Fast Proximity Computation among Deformable Models Using Discrete Voronoi Diagrams

Avneesh Sud Naga Govindaraju Russell Gayle Ilknur Kabul Dinesh Manocha

Dept of Computer Science, University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/DVD

Figure 1: **Multiple deformable models simulation:** *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.*

Abstract

We present novel algorithms to perform collision and distance queries among multiple deformable models in dynamic environments. These include inter-object queries between different objects as well as intra-object queries. We describe a unified approach to compute these queries based on N-body distance computation and use properties of the 2^{nd} *order discrete Voronoi diagram to perform N-body culling. Our algorithms involve no preprocessing and also work well on models with changing topologies. We can perform all proximity queries among complex deformable models consisting of thousands of triangles in a fraction of a second on a high-end PC. Moreover, our Voronoi-based culling algorithm can improve the performance of separation distance and penetration queries by an order of magnitude.*

CR Categories: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, virtual reality

Keywords: Deformable collisions, self-collisions, penetration computation, distance fields, N-body queries

1 Introduction

We address the problem of geometric proximity query computation among multiple deformable models for interactive applications. 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).

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. For example, penetration depth (PD) computation is often used to compute contact forces in penalty-based methods [Heidelberger et al. 2004; Kim et al. 2002]. Separation distances are useful in computing the repulsive forces or estimating the time of contact between moving objects in a discretized simulation [Baraff and Witkin 2001; Kim et al. 2002]. Robust simulations of cloth dynamics may require penetration depth computation [Baraff et al. 2003] or continuous collision detection [Bridson et al. 2002].

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 [Teschner et al. 2005; Mueller et al. 2005]. 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.

1.1 Main Results

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 *2^{nd}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 compute discrete Voronoi diagrams on a uniform grid using graphics hardware. We exploit properties of the 2^{nd}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.

We have implemented our algorithms on a desktop PC with a high-end CPU and GPU. We demonstrate the effectiveness of our algorithms for different proximity queries in several scenarios: N-body deformable simulation with tens or hundreds of deforming objects and cloth simulation with many thousands of triangles. The performance of our algorithms varies between $100-800$ msec, depending on the complexity of the scene and the relative configuration of the primitives. As compared to prior methods, our algorithm offers the following advantages:

- Direct applicability to multiple breaking objects or models with changing topologies;
- Improved culling efficiency and significant reduction in the number of false positives;
- Up to an order of magnitude faster runtime performance over prior techniques for separation distance and local penetration depth computation between multiple, dynamically deforming objects;
- Interactive self-proximity query computation on complex deformable models.

1.2 Organization

The rest of the paper is organized as follows. We briefly survey previous work on proximity queries in Section 2. Section 3 describes the N-body distance query that is used to perform different proximity queries. We present our Voronoi-based distance culling algorithm in Section 4 and use this algorithm to perform inter-object and intra-object queries in Section 5. Section 6 describes our implementation and highlights the performance of our algorithms. We analyze our algorithms and compare their performance with prior methods in Section 7.

2 Related Work

The problems of collision detection and distance computation are well studied in the literature. We refer the readers to recent surveys [Ericson 2004; Lin and Manocha 2004; Teschner et al. 2005]. In this section, we briefly discuss some of the prior algorithms for deformable models.

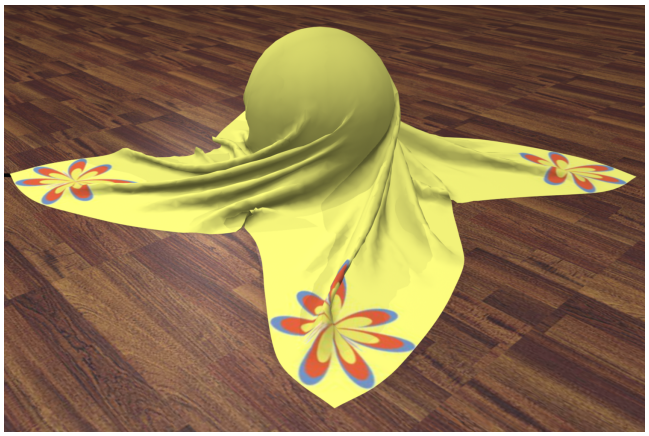


Figure 2: **Cloth simulation:** *The cloth mesh is composed of* $15K$ *triangles and has a high number of triangles in close proximity. As the simulation progresses, the cloth wraps around the sphere and the simulation generates many complex folds. Our algorithm is able to perform continuous self-collision detection among all the triangles within* 800 *msec.*

2.1 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 [Ericson 2004], and 3D sorting based on tight fitting axis-aligned bounding boxes [Cohen et al. 1995]. More recently, GPU-based algorithms [Govindaraju et al. 2003; Govindaraju et al. 2005] 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.

2.2 Bounding volume hierarchies

Bounding volume (BV) hierarchies are widely used for collision detection and separation distance computation. Most proximity computation algorithms for deformable models use hierarchies of spheres or use axis-aligned bounding boxes (AABBs) [Agarwal et al. 2004; van den Bergen 1997; Larsson and Akenine-Möller 2001; James and Pai 2004]. 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.

2.3 Deformable model collision detection

Many specialized algorithms have been proposed to perform collision queries on deformable models. These include GPU-based algorithms [Knott and Pai 2003; Govindaraju et al. 2005] for inter-object or intra-object collisions. Other methods for self-collisions are based on the "curvature test" [Volino and Thalmann 2000] and these can be combined with BV hierarchies. Teschner et al. [2003] use spatial hashing techniques to check for inter-object collisions and self-collisions. All of these algorithms perform only collision queries.

2.4 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 [Quinlan 1994]

or rectangular swept spheres [Larsen et al. 2000] or different model types [Johnson and Cohen 2004]. Techniques have been proposed to update the hierarchies incrementally for deformable models [Sundaraj and Laugier 2000].

Distance Fields: 3D discrete distance fields can be efficiently computed using graphics hardware [Fischer and Gotsman 2005; Sigg et al. 2003; Sud et al. 2004; Sud et al. 2006a]. The discrete distance fields can be used to perform inter-object proximity queries between deformable models at image-space resolution [Hoff et al. 2002; Sud et al. 2006a].

Penetration Depth Computation: Efficient penetration depth computation algorithms have been proposed for rigid polyhedral models [Kim et al. 2002], but they involve considerable preprocessing. Many approximate PD computation algorithms for deformable models are based on GPU-based computations [Hoff et al. 2002; Redon and Lin 2006], precomputed distance fields [Fisher and Lin 2001] or spatial hashing [Heidelberger et al. 2004].

2.5 Voronoi diagrams

The Voronoi diagram is regarded as a powerful proximity data structure in computational geometry [Okabe et al. 1992]. 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 [Lin and Canny 1991; Ehmann and Lin 2001; Mirtich 1998; Kawachi and Suzuki 2000]. 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.

3 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.

3.1 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 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 computation based on N-body distance query.

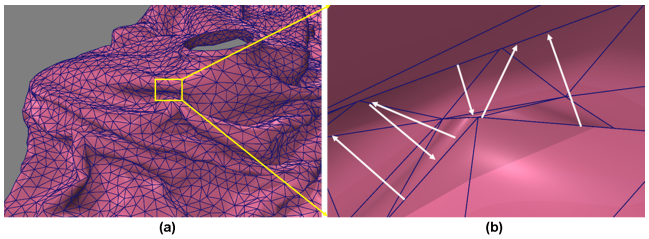


Figure 3: **N-body distance query:** *In this cloth mesh, we compute the closest non-adjacent triangle for every triangle in the mesh. The white arrows highlight the closest triangle to each triangle.*

3.2 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 [Redon et al. 2004]. 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). 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^{nd}order Voronoi diagram of the n primitives.

3.3 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

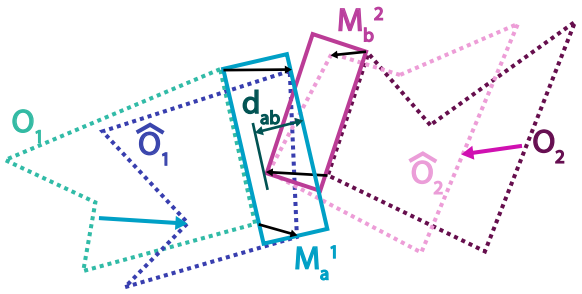


Figure 4: **Continuous Collision Detection for two polygons O_1 and O_2:** *Two polygons, O_1 and O_2, move to positions $\widehat{O}_1$ and $\widehat{O}_2$ at time $t+\Delta t$. The volume swept by a pair of features is bounded by the prisms, M_a^1 and M_b^2, respectively. A conservative CCD check is 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.*

minimum translational distance needed to make the two objects disjoint [Dobkin et al. 1993]:

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

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 [Kim et al. 2002]. 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 3.2 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 5).

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 $\boldsymbol{T}$.

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.

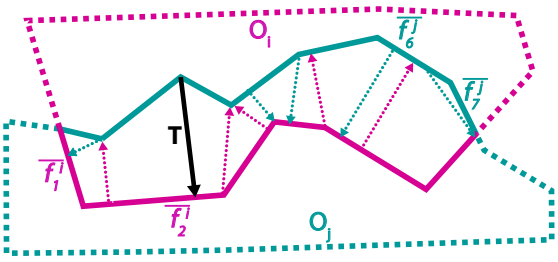


Figure 5: **Local PD Computation for two polygons O_i and O_j :** *Dotted arrows represent direction vectors showing the separation distance between pairs of overlapping features of O_1 and O_2. The maximum separation distance is shown by a thick black arrow and is the local penetration direction* $\mathbf{T}$.

4 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.

4.1 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:

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

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 2nd order Voronoi diagrams, denoted as $\mathrm{VD}^1(\mathcal{P})$ and $\mathrm{VD}^2(\mathcal{P})$. A 1st order Voronoi region $\mathcal{V}^1(p_i|\mathcal{P})$ contains points closest to site p_i, and the 2nd order Voronoi region $\mathcal{V}^2(\{p_i, p_j\}|\mathcal{P})$ contains points closest to two sites p_i and p_j (see figure 6).

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 $\mathcal{G}^2(p_i|\mathcal{P}) = \bigcup_{\mathbf{q} \in p_i} \mathcal{G}^2(\mathbf{q}|\mathcal{P})$.

4.2 PNS Computation Using 2nd Order Voronoi Diagrams

We use the 2nd order 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 $\mathcal{P} \backslash \mathrm{PNS}(p|\mathcal{P})$. To perform the N-body distance query, we compute a tight PNS for each site. The 2nd order Voronoi diagram provides the 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

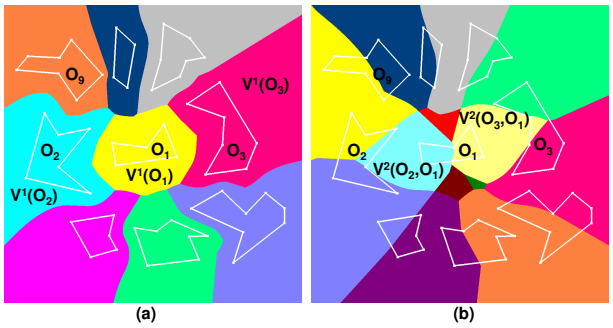


Figure 6: **The 1st and 2nd order Voronoi diagrams of 9 polygons (denoted as O_i) in a plane.** *(a) The 1st order Voronoi diagram: Each color represents the set of points closest to one of the polygon. In this case, O_1 has 8 1st order Voronoi neighbors. The PNS of O_1 is all the objects which share a Voronoi edge, i.e. all other 8 objects. (b) The 2nd order Voronoi diagram: Each color represents a region with same two closest objects. O_1 is contained completely inside two 2nd order Voronoi regions. Therefore, PNS of $O_1 = \{O_2, O_3\}$. We get a tighter PNS with 2nd order Voronoi diagram.*

is ignored in $\mathrm{PNS}(p|\mathcal{P})$. We use the 2nd order Voronoi diagram and the 2nd order governor set to compute a tight PNS. Then we have the following property (illustrated in figure 6):

Lemma 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 $\mathrm{PNS}(p_i|\mathcal{P})$.*

Lemma 1 provides a culling scheme to compute the closest sites for a given set of sites. In addition to a tighter culling scheme, the 2nd order 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 4.4.

N-body distance query: We use the 2nd order Voronoi diagram to perform the N-body query. Given n independent sites $\mathcal{P}$, we compute $\mathrm{VD}^2(\mathcal{P})$ and the 2nd order 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 5.

4.3 Discrete Voronoi Diagram Computation

In the previous subsection, we showed that the PNS for each site can be efficiently computed based on the 2nd order 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 2nd order 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 [Sud et al. 2006a; Sigg et al. 2003]. 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 1st order Voronoi diagram and we read it back to the CPU.

Figure 7: **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.*

In addition to the 1st order Voronoi diagram of triangulated models, we compute the 2nd order 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 [Hoff et al. 2002]. 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 2nd order 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 [Hoff et al. 2002; Sud et al. 2006a]. We first list the sources of under-sampling errors and present our approach to compute a conservative PNS in Section 4.4.

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.

4.4 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 $\mathrm{SD}(p_i)$ from the approximate separation distance $\widetilde{\mathrm{SD}}(p_i)$.

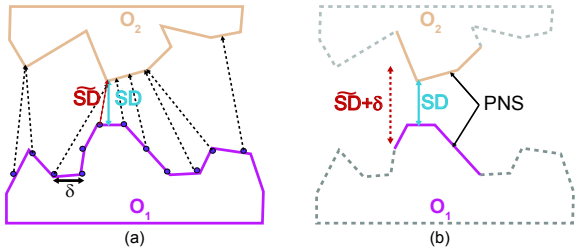


Figure 8: **Conservative PNS using discrete Voronoi diagram:** *Given 2 objects O_1 and O_2. (a) O_1 is sampled at a finite set of points. The closest points on O_2 are shown using dotted vectors. SD is the exact separation distance $\mathrm{SD}(O_1)$, $\widetilde{SD}$ is the approximate separation distance $\widetilde{\mathrm{SD}}(O_1)$. 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.*

Our exact 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 2 (Distance Bound using DVD). *Given the approximate separation distance, $\widetilde{\mathrm{SD}}(p_i)$, the exact separation distance $\mathrm{SD}(p_i)$ is bounded by $\widetilde{\mathrm{SD}}(p_i) - \delta \leq \mathrm{SD}(p_i) \leq \widetilde{\mathrm{SD}}(p_i) + \delta$.*

Lemma 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 [Govindaraju et al. 2005]. When we rasterize the Minkowski sum, we ensure that every point on a site gets sampled. In Section 5, we use these queries to perform accurate inter-object and intra-object queries.

5 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 9. 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.

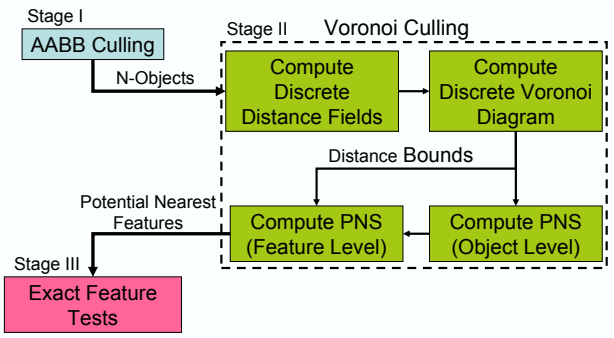


Figure 9: **Overall algorithm:** *Our proximity computation algorithm proceeds in three stages: AABB-based culling, Voronoi culling and exact distance tests on the PNS.*

5.1 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 [Cohen et al. 1995]. 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.

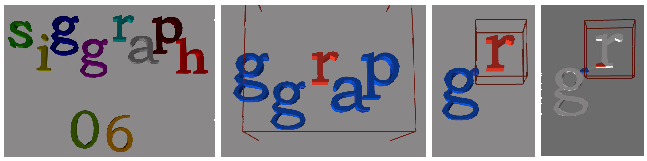
5.2 Stage II: Voronoi-based Culling

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.

5.2.1 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.

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 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



(a) (b) (c) (d)

Figure 10: **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* 2^{nd} *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.*

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 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 8. 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.

5.2.2 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 2^{nd}order 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 $\mathcal{P} \setminus \mathcal{I}(f_k^i)$

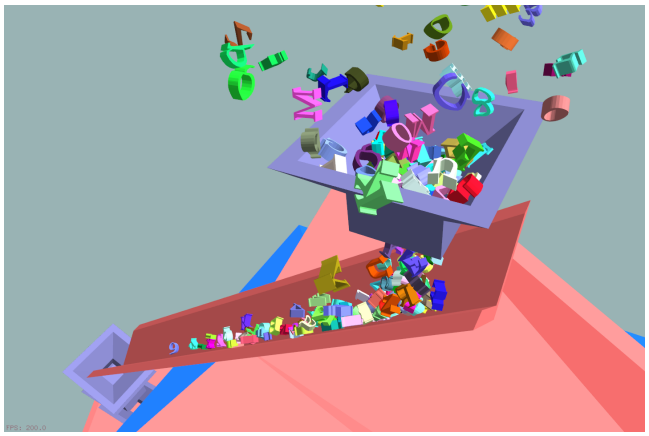


Figure 11: **Multiple deformable object simulation:** *In this simulation, many deforming letters are falling inside a funnel and will eventually slide through a ramp. Each object is composed of nearly* 175 *triangles and there are a total of* 200 *letters in many close-proximity scenarios. Our algorithm is able to perform both inter-object and intra-object queries in this simulation within a second.*

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 $O(n_i)$ 2^{nd}order 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 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 $\mathrm{PNS}(f_k^i|\mathcal{P} \setminus \mathcal{I}(f_k^i))$.

5.3 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 [Bridson et al. 2002]. We use the triangle-triangle distance computation algorithm described in [Larsen et al. 2000] to compute the separation distance between the primitives. We also compute the local penetration depth between the overlapping features.

6 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.

6.1 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

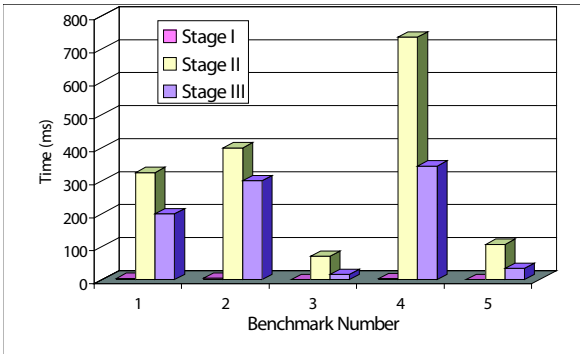


Figure 12: *This graph highlights the average time spent in the three stages of our algorithm for the five benchmarks described in Section 6.2. Due to the high culling efficiency obtained during stage II, we observe that the average time spent in performing exact overlap tests is lower than* 300*ms.*

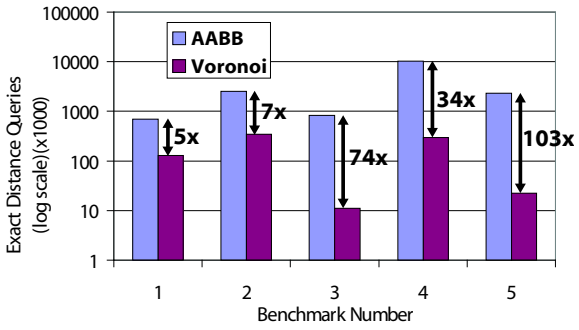


Figure 13: *In this log-scale plot, we show the average number of exact triangle-triangle distance queries performed using an AABB-based algorithm and using Voronoi diagrams. We observe a* $5-100$ *times higher culling efficiency using Voronoi diagrams on the five benchmarks. The high culling efficiency is due to the tight distance bounds obtained using the* 2^{nd} *order Voronoi diagrams.*

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 2^{nd} order DVD computation is done in 2 passes. In the first pass, we scan convert each object, and store the object id in the stencil buffer for all pixels that lie on the object. The feature id is stored in the red channel of the color buffer. In the second pass, we perform distance field computation. For inter-object queries, the reference value and function for stencil test are set to discard the fragment if current object id is equal to the value in the 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. For intra-object queries, we store the list of adjacent feature ids in a 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 2^{nd}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 [Larsen et al. 2000] for computing the separation tests. The average time to perform one separation distance query between two triangles is 1–2 microseconds.

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 [Govindaraju et al. 2005]. 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.

6.2 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 7), (2) a cloth folding on a rotating sphere (figure 2), (3) ten deforming letters falling in a bowl (figure 10), (4) two hundred deforming objects falling through a funnel and sliding over a ramp (figure 11) and, (5) fourteen objects undergoing dynamic topological fractures (figure 1). Our algorithm involves no pre-processing and is able to compute the separation distances, inter-object and intra-object proximity queries.

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 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 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 14, the resolution is chosen such that the total computation time is minimized.

7 Comparison with Prior Approaches

In this section, we compare our algorithms with prior methods. These include 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 [van den Bergen 1997; Larsson and Akenine-Möller 2001] as bounding volumes, because the computation or update cost of hierarchies of OBBs or k-DOPs can be high. In Fig. 15, 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

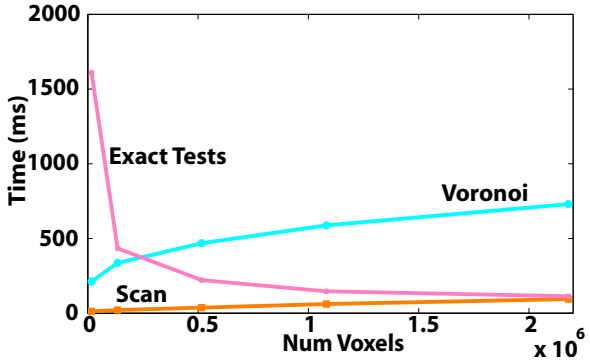


Figure 14: *We show the time taken in computing the discrete Voronoi diagram and the culling efficiency as a function of the Voronoi grid resolution on a deformable simulation with* 200 *objects. The culling efficiency is measured in terms of the number of exact distance tests. The scan operation reads back the Voronoi diagram, and performs a linear scan. We observe the number of exact tests decreases as the grid resolution increases.*

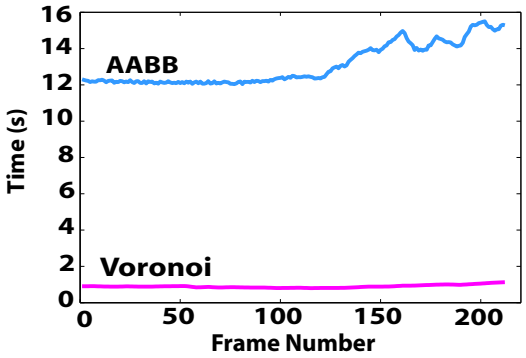


Figure 15: *This graph highlights the performance improvement obtained using our Voronoi-based algorithm over an efficient AABB-based algorithm on the deformable simulation with* 200 *objects. Due to the high culling efficiency obtained using Voronoi diagrams, we are able to achieve nearly one order of magnitude performance improvement over AABBs.*

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. 13). 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.* [2005]. 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 proposed by Govindaraju *et al.* [2005] 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 [Fisher and Lin 2001; Hoff et al. 2002; Sud et al. 2006a], 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 [Teschner et al. 2003; Heidelberger et al. 2004]. 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.

8 Analysis and Limitations

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 2^{nd}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 2^{nd}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.

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 [Baraff et al. 2003; Heidelberger et al. 2004]. Our local PD algorithm may not work well in such situations.

9 Conclusions and Future Work

We present a unified and general approach to perform collision and distance queries in complex environments composed of multiple deforming objects. We use properties of Voronoi diagrams to perform N-body culling and conservatively compute the Voronoi neighbors using 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 observe one order of magnitude improvement over prior distance and penetration depth computation algorithms.

There are many avenues for future work. We would like to reduce the overhead of Voronoi-based culling. Instead of computing the discrete Voronoi diagrams along a 3D grid, we may only compute them along a set of points that lie on the 2D-surface [Sud et al. 2006b]. This could result in faster computation and reduce the overhead of readback and scan conversion. We would like to use our 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 PD computation algorithm to robustly handle deep penetrations and multiple contacts.

Acknowledgments

This research is supported in part by ARO Contracts DAAD19-02-1-0390, and W911NF-04-1-0088, NSF Awards 0400134, 0118743, ONR Contract N00014-01-1-0496, DARPA RDECOM Contract N61339-04-C-0043 and Intel Corporation. We thank Walt Disney Feature Animation for the skirt simulation. We would like to acknowledge Kirstin Williams for audio recording, Prof Ming Lin and other members of UNC GAMMA group for useful discussions and feedback. We are also grateful to the reviewers for their comments.

References

AGARWAL, P., GUIBAS, L., NGUYEN, A., RUSSEL, D., AND ZHANG, L. 2004. Collision detection for deforming necklaces. *Computational Geometry: Theory and Applications 28*, 2-3, 137–163.

BARAFF, D., AND WITKIN, A. 2001. *Physically Based Modeling*. ACM SIGGRAPH Course Notes.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *Proc. of ACM SIGGRAPH*, 862–870.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treament for collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH*, 594–603.

COHEN, J., LIN, M., MANOCHA, D., AND PONAMGI, M. 1995. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, 189–196.

DOBKIN, D., HERSHBERGER, J., KIRKPATRICK, D., AND SURI, S. 1993. Computing the intersection-depth of polyhedra. *Algorithmica 9*, 518–533.

EHMANN, S., AND LIN, M. C. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3, 500–510.

ERICSON, C. 2004. *Real-Time Collision Detection*. Morgan Kaufmann.

FISCHER, I., AND GOTSMAN, C. 2005. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University.

FISHER, S., AND LIN, M. C. 2001. Deformed distance fields for simulation of non-penetrating flexible bodies. *Proc. of EG Workshop on Computer Animation and Simulation*, 99–111.

GOVINDARAJU, N., REDON, S., LIN, M., AND MANOCHA, D. 2003. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 25–32.

GOVINDARAJU, N., KNOTT, D., JAIN, N., KABAL, I., TAMSTORF, R., GAYLE, R., LIN, M., AND MANOCHA, D. 2005. Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 24*, 3, 991–999.

HEIDELBERGER, B., TESCHNER, M., KEISNER, R., MUELLER, M., AND GROSS, M. 2004. Consistent penetration depth estimation for deformable collision response. *Proc. of Vision, Modeling and Visualization*, 315–322.

HOFF, K., ZAFERAKIS, A., LIN, M., AND MANOCHA, D. 2002. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Tech. Rep. TR02-004, Department of Computer Science, University of North Carolina.

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH*, 393–398.

JOHNSON, D. E., AND COHEN, E. 2004. Unified distance queries in a heterogeneous model environment. In *ASME DETC*.

KAWACHI, K., AND SUZUKI, H. 2000. Distance computation between non-convex polyhedra at short range based on discrete Voronoi diagrams. *IEEE Geometric Modeling and Processing*, 123–128.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation for physically-based animation. In *Proc. of ACM/Eurographics Symposium on Computer Animation*, 23–31.

KNOTT, D., AND PAI, D. K. 2003. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, 73–80.

LARSEN, E., GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 2000. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, 3719–3726.

LARSSON, T., AND AKENINE-MÖLLER, T. 2001. Collision detection for continuously deforming bodies. In *Eurographics*, 325–333.

LIN, M., AND CANNY, J. F. 1991. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, 1008–1014.

LIN, M. C., AND MANOCHA, D. 2004. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, ch. 35, 787–807.

MIRTICH, B. 1998. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics 17*, 3 (July), 177–208.

MUELLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformation based on shape matching. *Proc. of ACM SIGGRAPH*, 471–478.

OKABE, A., BOOTS, B., AND SUGIHARA, K. 1992. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK.

QUINLAN, S. 1994. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, 3324–3329.

REDON, S., AND LIN, M. 2006. A fast method for local penetration depth computation. *Journal of Graphics Tools*, To Appear.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2004. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*.

SIGG, C., PEIKERT, R., AND GROSS, M. 2003. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, 83–90.

SUD, A., OTADUY, M. A., AND MANOCHA, D. 2004. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics) 23*, 3, 557–566.

SUD, A., GOVINDARAJU, N., GAYLE, R., AND MANOCHA, D. 2006. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, 117–124.

SUD, A., GOVINDARAJU, N., GAYLE, R., AND MANOCHA, D. 2006. Surface distance maps. Tech. Rep. TR06-011, Dept of Computer Science, University of North Carolina.

SUNDARAJ, K., AND LAUGIER, C. 2000. Fast contact localization of moving deformable polyhedra. In *Proc. of IEEE Int. Conference on Control, Automation, Robotics and Vision*.

TESCHNER, M., HEIDELBERGER, B., MULLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. *Proc. of Vision, Modeling and Visualization*, 47–54.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. Collision detection for deformable objects. *Computer Graphics Forum 19*, 1, 61–81.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4, 1–14.

VOLINO, P., AND THALMANN, N. M. 2000. Accurate collision response on polygon meshes. In *Proc. of Computer Animation*, 154.