Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Fields

Susan Fisher **Ming C. Lin**

Department of Computer Science
University of North Carolina at Chapel Hill
{sfisher,lin}@cs.unc.edu

Abstract

We present a fast penetration depth estimation algorithm between deformable polyhedral objects. We assume the continuum of non-rigid models are discretized using standard techniques, such as finite element or finite difference methods. As the objects deform, the pre-computed distance fields are deformed accordingly to estimate penetration depth, allowing enforcement of non-penetration constraints between two colliding elastic bodies. This approach can automatically handle self-penetration and inter-penetration in a uniform manner. We demonstrate its effectiveness on moderately complex simulation scenes.

1 Introduction

Due to recent advancements in both hardware technology and modeling of the physical world, simulation techniques have been increasingly used to improve the efficiency and effectiveness of robotics in virtual prototyping and design automation. These techniques produce motion sequences directly from input objects (robots and obstacles), simulating their motion based on mathematical models that specify their physical behavior, their complex structures and the interaction among them.

However, most of the existing robotic simulation techniques are restricted to rigid objects, due to the computational challenges in modeling complex deformation among colliding bodies. But, many real-world objects are not rigid. Automatic, predictable and robust simulation of realistic deformation is one of the many challenges for robotics and intelligent systems. Some examples include realistic motion generation of soft tissues and organs for surgical planning of tele-robots, interaction among non-rigid bodies, and the use of elastic tubes for the design of miniature medical robots. [8] gives a recent survey on the simulation of deformable models. One of the most difficult issues in generating realistic motion of non-rigid objects is to simulate contact between between them.

When two flexible objects collide, they exert reaction forces on each other resulting in the deformation of both objects. Similarly when one flexible body self collides, multiple portions of the object may deform. The reaction force is called contact force, and where the two surfaces touch is often called the contact surface. Simulating such events is non-trivial. It is known as the *contact problem* in computational mechanics, and has been actively investigated for decades [5]. The difficulty of this problem for modeling deformation of non-rigid bodies arises from unclear boundary conditions; neither the contact force nor the position of the contact surface is known a priori.

Ideally, no two objects should share the same space. This is the *non-penetration constraint*. The non-penetration constraint can be imposed using techniques such as constrained optimization techniques or penalty-based methods. Due to dual unknowns in the contact problem for deformable models mentioned above, penalty-based methods are often preferred. When using a penalty based method, we need to first define a penetration potential energy that measures the amount of intersection between two models, or the degree of self-intersection of a deformable body. One of the more accurate measurements of the amount of intersection is the penetration depth, commonly defined as the minimum (translational) distance required to separate two intersecting objects. No general and efficient algorithm for computing penetration depth between two non-convex objects is known. In fact, an $O(n^6)$ time bound can be obtained for computing the Minkowski sum of two *rigid*, non-convex polyhedra to find the minimum penetration depth in 3D [7]. No complexity bound for this problem nor a formal definition of penetration depth is yet established for deformable models.

1.1 Main Contribution

We present a novel, efficient approach for estimating the penetration depth between non-penetrating elastic bodies. The underlying geometric models are composed of polygonal meshes. Models consisting of implicit representations or parametric surfaces, such as NURBS, can be tessellated into polygonal meshes with bounded error. We assume that each non-rigid body is modeled using finite element methods (FEM) [21] in our current implementation [11], but the algorithm itself is applicable to other discretization techniques such as finite difference methods or spring-mass systems. We employ the fast marching level-set method [19, 20] to precompute the internal distance field of each *undeformed* model. When two flexible bodies come into contact and deform, the distance fields are likewise deformed to compute the estimated penetration depth between two deforming objects.

Our penetration depth estimation algorithm can handle both self-collisions and contacts between soft objects in a uniform manner. It balances between accuracy and speed, and requires no

assumption or prior knowledge about the locations of contacts. This penetration measure can be incorporated into a penalty-based formulation to enforce the non-penetration constraint between two elastic bodies. This enables efficient computation of the contact force and helps to yield a versatile and robust contact resolution algorithm. We have successfully integrated our penetration depth estimation algorithm to compute collision response of two elastic bodies efficiently.

1.2 Organization

The rest of the paper is organized in the following manner. We briefly survey the state of the art in section 2. In section 3, we give an overview of our algorithm and the basic terminologies used in this paper. Section 4 describes the numerical method used to pre-compute the distance field and how it is updated *on the fly* as the objects deform. Section 5 presents our new penetration depth estimation method for deformable objects based on linear interpolation of precomputed distance fields and the resulting collision response. Section 6 describes the system implementation and demonstrates the effectiveness of our algorithm.

2 Related Work

2.1 Penetration Depth Computation

The notion of penetration depth between overlapping objects was introduced by Buckley and Leifer [2] and Cameron and Culley [4]. Several algorithms [7, 9, 16] have been proposed for computing a measure of penetration depth using various definitions. The enhanced GJK algorithm can also be modified to compute penetration depth between two convex polytopes [3]. Recently, Agarwal, et al. proposed a randomized algorithm that computes penetration depth between two convex polyhedra in $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$ expected time for any constant $\epsilon > 0$ [1]. However, all of them assume that at least one of the input models is a convex polytope.

It is well known that if two polytopes intersect, then the difference of their reference vectors lies in their convolution or Minkowski sum [10]. The problem of penetration depth computation reduces to calculating the minimum distance between the boundary of Minkowski sum of two polyhedra and a point inside it. However, the construction of the Minkowski sum can be quite expensive. In three-dimensional space, the size can be easily quadratic even for two convex polyhedra. An $O(n^6)$ time bound can be obtained for computing the Minkowski sum of two non-convex polyhedra to find the minimum penetration depth [7]. There seems to be little hope to compute the penetration depth at interactive rates based on some of these well-known theoretical algorithms.

Few methods have been proposed to compute the penetration depth for NURBS models or other *non-rigid* model representations. Lin, et al [15] has recently studied the penetration depth problem between quasi-rigid bodies and derived a closed-form formula for the first and second derivatives of this key parameter.

2.2 Distance Field

Computing the minimum geodesic distance from a point to a surface is a well known complex problem [17]. Osher and Sethian [19, 20], introduced a new perspective on this problem by using a partial differential method to perform curve evolution. Hoff, et al. introduced the use of graphics hardware to compute generalized Voronoi diagram and its corresponding discretized distance field [12]. Recently, this approach has been applied to perform general proximity queries in 2D [13].

3 Preliminaries

In this section, we define basic notations and methodologies used in this paper, give a brief overview of the simulation framework used to test our algorithm, and give an outline of our approach for estimating penetration depth between deformable models.

3.1 Finite Element Methods

Deformation induces movement of every particle within an object. It can be modeled as a mapping of the positions of all particles in the original object to those in the deformed body. Each point $\mathbf{p}$ is moved by the deformation function $\phi(\cdot)$:

$$\mathbf{p} \rightarrow \phi(t, \mathbf{p})$$

where $\mathbf{p}$ represents the original position, and $\phi(t, \mathbf{p})$ represents the position at time t. We limit the discussion to the static analysis, hence t is omitted: $\mathbf{p} \rightarrow \phi(\mathbf{p})$. Simulating deformation is in fact finding the $\phi(\cdot)$ that satisfies the laws of physics. Since there are an infinite number of particles, $\phi(\cdot)$ has infinite degrees of freedom.

For simulation of deformable bodies, spring networks, the finite difference method (FDM), the boundary element method (BEM), and the finite element method (FEM) have all been used for discretization. In our prototype simulator, we have chosen FEM as the discretization method due to its generality and diversity. The FEM uses a piecewise approximation of the deformation function $\phi(\cdot)$. Each "piece" is called an element, which is defined by several node points. The elements constitute a mesh. Since the FEMs pose relatively small restrictions on the mesh topology, they are suitable for representing a variety of shapes and topology.

Our algorithm uses a FEM with 4-node tetrahedral elements and linear shape functions. However, other non-linear shape functions can be used as well. But, this will affect the update of the distance field computation, as the objects deform (section 5).

The deformation function $\phi(\cdot)$ maps a point in a tetrahedral element at $\mathbf{p} = [x, y, z]^T$ to a new position $\phi(\mathbf{p})$. By definition, $\phi(\cdot)$ moves four nodes of an element from their original positions

$$\mathbf{n_i} = [n_{ix}, n_{iy}, n_{iz}]^T, 1 \leq i \leq 4,$$

to the new positions

$$\tilde{\mathbf{n}}_\mathbf{i} = [\tilde{n}_{ix}, \tilde{n}_{iy}, \tilde{n}_{iz}]^T, 1 \leq i \leq 4.$$

The displacements of the four nodes due to deformation is

$$\begin{aligned} \mathbf{U_i} &= [U_{ix}, U_{iy}, U_{iz}]^T \\ &= [\tilde{n}_{ix} - n_{ix}, \tilde{n}_{iy} - n_{iy}, \tilde{n}_{iz} - n_{iz}]^T, 1 \leq i \leq 4. \end{aligned}$$

3.2 Simulation Framework

Given the basics of FEM, we reformulate the problem of simulating deformable objects as a constrained optimization problem using Constitutive Law [22] by minimizing the total energy in the system due to deformation. Details of the simulator are given in [11]. Fig. 1 gives a brief overview of the simulator used to test our algorithm for computing estimated penetration depth between flexible models.

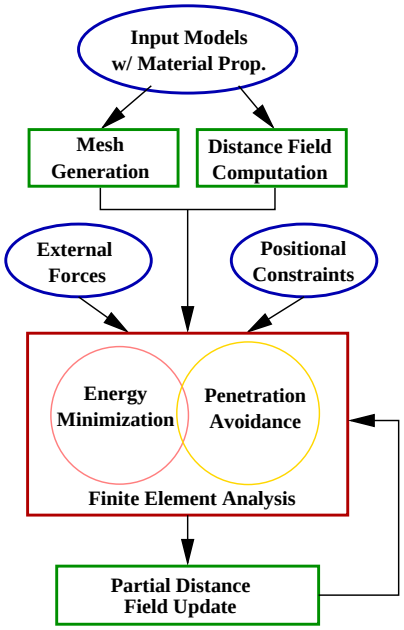

Figure 1: A system overview showing various components of the simulator used to test our algorithm

3.3 Overview of Our Approach

Given the finite element meshes of two flexible bodies, our algorithm precompute an internal distance field for each undeformed model using the fast marching level-set method (sec. 4). As the two objects come into contact and deform, the algorithm use

1. A *hierarchical sweep-and-prune* [14] when the NURB representations of the models are given;
2. A lazy evaluation of possible intersections using bounding volume hierarchies of axis-aligned bounding boxes [23].

The collision detection module identifies the "regions of potential contacts", as well as the intersecting tetrahedral elements. The intersecting tetrahedral elements are then used to compute the estimated penetration depth based on the pre-assigned distance values at the nodes of each element (sec. 5). This is a fast output-sensitive computation requiring $O(K)$ time, where K is the number of pairs of intersecting tetrahedral elements, and is normally small compared to the number of elements within each model.

Since the pre-assigned distance values at each node of the tetrahedral elements may no longer be valid after the deformation, we need to either recompute or adaptively update these distance values. Since recomputation of the entire internal distance field for each deformed model can be rather expensive, we perform a partial recomputation of distance field only at and near the regions of potential contacts indicated by the collision detection module and FEM simulation. We also ensure the continuity and differentiability of the distance field at the boundary of these regions. The values of updated distance fields are then used for the next simulation step.

This process continues iteratively to estimate the penetration depth between elastic bodies quickly and efficiently during the simulation.

4 Internal Distance Fields

The Fast Marching Level Set Method was first designed to track the evolution of fronts through a 3D space. In our application, the surface of an arbitrary 3D object is treated as a front. The surface is propagated *inwards*, opposite of the direction of the surface normal. As the surface evolves with uniform speed, distance values from the surface are assigned to points on a discretized grid.

The Fast Marching Level Set Method input to the algorithm consists of a polygonal mesh. Models consisting of implicit representations or parametric surfaces, such as NURBS, can be tessellated into polygonal meshes with bounded error. The user may also specify the resolution of the 3D grid, trading accuracy for speed. The output of the method is a discretized distance field for the volume encompassed by the 3D surface. In practice, interpolation methods are used when sampling the distance field for penetration depth computations.

Several key terms are used in the presentation of this algorithm. A gridp oint may be marked with one of three labels: ALIVE, NARROW_BAND, or FAR_AWAY. An ALIVE point represents a grid point who has already been assigned a distance value. A NARROW_BAND point represents a point on the evolving front. A FAR_AWAY point represents a point without an assigned distance value.

4.1 Initialization

To compute distance values for an arbitrary object requires initializing the location of the surface within a 3D grid. For each triangle of the polygonal mesh, an axis-aligned bounding box is created. Distance values for each grid point in the bounding box are then defined. When the initialized value is greater than or equal to zero, the grid point lies outside of the object or on the surface. These grid points are marked ALIVE. When the distance value is negative, it lies inside the object, and the grid point is marked NARROW_BAND.

The set of NARROW_BAND points represents those within a neighborhood of the zero level set. Restricting work to only this neighborhood of the zero level set yields a considerable reduction in computational cost. This method of computation is known as the *narrow band approach*, and is discussed in detail in [20].

4.2 Marching

Once the surface has been initialized in the 3D grid, the marching phase of the algorithm may commence. At each step,

the grid point with the minimum distance value is extracted from the set of NARROW_BAND grid points. The data structure underlying this phase of the algorithm is discussed in section 4.3. Upon selection of the minimum valued NARROW_BAND grid point, it is marked ALIVE, and any FAR_AWAY neighbors are moved to the set of NARROW_BAND points. The distance value for each neighboring NARROW_BAND point is then updated by solving for T in the following equation, selecting the largest possible solution to the quadratic equation:

$$\frac{1}{F_{ijk}} = \sqrt{A + B + C}$$

where

$$A = (\max(D^{-x}T + \frac{\Delta x}{2} D^{-x-x}T, D^{+x}T + \frac{\Delta x}{2} D^{+x+x}T, 0))^2$$

where the finite differences are given by

$$\begin{aligned} D^{+x} &= \frac{T_{i+1} - T}{\Delta x} \\ D^{-x} &= \frac{T - T_{i-1}}{\Delta x} \\ D^{+x+x} &= \frac{T_{i+2} - 2T_{i+1} + T}{2\Delta x} \\ D^{-x-x} &= \frac{T - 2T_{i-1} + T_{i-2}}{2\Delta x} \end{aligned}$$

Similarly,

$$B = (\max(D^{-y}T + \frac{\Delta y}{2} D^{-y-y}T, D^{+y}T + \frac{\Delta y}{2} D^{+y+y}T, 0))^2$$

$$C = (\max(D^{-z}T + \frac{\Delta z}{2} D^{-z-z}T, D^{+z}T + \frac{\Delta z}{2} D^{+z+z}T, 0))^2$$

The term F_{ijk} represents the speed of the propagating front. Because we wish to find the distance from each point to the surface, this value is uniform (constant) in our application.

The equations use a second order scheme whenever possible to produce higher accuracy. That is, both T_{i+2} and T_{i+1} must be ALIVE in order to compute D^{+x+x}, D^{+y+y} or D^{+z+z}, where $T_{i+2} > T_{i+1}$. The choice of when to use the second order scheme simply depends on whether two known (ALIVE), monotonically increasing values exist as neighbors of the test point. If not, then the first order scheme is used.

This process of selecting a minimum NARROW_BAND point, marking it ALIVE, and updating neighbors continues until no NARROW_BAND points remain. This algorithm to compute an internal distance field for each object can be summarized as follows.

```
GridPoint G;
InitializeGrid();
heap = BuildHeap();  //NARROW_BAND POINTS
while (heap.isEmpty() != TRUE)
{
      G = heap.extractMin();
      G.status = ALIVE;
      markNeighbors(G);
      updateNeighbors(G);
}
```

4.3 Data Structures

To satsify the need for an efficient sorted data structure and retain spatial information, we use both a minimum heap structure and a 3D array. Each heap node contains a pointer to the 3D array grid point that it references. Similarly, each NARROW_BAND grid point in the 3D array points to a node in the heap. ALIVE and FAR_AWAY points have NULL pointers as only NARROW_BAND points are included in the heap.

4.4 Partial Update of Distance Field

When an object deforms, the simulator identifies the regions of deformation using the collision detection module. This information is then used to quickly update portions of the internal distance field.

4.4.1 Collision Detection

For collision detection, we use the *hierarchical sweep-and-prune* described in [14], when the original, corresponding NURB representations of the models are available. Each surface patch is subdivided into smaller patches and represented hierarchically. Each leaf node corresponds to a spline patch whose surface area is less than an input parameter Δ used in generating the polygonal meshes of the patch. The resulting tree has a shallow depth and each node can have multiple children. An axis-aligned bounding box is computed for the control polytope of each patch and dynamically updated. At each level of hierarchy, the sweep-and-prune [6] is used to check for overlap of the projections of the bounding boxes onto $x-, y-, z-$ axes. Only when the boxes overlap in all three dimensions, a potential contact is returned. Coherence is exploited to keep the runtime linear to the number of bounding boxes at each level. The resulting hierarchical sweep-and-prune can be efficiently employed to check for potential overlaps of the hierarchies.

If the NURB representations of the models are not available, we lazily construct the bounding volume hierarchies (BVHs) based on axis-aligned bounding boxes for each model *on the fly* and check for collision between them using these binary BVHs. For more details, we refer the readers to [23].

4.4.2 Lazy Evaluation

Given the regions of potential contacts returned (as one or more bounding boxes) by the collision detection module, we perform partial update of the internal distance field by only recomputing the distance values at each grid point within these regions. With such methods as FEM and finite difference methods, this information is easy to obtain. These methods treat objects volumetrically, and therefore they retain information on how far the effects of deformation have propagated throughout the object.

Given the bounding box, a second 3D grid is created that overlays the first. The algorithm to compute this partial grid is the same algorithm previously described; the savings in computation time comes from the reduction of the number of grid points being computed. Once the marching completes, we have two datasets that need to be combined while preserving the continuity and differentiability of the solutions.

In practice, these separate data sets are almost always continuous. We verify continuity by examining the gradient across

the border of the two sets. In rare cases where the data sets are discontinuous, other options are available.

One option is to linear interpolate the two data sets to obtain a continuous solution. This option is only viable when the degree of discontinuity is low. In cases where the resulting data set is highly discontinuous, the entire object is recomputed. In our test applications, this situation never occurred. This is due to the accuracy of the bounding boxes for partial update generated by our collision detection and FEM algorithm.

5 Penetration Depth Estimation

When using the penalty based method, we need to first define a penetration potential energy $W_{penet}(\cdot)$ that measures the amount of intersection between two polyhedra, or the degree of self-intersection of a single polyhedron. This definition requires an efficient method to compute it, its first and second derivatives.

5.1 Defining the Extent of Penetration

There are several known methods to define the extent of intersection. The node-to-node method is the simplest way to compute $W_{penet}(\cdot)$. This method computes $W_{penet}(\cdot)$ as a function of the distances between sampled points on the boundary of each objects. The drawback of this method is that once a node penetrates boundary polygons, the repulsive force flips its direction, and induces further penetration. Such penetration often occurs in intermediate steps of the aggressive numerical methods. Furthermore, once a node is inside a tetrahedral element, it is no longer clear which boundary polygon the node has actually penetrated.

The most complicated yet accurate method is to use the intersection volume. Using this method, $W_{penet}(\cdot)$ is defined based on the volume of intersection between two penetrating polyhedra. Since polyhedra deform as simulation steps proceed, it is difficult to create and reuse previous data from the original model. Furthermore, it is susceptible to accuracy problems and degenerate contact configurations. So, efficient computation of the intersection volume is rather difficult to achieve.

We have chosen a method that provides a balance between the two extremes by computing an *approximate* penetration depth between deformable objects. With our method, $W_{penet}(\cdot)$ is defined as a function of distances between boundary nodes and boundary polygons that the nodes penetrate. We define

$$W_{penet}(\cdot) = k * d^2 \tag{1}$$

where d is the minimum distance from a boundary node to the intruded boundary and k is a penalty constant.

5.1.1 Estimating Penetration Depth

Our algorithm estimates the computation of the penetration depth d by replacing it with the linear interpolation $\tilde{d}$ of preassigned distance values:

$$\tilde{d} = u_1\, d_1 + u_2\, d_2 + u_3\, d_3 + (1 - u_1 - u_2 - u_3)\, d_4 \tag{2}$$

where d_1, d_2, d_3 and d_4 are distance values at the four nodes of each tetrahedral element. These distance values are sampled from the distance field generated by the fast marching level set method as described in section 4. u_1, u_2 and u_3 are the interpolation parameters derived from the shape functions of the elements, and $0 \le u_i \le 1, 1 \le i \le 3$.

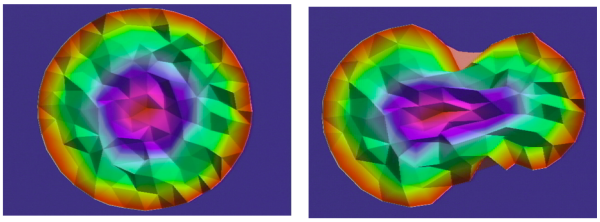

Figure 2: LEFT: The distance field of a sphere. RIGHT: The distance field of a deformed sphere computed using linear interpolation of the precomputed distance field.

Once an accurate value of distance is assigned to each node, no matter how the mesh is deformed, the value of $\tilde{d}$ is quickly computed at any point inside the object. Figure 2 shows an example where the distance field of a sphere is quickly re-computed as the sphere deforms.

This approximated distance field shares a few properties with the exact distance field. Some of these properties are essential for proper computation of penalty forces and their derivatives:

1. It vanishes on the boundary polygons.
2. It is twice differentiable inside the elements and C^0 continuous everywhere.

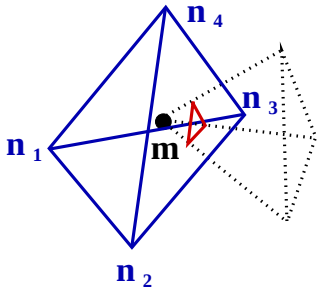

Figure 3: A node **m** penetrates into another tetrahedral element. The distance between **m** and the red triangle is the penetration depth.

A hierarchical sweep and prune method [14] or a lazy collision evaluation based on bounding volume hierarchies of axis-aligned bounding boxes [23] is used to find each instance where a boundary node from one element penetrates another element. Suppose a boundary node $\mathbf{m}$ is within an element with nodes $\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_3}$ and $\mathbf{n_4}$ as shown in Figure 3. $\mathbf{m}$ can be written in terms of linear interpolation of $\mathbf{n_1}, \ldots, \mathbf{n_4}$:

$$\mathbf{m} = u_1\, \mathbf{n_1} + u_2\, \mathbf{n_2} + u_3\, \mathbf{n_3} + (1 - u_1 - u_2 - u_3)\, \mathbf{n_4} \tag{3}$$

$\tilde{d}$ at $\mathbf{m}$ is obtained by solving Eqn. 2 and Eqn. 3:

$$\tilde{d} = [d_1 - d_4, d_2 - d_4, d_3 - d_4]\, G^{-1}\, [\mathbf{m} - \mathbf{n_4}] + d_4 \tag{4}$$

where

$$\mathbf{G} = [\mathbf{n_1} - \mathbf{n_4}, \mathbf{n_2} - \mathbf{n_4}, \mathbf{n_3} - \mathbf{n_4}]$$

Figure 4: A snake swallowing an apple from a bowl of fruits

$W_{penet}(\cdot)$ is computed by using $\tilde{d}$ instead of d in Eqn. 1.

This algorithm is insensitive to which object (or connected mesh) the nodes $\mathbf{m}$ and $\mathbf{n}$ belong to. Therefore, self-intersections and intersections between two objects are treated in a uniform manner. It is also robust enough to recover from penetrations of significant depth.

6 Implementation and Results

We have implemented the algorithm described in this paper and have successfully integrated it into a moderately complex simulation with video clips shown at our project website:

http://www.cs.unc.edu/∼geom/DDF/

We used Maya developed by Alias|Wavefront to generate the models used in our simulation sequences. We used a public domain mesh generation package, SolidMesh [18], to create tetrahedral elements used in our FEM simulation. Rendering of the simulation results was displayed using OpenGL on a 300MHZ R12000 SGI Infinite Reality.

6.1 Demonstration

Figure 4 are snapshots from a simulation sequence where a snake swallows a deformable red apple from a bowl of fruit. The snake and the apple models have a total of 23,000 elements. Eight major keyframes were used to set the positional constraints. The deformation of the apple and the snake was computed by the simulator using our algorithm to estimate penetration depths between deformable bodies.

6.2 Choice of Grid Resolution

The choice of the grid resolution has a significant effect the runtime performance and accuracy of the distance field computation using fast marching level-set methods. In fact, fast marching level-set methods runs in $O(kn^2)$ worst-case time using the "narrow band approach" [20], given the grid resolution of n x n x n and k is the number of cells in the narrow band. Table 1 gives an example of the computation results using different grid resolutions on a sphere of 1000 triangles with the correct distance value of 1.0 at the center of the sphere.

Note that the computed values for the internal distance field are much more accurate at the regions near the surface of the object. This is appropriate for our application where the penetration is normally not deep. The deviation between the correct distance value and the computed distance value at the center of the sphere indicates the maximum error possible due to the accumulation of numerical inaccuracies, as the level-set computation marching in toward the center.

Resolution	Ctr Value	Dist. Field	1/8 D. Field
60x60x60	0.921986	57.4696	2.02469
55x55x55	0.916389	28.9428	1.16319
50x50x50	0.912209	17.4810	0.71547
40x40x40	0.898008	3.81680	0.29566
30x30x30	0.878681	0.52117	0.08658
20x20x20	0.875549	0.10853	0.02734

Table 1: The effect of grid resolutions on the accuracy and performance (in seconds) of distance field & partial update computations

6.3 Partial Update of Distance Fields

Table 1 also illustrates the performance gain in computing partial update of the distance field over the recalculation of the entire distance field. The last two columns of Table 1 give the computation time (in seconds) required for computing the entire distance field of the sphere vs. updating only $1/8$ of its distance field. The speed up is quite substantial, especially for those with higher grid resolutions.

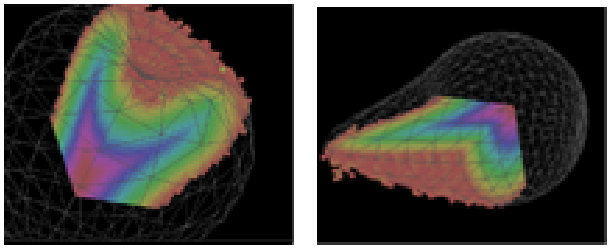

Figure 5: Partial distance field update of an apple (LEFT) and a cross section of a deformed sphere (RIGHT)

The timing (in seconds) for partial update vs. complete recomputation of the distance fields for various models, including a torus, an apple and a deformed sphere (Fig. 5), is given in Table 2. Note that the torus model with more triangles and the same grid resolution takes less time to compute than a simpler apple mode with far less polygons. This is due to the fact that the torus

model only occupies a small portion of the grids allocated; while the apple occupies majority of the grid space allocated.

Model	Resolution	Tri's	Dist. Field	1/8 D. Field
Torus	50x50x50	2048	1.04334	0.290281
Apple	50x50x50	384	10.6384	0.958958
Sphere	50x50x50	972	5.21021	0.516960

Table 2: Timing (in seconds) on partial update of the distance field vs. the recomputation of the entire distance field

6.4 Discussion

Although our current implementation is based on the use of FEM simulator [11], our algorithm can be applied to simulation methods using finite difference methods (FDM) and will require little modification. One can replace the linear interpolation step using shape functions of FEM (explained in section 5) with a linear interpolation suitable for FDM. For the spring-mass systems, each mass can be considered as a node of each finite element and the same formulation will apply.

There is some limitation to our approach. Our method computes the internal distance fields within each object. Therefore, it is not best suited for handling self-penetration of very thin objects, such as cloth or hair, which are often encountered in character animation.

7 Summary

As the frontier of robotics extends beyond its traditional domains and into medical and other more advanced applications, modeling deformation becomes a key component for robotic simulation. In this paper, we present a fast penetration depth estimation algorithm between elastic bodies for simulating complex deformation due to non-penetration constraints using penalty-based methods. And, we demonstrate its efficiency and effectiveness on moderately complex simulation scenarios.

Acknowledgements

We wish to thank Gentaro Hirota for the use of his simulator, and his assistance in testing our algorithm. Our implementation also benefited from discussions with David Adalsteinsson of the Department of Mathematics at UNC Chapel Hill. This work is supported in part by National Science Foundation, Office of Naval Research and Intel Corporation.

References

[1] P. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir. Penetration depth of two convex polytopes in 3d. *Nordic J. Computing*, 7:227–240, 2000.

[2] C. E. Buckley and L. J. Leifer. A Proximity Metric For Continuum Path Planning. *Proc. of Int. Conf. on Artificial Intelligence*, 1096-1102, 1985.

[3] S. Cameron. Enhancing gjk: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[4] S. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 591–596, 1986.

[5] P. G. Ciarlet and J. L. Lions, editors. *HANDBOOK OF NUMERICAL ANALYSIS*, volume I - VI. Elsevier Science B.V., 1994.

[6] 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.

[7] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[8] S. F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical Report Technical Report, Mitsubishi Electric Research Laboratory, 1997.

[9] E.G. Gilbert and C.J. Ong. New distances for the separation and penetration of objects. In *Proceedings of International Conference on Robotics and Automation*, pages 579–586, 1994.

[10] Leonidas J. Guibas and J. Stolfi. Ruler, compass and computer: the design and analysis of geometric algorithms. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume 40 of *NATO ASI Series F*, pages 111–165. Springer-Verlag, 1988.

[11] G. Hirota, S. Fisher, and M. C. Lin. Simulation of nonpenetrating elastic bodies using distance field. Technical report, Department of Computer Science, University of North Carolina, 2000.

[12] 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.

[13] K. Hoff, A. Zaferakis, M. C. Lin, and D. Manocha. Fast and simple geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*, 2001.

[14] M. Hughes, C. Dimattia, M. Lin, and D. Manocha. Efficient and accurate interference detection for polynomial deformation and soft object animation. In *Proceedings of Computer Animation*, pages 155–166, Geneva, Switzerland, 1996.

[15] Q. Lin, J. W. Burdick, and E. Rimon. Computation and analysis of compliance in grasping and fixturing", Proc. of Int. Conf. on Robotics and Automation, pp 93-99, 1997.

[16] K. Sridharan, H. E. Stephanou, K. C. Craig and S. S. Keerthi Distance measures on intersecting objects and their applications. In *Information Processing Letters*, Vol. 51, Aug. 1994, pp. 181-188.

[17] R. Kimmel, A. Amir, and A. M. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1), 1995.

[18] D.L. Marcum and N.P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal*, 33(9), September 1995.

[19] S. J. Osher and J. A. SEthian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *J. of Comp. Phys.*, 1988.

[20] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[21] Patrick Le Tallec. Equilibrium problems with frictionless contact. In P. G. Ciarlet and J. L. Lions, editors, *HANDBOOK OF NUMERICAL ANALYSIS*, volume 3, pages 565–578. Elsevier Science B.V., 1994.

[22] Patrick Le Tallec. Numerical methods for nonlinear three-dimensional elasticity. In P. G. Ciarlet and J. L. Lions, editors, *HANDBOOK OF NUMERICAL ANALYSIS*, volume 3, pages 465–622. Elsevier Science B.V., 1994.

[23] A. Wilson, E. Larsen, D. Manocha, and M. C. Lin. Partitioning and handling massive models for interactive collision detection. *Computer Graphics Forum (Proc. of Eurographics)*, 18(3):319–329, 1999.