6-DoF Haptic Rendering using Continuous Collision Detection Between Points and Signed Distance Fields

Hongyi Xu, *Member, IEEE,* Jernej Barbič, *Member, IEEE,*

Abstract—We present an algorithm for fast continuous collision detection between points and signed distance fields, and demonstrate how to robustly use it for 6-DoF haptic rendering of contact between objects with complex geometry. Continuous collision detection is often needed in computer animation, haptics and virtual reality applications, but has so far only been investigated for polygon (triangular) geometry representations. We demonstrate how to robustly and continuously detect intersections between points and level sets of the signed distance field. We suggest using an octree subdivision of the distance field for fast traversal of distance field cells. We also give a method to resolve continuous collisions between point clouds organized into a tree hierarchy and a signed distance field, enabling rendering of contact between rigid objects with complex geometry. We investigate and compare two 6-DoF haptic rendering methods now applicable to point-vs-distance field contact for the first time: continuous integration of penalty forces, and a constraint-based method. An experimental comparison to discrete collision detection demonstrates that the continuous method is more robust and can correctly resolve collisions even under high velocities and during complex contact.

Index Terms—haptics, 6-DoF, continuous collision detection, signed distance fields, contact, constraints, penalty forces

1 Introduction

HAPTIC rendering of contact between complex geometry can be efficiently performed by querying a point-cloud against an implicit function, such as a signed distance field. Such methods are commonly used in industrial practice; a common example is the Voxmap PointShell (VPS) method and its improvements [1]–[6]. To the best of our knowledge, all previous methods for point-cloud vs implicit function haptic rendering only investigated *discrete* collision detection, and are as such prone to interpenetrations, or even tunneling. In this paper, we demonstrate how to perform fast *continuous* collision detection which detects the exact time of contact between two objects within two successive time steps. Our continuous collision detection algorithm is applied to 6-DoF haptic rendering of contact between point-clouds and signed distance fields, making the simulations robust against high velocities and tunneling.

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 and haptics [7], 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 [8].

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, bringing the benefits of continuous collision detection to such simulations. 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 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 [4]) with the fast grid cell traversal, enabling contact between two rigid objects with complex geometry. Our algorithm 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 two haptic rendering methods for contact between rigid objects with complex geometry. These methods were previously designed for triangle mesh contact, and our work makes it possible to apply them to contact between point clouds and distance fields. Inspired by the continuous penalty force model [9], we present a penalty contact resolution method to match our continuous collision detection algorithm. In addition to forces [9], our method also computes continuous penalty torques and damping forces. The second method that we use and experimentally evaluate is a

• *Both authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA, 90089.*
E-mail: hongyixu@usc.edu, jnb@usc.edu.

constrained method, similar to [10], but adapted to points and distance fields. Our experiments show that both methods greatly benefit from continuous collision detection, all the while still maintaining a high simulation update rate. Both methods benefit from improved stability, and for the constrained method, also absence of penetrations and tunneling.

Our contributions include:

- an efficient algorithm for continuous collision detection between points and signed distance fields,
- exact continuous test between a trilinearly interpolated implicit function and a line segment,
- fast grid traversal using a spatial octree subdivision of the distance field,
- point-tree traversal algorithm for continuous collision culling of contact between two rigid objects with complex geometry,
- continuous penalty contact and damping model to be employed in concert with our continuous collision detection,
- an experimental evaluation of a continuous constraint method and a continuous penalty-based method, in 6-DoF haptic rendering applications involving complex geometry and distributed contact.

2 Related Work

Numerous approaches have been investigated to detect collisions between interfering objects; see, for example, the survey [8]. 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 [11]–[15]. 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 [16], [17]. The equations become cubic when linearly interpolating vertex motion [9], [11], [18]. Continuous collision between such pairs of primitives is sensitive to numerical error and the employed tolerances, requiring special care [19], [20]. 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 [21], [22] and rigid-deformable simulations [4], [23], [24]. 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 [25]–[27]. Specially, Onjřej [28] 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 [28], we accelerate the traversal using an octree hierarchy with multiple levels. Point tree hierarchies have been previously applied to discrete collision detection [4], [29]–[31]. We combine hierarchies of points with our distance field octree-based traversal, for fast continuous collision detection.

Haptic rendering has been an active area of research over the last decade [32]. Contact resolution methods in haptic rendering can be loosely categorized into penalty-based methods [1], [3], [4], [6], [33], [34] and constraint-based methods [10], [31], [35]–[38]. Penalty-based methods resolve contacts by the use of elastic repulsive forces. They are a simple, efficient and popular method in haptic rendering. However, with only discrete collision detection, penalty methods can result in non-smooth contact forces and torques when the contact stiffness is set to a high value. Continuous penalty method [9] alleviates this problem by performing continuous collision detection and integrating the contact impulses over the timestep interval. Constraint-based methods model collision response as a constrained optimization, typically either using velocity as the solution variable [36], [38] or using acceleration as target for quasi-static simulation [10]. Although the contact constraints do not permit penetration at the detected contact sites, continuous collision detection is still necessary to ensure a completely interpenetration-free trajectory due to numerical integration [39]. Ortega et al. [10] integrates these ideas together by first performing an explicit Euler integration with the optimized 6-DoF constrained acceleration and then using triangle mesh-based continuous collision detection to stop the God object at the first contact site. We apply our continuous collision detection between points and signed distance fields to a continuous penalty method (similar to [9]) and a constraint-based method (similar to [10]). We compare the methods in terms of speed, penetration, reliability and robustness.

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 $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$ and a scalar value σ, the isosurface (level set) corresponding to σ is defined as $S_\sigma = \{p \,|\, \phi(p) = \sigma\}$. 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 $t_{\min} \leq t \leq t_{\max}$, 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

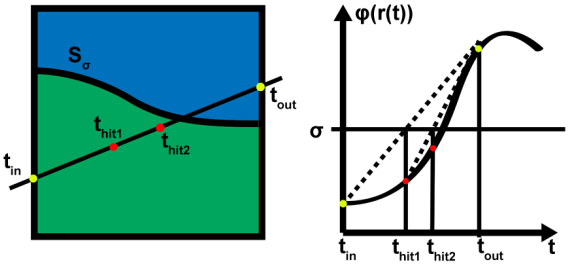

Fig. 1: ***Finding the intersection of the line segment and isosurface using bisection:*** *Left: line segment passes through a cell that contains the isosurface. Right: finding the root of* $\phi(r(t)) = \sigma$.

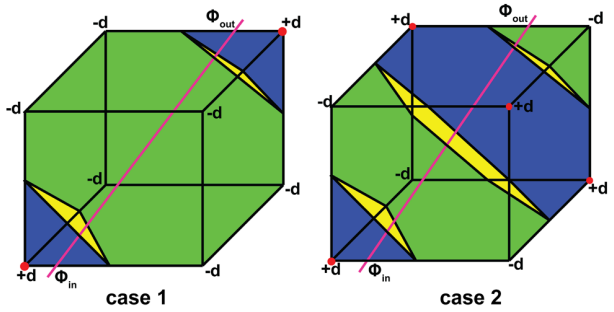

Fig. 2: ***Bisection method failures:*** *Assuming* $\sigma = 0$*, we give two examples where the bisection method either misses a contact, or returns an incorrect first contact. (a) Both* ϕ_{in} *and* ϕ_{out} *have positive signs and the bisection method will directly skip this cell and miss the contacts. (b) The bisection method will return the second contact site instead of the first one and even give the wrong sign flag for the line segment traversal. The exact cubic polynomial method does not suffer from these problems.*

assume that the trajectory $r(t)$ can be reasonably approximated by a line segment, $r(t) = r(t_{\min}) + (t - t_{\min})d$, where $d \in \mathbb{R}^3$ is the normalized direction from $r(t_{\min})$ to $r(t_{\max})$. 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 $\phi(r(t)) \leq \sigma$. 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 forces and torques (Section 4.1).

3.1 Line Segment vs Distance Field Cell Intersection

In our work, we use the uniform-grid distance field. 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 [40], 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 operations). We can proceed to the next cell if the distances at the eight cell corner grid points are all greater or smaller than σ. 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 for any query point p inside the cell using trilinear interpolation of the values at the eight grid cell vertices as $\phi(p) = \phi(w_x, w_y, w_z)$, where w_x, w_y, w_z are the barycentric weights at p. We skip the cell if the distance values ϕ_{in} and ϕ_{out} at the entering and exiting points are both larger or smaller than σ. Otherwise, for the intersection point $r(t_{\text{hit}})$, the barycentric weights can be computed by linearly interpolating the weights of entering and exiting points as,

$$(w_x^{\text{hit}}, w_y^{\text{hit}}, w_z^{\text{hit}}) = (1-\beta)(w_x^{\text{in}}, w_y^{\text{in}}, w_z^{\text{in}}) + \beta(w_x^{\text{out}}, w_y^{\text{out}}, w_z^{\text{out}}), \quad (1)$$

for $\beta \in [0,1]$. Therefore, we get a cubic equation $\phi(w_x(\beta), w_y(\beta), w_z(\beta)) = \sigma$ for the unknown $\beta \in [0,1]$. We calculate the smallest real root on the $[0,1]$ interval using Cardano's formulas [41]. The intersection time is then $t_{\text{hit}} = (1-\beta)t_{\text{in}} + \beta t_{\text{out}}$. This method gives the exact analytical first point of intersection, at the expense of forming the coefficients of the cubic polynomial and applying Cardano's formulas.

Alternatively, one can also approximately obtain the intersection point using bisection. Because distances beyond the resolution of each individual cell are not available, we assume that there is a single intersection inside each cell. 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 (5 in our implementation) is exceeded. The bisection is illustrated in Figure 1. In our experiments, the bisection method is about $1.2\times$ faster than the cubic polynomial exact method, but can miss collisions. The exact method guarantees that no contact is missed and returns the exact solution for the first contact (if any). In Figure 2, we give two concrete examples where bisection fails to detect the correct first contact. After obtaining all the intersecting points, we assemble all the colliding subintervals of $[t_{\min}, t_{\max}]$.

3.2 Octree-based Acceleration of the Cell Traversal

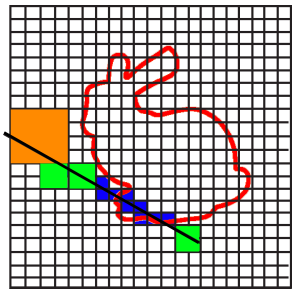

Fig. 3: *Octree cell traversal.*

Given a line segment, we now need to traverse the distance field cells in the search for the first contact (if any). If the line segment starts or ends outside of the bounding box of the distance field, it is clipped to the box. 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 the minimum distance value $\phi_{\min}$ inside the node subtree. The root node stores

	signed distance field		distance field octree	
resolution	time	memory	time	memory
128^3	1.9 sec	8.2 MB	0.01 sec	16.1 MB
256^3	6.8 sec	64.8 MB	0.09 sec	136.8 MB
512^3	45.6 sec	518.0 MB	0.65 sec	1.07 GB
1024^3	347.0 sec	4.1 GB	4.88 sec	8.55 GB

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

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.*

the minimum of the entire distance field grid, and we continue partitioning the nodes into 8 octants until reaching the grid cell size. Note that the minimum distance value $\phi_{\min}$ must be larger than the value stored in its parent node. 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. If the current sign flag is positive, then we skip all the cells in a subtree if $\sigma \leq \phi_{\min}$. Note that potentially we can also store $\phi_{\max}$ in each node and skip the cells if $\sigma \geq \phi_{\max}$, when the sign flag is negative. However, in practice, a point is seldom allowed to penetrate deeply into the object, but instead stays close to the surface, where knowing $\phi_{\max}$ does not help. Therefore, in haptic rendering applications, we choose to preserve memory and do not store $\phi_{\max}$. We analyze the time and memory to compute and store the signed distance fields and the distance field octree constructed from the signed distance field in Table 1.

To evaluate the performance speedup from octree, 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 at four different resolutions: $128^3, 256^3, 512^3, 1024^3$. Table 2 shows that with higher resolution, the computational cost increases almost linearly with resolution for uniform grid traversal, whereas octree-accelerated grid traversal time increase more slowly. Therefore, our speedup increases from $1.1\times$ to $12.8\times$ as the distance field resolution increases.

3.3 Intersection Between Point Sphere-Trees and Distance Fields

In this section, we explain how to perform continuous collision detection between a collection of points organized into a sphere hierarchy, and the distance field. At each 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 [4]. 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

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.*

continuous collisions between a bounding sphere and the zero isosurface (see Figure 4, 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 4, 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.

To verify the performance gain, 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. Table 3 shows that point tree traversal gives a $3\times$-$75\times$ speedup over the point-by-point collision detection.

4 6-DoF Haptic Rendering

In this section, we give two 6-DoF haptic rendering methods that use our continuous collision detection between signed distance fields and points: a continuous penalty-based method and a constraint-based method.

4.1 Continuous Penalty-based Haptic Rendering

We first give a continuous penalty force between points and a distance field that can be used in conjunction with our continuous collision detection. Our method was inspired by [9] 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. The penalty force F and torque τ vary continuously during the time interval $[t_{\min}, t_{\max}]$. The force is non-zero only when $d < 0$, i.e., point is in contact. We integrate the net impulse I and angular impulse M as

$$I = \int_{t_{\min}}^{t_{\max}} F(t)dt = \sum_{i=1}^{n} \int_{t^i_{\min}}^{t^i_{\max}} -kd(t)N(t)dt, \quad (3)$$

$$M = \int_{t_{\min}}^{t_{\max}} \tau(t)dt = -\sum_{i=1}^{n} \int_{t^i_{\min}}^{t^i_{\max}} r(t) \times \Big(kd(t)N(t)\Big)dt, \quad (4)$$

where $k > 0$ is the contact penalty force stiffness, $d(t)$ is the penetration depth, $N(t)$ is the point's outward normal in the world coordinate system, $r(t)$ is the torque handle at time t and $[t^i_{\min}, t^i_{\max}]$ is the ith contact subinterval. The handle is typically the vector joining the center of mass and the current point position. Note that we only get the penetration depth d from the distance field, while the force direction comes from the normal of the pointshell

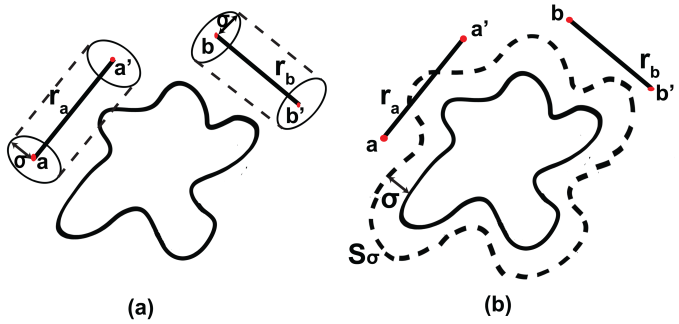

Fig. 4: ***Continuous collision detection using a sphere hierarchy:*** *(a) detecting continuous collisions between the bounding sphere and the zero isosurface, (b) computing intersections of the line segment with the isosurface S_σ.*

object N. Therefore the contact force does not suffer from force discontinuity even when the point crosses the object's interior medial axis. By analogy with Euler integration, the impulse and angular impulse in Equations 3 and 4 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,

$$F^* = I/(t_{\max} - t_{\min}), \quad \tau^* = M/(t_{\max} - t_{\min}). \tag{5}$$

Similarly, we also integrate the damping impulse and angular impulse and compute the averaged damping force and torque.

Starting from time $t_{\min}$, we first integrate the rigid object forward to $t_{\max}$ under the forces and torques from the previous step. 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 3 and 4. Using Equations 5, 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 $N = 5$ subintervals.

We now apply our continuous penalty method to 6-DoF haptic rendering. In haptic rendering, the penalty forces and torques are typically not rendered directly to the user (a.k.a, "direct rendering") [42], [43]. Instead, we connect the simulation position of the haptic object and the position imposed by the haptic manipulandum with a 6-DoF spring (virtual coupling [44], [45]). The virtual coupling force F_{VC} is designed to be linear in displacement between manipulandum position and simulation position and saturate to some maximum value F_{VC}^{max}, once displacement reaches a certain value x_{max} [33], [46]. The x_{max} typically corresponds to shallow penetrations, such as half a voxel, and we have $F_{VC}^{max} = kx_{max}$. Virtual coupling helps decrease penetrations, improves the simulation stability and enables a better control of the rendered stiffness. We use admittance mode of our Haption haptic device, and implicit integration [21], [33] for improved haptic rendering stability. At each haptic cycle, we read the forces and torques from the device. We use them to implicitly timestep rigid body dynamics of both the manipulandum and the simulation object, subject to the virtual coupling and the contact forces and torques, and their 6×6 gradients [21], [33]. We calculate the contact force and torque gradients [21] at the beginning of the timestep. The new position and velocity of the manipulandum object are sent to the device. Note that the timestep in haptic simulations is very short ($0.001s$ in our work), and the point can therefore only move at most a few distance field cells in one timestep. This effect somewhat negates the gain of employing the octree acceleration strategy. Therefore, in contrast to simulations running at graphics rates, in haptic rendering applications we prefer to keep only the leaf nodes of the distance field octree, i.e., the minimum distance inside each distance field cell. At run-time, we trace the point trajectory cell by cell. Such a strategy saves about 50% of the octree memory cost, while still accelerating the cell traversal about $1.5\times$. Using the full octree hierarchy for haptic rendering slows down the performance by $2-3\times$ in our examples. This is due to the extra computations to determine the maximum block that the cell traversal can skip. In most cases at haptic rates, only a single grid cell can be skipped.

4.2 Constraint-based Haptic Rendering

We also use our continuous collision detection in a 6-DoF constraint-based method for haptic rendering of rigid bodies, by building upon Ortega's method [10]. Instead of modeling the rigid bodies as triangle surface meshes, we model one object as a pointshell object and represent the other one implicitly using a signed distance field, similar to [4]. The haptic device object (also called the manipulandum object) is coupled with a "God" object that is constrained to stay on the surface of the other object. We run constraint-based quasi-static haptic simulation, implying the velocity of the God object is zero at all times. At each haptic cycle, given the current God object configuration $x_s \in \mathbb{R}^6$ and the manipulandum object configuration $x_h \in \mathbb{R}^6$ read from the device, we first compute an unconstrained acceleration $a^u \in \mathbb{R}^6$ for the God object as

$$a^u = k_s(x_h - x_s), \tag{6}$$

where k_s is a coupling constant (we use $k_s = 0.5$ in our examples). The acceleration a^c of the God object has to follow the non-penetration constraint on each contact point k, $1 \le k \le m$:

$$a_G^T n_k + \alpha^T (r_k \times n_k) \le 0, \tag{7}$$

where m is the number of contact points, and n_k and r_k are the inward contact normal and handle of the contact point k. Quantity $a^c = (a_G, \alpha)$ consists of the linear acceleration a_G and the angular acceleration α. We concatenate the m constraints into a matrix form as $Ja \le 0$, for matrix $J \in \mathbb{R}^{m\times 6}$. Note that different from penalty methods which always use the normals of the points, here we choose to use the normals of the static object which can be modeled either as distance field or pointshell (Figure 5). If the static object is modeled as a distance field, then the contact normal will just be the normalized gradient of the distance field. The discontinuity of the gradient at the medial axis can cause stability problem for penalty-based methods. However, since penetrations are not allowed in the constraint-based method, the gradient discontinuity is not an issue. The constrained acceleration is obtained by minimizing the kinetic distance $\|a^c - a^u\|_M$ between the constrained acceleration a^c and unconstrained acceleration a^u over the feasible set $\{a \mid Ja \le 0\}$, where $M \in \mathbb{R}^{6\times 6}$ is a block diagonal

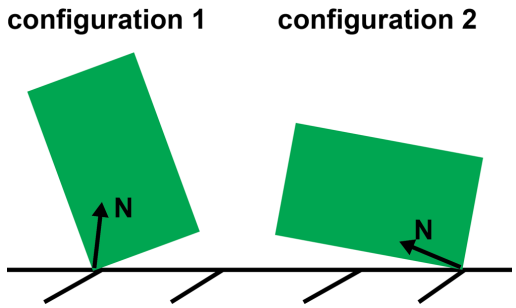

Fig. 5: ***Normal selection.*** *If we choose to use the normals of the pointshell (here the green peg), then the rotated contact normal at configuration 2 can cause the object to penetrate into the ground. Using the contact normal of the static object, i.e., the ground, prevents the penetration.*

matrix with the mass and inertia matrix blocks. We solve the quadratic programming problem using the library qpOASES [47]. Next, the tentative final configuration is obtained by one explicit symplectic Euler step [48] of the God object using the constrained acceleration. We then perform our continuous collision detection between the pointshell and the distance object. If the trajectory is free of new contacts, we move the God object to the final configuration. However, if continuous collision detection detects contact, we only move the God object to the earliest contact time t_c. This algorithm is theoretically correct if the computation was performed with infinite precision. With floating point arithmetic, however, special care needs to be taken. For example, if a block is in contact with a plane, then, the block can never slide tangentially on the plane, as it is in contact already at $t_c = 0s$. The algorithm will always find contact at $t_c = 0$ and the God object will get stuck in the contact position forever. The reference [10] does not specify how such an issue is remedied. Our solution is as follows. When we perform continuous collision detection, we ignore all the contact points detected in the last timestep. After we move the God object to t_c, we perform discrete collision detection on the disabled points. If they are still in contact, we add them to the contact set. We then render the constraint force and torque $F^c = k_h M(a^c - a^u)$ to the haptic device.

The haptic rendering requires high force update rates, typically $1,000$ Hz. However, the continuous collision detection and the quadratic programming solver can potentially cause a lower update rate. Therefore, similarly to [10], we divide our algorithm in two asynchronous loops. We run our examples in admittance mode. In the haptic loop, we read the device force and torque and combine it with the constraint force and torque $-F^c$ from the simulation cycle to compute the new manipulandum position and velocity based on symplectic Euler integration and then send them to the device. The haptic loop is executed at a fixed rate of $1,000$ Hz. The simulation cycle executes the QP solver and continuous collision detection, and sends the computed constraint force and torque to the haptic loop. The frequency of the simulation loop depends on the complexity of the models and the contact configuration; in practice, it stays well above $1,000$ Hz in our examples.

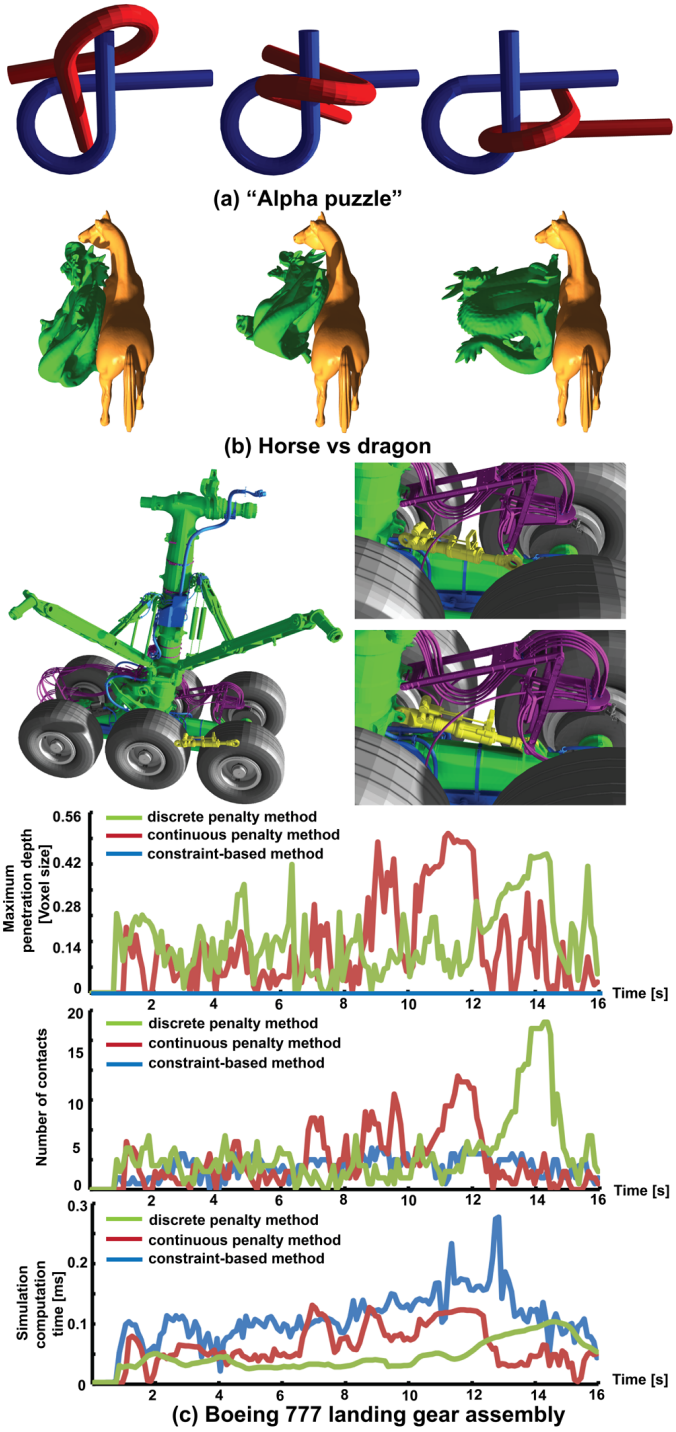

Fig. 6: ***Haptic simulation results.*** *Both continuous penalty-based and constraint-based haptics rendering work on all of these examples. Here we only present the results using the constraint-based method. The continuous penalty method produces similar results. The three curves of each color correspond to the same manipulandum trajectory.*

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 [49]. 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

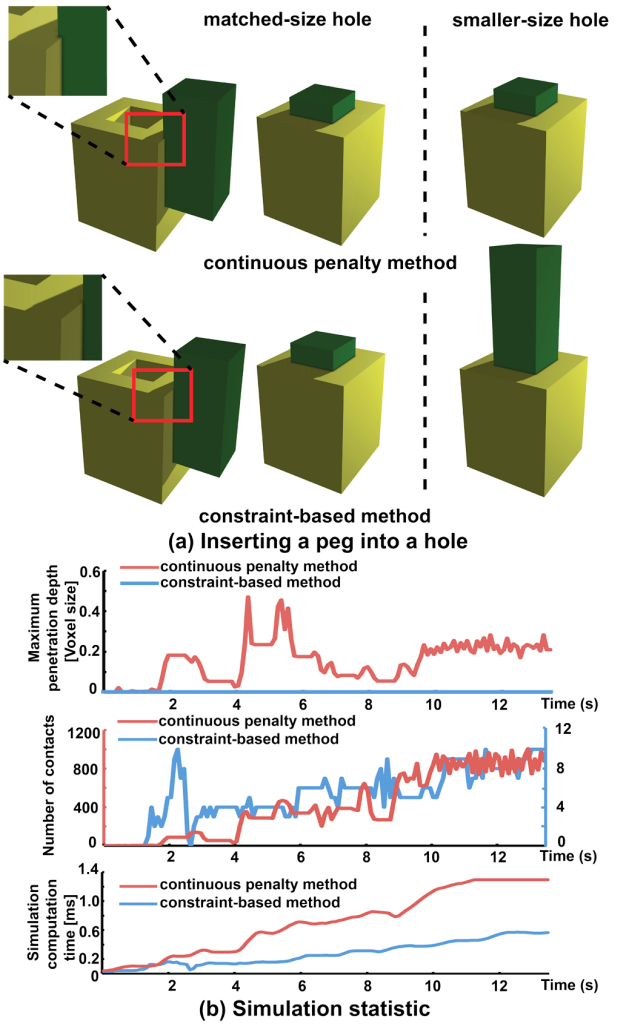

Fig. 7: ***Peg insertion.*** *(a) When the size of the peg and hole matches, both continuous penalty method and constraint-based method successfully inserts the peg into a size-matched hole. We intentionally decrease the hole size by offsetting the distance field with* 0.1 *voxel size. Constraint-based method reflects the ground truth that the peg cannot be inserted into the hole. However, continuous penalty method can still insert the peg in with deeper penetrations. (b) Continuous penalty method produces many more contacts and performs slower than constraint-based method.*

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). The points are either set to the vertices of the object, or precomputed using particle repulsion [4].

We select the distance field resolution by considering both performance and quality. The signed distances for arbitrary query locations can be approximated in $O(1)$ time, independently of the geometric complexity and the distance field resolution. However, our continuous collision detection algorithm still scales sublinearly with the distance field resolution, due to the differences in grid traversal (Table 2). High-resolution distance fields lead to high memory costs and longer distance field computation times (Table 1). However, low-resolution distance fields may not represent

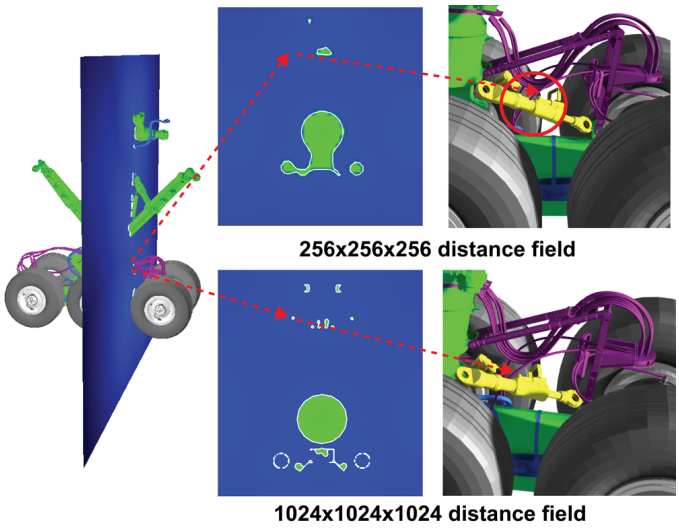

Fig. 8: ***Penetrations due to low-resolution distance field.*** *Thin wires can not be represented by the* 256^3 *distance field and therefore the hydraulic actuator tunnels through the wires. The higher-resolution* $(1,024^3)$ *distance field presents the wire geometry and prevents the tunneling artifacts.*

small or thin geometric details, leading to penetrations or even incorrect results (Figure 8). A tradeoff resolution should therefore be selected in practice.

Haptic Rendering. We also demonstrate our continuous collision detection algorithm by applying it to haptic rendering, investigating both constraint-based and penalty-based methods (Figure 6). Our haptic device is the 6-DoF Haption Virtuose 6D, capable of rendering both forces and torques. In our first example, we present the classic "alpha" path planning puzzle [50], [51]. The red and blue alphas have identical shapes. The blue alpha (distance field object, $512 \times 512 \times 512$) is fixed in space while the red one (pointshell object, $25,269$ sample points) is manipulated to position inside the loop of blue alpha. Our haptic simulator correctly finds the path to resolve the puzzle (Figure 6 (a)). The second example involves one static horse model (distance field object, $256 \times 256 \times 256$) and one movable dragon model (pointshell object, $7,537$ sample points). Our simulator permits smooth sliding over the surface and provides high-quality haptic display of the geometric details (Figure 6 (b)).

In our third example, we perform a virtual assembly task using the Boeing 777 landing gear model. The pointshell object is a hydraulic actuator ($3,426$ sample points). The goal is to find the path to install the hydraulic actuator into the bottom of the static landing gear (distance field object, $1024 \times 1024 \times 1024$) (Figure 6 (c)). The constraint-based method with our continuous collision detection guarantees an interpenetration-free path, even in the presence of thin features only 2 voxels wide. Due to virtual coupling saturation, both the discrete and continuous penalty methods remained limited to a shallow penetration of only half a voxel, and therefore work well under low-velocity manipulation. However, tunneling occurs when the manipulandum moves too fast for penalty method (Figure 12). Under a small number of contact points, collision detection dominates the computation time. Therefore, Figure 6 (c) also demonstrates that under a similar number of contact points, the discrete penalty method is the fastest whereas the continuous penalty method is a little bit faster than

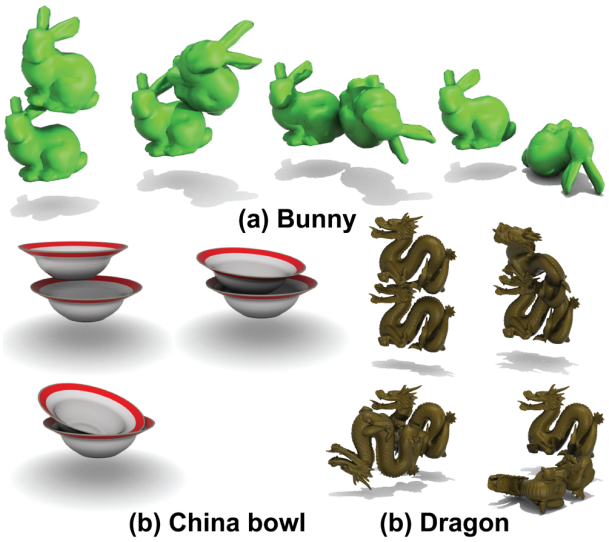

Fig. 9: *Simulation of bunnies, china bowls and dragons using continuous collision detection and continuous contact resolution.*

constraint-based method in simulation computation.

In our fourth example, we insert a peg (distance field object, $64 \times 64 \times 64$) into a hole (pointshell object, $6,691$ sample points) (Figure 7). Both methods can stably position the peg inside the hole when the sizes of the peg and the hole match well. However, the continuous penalty method produces many more contacts and performs slower than the constraint-based method. The continuous penalty method permits penetrations, albeit smaller than half a voxel, whereas the constraint-based method ensures that there are no penetrations. Therefore, when the size of the hole becomes smaller than the peg, the constraint-based method correctly detects that one cannot insert the peg, whereas the continuous penalty method still indicates that the insertion is possible. The problem can be alleviated by maintaining a small security distance beyond the colliding surface, but technically speaking, the continuous penalty method cannot guarantee that it will discover an interpenetration-free path. For each contact point, continuous penalty method needs to integrate the contact force and torque, and even gradients if implicit integration is applied. In this example, continuous penalty method produces many more contacts (around $1,000$ when the peg is fully inserted) than constraint-based method (always under 10). Due to a smaller number of contacts, the constraint-based method runs faster than the continuous penalty method in this example.

Graphical simulation. We also applied our continuous collision detection to rigid body graphical simulations. We drop one object onto another fixed object and compare the performance of point-by-point continuous collision detection versus using a 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 $\phi(p) = p_y$, where p_y is the y coordinate of point p. Figure 9 gives the simulation results for the bunny, china bowl, and dragon examples. 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

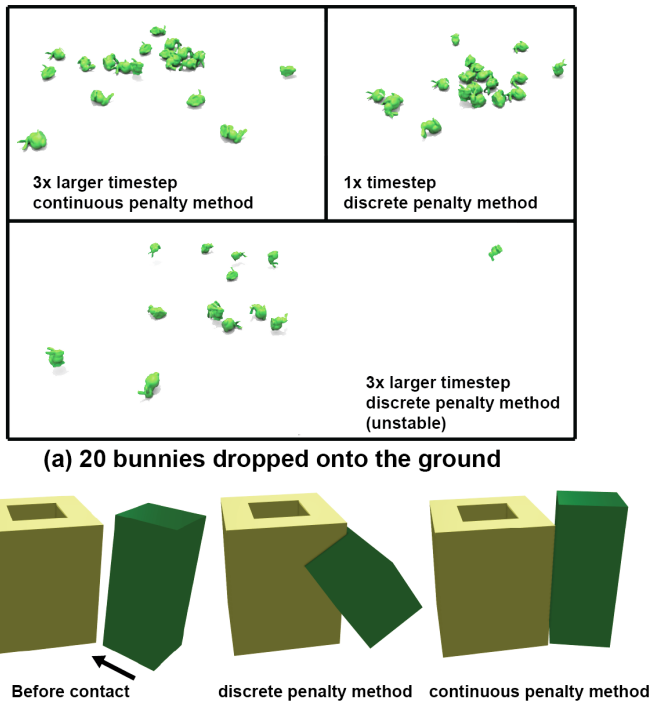

Fig. 10: ***Stability comparison between the discrete penalty method and the continuous penalty method.*** *(a) 20 rigid bunnies are dropped to the ground. Under the same stiffness, continuous collision detection with continuous forces and torques allows a $3\times$ larger timestep than discrete collision detection with discrete forces and torques. (b) A peg is manipulated into contact with a hole by a haptic device. Under the same timestep, the continuous penalty method allows $2.5\times$ larger stiffness.*

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.

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 physically based simulation. We compare linear (point-by-point) traversal vs using a sphere point hierarchy.*

Discussions. Continuous collision detection with continuous penalty forces and torques improves the simulation stability, compared to discrete collision detection with discrete forces and torques. In Figure 10 (a), we drop 20 rigid bunnies onto the ground using symplectic Euler integrator. Each bunny carries a pointshell and a signed distance field. The continuous collision detection and contact computation is performed twice, with each object assuming either role. Under a fixed stiffness level, the maximum stable timestep, i.e., the largest timestep for visually stable simulation, is $3\times$ larger in our continuous penalty method as compared to the discrete penalty method. We then fixed this timestep, and decreased the stiffness of the discrete method until

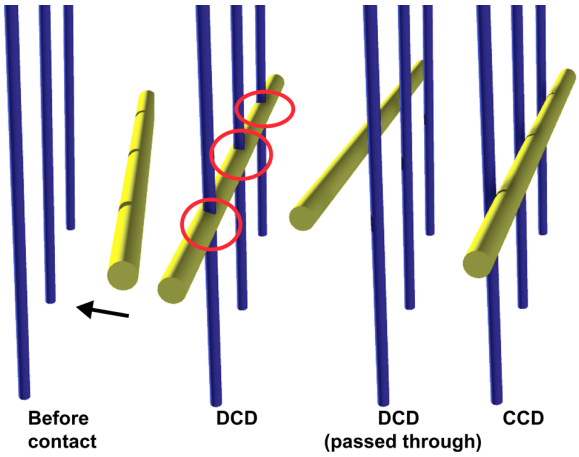

Fig. 11: ***Constraint-based method with discrete vs continuous collision detection.*** *A cylinder (yellow) is moved into contact with three vertical cylinders (blue). With discrete collision detection, the yellow cylinder either becomes stuck in deep penetration, or passes through the three vertical cylinders. Continuous collision detection correctly resolves the contact by detecting the first contact with the surface, and constrains the object to the surface.*

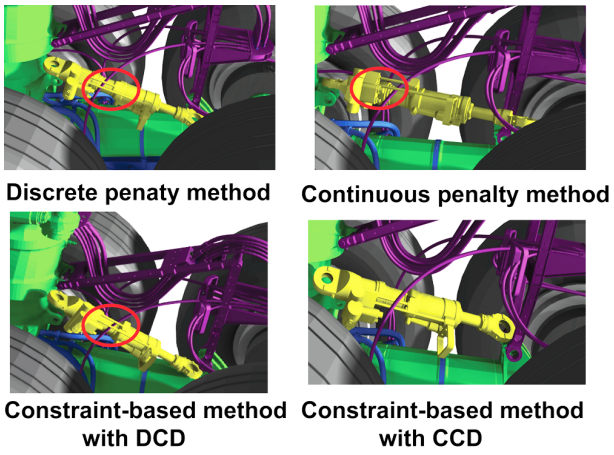

Fig. 12: ***Comparison between the penalty-based method and the constraint-based method.*** *Under both discrete and continuous penalty-based method, the hydraulic actuator can tunnel through the thin wires of the landing gear (shown in red circle). The constraint-based method, combined with continuous collision detection, guarantees an interpenetration-free path.*

it became stable. This caused a 2.75× 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× larger maximum stable timestep. In Figure 10 (b), we manipulate the peg into contact with the hole, using the haptic device with implicit integration in admittance mode. Under a fixed timestep, the continuous penalty method remains stable with a 2.5× larger stiffness than the discrete penalty method.

Continuous collision detection is also necessary to prevent deep penetration and tunneling artifacts in constraint-based haptic rendering. In Figure 11, we replace continuous collision detection in our constraint-based haptic rendering by only performing discrete collision detection at the end of the timestep. Discrete collision detection does not check for the contacts during the timestep. When the simulation object is moving towards contact with a high velocity, it can end up in a deep penetration, or even passes through the obstacle within one timestep. When the simulation object is in deep penetration, its contacts may have directions spanning most of the orthogonal plane, which in turn prohibits motion to resolve contacts, causing the object to be stuck (Figure 11). Therefore, the constraint-based method with discrete collision detection may not resolve the contact correctly because the contact normals are corrupted, or even no contacts are detected. This problem is more pronounced in the presence of thin geometries. Continuous collision detection guarantees that no contact is missed and returns the correct contact normal(s) of the earliest contact(s) along the trajectory.

Figure 12 compares the penalty-based method to the constraint-based method. The discrete penalty method does not prevent the tunneling artifacts, since it only performs collision detection at the end of the timestep. The continuous penalty method alleviates the problem since it performs collision detection along the trajectory and integrates the contact impulse over the time interval. However, when the obstacle is thin, such as the wire in the landing gear, and the manipulandum is moving fast, the contact impulse averaged by the timestep will be too small to enforce an interpenetration-free path. The constraint-based method, combined with continuous collision detection, prevents the penetration problem by constraining the simulation object at the first contact site.

Penalty-based methods can produce *"false-positive"* results, namely report that a path is possible when in reality it violates contact. In contrast, the constraint-based method can cause the manipulandum to become stuck in very complex configurations (Figure 13), even with continuous collision detection. These are "false-negative" results, namely reporting that insertion cannot occur when in reality it can. The contact force and torque directions can be analyzed under the Gauss map which is displayed as a unit sphere (circle in 2D). The inequality $Ja \leq 0$ in the quadratic program is feasible only if there exists a plane passing through the origin such that all the contacts are on one side of the plane. We found that this condition can become violated in very complex contact configuration in the presence of tight passages and narrow clearances. When this happens, the set of permitted accelerations is empty and the manipulandum becomes stuck, causing a false negative. Such configurations violate the $Ja \leq 0$ condition and occur because the contacts and their normals are necessarily sampled discretely on the object (at the points of the point-cloud), and because the continuous collision detection and the QP solver operate in finite-precision floating-point arithmetics. In this sense, penalty-based methods have an advantage in that they "relax" the strict contact conditions, permitting the simulation to proceed despite the infeasible constraints. This is similar to how hard constraints in general physically based simulation (say, between deformable objects) can cause locking, whereas soft constraints regularize inconsistent constraints, permitting the simulation or optimization to still find a reasonable solution.

6 Conclusion

We presented an efficient algorithm for continuous collision detection between points and distance fields, and showed how it be used

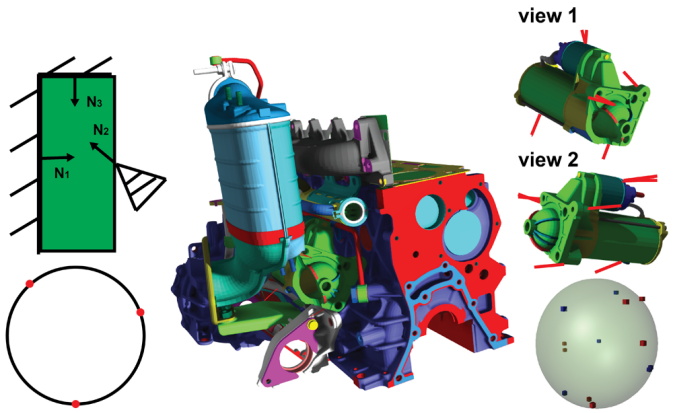

Fig. 13: ***False negatives with tight contact in the constraint-based method.*** *A vector from the sphere center to each red and blue dot represents a single contact force and torque direction, respectively. (a) A 2-D peg (shown in green) is inserted into a narrow passage. Before colliding with the top wall, contacts N_1 and N_2 permit upward insertion, but prohibit downward motion. Once the peg touches the top wall, it becomes stuck because both the upward and downward motions are forbidden. (b) We tried to insert a starter motor into its proper place in the car engine compartment. Insertion works fine if we offset the distance field slightly (about 1 mm), permitting a larger tolerance. However, under a narrower tolerance, the starter motor becomes stuck in a configuration where both the contact force and torque directions are distributed too broadly (the $Ja \leq 0$ condition is infeasible).*

with a penalty-based method, and a constraint-based method, for 6-DoF haptic rendering. 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.

ACKNOWLEDGMENTS

This research was sponsored by the National Science Foundation (CAREER-1055035, IIS-1422869), the Sloan Foundation, the Okawa Foundation, and USC Annenberg Graduate Fellowship to Hongyi Xu.

REFERENCES

[1] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy, "Six degree-of-freedom haptic rendering using voxel sampling," in *Proc. of ACM SIGGRAPH 99*. ACM, 1999, pp. 401–408.

[2] M. Renz, C. Preusche, M. Pötke, H.-P. Kriegel, and G. Hirzinger, "Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm," in *Proc. of Eurohaptics*, 2001, pp. 149–154.

[3] W. McNeely, K. Puterbaugh, and J. Troy, "Voxel-Based 6-DOF Haptic Rendering Improvements," *Haptics-e*, vol. 3, no. 7, 2006.

[4] J. Barbič and D. L. James, "Six-dof haptic rendering of contact between geometrically complex reduced deformable models," *IEEE Transactions on Haptics*, vol. 1, no. 1, pp. 39–52, 2008.

[5] M. Sagardia, T. Hulin, C. Preusche, and G. Hirzinger, "Improvements of the voxmap-pointshell algorithm-fast generation of haptic data-structures," in *53rd IWK-Internationales Wissenschaftliches Kolloquium, Ilmenau, Germany*, 2008.

[6] H. Xu and J. Barbič, "Adaptive 6-dof haptic contact stiffness using the gauss map," *IEEE Transactions on Haptics*, 2016, accepted for final publication.

[7] M. Jones, J. Bærentzen, and M. Sramek, "3D distance fields: a survey of techniques and applications," *IEEE Trans. on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581–599, 2006.

[8] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision Detection for Deformable Objects," *Computer Graphics Forum*, vol. 24, no. 1, pp. 61–81, 2005.

[9] M. Tang, D. Manocha, M. A. Otaduy, and R. Tong, "Continuous penalty forces," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 107:1–107:9, 2012.

[10] M. Ortega, S. Redon, and S. Coquillart, "A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 3, pp. 458–469, May 2007.

[11] R. Bridson, R. Fedkiw, and J. Anderson, "Robust Treatment of Collisions, Contact, and Friction for Cloth Animation," *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 594–603, 2002.

[12] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," *Journal of Computing and Information Science in Engineering*, vol. 5, no. 2, pp. 126–137, 2005.

[13] M. Tang, S. Curtis, S.-E. Yoon, and D. Manocha, "Interactive continuous collision detection between deformable models using connectivity-based culling," *IEEE Trans. on Visualization and Computer Graphics*, vol. 15, pp. 544–557, 2009.

[14] M. Tang, D. Manocha, and R. Tong, "Fast continuous collision detection using deforming non-penetration filters," in *Proc. ACM Symp. on Interactive 3D Graphics and Games (I3D)*, 2010, pp. 7–13.

[15] T. Brochu, E. Edwards, and R. Bridson, "Efficient geometrically exact continuous collision detection," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 96:1–96:7, 2012.

[16] J. M. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, and A. H. Barr, "Interval methods for multi-point collision between time-dependent curved surfaces," in *Proc. of ACM SIGGRAPH 93*, 1993, pp. 321–334.

[17] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," in *Computer graphics forum*, vol. 21, no. 3, 2002, pp. 279–287.

[18] X. Provot, "Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments," in *Graphics Interface*, 1997, pp. 177–189.

[19] T. Brochu and R. Bridson, "Numerically robust continuous collision detection for dynamic explicit surfaces," University of British Columbia, Tech. Rep., 2009.

[20] H. Wang, "Defending continuous collision detection against errors," *ACM Trans. on Graphics (SIGGRAPH 2014)*, vol. 33, no. 4, 2014.

[21] Y. Z. Hongyi Xu and J. Barbič, "Implicit multibody penalty-based distributed contact," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 9, 2014.

[22] K. Moustakas, "6-dof haptic rendering using distance maps over implicit representations," *Multimedia Tools and Applications*, vol. 75, no. 8, pp. 4543–4557, 2016.

[23] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of Clothing with Folds and Wrinkles," in *Proc. of the Symp. on Comp. Animation 2003*, 2003.

[24] A. Fuhrmann, G. Sobotka, and C. Groß, "Distance fields for rapid collision detection in physically based modeling," in *Proceedings of GraphiCon*, 2003, pp. 58–65.

[25] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan, "Interactive ray tracing for isosurface rendering," in *Proceedings of the conference on Visualization'98*. IEEE Computer Society Press, 1998, pp. 233–238.

[26] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl, "Cell-based first-hit ray casting," in *Proc. of the Symposium on Data Visualisation*. Eurographics Association, 2002, pp. 77–ff.

[27] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek, "Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing." in *Proc. of Virtual Reality, Modeling, and Visualization*, vol. 4, 2004, pp. 429–435.

[28] O. Jamriška, "Interactive ray tracing of distance fields," *The 14th Central European Seminar on Computer Graphics*, 2010.

[29] L. Glondu, S. C. Schvartzman, M. Marchal, G. Dumont, and M. A. Otaduy, "Efficient collision detection for brittle fracture," in *Proc. of the Symp. on Comp. Animation 2012*. Eurographics Association, 2012, pp. 285–294.

[30] M. Sagardia and T. Hulin, "Fast and accurate distance, penetration, and collision queries using point-sphere trees and distance fields," in *ACM SIGGRAPH 2013 Posters*, pp. 83:1–83:1.

[31] X. Zhang, D. Wang, Y. Zhang, and J. Xiao, "Configuration-based optimization for six degree-of-freedom haptic rendering using sphere-trees," in *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2011, pp. 2602–2607.

[32] S. Laycock and A. Day, "A survey of haptic rendering techniques," *Computer Graphics Forum*, vol. 26, pp. 50–65, 2007.

[33] M. A. Otaduy and M. C. Lin, "Stable and Responsive Six-Degree-of-Freedom Haptic Manipulation Using Implicit Integration," in *Proc. of the World Haptics Conference*, 2005, pp. 247–256.

[34] R. Weller and G. Zachmann, "A unified approach for physically-based simulations and haptic rendering," in *Proc. of ACM SIGGRAPH Symposium on Video Games*, 2009, pp. 151–159.

[35] C. Zilles and J. Salisbury, "A Constraint-based God-object Method for Haptics Display," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 1995, pp. 146–151.

[36] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot, "Realistic Haptic Rendering of Interacting Deformable Objects in Virtual Environments," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 12, no. 1, pp. 36–47, 2006.

[37] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai, "Staggered Projections for Frictional Contact in Multibody Systems," *ACM Transactions on Graphics*, vol. 27, no. 5, pp. 164:1–164:11, 2008.

[38] M. A. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross, "Implicit contact handling for deformable objects," in *Computer Graphics Forum*, vol. 28, no. 2, 2009, pp. 559–568.

[39] S. Redon, "Fast continuous collision detection and handling for desktop virtual prototyping," *Virtual Reality*, vol. 8, no. 1, pp. 63–70, 2004.

[40] J. Amanatides, A. Woo *et al.*, "A fast voxel traversal algorithm for ray tracing," in *Proceedings of EUROGRAPHICS*, vol. 87, 1987, pp. 3–10.

[41] J. Nathan, *Basic Algebra (second edition)*. Dover, 2009.

[42] D. D. Nelson, D. E. Johnson, and E. Cohen, "Haptic rendering of surface-to-surface sculpted model interaction," in *ACM SIGGRAPH 2005 Courses*, 2005, p. 97.

[43] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha, "Six degree-of-freedom haptic display using incremental and localized computations," *Presence-Teleoperators and Virtual Environments*, vol. 12, no. 3, pp. 277–295, 2003.

[44] J. Colgate, M.C.Stanley, and J.M.Brown, "Issues in the Haptic Display of Tool Use," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 1995, pp. 140–145.

[45] R. J. Adams and B. Hannaford, "A Two-Port Framework for the Design of Unconditionally Stable Haptic Interfaces," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 1998, pp. 1254–1259.

[46] M. Wan and W. A. McNeely, "Quasi-Static Approximation for 6 Degrees-of-Freedom Haptic Rendering," in *Proc. of IEEE Visualization 2003*, 2003, pp. 257–262.

[47] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[48] E. Hairer, C. Lubich, and G. Wanner, "Geometric numerical integration illustrated by the störmer–verlet method," *Acta Numerica*, vol. 12, pp. 399–450, 2003.

[49] H. Xu and J. Barbič, "Signed distance fields for polygon soup meshes," in *Proc. of the Graphics Interface Conference*, 2014, pp. 35–41.

[50] O. B. Bayazit, G. Song, and N. M. Amato, "Enhancing Randomized Motion Planners: Exploring with Haptic Hints," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 529–536.

[51] J. J. Kuffner, "Effective sampling and distance metrics for 3d rigid body path planning," in *IEEE Int. Conf. on Robotics and Automation 2004*, April 2004.

Hongyi Xu is a PhD student in computer science at the University of Southern California. He obtained his BS degree from Zhejiang University. His research interests are in computer graphics, physically based animation, contact and interactive physics.

Jernej Barbič is associate professor of computer science at USC. In 2011, MIT Technology Review named him one of the Top 35 Innovators under the age of 35 in the world (TR35). Jernej's research interests include nonlinear solid deformation modeling, model reduction, collision detection and contact, optimal control and interactive design of deformations and animations. He is the author of Vega FEM, an efficient free C/C++ software physics library for deformable object simulation. Jernej is a Sloan Fellow (2014).