbf{x}) = f$ and $h(1,\mathbf{x}) = g$ for any $\mathbf{x} \in \mathcal{X}$. Thus, a homotopy is a deformation of one map to another. Two spaces $\mathcal{X}$ and $\mathcal{Y}$ are homotopy equivalent if there exist continuous maps $f\colon \mathcal{X} \to \mathcal{Y}$ and $g\colon \mathcal{Y} \to \mathcal{X}$ such that $g \circ f$ and $f \circ g$ are homotopic to the identity maps on their respective spaces. As an example, f could be the inclusion of a circle into an annulus, and g could be radial projection of the annulus onto the circle. In situations such as this one, where f is an inclusion and $f \circ g$ is actually equal to the identity map, the homotopy equivalence is called a deformation retraction [Spa89].

1.1.3 Proximity Queries

The set of proximity queries required for dynamic simulation includes collision detection, separation distance and penetration depth computation. These queries are performed among different objects (i.e. *inter-object queries*) or among primitives lying on the same object (i.e. *intra-object queries*).

The *Collision detection* query checks whether two objects intersect and returns all pairs of overlapping primitives (faces, edges and vertices). We consider two kinds of collision queries: *discrete* and *continuous*. The discrete collision query is performed at a specific or discrete instance of the simulation. In continuous collision detection (CCD), the motion between primitives from two successive instances of the simulation is interpolated. The CCD query computes the first time of contact between any two primitives within the time interval.

The *Separation Distance* query computes the pair of closest points (and containing primitives) between two objects. In addition, the distance between the two objects is also returned.

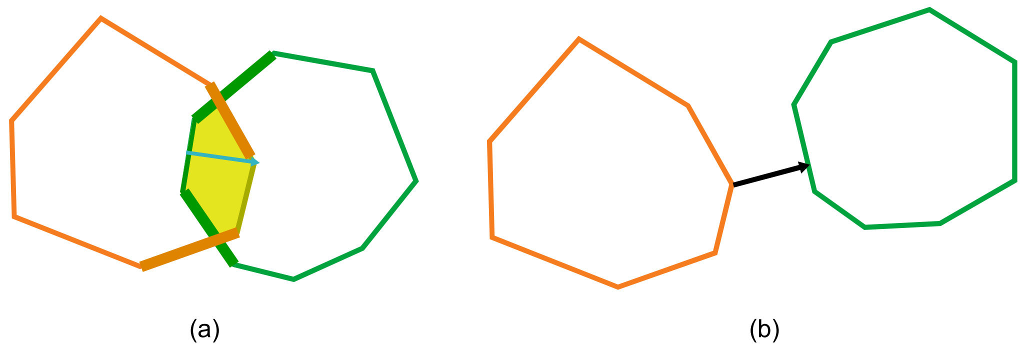


Figure 1.5: Different proximity queries between 2 polygons. *(a) The two polygons overlap. The colliding features are shown in bold lines. The penetration depth and direction is given by the blue arrow. (b) The two polygons do not overlap. The separation distance is given by the black arrow.*

The *Penetration Depth* (PD) query measures the extent of overlap between two intersecting objects. The two objects are assumed to be orientable 2-manifolds in the region of penetration. This guarantees that we have a well defined 'interior' each penetrating object.

The translational penetration depth is defined as the minimum translational distance needed to make the two overlapping objects disjoint [DHKS93]. In this thesis, we shall restrict discussion to translational penetration depth. More recently work has been done in handling rotational penetation depth. Examples of discrete collision, penetration depth and separation distance queries among 2D polygonal objects is provided in figure 1.5.

1.2 Previous Work

There has been extensive study on Voronoi diagrams and medial axis computation in various areas of computer science. This section describes some of the previous work and challenges in computation of the Voronoi diagram of 3D polygonal models. Algorithms for computation of the Voronoi diagram of a general 3D polyhedron are broadly classified into *discrete* and *continuous* algorithms.

The discrete algorithms compute an approximation of the Voronoi diagram using a discrete sampling of the space. One class of these algorithms approximates the polyhedron by a finite set of point samples on the boundary. There are several efficient and robust algorithms for computing the Delaunay triangulation and Voronoi diagram of a set of points in 3D (see for e.g. [AK00]). However the Voronoi diagram of a set of points in 3D does not converge to the medial axis of the polyhedron. Given a sufficiently dense point sampling of the boundary of a smooth 3D object, practical algorithms have been proposed to compute a subset of the Voronoi diagram of points which converges to its medial axis [ACK01a, DZ02a]. However these approaches can guarantee convergence only for smooth objects - polyhedral models with sharp edges require infinite sampling.

Another class of discrete algorithms sample a subset of $\mathbb{R}^3$ at a finite set of points and compute distances to the higher order sites at this set of points. Thus, these algorithms compute a discrete Voronoi diagram of the 3D polyhedral model. The point samples may lie on uniform grids [HCK$^+$99b, FLM03], adaptive grids [VO98, ER02, BCMS05] or unstructured spatial subdivisions [TT97, YBM04, SS06]. Both classes of discrete algorithms compute the Voronoi diagram only to a certain resolution. The exact Voronoi diagram is computed as the underlying sampling approaches infinity. In practice, these algorithms stop at a finite resolution and return an approximate Voronoi diagram. A common deficiency of these methods is that there are no guarantees on the topology of the approximate Voronoi diagram.

Algorithms for accelerating the discrete Voronoi diagram computation on a uniform grid using graphics hardware have been presented [WND97, HCK$^+$99b, Den03b]. The graphics processing unit (GPU) is a programmable parallel vector processors designed for rapid rasterization of geometric primitives. These approaches present distance field and Voronoi diagram computation as a rasterization problem which can be efficiently performed in parallel on a uniform 2D grid. However the existing algorithms may not be interactive for computations on complex deformable 3D models.

A continuous Voronoi diagram algorithm is independent of the sampling resolution parameter. The continuous approaches for computing the exact Euclidean Voronoi diagram are based on *tracing* algorithms [Mil93, SPB96, Cul00]. The tracing algorithm operates on the graph formed by Voronoi edges (seam curves) and Voronoi vertices (junction points) of the 3D Voronoi diagram. The main idea is to construct this graph using breadth-first or depth-first search. The algorithm requires exact computation of intersection points between (non-linear) algebraic curves and ordering of points along an algebraic curve. Thus, the required precision varies as a function of root isolation algorithms and bounds. As a result, continuous methods based on floating-point arithmetic are sensitive to round-off error, while methods based on exact arithmetic are not efficient.

The challenges in computation and application of the Voronoi diagram and medial axis of 3D polyhedral models are as follows:

Combinatorial complexity: The combinatorial complexity of the 3D polyhedral Voronoi diagram is defined as the total number of Voronoi faces (sheets), edges (seams) and vertices (Junctions). Tight bounds on the worst-case complexity of the medial axis are not known. It is currently known to be $\Omega(n^2)$ and $O(n^{3+\epsilon})$, where n is the total number of faces, edges and vertices on the boundary of the polyhedron. Examples of the $\Omega(n^2)$ bound are provided by Culver [Cul00]. The upper bound follows from Sharir and Agarwal [SA95].

In the same work, Sharir and Agarwal present a randomized algorithm for computing the Voronoi diagram in $O(n^{3+\epsilon})$ time. However, they assume that certain operations, such as junction location, take constant time, and this approach may not be suitable for practical implementation. The tracing algorithms have a cost of $O(nm)$, where m is the complexity of the Voronoi diagram. The cost of the discrete methods depends on the desired resolution and input complexity. For a dense point sampling of a polyhedral surface, the complexity of the Delaunay triangulation is linear in number of points [AB02].

Robust Computation: The accuracy of continuous geometric algorithms depends on the reliable computation of underlying primitives. The two common reliability problems in implementations of geometric algorithms are: failure due to round-off error, and the inability to handle degenerate configurations. These problems are collectively called robustness problems.

Round-off error is introduced due to finite precision of computer arithmetic. Geometric algorithms are quite sensitive to roundoff error. A small numerical error can lead to incorrect evaluation of a geometric predicate possibly resulting in invalid output.

In addition, often a geometric algorithm assumes, for simplicity, that its input is in *general position*. This means that certain rare configurations are disallowed, and an infinitesimal perturbation of the input usually breaks the configuration. Such a configuration is called a *degenerate configuration*. This problem is exaggerated by finite-precision arithmetic. In a floating-point implementation, non-degenerate data may become degenerate. Also, degeneracies may be hard to detect and resolve. Moreover, real world data is typically not in general position. In particular, models of synthetic objects (for e.g. CAD parts) often exhibit symmetry which may lead to degeneracies for geometric algorithms.

One approach for handling robustness problems is to design algorithms which are stable in the presence of round-off error, and insensitive to presence of degeneracies. However, such an approach is rather limited. Many approaches require either a systematic detection of degenerate cases or a systematic way of applying a small perturbation. Such approaches require some form of exact computation.

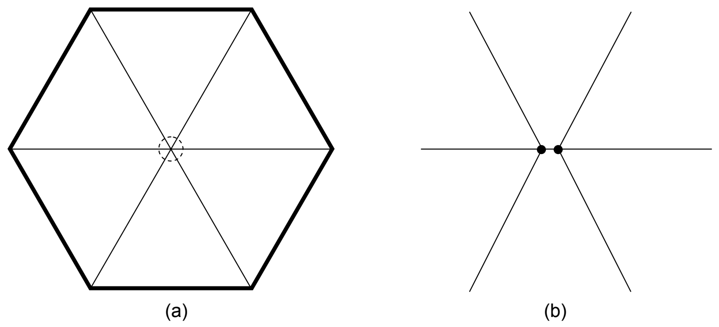


Figure 1.6: Near-degenerate configuration of the Voronoi diagram. *An almost regular hexagon. The right-most vertex has been perturbed by an infinitesimal amount. (a) The Voronoi diagram of the hexagon (b) Magnified view of the center showing two junctions. Arbitrarily large precision may be required to distinguish the junctions.*

Specifically for the continuous algorithms for Voronoi diagram computation, one needs reliable computation of locations of junction points and their ordering on seam curves. This involves solving a system of non-linear tri-variate equations, which reduces to solving equations up-to degree eight. The degenerate cases for Voronoi diagrams of polyhedral models are listed by Culver [Cul00], and some of these cases can be common in solid modeling and processing. An example of a near-degenerate junction is shown in figure 1.6. To detect and correctly handle these degeneracies, Culver uses exact computation, which may not scale efficiently to complex models composed of tens of thousands of polygons.

Instability: The Voronoi diagram and medial axis of a polyhedral model are unstable.

This means that small modifications to the boundary of a polyhedron can induce large modifications in its Voronoi diagram. A 2D example is shown in figure 1.7. The problem is exaggerated by the combinatorial complexity of the Voronoi diagram. Noise on the boundary also leads to higher combinatorial complexity. Thus a practical application requires extraction of a stable approximation of the Voronoi diagram or medial axis.

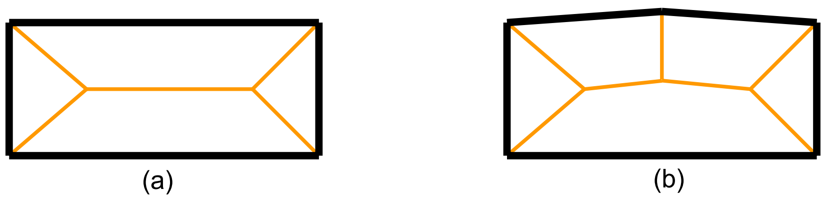


Figure 1.7: Instability of the medial axis. *(a) The medial axis (orange) of a simple polygon. (b) Small perturbation in boundary causes large change in the medial axis.*

The stability of the medial axis under small perturbations has been recently studied, and it has been shown that for small perturbation in the boundary of a shape, the original medial axis is contained inside a tight parallel body of the noisy medial axis [ABE04, CS02]. This result provides promise for extraction of a simplified medial axis corresponding to its stable part. The most common approach for medial axis simplification removes parts of the medial axis using a threshold on an importance measure (see [ABE04] for a survey). Typical importance measures used are distance to boundary and the angle subtended by the vectors from a point on the medial axis to its footprints.

In addition, several application require preservation of the homotopy type of the medial axis. However, existing medial axis simplification approaches are limited to discrete inputs, and cannot provide guarantees on the topological correctness of the output. For example, the medial axis is used to compute a high quality volumetric mesh for FEM analysis [DMB$^+$96]. An approach for computing a simplified approximate medial axis using point samples on the boundary may lead to introduction of artificial holes or miss certain topological features, as shown in figure 1.8.

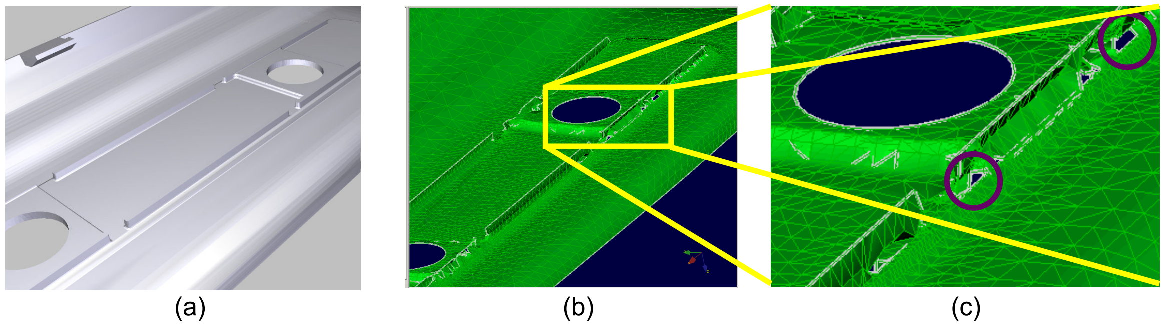


Figure 1.8: Approximate medial axis computation without topological guarantees. *(a) A thin CAD model. (b) Approximate medial axis computed using Voronoi diagram of points. (c) Zoomed view of the computed medial axis highlighting regions where homotopy type is not preserved. (Images courtesy GMSWorks, C-Solutions Inc.)*

Application to Proximity Queries: The problem of fast and reliable geometric proximity queries has been extensively studied. Despite the vast literature, real-time proximity queries remain one of the major bottlenecks for interactive deformable simulation [TKH+05, MHTG05]. Many existing methods are based on hierarchical representations and work well for rigid models. Several efficient collision detection algorithms have been proposed for deformable models, but they do not compute separation or penetration distances.

The upper bound on complexity of collision detection and separation distance computation is $O(m^2)$ where m is the number of primitives. In practice, only a small subset of primitive pairs needs to be tested for exact proximity computations. Most prior algorithms for proximity queries perform culling tests based on bounding volume hierarchies such as spheres or axis-aligned bounding boxes (AABBs). Tighter bounding volume hierarchies achieve higher culling at a cost of increased computation time. Thus these algorithms may not work well for dynamic environments involving close proximity scenarios or for intra-object queries. In addition, exact computation of PD between two polyhedral models is a global problem and cannot be solved using any 'divide-and-conquer' or localized approach [KOLM02]. Its worst complexity can be as high as $O(n_1^3 n_2^3)$, where n_1, n_2 are the number of boundary primitives on each polyhedral model.

External Voronoi regions of convex polytopes have been used to perform collision and distance queries between rigid objects that can be represented as union of convex polytopes [LC91b, Mir98]. These algorithms have been implemented within different proximity query packages such as I-COLLIDE, V-CLIP and SWIFT++ [Eri04]. However, it is hard to extend these algorithms to general non-convex deformable models. This is due to lack of robust algorithms for computing Voronoi diagrams of polygonal models at interactive rates.

Thus, robust and efficient computation of the Voronoi diagram and medial axis of 3D polyhedral models remains a challenging problem, limits their application.

1.3 Thesis

Our thesis is

The discrete Voronoi diagram and simplified medial axis of 3D polyhedral models can be computed efficiently with geometric and topological guarantees, and can be used for fast proximity queries among multiple deformable models.

In this thesis we present efficient algorithms for computing discrete Voronoi diagram and approximate medial axis of complex 3D polyhedral models. We describe an algorithm to compute 3D distance fields of geometric models by using a linear factorization of Euclidean distance vectors. This formulation maps directly to the linearly interpolating graphics rasterization hardware and enables us to compute distance fields of complex 3D models at interactive rates. We also use clamping and culling algorithms based on properties of Voronoi diagrams to accelerate this computation. We provide geometric guarantees on the result using Hausdorff distance bounds.

We present a unified approach for performing different proximity queries among multiple deformable models using second order discrete Voronoi diagrams. We show the reduction of different proximity queries to specializations of N-body distance queries. We also present an algorithm to reliably and efficiently compute N-body distance queries using the second order discrete Voronoi diagram.

We also present an adaptive sampling algorithm to provide topological guarantees on the approximate Voronoi diagram. Our algorithm uses subdivision criteria to compute an approximate Voronoi diagram which is homotopy equivalent to the exact Euclidean Voronoi diagram, and can handle degenerate configurations in the input polyhedron. The subdivision criteria is based on computing the arrangement of 2D conic sections, which can be performed accurately and efficiently [Be05, KCMh99]. Finally, we present an algorithm for efficiently computing the homotopy-preserving simplified medial axis from the approximate Voronoi diagram.

1.3.1 New Results

This dissertation presents contributions to interactive computation of discrete Voronoi diagrams, fast proximity computation among multiple deformable models and efficient computation of a homotopy-preserving Voronoi diagram and simplified medial axis for polyhedral models. The main results are described below.

Interactive computation of Discrete Voronoi Diagrams

- *Linear factorization of distance vectors*. We present an elegant geometric formulation to represent the Euclidean norm distance vector from a point on a plane to a site as a bilinear interpolation of the distance vectors along the principal axes. This enables efficient computation of distance functions using linearly interpolating graphics hardware.

- *Computation of Voronoi region bounds*. We present a multi-pass algorithm that exploits geometric properties of Voronoi diagrams to compute Voronoi region bounds in 2D and 3D. The underlying approach also extends to higher dimensions.

- *Culling techniques for distance computations*. We present culling techniques for 3D distance field and discrete Voronoi diagram computation using Voronoi region bounds and coherence of distance fields. The domain is divided into ranges, and sites which do not contribute to the Voronoi diagram within a range are culled away.

- *Surface Distance Maps:*. We present a new algorithm for computing the distance map and discrete Voronoi diagram on a triangulated mesh. We use simple texture representation to store a piecewise planar parametrization of the mesh. The parameterization defines an affine transformation for each primitive of the mesh. The affine transformation of the geometric primitive is applied to compute the distance functions of 3D primitives using the texture mapping hardware. This representation, called surface distance map, is used to perform efficient and accurate proximity queries.

Fast proximity computation among multiple deformable models

- *N-body distance query:* We introduce a unified approach to perform different proximity queries using *N-body distance computation*: given a set $\mathcal{P}$ of primitives, for each primitive p_i we compute the closest primitive in $\mathcal{P} \setminus \{p_i\}$. We also present efficient algorithms for continuous collision detection and local penetration depth computation based on the N-body distance query.

- *Voronoi-based culling:* We use properties of Voronoi diagrams to perform the N-body distance query efficiently. The closest primitive to any primitive (p_i) is one of the Voronoi neighbors of p_i. Therefore, the Voronoi diagram of primitives is an efficient data structure to perform *N-body distance culling*. We use the *2ndorder Voronoi di-*

agram because it provides information about two closest primitives at each point in space and results in a higher culling efficiency.

- *Fast and conservative computations using discrete Voronoi diagrams:* The exact computation of continuous 3D Voronoi diagrams for general triangulated models is a hard problem. Instead, we compute discrete Voronoi diagrams on a uniform grid using graphics hardware. We exploit properties of the 2nd order Voronoi diagram to derive distance error bounds that take into account discretization and sampling errors in discrete Voronoi diagrams. We use the distance bounds to efficiently compute the closest primitive at object-space precision i.e. IEEE 64-bit floating point accuracy.

Computation of homotopy-preserving simplified medial axis

- *Topological properties*: We present an algorithm to compute an approximate Voronoi diagram that is homotopy equivalent to the exact Voronoi diagram. To provide this guarantee we exploit topological properties of the Voronoi diagram of a polyhedral model, and use a subdivision scheme based on accurate 2D tests.

- *Computing arrangement of 2D conic sections:* The topological tests used in the subdivision scheme reduce to computing an arrangement of 2D conic sections on a plane, instead of computing an arrangement of 3D quadric surfaces. The arrangement of 2D conics has been well studied and good implementations are available. As a result, this algorithm is relatively simple to implement as compared to exact 3D Voronoi diagram computation algorithms and does not require arbitarily high precision to compute the junctions.

- *Homotopy-preserving medial axis simplification*: We present an efficient algorithm to compute a medial axis approximation which is homotopy equivalent to the given polyhedron. The algorithm uses iterative pruning of medial axis parts based on a stability measure and efficient local tests.

- *Computing stability of medial axis parts*: We present a relationship between the stability of medial axis junctions and seams to stability of incident sheets using separation angles. We also present an algorithm to compute a bounded approximation of this stability measure using discrete sampling.

As compared to prior approaches, the algorithms presented in this thesis have several advantages:

- **Generality:** The distance field and discrete Voronoi diagram computation algorithms make no assumptions with regards to the input models. The objects can have complex topologies, may be non-orientable or non-manifold surfaces, or may be represented using voxel data.

- **Non-rigid Motion:** The algorithms involve no precomputation and are directly applicable to dynamic models undergoing non-rigid motion, and changing topologies.

- **Efficiency:** Discrete Voronoi diagram and proximity computation are up-to an order of magnitude faster than prior approaches, with improved culling efficiency and tighter bounds. As a result, we achieve interactive performance for complex models consisting of thousands of primitives.

- **Accuracy:** Discrete Voronoi diagram computation on the GPU is performed at 32-bit floating point precision. In addition, tight geometric bounds are provided on the accuracy of the discrete Voronoi diagram. In addition, the algorithms are designed to account for under-sampling errors due to limited frame buffer precision on the GPU.

- **Topological Guarantees:** Homotopy equivalence is guaranteed even in presence of near-degenerate configurations of the Voronoi diagram.

1.3.2 Organization

The rest of the dissertation is organized as follows. Chapter 2 presents algorithms for interactive 3D distance field and discrete Voronoi diagram computation using graphics hardware. Chapter 3 introduces surface distance maps. Chapter 4 demonstrates the application of discrete Voronoi diagrams and surface distance maps to fast proximity computation among multiple deformable models. Chapter 5 presents the algorithms for computing homotopy-preserving approximate Voronoi diagram and simplified medial axis. Finally, chapter ?? summarizes the main conclusions and discusses future research directions.

Chapter 2

Fast 3D Discrete Voronoi Diagram Computation

Many algorithms have been proposed to compute discretized Voronoi diagrams and distance fields along uniform grids using graphics rasterization hardware [WND97, HCK$^+$99b, SPG03]. Graphics processing units (GPUs) are programmable vector processors designed for rapid rasterization of geometric primitives. Moreover, GPUs have been growing at a rate faster than the Moore's Law over the last decade, making them attractive for certain general purpose parallel computations.

These existing algorithms rasterize the distance functions of the geometric primitives and use the depth buffer hardware to compute an approximation of the lower envelope of the distance functions. The algorithms for general polygonal primitives approximate the non-linear distance functions using a distance mesh. This can be expensive for complex models and the accuracy of the overall approach is governed by the tessellation error. As a result, previous techniques are unable to compute 3D distance fields of complex models at interactive rates.

In this chapter we present algorithms for interactive computation of 3D distance fields and discrete Voronoi diagrams using graphics hardware. we present the terminology, and provide a brief overview of general purpose computation capabilities of current graphics hardware. We then present linear factorization of Euclidean distance vectors. This formulation maps directly to the linearly interpolating graphics rasterization hardware and enables us to compute distance fields of complex 3D models at interactive rates. We also use clamping and culling algorithms based on properties of Voronoi diagrams to accelerate this computation.

2.1 Previous Work

In this section we give a brief overview of previous work on computing discrete Voronoi diagrams and distance fields. We also mention some GPU-based algorithms to evaluate non-linear functions.

2.1.1 Distance Fields and Discrete Voronoi Diagrams

The algorithms for distance field and discrete Voronoi diagram computation can be broadly categorized based on different model representations such as images, volumes or polygonal representations.

Image datasets: Given discrete binary image data, many exact and approximate algorithms for distance field and discrete Voronoi diagram computation have been proposed. A good overview of these algorithms has been given in [Cui99]. The approximate methods compute the distance field in a local neighborhood of each voxel. Danielsson [Dan80] uses a scanning approach in 2D based on the assumption that the nearest object pixels are similar. The Fast Marching Method (FMM) [Set99] propagates a contour to compute the distance transformation from the neighbors. This provides an approximate finite difference solution to the Eikonal Equation $|\nabla u| = 1/f$. Repeated application of the local masks of the approximate algorithms till a stable solution is reached provides exact distance transforms. A parallel algorithm for this is proposed in [Yam84]. Efficient implementations of this store a propagation front in dynamic lists [Egg98, Rag92]. Propagation methods can be augmented by storing additional information like direction vectors to nearest voxel [Mul92] and closest features [HLC$^+$01]. Propagating additional information along with the FMM, leads to an exact distance transform algorithm [BMW00]. A linear time algorithm for computing exact Euclidean distance transform of a 2-D binary image is presented in [BGKW95]. This is extended to k-D images and other distance metrics [MQR03].

Geometric Models. There is extensive work in computing the exact Voronoi diagram of a set of points as the dual of the Delaunay triangulation of the points. A good survey of these algorithms is given in [Aur91].

For geometric models represented using polygonal or higher order surfaces in 3D, many algorithms compute an approximation to the Voronoi diagram by computing distance fields on a uniform grid or an adaptive grid. A key issue in generating discrete distance samples is the underlying sampling rate used for adaptive subdivision. Adaptive refinement strategies for distance field computation use trilinear interpolation or curvature information to generate an octree spatial decomposition [SFYC96, FPRJ00, PF01]. Although, these algorithms optimize the sparsity of the octree representation, the approximation using a trilinear interpolation may not work well for curved primitives or when the final surface has a lot of sharp features. Adaptive refinement approaches for Voronoi diagram computation use a Voronoi region based labeling of the sample points to generate an spatial decomposition [VO98, TT97, ER02].

Computation of a discrete Voronoi diagram on a uniform grid can be performed efficiently using parallel algorithms implemented on graphics hardware. Hoff et al. [HCK$^+$99a] render a polygonal approximation of the distance function on depth-buffered graphics hardware and compute the generalized Voronoi Diagrams in two and three dimensions. This approach works on any geometric model that can be polygonized and is applicable to any distance function that can be rasterized. An efficient extension of the 2-D algorithm for point sites is proposed in [Den03a]. It uses precomputed depth textures, and a quadtree to estimate Voronoi region bounds. However, the extension of this approach to higher dimensions or higher order primitives is not presented. A class of exact distance transform algorithms is based on computing partial Voronoi diagrams [Lin93]. A scan-conversion method to compute the 3-D Euclidean distance field in a narrow band around manifold triangle meshes is the Characteristics/Scan-Conversion (CSC) algorithm [Mau03]. The CSC algorithm uses the connectivity of the mesh to compute polyhedral bounding volumes for the Voronoi cells. The distance function for each site is evaluated only for the voxels lying inside this polyhedral bounding volume. An efficient GPU based implementation of the CSC algorithm is presented in [SPG03]. The number of polygons sent to the graphics pipeline is reduced and the non-linear distance functions are evaluated using fragment programs.

2.1.2 GPU-Based Non-linear Computations

Many algorithms have been proposed to exploit the programmability features of GPUs to evaluate and render higher order functions or surfaces. Shieu etal. [SJP05] used fragment programs to evaluate the Catmull-Clark subdivision surfaces. Their approach represented the control points in texture memory and used a fragment program to compute bicubic B-spline surfaces. Purcell et al. [PDC$^+$03] presented ray-tracing algorithms by using fragment programs to evaluate ray-primitive intersections. Kanai and Yashui [KY04] presented an improved algorithm to compute per-pixel normals on subdivision surfaces. Elegant algorithms to directly render curves and algebraic and parametric surfaces such as NURBS and T-spline surfaces have also been proposed [GBK05, LB05, LB06]. In contrast with these approaches, our algorithm explicitly decomposes the distance functions into linear factors and uses bilinear interpolation capabilities of the texture mapping hardware to evaluate these functions on a planar domain.

2.2 Notation and Background

In this section we provide notation used in the chapter and brief overview of graphics processing units (GPUs) with specific details on features useful for interactive distance field computation. We also summarize the existing work on distance field computation using GPUs.

2.2.1 Terminology

Let $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ denote a point in n dimensions. For points in 3-dimensions, we use the standard Cartesian coordinates, $\mathbf{q} = (q_x, q_y, q_z)$. A geometric primitive or an object is called a *site*. In this work, a site is a vertex, an open edge or an open triangle. The *pivot* point of a site is any point lying on the site, and is represented $\kappa(p_i)$. In practice, we use the centroid of a site as its pivot point.

Given a site p_i, the scalar distance function $d(\mathbf{q}, p_i)$ denotes the distance from the point $\mathbf{q} \in \mathbb{R}^n$ to the closest point on p_i. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ denote a set of m sites. The minimum distance of $\mathbf{q}$ to $\mathcal{P}$ is represented as $d(\mathbf{q}, \mathcal{P}) = \min_{p_i \in \mathcal{P}}(d(\mathbf{q}, p_i))$. The *distance field* $D_{\mathcal{D}}(\mathcal{P})$, for a domain $\mathcal{D} \subset \mathbb{R}^3$, is the scalar field given by the minimum distance function $d(\mathbf{q}, \mathcal{P})$ for all points $\mathbf{q} \in \mathcal{D}$. For ease of notation, let $D_{\mathcal{D}} = D_{\mathcal{D}}(\mathcal{P})$. Given a subset, $\mathcal{X} \subset \mathcal{P}$, $d(\mathbf{q}, \mathcal{X}) \geq d(\mathbf{q}, \mathcal{P}) \forall \mathbf{q} \in \mathcal{D}$.

Distance fields are closely related to Voronoi diagrams. The Voronoi region for p_i is defined as:

$$\mathcal{V}(i) = \{\mathbf{q} \mid d(\mathbf{q}, p_i) \leq d(\mathbf{q}, p_j) \forall p_j \in \mathcal{P}, \mathbf{q} \in \mathcal{D}\}$$

The Voronoi diagram is a partition of $\mathcal{D}$ into m Voronoi regions:

$$\mathrm{VD}(\mathcal{P}, \mathcal{D}) = \bigcup_{p_i, p_j \in \mathcal{P}, i \neq j} \mathcal{V}(p_i) \cap \mathcal{V}(p_j).$$

The Voronoi diagram can be represented as the projection of the distance field to the domain $\mathcal{D}$ [ES86].

Our goal is to compute a *discrete Voronoi diagram* within a bounded domain $\mathcal{D}$. Given a finite set of point samples $\widetilde{\mathcal{D}}$ in domain $\mathcal{D}$, and a set of sites $\mathcal{P}$, the *discrete Voronoi diagram* (DVD) is a partition of the point samples onto discrete Voronoi regions, and is denoted as $\widetilde{\mathrm{VD}}(\mathcal{P})$. The discrete Voronoi region is a finite set of points which are closer to one site than to any other site, and is denoted $\widetilde{\mathcal{V}}(p_i)$. We require that the domain $\mathcal{D}$ is a superset of the bounding box of all sites. This assumption is used to guarantee correctness of the culling algorithm. The finite set of point samples $\widetilde{\mathcal{D}}$ lie along a uniform grid. Without loss of generality, we can scale the domain $\mathcal{D}$ to be the half-open unit interval in n-dimensions $(0, 1]^n$, i.e a unit square in 2D and the unit cube in 3D. To apply our culling algorithm in n-dimensions, we introduce n-D ranges. We shall refer to an n-dimensional hypercube as an n-D *range* (a rectangular tile in 2D, a cube in 3D). The n-D range $(a_0, b_0] \times (a_1, b_1] \times \ldots \times (a_n, b_n]$ is represented as $T_{(a_0,b_0](a_1,b_1]\ldots(a_n,b_n]}$. We define each range to be a half-open set such that the intersection between two ranges is empty, and any point in $\mathcal{D}$ belongs to exactly one range. For ease of notation, let $\mathrm{VD}(\mathcal{P}, \mathcal{D}) = \mathrm{VD}(\mathcal{P})$. In this chapter, we shall denote the Voronoi diagram computed on a range T as $\mathrm{VD}^T(\mathcal{P})$. Let $\mathcal{X}^c$, $\partial\mathcal{X}$ and $\mathrm{Int}(\mathcal{X})$ denote the complement, boundary and interior of a set $\mathcal{X}$, respectively. For any domain $N \subseteq \mathcal{D}$, $N^c = \mathcal{D} \setminus N$. For 3D grids, the set of cells with a constant z-value represents a uniform 2D grid and is called a *slice*. A slice s_k is defined as $s_k = \{(x, y, z) | (x, y, z) \in \mathcal{D}, z = z_k\}$.

2.2.2 Distance Field Computation using GPUs

A brute-force algorithm to compute $\widetilde{\mathrm{VD}}(\mathcal{P})$ would evaluate $d(\mathbf{q})p_i$ for all sites $p_i \in \mathcal{P}$ and store the minimum at each grid point $\mathbf{q} \in \widetilde{\mathcal{D}}$. If there are m sites, and the grid has M cells, the time complexity of this algorithm is $O(mM)$. This brute force algorithm can be easily parallelized using depth-buffered graphics rasterization hardware [HCK+99a]. Graphics hardware is well suited for performing parallel computations on a 2D grid. Computation of the lower envelope is posed as a visibility problem along a view direction that is orthogonal to the 2D grid. The visibility test is efficiently performed using the depth test on graphics hardware. The discrete Voronoi diagram is computed along with the distance field. In 2D, the resolution of the grid is governed by the image-space resolution of the graphics processors (e.g. 1000×1000).

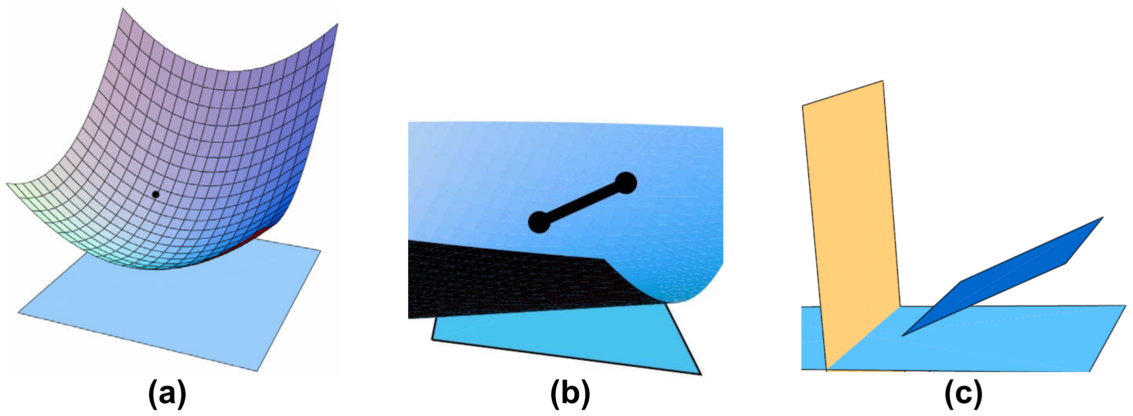


Figure 2.1: Quadratic distance function from a site to a plane*: The graph representing the Euclidean distance from a site to a point on a plane is a quadric surface. (a) A paraboloid for a point site (b) An elliptical cone for a line site (c) A pair of planes for a planar polygon.*

The 3D distance field is computed by sweeping along the Z axis. For each slice, the distance field is computed using the distance from the sites to the plane [HCK+99b, SPG03]. The underlying distance function is a degree two function (i.e. a quadric surface in 3D). For example, the Euclidean distance function of a point site to a plane is one sheet of the hyperboloid, and of a line to a plane is an elliptical cone, shown in figure 2.1.

Given a complex model with tens of thousands of sites, evaluating the non-linear distance function for each site can be expensive. Hoff et al. [HCK+99b] computed a piece-wise linear approximation of the distance function using a polygonal distance mesh. However, the linear approximation introduces tessellation error. Moreover, the overhead of computing the polygonal approximation can be high for interactive applications. Sigg et al. [SPG03] used the programmable capabilities of GPUs to evaluate the non-linear distance function at each point (or pixel) on the plane. They briefly mention use of bilinear interpolation, and present an approach which uses several instructions per fragment to compute the distance function. However, their approach does not compute the exact discrete Voronoi diagram.

2.3 Linear Factorization

In contrast to earlier approaches, we compute the distance function for each site by evaluating the distance vector field on the GPU. A distance vector field consists of vectors from the 3D points to the closest point on the site. The magnitude of the distance vector provides the value of the distance function of the site at a grid point. We first present a formulation to compute the distance vector at any point on a planar polygon by using the distance vectors of the polygon vertices to the site. Next, we present techniques to compute these planar polygons which bound a site's Voronoi region on a slice. Finally, we map the problem of distance vector computation to texture mapping hardware on the GPU. Linear factorization

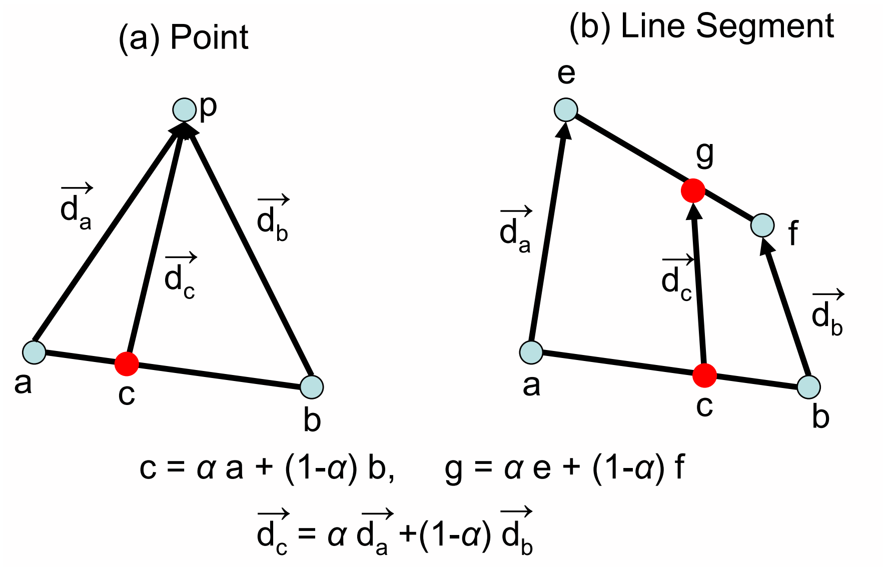


Figure 2.2: Distance vector computation: *This figure illustrates the distance vector computation at any point on a plane. The distance vector at an interior point is a bilinear interpolant of the distance vectors at the vertices.*

of distance vectors is used to evaluate the distance functions. Formally speaking, the linear factorization expresses the distance vector at each point inside the polygon in terms of bilinear interpolation of the distance vectors of the polygon vertices. Given a convex polygon P with vertices $(v_1, \ldots, v_k)$, the linear factorization expresses the distance vector at any interior point $\mathbf{p}$ of the polygon as

$$\vec{\mathbf{d}}(\mathbf{p}, p_i) = \sum_{i=1}^{k} \alpha_i \vec{\mathbf{d}}(\mathbf{v_i}, p_i),$$

where

$$\mathbf{p} = \sum_{i=}^{k} \alpha_i \mathbf{v_i}, 0 \leq \alpha_i \leq 1 \text{ and } \sum_{i=1}^{k} \alpha_i = 1.$$

We present three key properties of distance vector computation at a point on a plane to point sites, line segment sites, and triangular sites. These properties are used to evaluate the distance functions efficiently. We first highlight the property to perform linear factorization of the distance vector of a point on a line to a point site.

Property 2.1. *Given two points* $\mathbf{a}$ *and* $\mathbf{b}$ *on a plane and a point site* $\mathbf{p}$*. Let* $\vec{\mathbf{d_a}}$ *and* $\vec{\mathbf{d_b}}$ *denote the distance vectors of* $\mathbf{a}$ *and* $\mathbf{b}$ *to* $\mathbf{p}$ *respectively. Then, the distance vector* $\vec{\mathbf{d_x}}$ *of any point* $\mathbf{x} = \alpha\mathbf{a} + (\mathbf{1} - \alpha)\mathbf{b}, \mathbf{0} \leq \alpha \leq \mathbf{1}$ *is the linear combination of distance vectors of* $\mathbf{a}$ *and* $\mathbf{b}$*, and* $\vec{\mathbf{d_x}} = \alpha\vec{\mathbf{d_a}} + (1 - \alpha)\vec{\mathbf{d_b}}$.

Property 2.1 indicates that given the distance vectors of the vertices of any planar primitive to a point site, the distance vector of any interior point can be computed using a bilinear interpolation of the distance vectors of vertices.

The distance vector of a point $\mathbf{x}$ that projects orthogonally to the interior of a line segment is a vector perpendicular to the line. We use the following property to perform linear factorization of the distance vector of a point that projects onto a line segment.

Property 2.2. *Given two points* $\mathbf{a}$ *and* $\mathbf{b}$ *on a plane and a line segment* $\mathbf{l}$ *with end points* $\mathbf{e}$ *and* $\mathbf{f}$*. Let* $\vec{\mathbf{d_a}} = \mathbf{e} - \mathbf{a}$ *and* $\vec{\mathbf{d_b}} = \mathbf{f} - \mathbf{b}$ *denote the distance vectors of* $\mathbf{a}$ *and* $\mathbf{b}$ *to the site* $\mathbf{l}$

respectively. Then, the distance vector $\vec{\mathbf{d_x}}$ *of any point* $\mathbf{x} = \alpha\mathbf{a} + (\mathbf{1}-\alpha)\mathbf{b}, \mathbf{0} \leq \alpha \leq \mathbf{1}$ *is the linear combination of distance vectors of* $\mathbf{a}$ *and* $\mathbf{b}$*, and* $\vec{\mathbf{d_x}} = \alpha\vec{\mathbf{d_a}} + (1-\alpha)\vec{\mathbf{d_b}}$.

We use property 2.2 to compute the distance vector of any point that projects onto $\mathbf{l}$. This property indicates that given the vertices of a convex polygon whose projection lies within $\mathbf{l}$, the distance vector of any interior point in the convex polygon is the bilinear interpolation of the distance vectors of the convex polygon vertices to $\mathbf{l}$.

We extend Property 2.1 and Property 2.2 to compute the distance vector of a point $\mathbf{x}$ that projects interior to a triangular site. It can be seen that the distance vector of $\mathbf{x}$ is the normal to the triangle.

Property 2.3. *Given three points* $\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$ *on a plane and a triangular site* $\mathbf{t}$ *with vertices* $\mathbf{e}$, $\mathbf{f}$ *and* $\mathbf{g}$*. Let* $\vec{\mathbf{d_a}} = \mathbf{e}-\mathbf{a}$, $\vec{\mathbf{d_b}} = \mathbf{f}-\mathbf{b}$ *and* $\vec{\mathbf{d_c}} = \mathbf{g}-\mathbf{c}$ *denote the distance vectors of* $\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$ *to the site* $\mathbf{t}$ *respectively. Then, the distance vector* $\vec{\mathbf{d_x}}$ *of any point* $\mathbf{x} = \alpha_\mathbf{1}\mathbf{a} + \alpha_\mathbf{2}\mathbf{b} + (\mathbf{1}-\alpha_\mathbf{1}-\alpha_\mathbf{2})\mathbf{b}, \mathbf{0} \leq \alpha_\mathbf{1}, \alpha_\mathbf{2} \leq \mathbf{1}$ *is the linear combination of distance vectors of* $\mathbf{a}, \mathbf{b}$*, and* $\mathbf{c}$ *and* $\vec{\mathbf{d_x}} = \alpha_1\vec{\mathbf{d_a}} + \alpha_2\vec{\mathbf{d_b}} + (1-\alpha_1-\alpha_2)\vec{\mathbf{d_c}}$.

Property 2.3 indicates that given the vertices of a convex polygon projecting onto $\mathbf{t}$, the distance vector of any interior point is the bilinear interpolation of the distance vectors of the polygon vertices. Furthermore, the distance vectors of the polygon vertices are normal to the triangular site.

Lemma 2.1. *There exists a linear factorization to compute the distance vector of any point to a planar site.*

Proof. Trivial. Based on properties 2.1, 2.2, and 2.3. □

2.4 Domain Computation

In the previous section we showed that the distance vector from a point in the interior of a convex polygon to a site can be expressed as a bilinear interpolant of the distance vectors at the polygon vertices. In this section, we define the convex domain on a slice for which the distance function of a site is computed. We also present an approach to compute conservative bounds on the domain.

The region where the distance function of a site contributes to the distance field is exactly its Voronoi region. However, it is non-trivial to compute the exact Voronoi regions for higher order sites (i.e. lines, polygons) [CKM98]. Moreover, the Voronoi regions are not necessarily convex. Instead of computing the exact Voronoi region, we compute a convex polygonal domain $Q_{i,k}$ on the slice s_k, which bounds the intersection of the Voronoi region of site p_i with slice s_k.

We present separate algorithms for manifold and non-manifold sites. We first present an algorithm to compute the $Q_{i,k}$ for non-manifold sites. Later, we present an improved algorithm for manifold sites that exploits the connectivity to compute tighter convex polygonal domain.

Non-Manifold Sites we present the algorithm for a triangle and it can be easily extended to a convex polygon. For a triangle t_j with vertices $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$ and unit normal $\mathbf{n_t}$, consider the three (open) half-spaces given by planes through the three edges and perpendicular to the plane of the triangle. The half-spaces are given as $H_i = (\mathbf{x} - \mathbf{p_i}) \cdot \mathbf{n_i} \geq 0, i \in \{1, 2, 3\}$, where $\mathbf{n_i}$ satisfies $(\mathbf{p_i} - \mathbf{p_{i+1}}) \cdot \mathbf{n_i} = 0, \mathbf{n_t} \cdot \mathbf{n_i} = 0$. Any point in the intersection of these halfspaces is closer to the interior of the triangle, and any point not contained in the intersection $H_1 \cap H_2 \cap H_3$ is closer to one of the sites on the boundary of the triangle. The convex polygonal domain $Q_{j,k}$ is the intersection of the three half-spaces with slice s_k.

Given a line segment $\mathbf{e}_j$ with end points $\mathbf{p_1}$ and $\mathbf{p_2}$, consider the two (open) halfspaces defined by planes perpendicular to the line through the end points, $H_1 = (\mathbf{p_2} - \mathbf{p_1}) \cdot (\mathbf{x} - \mathbf{p_1}) >$

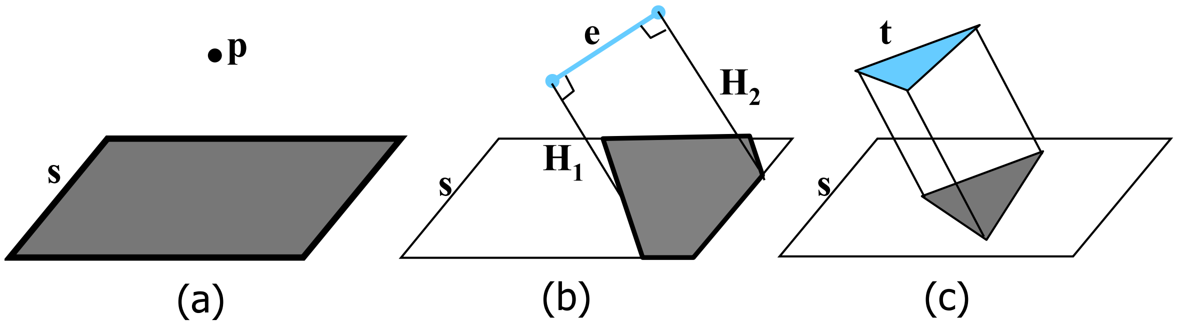


Figure 2.3: Domain of distance field computation on a slice s for non-manifold sites*: The domain of computation is shaded in grey. (a) For a point site p, the domain is the entire slice (b) For a line segment e, the domain is bounded by two parallel half-planes H_1 and H_2, perpendicular to e. (c) For a triangle t, the domain is bounded by triangular prism defined by intersection of three half planes perpendicular to triangle edges.*

0 and $H_2 = (\mathbf{p_1} - \mathbf{p_2}) \cdot (\mathbf{x} - \mathbf{p_2}) > 0$. Any point $\mathbf{x}$ in the intersection of the two half spaces is closer to the interior of the line segment $\mathbf{e}_,$ and a point not in the intersection will be closer to one of the end points. Thus, the convex polygonal domain $Q_{j,k}$ is the intersection of the two half spaces H_1, H_2 with the slice s_k.

Finally, given a point $\mathbf{p_j}$, the domain $Q_{j,k}$ is the entire slice s_k.

Manifold Sites For a manifold site, we exploit the neighborhood information to compute a convex polytope G_i which bounds the Voronoi region of a site p_i [Cul00, Mau03, SPG03]. For a particular slice s_k, the domain of computation of a site p_i is given by the intersection of the polytope with the slice, $Q_{i,k} = G_i \cap s_k$. For a triangle site, the bounding polytope is given by a triangular prism defined by intersection of three half spaces (as described above).

For an edge $\mathbf{e}$ incident on two triangles with normals $\mathbf{n_1}$, $\mathbf{n_2}$, the convex polytope is a wedge obtained by the intersection of four half-spaces. Two of the half-spaces are defined by parallel planes through the end vertices of the edge as shown above. The other two half-spaces are defined by planes containing the edge $\mathbf{e}$ and have normals $\mathbf{n_1}$ and $\mathbf{n_2}$.

For a point site $\mathbf{p}$, with n incident edges $\mathbf{e}_1, \ldots, \mathbf{e}_n$, the polytope $G_{\mathbf{p}}$ is given by intersection of half-spaces corresponding to planes through the point $\mathbf{q}$ and orthogonal to each incident edge $\mathbf{e}_i$, i.e. $G_{\mathbf{p}} = \bigcap_{1 \leq i \leq n} H_i, H_i = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{e_i} \geq 0$. Instead of exactly computing the half-space intersection, with time complexity $O(n \log n)$, we present a simple algorithm to compute a conservative approximation of the bounding polytope $G_{\mathbf{p}}$ for a point site $\mathbf{p}$ in $O(n)$ time.

A point site is defined as convex iff all edges incident on the point have an internal dihedral angle less than π. Let $\mathbf{p}$ be a convex point, with incident edges $\mathbf{e}_i$, $1 \leq i \leq n$ in order. Then edges of the convex polytope $G_{\mathbf{p}}$ are given by $\mathbf{p} + \lambda \mathbf{n_i}$, where $n_i = \mathbf{e}_i \times \mathbf{e}_{i+1}$ (modulo n) and $\mathbf{n_i}$ is the normal of the triangle t_i containing edges $\mathbf{e}_i$ and $\mathbf{e}_{i+1}$.

This construction does not work for hyperbolic points[PS05]. Previous approaches expand the bounding polytopes of adjacent triangles to handle hyperbolic points, however this results in a complex fragment program to compute the distance functions [SPG03]. We present an efficient algorithm to compute a bounding polytope $G_{\mathbf{p}}$ for a hyperbolic point $\mathbf{p}$ (see figure 2.4(a)). Let $\mathbf{n_a}$ be the average of the normals of all incident triangles. Let $\mathbf{n_j}$ be the normal which maximizes $\theta(i) = \frac{\mathbf{n_a} \times \mathbf{n_i}}{|\mathbf{n_a} \times \mathbf{n_i}|}, i = 1, \ldots, n$. We consider the case when $\theta(i) < \pi/2$. Let C be a right circular cone with axis $\mathbf{n_a}$ and opening angle $2\theta(j)$. We now prove that the bounding polytope $G_{\mathbf{p}}$ is a subset of cone C.

Theorem 2.1. *Let $\mathbf{p}$ be a manifold point, and C be a cone constructed as above. Then the convex polytope $G_{\mathbf{p}}$ bounding the Voronoi region of $\mathbf{p}$ is a subset of C.*

Proof. If $G_{\mathbf{p}} = \emptyset$, then the result trivially holds. It is sufficient to show that $G_{\mathbf{p}} \cap \pi_k \subseteq C \cap \pi_k$ for any plane π_k. Consider a plane π orthogonal to $\mathbf{n_a}$, and let Q be the convex polygon obtained by intersection of $G_{\mathbf{p}}$ with π, $Q = G_{\mathbf{p}} \cap \pi$. Let $\mathbf{x_i}$ be the intersection of the ray from $\mathbf{p}$ along direction $\mathbf{n_i}$. $G_{\mathbf{p}} \cap \pi$ is the convex region given by the intersection of 2D half spaces, each half space is given by the line through $\mathbf{x_i}$ and $\mathbf{x_{i+1}}$ (modulo n) (see figure 2.4(b)). For any point $\mathbf{y} \in \pi$, let $r = \max_{i=1,\ldots,n} d(\mathbf{x_i})\mathbf{y}$. By construction, a circle c with center $\mathbf{y}$ and radius r will contain Q. Taking $\mathbf{y} = \mathbf{x_a}$, we get $c = C \cap \pi$ and $G_{\mathbf{p}} \subseteq C$. □

Theorem 2.1 implies that any convex polytope containing the cone C will bound the

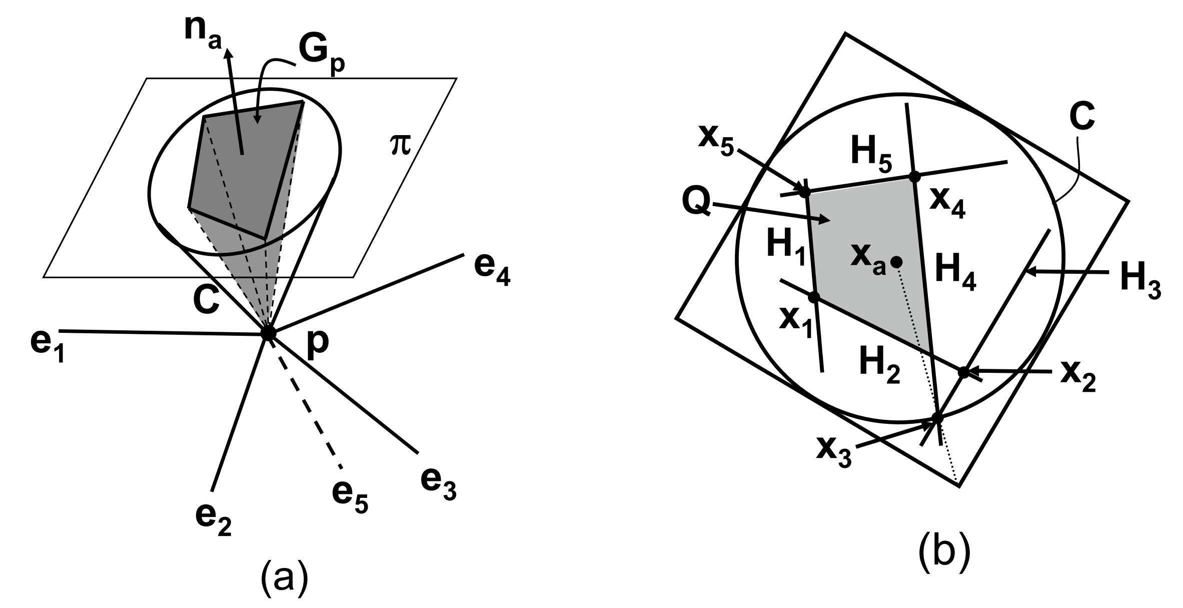


Figure 2.4: Bounding polytope computation for a hyperbolic point: $\mathbf{p}$ *is hyperbolic point with* 5 *incident edges* $\mathbf{e}_i$, $i = 1, \ldots, 5$. *(a) The polytope* $G_{\mathbf{p}}$ *bounds the Voronoi region of* $\mathbf{p}$. $G_{\mathbf{p}}$ *is bounded by a cone* C. *(b) Intersection of* $G_{\mathbf{p}}$ *with plane* π*, showing construction of* C.

bounding polytope $G_{\mathbf{p}}$ of a hyperbolic point $\mathbf{p}$. We use a square pyramid to approximate the bounding polytope as shown in figure. If $\theta(i) \geq \pi/2$, then we treat the hyperbolic point as non-manifold.

2.5 Distance Field Computation using GPUs

In this section, we present our algorithm for distance computation on the GPU using linear factorization. Given a site and a slice, the convex domain is computed as described in Section 2.4. The distance vector to the site is computed at each vertex of the convex polygon. The distance vector is encoded as a 3D texture coordinate, and the polygon is rasterized on the GPU.

Graphics processors (GPUs) have many specialized hardware units to perform bilinear interpolation on the attributes of vertices of polygons. These vertex attributes consist of the color, position, normal, or texture co-ordinates of the vertices. These attributes are 4-D vectors of the form (v_x, v_y, v_z, v_w) and transformations are applied to the attributes in the vertex processing unit. The transformed vertices are bilinearly interpolated by the rasterization hardware and the interpolated vectors at each fragment are used for lighting computations using Gouraud or Phong shading, or environment mapping applications. In order to achieve higher performance, the rasterization hardware consists of multiple vector units to compute the interpolated vectors.

We use the fast bilinear interpolation capabilities of GPUs for distance vector computation. Based on the linear factorization formulation, we map the distance vector computation to the GPUs using vertex attributes of the polygons. We use orthographic transformations and perform a one-to-one mapping between the points on the plane and the pixels on the screen. The bilinear interpolation of the vertex attributes is used for distance vector computation at each point on the plane.

The interpolation units in current GPUs can perform computations on different vector representations. For example, the vectors can be represented using 8-bit, 16-bit or 32-bit floating point values and the vector components may be signed or unsigned. Since the components of the distance vectors are signed and represented in 32-bit floating point precision, we use the 3D texture co-ordinates of the vertices to represent the distance vectors of vertices. The interpolated texture co-ordinates are used in a single instruction fragment program to compute the magnitude of the distance vector. The distance field is updated to compute the minimum either using the MIN instruction in the fragment program or by using the depth test functionality.

We now present the expressions for computing the distance vectors from a convex polygon vertex to a point, an edge and a triangle. These are:

- **Point:** Given a point site $\mathbf{p}$ and a vertex $\mathbf{v}$, the distance vector from $\mathbf{v}$ to $\mathbf{p}$ is $\vec{\mathbf{d}}(\mathbf{p}, \mathbf{v}) = \mathbf{p} - \mathbf{v}$.

- **Edge:** For an edge $\mathbf{e}$ with end points $\mathbf{p_1}$ and $\mathbf{p_2}$, the distance vector from a vertex $\mathbf{v}$ to

$\mathbf{e}$ is $\vec{\mathbf{d}}(\mathbf{e}, \mathbf{v}) = (\mathbf{p_1} - \mathbf{v}) + \lambda \frac{(\mathbf{p_2} - \mathbf{p_1})}{|\mathbf{p_2} - \mathbf{p_1}|}$, where $\lambda = \frac{(\mathbf{v} - \mathbf{p_1}) \cdot (\mathbf{p_2} - \mathbf{p_1})}{|\mathbf{p_2} - \mathbf{p_1}|}$.

- **Triangle:** Given a triangle t with a unit normal $\hat{\mathbf{n}}$, the distance vector from a vertex $\mathbf{v}$ to t is given as $\vec{\mathbf{d}}(t, \mathbf{v}) = [(\mathbf{p_i} - \mathbf{v}) \cdot \hat{\mathbf{n}}]\, \hat{\mathbf{n}}$, where $\mathbf{p_i}$ is one of the three vertices of the t.

The pseudocode for computing the distance field for a slice s_k is presented in Algorithm 1. The function *ComputePolygon*(p_i,s_k) computes the convex polygonal domain bounding the Voronoi region of site p_i on slice s_k.

```
  Input: slice $s_k$, site set $\mathcal{P}$
  Output: distance field $D_{s_k}(\mathcal{P})$
1 foreach site $p_i \in \mathcal{P}$ do
2     $Q_{i,k} \leftarrow$ ComputePolygon($p_i, s_k$)
3     foreach vertex $\mathbf{v} \in Q_{i,k}$ do
4         Compute distance vector $\vec{\mathbf{d}}(p_i, \mathbf{v})$
5         Assign texture coordinates of $\mathbf{v}$, $(r, s, t) \leftarrow \vec{\mathbf{d}}(p_i, \mathbf{v})$
6     end
7     Draw textured polygon $Q_{i,k}$ at depth $z = 0$
8 end
```

Algorithm 1: *ComputeSlice(s_k, $\mathcal{P}$): This algorithm computes the distance field for s_k for a set of primitives $\mathcal{P}$*

The algorithm for computing the distance field in the entire domain $\mathcal{D}$ is presented in Algorithm 2. The function *SetOrthoProjection*($\mathcal{D}$) sets up the projection matrix to be the bounds of the (axis-aligned) domain of computation $\mathcal{D}$. *NormProgram*() is a single instruction fragment program that computes the Euclidean norm of the 3D texture coordinate at a pixel and writes it out the value to the depth buffer. The functions *StartSlice*(s_k) and *EndSlice*(s_k) setup the rendering state at the beginning and end of computation of a given slice s_k. The state setup involves enabling a floating point rendering buffer, clearing the buffer and reading it back to the CPU after computation.

```
    Input: site set $\mathcal{P}$, domain $\mathcal{D}$, number of slices $m$
    Output: distance field $D_{\mathcal{D}}(\mathcal{P})$
 1  Enable depth test
 2  Set depth test function to less than
 3  SetOrthoProjection ($\mathcal{D}$)
 4  Enable fragment program NormProgram
 5  foreach slice $s_k$, $k = 1, \ldots, m$ do
 6      StartSlice ($s_k$)
 7      ComputeSlice ($s_k$,$\mathcal{P}$)
 8      EndSlice ($s_k$)
 9  end
10  Disable fragment program NormProgram
```
Algorithm 2: *Compute3D($\mathcal{P}$, $\mathcal{D}$, m): Computes the 3D discretized distance field on a uniform grid $\mathcal{D}$ with m slices.*

2.6 Culling Overview

The complexity of the algorithm presented in Section 2.5 is linear in m for each slice and the running time can be slow when m is large. In this section I will provide an overview of culling and clamping techniques we use to accelerate this computation.

We speed up the 3D distance field computation by reducing the number of distance functions that are rasterized for each slice. We exploit the following properties of Voronoi regions and distance fields to accelerate the computation:

1. **Connectivity:** We consider distance metrics that are symmetric, positive definite and satisfy the triangle inequality. Thus, Voronoi regions defined by that distance metric are connected. This is true for all L_p norms, including Euclidean distance and max-norm [CD85]. Note that for higher order sites, like line segments and polygons, each Voronoi region may consist of non-linear boundaries and may not be convex. But each Voronoi region is connected.

2. **Spatial Coherence:** The distance fields of adjacent slices, s_k and s_{k+1}, can have high spatial coherence. The distance values associated with two points in adjacent voxels on

a 3D grid will be very close to each other. We use this coherence to compute bounds on the maximum change in the distance field between adjacent slices.

3. **Monotonicity:** Given a slice, the distance function of a site is a monotonic function. It has a minimum value in the interior of the slice and is maximum on the boundary of the slice.

Our goal is to cull away sites that do not contribute to the final distance field for a particular slice. Furthermore, the distance field for each site should be computed in the region of the slice where it contributes to the final distance field (see Figure 2.5). Our algorithm utilizes the above mentioned properties and computes conservative bounds on the Voronoi regions. We use these bounds in two steps: to cull the set of sites for each slice (described in Section 2.7) and clamp the region of computation for each site in the non-culled set (described in Section 2.8).

2.6.1 Site Classification

We introduce a classification of the sites used by our algorithm to cull away sites that do not contribute to the distance field for a slice. For simplicity, first we shall introduce the notation for culling along Z-axis only. In section 2.9, we generalize the notation for culling along each axis in 2D and 3D. Let us assume that the sweep direction is along the $+Z$ direction. For a slice s_k at $z = z_k$, we partition the set of sites $\mathcal{P}$ into three subsets depending on the *Voronoi region bounds of each site* along the Z axis (shown in Figure 2.5):

Intersecting, $\mathcal{I}^+(k) = \{p_i \mid \mathcal{V}(i).z_{min} \leq z_k \leq \mathcal{V}(i).z_{max}\}$. Only the distance functions of these sites contributes to the final distance field of slice s_k.

Approaching, $\mathcal{A}^+(k) = \{p_i \mid \mathcal{V}(i).z_{min} > z_k\}$. The Voronoi region of an approaching site does not intersect with current slice, but could potentially intersect with a slice s_l, where $z_l > z_k$.

Receding, $\mathcal{R}^+(k) = \{p_i \mid \mathcal{V}(i).z_{max} < z_k\}$. Due to the connectivity property of Voronoi regions, a receding site can never become *intersecting*, hence it can be discarded for any slice s_l, where $z_l > z_k$.

For efficient computation, the algorithm presented in Section 2.7 performs two passes along $+Z$ and $-Z$ directions and considers only the sites swept up-to the current slice. We also partition $\mathcal{P}$ based on the *spatial bounds of each site* along Z axis. Let $p_i.z_{max}$ denote the maximum Z value of a site p_i. Then the set $\mathcal{P}$ is partitioned as (shown in Figure 2.5):

Swept, $\mathcal{S}^+(k) = \{p_i \mid p_i.z_{max} \leq z_k\}$

Unswept, $\mathcal{U}^+(k) = \{p_i \mid p_i.z_{max} > z_k\}$

The intersecting set $\mathcal{I}^+(k)$ can be further partitioned into an *intersecting swept* set $\mathcal{IS}^+(k) = (\mathcal{I}^+(k) \cap \mathcal{S}^+(k))$ and an *intersecting unswept* set $\mathcal{IU}^+(+,k) = (\mathcal{I}^+(k) \cap \mathcal{U}^+(k))$.

$$\mathcal{I}^+(k) = \mathcal{IS}^+(k) \cup \mathcal{IU}^+(+,k) \tag{2.1}$$

The set of sites, $\mathcal{P}$, is also partitioned into subsets along the $-Z$ sweep direction. The

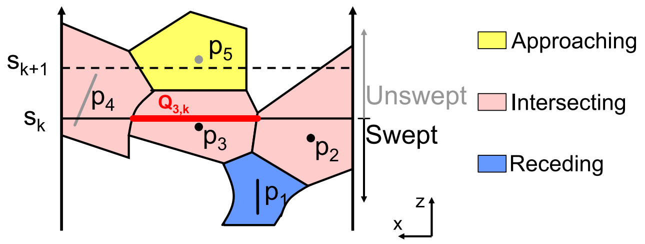


Figure 2.5: Site Classification*: Shaded areas represent the connected Voronoi regions for a subset of sites* $\{p_1, \ldots, p_5\}$*. Sweep direction is along* $+Z$*. For slice* s_k*, the site sets are: Intersecting* $\mathcal{I}^+(k) = \{p_2, p_3, p_4\}$*, Approaching* $\mathcal{A}^+(k) = \{p_5\}$*, Receding* $\mathcal{R}^+(k) = \{p_1\}$ *and Swept* $\mathcal{S}^+(k) = \{p_1, p_2, p_3\}$*, Unswept* $\mathcal{U}^+(k) = \{p_4, p_5\}$*. Distance functions have to be drawn for set* $\mathcal{I}^+(k)$ *only. For site* p_3*, the distance function has be drawn only in the region* $Q_{3,k} = \mathcal{V}(3) \cap s_k$*. For the next slice* s_{k+1}*,* p_4 *is moved to* $\mathcal{S}^+(k+1)$*,* p_5 *is moved to* $\mathcal{I}^+(k+1)$ *and* p_3 *is moved to* $\mathcal{R}^+(k+1)$*.*

intersecting, swept and unswept subsets are represented as $\mathcal{I}^-(k)$, $\mathcal{S}^-(k)$, $\mathcal{U}^-(k)$, and are defined as

$$\begin{aligned}
&\mathcal{I}^-(k) = \{p_i \mid \mathcal{V}(i).z_{min} \leq z_k \leq \mathcal{V}(i).z_{max}\}\\
&\mathcal{S}^-(k) = \{p_i \mid p_i.z_{max} > z_k\}\\
&\mathcal{U}^-(k) = \{p_i \mid p_i.z_{max} \leq z_k\}
\end{aligned}$$

Consequently,

$$\mathcal{U}^+(k) = \mathcal{S}^-(k)\ ,\ \mathcal{I}^+(k) = \mathcal{I}^-(k) = \mathcal{I}(k)$$

and Eq. (2.1) reduces to

$$\mathcal{I}(k) = \mathcal{IS}^+(k) \cup \mathcal{IS}^-(k) \tag{2.2}$$

The key idea for speedup is that for a large number of sites m and any given slice s_k, the size of $\mathcal{I}(k)$ is typically much smaller than m. By computing a conservative estimate of $\mathcal{I}(k)$ one can cull away a large number of sites and considerably speed up the distance field computation.

2.7 Site Culling

In this section, we present our culling algorithm that reduces the number of distance functions that are rasterized for each slice. Our goal is to compute the distance field D_k for each slice s_k. Since only the set $\mathcal{I}(k)$ contributes to D_k, we have $D_k = D_k(\mathcal{I}(k))$. Using Eq. (2.2), D_k can be expressed as:

$$D_k(\mathcal{I}(k)) = D_k(\mathcal{IS}^+(k) \cup \mathcal{IS}^-(k)) = \min\left(D_k(\mathcal{IS}^+(k)), D_k(\mathcal{IS}^-(k))\right)$$

Therefore, the problem is reduced to computing two distance fields $D_k(\mathcal{IS}^+(k))$ and $D_k(\mathcal{IS}^-(k))$ for each slice s_k. We present an algorithm to compute $D_k(\mathcal{IS}^+(k))$ for s_k with a sweep direction along $+Z$. The same algorithm is used to compute $D_k(\mathcal{IS}^-(k))$ by using a sweep direction along $-Z$. In the rest of the paper, we will present our algorithm for $+Z$ sweep direction and drop the $+$ sign to simplify the notation.

We utilize the spatial coherence between successive slices and compute the intersecting swept set $\mathcal{IS}(k+1)$ by performing incremental computations on $\mathcal{IS}(k)$ (see Figure 2.5). We use the following formulation:

$$(\mathcal{IS}(k+1)) = (\mathcal{IS}(k) \cup (\mathcal{S}(k+1) \setminus \mathcal{S}(k))) \setminus (\mathcal{R}(k+1) \setminus \mathcal{R}(k)) \tag{2.3}$$

where $\setminus$ represents the set-difference operation. The exact computation of $\mathcal{IS}(k)$ and $\mathcal{IS}(k+1)$ is equivalent to exact Voronoi computation. Instead, we conservatively compute a set of *potentially intersecting swept* sites $\widehat{\mathcal{IS}}(k)$ using Equation (2.3), where $\widehat{\mathcal{IS}}(k) \supseteq \mathcal{IS}(k)$.

Given the sets $\widehat{\mathcal{IS}}(k)$ and $\mathcal{R}(k)$, the algorithm for computing D_{k+1}, $\widehat{\mathcal{IS}}(k+1)$ and $\mathcal{R}(k+1)$ proceeds as follows:

1. **Initialize** $\widehat{\mathcal{IS}}(k+1) = \widehat{\mathcal{IS}}(k)$, $D_{k+1} = \infty$.

2. **Update** $\widehat{\mathcal{IS}}(k+1) = \widehat{\mathcal{IS}}(k+1) \cup (\mathcal{S}(k+1) \setminus \mathcal{S}(k))$. Add the additional sites swept by slice s_{k+1} to $\widehat{\mathcal{IS}}(k+1)$.

3. **Compute** D_{k+1}. For each site $\hat{p}_i \in \widehat{\mathcal{IS}}(k+1)$, compute $D_{k+1}(\hat{p}_i)$ in order of increasing i. Each $D_k(\hat{p}_i)$ is tested for visibility with respect to $D_{k+1}(\mathcal{X}_{i-1})$, which is the distance field of set $\mathcal{X}_{i-1} = \{\hat{p_1}, \hat{p_2}, \ldots, \hat{p_{i-1}}\}$. If $D_{k+1}(\hat{p}_i)$ is not visible along the direction orthogonal to s_{k+1}, then it does not contribute to D_{k+1}.

4. **Compute** $(\mathcal{R}(k+1) \setminus \mathcal{R}(k))$. All sites $\hat{p}_i$ for which $D_{k+1}(\hat{p}_i)$ is not visible can be moved from $\widehat{\mathcal{IS}}(k+1)$ to $\mathcal{R}(k+1)$.

5. **Update** $\widehat{\mathcal{IS}}\,(k+1) = \widehat{\mathcal{IS}}\,(k+1) \setminus (\mathcal{R}\,(k+1) \setminus \mathcal{R}\,(k))$

Initially we set $k = 0, \mathcal{R}\,(k) = \emptyset, \widehat{\mathcal{IS}}\,(k) = \{p_i | p_i.z_{max} = 0\}$. We proceed along the Z-axis and compute the distance field for each slice as described above. Each site p_i is bucketed into a list according to $p_i.z_{max}$. This allows the addition of swept sites in Step (2) to be performed in constant time. The distance fields are rasterized approximately in order of increasing distance to the current slice. This results in better culling of the receding sites in Steps (3) and (4) of the algorithm. The complexity of this algorithm for slice s_{k+1} is a linear function of the size $|\ \widehat{\mathcal{IS}}\,(k+1)\ |$.

The visibility computations are performed using occlusion queries (e.g. GL_NV_occlusion_query) available on current graphics systems. As the distance meshes are scan converted, these queries check for updates to the depth buffer and return the number of pixels that are visible.

2.7.1 Conservative Sampling

The occlusion queries sample the visibility at fixed locations in each pixel and can result in sampling errors. In particular, the algorithm presented above classifies a swept site p_i as receding if its Voronoi region $\mathcal{V}(i)$ does not cover any grid cells, i.e. the occlusion query returns zero visible pixels for the distance field $D_k(p_i)$ in Step (3). This may introduce errors when $\mathcal{V}(i)$ intersects slice s_k but its intersection with s_k is not sampled by the rasterization hardware. An incorrect classification of p_i as receding can introduce errors in D_l for a subsequent slice s_l, $l > k$. One such case is shown in Figure 2.6(a), for $i = 2$.

We modify the algorithm for distance field computation to account for these sampling errors. The approach is based on a lemma that states the condition for a Voronoi region to be sampled.

Lemma 2.2. *Let $\mathcal{V}(p_i)$ be a voronoi region for a slice s_k that is undersampled, and the closest cell $\mathbf{q}$ is at a distance ϵ from $\mathcal{V}(i)$. If we reduce $d(\mathbf{q}, p_i)$ by ϵ without changing $d(\mathbf{q}, \mathcal{P} \setminus \{p_i\})$, we ensure that $\mathbf{q} \in \mathcal{V}(p_i)$.*

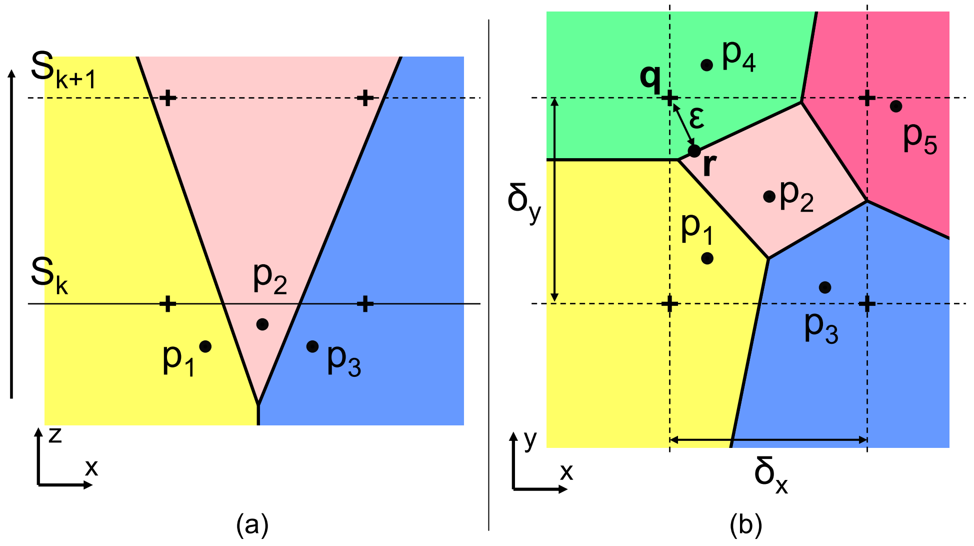


Figure 2.6: Sampling Error*: (a) The Voronoi region $\mathcal{V}(p_2)$ of a swept site p_2 does not lie on any cell (represented by crosses) on slice s_k, but lies on a cell for slice s_{k+1}. (b) The XY intersection of the Voronoi regions with slice s_k. The closest cell $\mathbf{q}$ to $\mathcal{V}(p_2)$ is at a distance ϵ.*

Proof. Let $\mathbf{q}$ belong to the voronoi region $\mathcal{V}(p_j)$ of site p_j, and the point in $\mathcal{V}(p_i)$ closest to $\mathbf{q}$ be $\mathbf{r}$ (see Figure 2.6, with $i = 2, j = 4$, $d(\mathbf{q})\mathbf{r} = \epsilon$). We shall first prove the result for the case when $\mathcal{V}(i)$ shares a boundary with $\mathcal{V}(j)$. Then using the fact $\mathbf{r} \in \mathcal{V}(p_i)$ and $\mathbf{r} \in \mathcal{V}(p_j)$, and the triangle inequality, we have

$$d(\mathbf{r}, p_j) = d(\mathbf{r}, p_i)$$

$$d(\mathbf{q}, p_i) \leq d(\mathbf{q}, \mathbf{r}) + d(\mathbf{r}, p_i)$$

$$d(\mathbf{r}, p_j) \leq d(\mathbf{r}, \mathbf{q}) + d(\mathbf{q}, p_j)$$

$$\Rightarrow d(\mathbf{q}, p_i) - \epsilon \leq d(\mathbf{r}, p_j) < d(\mathbf{q}, p_j)$$

Thus, by reducing $d(\mathbf{q}, p_i)$ by ϵ and keeping $d(\mathbf{q}, p_j)$ the same, $\mathbf{q}$ will lie in Voronoi region $\mathcal{V}(p_i)$. This directly extends to the case when $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$ do not share a boundary, by using a sequence of triangle inequalities across Voronoi boundaries between $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$. $\square$

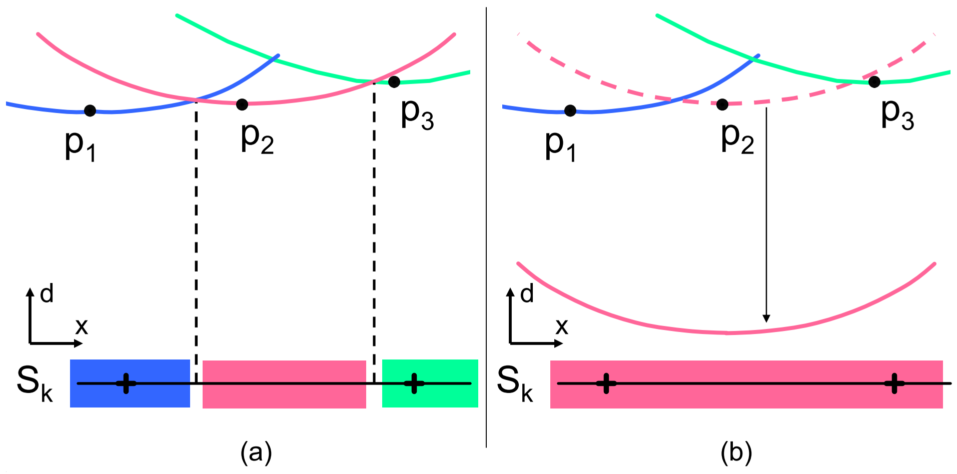


Figure 2.7: Conservative Sampling*: (a) Distance field $D_k(p_2)$ of site p_2 is occluded at all pixels on s_k. (b) Translating $D_k(p_2)$ by δ_{xy} ensures it is visible at at least one pixel.*

We apply the result of Lemma 2.2 in the following manner. In practice, we do not know the point $\mathbf{q}$ or ϵ but use the fact that ϵ is bounded by pixel size, $\epsilon \leq \delta_{xy} = \frac{\sqrt{\delta_x^2+\delta_y^2}}{2}$. Therefore, we move p_i closer to all the points in slice s_k, by subtracting δ_{xy} from each value of the distance field $D_k(p_i)$. This is equivalent to translating $D_k(p_i)$ along $-Z$ by δ_{xy} and is shown in Figure 2.7.

Given a slice s_{k+1}, we redraw the translated distance field of each site p_i marked as receding in Step (4) of the algorithm given above (i.e. $p_i \in \mathcal{R}\,(k+1) \setminus \mathcal{R}\,(k)$). The redrawn distance field is tested for visibility with respect to D_{k+1}. This redrawing is performed to ensure conservative sampling for site-culling. During this step, updates to the final distance field in the depth buffer are disabled. Moreover, the translated field is clamped to 0 for negative values. For line and triangle sites, the size of the Voronoi region is also limited by the spatial size of the line segment or the triangle. To ensure that the Voronoi region covers at least one of the four neighboring cells, we increase the size of these sites by δ_{xy} in addition to translating the distance field.

2.8 Distance Function Clamping

In Section 2.7, we presented an algorithm to cull away the sites that do not contribute to the distance field D_k of slice s_k. In this section, we present a clamping algorithm to reduce the rasterization cost of the distance function of each potentially intersecting swept site. Given a slice s_k and each site $p_i \in \widehat{\mathcal{IS}}\,(k)$, we compute the distance function $d(\mathbf{q}, p_i)$ only for the set of points on s_k that lie in the Voronoi region of p_i. In other words, our goal is to evaluate the distance function for the set $Q_{i,k} = \{\mathbf{q}|\mathbf{q} \in \mathcal{V}(i) \cap s_k\}$. We first present an approach to compute a conservative estimate $\widehat{Q}_{i,k}$ of $Q_{i,k}$ for any arbitrary set of sites. We further improve the performance of the clamping algorithm for manifold surfaces by using domain bounds from Section 2.4.

2.8.1 Conservative Clamping

The connectivity of the Voronoi regions implies that $Q_{i,k}$ is a connected set. We exploit the monotonicity property and compute a superset $\widehat{Q}_{i,k}$. Initially, we assume that we are given the maximum value $\max(D_k(p_i))$ of the distance field $D_k(p_i)$ of site p_i on slice s_k. We compute a set of extreme points on s_k where the value of the distance field $D_k(p_i)$ is equal to the maximum value. By the monotonicity property of distance functions, the set of points whose distance function is less than or equal to $\max(D_k(p_i))$ belong to $\widehat{Q}_{i,k}$. An example is shown in Figure 2.8.

The problem of distance function clamping reduces to computing $\max(D_k(p_i))$ for each site p_i in $\widehat{\mathcal{IS}}\,(k)$ for a slice s_k. We use the following lemma to compute an upper bound on $\max(D_k(p_i))$.

Lemma 2.3. *Let* $\max(D_k(\mathcal{S}\,(k)))$ *denote the maximum value of the distance fields* $D_k(\mathcal{S}\,(k))$ *of set* $\mathcal{S}\,(k)$ *on a slice* s_k *and* $\max(D_{k+1}(\mathcal{S}\,(k+1)))$ *be defined similarly. Let the distance*

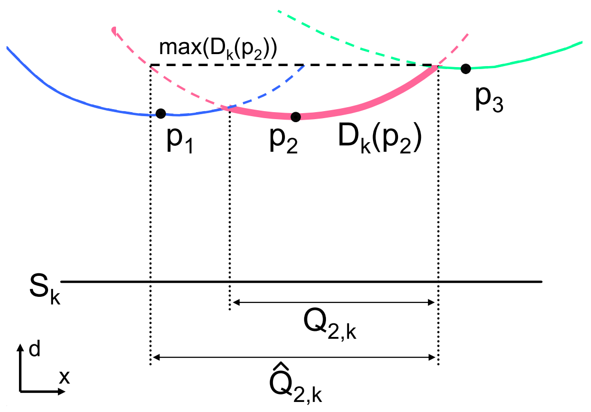


Figure 2.8: Clamping distance field computation to Voronoi region bounds on a slice. $Q_{2,k} = \mathcal{V}(i) \cup s_k$. $\widehat{Q}_{2,k} \supseteq Q_{2,k}$ *and is computed from* $\max(D_k(p_2))$.

between s_{k+1} *and* s_k *be* $|z_{k+1} - z_k| = \delta_z$. *Then*

$$\max(D_{k+1}(\mathcal{S}\ (k+1))) \leq \max(D_k(\mathcal{S}\ (k))) + \delta_z \tag{2.4}$$

Proof. Given two points $\mathbf{q}_k(x, y, z_k) \in s_k$ and $\mathbf{q}_{k+1}(x, y, z_{k+1}) \in s_{k+1}$ that lie in the Voronoi regions of some two sites. Then

$$|d(\mathbf{q}_{k+1}, \mathcal{P}) - d(\mathbf{q}_k, \mathcal{P})| \leq \delta_z. \tag{2.5}$$

This follows directly from the triangle inequality, and the definition of the distance function $d(\mathbf{q}, \mathcal{P})$. Moreover, $\max(D_k(\mathcal{X})) = \max_{\mathbf{q} \in s_k}(d(\mathbf{q}, \mathcal{X}))$. This implies that

$$\max(D_{k+1}(\mathcal{X})) \leq \max(D_k(\mathcal{X})) + \delta_z \tag{2.6}$$

Moreover, for a slice s_k and any two sets of sites $\mathcal{X}_\infty$ and $\mathcal{X}_\in$, $\mathcal{X}_\infty \subseteq \mathcal{X}_\in \Rightarrow D_k(\mathcal{X}_\in) \leq D_k(\mathcal{X}_\infty)$. We know $\mathcal{S}\ (k) \subseteq \mathcal{S}\ (k+1)$. This combined with Eq. (2.6), where $\mathcal{X} = \mathcal{S}\ (k+1)$, leads to the result in Eq. (2.4). $\square$

Given the maximum value $\max(D_k(\mathcal{S}\ (k)))$ of the distance field for slice s_k, we use

Eq. (2.4) to compute the maximum value $\max(D_{k+1}(\mathcal{S}\ (k+1)))$ of the distance field for slice s_{k+1}. This also gives a conservative bound on maximum value of the distance function for each site p_i on slice s_{k+1}, $\max(D_{k+1}(\mathcal{S}\ (k+1))) \geq \max(D_{k+1}(p_i))\ \forall\ p_i \in \mathcal{S}\ (k+1)$. We use it to compute a conservative bound on the set of points $Q_{i,k+1}$ on slice s_{k+1} and use this bound for clamping.

Note that the maximum distance value, $\max(D_k(\mathcal{S}\ (k)))$, may be infinity, if one is computing the distance field in a narrow band at a distance d_{max} [Mau03], or if one is computing the signed distance field for a closed manifold. For the first case we define $\max(D_k(\mathcal{S}\ (k)))$ to be the maximum *finite* value of the distance field, and set the update rule to be

$$\max(D_{k+1}(\mathcal{S}\ (k+1))) = \min(d_{max}, \max(D_k(\mathcal{S}\ (k))) + \delta_z)$$

For the second case, if $\mathbf{q}_k$ does not lie in region where the signed distance field is computed, and $\mathbf{q}_{k+1}$ does, then the manifold surface lies between $\mathbf{q}_k$ and $\mathbf{q}_{k+1}$ and

$$\max(d(\mathbf{q}_{k+1}, \mathcal{S}\ (k+1))) \leq \delta_z$$

. This is shown in Figure 2.9.

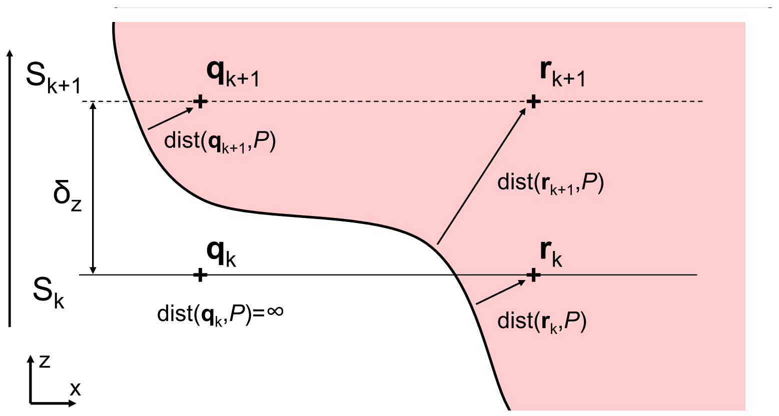


Figure 2.9: Change in distance field for signed distance computation.

2.8.2 Manifold Surfaces

In many cases, the primitives lie on manifold surfaces and we have the connectivity information In these cases, we also use the domain bounds presented in Section 2.4 to further refine $\widehat{Q}_{i,k+1}$ for signed Euclidean distance fields. For each site in the interior of a manifold surface, we compute a polyhedron bounding its Voronoi region. This polyhedron is intersected with s_{k+1} to compute a convex polygon $G_{i,k+1}$. In this case, $G_{i,k+1} \cap \widehat{Q}_{i,k+1}$ results in a tighter approximation of $Q_{i,k+1}$. Sites on the boundary of a manifold surface are handled similar to non-manifold sites.

2.8.3 Complete Algorithm

Given $\widehat{\mathcal{IS}}(k)$, $\mathcal{R}(k)$ and D_k, the algorithm for computing D_{k+1} as presented in Section 2.7 is refined to perform clamping as follows:

1. **Compute** $\max(D_k)$ by using multiple occlusion queries as described in [GLW+04]. Compute $\max(D_{k+1}) = \min(d_{max}, \max(D_k) + \delta)$.

2. **Initialize** $\widehat{\mathcal{IS}}(k+1) = \widehat{\mathcal{IS}}(k)\,, D_{k+1} = \infty$.

3. **Update** $\widehat{\mathcal{IS}}(k+1) = \widehat{\mathcal{IS}}(k+1) \cup (\mathcal{S}(k+1) \setminus \mathcal{S}(k))$.

4.1. **Compute** $\widehat{Q}_{i,k+1}$. For each site $p_i \in \widehat{\mathcal{IS}}(k+1)$, compute $\widehat{Q}_{i,k+1}$ from $\max(D_{k+1})$

4.2. **Refine** $\widehat{Q}_{i,k+1}$. For each *CSC-valid* site $p_i \in \widehat{\mathcal{IS}}(k+1)$, compute the convex polygon $G_{i,k+1}$. Refine $\widehat{Q}_{i,k+1} = \widehat{Q}_{i,k+1} \cap G_{i,k+1}$.

4.3. **Compute** D_{k+1}. For each site $p_i \in \widehat{\mathcal{IS}}(k+1)$, compute $D_{\widehat{Q}_{i,k+1}}(p_i)$ and test for visibility as before.

4.4. **Perform Conservative Sampling** Disable distance field updates. For each site $p_i \in \widehat{\mathcal{IS}}(k+1)$ which is marked as occluded, expand the site and compute $D_{\widehat{Q}_{i,k+1}}(p_i) -$

δ_{xy}. Test for visibility against the computed distance field D_{k+1} as before. Enable distance field updates.

5. **Compute** $(\mathcal{R}(k+1) \setminus \mathcal{R}(k))$ from the results of the visibility tests of Step 4.4.

6. **Update** $\widehat{\mathcal{IS}}(k+1) = \widehat{\mathcal{IS}}(k+1) \setminus (\mathcal{R}(k+1) \setminus \mathcal{R}(k))$.

Given a 3D grid with $k+1$ slices and a Z range $[z_{min}, z_{max}]$, we make 2 passes. In the first pass, we increment k from 0 to k. Initially, $\mathcal{R}^{+}(0) = \emptyset, \widehat{\mathcal{IS}}^{+}(0) = \{p_i | p_i.z_{max} = z_{min}\}$. In the second pass, k is decremented from k down to 0. Initially, $\mathcal{R}^{-}(k) = \emptyset, \widehat{\mathcal{IS}}^{-}(k) = \{p_i | p_i.z_{max} = z_{max}\}$. The final distance field for each slice is the lower envelope of both passes.

2.9 Range Based Culling

The algorithm presented in section 2.8 performs culling along a single spatial dimension only. In addition, the bound computed using clamping is a global bound per slice, and may be too conservative. In this section we extend our algorithm to perform better culling within an n-dimensional range.

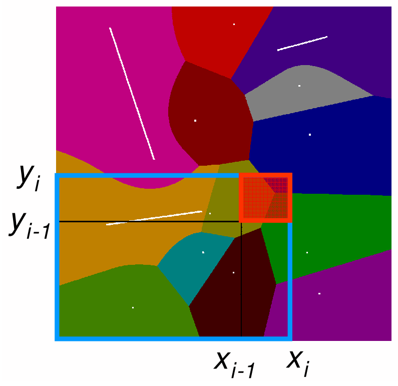


Figure 2.10: Ranges in 2D*: The range T_{ij} is shown using a filled red rectangle. The range $T_{i^+j^+}$ is shown using thick blue borders.*

Our culling algorithm performs two sweeps in each dimension to obtain conservative bounds, along each dimension, of the Voronoi region of a site. The conservative bounds are used to reduce the set of points in the domain at which the distance function of a given site needs to be evaluated. We use the connectivity property and range-based sweeps to compute bounds of a Voronoi region (see figure 2.12). We first modify the definition of Intersection, Receding and Swept sets. We shall our range-based culling algorithm in 2D and later extend it to higher dimensions.

2.9.1 Set Definitions

We introduce the classification of sites used by our algorithm to cull away sites that do not contribute to the distance field of a given range (see figure 2.11). Using the pivot point of the sites, the *swept set* for a range T is defined as

$$\mathcal{S}\,(T) = \{p_i \mid \kappa(p_i) \in T, p_i \in \mathcal{P}\}.$$

Using bounds on the Voronoi regions of a site p_i, the *intersecting set* of a range T is defined as

$$\mathcal{I}\,(T) = \{p_i \mid \mathcal{V}(p_i) \cap T \neq \emptyset, p_i \in \mathcal{P}\}.$$

Thus for each point inside a range T, we have to compute the distance values to all sites in the intersecting set $\mathcal{I}\,(T)$. The *intersecting swept set* for two ranges T_1, T_2 is defined as

$$\mathcal{IS}\,(T_1, T_2) = \mathcal{I}\,(T_1) \cap \mathcal{S}\,(T_2)$$

The intersecting swept set represents the set of sites, which are swept by the second range and their Voronoi regions intersect the first range. Note that the definition is not symmetric. The receding set for a range T is the set of sites with Voronoi regions contained entirely inside

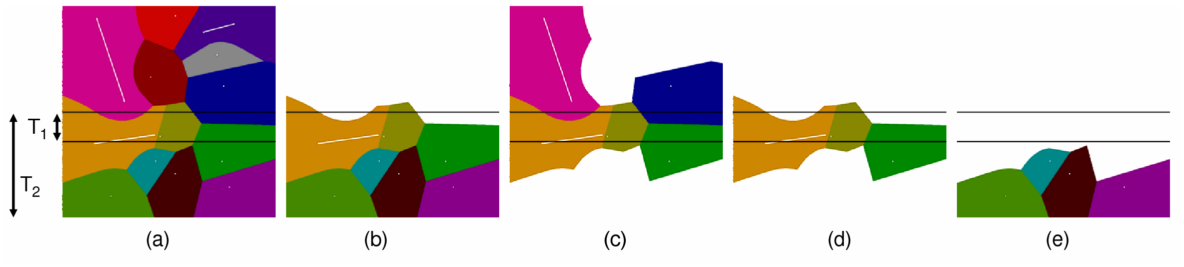


Figure 2.11: Set Definitions*: (a) Voronoi diagram of 10 points and 3 lines and two ranges* T_1 *and* T_2*. (b) Swept set* $\mathcal{S}(T_2)$ *(c) Intersecting Set* $\mathcal{I}(T_1)$ *(d) Intersecting Swept set* $\mathcal{IS}(T_1, T_2) = \mathcal{I}(T_1) \cap \mathcal{S}(T_2)$ *(e) Receding set* $\mathcal{R}(T_2 \setminus T_1) = \mathcal{S}(T_2) \setminus \mathcal{I}(T_1)$

T, and is defined as

$$\mathcal{R}(T) = \{p_i \mid \mathcal{V}(p_i) \subset \text{Int}(T), p_i \in \mathcal{P}\}.$$

For two ranges T_i and T_j, if $T_i \subseteq T_j$ then $\mathcal{R}(T_i) \subseteq \mathcal{R}(T_j)$. By computing a receding set for a given range T, we can cull away the sites belonging to the receding set while computing the Voronoi diagram of its complement T^c. Our range based culling algorithm partitions $\mathcal{D}$ into a set of ranges, and computes the Voronoi diagram constrained to each range by computing a superset of the intersecting set for each range. The computation of a subset of the receding set is used to compute conservative estimate for each intersecting set.

2.9.2 2D Culling

In 2D, $\mathcal{D} = (0, 1] \times (0, 1]$ and we have point, line and polygonal sites. The domain is partitioned into a set of rectangular ranges, called *tiles*. Our culling algorithm performs two sweeps along each dimension and incrementally culls away a subset of sites that do not belong to the intersecting swept set of the current tile. Next, we define the tiles, decompose of the Voronoi diagram computation into four sweeps as each tile decomposes the domain into 4 quadrants. Finally we present the update rule for incrementally computing the intersecting swept set in one sweep.

Given a set of $l + k + 2$ real numbers $x_0, x_1, \ldots, x_l, y_0, y_1, \ldots, y_k$ s.t. $x_0 = y_0 = 0, x_l =$

$y_k = 1, x_i \in (0,1], y_j \in (0,1], x_i \geq x_{i-1}, y_j \geq y_{j-1}, 1 \leq i \leq l, 1 \leq j \leq k$. These $l+k+2$ points partition $\mathcal{D}$ into $l \times k$ ranges with $T_{ij} = T_{(x_{i-1},x_i](y_{j-1},y_j]}$. Define the ranges $T_{i^+j^+} = (0,x_i] \times (0,y_j]$, $T_{i^-j^+} = (x_i,1) \times (0,y_j]$, $T_{i^+j^-} = (0,x_i] \times (y_j,1)$, and $T_{i^-j^-} = (x_i,1) \times (y_j,1)$ (see figure 2.10). We use the following lemma to compute the Voronoi diagram within the range T_{ij} using the intersecting swept sets.

Lemma 2.4. *Given* $l \times k$ *disjoint ranges which partition* $(0,1]^2$*,*

$$\begin{aligned}\mathrm{VD}^{T_{ij}}(\mathcal{P}) = \mathrm{VD}^{T_{ij}}(&\mathcal{IS}\,(T_{ij}, T_{i^+j^+}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^+}) \cup \\ &\mathcal{IS}\,(T_{ij}, T_{i^+j^-}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^-}))\end{aligned}$$

Proof. By definition, $\mathrm{VD}^{T_{ij}}(\mathcal{P}) = \mathrm{VD}^{T_{ij}}(\mathcal{I}\,(T_{ij}))$. Also,

$$\begin{aligned}\mathcal{I}\,(T_{ij}) =&\mathcal{I}\,(T_{ij}) \cap \mathcal{P} \\ =&\mathcal{I}\,(T_{ij}) \cap (\mathcal{S}\,(T_{i^+j^+}) \cup \mathcal{S}\,(T_{i^-j^+}) \cup \\ &\mathcal{S}\,(T_{i^+j^-}) \cup \mathcal{S}\,(T_{i^-j^-})) \\ =&\mathcal{IS}\,(T_{ij}, T_{i^+j^+}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^+}) \cup \\ &\mathcal{IS}\,(T_{ij}, T_{i^+j^-}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^-})\end{aligned}$$

Thus,

$$\begin{aligned}\mathrm{VD}^{T_{ij}}(\mathcal{P}) = \mathrm{VD}^{T_{ij}}(&\mathcal{IS}\,(T_{ij}, T_{i^+j^+}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^+}) \cup \\ &\mathcal{IS}\,(T_{ij}, T_{i^+j^-}) \cup \mathcal{IS}\,(T_{ij}, T_{i^-j^-}))\end{aligned}$$

□

As a result of Lemma 2.4, we compute the Voronoi diagram $\mathrm{VD}(T_{ij})$ by computing four intersecting swept sets. We perform two passes along each axis, sweeping from 0 to 1 and

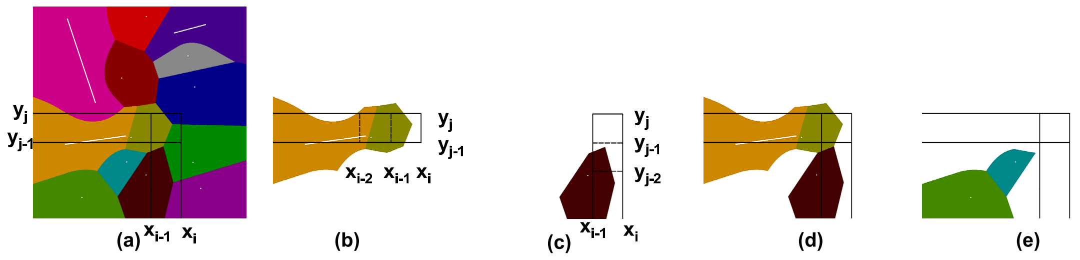


Figure 2.12: PIS Computation in 2D*: This image highlights the Voronoi computation in a 2D range $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ based on the sweep along the $+X$ and $+Y$ axes. Fig. 2.12(a) shows the 2D Voronoi diagram of a set of points and lines and the 2D range. In Fig. 2.12(b), we highlight the PIS for the range $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ computed by sweeping along the +X direction. Note that the PIS is conservatively computed as the union of PIS for the range $(x_{i-2}, x_{i-1}] \times (y_{j-1}, y_j]$ and the set of sites that intersect the range $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$. Similarly, in Fig. 2.12(c), we show the computation of PIS for the range $(x_{i-1}, x_i] \times (y_{j-1}, y_j]$ computed using a sweep along the +Y axis. The PIS for the sweep along both $+X$ and $+Y$ directions is shown in Fig. 2.12(d). The receding set is highlighted in Fig. 2.12(e). Based on the connectivity property of Voronoi diagrams, the sites in the receding set are ignored in the Voronoi diagram computation for any range beyond $(0, x_i]$ in the $+X$ direction and beyond $(0, y_j]$ in the $+Y$ direction.*

then sweeping from 1 to 0, and compute the intersecting swept sets. Our approach for computing the intersecting swept sets for three other ranges (i^-j^+, i^+j^-, i^-j^-) is similar to the approach for computing the intersecting swept set $\mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right)$. However, the computation of exact intersecting swept set is equivalent to computing the exact Voronoi diagram. Instead, we present a simple theorem to efficiently compute a superset of the intersecting swept set. This conservative computation does not affect correctness of the algorithm, but influences the level of culling achieved for each range.

Theorem 2.2. *A superset of the intersecting swept set $\mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right)$ is given by the relation*

$$\begin{aligned} &\mathcal{IS}\left(T_{ij}, T_{i^+j^+}\right) \\ &\subseteq \ \mathcal{IS}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right) \cup \mathcal{IS}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right) \cup \\ &\quad\ \mathcal{S}\left(T_{ij}\right) \end{aligned} \tag{2.7}$$

Proof. Let $\mathcal{X}$ denote the l.h.s of eq (2.7) and $\mathcal{Y}$ denote the r.h.s of eq (2.7). Let $p_a \in \mathcal{X} \Rightarrow$

$\mathcal{V}(p_a) \cap T_{ij} \neq \emptyset$ and $\kappa(p_a) \in T_{i^+j^+}$. We have two cases.

1. $\kappa(p_a) \in T_{ij} \Rightarrow p_a \in \mathcal{S}(T_{ij}) \Rightarrow p_a \in \mathcal{Y}$.

2. $\kappa(p_a) \in T_{i^+j^+} \setminus T_{ij} \Rightarrow \mathcal{V}(p_a) \cap (T_{i^+j^+} \setminus T_{ij}) \neq \emptyset$. Since $\mathcal{V}(a)$ is connected, $\mathcal{V}(p_a) \cap (T_{i^+j^+} \setminus \partial T_{ij}) \neq \emptyset$. This implies either $\mathcal{V}(p_a) \cap (x_{i-1} \times (y_{j-1}, y_j] \neq \emptyset$ or $\mathcal{V}(p_a) \cap (x_{i-1} \times (y_{j-1}, y_j] \neq \emptyset$. Hence $p_a \in \mathcal{IS}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right) \cup \mathcal{IS}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right) \Rightarrow p_a \in \mathcal{Y}$.

□

Similarly, we can conservatively compute $\mathcal{IS}(T_{ij}, T_{i^-j^+})$, $\mathcal{IS}(T_{ij}, T_{i^+j^-})$, and $\mathcal{IS}(T_{ij}, T_{i^-j^-})$. Theorem 2.2 indicates that the voronoi diagram $\mathrm{VD}^{T_{ij}}(\mathcal{P})$ can be computed incrementally within each range $(i^+j^+, i^+j^-, i^-j^+, i^-j^-)$. For example, in the range i^+j^+, both $\mathcal{IS}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right)$ and $\mathcal{IS}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right)$ have already been computed before the sweep reaches T_{ij} and these sets are then used for *incrementally* computing $\mathcal{IS}(T_{ij}, T_{i^+j^+})$. The swept set $\mathcal{S}(T_{ij})$ is easily computed by binning the sites into ranges using the pivot points. Fig. 2.12 highlights the incremental computation of the $\mathrm{VD}^{T_{ij}}(\mathcal{P})$ using sweep along +X and +Y directions.

Corollary 2.1. *Let a site* $p_a \in \mathcal{R}\left(T_{(i-1)^+(j-1)^+}\right)$. *Then* $p_a \notin \mathcal{IS}(T_{ij}, T_{i^+j^+})$.

Proof. $p_a \in \mathcal{R}\left(T_{(i-1)^+(j-1)^+}\right)$
$\Rightarrow \mathcal{V}(p_a) \subset \mathrm{Int}(T_{(i-1)^+(j-1)^+}) \Rightarrow \kappa(p_a) \in T_{(i-1)^+(j-1)^+}$
$\Rightarrow p_a \notin \mathcal{S}(T_{ij})$. Also $\mathcal{V}(p_a) \cap \partial T_{(i-1)^+(j-1)^+} = \emptyset$ and by connectivity of Voronoi regions, $\mathcal{V}(p_a) \cap \mathcal{I}\left(T_{(i-1)j}\right) = \emptyset, \mathcal{V}(p_a) \cap \mathcal{I}\left(T_{i(j-1)}\right) = \emptyset$. Using the result of theorem 2.2, $p_a \notin \mathcal{IS}(T_{ij}, T_{i^+j^+})$. □

A direct consequence of Corollary 2.1 is that one can check if a site belongs to the receding set of range $T_{i^+j^+}$ and cull it for Voronoi diagram computation in $T^c_{i^+j^+}$. Furthermore, in the three other passes, let $p_a \in \mathcal{R}(T_{k^-l^+}), \mathcal{R}(T_{m^+n^-}), \mathcal{R}(T_{p^-q^-})$. Then $\mathcal{V}(p_a) \subset (\min(x_k, x_p), \max(x_i, x_m)] \times (\min(y_n, y_q), \max(y_j, y_l)]$, giving us spatial bounds on $\mathcal{V}(p_a)$.

2.9.3 Culling in 3D and Higher Dimensions

Our approach for range-based culling extends directly to higher dimensions. In n-D, let $\mathcal{D} = (0,1]^n$. As in section 2.9.2, let there be k_i ranges along each dimension, giving a total of $\prod_{i=1}^{n} k_i$ ranges. Let range $T_{i_1 i_2 \ldots i_n} = (x_{i_1-1}, x_{i_1}] \times (x_{i_2-1}, x_{i_2}] \times \ldots \times (x_{i_n-1}, x_{i_n}]$, where $1 \leq i_j \leq k_j \; \forall \; 1 \leq j \leq n$, and x_{i_k} is the $i^t h$ coordinate in $k^t h$ dimension. Also, $T_{i_1^+ i_2^+ \ldots i_n^+} = (0, x_{i_1}] \times (0, x_{i_2}] \times \ldots \times (0, x_{i_n}]$, and the symmetric ranges along other sweep directions are defined similarly. In particular, range $T_{i_1 i_2 \ldots i_n}$ partitions $\mathcal{D}$ into 2^n swept ranges. Thus the intersecting set $\mathcal{I}\left(T_{i_1 i_2 \ldots i_n}\right)$ is partitioned into 2^n intersecting swept sets. We present a theorem that is used to compute a superset of the intersecting swept set:

Theorem 2.3. *A superset of the intersecting swept set* $\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$ *is given by the relation*

$$\begin{aligned}
&\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right) \\
&\subseteq \; \mathcal{IS}\left(T_{(i_1-1) i_2 \ldots i_n}, T_{(i_1-1)^+ i_2^+ \ldots i_n^+}\right) \cup \\
&\qquad \mathcal{IS}\left(T_{i_1 (i_2-1) \ldots i_n}, T_{i_1^+ (i_2-1)^+ \ldots i_n^+}\right) \cup \\
&\qquad \ldots \\
&\qquad \mathcal{IS}\left(T_{i_1 i_2 \ldots (i_n-1)}, T_{i_1^+ i_2^+ \ldots (i_n-1)^+}\right) \cup \\
&\qquad \mathcal{S}\left(T_{i_1 i_2 \ldots i_n}\right)
\end{aligned} \tag{2.8}$$

The proof is similar to that of Theorem 2.2 and uses the connectivity property to ensure that any Voronoi region intersecting the range $T_{i_1 i_2 \ldots i_n}$ must intersect one of its adjacent ranges, or the site must lie inside the range $T_{i_1 i_2 \ldots i_n}$. The following corollary gives a similar relation between the receding set $\mathcal{R}\left(T_{(i_1-1)^+ (i_2-1)^+ \ldots (i_n-1)^+}\right)$ and the intersecting swept set $\mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$.

Corollary 2.2. *Let a site* $p_a \in \mathcal{R}\left(T_{(i_1-1)^+ (i_2-1)^+ \ldots (i_n-1)^+}\right)$*. Then* $p_a \notin \mathcal{IS}\left(T_{i_1 i_2 \ldots i_n}, T_{i_1^+ i_2^+ \ldots i_n^+}\right)$.

As in 2D, Corollary 2.2 provides conservative bounds on the spatial bounds of the Voronoi region of a site.

2.10 GPU Based Algorithm

In this section, we present our algorithm which uses the graphics hardware to efficiently compute the discrete generalized Voronoi diagram. Computation of the exact intersecting swept set $\mathcal{IS}\,(T_1, T_2)$ is equivalent to exact Voronoi computation. Instead we compute a set of *potentially intersecting swept* (PIS) sites, $\widehat{\mathcal{IS}}\,(T_1, T_{1'})$ which is a superset of the intersecting swept set $\mathcal{IS}\,(T_1, T_{1'})$. We use Corollary 2.1 to check if a site belongs to the receding set and use it to cull receding sites from the potentially intersecting swept set. To check for the membership in the receding set, we maintain conservative bounds $\widehat{\mathcal{V}}(p_a)$ on the Voronoi region $\mathcal{V}(p_a)$ of each site p_a, where $\widehat{\mathcal{V}}(p_a) \supseteq \mathcal{V}(p_a)$. The bounds are maintained at the resolution of a range T, i.e. $T \subseteq \widehat{\mathcal{V}}(p_a)$ if $\mathcal{V}(p_a) \cap T \neq \emptyset$. The key operation is to test if a Voronoi region $\mathcal{V}(p_a)$ intersects a given range T. A Voronoi region $\mathcal{V}(p_a)$ intersects a given range T if and only if the distance field of the site p_a $D_T(p_a)$ contributes to the final distance field $D_T(\mathcal{P})$. This computation is performed by testing the distance field $D_T(p_a)$ for visibility. The visibility computations are performed using occlusion queries (e.g. GL_NV_occlusion_query) available on current graphics systems. As the distance values are written to the depth buffer, these queries check for updates to the depth buffer and return the number of pixels that are visible.

We first describe the algorithm for computing 2D discrete Voronoi diagrams and then extend it to 3D discrete Voronoi diagrams.

2.10.1 2D Culling

In 2D, the domain is divided into $k \times l$ rectangular ranges, each range called a tile. All tiles with the same X limits form a row. The Voronoi diagram for the domain is computed by performing two sweeps across all rows. Within a row, we perform two sweeps across all tiles and compute the Voronoi diagram for the tile. The algorithm for computing the Voronoi diagram for the domain is given in Algorithm 3.

The function ComputeTile(T_{ij},$\widehat{\mathcal{IS}}\,(T_1, T_{1'})$ $\widehat{\mathcal{IS}}\,(T_2, T_{2'})$) computes the Voronoi diagram in the range T_{ij} using our incremental culling algorithm, where T_1, T_2 are adjacent to T_{ij}, and $T_{1'}$, $T_{2'}$ are the corresponding swept sets. It returns the updated potential intersecting swept set $\widehat{\mathcal{IS}}\,(T_{ij}, T_{i'j'})$ for T_{ij}. The details are given in Algorithm 4.

Based on Corollary 2.1, we need to check if the Voronoi region $\widehat{\mathcal{V}}(p_a)$ is a subset of the *interior* of the range T_{ij}, or equivalently if $\widehat{\mathcal{V}}(p_a)$ does not intersect the boundary of T_{ij}. The intersection test is performed with the entire range T_{ij} using visibility queries. To test if $\widehat{\mathcal{V}}(p_a)$ intersects the boundary of T_{ij}, we compute the intersection with the adjacent ranges $T_{(i+1)j}$, $T_{(i-1)j}$, $T_{i(j-1)}$, $T_{i(j+1)}$. The function *UpdateBounds*(p_a,T_{ij}) in Algorithm 4 updates the Voronoi region bound $\widehat{\mathcal{V}}(p_a)$ by adding T_{ij}. Thus *UpdateBounds*(p_a,T_{ij}) adds the adjacent ranges to the Voronoi region bounds $\mathcal{V}(p_a)$.

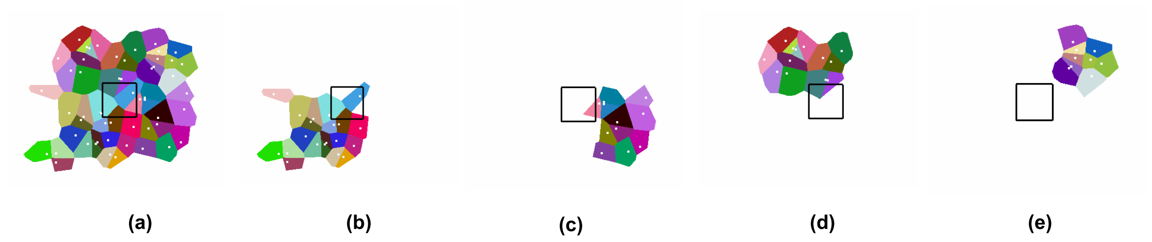


Figure 2.13: GPU Based PIS Computation in 2D*: This image highlights the PIS sets and the Voronoi regions computed for the tile T_{ij} shown in black during each of the four sweeps in 2D (a) The final potential intersecting set $\mathcal{I}\,(T_{ij})$ (b) PIS $\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^+j^+})$ (c) PIS $\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^-j^+})$ (d) PIS $\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^+j^-})$ (e) PIS $\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^-j^-})$*

```
    **Input**: Domain $\mathcal{D}$, site set $\mathcal{P}$, num tiles $k, l$
    **Output**: Voronoi Diagram $\mathrm{VD}^{\mathcal{D}}(\mathcal{P})$
1  **foreach** *site* $p_a \in \mathcal{P}$ **do**
2      Find tile $T_{ij}$ s.t. $\kappa(p_a) \in T_{ij}$
3      Initialize $\widehat{\mathcal{V}}(p_a) \leftarrow T_{ij}$
4  **end**
5  **for** *j=1* to *l* **do**
6      **for** *i=1* to *k* **do**
7          $\left(\mathrm{VD}^{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}\left(T_{ij}, T_{i^+j^+}\right)\right) \leftarrow$
           ComputeTile$(T_{ij}, \widehat{\mathcal{IS}}\left(T_{(i-1)j}, T_{(i-1)^+j^+}\right),$
8          $\widehat{\mathcal{IS}}\left(T_{i(j-1)}, T_{i^+(j-1)^+}\right))$
9          $\mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \leftarrow \mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \cup \mathrm{VD}^{T_{ij}}(\mathcal{P})$
10     **end**
11     **for** *i = k* downto *1* **do**
12         $\left(\mathrm{VD}^{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}\left(T_{ij}, T_{i^-j^+}\right)\right) \leftarrow$
           ComputeTile$(T_{ij}, \widehat{\mathcal{IS}}\left(T_{(i+1)j}, T_{(i+1)^-j^+}\right),$
13         $\widehat{\mathcal{IS}}\left(T_{i(j-1)}, T_{i^-(j-1)^+}\right))$
14         $\mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \leftarrow \mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \cup \mathrm{VD}^{T_{ij}}(\mathcal{P})$
15     **end**
16 **end**
17 **for** *j=l* downto *1* **do**
18     **for** *i=1* to *k* **do**
19         $\left(\mathrm{VD}^{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}\left(T_{ij}, T_{i^+j^-}\right)\right) \leftarrow$
           ComputeTile$(T_{ij}, \widehat{\mathcal{IS}}\left(T_{(i-1)j}, T_{(i-1)^+j^-}\right),$
20         $\widehat{\mathcal{IS}}\left(T_{i(j+1)}, T_{i^+(j+1)^-}\right))$
21         $\mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \leftarrow \mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \cup \mathrm{VD}^{T_{ij}}(\mathcal{P})$
22     **end**
23     **for** *i = k* downto *1* **do**
24         $\left(\mathrm{VD}^{T_{ij}}(\mathcal{P}), \widehat{\mathcal{IS}}\left(T_{ij}, T_{i^-j^-}\right)\right) \leftarrow$
           ComputeTile$(T_{ij}, \widehat{\mathcal{IS}}\left(T_{(i+1)j}, T_{(i+1)^-j^-}\right),$
25         $\widehat{\mathcal{IS}}\left(T_{i(j+1)}, T_{i^-(j+1)^-}\right))$
26         $\mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \leftarrow \mathrm{VD}^{\mathcal{D}}(\mathcal{P}) \cup \mathrm{VD}^{T_{ij}}(\mathcal{P})$
27     **end**
28 **end**
```

Algorithm 3: *Compute2D($\mathcal{D}$, $\mathcal{P}$, k, l)*

```
   Input: Tile $T_{ij}$, PIS $\widehat{\mathcal{IS}}(T_1, T_{1'})$, PIS $\widehat{\mathcal{IS}}(T_2, T_{2'})$ where $T_1$, $T_2$ are adjacent to $T_{ij}$
   Output: Voronoi Diagram $\mathrm{VD}^{T_{ij}}(\mathcal{P})$, PIS $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'})$
1  Update $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \leftarrow \widehat{\mathcal{IS}}(T_1, T_{1'}) \cup \widehat{\mathcal{IS}}(T_2, T_{2'}) \cup \mathcal{S}(T_{ij})$
2  foreach site $p_a \in \widehat{\mathcal{IS}}(T_{ij}, T_{i'j'})$ do
3      if $\widehat{\mathcal{V}}(p_a) \cap T_{ij} = \emptyset$ then
4          $\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \leftarrow \widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}) \setminus \{p_a\}$
5      Compute distance field $D_{T_{ij}}(p_a)$
6      Update $\mathrm{VD}^{T_{ij}}(\widehat{\mathcal{IS}}(T_{ij}, T_{i'j'}))$
7      Check $D_{T_{ij}}(p_a)$ for visibility
8      if $D_{T_{ij}}(p_a)$ is visible then
9          UpdateBounds($p_a$,$T_{ij}$)
10 end
```
Algorithm 4: *ComputeTile(T_{ij},$\widehat{\mathcal{IS}}(T_1, T_{1'})$)*

2.10.2 3D Culling

In 3D, the domain is divided into $k \times l \times m$ cubical ranges. The set of ranges with the same Z coordinate forms a 2D domain called a slice. A slice is further divided into $k \times l$ rectangular tiles. We compute the 3D Voronoi diagram by computing m slices. Computation of a 2D slice is done as shown in Algorithm 3.

2.10.3 Conservative Sampling

The occlusion queries sample the visibility at fixed locations in each pixel and can result in sampling errors. In particular, the algorithm presented above may incorrectly classify a site p_a as receding if its Voronoi region $\mathcal{V}(p_a)$ does not cover any grid cells, i.e. the occlusion query returns zero visible pixels for the distance field $D_{T_{ij}}(p_a)$. This may introduce errors when $\mathcal{V}(p_a)$ intersects the range T_{ij} but its intersection with T_{ij} is not sampled by the rasterization hardware. An incorrect classification of p_a as receding can lead to errors in the Voronoi diagram of subsequent ranges.

We account for these sampling errors using a conservative sampling approach presented

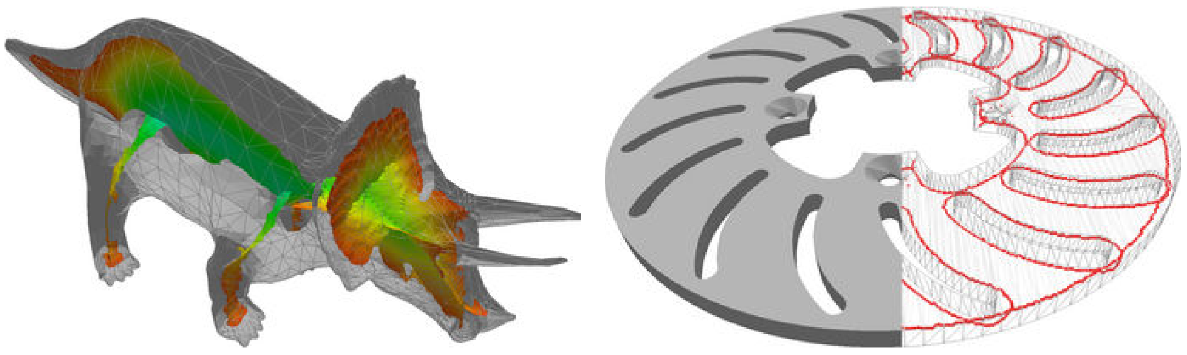


Figure 2.14: Approximate Medial Axis Transform*:* **Left***: Triceratops model (*$5.6k$ *polygons, Grid Size=*$255 \times 111 \times 84$*, Computation Time=*$0.7s$*) The medial surface is color coded by the distance from the boundary.* **Right***: Brake rotor model (*$4.7k$ *polygons, Grid Size=*$4 \times 128 \times 128$*, Computation Time=*$0.31s$ *). The medial seam curves are shown in red.*

in [SOM04]. This involves expanding the Voronoi region of each site by the size of a grid cell and again testing for visibility. Updates to the depth and color buffers are disabled during this computation.

2.11 Applications

We have applied our distance field algorithm to compute an approximate medial axis transform of polyhedral models and path planning. These applications require global distance field computation along a 3D grid.

Simplified Medial Axis Computation: We compute a simplification of the Blum medial axis, called the θ-*simplified medial axis* (θ-SMA) [FLM03]. The θ-SMA provides a good approximation of the stable subset of the medial axis. The algorithm for computing the θ-SMA of an object X is based on computing the vector field called the *neighbor direction field* of the object X and denoted by $N(X)$. $N(X)$ is the negated gradient of the distance field defined by the boundary of X. Given $N(X)$, a separation criterion is defined using the separation angle θ. The criterion is used to check whether a line segment connecting the centers of adjacent voxels of a grid crosses a sheet of the medial axis. When a pair of points passes the separation criterion, we add the facet between them to the approximation of θ-SMA and compute a polygonal approximation of the medial axis. In some cases a discrete voxel representation of the θ-SMA is desirable. A voxel is added to the medial axis if it lies on one side of a facet on the medial axis, which is determined as above. This selection operation can be efficiently performed on modern programmable graphics hardware using fragment programs. The gradient field is stored on graphics card texture memory. This avoids the costly readbacks of the entire distance field to the CPU.

Interactive Path Planning in Dynamic Environments: We have used our distance field computation algorithm within a constraint-based path planner [GL02]. The path planning problem is reduced to simulating a constrained dynamic system, and computes an approximation of the generalized Voronoi diagrams (GVD) of the robot and obstacles in the environment. Each robot is subject to virtual forces introduced by geometric and mechanical constraints, such as making the robot follow an estimated path computed using the GVD and linking the rigid objects together to represent an articulated robot. The distance field is used to compute an approximate GVD and a Voronoi graph. The distance field is also used to perform proximity tests between the robot and the obstacles and maintain a minimum clearance.

Given a pair of objects, R_1 and R_2, the distance field of R_2 is drawn in a potentially overlapping region. The surface of R_1 is sampled at points inside the overlapping region, and a force is generated at each sample point $\mathbf{q}_i$. The force is in the direction of the gradient of the distance field and proportional to the distance between $\mathbf{q}_i$ and the surface of R_2. As the obstacles in the environment undergo motion, our algorithm recomputes the distance field and uses it for path computation. We have used this path planner for virtual prototyping applications.

2.12 Implementation and Results

In this section we describe the implementation of our discrete generalized Voronoi diagram computation algorithm and highlight its performance on different benchmarks.

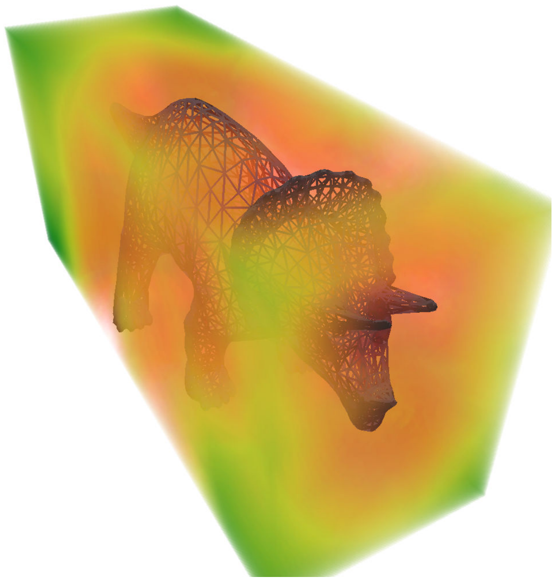


Figure 2.15: Triceratops Model*: Computation of 3D distance field and discrete Voronoi diagram of Triceratops model (5660 polygons). Distance increases from red to green. Grid size* = $255 \times 111 \times 84$*, Computation time* = 720*ms.*

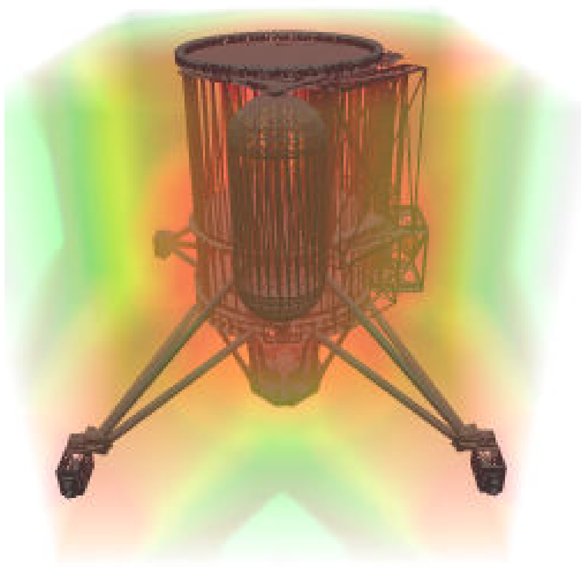


Figure 2.16: Cassini Model*: A volume rendering of the distance field of the Cassini with* 93*K polygons. The distance to the surface is color coded, increasing from red to green to blue. Grid size* = $186 \times 254 \times 188$*, Computation time* = 5.86*s.*

2.12.1 Implementation

We have implemented our algorithm on a PC with a 3.2 GHz Pentium IV CPU with 2 GB memorywith an NVIDIA GeForce 7800 GPU connected via 16x PCI Express bus and running Windows XP operating system. We used Microsoft Visual C++ 6.0 compiler and OpenGL graphics API. The distance functions for each primitive are computed at each grid cell on programmable graphics hardware using the OpenGL's ARB_fragment_program extension. The fragment program is used to compute the magnitude of the distance vector. We computed the distance field using a render texture with 32-bit floating point precision. The slices of the 3D domain are laid out on a 2D texture using flat 3d textures [HBSL03]. Our run-time algorithm computes the distance vectors for the vertices of the convex polygon associated with each site. The polygons are rasterized onto the rendertexture. We incorporated the optimizations to improve the performance of our algorithm on manifold objects. The visibility test is performed using the OpenGL occlusion query extension GL_NV_occlusion_query [GL-02]. We efficiently utilize the parallelism on a GPU by batching together the occlusion queries for an entire set of potentially intersecting sites. Our implementation involves no pre-computation and is directly applicable to deformable models.

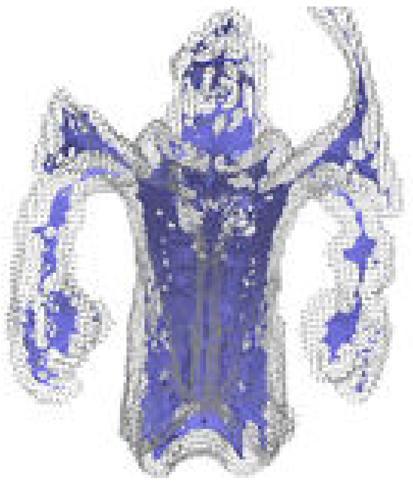


Figure 2.17: Brain Model *(*$78 \times 110 \times 60$ *image,* 18944 *boundary voxels): Voxel centers are shown as points. The θ-SMA ($\theta = 100°$) is shown in blue. Computation Time* = $0.75s$*. The θ-SMA does not provide any topological guarantees on the output.*

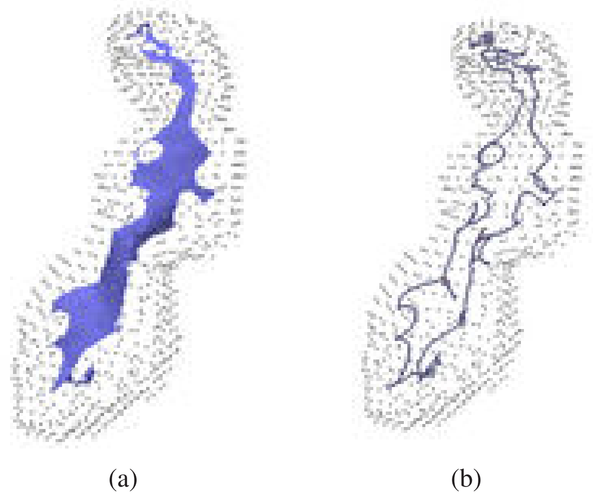


Figure 2.18: Right Hippocampus in the Brain Model *(*813 *boundary voxels),* $\theta = 90°$*: Voxel centers are shown as points. (a) Medial Axis approximation (θ-SMA) (b) Boundary and Seam Curves. The θ-SMA does not provide any topological guarantees on the output.*

In 2D, our sweep based algorithm computes the Voronoi diagram for all tiles in row i before computing the Voronoi diagram for tiles in row $(i + 1)$. We store the Voronoi region bounds as intervals along X and Y axes. In particular, we need to store the interval along X for the current and previous rows only, giving tighter bounds compared to an AABB. In 3D, we store the Voronoi region bounds along X and Y for the current and previous slices. In addition, all

For medial axis computation, we generate the gradient vector field along with the distance field. The gradient vectors are stored at 32-bit precision in the color buffer of floating point textures. The voxel representation of the θ-SMA is computed on the graphics processor using OpenGL's ARB_fragment_program extension. The medial axis is rendered directly from the GPU as a volume grid.

Given a discrete image data set, we compute the set of boundary voxels, which are all background voxels adjacent to at least one feature voxel. A point primitive is placed at the center of the boundary voxels. We use two optimizations to improve the performance on these datasets:

1. The distance of the point primitive to a slice belongs to a finite discrete domain. We precompute a set of distance meshes [HCK+99a] corresponding to set of distance value and store them as vertex buffer objects in the GPU memory.

2. Instead of encoding the gradient vector into the color buffer for each vertex of the distance mesh, we encode the position of the closest boundary voxel. We make a second pass during which a fragment program efficiently computes the gradient vector at each voxel.

2.12.2 Performance

We have applied our algorithm to various benchmarks, such as 2D data sets, 3D polygonal as well as image models. These include scanned models and CAD models. Some of them are non-manifold.

2D Models: We have applied our algorithm to several 2D models of points and lines. The sites are distributed randomly across the domain. We have compared the performance of our

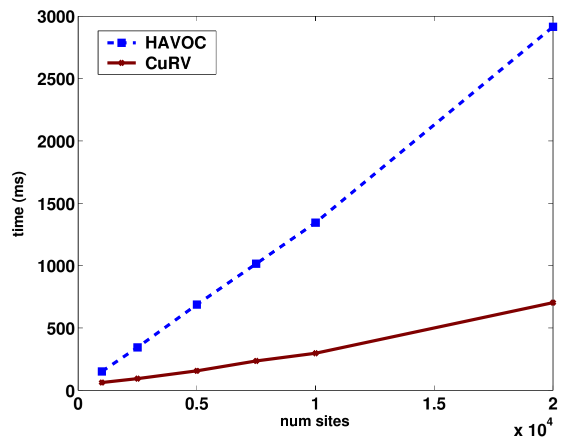


Figure 2.19: Timing Comparison*: Growth of time to compute the 2D discrete Voronoi diagram with number of random sites, using HAVOC(Hoff et al. 99) and CuRV (our algorithm). We have used a high grid resolution of* 1200×1200 *for the computation of discrete Voronoi diagram and used a tile size equal to* 75×75.

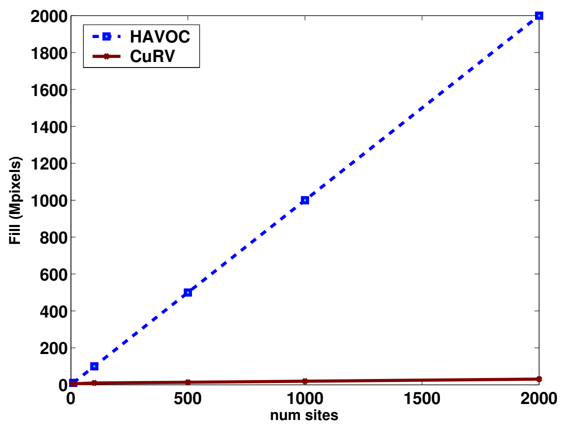


Figure 2.20: Fill Rate*: Number of pixels where distance function is computed, using HAVOC(Hoff et al. 99) and CuRV (our algorithm). We have used a grid resolution of* 1024×1024*, and a tile size of* 32×32 *to compute the Voronoi diagram. Our results indicate upto two orders of magnitude reduction in fill over HAVOC.*

Voronoi diagram computation algorithm (called CuRV) with the algorithm presented by Hoff et al. [HCK$^+$99a] (called HAVOC).

We have measured the performance of our algorithm with varying number of sites. Fig. 2.19 highlights the performance on upto 20K sites. We observe that the Voronoi computation time scales linearly with the number of sites. Furthermore, the computation time scales better than HAVOC. The computation cost of the Voronoi diagram is directly proportional to the *fill rate*. The fill rate is the number of sample points (pixels) where the distance function computation is evaluated. Fig. 2.20 shows the fill rate requirements of CuRV and HAVOC. Our experimental results indicate up to 5 times performance improvement over HAVOC and approximately two orders of magnitude reduction in the overall fill rate.

3D Models: We have applied our algorithm to compute 3D distance field and discrete generalized Voronoi diagram of polyhedral models (see figure 2.15).

For polygonal models, we have compared the performance of our distance field com-

Model	Polys	Resolution	CSC	HAVOC	HAVOC+CSC	DiFi
Rotor	4736	4x128x128	59.22	3.87	2.18	0.31
Rotor	4736	8x254x254	424.89	9.23	6.12	0.48
Triceratops	5660	128x56x42	127.81	2.11	1.10	0.41
Triceratops	5660	254x111x84	990.48	3.87	3.65	0.76
Hugo	17000	73x45x128	X	20.55	15.24	1.22
Hugo	17000	145x90x254	X	85.84	55.85	2.63
Head	21764	78x105x128	201.12	17.47	12.76	0.846
Shell	22598	254x252x252	X	81.97	41.31	2.12
Cassini	93234	186x254x188	X	186.03	148.55	5.86
Dragon	108926	57x90x128	X	89.13	49.69	4.76

Table 2.1: Distance Field Computation (Polygonal Models)*: Times (in seconds) to compute the global distance fields using approaches by Mauch [Mau03] (CSC), Hoff et al. [HCK$^+$99a](HAVOC), an implementation combining CSC with HAVOC on graphics hardware (HAVOC+CSC), and our algorithm (DiFi). For the entries marked X, CSC algorithm fails as the model contains non-manifold sites.*

putation algorithm (DiFi) with the algorithm presented by Hoff et al. [HCK$^+$99a](called HAVOC), a software implementation of CSC algorithm [Mau03], and an implementation that combines HAVOC with CSC. The timings are presented in Table 2.1. For the image data set, we have compared our algorithm with an implementation of Danielsson's 4SED algorithm [Dan80] from the ITK toolkit library. The results are shown in Table 2.2. The polygonal representation of θ-SMA was smoothed using the algorithm presented by Taubin [Tau95]. In our

Model	Resolution	Points	4SED	DiFi
Octahedron Image	$256 \times 256 \times 58$	4862	4.62	0.45
Brain	$78 \times 110 \times 60$	18944	0.55	0.75
Brain Lat Vent	$78 \times 110 \times 60$	4988	0.55	0.32
Sinus1	$406 \times 363 \times 392$	34507	66.1	5.1
Sinus2	$406 \times 363 \times 392$	104154	66.1	9.7

Table 2.2: Distance Field Computation (Image Models): *Times (in seconds) to compute the gradient field of image models using the 4SED algorithm [Dan80] and our algorithm (DiFi). Points refers to the number of boundary voxels.*

benchmarks, DiFi obtains more than two orders of magnitude over a software implementation of the CSC algorithm and upto one order of magnitude performance improvement over an implementation combining HAVOC and CSC for manifold objects. For non-manifold models, we obtain upto 30 times speedup over HAVOC. For 3D image models, we are able to obtain up to 10 times performance improvement over the 4SED algorithm [Dan80].

Our approach takes a fraction of a second to compute the distance field of a model with thousands of polygons on a $256 \times 256 \times 256$ grid. We analyzed our implementation using Intel's vTune benchmarking software. The time spent on computation of the bounding convex polygons was approximately 12% of the total time. The observed maximum number of triangles sent to GPU was 3MTris/s; maximum number of pixels rendered was 1.6Gpixels/s and estimated memory bandwidth achieved was 26GB/s.

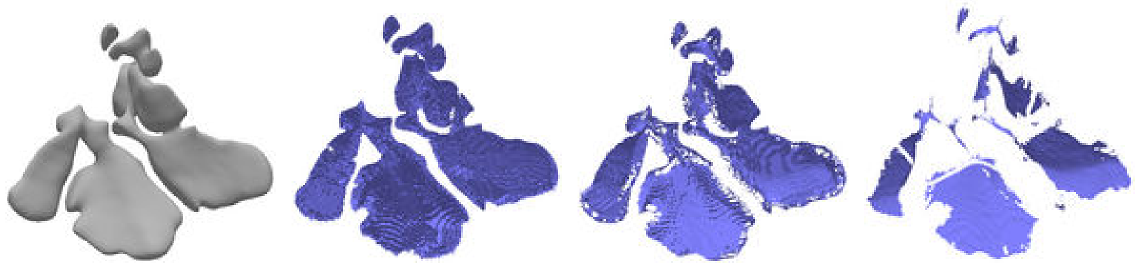

(a) Sinus model surface (b) $\theta = 15°$ (c) $\theta = 60°$ (d) $\theta = 105°$

Figure 2.21: Different θ-SMA for the Sinus Image Dataset Image *(*$406 \times 363 \times 392$*, 34507 boundary voxels). The surface representation shown was extracted using marching cubes.*

Medial Axis Computation: We have applied the distance field to compute the simplified medial axis of polyhedral models. The simplified medial axis for two models is shown in Figure 2.14. Our algorithm takes less than a second to compute the medial axis of polyhedral models consisting of thousands of polygons.

Path Planning: We have applied the path planning algorithm to an assembly environment (shown in Figure 2.22). The environment consists of an articulated robot arm with 6 degrees of freedom placed in the middle of a complicated piping structure. The robot arm reaches for a part moving on a conveyor belt and avoids collision with obstacles. Various links on the robot arm come in close proximity with the piping structures. We are able to dynamically compute the path at interactive rates using our fast distance field computation algorithm.

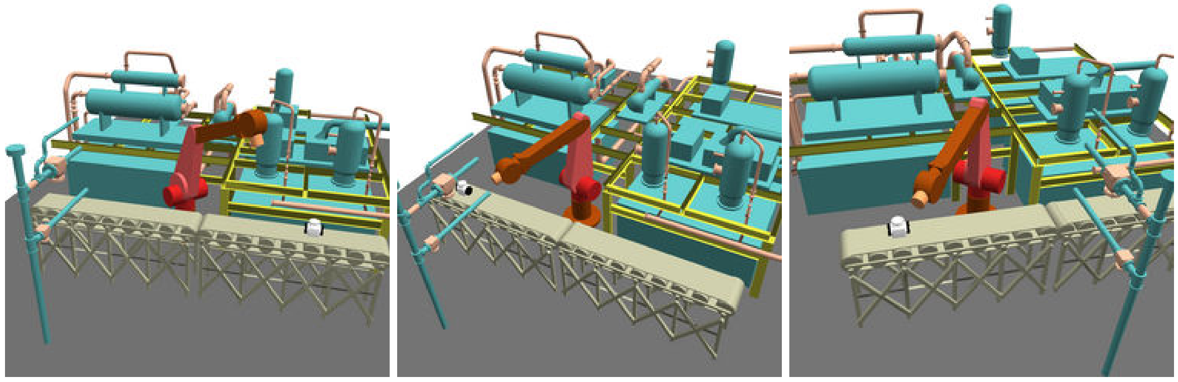


Figure 2.22: Planning in an assembly environment*: Constraint based planning in a dynamic environment consisting of* $26.9k$ *polygons using distance fields. The robot arm tracks a moving part on a conveyor belt, while avoiding contact with other obstacles in the environment. Our algorithm computes the distance field at interactive rates and uses the distance field to compute a collision free path.*

2.13 Discussion

In this section we analyze the computational complexity, space requirements and the accuracy of our algorithm. We also compare our algorithm with some existing distance fields and discrete Voronoi diagram computation algorithms.

2.13.1 Analysis

Accuracy: Our approach does not require tessellation of the distance functions. The computed distance field is accurate to 32-bit floating point precision. Hence the discrete Voronoi region deviates from the exact boundary by at most one cell (see figure 2.23).

Time Complexity: First we shall provide the time complexity for generalized range culling, and then provide the expressions for 3D distance field computation for the special case when each slice is treated as one tile (range). Let the model contain m sites.

Let the 2D domain $\widetilde{\mathcal{D}}$ contain $k \times l$ tiles, each covering p grid cells (pixels). We introduce the notion of *average PIS size* which gives the average number of sites for which the distance

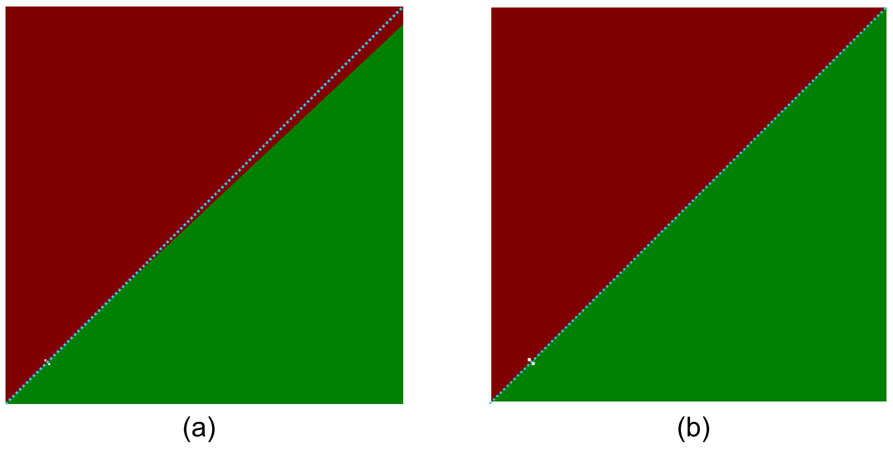


Figure 2.23: Voronoi Diagram Accuracy: *Error in Voronoi region computation in (a) HAVOC [Hoff et al 99] and (b) CuRV (our algorithm). There are two point sites close to the diagonal. The exact boundary is indicated using a dotted blue line. With HAVOC the error can be several pixels, whereas it is at most 1 pixel with CuRV*

field is computed per tile, and is defined as

$$\langle\widehat{\mathcal{IS}}\rangle = \frac{1}{kl}\sum_{i=1}^{k}\sum_{j=1}^{l}(|\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^+j^+})\,| + |\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^-j^+})\,| + \\ |\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^+j^-})\,| + |\widehat{\mathcal{IS}}\,(T_{ij}, T_{i^-j^-})\,|)$$

In higher dimensions, $\langle\widehat{\mathcal{IS}}\rangle$is similarly defined. Then the cost of updating the PIS in algorithm 4 is $O(\langle\widehat{\mathcal{IS}}\rangle\log\langle\widehat{\mathcal{IS}}\rangle)$. The cost of computing the distance field for each site is proportional to the number of grid cells (pixels) inside the tile, $O(p)$. Thus the total cost of one call to algorithm 4 is $O(\langle\widehat{\mathcal{IS}}\rangle\log(\langle\widehat{\mathcal{IS}}\rangle) + \langle\widehat{\mathcal{IS}}\rangle m)$, and the cost of algorithm 3 is $O\left(2^2kl(\langle\widehat{\mathcal{IS}}\rangle\log\langle\widehat{\mathcal{IS}}\rangle + p\langle\widehat{\mathcal{IS}}\rangle)\right)$. Similarly in n dimensions, the computational cost is $O\left(2^n\prod_{i=1}^{n}k_i(\langle\widehat{\mathcal{IS}}\rangle\log\langle\widehat{\mathcal{IS}}\rangle + m\langle\widehat{\mathcal{IS}}\rangle)\right)$. Note that $\prod_{i=1}^{n}k_i\times p$ gives the total number of grid cells in domain $\widetilde{\mathcal{D}}$.

In 3D, let the number of cells in domain $\widetilde{\mathcal{D}}$ be $M = k\times k\times k$, the cost of computing the distance field is proportional to the number of processed cells over which the distance function is evaluated. The optimal cost for computing the 3D distance field is $O(k^3) = O(M)$. For a slice s_k, the *optimal* number of processed cells is $\sum_{i=1}^{|\mathcal{I}(k)|}|Q_{i,k}| = k^2$. The *actual* number of processed cells is $\sum_{i=1}^{|\widehat{\mathcal{I}}_k|} |\widehat{Q}_{i,k}|$. We define the following average number of cells covered by one site:

optimal = $\langle |Q_{i,k}| \rangle = \frac{\sum_{i=1}^{|\mathcal{I}(k)|} |Q_{i,k}|}{|\mathcal{I}(k)|}$, actual = $\langle |\widehat{Q}_{i,k}| \rangle = \frac{\sum_{i=1}^{|\widehat{\mathcal{I}}_k|} |\widehat{Q}_{i,k}|}{|\widehat{\mathcal{I}}_k|}$

The per-slice efficiency of our algorithm can be measured by two ratios: the *clamping efficiency*, $e_{1k} = \frac{\langle |Q_{i,k}| \rangle}{\langle |\widehat{Q}_{i,k}| \rangle}$ and *culling efficiency*, $e_{2k} = \frac{|\mathcal{I}(k)|}{|\widehat{\mathcal{I}}_k|}$. The *average efficiency* per slice can be defined as $\langle e \rangle = \frac{1}{k} \sum_{k=1}^{k} e_{1k} \times e_{2k}$. The total cost of the algorithm is $O(M/\langle e \rangle)$, and is bounded between $O(M)$ and $O(mM)$. For non-manifold sites, the clamping efficiency e_{1k} approaches 1 as the sites are uniformly distributed on the 3D grid. For manifold sites, the complexity is similar to that of the CSC algorithm, i.e. $O(m + rM)$. However, our algorithm obtains tighter bounds on the parameter r, $r = 1/\langle e \rangle$. In practice, $e_{1k} \approx 1$, thus $r = \frac{1}{\langle e_{2k} \rangle}$.

Storage Cost: In terms of storage cost, we have to store the PIS for each tile. Thus the storage cost increases by $O(kl\langle \widehat{\mathcal{IS}} \rangle)$ in 2D and $O(\prod_{i=1}^{n} k_i \langle \widehat{\mathcal{IS}} \rangle)$ in n dimensions.

2.13.2 Comparison

We now compare our algorithm (DiFi) with some previous approaches to compute discrete generalized Voronoi diagrams using graphics hardware: HAVOC [HCK$^+$99a], a quadtree based culling algorithm [Den03b] and GPU-based CSC algorithm. [SPG03].

HAVOC: HAVOC ([HCK$^+$99a]) computes discrete generalized Voronoi diagrams in 2 and 3 dimensions, under any distance function. However the computational complexity of HAVOC is $O(mM)$, where m is number of sites in $\mathcal{P}$ and M is number of grid cells in $\widetilde{\mathcal{D}}$. For large models, the distance mesh approximation step becomes a bottleneck as all the triangles are sent from the CPU to the GPU during each frame for rasterization. This approach does not scale well with large number of sites and in higher dimensions. Furthermore, the discrete Voronoi diagram computed by HAVOC can have significant errors in computed Voronoi regions and can deviate from the exact Voronoi region by more than a single cell (see Fig. 2.23). This error is caused due to the tessellation error in the distance functions used by

HAVOC.

In contrast, our algorithm only computes the distance vector at the vertices of the convex polygons and uses the bilinear interpolation capabilities of the texture mapping hardware. Our distance computation algorithm has much lower CPU-GPU bandwidth requirements as we only transmit the distance vectors at the vertices of the convex polygon of each site. Our algorithm provides bounds on the region of distance computation for each site, and the approach is extensible to n dimensions, and scales well to large number of sites. Also, the computed distance field is accurate to 32-bit floating point precision, hence the discrete Voronoi region deviates from the exact boundary by at most one cell (see figure 2.23). However, in order to compute the Voronoi diagram efficiently, DiFi utilizes the connectivity property of Voronoi diagrams. Therefore, it is applicable for only distance functions which are metrics. Like HAVOC, it is applicable to generic models without connectivity information, and has the same error bounds.

Quadtree Culling: Denny [Den03a] presents an efficient approach for computing 2D discrete Voronoi diagrams for points under Euclidean distance, when the point distribution is approximately uniform in the domain. This approach increases the amount of tessellation to bound errors in the Voronoi region boundaries to 1 cell size. However, the approach is not directly extensible to 3D and higher order sites, and is sensitive to the order of computation of the distance functions. In comparison, our algorithm is simple and applicable to both higher order sites and dimensions.

GPU-based CSC algorithm: Sigg et al. [SPG03] also mentioned the idea of using bilinear interpolation and dot products. However, they do not provide any details on their derivation or implementation. Instead they present an approach which reduces the number of polyhedra that are scan converted. The fragment program used by [SPG03] is more complex and increases the load on the fragment processor. Moreover, Sigg et al.'s algorithm is restricted to inputs that are closed manifolds, and has the same asymptotic complexity as the CSC algorithm [Mau03], i.e. $O(m + rM)$. Overall, their approach is useful for very highly tessellated models and distance field computations with low-grid resolutions and narrow band sizes. In these cases, each polyhedron can become smaller than a few voxels. The polygon transform can become a bottleneck, and reducing number of polyhedra scan-converted provides speedups. For small band sizes, the parameter r is close to unity. However, for computing the global distance field of complex environments with multiple manifold surfaces and high depth-complexity, r can be $O(m)$. Further, it does not provide the complete generalized Voronoi diagram.

In contrast, we provide a formal presentation of the linear factorization and the necessary details to implement it. Our algorithm is applicable to general polygonal models. Furthermore, our approach computes a pixel accurate discrete generalized Voronoi diagram (as demonstrated in Figure 2.23). Our fragment program is much simpler. For large grid resolutions and global computations, the distance computation on the fragment processor becomes the bottleneck and our approach is more efficient that [SPG03]. This has been verified by our benchmarks. The cost of slicing the polyhedra is small and the observed triangle transfer rates are significantly less than the theoretical peak. Further, our approach makes efficient use of the fragment processor.

2.13.3 Limitations

Our algorithm has certain limitations. Our distance field computation is performed on a uniform grid and its accuracy is governed by grid resolution. Current graphics processors provide up to $4K \times 4K$ pixel resolution and this imposes an upper bound on the grid resolution. The accuracy of the algorithm is governed by that of the graphics hardware. For example, the current hardware provides support for 32-bit floating point representation and it is not fully compatible with the IEEE floating-point standard. Secondly, our algorithm involves a readback from the GPU back to the CPU, which can have additional overhead for high resolution distance fields. For narrow bands, and highly tessellated models, polygon transformation can become a bottleneck.

Our algorithm is only useful for computing discretized distance fields and the resulting algorithms for proximity queries and medial-axis computation only perform approximate computations (up to grid resolution). Hence there are no topological guarantees on the computed medial axis approximation. An adaptive approach that provides topological guarantees on the output is presented in Chapter 5.

Chapter 3

Surface Distance Maps

In this chapter, we consider the problem of computing the distance map on triangulated meshes in $\mathbb{R}^3$. The *surface distance map* is defined as follows: Given a set $\mathcal{P}$ of triangulated objects, at each point on an object O_i the surface distance map provides distance to closest object in $\mathcal{P} \setminus O_i$ (the closest object is trivially the one on which the point lies). The distance function varies continuously along the object surface and the gradient of the distance map at a point yields the direction vector to the closest object that does not contain the point. If the primitives are orientable, we can also associate a sign with the distance map.

Most of the prior techniques compute the distance field along a volumetric grid or a voxelized representation of space. At a broad level, these algorithms can be classified into object space methods that represent the distance field using adaptive grids or image space methods that compute the closest primitive at each grid point on a uniform grid. The latter methods can be accelerated by rasterizing the distance functions using the graphics hardware as shown in Chapter 2. These algorithms compute the distance field along each slice of a 3D grid and the computation can be accelerated by using spatial bounds on the Voronoi regions of the primitives [SOM04, PS05]. However, these volumetric techniques have many limitations. Their storage overhead and computation time is $O(k^3)$, where k is the resolution along the grid. As a result, current 3D distance field computation algorithms are not fast enough for interactive applications. Moreover, their accuracy can be low as most of the grid

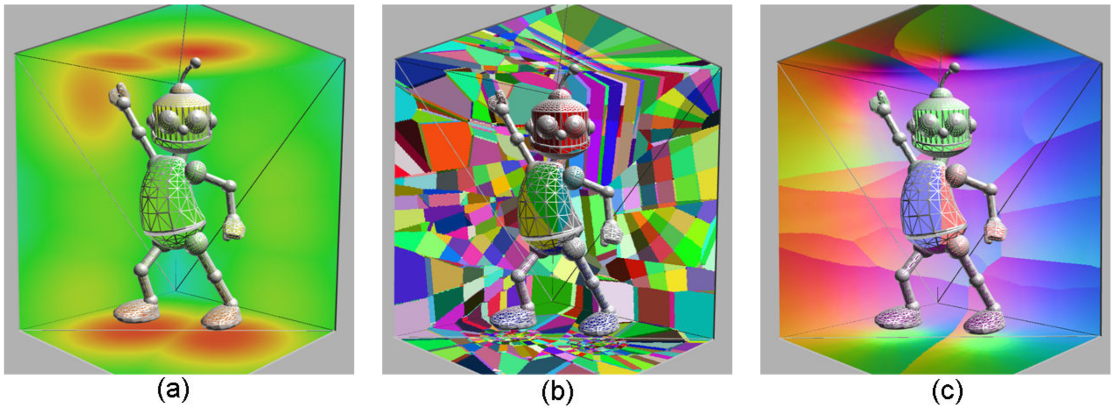


Figure 3.1: Surface distance map of the Hugo model enclosed in a box: *We show the surface distance maps between the Hugo model* ($17.2K$ *polys - in wireframe) and a box* (12 *polys - in wireframe). (a) The surface distance fields of Hugo on the box and of the box on the Hugo model. The distance increases from red to green. (b) The Voronoi diagrams of the Hugo and box that are used to compute the distance maps. Each colored region represents the intersection of the Voronoi region of a site on Hugo with the surface mesh of the box (and vice versa). (c) The normalized gradient of the distance field. The color of a point on the box encodes a vector representing the direction to its closest point on Hugo (and vice versa). Our algorithm can compute surface distance map of the Hugo and the box in 600ms on a grid of resolution* 256×256.

vertices do not exactly lie on the mesh, and adaptive sub-voxel refinement techniques are not well suited for graphics hardware [OLG$^+$05].

We present a new algorithm to compute surface distance maps of triangulated models. Our algorithm uses a simple texture representation to store a piecewise planar parametrization of the mesh. The parameterization defines an affine transformation for each primitive of the mesh. The 2D texture map is used as a discrete sampling of the mesh for distance map computation.

We apply the affine transformation of the geometric primitive to compute the distance functions of 3D primitives using the texture mapping hardware. We use the stencil test to clip the distance functions to regions corresponding to the geometric primitive in the 2D texture. Our algorithm employs spatial hierarchies to localize the distance field computations and improve the overall performance.

3.1 Related Work

In this section, we give a brief overview of related work on distance fields and surface mappings.

3.1.1 Distance Fields

Algorithms to compute distance fields are widely studied. At a broad level, these algorithms can be broadly classified based on the model representations such as images, volumes or polygonal representations. Good surveys of these algorithms are given in [Cui99, Aur91].

The algorithms for image-based data sets perform exact or approximate computations in a local neighborhood of the voxels. [Dan80, Set99, BGKW95, MQR03, GF03]. Exact algorithms for handling 2-D and k-D images have been propose to compute the distance transforms in voxel data in $O(M)$ time, where M is the number of voxels [BGKW95, MQR03].

There is extensive work in computing the exact Voronoi diagram of a set of points [Aur91]. However, exact computation of Voronoi regions of higher order primitives such as lines or triangles is a hard problem due to its algebraic and combinatorial complexity. As a result, most practical algorithms compute an approximation to the Voronoi diagram by computing distance fields on a uniform grid or an adaptive grid. A key issue is the underlying sampling criterion used for adaptive subdivision [VO98, TT97, ER02, PF01].

The computation of a discrete Voronoi diagram on a uniform grid can be performed efficiently using graphics rasterization hardware. This idea was original proposed for point primitives in [WND97]. Hoff et al. [HCK$^+$99b] render a polygonal approximation of the distance function on depth-buffered graphics hardware and computed the generalized Voronoi Diagrams in two and three dimensions. The 3D algorithm computes each slice separately. An efficient extension of the 2-D algorithm for point primitives is proposed in [Den03a]. Sud et al. [SOM04, SGGM06] present algorithms efficiently compute distance fields of polygonal primitives by using a combination of culling and clamping algorithms and map the computations to the texture mapping hardware. In practice, these algorithms can improve the performance of 3D distance field computation considerably, but are not fast enough for interactive applications. Fischer and Gotsman [FG05] describe techniques to approximate higher order Voronoi diagrams and distance fields using GPUs.

A class of exact distance computation and collision detection algorithms based on external Voronoi diagrams are described in [Lin93]. A scan-conversion method to compute the 3-D Euclidean distance field in a narrow band around manifold triangle meshes (CSC algorithm) is presented by Mauch [Mau03]. The CSC algorithm uses the connectivity of the mesh to compute polyhedral bounding volumes for the Voronoi cells. The distance function for each site is evaluated only for the voxels lying inside this polyhedral bounding volume. Sigg et al. [SPG03] describe an efficient GPU based implementation of the CSC algorithm. Peikert and Sigg [PS05] present algorithms to compute optimized bounding polyhedra of the Voronoi cell for GPU-based distance computation algorithms. Lefohn describe an algorithm for interactive deformation and visualization of level set surfaces using graphics hardware [LKHW03].

3.1.2 Surface Mapping and Parameterization

Surface distance maps can be regarded as a *mapping* computed on the surface. In some ways, this problem is related to other surface mapping problems such as texture mapping [Cat74], which is used to define the color on the surface; displacement mapping [Coo84], which consists of perturbations of the surface positions; bump mapping [Bli78], which give perturbations to the surface normals; and normal maps [Fou92], which contains the actual normals instead of the perturbations. All these mapping are supported by current graphics hardware.

The problem of computing a parameterization is well studied in the literature. A recent survey of these techniques is given in [FH05]. Given a closed model, these algorithms cut the model into charts such that each chart is homeomorphic to a disk. Each chart is parameterized separately and the final parameterization is an atlas of these chart parameterizations.

3.2 Surface Distance Maps

In this section, we present surface distance maps and our algorithm to compute them efficiently using texture mapping hardware. We first introduce the notation used in the paper.

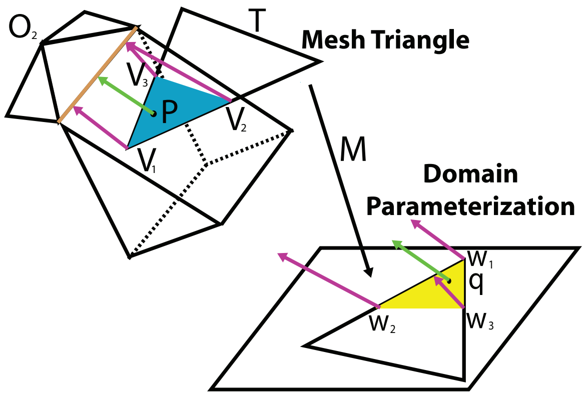


Figure 3.2: Affine map and distance computation: *We compute the distance map at a point* $\mathbf{q}$ *on triangle* t *(of* O_1*). The green vector shows the closest site of* O_2 *to* $\mathbf{q}$*. The affine map* $\mathbf{M^1}$ *maps triangle* t *to a triangle* $\bar{t}$ *in the 2D domain* $\mathcal{T}_1$*.*

3.2.1 Notation

We denote piecewise linear 2-manifold objects or meshes in 3D as O_i. Furthermore, O_i is decomposed into vertices, open edges and open faces, also known as *sites*. A site is denoted as p_i. Let $\mathcal{T}_i \subset \mathbb{R}^2$ represent the 2D parametric domain for object O_i. We use an overbar to represent the mapping of a 3D primitive to the 2D domain $\mathcal{T}$, for e.g. A point $\mathbf{q}$ and triangle t in 3D map to $\bar{\mathbf{q}}$ and $\bar{t}$ respectively on $\mathcal{T}$.

The distance function of a site p_i at a point $\mathbf{q} \in \mathbb{R}^3$ is denoted $d(\mathbf{q}, p_i)$. The distance function of a site p_i on a triangle t in the 3D mesh represents the closest distance from each point $\mathbf{q} \in T$ to p_i. The closest vector from $\mathbf{q}$ to p_i is known as the distance vector, denoted $\vec{\mathbf{d}}(\mathbf{q}, p_i)$.

Given two triangulated objects O_1 and O_2, the surface distance map $D(O_1)$ of an object O_1 at a point $\mathbf{q} \in O_1$ is the minimum value of the distance functions of all sites $p_k \in O_2$ at $\mathbf{q}$. We define an affine mapping $\mathbf{M_i^1} : t_i \rightarrow \mathcal{T}_1$ to transform the sampled points on the triangles $t_i \in O_1$ into the 2D domain $\mathcal{T}_1 \subset \mathbb{R}^2$. For ease of notation, when the object id j is implicit ($j = 1$ in this case), we shall drop the superscript from $\mathbf{M_i^j}$ and denote the affine map as $\mathbf{M_i}$.

3.2.2 Distance Fields: Background

Distance fields can be computed efficiently on discrete volumetric grids by rasterizing the distance function of each site to the points in the grid. Many algorithms compute the distance functions from each site to the points on the planes swept along the Z-axis of the grid [SOM04, SGGM06, SPG03]. These algorithms perform the distance field computation using one of these approaches:

1. Evaluate the distance function $d(\mathbf{q}, p_k)$ at each point $\mathbf{q}$ in the plane directly by rasterizing the distance functions and use the depth-buffer hardware.

2. Compute the distance vector from $\mathbf{q}$ to the site and use the magnitude of the distance vector to compute $d(\mathbf{q}, p_k)$. This computation can be efficiently performed using the bilinear interpolation capabilities of the texture mapping hardware.

In order to accelerate the computations, prior algorithms construct a convex bounding polytope G to bound the Voronoi region of site. As a result, the distance function is only evaluated at the points inside. Details on the computation of these polytopes are given in Section 2.4. We use similar techniques to accelerate the computation of surface distance maps.

3.2.3 Planar Parameterization

Given a 3D mesh O with triangles $t_k, k = 1, \ldots, n$, our algorithm transforms t_k into $\bar{t_k}$ by applying an affine mapping $\mathbf{M_k}$ (see Fig. 3.2). $\mathbf{M_k}$ is represented as a matrix and ensures the following properties:

- There is a one-to-one mapping from a point $\mathbf{q} \in t_k$ to the point $\mathbf{M_k q} \in \bar{t_k}$.
- No two transformed triangles $\bar{t_k} = \mathbf{M_k} t_k$ and $\bar{t_l} = \mathbf{M_l} t_l$ share a common interior point in the 2D domain $\mathcal{T}$.

These constraints are satisfied using piece-wise planar parameterizations of the surface in 3D space and the mapped triangles can be represented in a 2D texture atlas.

The affine transform for a triangle t_k with vertices $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}$ to a triangle $\bar{t_k}$ with vertices $\bar{\mathbf{v_0}}, \bar{\mathbf{v_1}}, \bar{\mathbf{v_2}}$ in 2D domain $\mathcal{T}$ is given as

$$\mathbf{M}(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mathbf{v_0}) + \mathbf{v_0} \tag{3.1}$$

where

$$\mathbf{A} = \begin{bmatrix} \bar{\mathbf{v_1}} - \bar{\mathbf{v_0}} & \bar{\mathbf{v_2}} - \bar{\mathbf{v_0}} & \bar{\mathbf{v_3}} - \bar{\mathbf{v_0}} \end{bmatrix} \begin{bmatrix} \mathbf{v_1} - \mathbf{v_0} & \mathbf{v_2} - \mathbf{v_0} & \mathbf{v_3} - \mathbf{v_0} \end{bmatrix}^{-1}$$
$$\mathbf{v_3} = (\mathbf{v_1} - \mathbf{v_0}) \times (\mathbf{v_2} - \mathbf{v_0})$$
$$\bar{\mathbf{v_3}} = (\bar{\mathbf{v_1}} - \bar{\mathbf{v_0}}) \times (\bar{\mathbf{v_2}} - \bar{\mathbf{v_0}})$$

Since $\mathbf{M}$ is affine, $\mathbf{A}$ can be written as a composition of a scale, shear and rotation matrices. Mathematically, $\mathbf{A} = \mathbf{A_s A_r}$ where $\mathbf{A_s}$ represents a scale and shear matrix in the XY plane and $\mathbf{A_r}$ is a rotation matrix. We shall use this representation to perform an error analysis in Section .

3.2.4 Surface Distance Computation

Surface distance maps compute the distance-to-closest-primitive in the the scene to the sampled points on the surface of the mesh, excluding primitives on same mesh. We first compute the affine mappings, $\mathbf{M_k}$ for each triangle t_k in the 3D mesh. These affine map defines a sampling on each triangle t_k in 3D space by sampling the projected triangle $\bar{t_k}$ in the 2D domain $\mathcal{T}$. The surface distance map samples the domain $\mathcal{T}$ uniformly using a 2D texture. Instead of computing distances using a volumetric grid, our algorithm computes the distance map on each triangle t_k using affine transforms of distance functions to a 2D plane containing $\bar{t_k}$.

We present an algorithm to compute distance functions on a set of sampled points on the triangles of the 3D mesh. For each site p_i, we compute a convex bounding polytope G_i, which acts as a spatial bound on the Voronoi region of p_i. In other words, any point outside G_i can not lie in the Voronoi region of p_i. We intersect G_i with the triangle t_k in 3D mesh. Let $\mathbf{x}_1, \ldots, \mathbf{x}_l$ denote the vertices of $G_i \cap t_k$. From the bilinear interpolation property of distance vectors presented in Section 2.3, for a point $\mathbf{q} \in t_k$ the distance vector $\vec{\mathbf{d}}(\mathbf{q}, p_i)$ is a convex combination of the distance vectors at the vertices $\mathbf{x}_j, j = 1, \ldots, l$. Since $\mathbf{M_k}$ is affine, the distance vector at a point $\bar{\mathbf{q}} \in \bar{t_k}$ is the convex combination of distance vectors at $\bar{\mathbf{x}_j} = \mathbf{M_k}\mathbf{x}_j, j = 1, \ldots, l$.

Thus the distance vector computation on $G_i \cap t_k$ can be performed as follows:

1. Assign to each vertex $\mathbf{x}_j$ a vector $\vec{\mathbf{d}}(\mathbf{x}_j, p_i)$.

2. Each vertex $\mathbf{x}_j, j = 1, \ldots, l$ is transformed to a vertex $\bar{\mathbf{x}_j}$ in the 2D domain $\mathcal{T}$, using the affine map $\mathbf{M_k}$.

3. The distance vector $\vec{\mathbf{d}}(\bar{\mathbf{q}}, p_i)$ at a point $\bar{\mathbf{q}} \in \bar{t_k}$ is computed as a convex combination of the vectors $\vec{\mathbf{d}}(\mathbf{x}_j, p_i)$ associated with the vertices $\bar{\mathbf{x}_j}, j = 1, \ldots, l$.

4. The distance vector for a point $\mathbf{q} \in t_k$ is given as $\vec{\mathbf{d}}(\mathbf{q}, p_i) = \vec{\mathbf{d}}(\bar{\mathbf{q}}, p_i)$, where $\bar{\mathbf{q}} = \mathbf{M_k}\mathbf{q}$.

3.3 Interactive Distance Map Computation

In this section, we will present our algorithm to efficiently compute surface distance maps using GPUs.

3.3.1 Mapping to GPUs

Surface distance maps can be computed by the rasterization hardware by using the transformation, clipping and interpolation capabilities of the GPUs. We use the approach mentioned in Section 3.2.4 to design an efficient pipeline for surface distance map computation using GPUs:

- **Polytope Computation and Intersection:** We compute the bounding polytope of site p_i, and intersect it with the plane π_k containing triangle t_k on the *CPU*. This gives a convex polygon g_i in the plane π_k containing t_k.

- **Distance Vector Computation and Transform:** We compute the distance vectors at each vertex of g_i and project the vertices of g_i to the 2D domain $\mathcal{T}$ using the affine map $\mathrm{M}_{\mathbf{k}}$. This per vertex computation is efficiently performed in parallel using the *vertex processor* on the GPU.

- **Clipping:** We restrict the computation of distance vectors to the domain given by $G_i \cap t_k$. This is equivalent to clipping the polygon g_i against t_k, or in the 2D domain $\mathcal{T}$ clipping the projection $\bar{g_i}$ against $\bar{t_k}$. We use the stencil functionality of GPUs to clip the projection of g_i to the region inside $\bar{t_k}$.

- **Bilinear Interpolation:** The linear interpolation of distance vectors is equivalent to the interpolation of texture co-ordinates assigned to the vertices of the triangle. This functionality computes the distance vectors in the texture atlas.

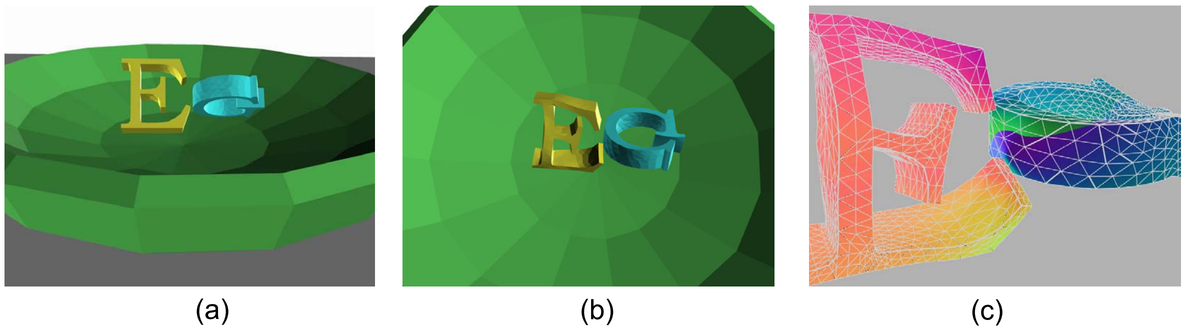


Figure 3.3: Surface Distance map computation on deforming letters "EG": *Deforming dynamic simulation on two letters,* $(3.7K$ *triangles total). (a)-(b) Two frames from the simulation. (c) The gradient of surface distance maps of each alphabet shows the direction of the closest point on the other alphabet. Our algorithm can compute the global distance maps for both bunnies in* $100ms$ *at a grid of resolution* 512×512.

- **Distance Computation:** The distance value at a texel in the texture atlas is the norm of the distance vector and computed using the fragment processor.
- **Distance Comparison:** The distance value is returned as depth and compared with the current minimum distance value using the depth test functionality of GPUs. The minimum distance value is stored in the depth buffer.

The complete algorithm to compute the surface distance map of object O_1 using sites in object O_2 is given in Algorithm 5. The algorithm requires computing of intersections between bounding polytopes of sites and the triangles in the 3D mesh, and clipping of polygons in 2D. We present details of stencil-based clipping and hierarchical culling techniques used to accelerate the performance of the algorithm.

3.3.2 Clipping

Surface distance maps require an efficient clipping algorithm for each triangle-site pair. Given a site p_i and a triangle t_k, we restrict the computation on the 2D domain to the interior of $\bar{t_k}$ using stencil. As a result, each triangle-site pair requires a valid stencil to be set in the region corresponding to t_k. We first describe an algorithm to perform clipping using a single valid stencil value, and present a more efficient stencil caching algorithm that uses multiple valid stencil values to perform clipping.

```
    Input: Two objects $O_1$, $O_2$. Parameterization from $O_1$ to $\mathcal{T}_1$.
    Output: The Surface Distance Map $D(O_1)$ of object $O_1$.
 1  Initialize $D(O_1)$ to $\infty$ for all points $\mathbf{q}$ in $\mathcal{T}_1$
 2  Update AABB hierarchy of $O_1$
 3  foreach triangle $t_k$ in $O_1$ do $\mathrm{M}^{\mathbf{1}}_{\mathbf{k}} \leftarrow \mathsf{UpdateAffine}(t_k)$
 4  foreach site $p_i$ in $O_2$ do
 5      $OBB(G_i) \leftarrow \mathsf{ComputeOBB}(p_i)$
 6      Intersect OBB($G_i$) against AABB hierarchy of $O_1$
 7      foreach triangle $t_k$ in $O_1$ intersecting $G_j$ do
 8          $g_i \leftarrow \mathsf{ClipPolytope}(G_i, t_k)$
 9          foreach vertex $\mathbf{x}_j$ in $g_i$ do
10              Compute distance vector $\vec{\mathbf{d}}(\mathbf{x}_j, p_i)$
11              Transform $\mathbf{x}_j$ to $\bar{x_j}$ using $\mathrm{M}^{\mathbf{1}}_{\mathbf{k}}$
12              Assign texture coordinates of $\bar{x_j}$, $(r, s, t) \leftarrow \vec{\mathbf{d}}(\mathbf{x}_j, p_i)$
13          end
14          Draw textured polygon $g_i$ on domain $\mathcal{T}_1$
15      end
16  end
17  Read-back $\mathcal{T}_1$
18  foreach triangle $\bar{t_k}$ in $\mathcal{T}_1$ do
19      Map distance values from $\bar{t_k}$ to $t_k$
20  end
```

***Algorithm 5**: Pseudo-code to compute the surface distance map of O_1 using sites in O_2. We initialize the distance values in the surface distance map $D(O_1)$ to ∞ (line 1). We then update the hierarchy and the affine transforms of triangles in O_1 using a linear-time algorithm (lines 2–3). Next, we update the surface distance map of O_1 using the sites in O_2 (lines 4 – 16). For each site, we compute its bounding polytope and intersect the OBB of bounding polytope with the AABB hierarchy (lines 5–6). For each intersecting polytope, we clip the bouding polytope using stencil tests (line 8) and compute the surface distance map (lines 7–15).*

The algorithm proceeds as follows. For each site p_i and a triangle t_k in a plane π_k in 3D space, we first compute the bounding polytope G_i and compute the convex polygon given by $G_i \cap \pi_k$. Next, we clip the transformed polygon $\mathrm{M}_{\mathbf{k}}(G_i \cap \pi_k)$ to $\bar{t_k}$ in the 2D domain. We use the stencil test functionality to perform the clipping operation. We first set the stencil value of the triangle to 1 by rendering $\bar{t_k}$. We then render $\mathrm{M}_{\mathbf{k}}(G_i \cap \pi_k)$ onto the portions of the surface distance map where the stencil value set is to 1. We then render $\bar{t_k}$ by setting the stencil value to 0 on the triangle.

For every two consecutive triangle-site pairs (t_k, p_i), $(t_l, p_j), k \neq l$, our algorithm resets stencil on regions corresponding to $\bar{t_k}$ and sets the stencil on regions corresponding to $\bar{t_l}$. Each reset and set stencil operations incurs a GPU state change and can become fill-bound. We improve the performance of our clipping algorithm using a buffer that maintains multiple stencil values to reduce number of stencil reset operations. Initially, all the stencil values are unassigned. As the surface distance map computations are performed on the triangles, the buffer allocates unassigned stencil values to each new triangle. In order to compute the valid stencil value for t_l, we first test if the stencil is set on regions corresponding to $\bar{t_l}$. If the stencil is set, we simply use that value for the clipping operations. On the other hand, if the stencil value is not set, we need to assign a valid stencil value to t_l. In order to assign a valid stencil value, we check if any of the stencil values in the buffer are unassigned. If an unassigned value is available, we assign that value to t_l. If no valid stencil value is available, the buffer uses the least recently used (LRU) replacement policy to determine the stencil value to be allocated to t_l. In this case, we first reset the stencil on the triangle whose stencil is least recently used and then allocate that stencil value to t_l.

3.3.3 Hierarchical Culling

We use a hierarchical distance culling algorithm to reduce the number of triangle-site pairs in the surface distance map computation. The distance functions are computed from a site p_i to a triangle t_k in 3D mesh only when the intersection of the bounding polytope with the triangle is non empty (i.e. $G_i \cap t_k \neq \emptyset$). We use an AABB-hierarchy of each object to quickly cull away sites whose bounding polytopes do not overlap with the triangles in the 3D mesh.

Our algorithm initially constructs an AABB hierarchy for each object. Each leaf of the hierarchy stores a triangle of the object. At run-time, we update the AABB-hierarchy and use it for culling bounding polytopes that do not intersect with the AABB-hierarchy. The hierarchy nodes are updated in a bottom-up manner. The update cost of a hierarchy is linear to the number of leaves in the AABB-hierarchy and is usually fast [GKJ$^+$05]. For each site p_i, we construct a bounding polytope G_i and compute a tight-fitting oriented bounding box $OBB(G_i)$ that encloses G_i. We perform overlap tests between $OBB(G_i)$ and the nodes of the AABB hierarchy. For each leaf with triangle t_k that overlaps with $OBB(G_i)$, we perform distance computations on $G_i \cap t_k$ as described in Section 3.3. The OBBs are constructed only once for each site, and therefore, the time taken to update the OBBs is linear to the number of sites in the scene.

We further improve the performance of our surface distance map algorithm by reducing the number of distance function rasterization operations using distance bounds computed using the AABB hierarchy. For each node in the AABB hierarchy, we maintain a lower bound on the maximum distance from the AABB of a triangle t_k to the AABB of the sites. Initially, the maximum distance bound of each node in the hierarchy is set to ∞. We do not perform distance evaluation of a site p_i for triangle t_k if the distance bound stored for a node in the hierarchy is less than than the minimum distance from the AABB of the node to the AABB of p_i. This culling test based on distance bounds is used to reject sites whose distance functions do not contribute to the distance map on t_k, as there exists some other sites that are closer to T_k.

If a site is not culled away, we intersect the bounding polytope G_i of the site with t_k and compute the distance vectors at the vertices of $G_i \cap t_k$. We then perform distance function computation on $G_i \cap t_k$.

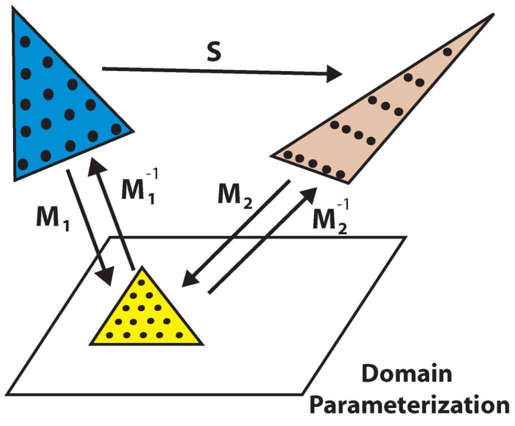


Figure 3.4: Distance map computation for a deforming triangle: *The triangle undergoes a non-rigid deformation (S) in terms of shear and scale. We compute a new affine mapping for the triangle (M_2) and use it to compute the distance map on the triangle. The sample locations are shown as dots in the 2D domain and the triangles.*

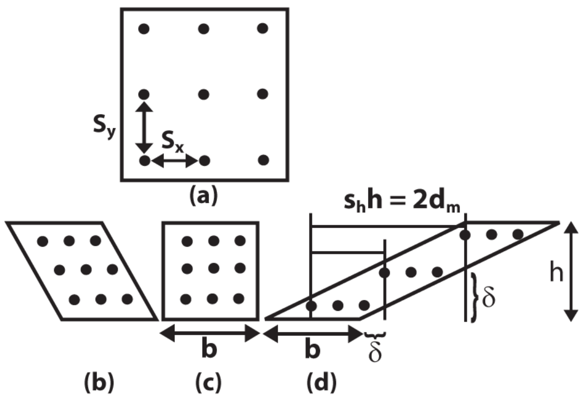


Figure 3.5:]
This figure highlights the distance between adjacent samples in the 2D plane when a rectangular planar primitive (shown in (c)) undergoes a scale (shown in (a)), or shear transformations ((b) and (d)). The shear in (b) does not increase the sampling error. The large shear in (d) increases the sampling error.

3.4 Error Analysis

In this section, we analyze the accuracy of our algorithm. We show that our algorithm can be used to compute a distance map up to a desired precision. We also consider the case when the triangles undergo non-rigid deformations and highlight the accuracy of distance maps based on the affine transformations.

The algorithm presented in Section 3.3 computes an accurate surface distance map at the sample points on the boundary of the objects. Its accuracy is governed by the precision of the texture mapping hardware that performs bilinear interpolation. Current GPUs offer 32-bit floating arithmetic to perform these computations. We also present an error bound on the computed distance for any point on the surface, as the object undergoes non-linear deformations. Given a sampling on the texture domain, we derive a function to compute the sampling density on the surface in 3D using the inverse of the affine map. Given the sampling density in 3D, we compute bounds on the distance. One can also use the inverse of the function to compute the sampling required in the texture domain to achieve a desired precision in the distance field.

Given two points $\mathbf{p}$ and $\mathbf{q}$ on an object O_1 and the surface distance map of O_1 w.r.t. object O_2, the change in the value of surface distance map from $\mathbf{p}$ to $\mathbf{q}$ is bounded by the distance between $\mathbf{p}$ and $\mathbf{q}$ [SOM04]:

$$\| d(\mathbf{p}, O_2) - d(\mathbf{q}, O_2) \| \leq \| \mathbf{p} - \mathbf{q} \| .$$

In order to bound the error in computed distances, we bound the distance between two adjacent samples on the mesh. This is bounded by the maximum distance between four adjacent samples in the 2D domain $\mathcal{T}$.

Let $\mathbf{p}$ and $\mathbf{q}$ be adjacent points on a triangle t_k on O_1. The corresponding points $\bar{p}$ and $\bar{q}$ on the texture domain $\mathcal{T}_1$ are given by the affine transform $\mathbf{M_k^1}$. The affine transform is defined using a combination of scaling, translation and rotations. The transform $\mathbf{M_k^1}$ is invertible since the scaling used to compute the affine transforms is non-zero. The distances are preserved under rigid transformations. Only scaling and shear change the distance between four adjacent samples and we derive error bounds under shear and scaling.

We assume the initial mapping $\mathbf{M_k^1}$ for each triangle t_k on object O_1 to the texture atlas $\mathcal{T}_1$ has unit scale and shear, and the spacing between two adjacent samples along each axis in $\mathcal{T}_1$ is δ. We provide a function $f(\delta)$ which bounds the distance between two adjacent samples in O_1, as triangles in O_1 undergo non-rigid deformation (see Figure 3.4).

In the initial position of O_1, since $s_x = 1, s_y = 1, sh = 0$, the distance between two samples is bounded by $f(\delta) \leq \sqrt{2}\delta$.

Let the maximum motion of a vertex in 3D, modulo any rigid body transformations, space be bounded by d_m. This gives a bound on the maximum deformation of a face on O_1. An upper bound on the scaling is given by $(s_x^2 + s_y^2) \leq 2d_m$. Maintaining the sample spacing

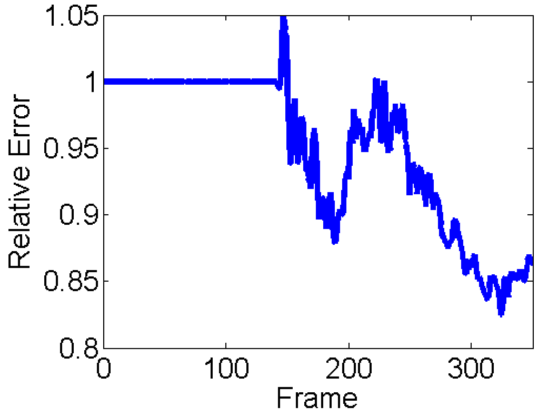


Figure 3.6: Relative error in distance map computation for a deformable model: *The relative error measures the ratio of maximum error in the surface distance map for a given frame to the maximum error in the beginning of the simulation. The error is introduced due to discrete sampling of the distance map. The graph highlights the relative error on a deformable simulation using a resolution of* 512×512. *The relative error provides an indicator of the error bounds in the discrete distance field during the simulation.*

in $\mathcal{T}_1$ to be δ, the maximum distance between two adjacent samples in O_1 is bounded by $f(\delta) \leq \sqrt{(s_x\delta)^2 + (s_y\delta)^2} \leq 2d_m\delta$.

We now show the change in distance between two adjacent samples when the shear exceeds a threshold, and derive the bounds. Consider a rectangular face in 2D with width b along X, and height h. Let the shear along Y be s_h. Assuming that the motion only produces shear (see figure 3.5(b)-(d)),

$$s_h = \frac{2d_m}{h} \tag{3.2}$$

Distance between two adjacent samples increases by more than $\sqrt{2}\delta$ only if the first sample in row $y + \delta$ beyond past the last sample in row y (see figure 3.5(d)), i.e. $s_h\delta > b + \delta$. Replacing from equation (3.2) we get

$$d_m > \frac{bh}{2\delta} + 1.$$

Thus, if $d_m \leq \frac{bh}{2\delta} + 1$, then there is no additional error due to shear. If $d_m > \frac{bh}{2\delta} + 1$,

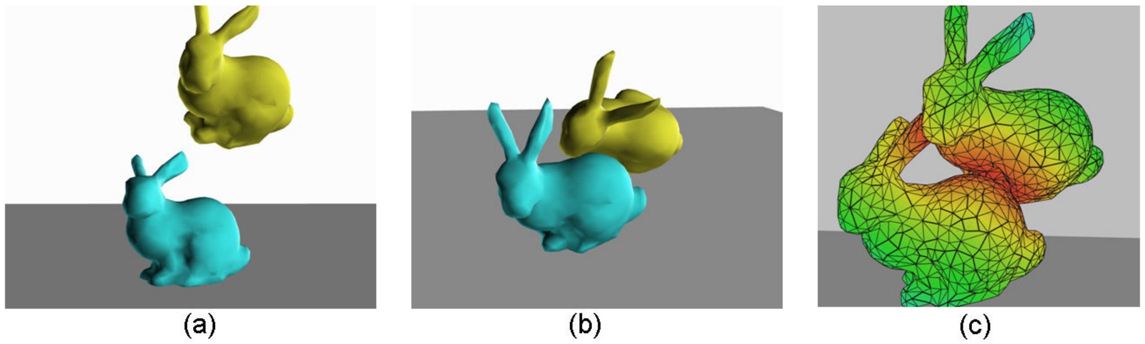


Figure 3.7: Surface distance map computation on deformable models: *Dynamic simulation of two deforming bunnies, each with* $2K$ *triangles . (a)-(b) Two frames from the simulation. (c) The surface distance map of both the bunnies that shows the distance field on the boundary. The distance increases from red to green. Our algorithm can compute the global distance maps in* $300-320ms$ *at a resolution of* 512×512.

then the effective increase in spacing along X axis between two rows of adjacent samples is $d_x = \max(s_h - (\frac{b}{\delta}+1), 0)$. In presence of scaling, the spacing along each axis is replaced by $s_x\delta$ and $s_y\delta$ respectively. We make the simplifying assumption that $s_x = s_y$. Then the increase in spacing along X is given by $(s_x + d_x)$, where $d_x = \max(s_h - (\frac{b}{s_x\delta}+1), 0)$, and the total error bound in the distance is $f(\delta) \leq \delta\sqrt{(s_x+d_x)^2 + s_y^2}$. A plot of this error for first 350 frames of a deformable model simulation is shown in figure 3.6.

3.5 Implementation and Performance

In this section we describe the implementation of our algorithm to compute surface distance maps between deformable models. We also compare our algorithm with prior distance field computation algorithms.

3.5.1 Implementation

We have implemented our algorithm on a PC with a 2.4Ghz Opteron 280 CPU, 2GB of memory and an NVIDIA 7800 GTX GPU connected via a PCI-Express bus, running Windows XP operating system. We used OpenGL as the graphics API and the Cg programming language for implementing the fragment programs. The initial mapping from the manifold objects to the texture atlas is computed using NVIDIA's Melody [1] software. The surface distance map of each object is computed on a floating point buffer using 32-bit floating point precision. The distance vectors are passed as texture parameters to the fragment program.

Our algorithm can compute high-resolution (512×512 to $1K \times 1k$) surface distance map of objects with tens of thousands of polygons in a fraction of a second. We highlight the performance of our algorithm on scenes with varying polygon counts is highlighted in the graph. We also compute the gradient of the distance field which gives the direction to the closest primitive for a point on the surface of an object. As compared to prior approaches based on volumetric techniques, our surface distance map computation algorithm is about $5 - 10$ times faster.

3.5.2 Proximity Queries

We use our algorithm to compute proximity information among 3D deformable models. This includes separation distance, collision detection, penetration depth and contact normal computation [HZLM01]. We first localize the region of overlap between two objects O_1 and O_2, and compute the surface distance map for all triangles of each object that lie inside the localized region. The separation distance between two objects is computed using minimum Euclidean distance from points on one object to points on the other object. We read back the surface distance maps of O_1 and O_2, and scan the pixels to determine the minimum distance. Collision detection is performed by checking for pixels with zero distance. In order to compute local penetration depth, we assign a sign to the distance values based on the orientation of the surface. In particular, all points of O_2 that are inside O_1 are assigned negative distance values. We then compute the maximum of these values to approximate the local penetration depth.

[1] http://developer.nvidia.com/object/melody_home.html

We used our algorithm for proximity query on 2 scenarios consisting of deforming objects. The first is a sequence of two deforming alphabets as shown in figure 3.3. The alphabet 'E' consists of 2.1K polygons, while the object 'G' consists of 1.6K polygons. At each frame, we compute a surface distance map at a resolution 512×512. The average time to perform all proximity queries is 110ms. As compared to [HZLM01, SGGM06], our surface distance algorithm results in speedup of 8 times. All these GPU-based algorithms are image-space algorithms. Since we are computing the distance map at a much higher resolution, the image-space error using our algorithm is much lower as compared to prior approaches. We also perform the proximity computation on a sequence of two deforming bunnies. Each bunny consists of 2K polygons. At each frame, we compute a surface distance map at a resolution 512×512. The distance map computation and proximity queries take about $300 - 320$ ms per frame.

3.6 Discussion

In this section we analyze the computational complexity and space requirements of our algorithm. We also compare our algorithm with some existing distance fields and discrete Voronoi diagram computation algorithms.

3.6.1 Comparison

We compare the features and performance of our surface distance map algorithm with prior approaches that compute the distance field on a uniform volumetric grid using GPUs. These include DiFi [SOM04], linear factorization [SGGM06] and efficient GPU implementations of CSC algorithm [SPG03, PS05]. All the prior approaches com pute the distance field along a uniform 3D grid. Since the GPU computes the distance field along one slice, these algorithm perform the computations along different slices and exploit spatial coherence between the slices to speed up the computation.

The precision of the distance field computed using a volumetric approach is governed by the cell size in the uniform grid. Let the number of cells in the grid $k \times k \times k$, and storage overhead is $O(k^3)$. Then the error of the distance field is $\frac{\sqrt{3}}{2k}$. In comparison, for a surface distance map of size $k \times k$, the storage cost is $O(k^2)$, and the error in the distance field is $\frac{\sqrt{2}}{2k}$ in absence of any scale and shear. As the model undergoes deformation, the error bound for surface distance map is given by the function $f(\frac{1}{k})$ presented in Section 3.4. Typically, the maximum amount of deformation d_m is small, and the error in the distance field is $O(\frac{1}{k})$. As a result, surface distance maps provide a more compact representation of the distance field with similar error bounds. Conversely, our approach results in higher resolution distance fields. Current GPUs have 512MB or 1GB of video memory. It may not even be possible to store a volumetric distance at a very high memory (e.g. $(1K)^3$) on current GPUs, as it would require 8GB of memory. Furthermore, the cost of reading back a 3D distance field of $(1K)^3$ and scanning is rather high, i.e. about 16 seconds using a readback bandwidth of 500MB/sec. On the other hand, we restrict the distance field computation to the surface of a mesh and can compute a high resolution mesh at interactive rates.

Let there be m sites in each object. Then the computation cost to compute the global distance field using a volumetric approach varies between $O(mk^3)$ and $O(k^3)$. For *narrow bands*, the cost is $O(m + n_1)$ where n_1 is the number of pixels near the surface. On the other hand, the rasterization cost of computing the global surface distance map on the GPU varies between $O(mk^2 + m \log m)$ and $O(k^2 + m \log m)$. For *narrow bands*, the cost is close to $O(k^2 + m \log m)$ - as all k^2 pixels lie on the surface. A quantitative comparison of average time to compute the distance fields on deformable models is shown in figure 3.8.

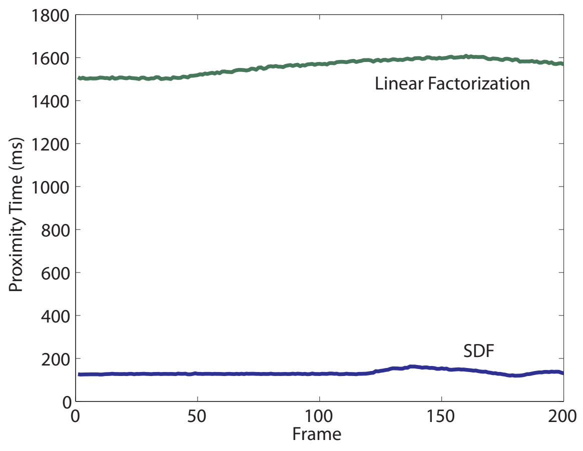


Figure 3.8: Timing comparison between our algorithm and a GPU-based volumetric distance field algorithm [SGGM06] labeled as **SDF** and **Linear Factorization** respectively: *Our algorithm is able to achieve 5–10 times speedup in proximity computation between two deforming alphabets. The scene is composed of 3.7K polygons. The surface distance field is computed at a resolution of* 512×512 *and the volumetric distance field is computed at* $180 \times 150 \times 256$. *Our algorithm is able to obtain higher accuracy in distance field computation on the surface and achieves an interactive performance of 5–10 frames per second.*

3.6.2 Limitations

Our approach has certain limitations. We compute a 2D domain triangle for each triangle in the 3D mesh. We pack all these 2D domain triangles in the texture atlas and our current packing algorithm may not be optimal. Our current approach is limited to deforming triangles with fixed connectivity. If the underlying simulation consists of objects with changing topologies, we may need to update the planar parameterization and recompute the spatial hierarchies.

Chapter 4

Fast Proximity Computation among Deformable Models using Discrete Voronoi Diagrams

Interactive simulation systems with deforming objects are used in many diverse applications, including surgical simulation, robotics, computer games, computer animation, haptics and bioinformatics. The three main components of such systems are dynamic simulation, collision detection and contact response. Different proximity queries are needed to perform each of these components. The set of proximity queries includes collision detection, separation distance and penetration depth computation. These queries are performed among different objects (i.e. inter-object queries) or within the same object (i.e. self-collision or intra-object queries). Penetration depth (PD) computation is often used to compute contact forces in penalty-based methods [HTK^{+}04, KOLM02]. Separation distances are useful in computing the repulsive forces or estimating the time of contact between moving objects in a discretized simulation [BW01, KOLM02]. Robust simulations of cloth dynamics may require penetration depth computation [BWK03] or continuous collision detection [BFA02].

The problem of fast and reliable geometric proximity queries has been extensively stud-

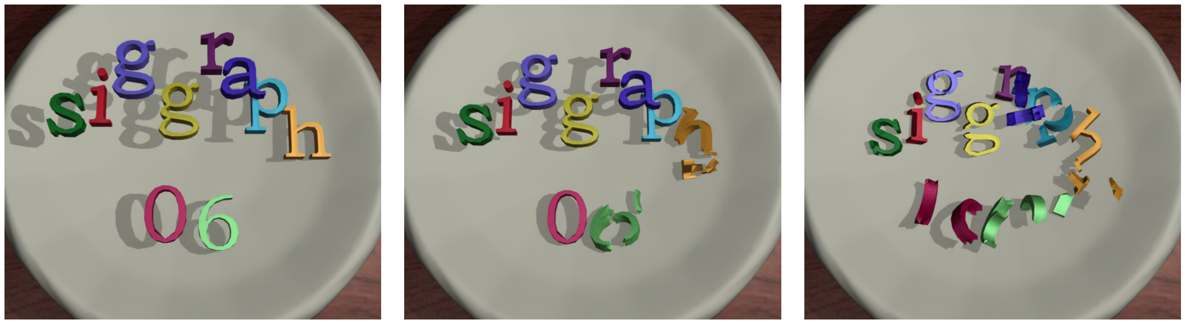


Figure 4.1: Multiple deformable models simulation with dynamic topology: *This sequence shows the positions of the objects at three time instances in a simulation. The environment initially consists of 10 deforming objects represented using* 5.5*K triangles. As the simulation proceeds, the objects break into 25 sub-objects. Our algorithm is able to perform collision and separation distance computations, including self-collisions, among dynamically generated objects within* 120 *ms on a high-end PC.*

ied. Despite the vast literature, real-time proximity queries remain one of the major bottlenecks for interactive deformable simulation [TKH+05, MHTG05]. Many existing methods are based on hierarchical representations and work well for rigid models. Several efficient collision detection algorithms have been proposed for deformable models, but they do not compute separation or penetration distances. One of the challenges in the area is to perform fast *N-body proximity queries* in scenes composed of multiple deforming objects. The Voronoi diagram is considered as one of the most powerful data structures for proximity queries. However, the application of Voronoi diagrams for proximity queries has been limited to rigid bodies that can be represented as a union of convex shapes.

In this chapter we present novel algorithms for fast proximity computation among multiple deformable models. Our approach involves no preprocessing and is applicable to all triangulated models undergoing non-rigid motion. In order to perform different proximity queries in complex environments, we present three key results:

N-body distance query: We introduce a unified approach to perform different proximity queries using *N-body distance computation*: given a set $\mathcal{P}$ of primitives, for each primitive p_i we compute the closest primitive in $\mathcal{P} \setminus \{p_i\}$. We also present efficient algorithms for continuous collision detection and local penetration depth computation based on the N-body distance query.

Voronoi-based culling: We use properties of Voronoi diagrams to perform the N-body distance query efficiently. The closest primitive to any primitive (p_i) is one of the Voronoi neighbors of p_i. Therefore, the Voronoi diagram of primitives is an efficient data structure to perform *N-body distance culling*. We use the *2nd order Voronoi diagram* because it provides information about two closest primitives at each point in space and results in a higher culling efficiency.

Fast and conservative computations using discrete Voronoi diagrams: The exact computation of continuous 3D Voronoi diagrams for general triangulated models is a hard problem. Instead, we efficiently compute discrete Voronoi diagrams using graphics hardware along uniform grids (Chapter 2) or along 2-manifolds (Chapter 3). We exploit properties of the 2nd order Voronoi diagram to derive distance error bounds that take into account discretization and sampling errors in discrete Voronoi diagrams. We use the distance bounds to efficiently compute the closest primitive at object-space precision i.e. IEEE 64-bit floating point accuracy.

4.1 Related Work

The problems of collision detection and distance computation are well studied in the literature. We refer the readers to recent surveys [Eri04, LM04, TKH$^+$05]. In this section, we briefly discuss some of the prior algorithms for deformable models.

4.1.1 N-body algorithms

Many N-body culling algorithms that reduce the number of pairwise tests have been proposed. These include algorithms based on spatial grids and octrees [Eri04], and 3D sorting based on tight fitting axis-aligned bounding boxes [CLMP95]. More recently, GPU-based algorithms [GRLM03, GKJ$^+$05] use occlusion queries to compute potentially colliding sets of overlapping objects. Most of these algorithms have been limited to N-body collision detection and their culling performance varies based on the relative configuration of the objects.

4.1.2 Bounding volume hierarchies

Bounding volume (BV) hierarchies are widely used for collision detection and separation distance computation. The choice of BVs include simple shapes such as spheres or AABBs or tight-fitting volumes such as oriented bounding boxes (OBBs), discretely oriented polytopes (k-DOPs) or swept sphere volumes [Eri04]. These hierarchies are precomputed for rigid models and dynamically updated or recomputed for deformable models [LAM01, vdB97]. However, these hierarchies may not be able to perform significant culling in close proximity configurations or for self-proximity queries. Thus, they can result in a high number of false positives.

4.1.3 Deformable model collision detection

Many specialized algorithms have been proposed to perform collision queries on deformable models. These include GPU-based algorithms [KP03, GKJ$^+$05] for inter-object or intra-object collisions. Other methods for self-collisions are based on the "curvature test" [VT00] and these can be combined with BV hierarchies. Teschner et al. [THM$^+$03] use spatial hashing techniques to check for inter-object collisions and self-collisions. All of these algorithms perform only collision queries.

4.1.4 Distance and penetration queries

Most prior distance and penetration computation algorithms are designed for pairwise inter-object separation distance queries. These include algorithms based on hierarchies of spheres [Qui94] or rectangular swept spheres [LGLM00] or different model types [JC04]. Techniques have been proposed to update the hierarchies incrementally for deformable models [SL00].

Distance Fields: 3D discrete distance fields can be efficiently computed using graphics hardware [FG05, SPG03, SOM04, SGGM06]. The discrete distance fields can be used to perform inter-object proximity queries between deformable models at image-space resolution [HZLM02, SGGM06]. However, there are two limitations of current algorithms based on distance fields. Firstly, currently algorithms can take many seconds to compute high resolution distance fields in 3D [SGGM06, TKH$^+$05] and do not provide interactive performance. Secondly, they perform approximate queries and give no bounds on the errors.

Penetration Depth Computation: Efficient penetration depth computation algorithms have been proposed for rigid polyhedral models [KOLM02], but they involve considerable preprocessing. Many approximate PD computation algorithms for deformable models are based on GPU-based computations [HZLM02, RL06], precomputed distance fields [FL01] or spatial hashing [HTK$^+$04].

4.1.5 Voronoi diagrams

The Voronoi diagram is regarded as a powerful proximity data structure in computational geometry [OBS92]. In relation to 3D proximity queries, external Voronoi regions have been used to perform collision and distance queries between rigid objects that can be represented as the union of convex polytopes [LC91b, EL01, Mir98, KS00]. These algorithms have been implemented within different proximity query packages such as I-COLLIDE, V-CLIP and SWIFT++. However, it is difficult to extend these algorithms to general non-convex or deformable models.

4.2 N-body Distance Query

Our goal is to perform both inter-object and intra-object queries. The inter-object queries are performed among different objects. The intra-objects queries are performed between the non-adjacent features of an object. Two given features are classified as adjacent if they share either a common edge or a vertex.

We make no assumptions about the motion of the objects and these scenes may include breaking objects or models with changing topologies. In this section, we introduce the "N-body distance query" and use this formulation to perform different proximity queries.

4.2.1 Notation and Terminology

We first describe the notation used in the paper. Given a simulation environment consisting of n deforming objects, $O_1, O_2, \ldots, O_n$, we assume that each object has been triangulated and we use the symbol f^i to denote the boundary features such as the triangles. For example, the boundary of O_i is represented as $\{f_1^i, f_2^i, \ldots, f_{n_i}^i\}$, where n_i is the number of features in O_i. The position of these features is updated during each step of the simulation.

N-body Distance Queries: Given m sites, $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, where the sites may correspond to the objects O_i or their features f_j^i, let $d(p_i, p_j)$ denote the Euclidean distance between p_i and p_j. The N-body distance query computes the closest site in $\mathcal{P} \setminus \{p_i\}$ to each p_i. A site, p_k, is the closest site to p_i, if $d(p_i, p_k) \leq d(p_i, p_l)$ for every $l \neq i$, where $k \neq i$. Later, in Section 4.3 we present Voronoi-based algorithms to perform the N-body distance query efficiently.

It is obvious that the N-body distance query can be used to perform separation distance queries. We now present algorithms for efficient collision detection and penetration depth

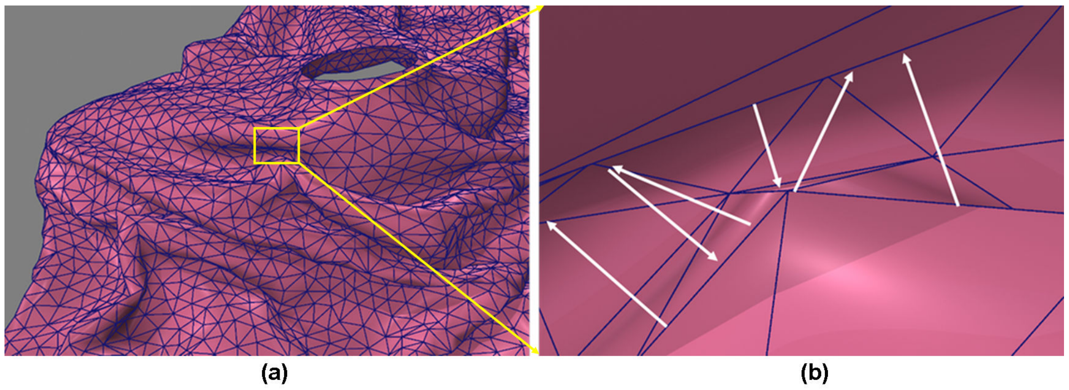


Figure 4.2: N-body distance query: *In this cloth mesh, we compute the closest non-adjacent triangle for every triangle in the mesh. The white arrows highlight the closest triangle to each triangle.*

computation based on N-body distance query.

4.2.2 Collision Detection

The collision query checks whether two objects intersect and returns all pairs of overlapping features. We consider two kinds of collision queries: discrete and continuous. The *discrete* collision query is performed at a specific or discrete instance of the simulation. The discrete collision detection query is a special case of the N-body distance query, in which we check whether any $d(p_i, p_k)$ is zero. Eventually, we report all the intersecting sites.

In *continuous* collision detection (CCD), we interpolate the motion between features from two successive instances of the simulation. The CCD query computes the first time of contact between any two primitives within the time interval. The query is efficiently performed by culling away primitive pairs whose swept *volumes* do not overlap [RKLM04]. As a result, CCD computation reduces to a volumetric collision detection problem between the swept volumes of the primitives. We use the N-body distance query to check for *volumetric* overlaps among the primitives. We first compute tight bounding prisms that enclose the swept volumes of the primitives. Given a pair of primitives, we perform the volumetric overlap test using the signed distance function between the bounding prisms of the primitives (see figure 4.3). The signed distance function of the bounding prisms represents the interior, and exterior regions of the prisms. By convention, the signed distance values in the interior of an object/prism are negative. Specifically, for any two primitives p_i, p_j, we use the following properties of the signed distance function to perform volumetric overlap culling:

- *Perform elementary CCD tests* between the primitives if there exists a point such that the signed distances of the point to p_i and to p_j are both ≤ 0.

- *Do not perform elementary CCD tests* between the primitives if there exists no point whose signed distances to p_i and p_j are both ≤ 0.

The above formulation corresponds to computing a distance query between the two primitives using signed Euclidean distance functions. Our approach can be directly extended to n primitives by performing N-body distance queries using the 2$^{\text{nd}}$order Voronoi diagram of the n primitives.

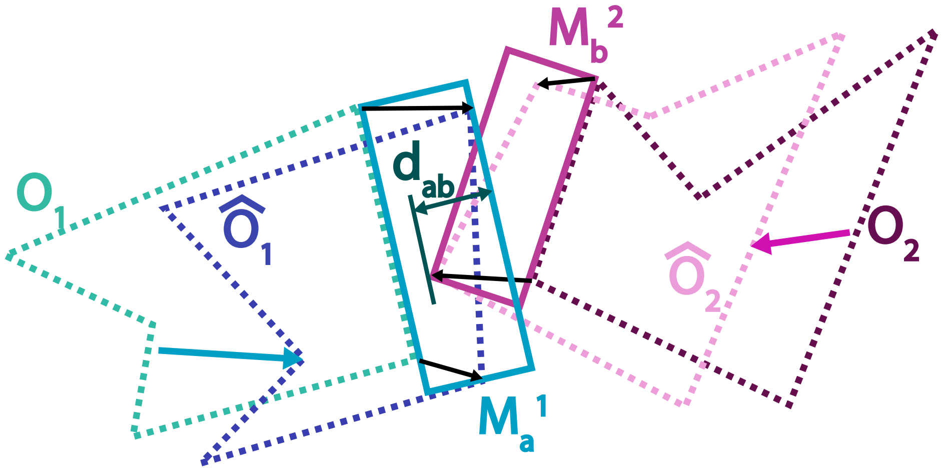


Figure 4.3: Continuous Collision Detection for two polygons O_1 and O_2: *Two polygons, O_1 and O_2, move to positions $\widehat{O}_1$ and $\widehat{O}_2$ at time $t + \Delta t$. The volume swept by a pair of features is bounded by the prisms, M_a^1 and M_b^2, respectively. A conservative CCD check is performed by volumetric collision detection between M_a^1 and M_b^2. We compute the signed distance between the prisms, shown as d_{ab}. Eventually, we use the N-body distance query to compute the signed distance functions for all the prisms.*

4.2.3 Penetration Depth (PD) Computation

The PD query measures the extent of overlap between two intersecting objects. We assume O_i and O_j are orientable 2-manifolds in the region of penetration. This guarantees that we have a well defined 'interior' for each penetrating object. We define PD as the minimum translational distance needed to make the two objects disjoint [DHKS93]:

$$\min\{\| \boldsymbol{T} \| | \ interior(O_i + \boldsymbol{T}) \ \cap \ O_j = \emptyset\},$$

where $\boldsymbol{T}$ is the translation vector computed by the algorithm. However, exact computation of PD between two polyhedral models is a global problem and cannot be solved using any 'divide-and-conquer' or localized approach [KOLM02]. Its worst complexity can be as high as $O(n_i^3 n_j^3)$. As a result, we restrict ourselves to computing an approximate local PD between deforming objects.

We compute the local PD between two objects O_i and O_j based on the N-body distance query. The same approach can also be used to compute self-penetrations. The local PD is computed among all locally overlapping features. We use the N-body distance query described in Section 4.2.2 to compute the overlapping features. Next, we use the orientation and connectivity information among the overlapping features to compute all of the features of O_i that are inside O_j and vice-versa. We denote these features as $\overline{f_a^i}$, $a = 1, \ldots, k$ and $\overline{f_b^j}$, $b = 1, \ldots, l$ (see figure 4.4).

Our PD algorithm proceeds in two stages. We first use a greedy strategy to estimate the direction of the translation vector and then compute the extent of penetration along that direction.

1. Penetration direction computation: We consider all overlapping features and perform the N-body distance query among them. For each feature, $\overline{f_a^i}$, we compute the closest feature among $\overline{f_b^j}$'s and represent the closest feature pairs as $(\overline{f_a^i}, \overline{f_a^j})$. Similarly, we compute the closest feature pairs $(\overline{f_b^j}, \overline{f_b^i})$. Given these $k+l$ closest feature pairs, we compute the distances between them and use the pair that represents the maximal distance. We use the direction of the maximal distance feature pair as the direction of $\boldsymbol{T}$.

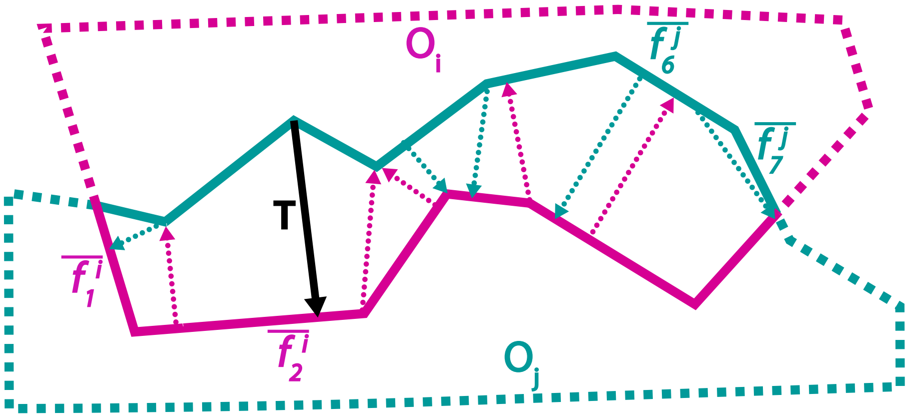


Figure 4.4: Local PD Computation for two polygons O_i and O_j : *Dotted arrows represent direction vectors showing the separation distance between pairs of overlapping features of O_1 and O_2. The maximum separation distance is shown by a thick black arrow and is the local penetration direction* $\mathbf{T}$.

2. Penetration depth computation: Given the direction of $\boldsymbol{T}$, we compute its magnitude by projecting all of the overlapping features onto $\boldsymbol{T}$. The maximal width of the projected features along $\boldsymbol{T}$ gives us the value for penetration depth.

We note that $\boldsymbol{T}$ is the locally optimal direction if the overlapping features of the objects are connected and convex. Therefore, $\boldsymbol{T}$ can be a good estimate for the penetration direction when the intersecting region is convex and has a small width.

4.3 Voronoi-based Culling for Proximity Queries

In this section we present our Voronoi-based culling algorithm to perform the N-body distance query. We first give an overview of 2nd order Voronoi diagrams and show how they can be used for proximity computations. Next, we describe our N-body distance culling algorithm based on discrete Voronoi diagrams.

4.3.1 2nd Order Voronoi diagrams

We first introduce some of the terminology related to Voronoi diagrams. Two sites are *independent* if there does not exist a path of edges on a triangle mesh connecting them. Given a set of sites $\mathcal{P}$ in domain $\mathcal{D}$, and a subset $\mathcal{T}$ of $\mathcal{P}$, with $|\mathcal{T}| = k$, the *k-th order Voronoi region* is the set of points closer to all sites in $\mathcal{T}$ than to any other site:

$$\mathcal{V}^k(\mathcal{T}|\mathcal{P}) = \{\mathbf{q} \in \mathcal{D} \mid d(\mathbf{q}, p_i) \leq d(\mathbf{q}, p_j) \, \forall \, p_i \in \mathcal{T}, p_j \in \mathcal{P} \setminus \mathcal{T}\}.$$

The *k-th order Voronoi diagram* is a partition of $\mathcal{D}$ into k-th order Voronoi regions:

$$\mathrm{VD}^k(\mathcal{P}) = \bigcup_{p_i \in \mathcal{P}} \mathcal{V}^k(\mathcal{T}, \mathcal{P}) \; , \; |\mathcal{T}| = k.$$

The standard Voronoi diagram is the same as $\mathrm{VD}^1(\mathcal{P})$. We are specifically interested in the 1st and 2ndorder Voronoi diagrams, denoted as $\mathrm{VD}^1(\mathcal{P})$ and $\mathrm{VD}^2(\mathcal{P})$. A 1storder Voronoi region $\mathcal{V}^1(p_i|\mathcal{P})$ contains points closest to site p_i, and the 2ndorder Voronoi region $\mathcal{V}^2(\{p_i, p_j\}|\mathcal{P})$ contains points closest to two sites p_i and p_j (see figure 4.5).

The 2nd order *governor set* of a point is the set of two closest sites. For a point $\mathbf{q} \in \mathcal{D}$, let the two closest sites be $\{p_i, p_j\}$, i.e. $\mathbf{q} \in \mathcal{V}^2(\{p_i, p_j\}|\mathcal{P})$. Then the 2nd order governor set of $\mathbf{q}$ is denoted as $\mathcal{G}^2(\mathbf{q}|\mathcal{P}) = \{p_i, p_j\}$. For a site p_i, the 2nd order governor set is given as $\mathcal{G}^2(p_i|\mathcal{P}) = \bigcup_{\mathbf{q} \in p_i} \mathcal{G}^2(\mathbf{q}|\mathcal{P})$.

4.3.2 PNS Computation Using 2nd Order Voronoi Diagrams

We use the 2ndorder Voronoi diagram to compute the *potentially neighboring set* (PNS) for each site. The PNS of a site p, denoted $\mathrm{PNS}(p|\mathcal{P})$, is a subset of $\mathcal{P}$ such that a site in $\mathrm{PNS}(p|\mathcal{P})$ is closer to p than any site in $\mathcal{P} \setminus \mathrm{PNS}(p|\mathcal{P})$. To perform the N-body distance query, we compute a tight PNS for each site. The 2ndorder Voronoi diagram provides the

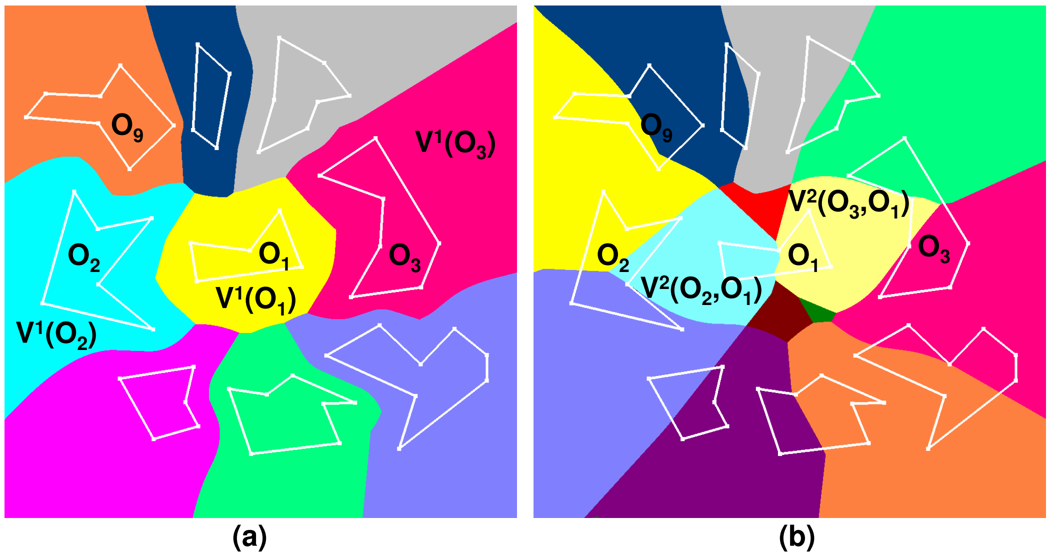


Figure 4.5: The 1st and 2ndorder Voronoi diagrams of 9 polygons (denoted as O_i) in a plane. *(a) The 1storder Voronoi diagram: Each color represents the set of points closest to one of the polygon. In this case, O_1 has 8 1storder Voronoi neighbors. The PNS of O_1 is all the objects which share a Voronoi edge, i.e. all other 8 objects. (b) The 2ndorder Voronoi diagram: Each color represents a region with same two closest objects. O_1 is contained completely inside two 2ndorder Voronoi regions. Therefore, PNS of $O_1 = \{O_2, O_3\}$. We get a tighter PNS with 2ndorder Voronoi diagram.*

two closest sites for each point in space. At a point that lies on a given site, p, the closest site is trivially p, thus p is ignored in $\mathrm{PNS}(p|\mathcal{P})$. We use the 2ndorder Voronoi diagram and the 2ndorder governor set to compute a tight PNS. Then we have the following property (illustrated in figure 4.5):

Lemma 4.1 (PNS Computation). *Given a set of independent sites $\mathcal{P}$, the PNS of a site p_i is given by $\mathrm{PNS}(p_i|\mathcal{P}) \supseteq \mathcal{G}^2(p_i|\mathcal{P})$. The closest site(s) to p_i is (are) contained in $\mathrm{PNS}(p_i|\mathcal{P})$.*

Proof. Let p_j be the closest site to p_i. Since p_j is closest to p_i then there is a point $\mathbf{r} \in p_i$ s.t. the two closest sites to $\mathbf{r}$ are p_i and p_j. By definition, $\mathcal{G}^2(\mathbf{r}|\mathcal{P}) = \{p_i, p_j\} \Rightarrow p_j \in \mathrm{PNS}(p_i|\mathcal{P})$. □

Lemma 4.1 provides a culling scheme to compute the closest sites for a given set of sites. In addition to a tighter culling scheme, the 2ndorder Voronoi diagram also provides tight bounds on the separation distance, and we use them to perform conservative PNS computation using discrete Voronoi diagrams in Section 4.3.4.

N-body distance query: We use the 2ndorder Voronoi diagram to perform the N-body query. Given n independent sites $\mathcal{P}$, we compute $\mathrm{VD}^2(\mathcal{P})$ and the 2ndorder governor set of each site p_i. This computation gives $\mathrm{PNS}(p_i|\mathcal{P})$ for each site. We perform pairwise distance computations between p_i and each site in $\mathrm{PNS}(p_i|\mathcal{P})$ to compute the closest site to p_i. A key issue is defining an appropriate set of sites for inter-object and intra-object queries. More details are given in Section 4.4.

4.3.3 Discrete Voronoi Diagram Computation

In the previous subsection, we showed that the PNS for each site can be efficiently computed based on the 2ndorder Voronoi diagram. However, exact computation of the Voronoi diagram of triangulated models is a hard problem due to its algebraic and combinatorial complexity. In this section, we introduce discrete approximations of Voronoi diagrams and compute them efficiently using the graphics hardware.

Given a finite set of point samples $\widetilde{\mathcal{D}}$ in domain $\mathcal{D}$, and a set of sites $\mathcal{P}$, the k-th order *discrete Voronoi diagram* (DVD) is a partition of the point samples onto discrete k-th order Voronoi regions, and is denoted as $\widetilde{\mathrm{VD}}^k(\mathcal{P})$. For a set $\mathcal{T}$ of k sites, the k-th order discrete Voronoi region is a finite set of points which are closest to all sites in $\mathcal{T}$ than to any other site. The 1st and 2ndorder discrete Voronoi regions are obtained by using $k = 1$ and $k = 2$, and denoted by $\widetilde{\mathcal{V}}^1(p_i|\mathcal{P})$ and $\widetilde{\mathcal{V}}^2(\{p_i, p_j\}|\mathcal{P})$, respectively.

GPU-based DVD Computation: The discrete Voronoi diagram for a triangulated model can be efficiently computed along a uniform 3D grid $\widetilde{\mathcal{D}}$ using depth-buffered graphics hardware [SGGM06, SPG03]. The 3D domain is discretized into a set of 2D slices, and a discrete 2D distance field is computed for each slice by rasterizing the distance functions of the primitives. Specifically, we rasterize the distance functions corresponding to each vertex, edge and triangular face of the object. The distance values are stored in the depth buffer and the closest site identifier is computed in the color buffer. Together, these two buffers provide us with the discrete 1storder Voronoi diagram and we read it back to the CPU.

In addition to the 1storder Voronoi diagram of triangulated models, we compute the 2ndorder Voronoi diagram along the points that belong to a site. We first rasterize all the sites in $\mathcal{P}$ into a uniform grid. Each triangle is clipped to the volume between two 2D slices and is scan converted using graphics hardware [HZLM02]. The distance computations to a site p_i are performed on grid points belonging to $\mathcal{P} \setminus \{p_i\}$. We compute $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ in the color buffer of the graphics hardware. The depth buffer stores the distance values to the second closest site. We read back the color and depth buffers from the GPU to the CPU and use them to compute the PNS. However, each site p_i is sampled (rasterized) at a finite set of points $\mathcal{Q}$ on the uniform grid. Finally, we compute the 2ndorder governor sets for all points in $\mathcal{Q}$ using $\widetilde{\mathrm{VD}}^2(\mathcal{P})$.

The $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ computed using graphics hardware is not accurate and can have errors due to under-sampling [HZLM02, SGGM06]. We first list the sources of under-sampling errors and present our approach to compute a conservative PNS in Section 4.3.4.

1. Discretization of Sites: The grid $\mathcal{Q}$ only consists of a finite number of points. The point on a site corresponding to the minimum separation distance may not get sampled on the grid. As a result, we may not compute the correct separation distance.

2. Discretization of the Voronoi Diagram: The Voronoi region of the closest site may not get sampled on the uniform grid. Therefore, $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ may return an incorrect closest site.

3. GPU Precision: Current GPUs support 32-bit floating point precision for distance computation, and 24-bit fixed point precision for distance comparisons on depth buffer. These can lead to precision errors in the distance values.

4.3.4 Conservative PNS Computation using Distance Bounds

We present an approach to compute a conservative PNS using bounds on the distance values computed using GPUs. First we define an *approximate separation distance*, which is computed using the discrete Voronoi diagram as described above. The accuracy in the approximation is given by the image-resolution used for second order Voronoi computation. Given a discrete Voronoi diagram $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ and a finite set of points $\mathcal{Q}$ on a site p_i, the approximate separation distance of p_i, denoted $\widetilde{\mathrm{SD}}(p_i)$, is the minimum of the distance values from $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ for all points in $\mathcal{Q}$. We now present our approach to compute the bounds on the exact separation distance $\Theta(p_i)$ from the approximate separation distance $\widetilde{\mathrm{SD}}(p_i)$. Our exact

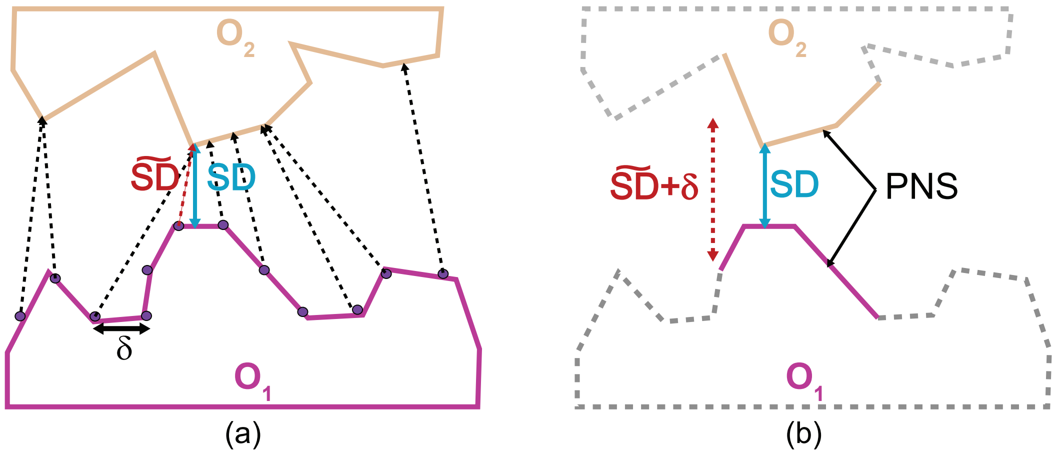


Figure 4.6: Conservative PNS using discrete Voronoi diagram: *Given 2 objects O_1 and O_2. (a) O_1 is sampled at a finite set of points. The closest points on O_2 are shown using dotted vectors. SD is the exact separation distance $\Theta(O_1)$, $\widetilde{SD}$ is the approximate separation distance $\widetilde{\mathrm{SD}}(O_1)$. δ is the distance between 2 adjacent samples. (b) $\widetilde{SD}+\delta$ is the bounded separation distance for O_1. First we compute the features on O_1 that are within distance $\widetilde{SD}+\delta$ to O_2. For these features of O_1, we compute features of O_2 that are within a distance $\widetilde{SD}+\delta$. These features of O_2 constitute the PNS of O_1.*

distance computation algorithm exploits the fact that Euclidean distance field is a continuous scalar field. Moreover, the change in distance to the closest site between two adjacent points on the uniform grid is bounded by the distance between the two points. We use this property to compute a bound on separation distances between two sites computed using the discrete

Voronoi diagram. Let δ_1 be the diagonal length of a cell in the uniform grid $\widetilde{\mathcal{D}}$, and δ_2 be the error due to limited GPU precision (typically $\delta_2 \ll \delta_1$). Let $\delta = \frac{\delta_1}{2} + \delta_2$ represent the total error in discrete Voronoi diagram computation.

Lemma 4.2 (Distance Bound using DVD). *Given the approximate separation distance, $\widetilde{\mathrm{SD}}(p_i)$, the exact separation distance $\Theta(p_i)$ is bounded by $\widetilde{\mathrm{SD}}(p_i) - \delta \le \Theta(p_i) \le \widetilde{\mathrm{SD}}(p_i) + \delta$.*

Proof. We first prove that the change in distance to the closest site between two adjacent points on the uniform grid is bounded by the distance between the two points. Let $D(\mathbf{q}, \mathcal{P})$ be the value of the distance field of a set of sites $\mathcal{P}$ at a point $\mathbf{q} \in \mathcal{D}$. $D(\mathbf{q}, \mathcal{P}) = \min_{p_i \in \mathcal{P}}(d(\mathbf{q}, p_i))$. We need to prove for two points $\mathbf{q}$, $\mathbf{r}$ $|D(\mathbf{q}, \mathcal{P}) - D(\mathbf{r}, \mathcal{P}) \le |\mathbf{q} - \mathbf{r}|$. We proceed in two cases: (a) $\mathbf{q}, \mathbf{r}$ lie in same Voronoi region, (b) $\mathbf{q}, \mathbf{r}$ lie in separate Voronoi regions.

The distance field is continuous scalar field for all $\mathbf{q} \in \mathcal{D}$. Further, the gradient of the distance field is unity at all points where defined, i.e $|\nabla D(\mathbf{q}, \mathcal{P})| = 1$ at all points in the interior of a first order Voronoi region. The gradient is undefined at all points on the boundary of a Voronoi region.

(a) $\mathbf{q}, \mathbf{r}$ lie in same Voronoi region. Then $\nabla D(\mathbf{p}, \mathcal{P})$ is defined at all points in the line segment joining $\mathbf{q}$ and $\mathbf{r}$. Then $|\nabla D(\mathbf{p}, \mathcal{P}) \cdot (\mathbf{q} - \mathbf{r})| \le 1 \Rightarrow |D(\mathbf{q}, \mathcal{P}) - D(\mathbf{r}, \mathcal{P})| < |\mathbf{q} - \mathbf{r}|$.

(b) $\mathbf{q}, \mathbf{r}$ lie in different Voronoi regions. Then $\nabla D(\mathbf{p}, \mathcal{P})$ is not defined at all points where the line segment joining $\mathbf{q}$ and $\mathbf{r}$ intersects a Voronoi boundary. We proceed along segments contained inside each Voronoi region. Since distance field is continuous, the values are equal across a Voronoi boundary. This leads to above result. $\square$

Lemma 4.2 gives tight lower and upper bounds on the exact separation distance for a site. These bounds are used to cull objects or features and compute a PNS. Thus, we are able to address the last two issues of under-sampling on a discrete grid. In order to address the first issue, we use the idea of growing a site by taking its Minkowski sum with a pixel [GKJ+05]. When we rasterize the Minkowski sum, we ensure that every point on a site gets sampled. In Section 4.4, we use these queries to perform accurate inter-object and intra-object queries.

4.4 Proximity Queries using Discrete Voronoi Diagrams

In this section, we present our overall approach to compute inter-object and intra-object queries. Our algorithm proceeds in three stages, as shown in figure 4.7. We first use an AABB based culling approach to compute a very conservative PNS for each object. Next, we present algorithms to perform inter-object or intra-object proximity queries using Voronoi-based culling. Finally, we perform exact tests between the triangle primitives in the conservative PNS.

4.4.1 Stage I: AABB Culling

In this stage we compute the AABBs of each object and perform the N-body distance query between the AABBs, by computing overlaps along the three axes. For example, we compute AABB$_i$ for O_i and use that AABB to compute a conservative upper bound on the separation distance of O_i. As a result, all AABBs whose distances are more than this bound, do not belong to $\mathrm{PNS}(O_i)$. The projections of AABBs are sorted along each axis to compute a sequence of intervals along each axis [CLMP95]. If the projection of AABB$_i$ does not overlap with any other interval, we compute the closest AABB along that axis. Otherwise, we consider all other AABBs that overlap with the projection of AABB$_i$ and use the one with maximal overlap. This computation is repeated along the three axes to compute the potentially closest AABB to AABB$_i$. We compute an upper bound to the separation distance for each O_i by computing the maximal distance between the vertices of AABB$_i$ and its closest AABB. For each object O_i, all objects that are at a distance less than this conservative distance bound constitute a conservative bound to an object level PNS of O_i.

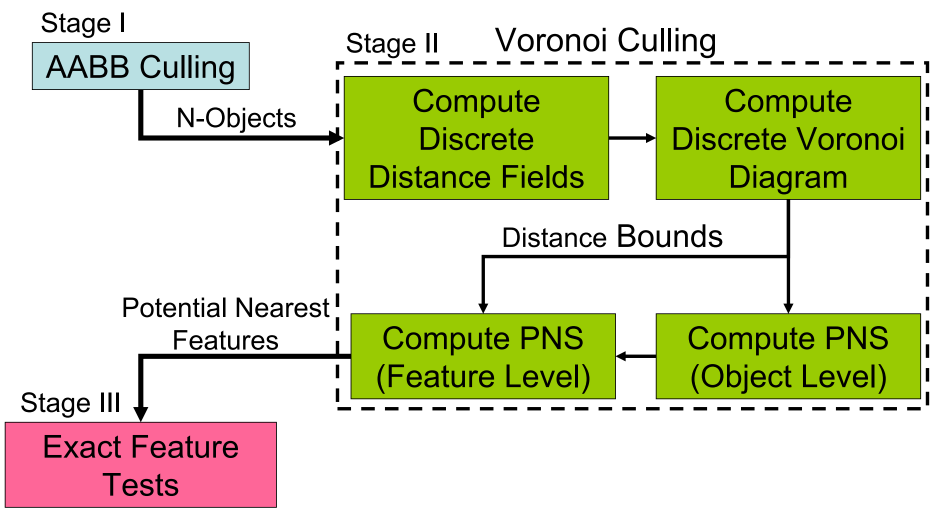

Stage I
AABB Culling
N-Objects
Stage II
Voronoi Culling
Compute Discrete Distance Fields
Compute Discrete Voronoi Diagram
Distance Bounds
Potential Nearest Features
Compute PNS (Feature Level)
Compute PNS (Object Level)
Stage III
Exact Feature Tests

Figure 4.7: Overall proximity computation algorithm: *Our proximity computation algorithm proceeds in three stages: AABB-based culling, Voronoi culling and exact distance tests on the PNS.*

4.4.2 Stage II: Voronoi-based Culling

We use the distance bound from AABB culling as an upper bound to localize distance field computation. The distance computation for object O_i is performed in a banded region around O_i. The width of this band is the maximum distance between an object and its potential neighbors. For each object O_i, we compute the set of objects O_j such that O_i belongs to $\mathrm{PNS}(O_j)$, and use the maximum separation distance as a bound on the width of the banded region of O_i. We use these bands to narrow the grid region for discrete Voronoi diagram computation. Eventually, we use the discrete Voronoi diagram to compute a tighter PNS for inter-object and intra-object proximity queries.

Inter-Object Proximity Queries

The set of sites is the set of objects $\mathcal{P} = \{O_i, \ldots, O_n\}$. Our algorithm for inter-object proximity queries proceeds in two phases. First we compute a tighter object level PNS for each object. Secondly, we perform PNS computations at feature level to compute a set of potentially closest features between a pair of objects.

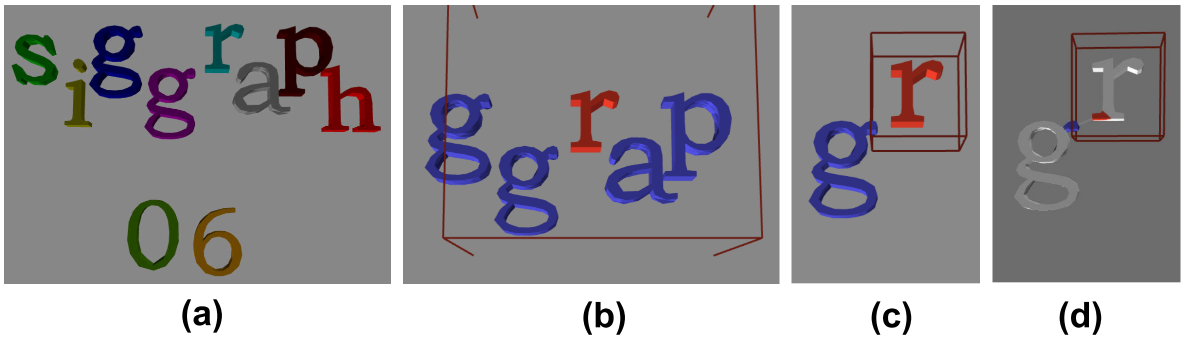


Figure 4.8: Application of our proximity query algorithm to a simulation with 10 objects: *(a) Position of* 10 *deforming objects - 'siggraph* 06*' (with the bowl removed). (b)-(d) Stages in PNS computation. The red wireframe represents conservative bound on the separation distance between 'r' and other letters. This bound is used to compute the PNS of 'r'. (b) The object level PNS of letter 'r' after stage I that uses AABB-based culling. (c) Object level PNS computed using our 2nd order DVD based algorithm. (d) Zoomed view of feature level PNS between 'r' and 'g'. The exact distance tests are performed between red triangles in 'r' and blue triangles in 'g'. Total number of pairs in feature level PNS=*12*K. Total computation time is around* 60 *ms per frame.*

Object-level PNS computation: We compute $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ using GPUs. Next, we compute an upper bound on the separation distance of each object using Lemma 4.2. Let $D_u(O_i) = \widetilde{\mathrm{SD}}(O_i) + \delta$ denote the upper bound on the separation distance for O_i. The $\mathrm{PNS}(O_i|\mathcal{P})$ of an object O_i is computed as a set of objects, whose distance to O_i is less than $D_u(O_i)$. We expand the AABB of O_i by $D_u(O_i)$ along each axis and reduce the distance query to a collision query between the expanded AABB of O_i and AABB of O_j. The overlap tests are efficiently performed using the sorted intervals computed in Stage I.

Feature-level PNS computation: Given a feature f_k^i in object O_i, our goal is to compute the minimum distance to all features in $\mathrm{PNS}(O_i|\mathcal{P})$, but ignore the features on O_i as part of this computation. During this stage, we compute the feature level PNS for a subset of features in object O_i, as explained below. We use the upper bound on the separation distance of object O_i to cull away features in O_i that do not contribute to closest site computation.

We compute $\widetilde{\mathrm{VD}}^2(\mathcal{P})$ for all the points on f_k^i and use it to compute the approximate separation distance of f_k^i, denoted $\widetilde{\mathrm{SD}}(f_k^i)$, to its closest feature. Based on Lemma 4.2, the lower bound on the separation distance of f_k^i is given as $D_l(f_k^i) = \widetilde{\mathrm{SD}}(f_k^i) - \delta$. We cull away a feature f_k^i, if $D_l(f_k^i) > D_u(O_i)$, as the closest object to f_k^i is further away than the separation distance between O_i and $\mathcal{P} \setminus O_i$. Finally, for each feature f_k^i with $D_l(f_k^i) \leq D_u(O_i)$, we compute a set of features in $\mathrm{PNS}(O_i|\mathcal{P})$ which are at a distance less than the separation distance of O_i. This is illustrated in figure 4.6. This computation is performed by expanding the AABB of each feature by $D_u(O_i)$ and performing overlap tests as mentioned in Stage I. In the end we compute the PNS for each object and its features.

Intra-Object Queries

Our goal is to perform the N-body distance query on all the features of an object. Given a feature, we ignore its adjacent features and compute the closest among the non-adjacent features. In order to classify the features into adjacent and non-adjacent, we define the notion of *1-ring* and *2-ring* for each feature, f_k^i. The 1-ring, denoted as $\mathcal{I}(f_i)$, is the set of features that are adjacent to f_k^i (i.e share a vertex with f_k^i). The 2-ring is the set of features that are adjacent to the features in the 1-ring, excluding f_i and $\mathcal{I}(f_i)$.

We first compute the minimum distance between f_k^i and the set of features in the 2-ring of f_k^i. This minimum distance provides an upper bound to the separation distance of f_k^i. This computation can be performed in $O(n_i)$ time for all the features in the deforming object, where n_i is the number of features in the object.

Our next goal is to refine the upper bound computed using the 2-ring based on 2ndorder Voronoi diagrams. The set of sites is the set of features in an object, $\mathcal{P} = \{f_1^i, \ldots, f_{n_i}^i\}$. Then f_k^i and $\mathcal{P} \setminus \mathcal{I}(f_k^i)$ are mutually independent sets. We perform proximity computations on f_k^i by using the discrete Voronoi diagram $\widetilde{\mathrm{VD}}^2(\mathcal{P} \setminus \mathcal{I}(f_k^i))$ and compute the PNS, $\mathrm{PNS}(f_k^i|\mathcal{P} \setminus \mathcal{I}(f_k^i))$. This process is repeated for all the features f_k^i. In practice, we do not compute $O(n_i)$ 2ndorder Voronoi diagrams. Rather, we store the adjacency information of each feature in a texture. For a grid cell on a feature f_k^i, we perform vector comparisons on programmable graphics hardware to avoid distance computations to $\mathcal{I}(f_k^i)$. This computes the discrete Voronoi region $\widetilde{\mathcal{V}}^2(\mathcal{P} \setminus \mathcal{I}(f_k^i))$ at all points on a feature f_k^i, and the approximate separation distance $\widetilde{\mathrm{SD}}(f_k^i|\mathcal{P} \setminus \mathcal{I}(f_k^i))$ is computed using the distance values at these points. An upper bound $D_u(f_k^i)$ on the separation distance of feature f_k^i is computed from the approximate separation distance using Lemma 4.2. Eventually all non-adjacent features, f_l^i, whose distance to f_k^i is less than $D_u(f_k^i)$ are added to $\mathrm{PNS}(f_k^i|\mathcal{P} \setminus \mathcal{I}(f_k^i))$.

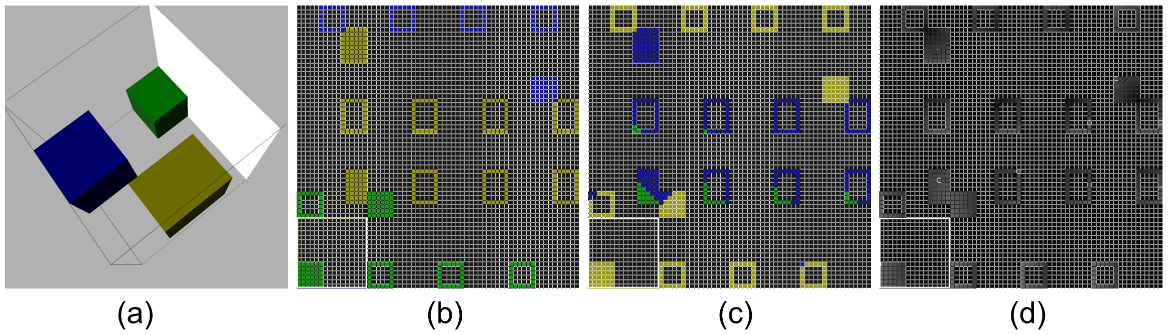


Figure 4.9: Computation of 2nd order Discrete Voronoi Diagram (DVD) on GPU*: (a) A scene with* 3 *cuboids. Each cuboid is one site. The DVD is computed on a grid of resolution* $16 \times 16 \times 16$. *The first slice is shown in white. (b)-(c) The flat rendertexture during various stages of DVD computation. The rendertexture stores all slices of the 3D DVD computation. Each slice is stored as one tile of size* 16×16, *the rendertexture has* 4×4 *tiles. The tile corresponding to first slice is shown in white outline. The DVD is computed at pixels lying on the boundary of a site (b) The 1st order DVD obtained by rasterizing the boundary of each site. (c) The 2nd closest site at each pixel on boundary of a site. The combination of (b) and (c) gives the 2nd order DVD. In our implementation the two closest sites are stored in red-green color channels of the rendertexture. (d) The depth buffer corresponding to the distance to 2nd closest sites. This rendertexture is read back to the CPU, which scans the boundary (non-zero) pixels to compute the approximate separation distance for each site.*

4.4.3 Stage III: Exact Proximity Tests

Given a feature f^i, we perform exact queries between f^i and the features in $\mathrm{PNS}(f^i)$. In order to perform discrete collision detection or penetration depth computations, we check whether two triangles overlap. In order to perform continuous collision test between two triangles whose prisms overlap, we perform 15 elementary tests described in [BFA02]. We

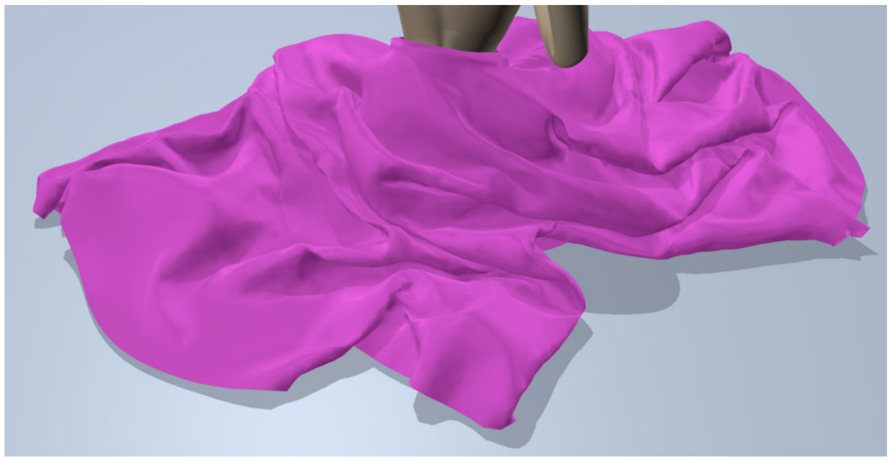


Figure 4.10: Skirt cloth simulation: *The cloth is modeled using* $12.5K$ *triangles. Our proximity computation algorithm is able to perform the N-body distance query at object-space precision within* $400 - 600$ *ms.*

use the triangle-triangle distance computation algorithm described in [LGLM00] to compute the separation distance between the primitives. We also compute the local penetration depth between the overlapping features.

4.5 Implementation and Performance

In this section we describe the implementation of our N-body distance query algorithm and highlight its application to perform various proximity queries between multiple deformable models.

4.5.1 Implementation

We have implemented our algorithm on a PC running Windows XP operating system with an AMD Athlon 4800 X2 CPU, 2GB memory and an NVIDIA GeForce 7800 GPU . We used OpenGL as the graphics API and Cg language for implementing the fragment programs. The discrete Voronoi diagram and discrete distance field are computed using a flat 2D render texture with 32-bit floating point precision. The 2nd order DVD is computed only at pixels

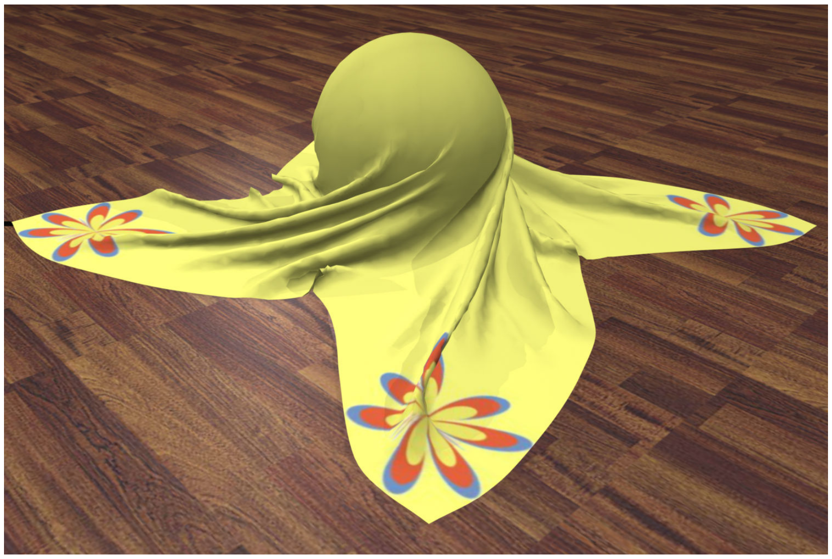


Figure 4.11: Cloth-Sphere simulation: *The cloth mesh is composed of* $15K$ *triangles and has a high number of triangles in close proximity. As the simulation progresses, the cloth wraps around the sphere and the simulation generates many complex folds. Our algorithm is able to perform continuous self-collision detection among all the triangles within* 800 *msec.*

that lie on an object using 2 rendering passes. In the first pass, we scan convert the triangles into the red channel of the grid, giving the 1st order DVD. In the second pass, we perform distance field computations [SGGM06]. The 2nd order DVD is concurrently computed in the green channel.

Inter-object queries: During scan conversion, the id of each object is stored in the stencil buffer. During distance field computation, the reference value and function for the stencil test are set to discard the fragment if the current object id is equal to value in stencil buffer. This avoids distance computation to an object O_i on grid points that belong to O_i. The nearest object and triangle ids are stored in the green and blue channels of the color buffer, and distance values are stored in depth buffer.

Intra-object queries: The list of adjacent feature ids is stored in an adjacency texture. During distance field computation, a dependent texture lookup is performed to query this list, and the fragment is discarded if the current feature id is present in the adjacency list.

We maintain a sorted list of intervals corresponding to the projection of an AABB along each axis. We compute the PNS used for the exact distance computation using the distance bounds computed from 2nd order Voronoi diagrams. We expand the sorted intervals with the distance bounds and compute features that overlap along the three axis. Next, we perform exact feature level distance tests. We used the code from [LGLM00] for computing the separation tests. The average time to perform one separation distance query between two triangles is 1–2 microseconds.

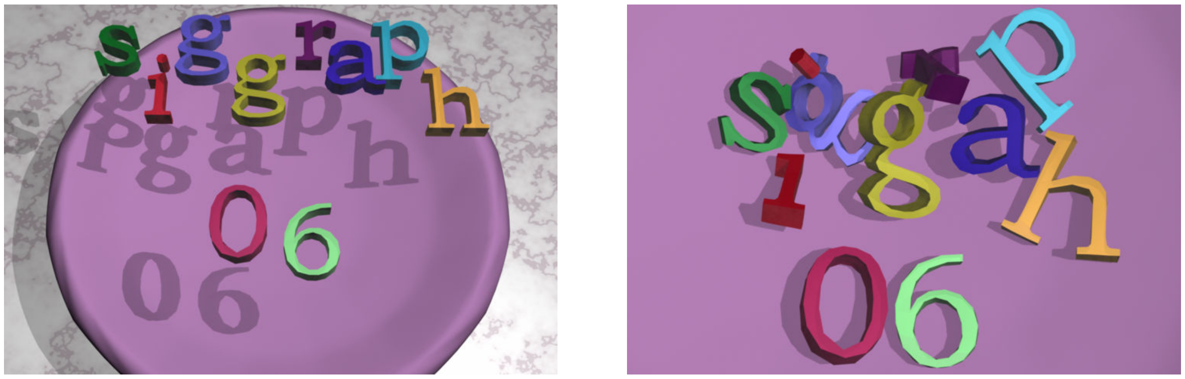


Figure 4.12: Multiple deformable models simulation: *The simulation of* 10 *deforming objects (*4.5*K triangles) falling in a bowl. Our algorithm is able to perform collision and separation distance computations among dynamically generated objects within* 70 *ms on a high-end PC. Our unified approach to perform N-body distance queries is based on efficient computation of the 2nd order discrete Voronoi diagram on the GPU.*

We have implemented PD computation by first computing the intersecting triangles using AABB hierarchies. We then perform a local walk to compute the overlapping features. Finally, we perform the N-body query to compute local PD.

In order to perform CCD tests, we compute tight prisms that enclose the swept volumes of the primitive [GKJ+05]. We then perform distance computations among the prisms and cull away primitive pairs not in close proximity. Finally, we perform elementary tests among the primitives in close proximity. The average time for performing a CCD test among two primitives is 50 microseconds.

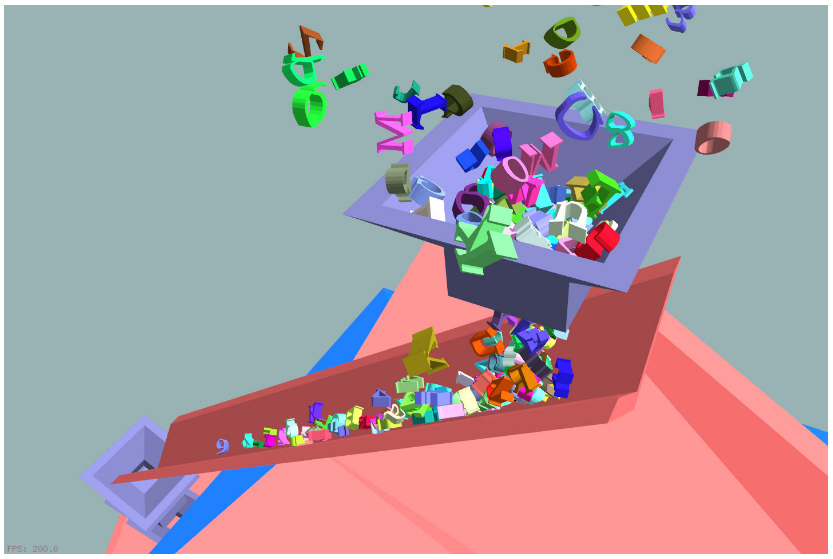


Figure 4.13: Large scale deformable object simulation: *In this simulation, many deforming letters are falling inside a funnel and will eventually slide through a ramp. Each object is composed of nearly* 175 *triangles and there are a total of* 200 *letters in many close-proximity scenarios. Our algorithm is able to perform both inter-object and intra-object queries in this simulation within a second.*

4.5.2 Benchmarks Used

We now highlight the performance of our algorithm on various benchmarks with multiple deformable objects. The set of benchmarks include:

1. A cloth simulation of a skirt (figure 4.10)
2. A cloth folding on a rotating sphere (figure 4.11)
3. Ten deforming letters falling in a bowl (figure 4.12)
4. Two hundred deforming objects falling through a funnel and sliding over a ramp (figure 4.13)
5. Fourteen objects undergoing dynamic topological fractures (figure 4.1)

Our algorithm involves no pre-processing and is able to compute the separation distances, inter-object and intra-object proximity queries.

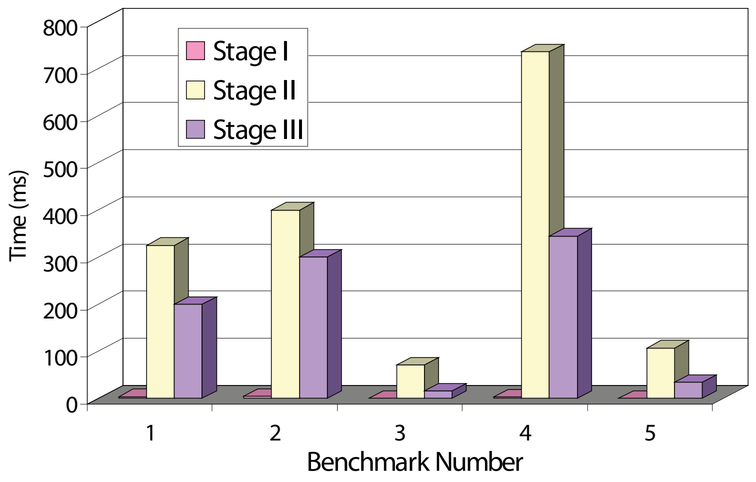


Figure 4.14: This graph highlights the average time spent in the three stages of our algorithm for the five benchmarks described in Section 6.2. Due to the high culling efficiency obtained during stage II, we observe that the average time spent in performing exact overlap tests is lower than 300*ms.*

Benchmark	**Tris**	**Resolution**	**AABB(s)**	**Voronoi(s)**
1. Skirt	12K	$200 \times 175 \times 45$	1.8	0.53
2. Cloth-Ball	15K	$190 \times 200 \times 60$	3.8	0.70
3. Bowl	4.5K	$150 \times 100 \times 30$	1.1	0.07
4. Ramps	38K	$45 \times 300 \times 40$	13.5	1.10
5. Breaking	5.5K	$100 \times 100 \times 60$	2.6	0.12

Table 4.1: Timings on deformable simulation benchmarks: Average time per frame (in seconds) to perform proximity queries on different benchmarks. AABB = Avg time/frame using an efficient AABB-based algorithm. Voronoi= Avg time/frame using our Voronoi-based algorithm.

A comparison of the performance of our Voronoi-based algorithm against an efficient AABB-based algorithm is provided in table 4.1. The grid resolution is a function of the bounding box of the environment. We use a different resolution along each axis to ensure that the resulting voxels have the same dimension along the 3 axes. As noted from figure 4.16, the resolution is chosen such that the total computation time is minimized.

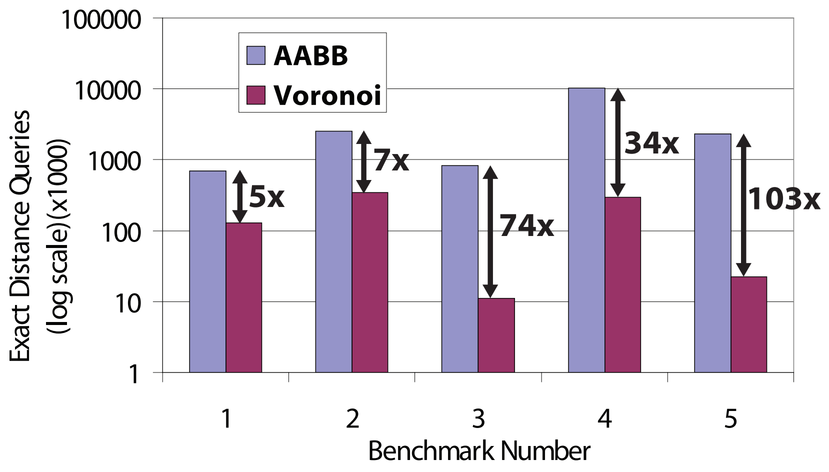


Figure 4.15: In this log-scale plot, we show the average number of exact triangle-triangle distance queries performed using an AABB-based algorithm and using Voronoi diagrams. We observe a $5-100$ *times higher culling efficiency using Voronoi diagrams on the five benchmarks. The high culling efficiency is due to the tight distance bounds obtained using the* 2^{nd}*order Voronoi diagrams.*

4.6 Discussion

In this section, we compare our algorithms with prior methods and analyse the performance benefits of our approach.

4.6.1 Comparison

We compare our algorithms with prior methods including distance and penetration depth computation, as well as continuous collision detection.

Separation distance and penetration depth: Most of the algorithms for inter-object queries use N-body techniques for the broad phase and bounding volume hierarchies for the narrow phase. However, prior N-body techniques are limited to collision or penetration queries, and may not provide sufficient culling for distance queries. Algorithms based on hierarchies for deformable models typically use AABBs or spheres [vdB97, LAM01] as bounding volumes, because the computation or update cost of hierarchies of OBBs or k-DOPs can be high. In

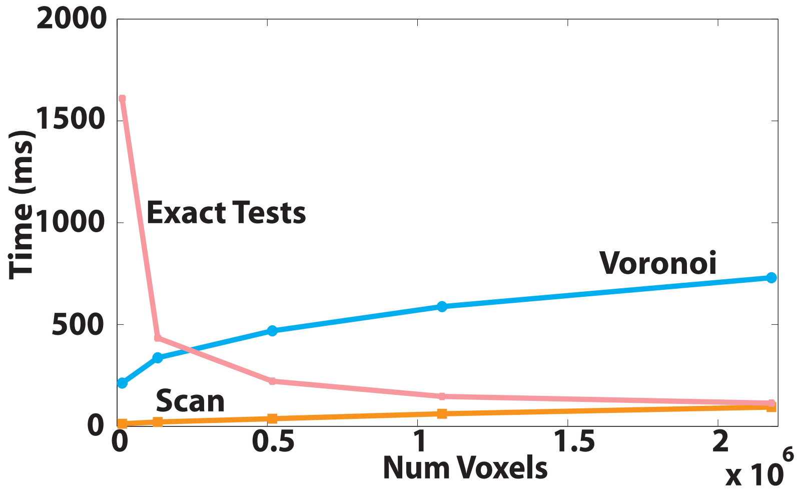


Figure 4.16: We show the time taken in computing the discrete Voronoi diagram and the culling efficiency as a function of the Voronoi grid resolution on a deformable simulation with 200 *objects. The culling efficiency is measured in terms of the number of exact distance tests. The scan operation reads back the Voronoi diagram, and performs a linear scan. We observe the number of exact tests decreases as the grid resolution increases.*

Fig. 4.17, we compare the performance our Voronoi-based culling algorithm with AABB hierarchies for separation distance computation in Benchmark 4. We observe more than an order of magnitude performance improvement in the query timings. This is due to the fact that Voronoi-based culling results in $5 - 100$x times reduction in the number of exact primitive tests as compared to the AABBs (shown in Fig. 4.15). The higher culling efficiency also reduces the additional overhead of hierarchy traversal for performing exact distance tests. As the number of objects in the scene increase, we obtain higher culling efficiency and performance improvement. Furthermore, hierarchical approaches may not work well for objects with changing topologies. The entire hierarchy has to be computed from scratch during each frame.

Collision detection: We compared the performance of our continuous collision detection algorithm with the one proposed by Govindaraju *et al.* [GKJ$^+$05]. In particular, we performed self-collision queries on Benchmark 1 and found that the performance of both algorithms was comparable and in the range of $400 - 800$ msec per frame. However, the algorithm

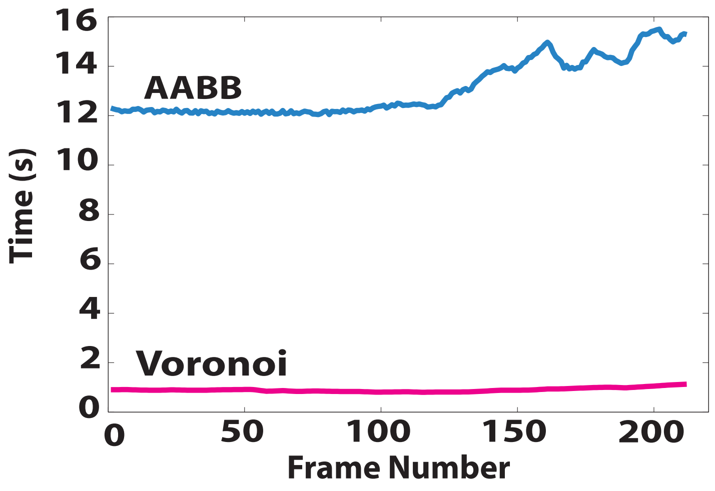


Figure 4.17: This graph highlights the performance improvement obtained using our Voronoi-based algorithm over an efficient AABB-based algorithm on the deformable simulation with 200 *objects. Due to the high culling efficiency obtained using Voronoi diagrams, we are able to achieve nearly one order of magnitude performance improvement over AABBs.*

proposed by Govindaraju *et al.* [GKJ$^+$05] assumes that the mesh connectivity is fixed and precomputes a chromatic decomposition. As a result, such an approach would not work on a scene with breaking objects (e.g. Benchmark 5). On the other hand, our approach involves no preprocessing and is applicable to all deformable models.

Distance field based algorithms: As compared to prior distance field algorithms [FL01, HZLM02, SGGM06], our approach is more accurate and we can perform queries at object-space precision. Furthermore, we can handle N-body, inter-object and intra-object queries. On the other hand, prior algorithms are restricted to performing these queries at image-space precision on a pair of objects.

Spatial hashing: Spatial grid and hashing techniques have been used to accelerate collision detection and penetration depth queries between a pair of objects [THM$^+$03, HTK$^+$04]. They work well when the models are represented as a union of tetrahedra or on queries involving points. In our benchmarks, spatial hashing-based methods resulted in a higher number of exact primitive tests as compared to AABB-based hierarchies. Moreover, the overhead of scan-converting the polygons among 3D grids can be high as compared to updating the hierarchies.

4.6.2 Analysis

Voronoi diagram in computational geometry is considered as one of the most powerful data structure for proximity queries. Our algorithm computes a tight superset (PNS) of potential Voronoi neighbors of primitives using discrete Voronoi diagrams and distance bounds. We use the PNS to perform N-body distance culling in complex environments composed of multiple deforming objects. Moreover, we show that other proximity queries such as continuous collision detection and penetration depth computation can also be efficiently performed using N-body distance culling. The overall benefit of our approach is due to two reasons:

- **Culling efficiency:** The 2nd order discrete Voronoi diagrams and tight distance bounds are used to cull away a high fraction of primitives that are not in close proximity. As a result, we have observed $30 - 50$ times improvement in culling efficiency over prior methods based on AABBs in complex deformable simulations.

- **Runtime performance:** We use the rasterization power of current GPUs for fast computation of 2nd order discrete Voronoi diagrams. We also localize the region for distance field computation. Our algorithm can compute the Voronoi information in a few hundred milli-seconds for complex environments. Moreover, our algorithm involves no hierarchy computation or update.

Based on these two reasons, we obtain considerable speedups over prior methods based on hierarchies. Moreover, we are able to perform various queries at almost interactive frame rates.

4.6.3 Limitations

Our approach has a few limitations. The computation of discrete Voronoi diagrams has overhead, in terms of rasterizing the distance functions and reading back the color and depth buffer. Even for small environments, the readback overhead can be $20 - 30$ msec. As a result, our current implementation would take at least $50 - 60$ msec to perform these queries, even on a simple environment. The main benefit of Voronoi-based culling arises in complex environments with a high number of primitives (e.g. a few thousand triangles). Our PNS computation can be conservative if the resolution of the discrete 3D grid is low. This can result in a high number of exact tests between the triangle primitives. Finally, our PD algorithm only computes a local PD. Our approach only works well if there is an isolated contact between the two objects. Many deformable simulations can result in deep penetrations or multiple contacts [BWK03, HTK$^+$04]. Our local PD algorithm may not work well in such situations.

Chapter 5

Homotopy Preserving Simplified Medial Axis

In this chapter, we introduce the *θ-Homotopy Medial Axis* (θ-HMA) of a 3D polyhedron. The θ-HMA is a simplified medial axis approximation which tends to remove unstable features of Blum's medial axis, and has the same homotopy type as Blum's medial axis.

Homotopy equivalence enforces a one-to-one correspondence between the connected components, holes, tunnels or cavities and the way they are related in the exact Voronoi diagram and the computed approximation. Thus, the θ-HMA is useful for applications that exploit the topological structure of the polyhedron including motion planning, topology preserving simplification, shape analysis and feature identification.

Our approach for computing the θ-HMA involves two key steps. In the first step we compute an approximate Voronoi diagram of the polyhedron. The computation is based on a spatial subdivision scheme and performs simple and efficient tests to compute a simplification of the exact Voronoi diagram. Moreover, we also describe algorithms to perform topological tests to guarantee homotopy equivalence of the approximate Voronoi diagram. We also provide Hausdorff distance bounds on the geometric structure of the approximate Voronoi diagram.

In the second step, we compute the θ-HMA from the homotopy preserving approximate Voronoi diagram. We use the separation angle formed connecting a point on the medial axis to closest point on the boundary as a measure of the stability of the medial axis at the point. The medial axis is decomposed into its parts, that are the sheets, seams and junctions. We present a stability measure of each part of the medial axis based on separation angles. Our simplification algorithm uses iterative pruning of the parts based on efficient local tests.

5.1 Related Work

The problem of Voronoi diagram and medial axis computation is well studied in computational geometry, solid modeling and their applications. In this section, we give a brief overview of previous algorithms on Voronoi diagram computation as well as medial axis simplification. We make this separation for convenience, but it is important to realize that the two are often integrated in practice.

5.1.1 Voronoi Diagram and Medial Axis Computation

Previous work on computation of the Voronoi diagram and the medial axis of 3D shapes can be categorized based on the sampling of $\mathbb{R}^3$. The discretization based methods approximate either the boundary of a polyhedral model with finite point samples, or sample the domain inside the polyhedron using spatial subdivision. The analytic methods trace the components of the Voronoi diagram using algebraic techniques.

Discretization based methods

Image datasets. The problem of MAT computation of a point dataset has been extensively studied in computer vision and image processing. In two and three dimensions, approximations to the medial axis have been computed using *thinning algorithms* [LLC92, ZW93]. Many algorithms based on partial differential equations of front propagation have also been proposed [KSKB95, SBS97]. Pizer et al. [PSS$^+$03] have generated structures related to the medial axis using filters which yield high values for points near the medial axis of an object.

Voronoi Graph of finite point samples: These methods approximate the boundary of the 3D polyhedron by a finite set of points and compute the Voronoi graph. Robust and efficient methods for computing the Voronoi diagram of point samples are well known. We refer the reader to a survey by[AK00]. The Voronoi graph of a finite set of points provides an approximation to the exact Voronoi diagram of the polyhedron[ACK01b]. The convergence to the exact Voronoi diagram has been shown for a sufficient dense sampling of smooth shapes. However, these methods algorithms may fail to provide a high quality approximation near sharp features of the original. Dey and Zhao [DZ02b] present an algorithm to approximate Voronoi diagrams and also give a convergence guarantee.

Spatial Subdivision techniques: These methods subdivide the space into cells and compute an approximate Voronoi diagram of a polyhedral model. The key step common to these algorithms is to compute and label each cell with a set of Voronoi governors and compute an approximate arrangement of Voronoi elements inside each cell. Vleugels and Overmars [VO98] present a technique to compute an approximate Voronoi diagram by determining cells that lie near Voronoi region boundaries. Approaches to efficiently perform labeling of a cell using propagation techniques have been presented for tetrahedral [TT97] and octree grids [BCMS05]. Etzion and Rappoport [ER02] decouple the computation of the symbolic part (the topology) of the Voronoi diagram from the geometric part and trace Voronoi elements across cell boundaries. We provide more detailed comparison with these approaches in Section 7.

There is also work on computing a discrete approximation to the Voronoi diagram by sampling the domain on a uniform grid. In such methods, the Voronoi regions are approximated using a finite set of points along a uniform grid. These approaches are well suited for interactive computation using graphics hardware [HCK$^+$99a, Den03b].

Foskey et al. [FLM03] used graphics hardware to generate an image-space representation of the gradient of the distance field to the boundary, which can be analyzed to find the medial axis. The gradient field in their method is actually the same as the velocity field of the propagating front in the methods of Siddiqi et al. [SBTZ02] mentioned above. Du and Qin [DQ04] also computed an approximation of the medial axis using diffusion partial differential equations solved at a discrete sample of boundary points. Yang et al. [YBM04] generated sample points on the boundaries of maximal spheres, and apply a separation angle criterion to select the points approximately on the medial axis.

However, previous spatial subdivision approaches cannot provide topological guarantees and may require extremely high level of subdivision to resolve near degenerate configurations in the Voronoi diagram.

Analytic methods

These methods detect topological events in the structure of the Voronoi diagram by tracing through a continuous domain. The correctness of continuous methods are not restricted by sampling parameters. For line segments in 2D, a sweep algorithm has been presented [For87]. Hanniel et al [HREK05] present a method for extracting the Voronoi regions of free-form rational planar closed curves based on tracing of the bisector curves. In 3D, these algorithms trace the 3D Voronoi edges (seams) [Mil93, SPB96, RT95a]. The approaches are highly sensitive to numerical precision, and robust implementations are difficult since it requires solving systems of tri-variate non-linear equations. In presence of degenerate configurations of the Voronoi diagram, such algorithms may fail to produce a valid output. A technique based on exact curve tracing is presented in [CKM04], however it does not scale well to large models. Furthermore, extremely high arithmetic precision is required to resolve near-degenerate configurations.

5.1.2 Medial Axis Simplification

In this section we give a brief overview of medial axis simplification algorithms. The instability of the medial axis, and its resulting complexity for objects with boundaries exhibiting fine detail, has been known for some time (see for instance, Blum and Nagel [BN78]). A number of methods for simplifying the medial axis have been proposed. Pizer et al. [PSS$^+$03] have presented an extensive survey of methods for approximating and simplifying the medial axis.

A well known criterion for medial axis simplification is based on the *object angle* [DDS03]. The separation angle is twice the object angle at any point on the medial axis. The underlying methods involve computing subsets for which the object angle is above a certain threshold. Malandain and Fernández-Vidal [MFV98] traced the idea, in varying forms, back to Meyer [Mey79] and Kruse [Kru91]. Our simplification algorithm also uses this criterion.

Siddiqi et al. [SBTZ02] formulated the detection of gradient discontinuities in terms of the average gradient flux into a neighborhood, which has been shown to be closely related to the object angle [DDS03]. Malandain and Fernández-Vidal [MFV98] used a criterion combining the object angle and the distance between the two points nearest to the medial axis point. Foskey et al. [FLM03] detected gradient discontinuities across adjacent voxels by comparing the directions of neighboring vectors.

Another class of approaches are based on using a point sampling of the boundary. These algorithms approximate the medial axis by computing the Voronoi diagram of the set of points and eliminating some of the Voronoi faces using different criteria. Amenta et al. [ACK01b] used the distance between the two points nearest to the medial axis point as a criterion for medial axis simplification. Dey and Zhao [DZ02a] combined a similar distance criterion with an object angle criterion and observed that the two criteria together tend to eliminate spurious holes. Tam and Heidrich [TH03] used a volume criterion to remove parts of the medial axis while preserving the topology. Leymarie and Kimia [LK01] also began with surface point samples, but their algorithms are based on the differential equations of front propagation.

5.1.3 Topological and Smoothness Properties

Attali, Boissonat, and Edelsbrunner [ABE04] survey different techniques that generate a stable and homotopy preserving medial structure. The homotopy relationship between an object and its medial axis has been proven in a particularly general form by Lieutier [Lie03], who shows that homotopy preservation holds for any bounded open subset of $\mathbb{R}^n$. Chazal and Soufflet [CS04] present smoothness constraints on the boundary of a solid, which need not be polyhedral, under which the medial axis obeys certain stability and finiteness conditions. Chazal and Lieutier [CL04] have also proven results about stability, and present a homotopy preserving medial axis simplification, however the approach has not been demonstrated on complex models.

5.2 Notation and Background

In this section, we introduce some of the terminology used in the rest of the chapter. We also provide a brief overview of the θ-simplified medial axis (θ-SMA).

5.2.1 Basic Terminology

The notations are summarized in Table 5.2.1. We explain some of those terms below. Given a closed polyhedral solid O in 3D, it boundary ∂O can be decomposed disjointly into vertices, open edges, and open faces, which we refer to collectively as *sites*. We shall denote the set of sites in ∂O as $\mathcal{P}$.

The *carrier* of an edge (face) site is the infinite line (plane) containing the site. The carrier

Notation	Meaning
$\overline{\mathcal{X}}$	Closure of a set $\mathcal{X}$
$\mathcal{X}^c$	Complement of $\mathcal{X}$
$\mathrm{Int}(\mathcal{X})$	Interior of $\mathcal{X}$
$\partial\mathcal{X}$	Boundary of $\mathcal{X}$
$\lvert\mathcal{X}\rvert$	Cardinality of $\mathcal{X}$
O	A polyhedral solid in $\mathbb{R}^3$
p_i	A face, edge or vertex site in $\mathbb{R}^3$
$car(p_i)$	*Carrier* of a site p_i
$d(\mathbf{q}, \mathbf{p})$	Distance between points $\mathbf{q}$ and $\mathbf{p}$
$d(\mathbf{q}, p_i)$	Distance between a site p_i and point $\mathbf{q}$
	$d(\mathbf{q}, p_i) = \min_{\mathbf{p}\in p_i}(d(\mathbf{q}, \mathbf{p}))$
$\pi_{p_i}(\mathbf{q})$	*Projection* of a point $\mathbf{q}$ on a site p_i
$\mathbf{n_i}(\mathbf{x})$	Normal to a site p_i from a point $\mathbf{x}$
$NB(\mathbf{x})$	Set of boundary points closest to $\mathbf{x} \in O$
$\mathcal{G}(\mathbf{x})$	Set of governors of a point $\mathbf{x} \in O$
$\mathcal{X} \sim \mathcal{Y}$	Sets $\mathcal{X}$, $\mathcal{Y}$ are homotopy equivalent
$\mathcal{X} \cong \mathcal{Y}$	Sets $\mathcal{X}$, $\mathcal{Y}$ are homeomorphic
B^d	A topological ball in d dimensions
S^d	A topological d-sphere in $d+1$ dimensions
$\mathcal{M}$	Medial axis of O
$\mathcal{F}$, f_i	Set of sheets of $\mathcal{M}$, one sheet of $\mathcal{M}$
$\mathcal{E}$, e_i	Set of seams of $\mathcal{M}$, one seam of $\mathcal{M}$
$\mathcal{V}$, v_i	Set of junctions of $\mathcal{M}$, one junction of $\mathcal{M}$
$\mathcal{R}(f_i)$	Set of *rim curves* of a sheet f_i
$\mathcal{S}(f_i)$	Set of *seam curves* of a sheet f_i

Table 5.1: This table highlights the notation used in the chapter

of a vertex site is the vertex itself. The *projection* of a point $\mathbf{q}$ on a site p_i, represented as $\pi_{p_i}(\mathbf{q})$, is the closest point on the the site p_i to the point $\mathbf{q}$:

$$\pi_{p_i}(\mathbf{q}) = \{\mathbf{x} \in p_i \mid d(\mathbf{q}, \mathbf{x}) \leq d(\mathbf{q}, p_i)\},$$

where $d()$ is the distance function.

The closed Voronoi region of a site p_i is defined as:

$$\mathcal{V}(p_i) = \overline{\mathcal{X}}, \text{ where } \mathcal{X} = \{\mathbf{q} \mid d(\mathbf{q}, p_i) < d(\mathbf{q}, p_j) \forall p_j \in \mathcal{P}\}.$$

For each point $\mathbf{x}$, we define the set of *governors*, $\mathcal{G}(\mathbf{x})$, to be the set of sites for which $\mathbf{x}$ belongs to the Voronoi region.

$$\mathcal{G}(\mathbf{x}) = \{p_i \mid \mathbf{x} \in \mathcal{V}(p_i), p_i \in \mathcal{P}\}$$

The governor set of a set of points is the union of governors of each point. For a point $\mathbf{x} \in O$, any point on the boundary of O that is at least as close to $\mathbf{x}$ as any other will be called a *nearest neighbor* of $\mathbf{x}$, and the set of nearest neighbors will be called the *neighbor set* of $\mathbf{x}$ and denoted $NB(\mathbf{x})$. With a distance function $d()$,

$$NB(\mathbf{x}) = \{\mathbf{y} \in \partial O \mid d(\mathbf{x}, \mathbf{y}) = d(\mathbf{x}, \partial O)\}.$$

Each nearest neighbor of $\mathbf{x}$ will be in exactly one site, hence we can also define the set of governors $\mathcal{G}(\mathbf{x})$ to be the set of sites containing a nearest neighbor of $\mathbf{x}$:

$$\mathcal{G}(\mathbf{x}) = \{p_i \mid \mathbf{y} \in p_i \text{ for some } \mathbf{y} \in NB(x)\}.$$

A cell in the spatial subdivision of the space is denoted C, and is homeomorphic to a closed ball $\overline{B^3}$. The elements of a cell are the cell faces, edges and vertices. For a cell C, $\mathcal{G}(C)$ is the set of sites whose Voronoi regions intersect C. A cell C is called a *boundary* cell if $C \cap \mathcal{P} \neq \emptyset$, i.e. the cell intersects one or more sites. A cell which is not a boundary cell is called an *interior* cell.

5.2.2 Voronoi Diagram Point Classification

Let α denote a set of two or more sites. The boundary of the Voronoi region is composed of bisectors with other sites called Voronoi faces. A *Voronoi face or a sheet*, denoted f_α, is a maximally connected 2-manifold surface which has the same 2 governors, i.e. $|\alpha| = 2$. The

2-D Voronoi faces meet in maximally connected 1-manifold curves called *Voronoi edges or seams*, which have the same set of governors. Each Voronoi edge has 3 or more governors. A Voronoi edge is denoted $e_\alpha, |\alpha| \geq 3$. Finally, the Voronoi edges meet at points called *Voronoi vertices or junctions* which are equidistant from four or more sites. A Voronoi vertex is denoted $v_\alpha, |\alpha| \geq 4$. The set of all Voronoi faces, edges and vertices is the generalized Voronoi diagram of $\mathcal{P}$, represented as $\mathrm{VD}(\mathcal{P})$ [AK96]. Formally,

$$\mathrm{VD}(\mathcal{P}) = \bigcup_{p_i, p_j \in \mathcal{P}, i \neq j} \mathcal{V}(p_i) \cap \mathcal{V}(p_j).$$

The Voronoi diagram decomposes the space into Voronoi regions. For each point $\mathbf{x} \in \mathrm{Int}(\mathcal{V}(p_i))$, $|\mathcal{G}(\mathbf{x})| = 1$. The Voronoi faces, edges and vertices are collectively called the elements of the Voronoi diagram.

We use the formulation described in [ER02] and define the Voronoi graph $\mathrm{VG}(\mathcal{P})$ as an undirected graph with the following properties:

1. Each node n in $\mathrm{VG}(\mathcal{P})$ corresponds to a Voronoi element (face, edge or vertex).

2. Two nodes in $\mathrm{VG}(\mathcal{P})$ share an arc iff there is an incidence relationship between the two corresponding Voronoi elements.

3. Each node is labeled by the governor set of its corresponding elements.

The Voronoi graph encodes the symbolic part of the Voronoi diagram. The approximate Voronoi diagram computed by our algorithm has the following property: a node in the graph of the approximate Voronoi diagram replaces a sub-graph in the graph of the exact Voronoi diagram.

5.2.3 Medial Axis Point Classification

The *medial axis* of O, denoted $\mathcal{M}$, is defined as the set of points inside O with at least two nearest neighbors.

$$\mathcal{M} = \{\mathbf{x} \in O, |NB(\mathbf{x})| \geq 2\}$$

Clearly, $|\mathcal{G}(\mathbf{x})| \geq 2$ for any point $\mathbf{x}$ on the medial axis.

We define a *sheet set* to be the set of all medial axis points governed by a specified pair of sites (or at least having that pair among their governors), and we define a *sheet* to be a connected component of a sheet set. The interior of a sheet is a smooth surface. A *seam curve*, or *seam*, is a connected component of the intersection of two or more sheets. The intersection of three or more seams is a *junction*. This definition corresponds approximately to those given in [CKM99] and [SPB96]. Finally, for any subset $\mathcal{M}'$ of $\mathcal{M}$, the intersection of a seam with the boundary will be a *seam end*. The intersection of a sheet with seam ends removed, and the boundary of $\mathcal{M}'$ will be a *rim set*. An example of seam points, junction points, and rim points is given in figure 5.1. A similar classification of medial axis points for any bounded set in $\mathbb{R}^3$ is given in [GK00]. Since the medial axis is a subset of the Voronoi diagram, the sheets, seams and junctions correspond to Voronoi faces, Voronoi edges and Voronoi vertices.

We make one special proviso about rims and seam ends. In general, including the case when $\mathcal{M}' = \mathcal{M}$, the boundary of $\mathcal{M}'$ will not be contained in $\mathcal{M}'$. In this case, it is possible that two sheets that do not intersect will have boundaries that do intersect. If this occurs, their rim curves and seam ends will be treated as distinct combinatorial entities, since the goal is to reflect the connectivity properties of $\mathcal{M}$, not its closure.

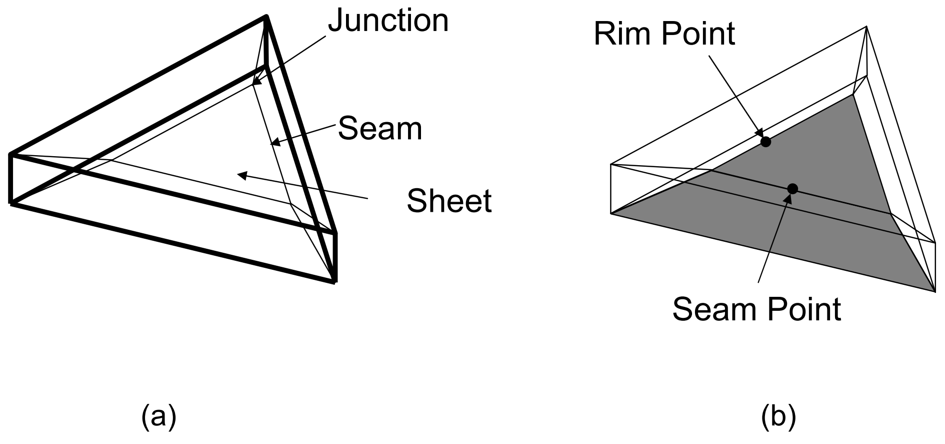


Figure 5.1: Medial axis point classification*: (a) Classification of the points on the medial axis (thin lines) of a simple polyhedron (thick lines) (b) A subset $\mathcal{M}' \subset \mathcal{M}$ is shaded in gray. A* rim point *and a* seam point *on the boundary of the central sheet are shown.*

5.2.4 Homotopy Equivalence

One of the major goals of our work is to compute a simplification of the MAT that is homotopy equivalent to the exact MAT. The notion of homotopy equivalence between topological sets enforces a one-to-one correspondence between connected components, holes, tunnels or cavities and also the way in which they are related. It has been shown by Lieutier [Lie03] that any bounded open subset $\mathcal{X} \subseteq \mathbb{R}^n$ is homotopy equivalent to its medial axis. Intuitively this implies that the medial axis and the shape are connected in the same way. Thus by computing a homotopy preserving Voronoi diagram, one can compute a simplified medial axis homotopy equivalent to the original shape [SFM05].

Formally, two maps $f\colon \mathcal{X} \to \mathcal{Y}$ and $g\colon \mathcal{X} \to \mathcal{Y}$ are *homotopic* if there exists a continuous family of maps $h_t\colon \mathcal{X} \to \mathcal{Y}$, for $t \in [0, 1]$, such that $h_0 = f$ and $h_1 = g$. Thus, a homotopy is a deformation of one map to another. Two spaces $\mathcal{X}$ and $\mathcal{Y}$ are *homotopy equivalent* if there exist continuous maps $f\colon \mathcal{X} \to \mathcal{Y}$ and $g\colon \mathcal{Y} \to \mathcal{X}$ such that $g \circ f$ and $f \circ g$ are homotopic to the identity maps on their respective spaces. As an example, f could be the inclusion of a circle into an annulus, and g could be radial projection of the annulus onto the circle.

In situations such as this one, where f is an inclusion and $f \circ g$ is actually equal to the identity map, the homotopy equivalence is called a *deformation retraction*. See Spanier [Spa89] for details of these definitions. Our medial axis simplification algorithm also performs a sequence of deformation retractions on the original medial axis to generate a simplified medial axis with the same homotopy type as the original.

5.2.5 θ-Simplified Medial Axis

Given a polyhedral model O and a medial axis $\mathcal{M}$, the *separation angle* $\Theta(\mathbf{x})$ at each point $\mathbf{x}$ on $\mathcal{M}$ is the largest angle subtended by a pair of nearest neighbor points on ∂O, and is given by

$$\Theta(\mathbf{x}) = \max_{\mathbf{y}_i, \mathbf{y}_j \in NB(\mathbf{x})} (\angle \mathbf{y}_i \mathbf{x} \mathbf{y}_j)$$

Given an angle θ, the *θ-simplified medial axis* (θ-SMA) of O, denoted by $\mathcal{M}_\theta$, is the set of points of $\mathcal{M}$ with separation angle greater than θ [FLM03] (see figure 5.2). Foskey et

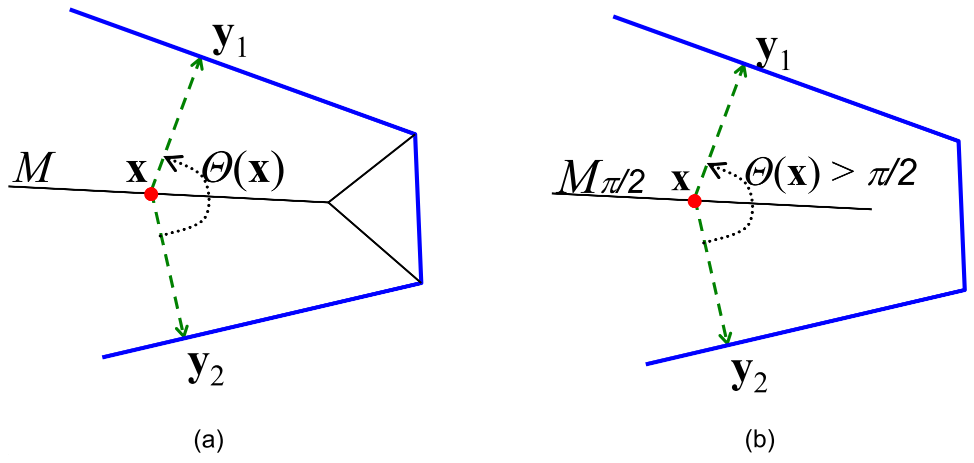


Figure 5.2: θ-Simplified Medial Axis, $\mathcal{M}_\theta$: (a) The medial axis (black) of a part of a polyhedron (blue) (b) $\mathcal{M}_\theta$ for $\theta = \pi/2$.

al. [FLM03] discuss the convergence and stability of $\mathcal{M}_\theta$ and provide error bounds on the boundary reconstructed from $\mathcal{M}_\theta$. The speed of medial axis formation at point $\mathbf{x}$ is proportional to $\frac{1}{\sin\Theta(\mathbf{x})}$ [PSS+03]. Parts of the medial axis with a higher speed of formation are regarded as more important [Blu67], and the separation angle $\Theta(\mathbf{x})$ has been used as a measure of the stability of the medial axis at the point $\mathbf{x}$.

5.3 Homotopy Preserving Voronoi Diagram

In this section, we provide an overview of our approach for computing the homotopy preserving approximate Voronoi diagram of a 3D polyhedron. We then present our theoretical results and subdivision criterion to guarantee homotopy equivalence between the approximate Voronoi diagram and the exact Voronoi diagram in a cell. We use this criterion as part of the algorithm presented in Section 5.4. Finally, we show that our criterion is satisfied at some finite level of subdivision, and thereby proving completeness.

5.3.1 Overview

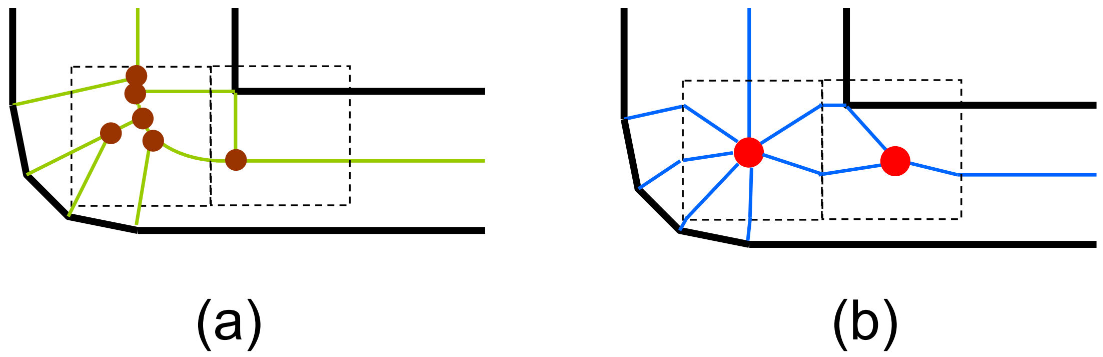


Figure 5.3: Homotopy Preserving Approximate Voronoi Diagram*: A subset of a 2D polygon is shown in bold. (a) The exact Voronoi diagram is shown in green. Two cells of a spatial subdivision are shown with dotted lines. Brown points represent Voronoi vertex nodes. (b) Each cell satisfies the homotopy preserving criterion. The corresponding homotopy preserving approximate Voronoi graph is shown in blue. The red points represent nodes approximating the Voronoi subgraph inside the cell.*

We assume that the Voronoi diagram is defined with respect to the Euclidean metric. We construct the Voronoi diagram by separately computing the symbolic and geometric parts. We compute an approximate Voronoi graph, such that the corresponding approximate Voronoi diagram is homotopy equivalent to the exact Voronoi diagram.

The computation of the symbolic part of the Voronoi diagram is based on spatial subdivision that is used to compute the incidence relationships between Voronoi diagram elements. During spatial subdivision, each cell and the cell elements are labeled by their respective governors. The subdivision is terminated when the portion of the Voronoi diagram constrained to the interior of the cell is homotopy equivalent to a point. Under this condition, multiple vertex nodes in the Voronoi graph inside the cell can be replaced by a single vertex node. An example is shown in figure 5.3.

To guarantee homotopy equivalence, we first highlight some topological properties of Voronoi regions under the Euclidean distance metric. Moreover, we present a criterion to guarantee that the Voronoi diagram computed within a cell is homotopy equivalent to a point. The criterion is based on computing the arrangement of conics (i.e. degree two algebraic curves) on a plane and involves solving univariate quartic equations. The criterion is presented in Section 5.3. In order to accelerate the computation and reduce the number of non-linear tests, we perform spatial subdivision and update the governor set associated with each cell. The algorithms to evaluate the homotopy criteria and computing a homotopy preserving approximate Voronoi graph are presented in Section 5.4.

Given the graph of homotopy preserving approximate Voronoi diagram, we compute a geometric approximation to the Voronoi diagram using techniques presented in [ER02, BCMS05]. This involves computing an approximation of the seams and sheets. Furthermore, the diameter of the cell used for spatial subdivision algorithm provides bounds on the two sided Hausdorff distance between the geometric approximation and the exact Voronoi diagram. In other words, the cell size is chosen as a function of the Hausdorff bound.

In our approach, we ignore degenerate Voronoi regions. A Voronoi region $\mathcal{V}(p_i)$ is said to be degenerate if it has zero volume, i.e. there does not exist a ball B^3 such that $B^3 \subset \mathcal{V}(p_i)$.

Such Voronoi regions belong to an edge shared between two co-planar triangles, or a vertex for which all incident triangles are co-planar. Sites with degenerate Voronoi regions are removed from $\mathcal{P}$ as a preprocess. Note that removal of degenerate Voronoi regions does not change the homotopy type of the Voronoi diagram, since a degenerate Voronoi region is a subset of the closure of an adjacent Voronoi region. Furthermore, we constrain the domain of computation to be inside a bounding box of the polyhedron, so that each Voronoi region is closed and bounded (i.e. it is a compact set).

5.3.2 Homotopy Criterion

We begin by presenting a topological property of Voronoi regions used by the homotopy criterion to guarantee homotopy equivalence. Then we present the criterion to check if the Voronoi diagram inside a cell in the spatial subdivision is homotopy equivalent to a point, and prove the homotopy equivalence.

Proposition 5.1 (Voronoi regions are topological balls). *If each site p_i is a convex set, then each bounded Voronoi region $\mathcal{V}(p_i)$, under the Euclidean distance metric, is homeomorphic to an open ball B^3.*

Proof. We show that $\mathcal{V}(p_i)$ is contractible, i.e. homotopy equivalent to a point in $\mathbb{R}^3$. and rely on the fact that a contractible compact subset of $\mathbb{R}^3$ is homeomorphic to a ball B^3 [CZ06]. We prove contractibility by constructing an explicit map. We define a continuous map $F : \mathcal{V}(p_i) \times I \rightarrow \mathcal{V}(p_i)$, such that $F(\mathbf{x}, 0) = \mathbf{x}$, for any $\mathbf{x} \in \mathcal{V}(p_i)$, and $F(\mathbf{x}, 1) = \mathbf{c}$ for some point $\mathbf{c}$. Here I is the unit interval $[0, 1]$. Let $I_1 = [0, 0.5], I_2 = [0.5, 1]$. We construct F in two stages,

$$\begin{aligned} F(\mathbf{x}, t) &= G(\mathbf{x}, t) \forall t \in I_1 \\ &= H(G(\mathbf{x}, 0.5), t) \forall t \in I_2 \end{aligned}$$

where, $G : \mathcal{V}(p_i) \times I_1 \to \mathcal{V}(p_i)$ and $H : p_i \times I_2 \to p_i$, $G(\mathcal{V}(p_i), 0.5) \subseteq p_i$ and $H(p_i, 1) = \mathbf{c}$.

First we shall construct G. Consider the map $\pi_{p_i}(\mathbf{x}) : \mathcal{V}(p_i) \to p_i$. Let $G(\mathbf{x}, t) = (1 - 2t)\mathbf{x} + 2t\pi_{p_i}(\mathbf{x})$, where $t \in I_1, \mathbf{x} \in L$. To prove that G is continuous, we need to show that $\pi_{p_i}(\mathbf{x})$ is continuous. Assume that $\pi_{p_i}(\mathbf{x})$ is not continuous. Then some point $\mathbf{x} \in \mathcal{V}(p_i)$ has 2 unique closest points on p_i, let $\pi_{p_i}(\mathbf{x}) = \{\mathbf{p_1}, \mathbf{p_2}\}$. Consider the isosceles triangle $\Delta\mathbf{x}\mathbf{p_1}\mathbf{p_2}$ and the mid-point, $\mathbf{p} = \frac{\mathbf{p_1}+\mathbf{p_2}}{2}$. Then $\overline{\mathbf{xp}}$ is an altitude from $\mathbf{x}$ to $\overline{\mathbf{p_1p_2}}$ and $d(\mathbf{x})\mathbf{p} < d(\mathbf{x})\mathbf{p_1} = d(\mathbf{x})\mathbf{p_2}$. Since p_i is convex, $\mathbf{p} \in p_i$ and leads to the contradiction $\pi_{p_i}(\mathbf{x}) = \mathbf{p}$. Thus the maps $\pi_{p_i}(\mathbf{x})$ and G are continuous. Further $G(\mathbf{x}, t)$ gives the shortest path from $\mathbf{x}$ to p_i. Sherbrooke et al. [She95] show that (a) the shortest path from a point on the Voronoi diagram (medial axis) to the closest site lies entirely inside the Voronoi region, and (b) the shortest paths from two points on the Voronoi diagram to the closest site can intersect only at the site. Thus $G(\mathbf{x}, t) \in \mathcal{V}(p_i)$ for all $\mathbf{x} \in \mathcal{V}(p_i), t \in I_1$, and $G(\mathbf{x_1}, t) \cap G(\mathbf{x_2}, t) = \emptyset$ for $\mathbf{x_1}, \mathbf{x_2} \in \mathcal{V}(p_i), \mathbf{x_1} \neq \mathbf{x_2}, t \in [0, 0.5)$.

Now we construct the map H. Let $\mathbf{c}$ be the centroid of p_i. Since p_i is convex, $\mathbf{c} \in p_i$. Let $H(\mathbf{x}, t) = 2(1 - t)\mathbf{x} + 2(t - 0.5)\mathbf{c}, t \in I_2, \mathbf{x} \in p_i$. $H(\mathbf{x}, t)$ is a continuous function, by definition. Since each site is simply connected, $H(\mathbf{x}, t) \in p_i$ for all $\mathbf{x} \in p_i, t \in I$. By definition, $F(\mathbf{x}, t)$ is continuous at $t = 0.5$. Thus $\mathcal{V}(p_i)$ is contractible. $\square$

Definition (Homotopy Criterion): An AABB cell C, with governor set $\mathcal{G}(C)$, satisfies the *homotopy criterion* if $\mathcal{V}(p_i) \cap \partial C$ is homeomorphic to a topological disk B^2, for each $p_i \in \mathcal{G}(C)$.

We will show that the Voronoi diagram inside a cell satisfying the homotopy criterion is contractible (i.e. homotopy equivalent to a point). Following this result, all Voronoi vertices inside the cell can be replaced by a single vertex while preserving the homotopy type. We now present some results that follow from the homotopy criterion and prove that the Voronoi diagram in the cell is indeed contractible.

Lemma 5.1. *Let C be a cell satisfying the homotopy criterion. For all $p_i \in \mathcal{G}(C)$, (a) $\mathcal{V}(p_i) \cap \partial C \neq \emptyset$, (b) $\partial\mathcal{V}(p_i) \cap C \cong B^2$ and (c) $\mathcal{V}(p_i) \cap C \cong B^3$.*

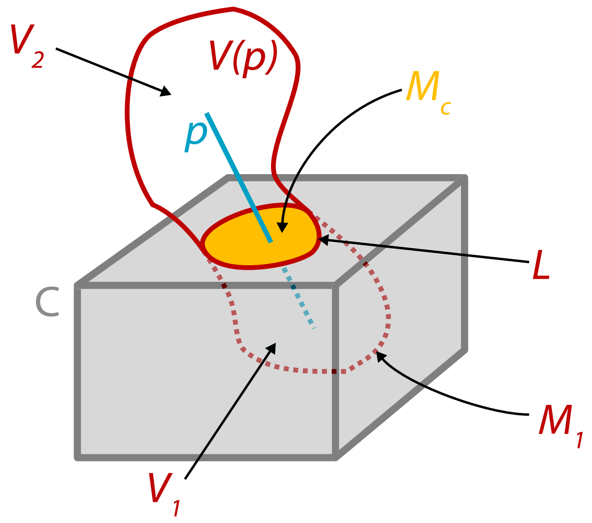


Figure 5.4: Proof of Lemma 5.1: The Voronoi region $\mathcal{V}(p)$, of a line site p, intersecting a 3D cell C. The boundary of the cell partitions $\mathcal{V}(p)$ into 2 regions: V_1 in the interior of C and V_2 in the exterior. $\partial V_1 = M_1$. Intersection of $\mathcal{V}(site)$ with ∂C is a topological disk, denoted M_c

Proof. The result (a) follows from the definition of $\mathcal{G}(C)$. ∂C partitions $\mathcal{V}(p_i)$ into 2 spaces, $V_1 = \mathcal{V}(p_i) \cap C, V_2 = \mathcal{V}(p_i) \cap C^c$ (i.e. V_2 is outside the cell C) (see figure 5.4). Let $M_1 = \partial\mathcal{V}(p_i) \cap C, M_c = \mathcal{V}(p_i) \cap \partial C, L = \partial M_c = \partial\mathcal{V}(p_i) \cap \partial C$. We need to show that $M_1 \cong B^2, V_1 \cong B^3$. From the homotopy criterion, $M_c \cong B^2$, and the boundary L is a simple closed curve, $L \cong S^1$. This boils down to proving that $V_1 \cong B^3$. From property 5.1, it follows that $\partial\mathcal{V}(p_i) \cong S^2$. Furthermore, $L \subset \partial\mathcal{V}(p_i)$, and using the Jordan curve theorem on the 2-sphere, it partitions $\partial\mathcal{V}(p_i)$ into 2 topological disks. Thus, $M_1 \cong B^2$. We now define a homeomorphism $f : \partial M_1 \longrightarrow \partial M_c$ which glues M_1 to M_c. We also have $M_1 \cap M_c = L \cong S^1$. Thus f is the identity map on L, which maps each point on ∂M_1 to the identical point on ∂M_c. Thus the connected sum of M_1 and M_c is homeomorphic to a 2-sphere. Then $M_1 \cup M_c \cong S^2$, thus $\partial V_1 \cong S^2, V_1 \cong B^3$. □

Using the above results, we provide an explicit construction to prove that the homotopy criterion is sufficient for the Voronoi diagram constrained to the cell is contractible. To prove this, we perform a series of retractions on the Voronoi regions contained inside a cell.

We define a retraction $g_i : C \to C$ to be the exclusion of the interior of Voronoi region from the cell C (see figure 5.5). Given cell C satisfies homotopy criterion, let V_k be the subset of C left after k retractions, where $k = 0, 1, \ldots, |\mathcal{G}(C)|$. We now prove a result on the retractions.

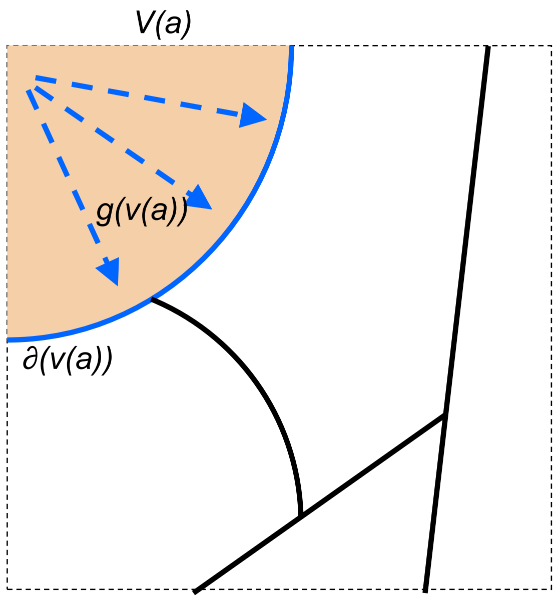


Figure 5.5: Deformation retract of a Voronoi region*: A 2D cell is shown with dotted boundary. The solid curves represent a Voronoi diagram. Each Voronoi region satisfies the homotopy criterion (in 2D). The retraction g takes all points in the Voronoi region $\mathcal{V}(a)$ to its boundary $\partial\mathcal{V}(a)$.*

Lemma 5.2. *V_{k+1} is homotopy equivalent to V_k.*

Proof. $V_{k+1} = V_k \setminus \mathrm{Int}(\mathcal{V}(p_i))$. Since C satisfies homotopy criterion, $\mathcal{V}(p_i) \cap C \cong B^3$ and $\partial\mathcal{V}(p_i) \cap C \cong B^2$. Also, $(\mathcal{V}(p_i) \cap C) \setminus \mathrm{Int}(\mathcal{V}(p_i)) = \partial\mathcal{V}(p_i) \cap C$. There exists a deformation retract from a ball B^3 to a disc B^2. This implies existence of a map $G : \mathcal{V}(p_i) \cap C \to \partial\mathcal{V}(p_i) \cap C$ such that: (a) the restriction of H to $\partial\mathcal{V}(p_i) \cap C$ is equal to the identity on

$\partial\mathcal{V}(p_i) \cap C$, and (b) $H \circ G$ is homotopy equivalent to the identity on $\partial\mathcal{V}(p_i) \cap C$, where H is the inclusion $\partial\mathcal{V}(p_i) \to \mathcal{V}(p_i)$.

We then define $\hat{G} : V_k \to V_{k+1}$ to be the identity on $V_{k+1} \subset V_k$ and equal to G on $\mathrm{Int}(\mathcal{V}(p_i))$. Then if $\hat{H}$ is the inclusion $V_{k+1} \to V_k$, it is clear that $\hat{G} \circ \hat{H}$ is homotopic to the identity on V_k and $\hat{H} \circ \hat{G}$ is homotopic to the identity on V_{k+1}. Thus $V_k \sim V_{k+1}$. □

Theorem 5.1. *If a cell satisfies the homotopy criterion, then the Voronoi diagram constrained to the cell is contractible.*

Proof. Initially $V_0 = C$ and finally $V_f = \mathrm{VD}(\mathcal{P}) \cap C$ where $f = |\mathcal{G}(C)|$. From lemma 5.2, V_0 and V_f are homotopy equivalent. We know that the cell C is contractible. Thus V_f is contractible. □

5.3.3 Completeness

In this section, we prove the completeness. To do this we use the following theorem:

Theorem 5.2. *For any point on the boundary of a Voronoi region $\mathcal{V}(p_i)$, there exists an open ball B_r of strictly positive radius r such that $\partial\mathcal{V}(p_i) \cap B_r \cong B^2$.*

Proof. We perform case analysis on the location of the point.

(a) The point lies in the interior of a Voronoi face. Each face is a 2-manifold embedded in $\mathbb{R}^3$. Then at each point on the face, there exists an open ball of finite radius such that intersection of the ball with the face is 2-manifold - i.e. homeomorphic to a disk.

(b) The point lies in the interior of a Voronoi edge. At a Voronoi edge, the Voronoi region is bounded by 2 Voronoi faces. Each bisector surface (i.e. a quadric surface) is diffeomorphic to a disk. In a small neighborhood of the point, the arrangement of the Voronoi faces incident at the Voronoi edge is homeomorphic to the arrangement of a set of half-planes incident at an edge. The intersection of a half-plane with a sphere centered on the edge is a single curve segment. Then the 2 curve segments, arising from the intersection of the sphere and the two bounding Voronoi faces meet at exactly 2 points - the end points of the 2 curves. Thus the boundary of the intersection of Voronoi region boundary at a Voronoi edge and the boundary of ball (centered on edge) is a circle. Therefore, the intersection with the ball is a disk.

(c) The point lies on a Voronoi vertex. The proof for case (b) extends to this case. The boundary of a Voronoi region in the neighborhood of a vertex consists of a finite number of Voronoi faces meeting at Voronoi edges. □

Theorem 5.2 implies that for any point on the Voronoi diagram $\mathrm{VD}(\mathcal{P})$, we can find a ball of a finite radius such that the intersection of the Voronoi regions with the ball satisfy the homotopy criterion. Thus the subdivision will terminate once the current cell is contained inside such a ball.

5.4 Approximate Voronoi Diagram Computation

In this section, we present details of our algorithm. First we describe how we evaluate the homotopy criterion for each Voronoi region in a given cell. Then we present our algorithm to compute the graph of the approximate Voronoi region.

5.4.1 Homotopy Criterion Computation

Theorem 5.1 in Section 5.3.2 implies that this test reduces to checking whether the intersection of the Voronoi diagram with the boundary of a cell is homeomorphic to a disk. This is equivalent to determining if the intersection of the boundary of a Voronoi region with a cell is homeomorphic to a circle. We compute the boundary of the Voronoi region along each face of the cell and compute the union over all faces.

The boundary of a Voronoi region consists of sheets, seams and junctions. Each sheet is a subset of the bisector between the carriers of two sites. Given a sheet f_α and a cell face F, a *Voronoi face event* is the intersection of f_α and F and corresponds to a conic curve on F

in the general case. We compute an arrangement of the conics on the face [KCMh99]. The intersection of the conic sections gives *Voronoi edge events* [ER02], representing intersection of seams with a cell face. Along with each edge event, we store the set of governors of the Voronoi edge. If the sheet is a plane tangential to cell face, we compute the intersection with the face vertices. In case the Voronoi edge event consists of infinite number of points, we compute its intersection with the boundary of a face.

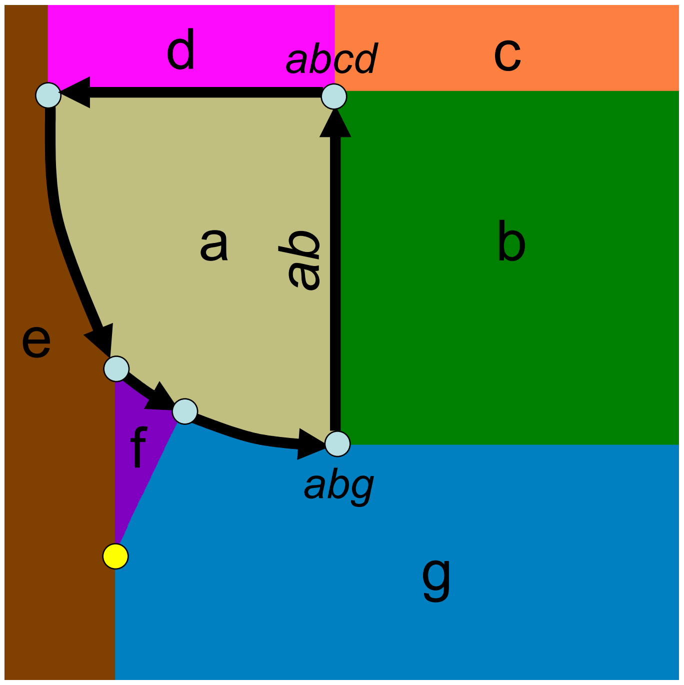


Figure 5.6: Homotopy criterion computation*: We show a face of a cell in the computation of approximate Voronoi diagram of the L-shape. Each colored region represents the intersection of a Voronoi region with the face, and is labeled by its governor. Each region is homeomorphic to a disc, hence satisfies the homotopy criterion. The circles represent Voronoi edge events: e.g. the point* $(abcd)$ *represents intersection of a degenerate Voronoi edge and the face. The bold conic segments represent the face events, representing boundary of the Voronoi region of site* a*, computed by our tracing algorithm.*

All intersections of conics do not provide the valid edge events. We compute the valid edge events based on the algorithm `CellFaceVoronoiEdgeIntersection` presented in [ER02]. Given the set of edge events, we trace the conic segments between edge events sharing a common governor to obtain the Voronoi face events. A closed sequence of face events sharing a common governor provides the boundary of the Voronoi region of the site on the cell face. Two edge events are connected by a face event if they share at least two common governors (corresponding to the bisector between the governors). In case there are multiple points sharing same 2 governor labels, we sort them according to their parametric coordinates on the conic and connect the 2 closest points. In the presence of degenerate seams, each conic segment between two edge events may not represent a valid face event. Checking if a segment is a valid face event is equivalent to determining if it lies on the boundary of the Voronoi region of a site p_i. In order to perform this test, we enumerate all conic segments incident on an edge event and trace along the conic segment which is closer to the p_i than to all other governors of the edge event. Finally, we join the face events at boundaries of adjacent faces to compute the intersection of the Voronoi region with the boundary of the cell. A cell satisfies the homotopy criterion if all the Voronoi region boundaries on the cell boundary form one simple closed loop.

5.4.2 Computing cell governors

The homotopy criterion needs to be satisfied for all sites that belong to the governor set of a cell. Here we present our scheme to compute a set of governors of the cell. We use a sequence of culling tests to prune the set of governors of a cell. A site p_i can be removed from the governor set of a cell C of diameter δ if:

1. **Distance exclusion**: There exists another governor $p_j \in \mathcal{G}(C)$ such that centroid of C is closer to p_j and difference in distance is greater than δ.

2. **Polytope exclusion**: The domain polytope (a polytope bounding the Voronoi region of site p_i) does not intersect C.

3. **Bisector exclusion**: There exists another governor $p_j \in \mathcal{G}(C)$ such the cell C is closer to p_j and lies inside the domain polytope of p_j.

Each of these tests involves solving inequalities or a system of linear equations [Cul00]. These tests provide a conservative estimate of the governors of a cell. The exact set of governors of the faces of a cell is computed from the arrangement of Voronoi regions on the faces, as described in section 5.4.1. We now present a result that ensures computing the arrangement on the boundary of a cell is sufficient for computing the cell governors.

Lemma 5.3. *For an interior cell C, if $\mathcal{V}(p_j) \cap \mathrm{Int}(C) \neq \emptyset$ then $\mathcal{V}(p_j) \cap \partial C \neq \emptyset$.*

The proof follows trivially from the facts that the Voronoi regions are connected (topological balls) and contain the site. A consequence of Lemma 5.3 is that it suffices to check the boundary of a cell to compute governors of an interior cell. For boundary cells, we impose further restrictions on the governor set of the cell to check if each Voronoi region intersects the cell boundary.

Boundary cell criterion: Given a boundary cell C, with a set of sites $\mathcal{X}$ intersecting C, C satisfies the boundary cell criterion if:

1. $\mathcal{X}$ contains at most one point site p_p, and $\mathcal{X} \setminus \{p_i\}$ contains sites incident on the point p_i.

2. The governor set $\mathcal{G}(C)$ is a subset of $\mathcal{X}$.

These two conditions ensure that each non point site in the governor set $\mathcal{G}(C)$ intersects the boundary of the cell - thus their Voronoi regions must intersect the boundary of the cell. For each point site, its Voronoi region constrained to the cell is given by intersection of its domain polytope and the cell, thus its Voronoi region must intersect the cell boundary if its domain polytope is non-empty. Condition (1) can be trivially tested. We conservatively test for condition (2) by checking if the conservative governor set does not include any sites from $\mathcal{P} \setminus \mathcal{X}$.

5.4.3 Approximate Voronoi Diagram Computation

In this section we provide our algorithm for computing a homotopy preserving approximate Voronoi diagram. We first compute a homotopy preserving approximate Voronoi graph using spatial subdivision. The steps are given as follows:

1. Compute a discrete distance field on uniform grid at some fixed resolution.

2. Compute the governor set of each cell using exclusion tests presented in Section 5.4.2.

3. Check if a cell satisfies the homotopy criterion. In addition, check if each boundary cell satisfies the boundary criterion. If either of the criteria are not met, subdivide and update the governor sets of the children cells.

4. If a cell satisfied the homotopy and boundary criteria, insert a subgraph node inside the cell. Connect the node to the edge events on the boundary of the cell.

This algorithm provides us with a homotopy preserving approximate Voronoi graph. To extract the homotopy preserving approximate Voronoi diagram, we further refine it to detect unique vertex nodes and edge nodes. We use a result from [ER02] to detect Voronoi vertices: If the number of intersection points of a Voronoi edge e_α and ∂C is odd, then there exists a Voronoi vertex in C. We subdivide a leaf cell if it contains more than two edge events with same governor set. If a cell has exactly two edge events with same governor set, we remove the subgraph node and directly connect the two edge events with a subset of the Voronoi edge. The refined approximate Voronoi graph consists of nodes of type Voronoi vertex and subgraph and edge nodes connecting the vertex and subgraph nodes. We follow a loop of Voronoi edge events joined by the same face event on the boundary of a cell to extract the Voronoi faces.

5.5 θ-Homotopy Medial Axis

In this section, we analyze the topological characterization of θ-SMA and present a formulation for computing a homotopy-preserving simplified medial axis, the *θ-homotopy medial axis*. The problem with the θ-SMA is that it does not in general preserve the homotopy type of the medial axis. The θ-SMA can be disconnected when the medial axis is connected, or have holes when the medial axis does not, and lack holes when the medial axis has them. An illustration of the failure of connectivity is shown in Figure 5.7. The other kinds of con-

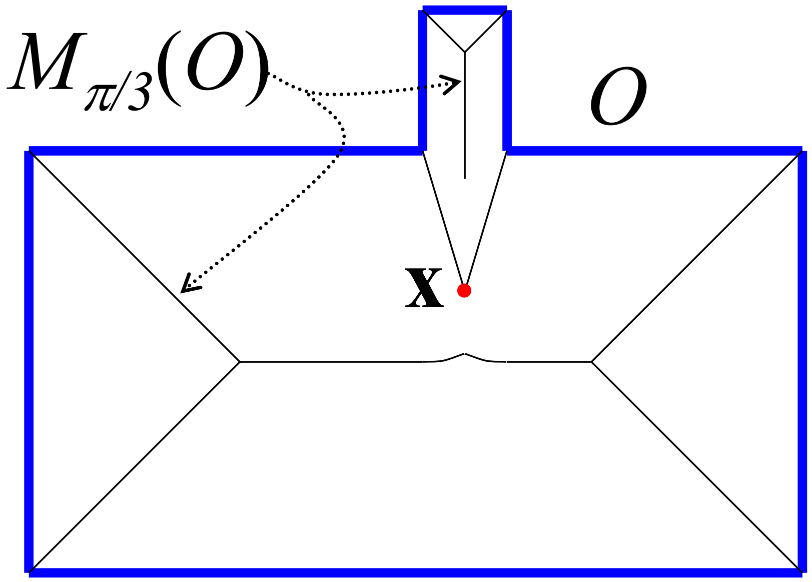


Figure 5.7: The θ-Simplified Medial Axis, $\mathcal{M}_{\pi/3}$ is disconnected even though the original object O is connected. Note that the separation angle at $\mathbf{x}$ is less than $\pi/3$, while it exceeds $\pi/3$ for the portions of the medial axis shown.

nectivity problems also arise because the angle criterion may discard topologically significant portions. The fundamental issue here is that homotopy type is a global property, whereas the separation angle is a local measure.

Decreasing the θ threshold does not provide a guaranteed solution to fix the problems. As illustrated in Figure 5.7, the problem is associated with local minima of the separation angles, and such a local minimum can occur for any value of θ. In any event, decreasing θ

only increases the number of unstable features of the θ-SMA.

Our goal is to compute a simplified medial axis that would allow significant simplification corresponding to large values of θ, while preserving the homotopy type of $\mathcal{M}$. Clearly such a simplified medial axis has to be a superset of $\mathcal{M}_\theta$. However, we would like such an axis to be minimal in some regard in order to minimize the unstable parts. We now formally present the desired subset of the medial axis. Let $\mathcal{H}_\theta$ denote the class of subsets of $\mathcal{M}$ which are supersets of $\mathcal{M}_\theta$ and are homotopy equivalent to $\mathcal{M}$.

$$\mathcal{H}_\theta = \{\mathcal{X} | \mathcal{X} \subseteq \mathcal{M}, \mathcal{X} \supseteq \mathcal{M}_\theta, \mathcal{X} \simeq \mathcal{M}\}$$

Define a set $\mathcal{X} \in \mathcal{H}_\theta$ to be *irreducible* if the removal of any sheet yields a set that either has a different homotopy type, or is no longer a superset of $\mathcal{M}_\theta$. That is,

$$\mathcal{M}^*_\theta = \{\mathcal{X} | \mathcal{X} \in \mathcal{H}_\theta, \text{ for all } f_i \in \mathcal{X}, (\mathcal{X} \setminus \{f_i\}) \notin \mathcal{H}_\theta\}$$

We will refer to any irreducible set in $\mathcal{H}_\theta$ as a *θ-homotopy medial axis*, or θ-HMA. We will typically denote a θ-HMA by $\mathcal{M}^*_\theta$. The set $\mathcal{M}^*_\theta$ is not unique. A discussion about lack of uniqueness is presented in Section 5.9.

5.6 θ-Homotopy Medial Axis Computation

In this section we present an algorithm for computing an approximate θ-HMA $\mathcal{M}^*_\theta$, given a homotopy preserving approximate Voronoi graph of the polyhedron. Given the approximate Voronoi graph, a sub-graph corresponding to a approximate medial axis of the polyhedron is computed using the property of Lemma 12 in [ER02]. Since the approximate Voronoi digram is homotopy preserving, the approximate medial axis corresponding to this sub-graph is homotopy equivalent to the exact medial axis $\mathcal{M}$. Hence, this approximate medial axis is a θ-HMA, for $\theta = 0$, denoted $\mathcal{M}_0^*$. The Voronoi faces, edges and vertices correspond to the medial axis sheets, seams and junctions respectively.

The diameter of a cell after the spatial subdivision gives a polygonal approximation to the geometric part of the Voronoi diagram. The approximation has bounded Hausdorff error to the exact Voronoi diagram, like the *Proximity Structure Diagram* [ER02]. This geometric approximation is used to construct a polygonal mesh approximation of the θ-HMA consisting of axis aligned faces.

Given the medial axis $\mathcal{M}_0^*$, our simplification algorithm is presented in Section 5.6.2 and it simplifies the medial axis by pruning sheets of the medial axis. We first define the separation angle of a sheet f_i to be the supremum of the separation angles for all points interior to the sheet:

$$\Theta(f_i) = \max_{\mathbf{x} \in \mathrm{Int}(f_i)} (\Theta(\mathbf{x})).$$

$\Theta(f_i)$ gives a measure of the stability of the sheet f_i. We use a conservative definition for the separation angle of the sheet to ensure that the simplified medial $\mathcal{M}_\theta^*$ is a superset of $\mathcal{M}_\theta$. Similarly we define the separation angle of a seam e_i as:

$$\Theta(e_i) = \max_{\mathbf{x} \in \mathrm{Int}(e_i)} (\Theta(\mathbf{x})).$$

5.6.1 Sheet Separation Angle Computation

The Voronoi diagram computation algorithm computes a piecewise linear approximation of each sheet based on a discrete sampling introduced by spatial subdivision. In this section, we address the problem of computing a bounded approximation of the sheet separation angle $\Theta(f_i)$. Each sheet of the medial axis of a polyhedron is trimmed quadric surface [Cul00]. Exact computation of the sheet separation angle involves computing the extreme value of a non-linear function on a quadric surface. Instead we present an efficient approach to compute a conservative upper bound on the separation angle using spatial subdivision. The tightness of the bound depends on the degree of subdivision. This approach fits well with our subdivision algorithm for computing the Voronoi graph.

Given a cell C and a sheet f_i intersecting the cell, our goal is to compute the maximum separation angle for all points on the sheet inside the cell. Let $\{p_1, p_2\}$ be the two governors of the sheet f_i and $\mathbf{c}$ be the center of the cell C. We classify the inputs into 2 cases:

1. The governors do not intersect the cell C, i.e. $C \cap \{p_1, p_2\} = \emptyset$
2. At least one of the governors intersects the cell C, i.e. $C \cap \{p_1, p_2\} \neq \emptyset$.

Case 1: $C \cap \{p_1, p_2\} = \emptyset$. We simplify the problem to computing the maximum of the separation angles for all points inside the cell to the two governors. We compute the separation angle from the center $\mathbf{c}$ of the cell to each of the two governors and add conservative error bounds to get the maximum separation angle. Let $\mathbf{x}$ be any point inside cell C. Let $\mathbf{n_i}(\mathbf{x})$ denote the normal vector from a point $\mathbf{x}$ to the sites p_i, $(i = 1, 2)$, and $\alpha_i(\mathbf{x})$ represent the angle between $\mathbf{n_i}(\mathbf{c})$ and $\mathbf{n_i}(\mathbf{x})$. If $\Delta\theta_i$ is an upper bound on $\alpha_i(\mathbf{x})$ for all $\mathbf{x} \in C$, then the maximum separation angle for sheet f_i inside cell C is given by:

$$\Theta(f_i \cap C) \leq \cos^{-1}\left(\frac{\mathbf{n_1}(\mathbf{c}) \cdot \mathbf{n_2}(\mathbf{c})}{|\mathbf{n_1}(\mathbf{c})||\mathbf{n_2}(\mathbf{c})|}\right) + \Delta\theta_1 + \Delta\theta_2$$

The computation of the error bounds $\Delta\theta_i$ for each of the three types of governors (point site, line site and triangle site) is presented below:

Point Site $p_i : \mathbf{p}$. The range of angles subtended by a point site to all points in the cell is given by a normal cone. The normal cone is the smallest cone enclosing the cell C with the apex at $\mathbf{p}$ and axis along $\mathbf{n_i}(\mathbf{c})$ (see figure 5.8). Let $\Delta\theta_i$ be the half opening angle of the cone. The angle $\alpha_i(\mathbf{x})$ is maximized when point $\mathbf{x}$ is one of the corner vertices $\mathbf{v_j}$ $(1 \leq j \leq 8)$ of the cell C. Thus, for the smallest cone enclosing the cell C,

$$\Delta\theta_i = \max_{1 \leq j \leq 8}\left[\cos^{-1}\left(\frac{\mathbf{n_i}(\mathbf{v_j}) \cdot \mathbf{n_i}(\mathbf{c})}{|\mathbf{n_i}(\mathbf{v_j})||\mathbf{n_i}(\mathbf{c})|}\right)\right], \text{ where } \mathbf{n_i}(\mathbf{x}) = \mathbf{p} - \mathbf{x}.$$

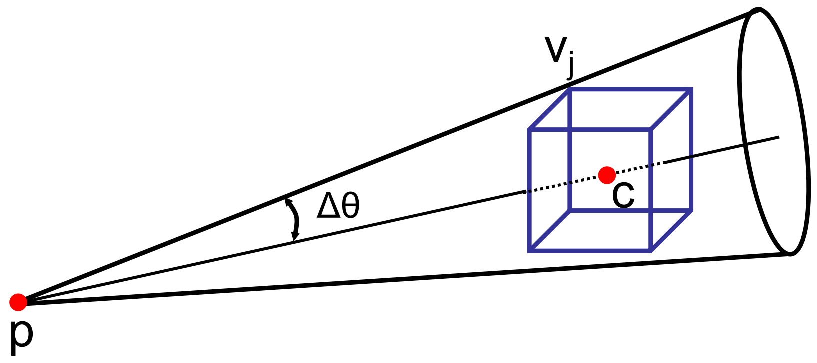


Figure 5.8: Normal Cone to compute $\Delta\theta$ for a point site

Line Site $p_i : \mathbf{p} + \lambda(\mathbf{q} - \mathbf{p})$. The range of angles subtended by a line to all the points in the

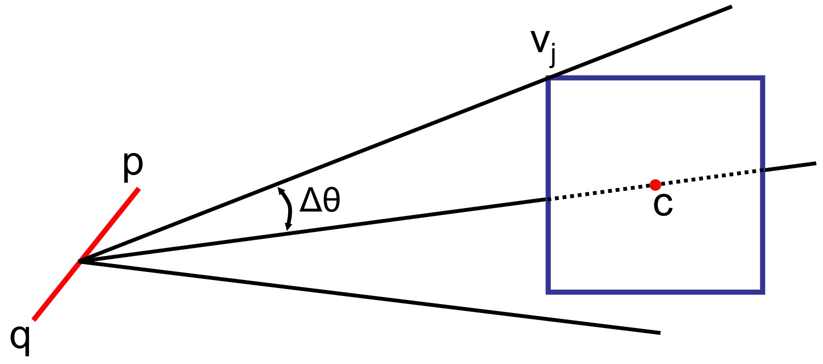


Figure 5.9: Wedge to determine $\Delta\theta$ for a line site

cell is given by the smallest wedge enclosing the cell, with the top edge of the wedge being the line site (see figure 5.9). Let $\Delta\theta_i$ be the half angle of the wedge. As in the point site case, the angle $\alpha_i(\mathbf{x})$ is maximized when point $\mathbf{x}$ is one of the corner vertices $\mathbf{v_j}$ $(1 \leq j \leq 8)$ of the cell C. Thus, for the smallest wedge enclosing the cell C,

$$\Delta\theta_i = \max_{1\leq j\leq 8}\left[\cos^{-1}\left(\frac{\mathbf{n_i}(\mathbf{v_j})\cdot\mathbf{n_i}(\mathbf{c})}{|\mathbf{n_i}(\mathbf{v_j})||\mathbf{n_i}(\mathbf{c})|}\right)\right],$$

$$\text{where } \mathbf{n_i}(\mathbf{x}) = \mathbf{p} + \lambda(\mathbf{q} - \mathbf{p}) - \mathbf{x},\, \lambda = \frac{(\mathbf{x} - \mathbf{p})\cdot(\mathbf{q} - \mathbf{p})}{(\mathbf{q} - \mathbf{p})^2}.$$

Triangle Site p_i with face normal $\hat{\mathbf{n}}$. The shortest path from any point to the triangle is perpendicular to the face. Thus $\mathbf{n_i}(\mathbf{x}) = \hat{\mathbf{n}}$ for all $\mathbf{x}$, and $\Delta\theta_i = 0$.

Case 2 $C \cap \{p_1, p_2\} \neq \emptyset$. If the two sites do not intersect ($p_1 \cap p_2 = \emptyset$), then the bisector surface (and sheet f_i) also do not intersect either site. In such a case we can subdivide the cell C into sub-cells $\{C_k\}$ such that $C_k \cap \{p_1, p_2\} = \emptyset$ if $C \cap f_i \neq \emptyset$. The computation of the sheet separation angle is then reduced to Case 1.

If the two sites intersect ($p_1 \cap p_2 \neq \emptyset$), then the sheet corresponds to one of the non-generic cases of a bisector surface [Cul00], and the separation angle $\Theta(f_i)$ can be determined exactly from the pairs of governors. The case of two point governors case never occurs, we examine each of the other 5 pairs of governors individually.

Point-Triangle The bisector surface is a redundant line, and never occurs on the medial axis [Cul00].

Point-Line The bisector surface is a plane through the point and perpendicular to the line, $\Theta(f_i) = 0$

Line-Line The bisector surface is an orthogonal plane pair, $\Theta(f_i)$ = angle between the two lines.

Line-Triangle The bisector surface is a right circular cone, or a plane if the line is incident on the triangle. In first case, the separation angle $\Theta(f_i) = \pi/2 - \cos^{-1}(\hat{\mathbf{l}} \cdot \hat{\mathbf{n}})$, where $\hat{\mathbf{l}}$ and $\hat{\mathbf{n}}$ are unit normals along the line and to the triangle respectively. In the second case, $\Theta(f_i) = 0$.

Triangle-Triangle The bisector surface is an orthogonal plane pair, and the separation angle is given by $\Theta(f_i) = \cos^{-1}(\hat{\mathbf{n_1}} \cdot \hat{\mathbf{n_2}})$, where $\hat{\mathbf{n_1}}, \hat{\mathbf{n_2}}$ are unit normals to the two triangles.

5.6.2 Simplification Algorithm

We now present our medial axis simplification algorithm. We treat the medial axis $\mathcal{M}$ as an abstract 2-dimensional complex consisting of faces, edges, and vertices. Initially, edges correspond either to the seam curves, which lie between sheets, or rim curves, which lie on the boundary.

The key idea in our algorithm is a simple criterion for determining whether a sheet can be removed without changing the homotopy type of the medial structure. We call such sheets *frontier sheets* (Figure 5.10). We will describe this criterion below, but first give an overview of how it is used in the algorithm. We maintain a set $\mathcal{Q}$ of all frontier sheets. We successively

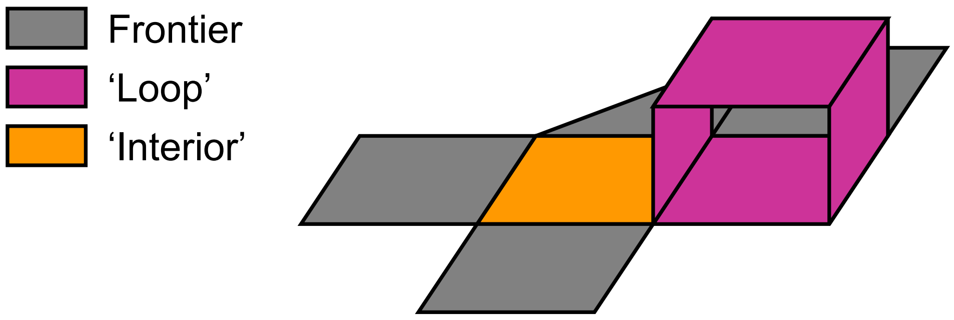


Figure 5.10: Classification of sheets for iterative pruning*: The sheets colored gray are frontier sheets, and can be removed without changing the homotopy type. For the 'loop' sheets the rim set is not connected and they will never become frontier sheets. The 'interior' sheet has an empty rim set, however it may become a frontier sheet after removal of one of its adjacent sheets.*

remove sheets from this set until it is empty. As each sheet is removed from $\mathcal{Q}$, it is also removed from the medial structure if its separation angle is no greater than θ. Removal of a sheet from the structure can affect whether its neighbors are frontier sheets, and so each time we remove a sheet we check each neighbor of that sheet to see if it needs either to be added or removed.

A sheet f_i is defined to be a frontier sheet provided that its set of rim points $\mathcal{R}(f_i)$ and its set of seam points $\mathcal{S}(f_i)$ are both connected and nonempty. The set $\mathcal{Q}$ is defined as:

$$\begin{aligned}\mathcal{Q} \;=\; & \{f_i \mid \mathcal{R}(f_i) \neq \emptyset, \mathcal{R}(f_i) \text{ is connected}, \\ & \mathcal{S}(f_i) \neq \emptyset, \mathcal{S}(f_i) \text{ is connected}\}. \end{aligned} \tag{5.1}$$

In Section 5.7.2 we will prove that the frontier sheets are precisely those sheets which may be removed without changing the homotopy type. We present an intuitive justification for that claim here. If the rim set and seam set are both connected then each set is a single curve, and removing the sheet is equivalent to retracting the sheet onto its seam set via a homotopy (see Figure 5.11(a)). On the other hand, if the rim set is disconnected or empty, removing the

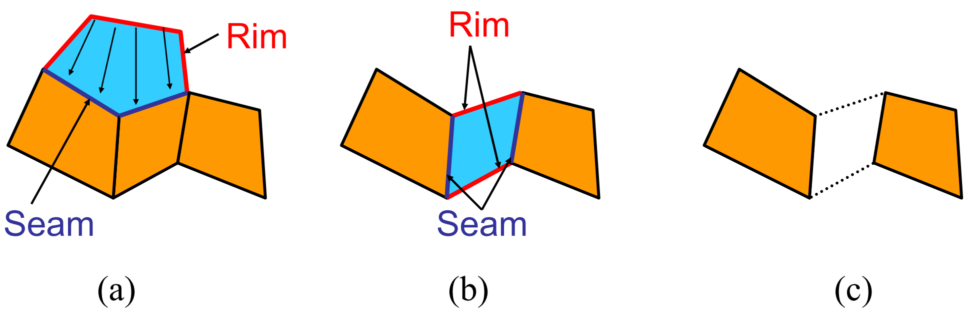


Figure 5.11: Sheet pruning*: (a) The cyan sheet is a valid frontier sheet, and has a deformation retract to its seam set. (b) The cyan sheet is not a frontier sheet. (c) Removing the sheet makes the two adjacent sheets disconnected.*

sheet removes a path between two points on different seam components and hence does not preserve the homotopy type (see Figures 5.11(b),(c)). Note that, when we remove a sheet, we remove its interior and its rim set, but not its seam set.

We noted earlier that removing a sheet can cause other sheets either to lose or gain frontier status, and we can now explain why this is true. A sheet with an empty rim set can gain a rim edge if one of its neighboring sheets is removed, and thereby become a frontier sheet. Conversely, a sheet with a single seam component can find that its seam set is broken into two components if an adjacent sheet is removed.

```
   Input: Initial medial axis $\mathcal{M}_0$, angle $\theta$
   Output: Final medial subset $\mathcal{M}_f$
1  Label all sheets in $\mathcal{M}$ as unmarked
2  Initialize $\mathcal{Q}_0$, $j \leftarrow 0$
3  repeat
4      $f_i \leftarrow$ ExtractSheet ($\mathcal{Q}_j$)
5      $(\mathcal{Q}_{j+1}, \mathcal{M}_{j+1}) \leftarrow$ RemoveSheet ($f_i$, $\mathcal{Q}_j$, $\mathcal{M}_j$, $\theta$)
6      $j \leftarrow j + 1$
7  until ($\mathcal{Q}_j = \emptyset$)
8  $\mathcal{M}_f \leftarrow \mathcal{M}_{j+1}$
```

Algorithm 6: *SimplifyMAT($\mathcal{M}_0$, θ): Computes a simplified medial axis $\mathcal{M}_f$, given an initial medial axis $\mathcal{M}_0$ and a separation angle θ.*

Algorithm 6 simplifies $\mathcal{M}$ based on removal of frontier sheets. Let the resulting medial subset after the jth iteration be $\mathcal{M}_j$, and let the corresponding frontier set be $\mathcal{Q}_j$. The frontier set is maintained as a priority queue, the priority determined by the sheet separation angle. Initially, $\mathcal{M}_0 = \mathcal{M}_0^*$. $\mathcal{Q}_0$ is computed using $\mathcal{M}_0$ in equation (5.1). The function *ExtractSheet($\mathcal{Q}_j$)* in line 4 returns a sheet with minimum separation angle from the set $\mathcal{Q}_j$ (but does not remove it from $\mathcal{Q}_j$). The key step in the algorithm is the removal of a frontier sheet in line 5, which is described in Algorithm 7.

```
Input: A frontier sheet $f_i$, frontier set $\mathcal{Q}_j$, medial subset $\mathcal{M}_j$, angle $\theta$
Output: Frontier set $\mathcal{Q}_{j+1}$, medial subset $\mathcal{M}_{j+1}$
1 if $\Theta(f_i) \geq \theta$ then
2     Label $f_i$ as fixed
3     $\mathcal{Q}_{j+1} \leftarrow \mathcal{Q}_j \setminus \{f_i\}$
4 else
5     $\mathcal{M}_{j+1} \leftarrow \mathcal{M}_j \setminus \{f_i\}$
6     $\mathcal{Q}_j \leftarrow \mathcal{Q}_j \setminus \{f_i\}$
7     $\mathcal{Q}_{j+1} \leftarrow$ UpdateFrontierNbrs$(f_i, \mathcal{Q}_j)$
8 end
```

***Algorithm 7**: RemoveSheet(f_i, $\mathcal{Q}_j$, $\mathcal{M}_j$, θ): Removes a frontier sheet f_i satisfying the separation angle threshold θ from a medial subset $\mathcal{M}_j$. The frontier set $\mathcal{Q}_j$ is also updated.*

Algorithm 7 removes a frontier sheet from the medial subset $\mathcal{M}_j$ only if the separation angle of the sheet lies below the angle threshold θ. (line 1). The removal of a frontier sheet does not change the homotopy type of $\mathcal{M}_j$. As we noted earlier, removal of the sheet from $\mathcal{M}_j$ may change the frontier status of its neighboring sheets. Neighboring sheets are checked for such changes and the frontier set is updated in (line 7), which is described in detail as Algorithm 8.

```
    Input: A frontier sheet $f_i$, frontier set $\mathcal{Q}_j$
    Output: Frontier set $\mathcal{Q}_{j+1}$
1 Initialize $\mathcal{Q}_{j+1} \leftarrow \mathcal{Q}_j$
2 foreach sheet $f_k$ sharing a seam point with $f_i$ do
3     if (Label($f_k$) $\neq$ fixed) then
4         if ($f_k$ is a frontier sheet) then
5             $\mathcal{Q}_{j+1} \leftarrow \mathcal{Q}_{j+1} \cup \{f_k\}$
6         else
7             $\mathcal{Q}_{j+1} \leftarrow \mathcal{Q}_{j+1} \setminus \{f_k\}$
8 end
```
Algorithm 8: *UpdateFrontierNbrs(f_i, $\mathcal{Q}_j$): Updates the frontier set $\mathcal{Q}_j$ after removal of a frontier sheet f_i.*

5.7 Correctness

In this section we demonstrate that Algorithm 6 is correct, i.e. the final medial subset is a valid θ-HMA. Let $\mathcal{M}_f$ be the subset of $\mathcal{M}_0^*$ obtained as the final out of algorithm 6. To prove correctness, we must show that $\mathcal{M}_f$ contains $\mathcal{M}_\theta$, $\mathcal{M}_f$ has the homotopy type of $\mathcal{M}$, and $\mathcal{M}_f$ is irreducible. We will first show that $\mathcal{M}_\theta \subset \mathcal{M}_f$.

5.7.1 Separation Angles of Medial Axis Parts

It is clear from the definition of the separation angle for a sheet that every sheet interior point that is removed will have a separation angle no greater than the threshold θ. So it remains to show that no seam or junction point is removed if its separation angle is greater than θ.

The set of governors for all points in the interior of the sheet, and on the boundary curves, remains the same and each governor is linear. Thus the separation angle $\Theta(x)$ is a continuous function of all points in the interior of a sheet, and on the rim points on the boundary of the sheet. However, the set of governors changes at a seam or a junction, causing the separation angle to be discontinuous on the boundary of the sheet (figure 5.12). Lemma 5.4 bounds the discontinuity in the separation angle at the seam and junction boundaries of a sheet.

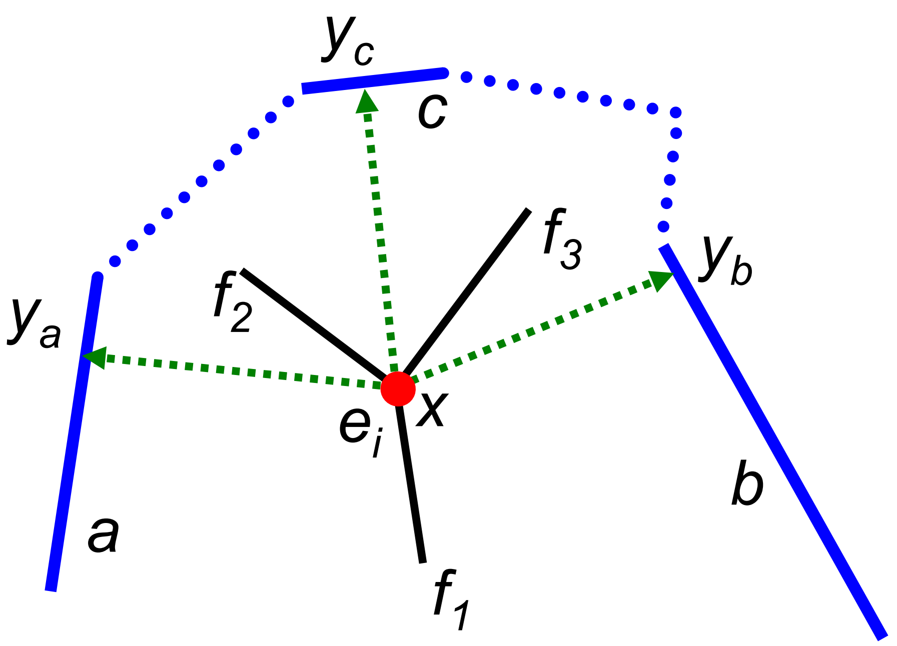


Figure 5.12: Separation angle of a seam e_i: Three sheets f_1, f_2 and f_3 meet at a seam e_i. For any point $\mathbf{y}$ on e_i, $\Theta(y) \leq \Theta(f_1)$.

Lemma 5.4.

(i) Let e_i be a (non-degenerate) seam of a medial axis, formed by intersection of three sheets f_1, f_2 and f_3. Then, $\Theta(e_i) \leq \max_{1\leq j\leq 3}(\Theta(f_j))$

(ii) Let v_i be a (non-degenerate) junction of a medial axis, formed by intersection of four sheets f_1, f_2, f_3 and f_4. Then, $\Theta(v_i) \leq \max_{1\leq j\leq 4}(\Theta(f_j))$

Proof. (i) Let the set of governors of sheet f_1 be $\mathcal{G}(f_1) = \{a, b\}$. Since f_1 and f_2 intersect, $\mathcal{G}(f_1) \cap \mathcal{G}(f_2) \neq \emptyset$. Also $\mathcal{G}(f_1) \neq \mathcal{G}(f_2)$ as two intersecting sheets cannot have same set of governors. Thus $|\mathcal{G}(f_1) \cap \mathcal{G}(f_2)| = 1$, and $\mathcal{G}(f_2) = \{b, c\}$. Similarly $\mathcal{G}(f_3) = \{a, c\}$, and the set of governors of e_i is $\mathcal{G}(e_i) = \{a, b, c\}$. For any point $\mathbf{x} \in e_i$, the closest points on a, b and c be $\mathbf{y_a}$, $\mathbf{y_b}$, $\mathbf{y_c}$. Then $NB(x) = \{\mathbf{y_a}, \mathbf{y_b}, \mathbf{y_c}\}$, and by definition of

$\Theta(\mathbf{x})$,

$$\Theta(\mathbf{x}) = \max(\angle \mathbf{y_a x y_b}, \angle \mathbf{y_b x y_c}, \angle \mathbf{y_a x y_c})$$
$$= \angle \mathbf{y_a x y_b} \text{ (assume WLOG)}$$

Let $\mathbf{y}$ be a point on sheet f_1 inside a δ-neighborhood of $\mathbf{x}$. Since $\Theta(f_i)$ is continuous and $\mathcal{G}(f_1) = \{a, b\}$, $\lim_{\mathbf{y}\to\mathbf{x}} \Theta(\mathbf{y}) = \angle \mathbf{y_a x y_b} = \Theta(\mathbf{x})$. By definition of $\Theta(f_1)$, $\Theta(f_1) \geq \lim_{\mathbf{y}\to\mathbf{x}} \Theta(\mathbf{y})$. Hence, $\Theta(x) \leq \Theta(f_1)$. Since choice of point $\mathbf{x}$ on e_i was arbitrary,

$$\Theta(e_i) = \Theta(\mathbf{x}) \leq \Theta(f_1) \leq \max_{1\leq j\leq 3}(\Theta(f_j))$$

(ii) Proof follows as above, using 4 governors of the junction, instead of 3 governors of the seam.

□

The implication of Lemma 5.4 is that we can get an upper bound on the separation angle of a non-degenerate seam (junction) from the separation angles of the incident sheets. This ensures that during simplification, if a seam (junction) belongs to $\mathcal{M}^*_\theta$, then at least one of the incident sheets will belong to $\mathcal{M}^*_\theta$. Conversely, if all incident sheets do not belong to $\mathcal{M}^*_\theta$, then the seam (junction) will not belong to $\mathcal{M}^*_\theta$. Hence, it suffices to compute separation angles and test the sheets for pruning during simplification.

Lemma 5.5. *For a non-degenerate $\mathcal{M}$, $\mathcal{M}_\theta \subseteq \mathcal{M}_f$.*

Proof. Let $\mathbf{x} \in \mathcal{M}_\theta$, i.e. $\Theta(x) \geq \theta$. If $\mathbf{x}$ is in the interior of a sheet f_i, then $\Theta(f_i) \geq \theta$. If $\mathbf{x}$ is in the interior of a seam e_j, then by Lemma 5.4, $\Theta(f_i) \geq \theta$ for some sheet f_i incident on that seam. Thus, f_i will never be removed from the medial subset, and so e_j, being incident on f_i will be in $\mathcal{M}_f$. Therefore, $\mathbf{x} \in e_j$ will also be in $\mathcal{M}_f$. In the same way, Lemma 5.4 also implies that $\mathbf{x} \in \mathcal{M}_f$ if $\mathbf{x}$ is a junction point. □

5.7.2 Homotopy Preservation

Lemma 5.6. *$\mathcal{M}_f$ is homotopy equivalent to $\mathcal{M}$.*

Proof. We perform induction on j. Since $\mathcal{M}_0^*$ is computed from a homotopy preserving approximate Voronoi diagram, $\mathcal{M}_0$ is homotopy equivalent to $\mathcal{M}$. Our proof is complete if we show that $\mathcal{M}_j$ is homotopy equivalent to $\mathcal{M}_{j+1}$, or, equivalently, that removing a frontier sheet f_i does not change the homotopy type. If both the seam set $\mathcal{S}(f_i)$ and the rim set $\mathcal{R}(f_i)$ are non-empty and connected, then the boundary of the sheet can have at most two components. If the boundary has one component, then the sheet is a topological disk, with a boundary consisting of two curves, the seam set and the rim set. If the boundary has two components, then one component must be the seam set, and the other the rim set. In that case, the sheet is an annulus, which can also be retracted onto the seam set.

The existence of a retraction means that there is a map $h : f_i \rightarrow \mathcal{S}(f_i)$ such that (a) the restriction of h to $\mathcal{S}(f_i)$ is equal to the identity on $\mathcal{S}(f_i)$, and (b) $g \circ h$ is homotopic to the identity on f_i, where g is the inclusion $\mathcal{S}(f_i) \rightarrow f_i$. We can then define $\hat{h} : \mathcal{M}_j \rightarrow \mathcal{M}_{j+1}$ to be equal to the identity on $\mathcal{M}_{j+1} \subset \mathcal{M}_j$, and equal to h on f_i. Then, if $\hat{g}$ is the inclusion $\mathcal{M}_{j+1} \rightarrow \mathcal{M}_j$, it is clear that $\hat{h} \circ \hat{g}$ is equal to the identity on $\mathcal{M}_{j+1}$, and $\hat{g} \circ \hat{h}$ is homotopic to the identity on $\mathcal{M}_j$. Thus, the two spaces are homotopy equivalent to one another. □

Lemma 5.7. *$\mathcal{M}_f$ is irreducible.*

Proof. Let f_i be any frontier sheet in the final subset $\mathcal{M}_f$. Then f_i is labeled fixed, and either $\Theta(f_i) \geq \theta$ or f_i is an isolated component. Thus $\mathcal{M}_f \setminus \{f_i\}$ is not a subset of $\mathcal{M}_\theta$, or does not have the same number of components as $\mathcal{M}$.

Let f_i be any non-frontier node in the final connectivity graph $\mathcal{M}_f$. If $\Theta(f_i) \geq \theta$, $\mathcal{M}_f \setminus \{f_i\}$ is not a subset of $\mathcal{M}_\theta$. If $\Theta(f_i) < \theta$, then $\mathcal{M}_f \setminus \{f_i\}$ is not homotopy equivalent to $\mathcal{M}_f$.

We prove this by treating $\mathcal{M}_j$ as a cell complex and considering its Euler characteristic. A 2-dimensional cell complex in $\mathbb{R}^3$ is a space that can be decomposed into open topological disks (faces), open curves (edges) and points (vertices) in such a way that the boundary of each face is a union of edges and vertices from the decomposition, and the boundary (that is, the endpoints) of each edge are vertices from the decomposition. Strictly speaking, the medial axis is not a cell complex, because the curves bounding frontier sheets are not in general part of the medial axis. However, we may add abstract edges without changing the homotopy type to construct a cell complex. The Euler characteristic, given by $\chi = F - E + V$ where F, E, and V are the numbers of faces, edges and vertices respectively, is a well-known homotopy invariant (see, e.g., [Spa89]).

When we remove a sheet f_i from $\mathcal{M}_j$ to get $\mathcal{M}_{j+1}$, we remove all of the faces, edges, and vertices of f_i except for the edges and vertices that are part of the seam set $\mathcal{S}(f_i)$. Thus, the change in Euler characteristic resulting from removing the sheet is given by

$$\chi(\mathcal{M}_j) - \chi(\mathcal{M}_{j+1}) = \chi(f_i) - \chi(\mathcal{S}(f_i)).$$

We wish to show that $\chi(f_i) - \chi(\mathcal{S}(f_i))$ is nonzero unless f_i is a frontier sheet.

The sheet f_i (which is connected by definition) is homotopy equivalent to a disk with n holes removed, for some n. The Euler characteristic of such a complex is given by $\chi = 1 - n$. The seam set consists of components of two types. There are loops, for which the number of vertices equals the number of edges, and $\chi = 0$. There are also unclosed chains of edges, for which there is one more vertex than edges, and $\chi = 1$. Therefore, $\chi(\mathcal{S}(f_i))$ cannot be negative, so that there are only two ways $\chi(f_i) - \chi(\mathcal{S}(f_i))$ can be zero. First we may have $\chi(f_i) = \chi(\mathcal{S}(f_i)) = 0$, in which case f_i is an annulus with connected, non-empty seam and rim sets. Second, we may have $\chi(f_i) = \chi(\mathcal{S}(f_i)) = 1$, in which case f_i is a (topological) disk, also with connected seam and rim sets. These cases are precisely the two kinds of frontier sets. $\square$

Together, the foregoing results show that $\mathcal{M}_f = \mathcal{M}^*_\theta$, as desired.

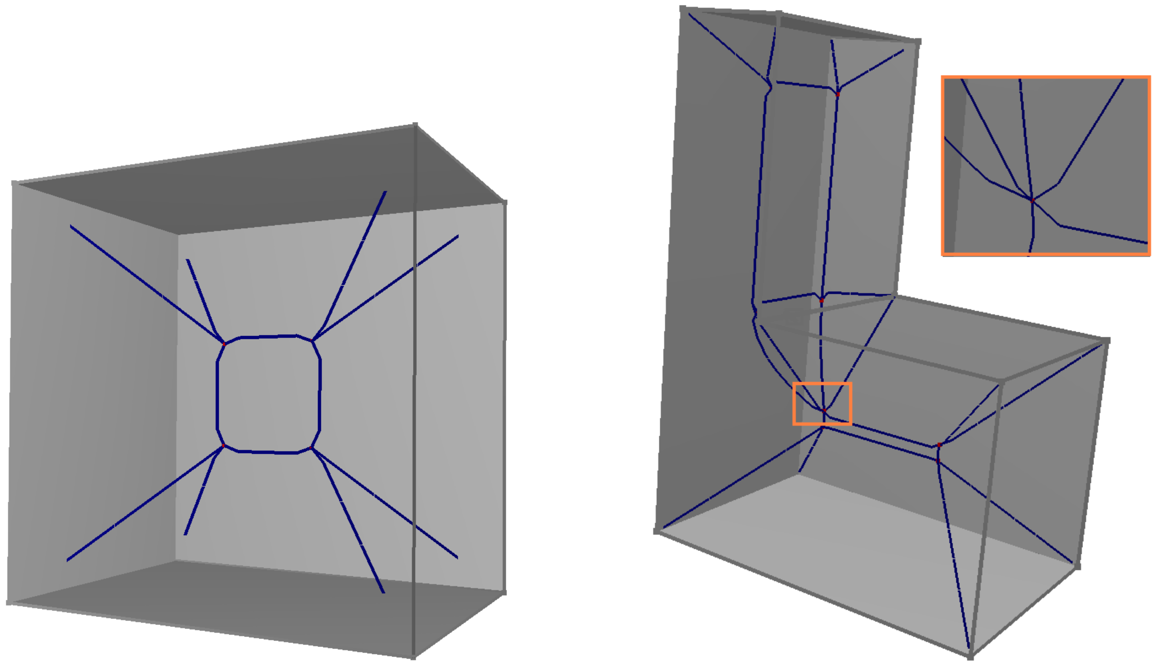

(a) Cuboid (b) L-Shape

Figure 5.13: The homotopy preserving approximate Voronoi diagram is computed for two simple models with degeneracies. *The edges of the approximate Voronoi diagram are shown in blue. The vertices are highlighted with red. (a) A cuboid with 2 equal dimensions. (b) An L-shape. The orange region shows a zoomed in view of a degenerate vertex with* 6 *seams incident on it.*

5.8 Implementation and Results

In this section, we briefly describe our implementation and highlight its performance on different benchmarks.

5.8.1 Implementation

We have implemented the system in C++, and use OpenGL to display the results. The timings reported in this paper were taken on a 2.4Ghz Opteron PC with 2GB of memory. The discrete distance field and spatial grid is computed efficiently using graphics hardware as presented in Chapter . The resolution of the uniform grid was chosen to be half of the length of the smallest edge of the polyhedron to ensure satisfiability of Condition (1) of the boundary criterion.

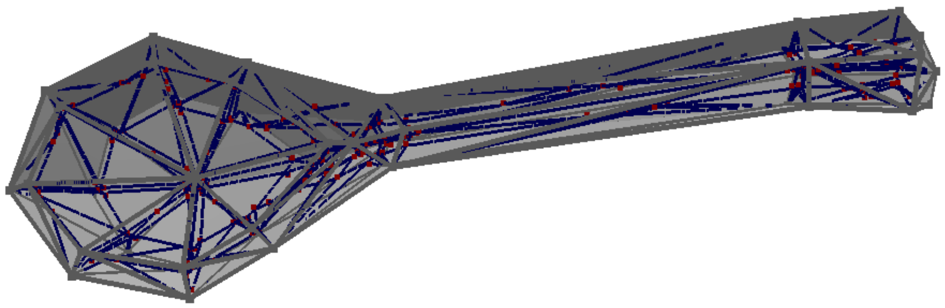


Figure 5.14: Spoon Model: The model has 254 sites, including 84 triangle sites, 126 edge sites, and 44 vertex sites. The computation for homotopy preserving approximate Voronoi diagram took 1.7s for this model. The edges and vertices of the approximate Voronoi graph are highlighted in blue and red respectively.

To test if a sheet f_i is a frontier sheet, we first extract a sub-graph of the connectivity graph. The sub-graph corresponds to f_i and its incident set of seam curves $\mathcal{S}(f_i)$. We then perform a depth-first-search on the sub-graph to determine the number of components in $\mathcal{S}(f_i)$ and in $\mathcal{R}(f_i)$). If sheet f_i is a frontier sheet, then number of components in $\mathcal{R}(f_i)$ and $\mathcal{S}(f_i)$ is 1. Iterative pruning during the medial axis simplification algorithm involves removal of nodes corresponding to the sheets and incident seam curves. The final graph captures the connectivity of the θ-HMA. The priority queue $\mathcal{Q}_j$ is implemented as a heap.

5.8.2 Approximate Voronoi Diagram Computation

We have tested our algorithm to compute the homotopy preserving approximate Voronoi diagram on a set of examples ranging from simple geometry with known degenerate configurations to more complex models consisting of thousands of sites.

Figure 5.13 shows a cuboid and an L-bracket with symmetric cubical sections. The models contain degenerate seams and junctions. Figure 5.14) shows a spoon model with 254 sites. Figure 5.15) shows a flattened chisel model with a radial axis of symmetry and random perturbations added to the handle. This benchmark is particularly difficult to handle with many several degenerate configurations near the axis of the handle. As a result, there is a large governor set for many cells.

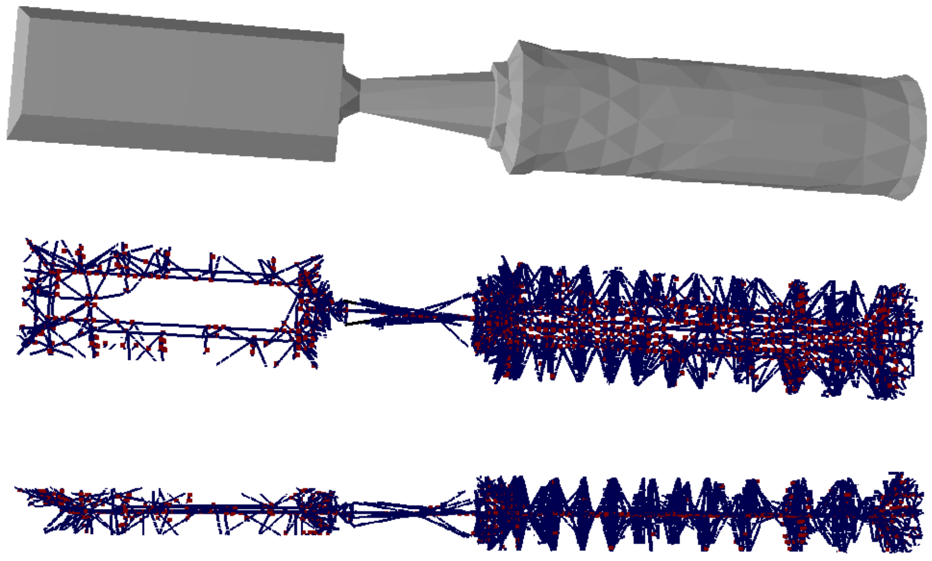
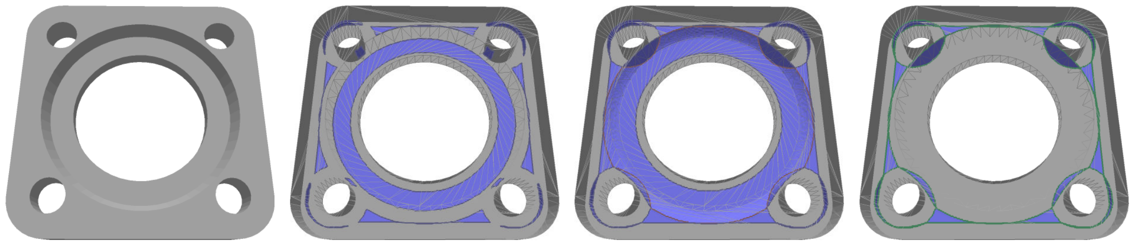


Figure 5.15: Chisel Model: The model has $1,797$ *sites, including 632 triangle sites, 847 edge sites, and 318 vertex sites. It has many degenerate configurations near the axis of the handle. Two views of the approximate Voronoi graph are shown in the bottom. The computation for homotopy preserving approximate Voronoi diagram took* $130.3s$ *for this model. The edges and vertices of the approximate Voronoi graph are highlighted in blue and red, respectively.*

5.8.3 θ-Homotopy Medial Axis Computation

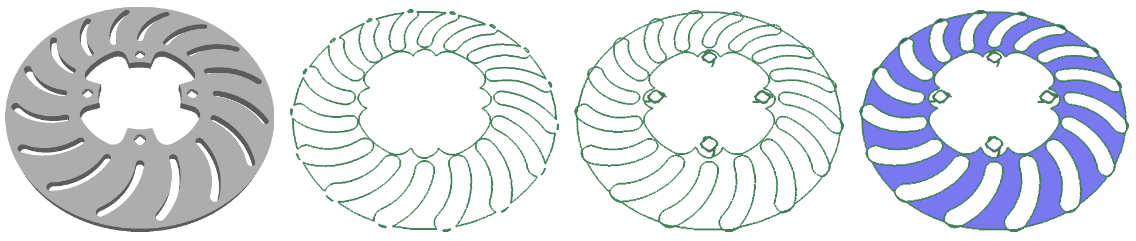
We have also applied our algorithm to compute the θ-HMA of polyhedral models of various sizes, ranging from 1000 triangles to 60k triangles. The complexity of the Blum medial axis ranged from 1.3k sheets to 89k sheets. Our benchmark models include CAD models with many sharp edges and high-aspect-ratio triangles. Such models can be relatively hard for medial axis algorithm that compute a point sampling on the boundary of the objects and a Voronoi diagram of the point samples.

Some of the benchmark models have a high genus and holes that are preserved during medial axis simplification. We also tested our algorithm on synthetic benchmark models



(a) Model (b) θ-SMA (c) θ-HMA (d) θ-HMA

Figure 5.16: Flange Plate Model (990 polygons): Medial axis sheets through a cut-out of the model (b) The sheets become disconnected, holes disappear for the θ-SMA ($\theta = 150°$). (c) In the θ-HMA the holes are preserved, and the entire medial axis remains connected ($\theta = 150°$). (d) The homotopy is preserved even as θ is set to maximum value of $\theta = 180°$



(a) Model (b) θ-SMA (c) θ-HMA (d) θ-HMA sheets

Figure 5.17: Brake Rotor Model ($4.7k$ polygons): Rim curves are shown in green, $\theta = 150°$ (b) The small holes in the center disappear in the θ-SMA, and the outer boundary becomes disconnected. (c) the θ-HMA the holes in the center are preserved, and the entire medial axis remains connected. (d) the entire θ-HMA with the sheets and rim curves.

with cavities and tunnels introduced obtained by performing boolean operations with various solids. A topologically accurate polygonal boundary approximation of CSG operations is computed using techniques presented in [VKSM04].

For simplicity, in the figures we only show seam curves that are the intersections of three or more sheets. Also, maximally connected 2-manifolds have been grouped into one sheet. The models and their corresponding medial axes are shown in Figures 5.16 - 5.23. The polygonal meshes corresponding to the θ-SMA and θ-HMA have been smoothed using Taubin's algorithm [Tau95]. Table 5.2 lists the complexity of the polyhedral models, and of the original medial axis and corresponding simplifications θ-HMA and θ-SMA. The time to simplify the Blum medial axis to θ-HMA is also listed.

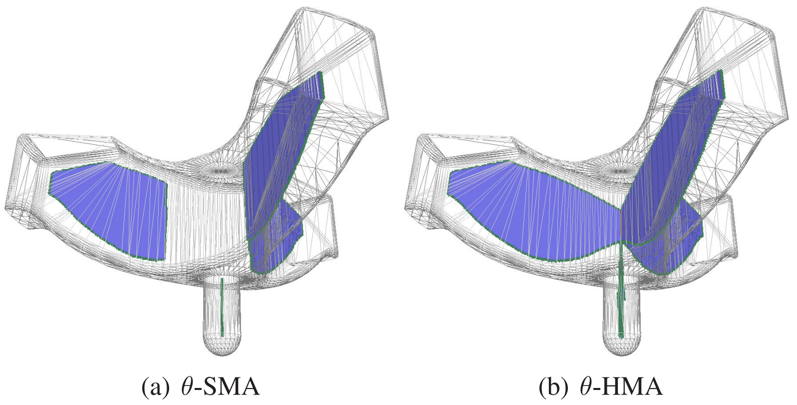
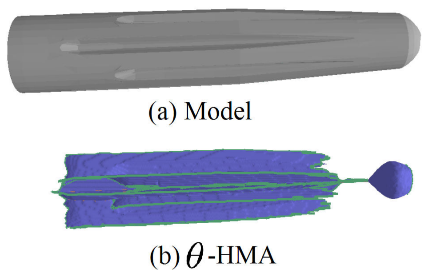

(a) θ-SMA (b) θ-HMA

Figure 5.18: Primer Anvil Model ($4.3k$ polygons): Model boundary is shown in wireframe. The medial axis sheets are in blue, rim curves in green, seam curves in magenta, $\theta = 150°$ (a) In the θ-SMA the sheets become disconnected, a thin sheet remains at the bottom (b) In the θ-HMA the sheets remain connected.


(a) Model

(b) θ-HMA

Figure 5.19: Ridged Rod ($5k$ polygons), $\theta = 120°$: The model has ridges near the surface, around which θ value is high. The medial sheets are shown in (b).

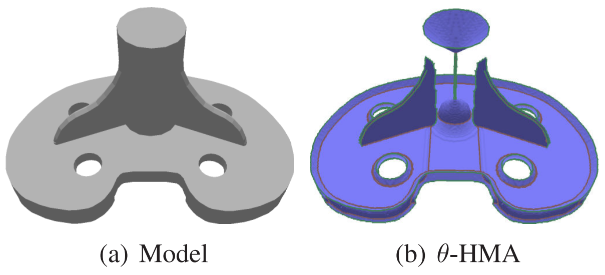
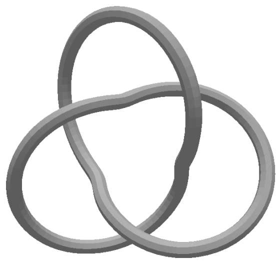

(a) Model (b) θ-HMA

Figure 5.20: CAD Mount ($2.4k$ polygons), $\theta = 45°$: The sheets emerging from the center of the vertical rod have low separation angle and have been removed. Note that the removal does not change the homotopy type.

Choice of angle θ: The angle θ used to guide the level simplification is provided as a user-defined parameter. The values of θ used in the results presented were chosen experimentally such that the computed θ-HMA exhibited significant simplification while preserving the topological structure of the medial axis. A statistical scheme for selecting the value of θ has been presented in [AM97].

Model	**Polys**	θ	**Num Sheets**			**Time**
		($^\circ$)	BMA	θ-HMA	θ-SMA	(s)
Plate	990	150	1896	21	22	1.29
Rotor	4736	150	1365	41	17	1.23
Mount	2442	45	7455	536	283	2.43
Knot	2562	150	13940	451	386	5.01
Ridge-Rod	5012	120	30676	74	36	18.10
Anvil	4340	150	32102	4	4	17.51
Drivewheel	60712	90	89885	16	759	24.94
Drivewheel	60712	150	89885	3	4	26.05

Table 5.2: Medial Axis Complexity: Polygon and sheet count of various models. θ is the separation angle (in degrees) used for computing θ-HMA and θ-SMA. Num Sheets refers to number of sheets in the exact Blum medial axis (BMA), and the simplified θ-HMA and θ-SMA. Time is the time in seconds used by Algorithm 6 to compute the θ-HMA from the Blum medial axis.


(a) Model

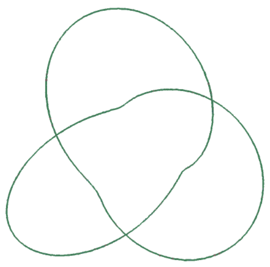

(b) θ-HMA curves

Figure 5.21: Knot Model ($2.5k$ polygons), $\theta = 150^\circ$: Rim curves are shown in bold green. The sheets consist of thin and long surfaces.

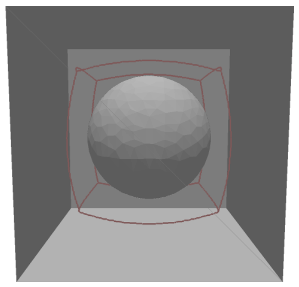
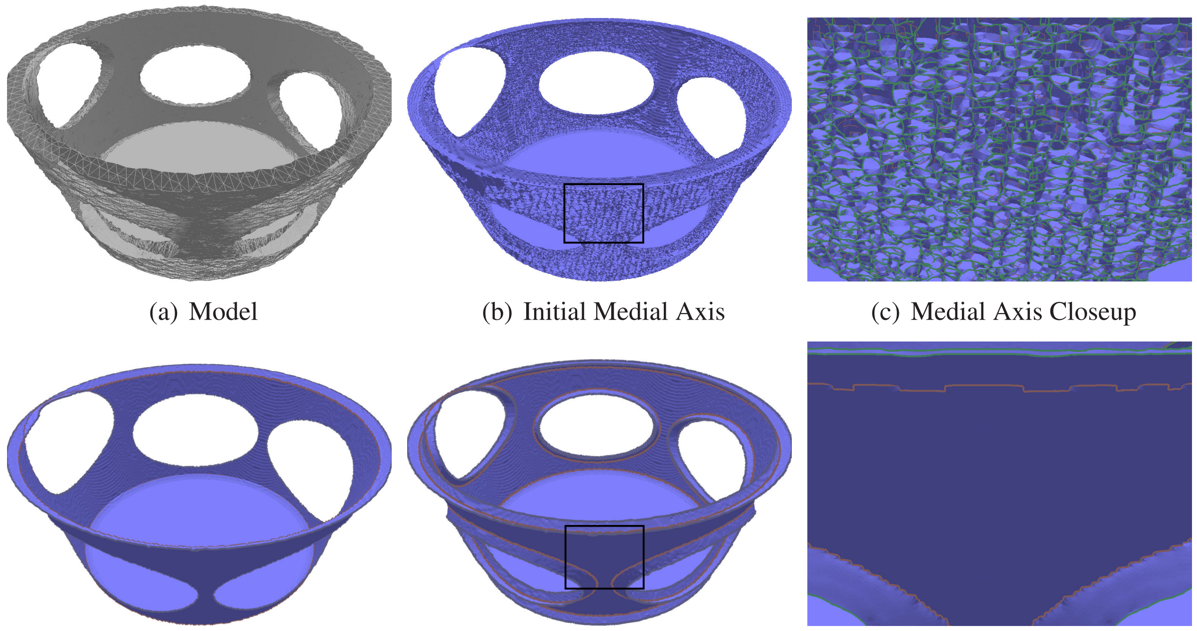


Figure 5.22: Cube with spherical void ($1.5k$ polygons: A cut-out showing the cube and the spherical void in the center. The θ-HMA curves are drawn in magenta, $\theta = 180^\circ$. The θ-HMA remains connected, and preserves the void.

5.9 Discussion

In this section we perform an analysis of the individual stages of our homotopy-preserving approximate Voronoi diagram computation algorithm and compare it with prior techniques. We also analyze the performance of our simplification algorithm. We highlight its computational complexity, and perform comparisons with some related algorithms for medial axis simplification.



(a) Model (b) Initial Medial Axis (c) Medial Axis Closeup

(d) θ-HMA sheets, $\theta = 150°$ (e) θ-HMA sheets, $\theta = 90°$ (f) θ-HMA sheets closeup, $\theta = 90°$

Figure 5.23: DriveWheel model (60k Polygons) and medial axis at different resolutions: Artificial noise was added to the model. Rim curves are shown in green, seam curves are shown in magenta. (a) The Model, with the front faces shown in wireframe (b) Blum medial axis, black box highlights the zoomed in region (c) A closeup highlighting the tiny sheets corresponding to the unstable parts. (d) Sheets of the θ-HMA, $\theta = 150°$. Connectivity of the model and all holes are preserved. (e) Sheets of the θ-HMA, $\theta = 90°$, black box highlights the zoomed in region (f) A closeup of the θ-HMA, $\theta = 90°$, showing the stable subset of the medial axis.

5.9.1 Approximate Voronoi Diagram Computation

Time Complexity: The total running time of the subdivision algorithm is depends on the depth of the subdivision performed and the relative configuration of the Voronoi faces. In this section, we provide time bounds on the computation cost per cell, specifically the cost of computing the homotopy criterion. Let the size of governor set of a cell be k. Then the number of intersection points is bounded by $O(k^2)$. Each intersection point is checked against remaining $O(k)$ governors to determine if it is a valid edge event. Given the set of edge events, they are sorted by their governor labels in $O(k^2 \log k)$ time. Next the algorithm used to trace the Voronoi edges in a single region boundary performs $O(1)$ computations at each edge event. Thus the total cost of computing the edge events and tracing the all Voronoi region boundaries on a cell is at most $O(k^3)$. Typically, the number of governors per cell is small, but in the worst case it can be $k = O(N)$, N = number of entities on the boundary) for degeneration configurations. The boundary criterion can be computed in $O(k)$ time.

Comparison: We compare our algorithm to prior approaches for computing the Voronoi diagram of polyhedral models.

The seam curve tracing methods [CKM04, SPB96, RT95a] compute the exact Voronoi diagram. In practice, they can compute a topologically correct Voronoi diagram, but they require use of exact arithmetic to solve a system of tri-variate non linear equations. Furthermore, they are prone to degenerate configurations. As a result, these approaches may not scale well to large models.

Our work is most similar to work on computing an approximate Voronoi diagram using spatial subdivision. The work of [VO98, BCMS05] does not provide any topological guarantees on the computed approximation - instead the subdivision is carried out to a predefined level. The work of Etzion and Rappoport [ER02] provides a topologically valid Voronoi graph for cells of size greater than some predefined constant ϵ. In general, it is not easy to select a good value of ϵ for large models. For degenerate and near-degenerate configurations, they compute an approximate Voronoi graph, with no topological guarantees. In case of large cells, their approach computes an approximation that is homeomorphic to the exact Voronoi diagram only for non-degenerate configurations. Moreover, they require that the cells are subdivided till the number of governors of a cell is small (typically $4 - 6$, except for special cases). As a result, their approach can be rather conservative.

In comparison, our algorithm provides a less strict topological guarantee on the output. We ensure homotopy equivalence between the exact Voronoi diagram and our approximation, even in the presence of degenerate and near degenerate configurations. We exploit the fact that in the neighborhood of a near-degenerate configuration, the Voronoi diagram is homotopy equivalent to a point and this property simplifies the overall computation. The homotopy criterion, introduced in Section 4, also checks for this condition in a cell containing a degenerate configuration. Furthermore, the homotopy criterion allows for early termination during subdivision, even if a call has a large number of governors. This results in fewer levels of subdivision. In practice, the size of leaf nodes in the subdivision is of similar scale as the input geometry.

5.9.2 θ-Homotopy Medial Axis Computation

Time Complexity: We provide the complexity of the algorithm as a function of the combinatorial complexity of the Blum MAT. A key step in our simplification algorithm is the operation to check if a sheet f_i is a frontier sheet. Let $|\mathcal{S}(f_i)|$ denote the number of seam curves incident on f_i, given by the number of sheets adjacent to f_i, and $\langle|\mathcal{S}(f)|\rangle$ be the average number of seam curves of a sheet. Then the cost of checking if a sheet f_i is frontier is $O(|\mathcal{S}(f_i)|)$. We first present the cost of Algorithm 8. In the worst case, the frontier sheet check is performed on each sheet f_k adjacent to a sheet f_i, i.e. $|\mathcal{S}(f_i)|$ times. The cost of each frontier check is $O(|\mathcal{S}(f_k)|)$. The cost of adding or deleting a sheet from the priority queue $\mathcal{Q}_j$ is $O(\log|\mathcal{Q}_j|)$. Hence the cost of a single instance Algorithm 8 is $\sum_{k=1}^{|\mathcal{S}(f_i)|}[O(|\mathcal{S}(f_k)|)+O(\log|\mathcal{Q}_j|)]$. Therefore, the cost of a single instance of Algorithm 7 is $O(\log|\mathcal{Q}_j|)+\sum_{k=1}^{|\mathcal{S}(f_i)|}[O(|\mathcal{S}(f_k)|)+O(\log|\mathcal{Q}_j|)] = O(\langle|\mathcal{S}(f)|\rangle^2+\log|\mathcal{Q}_j|)$. A sheet f_i can get added to the frontier set $\mathcal{Q}_j$ at most $|\mathcal{S}(f_i)|$ times. Hence, the number of iterations in Algorithm 6 is at most $\sum_{i=1}^{|\mathcal{F}|}|\mathcal{S}(f_i)| = O(|\mathcal{F}|\langle|\mathcal{S}(f)|\rangle)$. Moreover, the size of the frontier set is bounded by the number of sheets, $|\mathcal{Q}_j| \leq |\mathcal{F}|$. Thus the total cost of the Algorithm 6 is $O(|\mathcal{F}|\langle|\mathcal{S}(f)|\rangle^3 + |\mathcal{F}|\log|\mathcal{F}|\langle|\mathcal{S}(f)|\rangle)$. Typically, $\langle|\mathcal{S}(f)|\rangle$is a constant, the size of the frontier set is much smaller than $|\mathcal{F}|$, and the simplification cost is usually linear (or better) in $|\mathcal{F}|$.

Comparison: We compare some features of our MAT simplification algorithm with prior techniques. There are many known approaches for computing and simplifying the medial axis. It is hard to make direct comparisons between all these algorithms, as different algorithms make varying assumptions about the input and generate different kind of approximations.

The main feature of our approach is that we preserves the homotopy type of the medial axis while allowing for significant simplification of the medial axis. Our algorithm has been applied to polyhedral models as input, and faithfully captures the medial axis near sharp edges and corners in the input. Further, the algorithm preserves cavities corresponding to internal voids in the medial axis.

Some of the earlier analytic algorithms for MAT computation are based on tracing the seam curves [CKM99, RT95b, SPB96]. These algorithms are relatively expensive and the worst case complexity is $O(n^3)$, where n is the number of features in the input solid. In practice, they have been applied to polyhedral models with few thousand triangles and compute the Blum medial axis and not a simplification of the medial axis. The adaptive subdivision algorithms [VO95, ER02] compute the generalized Voronoi Diagram, rather than a simplified medial axis. Further, these approaches may not be able to handle polyhedral models with internal voids.

The surface sampling approaches, such as [ACK01b, DZ02a], take a point sampling on the surface as input and approximate the medial axis using the Voronoi diagram. Robust and efficient methods for computing the Voronoi diagram for point samples are well known. It is hard to make a direct comparison, as the output generated by these algorithms is different than our approaches which compute a distance field on a spatial grid. Many times the algorithms based on a point samples of the boundary may not be able to generate a good quality of approximation of the medial axis near the sharp features of the polyhedral model. The convergence of the Voronoi diagram to the medial axis with a finite discrete sampling has been proven, and extended algorithms have been proposed to generate good quality approximations for CAD models [DZ02a]. However, these methods guarantee a convergence to the medial axis in the limit, and may not provide topological guarantees on the computed medial axis approximation. Tam and Heidrich [TH03] describe an iterative algorithm to simplify the medial axis of polyhedral models while avoiding some topological artifacts during the construction. Their work builds upon point sampling approaches, and has been applied to scanned models without many sharp features. There are no guarantees on the homotopy equivalence of the medial axis. Furthermore, the pruning algorithm needs to perform expensive global operations for topology preservation.

The λ-medial axis [CL04] provides a simplification of the medial axis for any open bounded shape in $\mathbb{R}^n$ with homotopy equivalence to the original medial axis. The constraints on λ depend on the critical points in the gradient field of the distance function. An ϵ-sampling of the boundary of the shape is required, the choice of ϵ depends on different heuristics. Also, a single value of λ may not be appropriate to provide significant simplification for the entire shape. We are not aware of a practical implementation of this method. Attali et al. [ABE04] acknowledge these open issues and suggest a nested sequence of λ-Voronoi graphs with different values of λ for portions of the shape. In fact, a λ-medial axis with a small value of λ can be used as the original medial axis for our simplification algorithm, which subsequently allows significant simplification while preserving homotopy equivalence.

5.9.3 Limitations

Our algorithm has a few limitations. The approximate Voronoi diagram computed by our algorithm is not homeomorphic to the exact Voronoi diagram. Since it is based on spatial subdivision, the cost of computation and the complexity of the approximate Voronoi diagram varies based on the configuration the subdivision grid. In particular, one may encounter degenerate configurations in which the intersection of the Voronoi regions with the boundary of the cell may be a single point (i.e. a tangential intersection), and such cases cannot be easily resolved with only subdivisions. We believe a subdivision scheme which allows for perturbation of the cell faces may be able to alleviate this problem.

Our simplification algorithm depends on a spatial subdivision scheme to compute the Voronoi graph of the polyhedron, and relies on the accuracy of the computed approximate Voronoi diagram. The measure of stability that depends on separation angles, provides scale invariance but may retain noisy features if they exhibit high separation angles. The simplification algorithm uses a greedy approach for pruning the unstable parts of the medial axis and a global minimum of the stability measure is not guaranteed. The elementary primitive in our pruning algorithm is a sheet, and the amount of simplification is influenced by the size of sheets. Finally, the simplified medial axis is not unique for a fixed value of θ, but depends on the pruning order. Actually, determining a unique order for iterative pruning for 3D models using topological constraints alone is still an open problem [PSS$^+$03].

Chapter 6

Conclusions

The Voronoi diagram is one of the most fundamental data structures, and along with the medial axis axis provides a well defined shape representation. However, the use of Voronoi diagrams and medial axes to applications involving 3D polygonal models has been limited. This is due to difficulty in design and implementation of reliable and efficient algorithms for computation and application of the Voronoi diagram and medial axis of 3D polygonal models.

In this thesis, we present present efficient algorithms for computing discrete Voronoi diagram and simplified medial axis of complex 3D polyhedral models, with geometric and topological guarantees, and demonstrated the application to proximity queries among multiple deformable models. We describe algorithms to compute 3D distance fields of complex geometric models at interactive using culling and clamping techniques and an efficient mapping to graphics hardware. We provide geometric guarantees on the result using Hausdorff distance bounds. We also present an adaptive sampling algorithm to provide topological guarantees on the approximate Voronoi diagram and computing the homotopy-preserving simplified medial axis from the approximate Voronoi diagram. Finally, we present a unified approach for performing different proximity queries among multiple deformable models using second order discrete Voronoi diagrams.

In spite of the advances presented in this dissertation, there are still many open problems in application of Voronoi diagrams of 3D polygonal models. The techniques presented have certain limitations on the output, and have some performance limitations. In this chapter, I summarize the main results of my dissertation. I also discuss possible future research directions.

6.1 Summary of Results

In this thesis we have presented efficient algorithms for computing discrete Voronoi diagram and approximate medial axis of complex 3D polyhedral models. We described an algorithm to compute 3D distance fields of geometric models by using a linear factorization of Euclidean distance vectors. This formulation maps directly to the linearly interpolating graphics rasterization hardware and enables us to compute distance fields of complex 3D models at interactive rates. We also used clamping and culling algorithms based on properties of Voronoi diagrams to accelerate this computation. We used occlusion queries to speed up the computation and have presented a conservative scheme to overcome sampling errors. We provided geometric guarantees on the result using Hausdorff distance bounds.

We presented a unified and general approach to perform collision and distance queries in complex environments composed of multiple deforming objects. We showed the reduction of different proximity queries to specializations of N-body distance queries. We used properties of Voronoi diagrams to perform N-body culling and conservatively compute the Voronoi neighbors using second order discrete Voronoi diagrams and distance bounds. We have used our algorithms to perform different proximity queries in complex deformable models composed of tens of thousands of triangles. The performance of our collision detection algorithms is comparable to prior approach, except our algorithm can also handle models with changing topologies. Moreover, we observed one order of magnitude improvement over prior distance and penetration depth computation algorithms.

We also presented an adaptive sampling algorithm to provide topological guarantees on the approximate Voronoi diagram of a 3D polyhedron. Homotopy equivalence is a weaker topological guarantee compared to homeomorphism, however it captures all the topological features of the shape. Our algorithm uses subdivision criteria to compute an approximate Voronoi diagram which is homotopy equivalent to the exact Euclidean Voronoi diagram. The subdivision criteria is based on computing the arrangement of 2D conic sections, which can be performed accurately and efficiently [Be05, KCMh99]. Hence our algorithm is simpler than exact 3D Voronoi diagram computation and can handle near-degenerate configurations of the Voronoi diagram.

Finally, we have presented a simplified medial axis approximation, the θ-HMA, that computes a stable subset of Blum's medial axis, and preserves the homotopy type of Blum's medial axis. The stability of the medial axis is guided by a separation angle criterion, which has been well studied. For polyhedral models, we presented a formal characterization of the relationship between the stability of medial axis junctions and seams to the stability of incident sheets, based on separation angles. Our algorithm computes a bounded measure of stability of a medial axis sheet using discrete sampling. The construction of the θ-HMA is based on an iterative pruning algorithm which uses efficient local tests. We have highlighted the performance of our algorithm on many complex benchmarks and also used it to compute a homotopy preserving medial axis approximation.

6.2 Summary of Limitations

In previous chapters we have already discussed several limitations of the techniques presented in this thesis (Sections 2.13, 3.6, 4.6, 5.9). Here we summarize the key limitations.

The distance field computation is performed on a uniform grid on the GPU and its accuracy is governed by grid resolution. Current graphics processors provide up to $4K \times 4K$

pixel resolution and this imposes an upper bound on the grid resolution. The accuracy of the algorithm is governed by that of the graphics hardware. For example, the current hardware provides support for 32-bit floating point representation and it is not fully compatible with the IEEE floating-point standard. Secondly, our algorithm involves a read-back from the GPU back to the CPU, which can have additional overhead for high resolution distance fields. Furthermore, the computed discrete Voronoi diagram does not provide any topological guarantees on the output. Our algorithms are best suited for global distance field computations in complex environments. The culling techniques involve an occlusion query, which incur an overhead on current graphics hardware. For narrow bands, and highly tessellated models, polygon transformation can become a bottleneck. Finally, our current work is limited to Euclidean distance functions.

Surface distance map computation is limited to deforming meshes with fixed connectivity. If the underlying simulation consists of objects with changing topologies, we may need to update the planar parameterization and recompute the spatial hierarchies.

The proximity query computation incurs the overhead or computing the discrete Voronoi diagrams. Even for small environments, the read-back overhead can be high. Our PNS computation can be conservative if the resolution of the discrete 3D grid is low. This can result in a high number of exact tests between the triangle primitives. Finally, our penetration depth algorithm only computes a local penetration depth. Our approach only works well if there is an isolated contact between the two objects.

The approximate Voronoi diagram computed by our algorithm is not homeomorphic to the exact Voronoi diagram. Since it is based on spatial subdivision, the cost of computation and the complexity of the approximate Voronoi diagram varies based on the configuration the subdivision grid. In particular, one may encounter degenerate configurations in which the intersection of the Voronoi regions with the boundary of the cell may be a single point (i.e. a tangential intersection).

The medial axis simplification algorithm relies on separation angles, provides scale invariance but may retain noisy features if they exhibit high separation angles. The simplification algorithm uses a greedy approach for pruning the unstable parts of the medial axis and a unique result is not guaranteed. The elementary primitive in our pruning algorithm is a sheet, and the amount of simplification is influenced by the size of sheets.

6.3 Future Work

There are many avenues for future work. One possibility to improve the performance of discrete Voronoi diagram computation is utilizing temporal coherence between successive frames for dynamic or deformable models. We are also exploring hierarchical techniques to perform distance field computation on adaptive grids [SAC+07]. It would be useful to extend distance field computation to other distance metrics, (e.g. L_k norm), and higher order primitives including splines or algebraic surfaces. In this regard, the work on efficient rasterization of algebraic curves and surfaces on the GPU holds promise [LB06]. Finally, it would be interesting to explore the mapping of these algorithms to newer GPUs and multi-core architectures.

It may be possible to extend our algorithm for surface distance map computation to objects with changing topologies, where we incrementally recompute the affine transformation to the parametric domain. Surface distance maps could also be useful to accelerate ray tracing dynamic scenes [SKALP05].

We would like to use our proximity computation algorithms for other applications such as surgical or finite-element simulation, where the mesh connectivity or topologies of the objects may change. It may be useful to extend our penetration depth computation algorithm to robustly handle deep penetrations and multiple contacts. We are also exploring the use of higher-order Voronoi diagrams for motion planning of multiple agents [SAC+07].

For approximate homotopy preserving Voronoi diagram computation, we would also like to combine our algorithm to other subdivision schemes such as kd-trees, which offer a better choice of partitioning planes. As previously mentioned, the approximate homotopy preserving Voronoi diagram can have a complicated structure for large models. We would like to study various methods for simplifying this structure and within the context of specific applications like motion planning, feature identification and shape analysis. This may involve use of other medial axis simplification criteria in conjunction to separation angles. A challenging task is to compute a simplified medial axis approximation with better guarantees on the global minimum of the stability measures, possibly leading to a unique pruning order. We are interested in applying the simplification algorithm to other medial axis approximations. Finally, we would like to explore applications of the θ-HMA such as mesh generation and shape analysis.

BIBLIOGRAPHY

[AB02] D. Attali and J. Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. In *Proc. of ACM Solid Modeling*, pages 139–146, 2002.

[ABE04] Dominique Attali, J.-D. Boissonat, and Herbert Edelsbrunner. Stability and computation of the medial axis. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Springer-Verlag, 2004.

[ACK01a] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19:127–153, 2001.

[ACK01b] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 249–260, 2001.

[AK96] F. Aurenhammer and Rolf Klein. Voronoi diagrams. Technical Report 198, Department of Computer Science, FernUniversität Hagen, Germany, 1996.

[AK00] F. Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[AM97] D. Attali and A. Montanvert. Computing and simplifying 2d and 3d continuous skeletons. *Computer Vision and Image Understanding*, 67(3):261–273, 1997.

[Aur91] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[BBGS99] R. Blanding, C. Brooking, M. Ganter, and D. Storti. A skeletal-based solid editor. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 141–150, 1999.

[BCMS05] Imma Boada, Narcis Coll, Narcis Madern, and J. Antoni Sellares. Approximations of 3D generalized voronoi diagrams. In *Proc. 21st European Workshop on Computational Geometry*, pages 163–166, 2005.

[Be05] E. Berberich and et al. Exacus: Efficient and exact algorithms for curves and surfaces. *Proc. of the 13th European Symposium on Algorithms*, 2005.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust treament for collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH*, pages 594–603, 2002.

[BGKW95] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidean distance transform and Voronoi diagram algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:529–533, 1995.

[Bli78] J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 286–292, 1978.

[Blu67] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[BMW00] D. Breen, S. Mauch, and R. Whitaker. 3d scan conversion of csg models into distance, closest-point and color volumes. *Proc. of Volume Graphics*, pages 135–158, 2000.

[BN78] Harry Blum and Roger Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10:167–180, 1978.

[BW01] D. Baraff and A. Witkin. *Physically Based Modeling*. ACM SIGGRAPH Course Notes, 2001.

[BWK03] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *Proc. of ACM SIGGRAPH*, pages 862–870, 2003.

[Cat74] E. Catmull. *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah, 1974.

[CD85] L. P. Chew and R. L. Drysdale, III. Voronoi diagrams based on convex distance functions. In *ACM Symposium on Computational Geometry*, pages 235–244, 1985.

[CKM98] T. Culver, J. Keyser, and D. Manocha. Accurate computation of the medial axis of a polyhedron. Technical Report TR98-034, Department of Computer Science, University of North Carolina, 1998. Appeared in Proceedings of ACM Solid Modeling 99.

[CKM99] Tim Culver, John Keyser, and Dinesh Manocha. Accurate computation of the medial axis of a polyhedron. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 179–190, 1999.

[CKM04] T. Culver, J. Keyser, and D. Manocha. Exact computation of a medial axis of a polyhedron. *Computer Aided Geometric Design*, 21(1):65–98, 2004.

[CL04] Frédéric Chazal and André Lieutier. Stability and homotopy of a subset of the medial axis. In *Proc. ACM Symposium on Solid Modeling and Applications*, 2004.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

[Coo84] Robert L. Cook. Shade trees. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.

[CS02] S. W. Choi and H.-P. Seidel. Linear onesided stability of MAT for weakly injective 3D domain. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 344–355, 2002.

[CS04] Frédéric Chazal and Rémi Soufflet. Stability and finiteness properties of medial axis and skeleton. *Journal of Control and Dynamical Systems*, 10(2):149–170, 2004.

[Cui99] O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Universite Catholique de Louvain, 1999.

[Cul00] T. Culver. *Accurate Computation of the Medial Axis of a Polyhedron*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2000.

[CZ06] Huai-Dong Cao and Xi-Ping Zhu. A complete proof of the poincaré and geometrization conjectures of the Hamilton-Perelman theory of the Ricci flow. *Asian Journal of Mathematics*, 10(2):165 – 498, 06 2006.

[Dan80] P. E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[DDS03] Pavel Dimitrov, James N. Damon, and Kaleem Siddiqi. Flux invariants for shape. In *International Conference on Computer Vision and Pattern Recognition*, Madison, Wisconsin, 2003.

[Den03a] M. Denny. Solving geometric optimization problems using graphics hardware. *Computer Graphics Forum*, 22(3), 2003.

[Den03b] Markus Denny. Solving geometric optimization problems using graphics hardware. In *Proc. of Eurographics*, 2003.

[DHKS93] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[DMB$^+$96] R. J. Donaghy, W. McCune, S. J. Bridgett, C. G. Armstrong, D. J. Robinson, and R. M. McKeag. Dimensional reduction of analysis models. In *Proc. 5th International Meshing Roundtable*, pages 307–320, PO Box 5800, MS 0441, Albuquerque, NM, 87185-0441, 1996. Sandia National Laboratories. Also Sand. Report 96-2301.

[DQ04] H. Du and H. Qin. Medial axis extraction and shape manipulation of solid objects using parabolic PDEs. In *Proc. ACM Symposium on Solid Modeling and Applications*, 2004.

[DZ02a] Tamal K. Dey and Wulue Zhao. Approximate medial axis as a Voronoi subcomplex. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 356–366, 2002.

[DZ02b] Tamal K. Dey and Wulue Zhao. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. In *European Symposium on Algorithms*, pages 387–398, 2002.

[Egg98] H. Eggers. Two fast euclidean distance transformations in z^2 based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, 1998.

[EL01] S. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3):500–510, 2001.

[ER02] Michal Etzion and Ari Rappoport. Computing Voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry: Theory and Applications*, 21(3):87–120, March 2002.

[Eri04] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2004.

[ES86] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986.

[FG05] I. Fischer and C. Gotsman. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University, 2005.

[FG06] L. Fisher and C. Gotsman. Fast approximation of high order voronoi diagrams and distance transforms on the gpu. *Journal of Graphics Tools*, 2006.

[FGLM01] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[FH05] M. Floater and K. Hormann. Surface parameterization:: A tutorial and survey. Technical report, 2005.

[FL01] S. Fisher and M. C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. *Proc. of EG Workshop on Computer Animation and Simulation*, pages 99–111, 2001.

[FLM03] M. Foskey, M. Lin, and D. Manocha. Efficient computation of a simplified medial axis. *Proc. of ACM Solid Modeling*, pages 96–107, 2003.

[For87] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[Fou92] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[FPRJ00] S. Frisken, R. Perry, A. Rockwood, and R. Jones. Adaptively sampled distance fields: A general representation of shapes for computer graphics. In *Proc. of ACM SIGGRAPH*, pages 249–254, 2000.

[GBK05] Michael Guthe, Aakos Balazs, and Reinhard Klein. Gpu-based trimming and tessellation of nurbs and t-spline surfaces. *ACM Trans. Graph.*, 24(3):1016–1023, 2005.

[GF03] J. Gomes and O. Faugeras. The vector distance functions. *Int. Journal of Computer Vision*, 52(2):161–187, 2003.

[GK00] Peter Giblin and Benjamin Kimia. A formal classification of 3D medial axis points and their local geometry. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 566–573, 2000.

[GKJ$^+$05] N. Govindaraju, D. Knott, N. Jain, I. Kabal, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha. Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 24(3):991–999, 2005.

[GL-02] Nvidia occlusion query. http://oss.sgi.com/projects/ogl-sample/registry/NV/occlusion_query.txt, 2002.

[GL02] M. Garber and M. Lin. Constraint-based motion planning using voronoi diagrams. *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

[GLW$^+$04] N. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast computation of database operations using graphics processors. *Proc. of ACM SIGMOD*, 2004.

[GRLM03] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 25–32, 2003.

[HBSL03] M. Harris, B. Baxter, G. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 92–101, 2003.

[HCK$^+$99a] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[HCK$^+$99b] Kenneth E. Hoff, III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Computer Graphics Annual Conference Series (SIGGRAPH '99)*, pages 277–286, 1999.

[HCK$^+$00] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *IEEE Conference on Robotics and Automation*, pages pp. 2931–2937, 2000.

[HLC$^+$01] J. Huang, Y. Li, R. Crawfis, S.C. Lu, and S.Y. Liou. A complete distance field representation. In *Proceedings of IEEE Visualization*, pages 247–254, 2001.

[HREK05] Iddo Hanniel, M. Ramanathan, Gershon Elber, and Myung Soo Kim. Voronoi region extract of free-form rational planar closed curves. In *Symposium on Solid and Physical Modeling*, 2005.

[HTK$^+$04] B. Heidelberger, M. Teschner, R. Keisner, M. Mueller, and M. Gross. Consistent penetration depth estimation for deformable collision response. *Proc. of Vision, Modeling and Visualization*, pages 315–322, 2004.

[HZLM01] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 145–148, 2001.

[HZLM02] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Technical Report TR02-004, Department of Computer Science, University of North Carolina, 2002.

[JC04] D. E. Johnson and E. Cohen. Unified distance queries in a heterogeneous model environment. In *ASME DETC*, 2004.

[KCMh99] Joh"n Keyser, Tim Culver, Dinesh Manocha, and Shankar Kris hnan. MAPC: A library for efficient and exact manipulation of alge braic points and curves. In *Proc. 15th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1999.

[KOLM02] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *Proc. of ACM/Eurographics Symposium on Computer Animation*, pages 23–31, 2002.

[KP03] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, pages 73–80, 2003.

[Kru91] B. Kruse. An exact sequential Euclidean distance algorithm with application to skeletonizing. In *7th Scandinavian Conference on Image Analysis (SCIA '91)*, pages 517–524, 1991.

[KS00] K. Kawachi and H. Suzuki. Distance computation between non-convex polyhedra at short range based on discrete Voronoi diagrams. *IEEE Geometric Modeling and Processing*, pages 123–128, 2000.

[KSKB95] R. Kimmel, D. Shaked, N. Kiryati, and A. M. Bruckstein. Skeletonization via distance maps and level sets. *Computer Vision and Image Understanding*, 62(3):382–391, 1995.

[KY04] T. Kanai and Y. Yasui. Per-pixel evaluation of parametric surfaces on gpu. *ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.

[LAM01] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics*, pages 325–333, 2001.

[LB05] Charles Loop and Jim Blinn. Resolution independent curve rendering using programmable graphics hardware. *ACM Trans. Graph.*, 24(3):1000–1009, 2005.

[LB06] Charles Loop and Jim Blinn. Real-time gpu rendering of piecewise algebraic surfaces. *ACM Trans. Graph. (Proc ACM SIGGRAPH)*, 25(3):664–670, 2006.

[LC91a] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[LC91b] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[LGLM00] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, pages 3719–3726, 2000.

[Lie03] André Lieutier. Any open bounded subset of R^n has the same homotopy type than its medial axis. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 65–75, 2003.

[Lin93] M.C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[LK01] Frederic F. Leymarie and Benjamin B. Kimia. The shock scaffold for representing 3D shape. In *Visual Form 2001*, pages 216–229. Springer-Verlag, 2001. Lecture Notes in Computer Science, no. LNCS 2059.

[LKHW03] A. Lefohn, J. Kniss, C.D. Hanses, and R. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of IEEE Visualization*, page To Appear, 2003.

[LLC92] L. Lam, S.-W. Lee, and C. Y. Chen. Thinning methodologies—A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[LM04] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, chapter 35, pages 787–807. CRC Press LLC, Boca Raton, FL, 2004.

[Mau03] Sean Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Californa Institute of Technology, 4 2003.

[Mey79] F. Meyer. *Cytologie quantitative et morphologie Mathématique*. PhD thesis, École des Mines, 1979.

[MFV98] Grégoire Malandain and Sara Fernández-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16:317–327, 1998.

[MHTG05] M. Mueller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformation based on shape matching. *Proc. of ACM SIGGRAPH*, pages 471–478, 2005.

[Mil93] V. Milenkovic. Robust construction of the Voronoi diagram of a polyhedron. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 473–478, 1993.

[Mir98] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[MQR03] C.R. Maurer, R. Qi, and V. Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, February 2003.

[Mul92] James C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535, November 1992.

[OBS92] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.

[OLG$^+$05] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Tim Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, September 2005.

[PDC$^+$03] T. Purcell, C. Donner, M. Cammarano, H. Jensen, and P. Hanrahan. Photon mapping on programmable graphics hardware. *ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*, pages 41–50, 2003.

[PF01] R. Perry and S. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. of ACM SIGGRAPH*, pages 47–56, 2001.

[PG90] N. M. Patrikalakis and H. N. Gürsoy. Shape interrogation by medial axis transform. In *Proc. 16th ASME Design Automation Conference*, September 1990.

[PS05] R. Peikert and C. Sigg. Optimized bounding polyhedra for gpu-based distance transform. In *Scientific Visualization: The visual extraction of knowledge from data*, 2005.

[PSS$^+$03] Stephen M. Pizer, Kaleem Siddiqi, Gabor Szekely, James M. Damon, and Stephen W. Zucker. Multiscale medial loci and their properties. *International Journal of Computer Vision*, 55:155–179, 2003.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[Rag92] I. Ragnelmam. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics and Image Processing*, 56(3):399–409, 1992.

[RKLM04] S. Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004.

[RL06] S. Redon and M. Lin. A fast method for local penetration depth computation. *Journal of Graphics Tools*, page To Appear, 2006.

[RT95a] J. Reddy and G. Turkiyyah. Computation of 3D skeletons using a generalized Delaunay triangulation technique. *Computer-Aided Design*, 27:677–694, 1995.

[RT95b] Jayachandra Reddy and George Turkiyyah. Computation of 3D skeletons using a generalized Delaunay triangulation technique. *Comput. Aided Design*, 27(9):677–694, 1995.

[SA95] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[SAC$^+$07] Avneesh Sud, Erik Andersen, Sean Curtis, Ming Lin, and Dinesh Manocha. Real-time path planning for virtual agents in dynamic environments. In *Proc. of IEEE Virtual Reality*, page to appear, 2007.

[SBS97] K. Siddiqi, Kimia B.B., and Chi-Wang Shu. Geometric shock-capturing eno schemes for subpixel interpolation, computation and curve evolution. *Graphical Models and Image Processing*, 59(5):278–301, 1997.

[SBTZ02] Kaleem Siddiqi, Sylvain Bouix, Allen Tannenbaum, and Steven W. Zucker. Hamilton-Jacobi skeletons. *International Journal of Computer Vision*, 48:215–231, 2002.

[SERB98] A. Sheffer, M. Etzion, A. Rappoport, and M. Bercovier. Hexahedral mesh generation using the embedded voronoi graph. *7th International Meshing Roundtable*, pages 347–364, 1998.

[Set99] J. A. Sethian. *Level set methods and fast marching methods*. Cambridge, 1999.

[SFM05] Avneesh Sud, Mark Foskey, and Dinesh Manocha. Homotopy preserving medial axis simplification. In *Proc. ACM Symposium on Solid and Physical Modeling*, 2005.

[SFYC96] R. Shekhar, E. Fayyad, R. Yagel, and F. Cornhill. Octree-based decimation of marching cubes surfaces. *Proc. of IEEE Visualization*, pages 335–342, 1996.

[SGGM06] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006.

[SH75] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 151–162, 1975.

[She95] E. C. Sherbrooke. *3-D Shape Interrogation by Medial Axis Transform*. Ph.D. thesis, Dept. Ocean Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1995.

[SJP05] Le-Jeng Shiue, Ian Jones, and J&#246;rg Peters. A realtime gpu subdivision kernel. *ACM Trans. Graph.*, 24(3):1010–1015, 2005.

[SKALP05] L. Szirmay-Kalos, B. Aszodi, I. Lazanyi, and M. Premecz. Approximate ray-tracing on the gpu with distance impostor. *Proc. of Eurographics*, 2005.

[SL00] K. Sundaraj and C. Laugier. Fast contact localization of moving deformable polyhedra. In *Proc. of IEEE Int. Conference on Control, Automation, Robotics and Vision*, 2000.

[SOM04] A. Sud, M. A. Otaduy, and D. Manocha. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics)*, 23(3):557–566, 2004.

[Spa89] Edwin H. Spanier. *Algebraic Topology*. Springer, 1989.

[SPB96] E. C. Sherbrooke, N. M. Patrikalakis, and E. Brisson. An algorithm for the medial axis transform of 3d polyhedral solids. *IEEE Trans. Visualizat. Comput. Graph.*, 2(1):45–61, March 1996.

[SPG03] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, pages 83–90, 2003.

[SS06] Svetlana Stolpner and Kaleem Siddiqui. Revealing significant medial structure in polyhedral meshes. In *Third International Symposium on 3D Data Processing, Visualization and Transmission*, 2006.

[Sur03] K. Suresh. Automating the CAD/CAE dimensional reduction process. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 76–85, 2003.

[Tau95] G. Taubin. A signal processing approach to fair surface design. In *Proc. of ACM SIGGRAPH*, pages 351–358, 1995.

[TH03] R. Tam and W. Heidrich. Shape simplification based on the medial axis transform. *IEEE Visualization*, 2003.

[THM$^+$03] M. Teschner, B. Heidelberger, M. Muller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. *Proc. of Vision, Modeling and Visualization*, pages 47–54, 2003.

[TKH$^+$05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 19(1):61–81, 2005.

[TT97] M. Teichmann and S. Teller. Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. Technical Report 766, Laboratory of Computer Science, MIT, 1997.

[vdB97] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[VKSM04] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Eurographics Symposium on Geometry Processing*, 2004.

[VO95] Jules Vleugels and Mark Overmars. Approximating generalized Voronoi diagrams in any dimension. Technical Report UU-CS-1995-14, Department of Computer Science, Utrecht University, 1995.

[VO98] J. Vleugels and M. H. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–222, 1998.

[VT00] P. Volino and N. Magnenat Thalmann. Accurate collision response on polygon meshes. In *Proc. of Computer Animation*, page 154, 2000.

[WND97] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Second Edition*. Addison Wesley, 1997.

[Wol92] F. E. Wolter. Cut locus and medial axis in global shape interrogation and representation. Technical Report 92-2, MIT, Dept. Ocean Engg., Design Lab, Cambridge, MA 02139, USA, January 1992.

[Yam84] H. Yamada. Complete euclidean distance transformation by parallel operation. In *Proc. of 7th International Conf. on Pattern Recognition*, pages 336–338, Montreal, Canada, 1984.

[YBM04] Yuandong Yang, Oliver Brock, and Robert N. Moll. Efficient and robust computation of an approximated medial axis. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 15–24, 06 2004.

[ZW93] Y. Y. Zhang and P. S. P. Wang. Analytical comparison of thinning algorithms. *Int. J. Pattern Recognit. Artif. Intell.*, 7:1227–1246, 1993.