Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2014)
J. Bender, C. Duriez, F. Jaillet, and G. Zachmann (Editors)

Continuous Collision Detection Between Points and Signed Distance Fields

Hongyi Xu and Jernej Barbič

University of Southern California, USA

Abstract

We present an algorithm for fast continuous collision detection between points and signed distance fields. Such robust queries are often needed in computer animation, haptics and virtual reality applications, but have so far only been investigated for polygon (triangular) geometry representations. We demonstrate how to use an octree subdivision of the distance field for fast traversal of distance field cells. We also give a method to combine octree subdivision with points organized into a tree hierarchy, for efficient culling of continuous collision detection tests. We apply our method to multibody rigid simulations, and demonstrate that our method accelerates continuous collision detection between points and distance fields by an order of magnitude.

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Simulation and Modeling]: Types of Simulation—Animation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Given a query point x from some region of space, such as a bounding box enclosing the geometry, the *distance field* is a scalar function that gives the minimum distance from x to the geometry. Distance fields sampled on regular 3D grids are a popular datastructure in computer graphics [JBS06], and have been used in many applications, such as collision detection and morphing. Distance fields can be signed or unsigned. Signed distance fields store the sign specifying whether the query point is inside/outside of the object. Representing surfaces by a distance field is advantageous since there are no restrictions about the topology [TKH*05].

Distance fields have been employed to detect collisions, especially for rigid bodies, and even self-collisions. Their power originates from the fact that distances to the nearest geometry can be approximated for arbitrary query locations by simple trilinear interpolation in $O(1)$ time, independent of the geometric complexity of the object. However, existing methods only applied distance fields to discrete collision detection. Continuous collision detection is regarded as more robust as it finds the exact contacts of dynamically simulated objects between two successive time steps. Previous methods focused on explicit surface representations such as polygonal (triangular) geometry and pairwise face/vertex and edge/edge continuous collision detection tests. We propose a continuous collision detection algorithm between points and implicit surfaces represented by distance fields. Assuming a linear trajectory of points and the distance field object during each timestep, the intersection(s) with a distance field isosurface can be detected by checking a line segment against a signed distance field. We accelerate this process using a spatial octree subdivision of the distance field, storing the minimum and maximum distance values for octree subtrees. This enables a fast traversal of the distance field grid cells. We also demonstrate how to combine a sphere hierarchy of points (we use the nested point tree [BJ08]) with the fast grid traversal. This can effectively cull unnecessary continuous collision detection tests, especially in physically based simulations. Our experiments demonstrate that we can achieve significant speedups using our acceleration techniques. We demonstrate the effectiveness of our algorithm using rigid body contact simulations. Inspired by the continuous penalty force model [TMOT12], we present a contact resolution method to match our continuous collision detection algorithm. In addition to forces [TMOT12], our method also computes contin-

DOI: 10.2312/vriphys.20141218

uous penalty torques and damping forces. Our contributions include:

- an algorithm for continuous collision detection between points and distance fields, via line segment vs distance field intersection tests,
- fast grid traversal using a spatial octree subdivision of the distance field,
- point-tree traversal algorithm for continuous collision detection culling,
- continuous penalty contact and damping model to be employed in concert with our continuous collision detection.

2. Related Work

Numerous approaches have been investigated to detect collisions between interfering objects; see, for example, the survey [TKH*05]. Collision detection can be categorized into discrete and continuous. Discrete methods only check for collisions at specified time instances. For a greater computational cost, continuous collision detection provides more robustness by detecting all the collisions between two discrete time instances [BFA02, RKLM05, TCYM09, TMT10, BEB12]. Most of the existing continuous methods work by computing the roots of polynomial functions, to resolve the continuous collisions between basic pairs of polygonal primitives such as triangle/vertex or two edges. Collisions between analytical implicit and parametric functions that deform in time can be resolved using interval arithmetics [SWF*93, RKC02]. The equations become cubic when linearly interpolating vertex motion [Pro97, BFA02, TMOT12]. Continuous collision between such pairs of primitives is sensitive to numerical error and the employed tolerances, requiring special care [BB09, Wan14]. Different from these polygon-based methods, we detect continuous collisions between implicit functions and points, by intersecting distance fields against line segments. The complexity of our algorithm depends on the number of points and the distance field resolution, but is independent of the underlying triangular geometry.

A distance field datastructure can rapidly provide the distance to any isosurface for any location in space. Therefore, signed distance field can quickly detect collisions, and have been used in many rigid-rigid [HXB14] and rigid-deformable simulations [BMF03, FSG03, BJ08]. These previous methods were, however, designed for discrete collision detection, whereas we perform continuous collision detection. Ray tracing for implicit isosurface rendering has been well studied [PSL*98, NMHW02, MKW*04]. Specially, Onjřej [Jam10] proposes a method to ray-trace isosurfaces represented by distance fields. Similarly, we also traverse distance fields using straight lines to detect collisions with implicit surfaces. In contrast to rendering, however, we perform intersections between line segments and distance fields, as opposed to semi-infinite rays and distance fields as in ray tracing. This difference is substantial because in typical physically based simulations, line segments between consecutive timesteps are usually short and the point positions exhibit a lot of temporal coherence. In addition, instead of using two-level sparse grid blocks as in [Jam10], we accelerate the traversal using an octree hierarchy with multiple levels. Point tree hierarchies have been previously applied to discrete collision detection [BJ08, GSM*12]. We combine hierarchies of points with our distance field octree-based traversal, for fast continuous collision detection.

3. Continuous Collision Detection

We now describe how we perform continuous collision detection between point-sampled objects and signed distance fields. Both the point-sampled object and the distance field object undergo arbitrary rigid body motion. Given a distance field $\phi : \mathbb{R}^3 \to \mathbb{R}$ and a scalar value σ, the isosurface (level set) corresponding to σ is defined as $S_\sigma = \{p \,|\, \phi(p) = \sigma\}$. In contrast to general implicit functions, distance fields also provide the Euclidean distance to the zero isosurface. The penetration depth of a point at time t is determined by transforming the point position at time t into the frame of reference of the distance field object at time t, and looking up the signed distance value. Therefore, we study the trajectory $r(t)$ of the point in the frame of reference of the signed distance field object, for $t_{\min} \le t \le t_{\max}$, where $t_{\min}$ and $t_{\max}$ are the start and end of the timestep, respectively. The task of continuous collision detection is to determine the time(s) when $r(t)$ crosses the isosurface S_σ, for some chosen $\sigma \in \mathbb{R}$. Typically, we will use $\sigma = 0$, but other values of σ will also be useful when combining our algorithm with a bounding volume hierarchy of points.

For general rigid body motion, the trajectory $r(t)$ during the time interval $t \in [t_{\min}, t_{\max}]$ is a cycloide and not a polynomial function of t. As commonly done with continuous collision detection, we assume that it can be reasonably approximated by a line segment, $r(t) = o + (t - t_{\min})d$, where $o \in \mathbb{R}^3$ is the start position and $d \in \mathbb{R}^3$ is the direction, for

$$o = r(t_{\min}), \qquad d = \frac{r(t_{\max}) - r(t_{\min})}{t_{\max} - t_{\min}}. \tag{1}$$

Point continuous collision detection therefore amounts to checking for collisions between a line segment and the isosurface of the signed distance field. Specially, for $t \in [t_{\min}, t_{\max}]$, we want to detect all the roots of $\phi(r(t)) = \sigma$, and identify the subintervals of $[t_{\min}, t_{\max}]$ where $\phi(r(t)) \le \sigma$. Note that for some applications of continuous collision detection, only the first time of contact is needed. We can, however, also detect all the intersecting subintervals, which is needed for our continuous contact force and torque impulse calculation (Section 4).

3.1. Line Segment vs Distance Field Cell Intersection

In practice, the distance field is sampled on a regular three-dimensional grid. The grid partitions the distance field box into uniformly-sized cells whose corners are the grid points. Starting from o, we successively traverse the grid cells along the line segment. We identify the next grid cell using a 3D discrete differential analyzer algorithm [AW*87], similar to Bresenham's algorithm for rasterizing line segments into pixels. This algorithm is fast because it can identify the next cell only with integer arithmetics (avoids floating-point). When visiting a cell, we first check whether the cell contains the isosurface, by comparing the distances at the eight cell corner grid points to σ. If all the values are greater or smaller than σ, we can proceed to the next cell. Otherwise, a test for intersection is performed, and, if the line segment intersects the isosurface, the intersection point is calculated.

We detect the intersection point t_{hit} by finding the roots of $\phi(r(t_{\text{hit}})) = \sigma$. Distance fields are sampled discretely on the grid, and we can approximate the distance function at any location inside the cell using trilinear interpolation of the values at the eight grid cell vertices. We first interpolate the distance values $\phi_{\text{in}} = \phi(r(t_{\text{in}}))$ and $\phi_{\text{out}} = \phi(r(t_{\text{out}}))$ at points where the line segment enters and exits the cell. If the signs of $\phi_{\text{in}} - \sigma$ and $\phi_{\text{out}} - \sigma$ differ, the line segment has crossed the isosurface; otherwise we skip this cell. Because distance data beyond the resolution of each individual cell is not available, we assume that there is a single intersection point inside each cell, and we discover it using bisection, as follows. We first approximate t_{hit} as the time when the line connecting values ϕ_{in} and ϕ_{out} crosses the isosurface σ,

$$t_{\text{hit}} = t_{\text{in}} + (t_{\text{out}} - t_{\text{in}}) \frac{\phi_{\text{in}} - \sigma}{\phi_{\text{in}} - \phi_{\text{out}}}. \quad (2)$$

Next, we select the interval $[t_1, t_2]$ from the two subintervals $[t_{\text{in}}, t_{\text{hit}}]$ and $[t_{\text{hit}}, t_{\text{out}}]$ such that $\phi(r(t_1)) - \sigma$ and $\phi(r(t_2)) - \sigma$ have different signs. We repeat this process until $\phi(r(t_{\text{hit}}))$ has converged to σ, or the maximum number of iterations is exceeded. In our implementation, we set the maximum number of iterations to 5. The bisection is illustrated in Figure 1.

To assemble all the intervals where the line segment intersects the isosurface, we maintain a time list and a sign flag. The sign flag is initialized to the sign of $\phi_{\text{in}} - \sigma$. If it is negative, we add t_{in} to the list. When traversing the cells along the line segment, once an intersection has been found, we update the sign flag and also add t_{hit} to the list. If the sign flag is negative at the end of the line segment, we also add t_{out} to the list. Finally, pairs of successive elements in the list form the colliding subintervals of $[t_{\text{min}}, t_{\text{max}}]$.

3.2. Octree-based Acceleration of the Cell Traversal

The cell traversal is initiated by intersecting the line segment with the bounding box of the distance field.

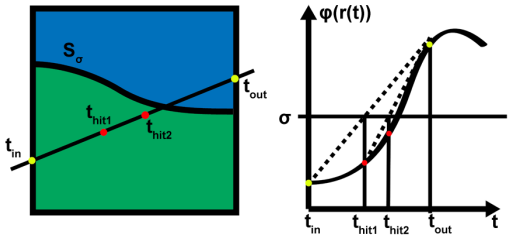

Figure 1: ***Finding the intersection of the line segment and isosurface:*** *Left: line segment passes through a cell that contains the isosurface. Right: finding the root of* $\phi(r(t)) = \sigma$.

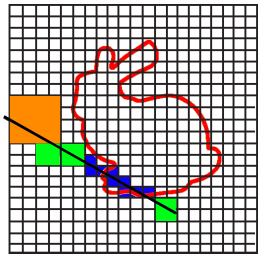

Figure 2: *Octree-based cell traversal.*

If the line segment does not intersect the bounding box, then definitely there will be no intersection with the isosurface. Otherwise, we start the traversal at the cell that contains the line segment's entrance into the distance field bounding box. If the line segment starts inside the grid, we start from the cell that contains the starting point. We terminate the traversal when we exit the bounding box or the end of the line segment is reached. In practice, only a small portion of the cells contains the isosurface and so intersection tests are not needed for the majority of cells. Consecutively, we can skip several cells, e.g., when far away from the surface. We formally exploit this observation using an octree. We precompute a spatial octree datastructure for the entire distance field grid, where each octree node contains two values: the minimum and maximum distance value ϕ_{min} and ϕ_{max} inside the node subtree. The root node stores the minimum and maximum of the entire distance field grid, and we continue partitioning the nodes into 8 octants until reaching the grid cell size. Note that the distance interval $[\phi_{\text{min}}, \phi_{\text{max}}]$ at each node must be a subinterval of its parent node interval. Therefore, in practice, we construct the octree in a bottom-up manner starting from the grid cells and proceeding to the root. When a line segment reaches a new cell, we traverse the octree starting from the root and find the largest block of cells we can safely skip. We can skip all the cells in a subtree if $\sigma \leq \phi_{\text{min}}$ or $\sigma \geq \phi_{\text{max}}$. We analyze the time and memory to compute and store the signed distance fields and the distance field octree in Table 1.

3.3. Intersection Between Point Sphere Trees and Distance Fields

In this section, we discuss how to perform continuous collision detection between a collection of points organized into a sphere hierarchy, and the distance field. At each

	signed distance field		distance field octree	
resolution	time	memory	time	memory
128^3	1.9 sec	8.2 MB	0.01 sec	34.2 MB
256^3	6.8 sec	64.8 MB	0.09 sec	273.6 MB
512^3	45.6 sec	518.0 MB	0.65 sec	2.14 GB
1024^3	347.0 sec	4.1 GB	4.88 sec	17.1 GB

Table 1: *Computation times and memory for the signed distance field and distance field octree. Bunny model.*

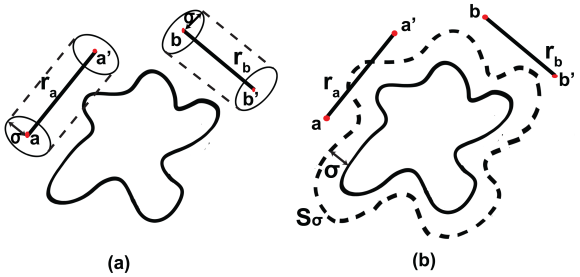

Figure 3: ***Continuous collision detection using a sphere hierarchy:*** *(a) detecting continuous collisions between the bounding sphere and the zero isosurface, (b) computing intersections of the line segment with the isosurface S_σ.*

timestep, naive continuous collision detection could proceed by traversing the points linearly (point by point). The traversal can be greatly accelerated using a bounding volume hierarchy. We use the nested bounding sphere hierarchy presented in [BJ08]. Each node in the hierarchy covers all the points in its subtree. Our algorithm traverses the hierarchy in breadth-first order. For each traversed tree node, it needs to find continuous collisions between a bounding sphere and the zero isosurface (see Figure 3, a). This test is equivalent to forming a line segment originating at the center of the node and checking whether it collides with the isosurface S_σ, where σ equals the radius of the bounding sphere (see Figure 3, b). If there is no collision between the line segment and S_σ, no point in the subtree can collide, and the entire subtree can be skipped. Otherwise, all the children of the node are added to a list for further traversal. Note that for a non-leaf tree node, we do not need to find the intersection intervals; we can terminate the check as soon as we establish that a collision exists.

4. Continuous Penalty Contact

In this section, we give a continuous penalty force model between points and a distance field that can be used in conjunction with our continuous collision detection. Our method was inspired by [TMOT12] who investigated triangle vs triangle contact, whereas we address points and distance fields, and also extend the model to continuous contact torques, as well as damping forces and torques. At any moment of time, we can evaluate the penalty contact force and torque as

$$F = -kdN, \qquad \tau = r \times F, \tag{3}$$

where $k > 0$ is the contact penalty force stiffness, d is the signed distance field value, N is the point's outward normal in the world coordinate system, and r is the torque handle. The handle is typically the vector joining the center of mass and the current point position. The force is non-zero only when $d < 0$, i.e., point is in contact. The penalty force F and torque τ vary continuously during the time interval $[t_{\min}, t_{\max}]$. The net impulse I and angular impulse M are

$$I = \int_{t_{\min}}^{t_{\max}} F(t)dt = -\int_{t_{\min}}^{t_{\max}} kd(t)N(t)dt, \tag{4}$$

$$M = \int_{t_{\min}}^{t_{\max}} \tau(t)dt = -\int_{t_{\min}}^{t_{\max}} r(t) \times \Big(kd(t)N(t)\Big)dt. \tag{5}$$

Since the penetration depth d is 0 when the objects are disjoint, we can split this integral into the contact subintervals,

$$I = \sum_{i=1}^{n} \int_{t_{\min}^i}^{t_{\max}^i} -kd(t)N(t)dt, \tag{6}$$

$$M = -\sum_{i=1}^{n} \int_{t_{\min}^i}^{t_{\max}^i} r(t) \times \Big(kd(t)N(t)\Big)dt. \tag{7}$$

By analogy with Euler integration, the impulse and angular impulse in Equation 6 can also be interpreted as the integral of a constant contact force and torque. Thus, the constant force and torque are simply the time-averages of the continuous penalty forces and torques,

$$F^* = I/(t_{\max} - t_{\min}), \tag{8}$$

$$\tau^* = M/(t_{\max} - t_{\min}). \tag{9}$$

Similarly, we also integrate the damping impulse and angular impulse,

$$I_D = \int_{t_{\min}}^{t_{\max}} F_D(t)dt =$$

$$= \int_{t_{\min}}^{t_{\max}} k_D\Big(v_A(t) + \omega_A(t) \times r_A(t) - v_B(t) - \omega_B(t) \times r_B(t)\Big)dt, \tag{10}$$

$$M_D = \int_{t_{\min}}^{t_{\max}} \tau_D(t)dt = \int_{t_{\min}}^{t_{\max}} r(t) \times F_D(t)dt, \tag{11}$$

where $k_D > 0$ is the damping coefficient, and v and ω are the linear and angular velocity, respectively.

We timestep the rigid bodies using the explicit symplectic Euler integrator [HLW03]. Starting from time $t_{\min}$, we first integrate the rigid object forward to $t_{\max}$ under the contact and damping forces and torques F^* and τ^* computed during the previous timestep that ended at $t_{\min}$. We also add the other external forces and torques such as user forces and gravity. This produces the position and orientation of the object at time $t_{\max}$. We then execute continuous collision detection for the linear trajectory between $t_{\min}$ and $t_{\max}$ and integrate the impulses I and M using Equations 6 and 7. Using Equations 8 and 9, we then obtain the contact force F^*

and torque τ^* to be applied during the next timestep starting at $t_{\max}$. The time-varying normals and handles $N(t)$ and $r(t)$ can be obtained either by linear interpolation of the corresponding values at $t_{\min}$ and $t_{\max}$ (constant velocity during the timestep) or by an Euler step of the rigid body dynamics forward from $t_{\min}$ with timestep $t - t_{\min}$ (constant acceleration). Because continuous collision detection dominates the timestep computation time, such Euler substepping of the rigid body does not introduce a significant overhead. We evaluate all integrals numerically using the midpoint rule, with $N = 20$ subintervals.

5. Results

Our experiments were performed on an Intel Xeon 2.9GHz CPU (2x8 cores) machine with 32GB RAM, and an GeForce GTX 680 graphics card with 2GB RAM. We computed the signed distance fields using an octree-based method using 8 threads [XB14]. Table 1 gives the computation times and memory sizes for the signed distance field and the distance field octree for the bunny model (777 vertices). The memory sizes of the signed distance field and distance field octree, as well as the distance field octree computation time only depend on the resolution, and scale linearly with resolution. The computation time of the signed distance field depends on the complexity of the surface mesh, in addition to resolution. In our experiments, we always precompute the signed distance field and load it from a disk file. We construct the distance field octree at runtime, which only imposes a small additional computational overhead (Table 1).

All of our objects carry a pointshell and a signed distance field, and the continuous collision detection and contact computation is performed twice, with each object assuming either role. The points are either set to the vertices of the object, or precomputed using particle repulsion [BJ08]. In the first experiment, we randomly sample 10^6 pairs of points in the distance field box and generate line segments between them. For each line segment, we execute continuous collision detection against the zero isosurface of the signed distance field of the bunny (Figure 4). We compute the signed distance fields at four different resolutions: $128^3, 256^3, 512^3, 1024^3$. In Table 2, we compare the performance between uniform grid traversal and octree-accelerated grid traversal. The results demonstrate that the octree-accelerated grid traversal reduces the continuous collision detection computation time by $13\times$ for high-resolution signed distance fields. With higher resolution, the computational cost increases almost linearly with resolution for uniform grid traversal, whereas octree-accelerated grid traversal time increase much more slowly. Therefore, the speedup becomes more significant as the resolution of signed distance field increases.

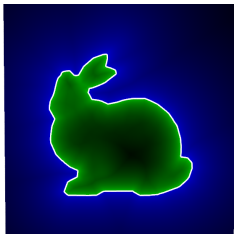

Figure 4: *Signed distance field slice (bunny).*

resolution	uniform grid traversal	octree-accelerated
128^3	12.7 sec	12.0 sec
256^3	38.8 sec	13.1 sec
512^3	97.2 sec	15.4 sec
1024^3	274.3 sec	21.4 sec

Table 2: *Computation times using uniform and octree-accelerated grid traversal. Bunny model.*

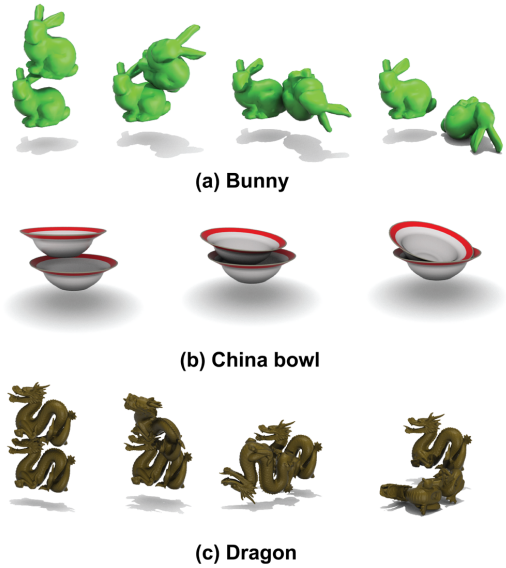

Figure 5: *Simulation of bunnies, china bowls and dragons using continuous collision detection and continuous contact resolution.*

In the second experiment, we compute 1024^3 signed distance fields for the bunny, china bowl and dragon models. In the $3\times$ expanded space of the distance field box, we sampled $1,000$ pairs of random position and orientations for the point-sampled object. We then perform continuous collision detection against the zero isosurface. We compare the performance of using a sphere tree hierarchy on points tree versus linear (point-by-point) traversal. Table 3 shows that point tree traversal gives a $3\times$-$75\times$ speedup over the point-by-point collision detection.

In the third experiment, we applied our continuous collision detection and contact to rigid body simulations. We drop one object onto another fixed object and compare the performance of point-by-point continuous collision detection versus using point tree. We resolve the collisions between the objects as well as against the static ground plane. We report the time for continuous collision detection between the objects. The ground is represented as a simple implicit function

model	#points	linear	hierarchy
bunny	777	1.8 sec	0.7 sec
china bowl	2,072	3.4 sec	0.3 sec
dragon	437,645	761.2 sec	9.9 sec

Table 3: *Continuous collision detection computation times for synthetic randomly sampled rigid body configurations. We compare linear (point-by-point) traversal vs using a sphere point hierarchy.*

model	#points	#frames	linear	hierarchy
bunny	777	2,600	8 sec	2.5 sec
china bowl	2,072	2,000	20.3 sec	4.5 sec
dragon	437,645	1,160	671.6 sec	2.6 sec

Table 4: *Continuous collision detection computation times during a physically based simulation. We compare linear (point-by-point) traversal vs using a sphere point hierarchy.*

$\phi(p) = p_y$, where p_y is the y coordinate of point p. Figure 5 gives the simulation results for the bunny, china bowl, and dragon simulations. Table 4 shows that using the point-tree traversal, we can accelerate continuous collision detection by $3\times$-$250\times$. The acceleration becomes more significant when the number of points increases. Note that in physically based simulations, objects typically do not overlap with each other because the contact response separates them. In the synthetic random-sampling strategy (Table 3), however, objects may overlap severely, with most of the points in contact, which is a situation where a tree hierarchy cannot help much. Therefore, the speedup of using the point tree in physically based simulation is typically much larger than in the synthetic case.

In the fourth experiment, we compare the stability of our method to discrete collision detection with discrete forces and torques (Figure 6). Our method uses continuous collision detection with continuous forces and torques. For a fixed stiffness level, the maximum stable timestep is $3\times$ larger in our method. We then fixed this timestep, and decreased the stiffness of the discrete method until it became stable. This caused a $2.75\times$ larger maximum penetration depth for the discrete method compared to our method. We also made a comparison between the maximum stable timestep for both methods, under the same stiffness and matching the maximum penetration depth. Our results demonstrate that our continuous method has a $2.6\times$ larger maximum stable timestep.

6. Conclusion

We presented an efficient algorithm for continuous collision detection between points and distance fields. We described a method for computing the intersection between a line segment and an implicit surface defined by a signed distance field. We also demonstrated how to apply two acceleration techniques: octree-based grid traversal and the point-tree. We integrated our method with continuous penalty-based contact, and successfully applied it to rigid body simulation. Although our octree requires additional memory, it can be computed quickly and substantially accelerates continuous collision detection. Our method suffers from the general limitation of point-sampled collision detection, namely the possibility that unsampled sharp features may cause deep penetrations. In the future, we would like to use adaptive distance fields to save the memory. Another extension would be to apply our algorithm to non-linear (polynomial) point trajectories, especially for rigid-body simulation where the linear trajectory assumption may not hold for large timesteps. We would also like to simulate deformable distance fields, dynamically update the precomputed octree data structure, and apply our method to haptic rendering.

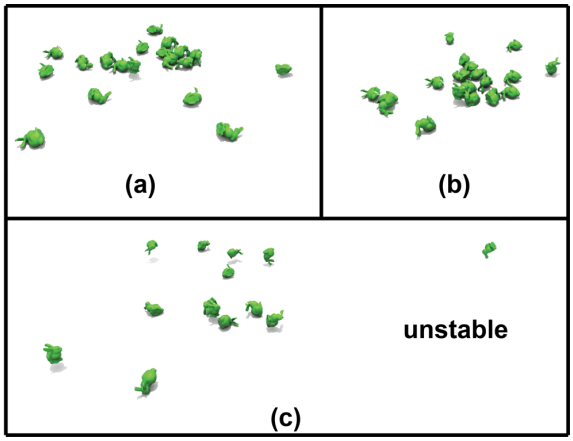

Figure 6: ***Stability comparison.*** *(a) continuous collision detection with continuous forces and torques at $3\times$ timestep; (b) discrete collision detection with discrete forces and torques at $1\times$ timestep; (c) discrete collision detection with discrete forces and torques at $3\times$ timestep (explosion).*

Acknowledgements: This research was sponsored in part by the National Science Foundation (CAREER-53-4509-6600, IIS-1422869), Sloan Foundation, USC Annenberg Graduate Fellowship to Hongyi Xu, and a donation of two workstations by the Intel Corporation.

References

[AW*87] AMANATIDES J., WOO A., ET AL.: A fast voxel traversal algorithm for ray tracing. In *Proceedings of EUROGRAPHICS* (1987), vol. 87, pp. 3–10. 3

[BB09] BROCHU T., BRIDSON R.: *Numerically robust continuous collision detection for dynamic explicit surfaces*. Tech. rep., University of British Columbia, 2009. 2

[BEB12] BROCHU T., EDWARDS E., BRIDSON R.: Efficient geometrically exact continuous collision detection. *ACM Trans. Graph. 31*, 4 (2012), 96:1–96:7. 2

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. *ACM Trans. on Graphics 21*, 3 (2002), 594–603. 2

[BJ08] BARBIČ J., JAMES D. L.: Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics 1*, 1 (2008), 39–52. 1, 2, 4, 5

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of Clothing with Folds and Wrinkles. In *Proc. of the Symp. on Comp. Animation 2003* (2003). 2

[FSG03] FUHRMANN A., SOBOTKA G., GROSS C.: Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon* (2003), pp. 58–65. 2

[GSM*12] GLONDU L., SCHVARTZMAN S. C., MARCHAL M., DUMONT G., OTADUY M. A.: Efficient collision detection for brittle fracture. In *Proc. of the Symp. on Comp. Animation 2012* (2012), Eurographics Association, pp. 285–294. 2

[HLW03] HAIRER E., LUBICH C., WANNER G.: Geometric numerical integration illustrated by the störmer–verlet method. *Acta Numerica 12* (2003), 399–450. 4

[HXB14] HONGYI XU Y. Z., BARBIČ J.: Implicit multibody penalty-based distributed contact. *IEEE Transactions on Visualization and Computer Graphics 20*, 9 (2014). 2

[Jam10] JAMRIŠKA O.: Interactive ray tracing of distance fields. *The 14th Central European Seminar on Computer Graphics* (2010). 2

[JBS06] JONES M., BÆRENTZEN J., SRAMEK M.: 3D distance fields: a survey of techniques and applications. *IEEE Trans. on Visualization and Computer Graphics 12*, 4 (2006), 581–599. 1

[MKW*04] MARMITT G., KLEER A., WALD I., FRIEDRICH H., SLUSALLEK P.: Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing. In *Proc. of Virtual Reality, Modeling, and Visualization* (2004), vol. 4, pp. 429–435. 2

[NMHW02] NEUBAUER A., MROZ L., HAUSER H., WEGENKITTL R.: Cell-based first-hit ray casting. In *Proc. of the Symposium on Data Visualisation* (2002), Eurographics Association, pp. 77–ff. 2

[Pro97] PROVOT X.: Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface* (1997), pp. 177–189. 2

[PSL*98] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.-P.: Interactive ray tracing for isosurface rendering. In *Proceedings of the conference on Visualization'98* (1998), IEEE Computer Society Press, pp. 233–238. 2

[RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies. In *Computer graphics forum* (2002), vol. 21, pp. 279–287. 2

[RKLM05] REDON S., KIM Y. J., LIN M. C., MANOCHA D.: Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering 5*, 2 (2005), 126–137. 2

[SWF*93] SNYDER J. M., WOODBURY A. R., FLEISCHER K., CURRIN B., BARR A. H.: Interval methods for multi-point collision between time-dependent curved surfaces. In *Proc. of ACM SIGGRAPH 93* (1993), pp. 321–334. 2

[TCYM09] TANG M., CURTIS S., YOON S.-E., MANOCHA D.: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Trans. on Visualization and Computer Graphics 15* (2009), 544–557. 2

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M., FAURE F., MAGNENAT-THALMANN N., STRASSER W., VOLINO P.: Collision Detection for Deformable Objects. *Computer Graphics Forum 24*, 1 (2005), 61–81. 1, 2

[TMOT12] TANG M., MANOCHA D., OTADUY M. A., TONG R.: Continuous penalty forces. *ACM Trans. Graph. 31*, 4 (2012), 107:1–107:9. 1, 2, 4

[TMT10] TANG M., MANOCHA D., TONG R.: Fast continuous collision detection using deforming non-penetration filters. In *Proc. ACM Symp. on Interactive 3D Graphics and Games (I3D)* (2010), pp. 7–13. 2

[Wan14] WANG H.: Defending continuous collision detection against errors. *ACM Trans. on Graphics (SIGGRAPH 2014) 33*, 4 (2014). 2

[XB14] XU H., BARBIČ J.: Signed distance fields for polygon soup meshes. In *Proc. of the Graphics Interface Conference* (2014), pp. 35–41. 5