Deformed Distance Fields for Simulation of Non-Penetrating Flexible Bodies

Susan Fisher Ming C. Lin

Department of Computer Science
University of North Carolina at Chapel Hill
U. S. A.

Abstract

We present a novel penetration depth estimation algorithm based on the use of deformed distance fields for simulation of non-penetrating flexible bodies. We assume that the continuum of non-rigid models are discretized using standard techniques, such as finite element methods. As the objects deform, the distance fields are deformed accordingly to estimate penetration depth, allowing enforcement of non-penetration constraints between two colliding elastic bodies. Our approach can automatically handle self-penetration and inter-penetration in a uniform manner. We demonstrate its effectiveness on moderately complex animated scenes.

1 Introduction

Due to recent advancements in physically-based modeling, simulation techniques have been increasingly used to improve the quality and efficiency of producing computer animation for major film productions, medical simulation and computer games. These techniques produce animation directly from input objects, simulating natural motions and shape deformations based on mathematical models that specify the physical behavior of characters and complex structures.

Modeling deformation is a key component of physically-based animation, since many real-world objects are not rigid. Some examples include realistic motion generation of articulated characters with passive objects (such as clothing, footwear and other accessories), deformation of soft tissues and organs and interaction among soft or elastic objects. Automatic, predictable and robust simulation of realistic deformation is one of the many challenges in computer animation and medical simulation [8].

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 the *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, a penetration potential energy must first be defined 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 rigid 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]. Neither a complexity bound for this problem nor a formal definition of penetration depth for deformable models has yet been established.

1.1 Main Contribution

We present an efficient algorithm based on the use of deformed distance fields for simulating deformation 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) [5] 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 [18, 19] to precompute the internal distance field of each *undeformed* model. When two flexible bodies come into contact and deform, the precomputed distance fields are likewise deformed to compute the estimated penetration depth between two deforming objects. 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 contact forces 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. Specifically, our penetration depth estimation algorithm has the following characteristics:

- Both **self-collisions** and **soft object contacts** are handled in a uniform manner.
- **No prior assumption** or knowledge about the locations of contacts is required.
- The algorithm can **trade off accuracy for speed or storage** if desired.

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 [3]. Several algorithms [7, 9, 15] have been proposed for computing a measure of penetration depth using various definitions. 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], where m and n are the number of vertices of the two polyhedra. However, all existing methods 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 the 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 rigid, non-convex polyhedra to find the minimum penetration depth [7], where n is the number of vertices for each polyhedron. 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. As it stands today, *interactive* computation of penetration depth between two general geometric models of high complexity remains an open research issue.

2.2 Distance Field

Computing the minimum geodesic distance from a point to a surface is a well known complex problem [16]. Osher and Sethian [18, 19], 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]. Frisken, et al. also presented an adaptive technique to compute distance fields [4].

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 Discretization 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 here, 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. In order to model a material's behavior using computer simulation, some type of *discretization* method must be used. 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.

3.2 Tetrahedral Elements

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.

Our algorithm uses a FEM with 4-node tetrahedral elements and linear shape functions. Other non-linear shape functions can be used as well, but the update of the distance field computation as the objects deform will be affected (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 $\phi(\mathbf{p})$. As shown in Fig. 1, 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

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

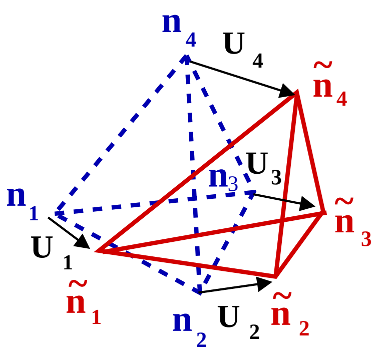

Figure 1: $\phi(\cdot)$ maps four nodes of a tetrahedral element, $\mathbf{n_1}, ..., \mathbf{n_4}$ to their new position at $\tilde{\mathbf{n}}_\mathbf{1}, ..., \tilde{\mathbf{n}}_\mathbf{4}$. $\mathbf{U_1}, ..., \mathbf{U_4}$ are the corresponding displacement vectors.

3.3 Simulation Framework

Given the basics of FEM, we reformulate the problem of simulating deformable objects as a constrained minimization problem using Constitutive Law [5]. Details of the simulator are given in [11]. Here we give a brief overview of the simulator used to test our algorithm for computing estimated penetration depth between flexible models:

1. Generate an internal distance field for each input object using the Fast Marching Level Set Method (sec. 4).
2. Construct a tetrahedral element mesh for each object.
3. Apply finite element analysis:
   (a) Estimate the penetration depth based on the deformed distance fields (sec. 5) for penetration avoidance.
   (b) Minimize the total energy due to deformation, taking into account all material properties and external forces, using our synthesized numerical method [11].
4. Incrementally update portions of the distance fields, given the new positions and orientations of the deformed bodies.

3.4 Algorithm Overview

As two objects come into contact and deform, the algorithm uses

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 [20].

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 input to the Fast Marching Level Set Method 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 gridpoint may be marked with one of three labels: ALIVE, NARROW_BAND, or FAR_AWAY. An ALIVE point represents a grid point that 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 [19].

4.2 Marching

Once the 3D grid has been initialized, 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:

$$\frac{1}{F_{ijk}} = \sqrt{M + N + O}$$

where

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

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,

$$\begin{aligned} N &= (\max(D^{-y}T + \frac{\Delta y}{2} D^{-y-y}T, D^{+y}T + \frac{\Delta y}{2} D^{+y+y}T, 0))^2 \\ O &= (\max(D^{-z}T + \frac{\Delta z}{2} D^{-z-z}T, D^{+z}T + \frac{\Delta z}{2} D^{+z+z}T, 0))^2 \end{aligned}$$

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

With each step of the algorithm, the minimum valued NARROW_BAND grid point must be extracted. The need for an efficient extraction operation, as well as an efficient insertion operation makes the use of a heap ideal. However, once the minimum valued grid point NARROW_BAND grid point has been identified, the algorithm updates each neighboring point. Thus, in addition to the need for an efficient sorted data structure, we must also retain spatial information.

Our solution is simply to 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 use the collision detection module to quickly identify the neighborhood where partial update of distance field needs to be performed. This information also finds the instances where boundary nodes penetrate other elements.

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 [20].

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 combine two sets of computed distance values (one from the precomputation and one from the partial update), while preserving the continuity and differentiability of the solutions that are crucial for computing collision response robustly [11].

In practice, these two separated sets of distance values are almost always continuous. We verify continuity by examining the gradient across the border of the two sets. In rare cases where the distance values 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.

Without partial updates, the distance field can become less and less accurate due to large deformations as the simulation proceeds. Eventually the accumulated errors in the resulting distance fields can cause visually disturbing artifacts in simulations.

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, and its first and second derivatives, for computing the collision response robustly [11].

5.1 Defining the Extent of Intersection

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. As a result, 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.

5.2 Estimating Penetration Depth

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 2. Using a linear shape function, $\mathbf{m}$ can be written in terms of linear interpolation of $\mathbf{n_1}, \ldots, \mathbf{n_4}$:

$$\mathbf{m} = u_1\ \mathbf{n_1} + u_2\ \mathbf{n_2} + u_3\ \mathbf{n_3} + (1 - u_1 - u_2 - u_3)\ \mathbf{n_4} \tag{2}$$

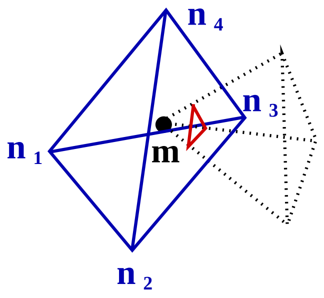

Figure 2: A node $\mathbf{m}$ penetrates into another tetrahedral element. The distance between $\mathbf{m}$ and the red triangle is the penetration depth.

Our algorithm estimates the computation of the penetration depth d by replacing it with $\tilde{d}$ at $\mathbf{m}$ using the linear interpolation of pre-assigned distance values:

$$\tilde{d} = u_1\ d_1 + u_2\ d_2 + u_3\ d_3 + (1 - u_1 - u_2 - u_3)\ d_4 \tag{3}$$

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 by solving Eqn. 2:

$$[u_1, u_2, u_3]^T = G^{-1}\ [\mathbf{m} - \mathbf{n_4}] \tag{4}$$

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

Thus, $$\tilde{d} = [d_1 - d_4, d_2 - d_4, d_3 - d_4]\ G^{-1}\ [\mathbf{m} - \mathbf{n_4}] + d_4 \tag{5}$$

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 3 shows an example where the distance field of a sphere is quickly re-computed as the sphere deforms.

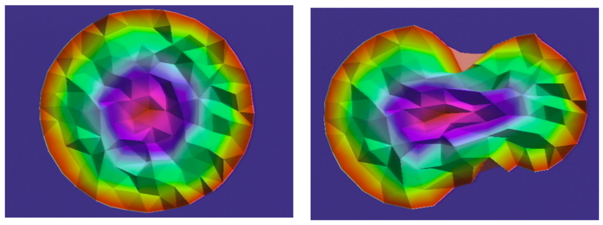

Figure 3: LEFT: The distance field of a sphere. RIGHT: The distance field of a deformed sphere computed using linear interpolation of the precomputed distance field.

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.

$W_{penet}(\cdot)$ 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 System 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 [17], to create tetrahedral elements used in our FEM simulation. Rendering of the simulation results was displayed using OpenGL on a 300MHZ R12000 SGI IR.

6.1 System Demonstration

Figure 4 shows a large deformation simulated by our algorithm. Two sets of positional constraints were specified for internal nodes in the head part and the tail part. Given the positional constraints, the head of snake is forced to move toward its tail. The snake model has about 14,000 elements. Our algorithms enables the simulation to automatically generate the natural coiling deformation. It is not obvious from the images, but many small self-penetrations were resolved during the deformation.

Figure 5 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

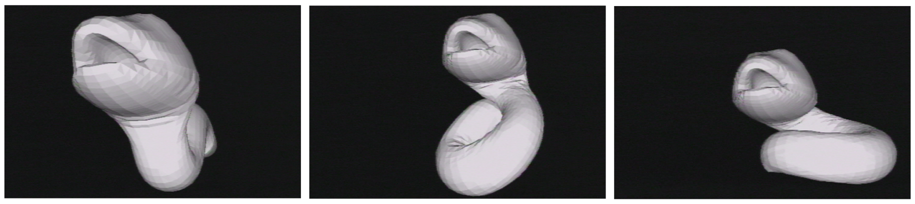
Figure 4: Large Deformation: A snake coiling up

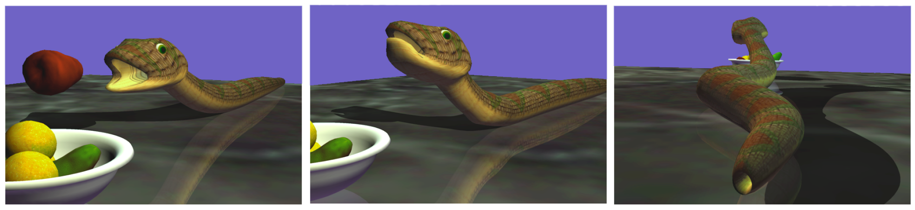
Figure 5: A snake swallowing an apple from a bowl of fruits

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" [19], 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 time using different grid resolutions n x n x n on a sphere of 1000 triangles with the correct distance value of 1.0 at the center of the sphere for the entire distance field vs. updating $1/8$ of the distance field.

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

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

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

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.

6.3 Partial Update of Internal Distance Fields

Table 1 also illustrates the performance gain in computing partial updates 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. The better performance gain on grids with higher resolution is due to faster cache access for smaller datasets.

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 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 model with far less polygons. This is due to the fact that the torus model actually 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 sec	0.290281 sec
Apple	50x50x50	384	10.6384 sec	0.958958 sec
Sphere	50x50x50	972	5.21021 sec	0.516960 sec

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 a 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

In this paper, we present a novel geometric technique, which first precomputes internal distance fields for each object, *deforms* each field *on the fly*, and then later utilizes them for enforcing the non-penetration constraints based on a penalty method. By taking advantage of precomputed distance fields that deform as the finite element mesh deforms, our algorithm enables efficient computation of penalty forces and their derivatives, and yields a versatile and robust contact resolution algorithm.

This algorithm can be useful for many applications, such as simulation of passive deformable tissues in computer animation. It can also be incorporated into medical simulation used for multi-modal image registration, surgical planning and instructional medical illustration.

Acknowledgements

We wish to thank Gentaro Hirota for the use of his simulator and his assistance in testing our algorithm, and the anonymous reviewers for their feedback. This implementation also benefited from discussions with David Adalsteinsson. This research is supported in part by ARO DAAG55-98-1-0322, NSF DMI-9900157, NSF IIS-9821067, ONR N00014-01-1-0067 and Intel.

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 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.

[4] S. Frisken, R. Perry, A. Rockwood, and T. Jones", Adaptively sampled distance fields: a general representation of shapes for computer graphics. *Proc. of ACM SIGGRAPH*, pages 249–254, 2000.

[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] 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.

[16] 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.

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

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

[19] 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.

[20] 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.