Six-DoF haptic rendering of contact between geometrically complex reduced deformable models

Jernej Barbič, *Member, IEEE,* Doug L. James, *Member, IEEE*

***Abstract*— Real-time evaluation of distributed contact forces between rigid or deformable 3D objects is a key ingredient of 6-DoF force-feedback rendering. Unfortunately, at very high temporal rates, there is often insufficient time to resolve contact between geometrically complex objects.**

We propose a spatially and temporally adaptive approach to approximate distributed contact forces under hard real-time constraints. Our method is CPU based, and supports contact between rigid or reduced deformable models with complex geometry. We propose a contact model that uses a point-based representation for one object, and a signed-distance field for the other. This model is related to the Voxmap Pointshell Method (VPS), but gives continuous contact forces and torques, enabling stable rendering of stiff penalty-based distributed contacts. We demonstrate that stable haptic interactions can be achieved by point-sampling offset surfaces to input "polygon soup" geometry using particle repulsion. We introduce a multi-resolution nested pointshell construction which permits level-of-detail contact forces, and enables graceful degradation of contact in close-proximity scenarios. Parametrically deformed distance fields are proposed for contact between reduced deformable objects.

We present several examples of 6-DoF haptic rendering of geometrically complex rigid and deformable objects in distributed contact at real-time kilohertz rates.

***Index Terms*— haptics, 6-DoF, distributed contact, deformations, distance fields, VPS**

I. Introduction

THIS paper presents an efficient hard-real-time CPU-based algorithm for 6-DoF haptic rendering of contact between two geometrically complex 3D objects (see Figure 1), each of which can undergo deformations. Our time-critical algorithm resolves *distributed* contact forces between two 3D objects, i.e., contacts with potentially several simultaneous contact sites each distributed over a non-zero surface area (also called multi-point or multi-site contact; see Figure 2). Distributed contact is essential to 6-DoF haptic rendering and is challenging to simulate due to the competing demands of high temporal rates for contact force evaluation, e.g., 1 kHz in our examples, and the computational burden introduced by geometrically complex rigid and (more notoriously) deformable objects. Deformable object simulators can not timestep complex simulation meshes at haptic rates. We drive deformations of our geometrically complex meshes with low-degree-of-freedom deformable models, and demonstrate how such deformable model simplification (reduction) can be exploited by our contact processing algorithm to enable haptic update rates.

Our approach is most closely related to the Voxmap-PointShell method (VPS, [1]–[3]), an industrial 6-DoF haptic rendering algorithm capable of simulating distributed geometrically detailed

J. Barbič is with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Boston, MA, 02139, USA, e-mail: barbic@mit.edu

D. L. James is with the Department of Computer Science, Cornell University, Ithaca, NY, 14853, USA, e-mail: djames@cs.cornell.edu

Manuscript received Feb 22, 2008; revised May 22, 2008.

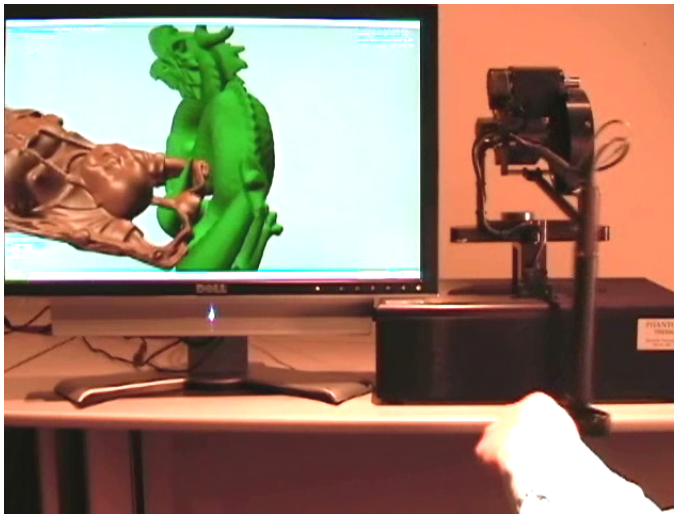

Fig. 1. **Six-DoF haptic rendering with complex deformable geometry:** the user is manipulating a rigid Buddha (distance field object) against the deformable dragon (pointshell object). The simulation computes deformations, detects collision and computes contact forces and torques in one loop running at 1000 frames per second.

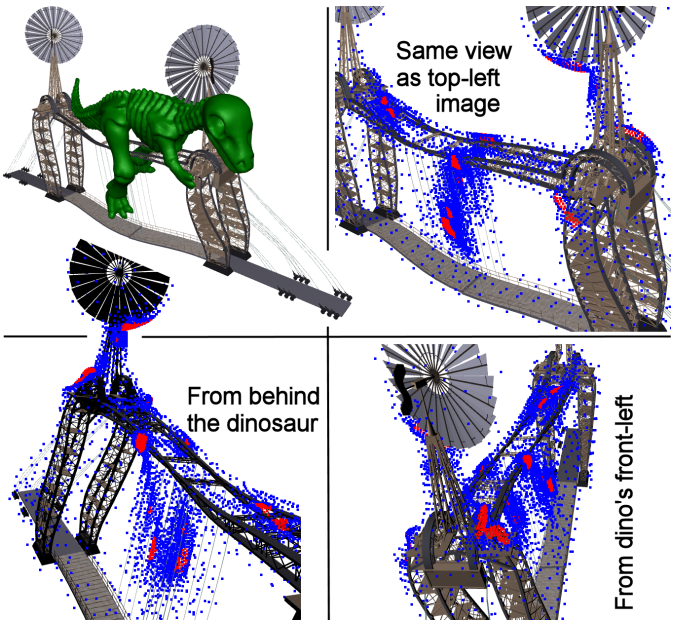

Fig. 2. **Distributed deformable contact:** (**Top Left**) contact configuration between a nonlinear reduced deformable bridge (15 reduced deformable DoFs) and a rigid dinosaur, with 12 separate contact sites; (**Other images**) contact sites viewed from three different camera angles. A total of 5200 points (in blue) were traversed, resulting in 426 points (in red) in contact. Note: temporal coherence was disabled for this figure.

contact between a *rigid* environment and a *rigid* point-sampled object (*pointshell*). The pointshell is a collection of points (with normals) positioned near-regularly at the surface of the object. The VPS environment is modeled as a *voxmap* datastructure: a voxelization of the polygonal environment geometry where each voxel contains a fixed-width value, such as 2-bit values (interior, surface, next to surface, free space) in the original VPS, or more bits in recent VPS extensions [3]. In every VPS haptic cycle, pointshell points are queried against the voxmap to determine contact.

Unfortunately, objects modeled by VPS are entirely rigid, and for good reason: geometrically complex deformable objects are computationally expensive since (a) deformable object simulations can not easily achieve haptic rates (1000 Hz) for geometrically interesting models, and (b) deformable collision detection is expensive especially for close-proximity scenarios typical of contact. These fundamental difficulties have seriously hampered the progress of a variety of emerging haptic rendering applications, such as haptic gaming. One other application of haptic rendering is virtual prototyping: haptics can allow airplane designers to check if a certain mechanical part can be positioned into (or removed from) designated locations, thereby avoiding critical design flaws before manufacturing the structure, e.g., fire detector removal from an aircraft engine assembly during routine servicing [2]. Supporting deformable geometry is important since airplanes include numerous deformable parts such as hoses, wires, plastic parts, sheet metal, etc. Part-removal analysis using only rigid simulations can lead to false reports that a part can not be (dis)assembled when in reality it can. For example, wires and hoses may be deformed to let an object through a narrow passage (see Figure 3).

A. Overview

Our approach is designed to work with a variety of deformable models that support a basic two-step simulation process: (1) a fast timestep of deformable dynamics; (2) fast evaluation of individual deformed surface point positions and normals to adaptively resolve contact. Suitable reduced models include classic linear modal vibration models, reduced nonlinear models, low-resolution deformable models with embedded geometry, multi-resolution models, articulated rigid or flexible multibodies or skinned deformable models. We collectively refer to such models as *reduced* deformable models. In our implementation we employ reduced geometrically nonlinear FEM models [4]. These models run at haptics rates, e.g., much less than one hundred microseconds per timestep in our examples, and are suitable for large deformations with large rotations but small local strain. We make extensive use of the low-dimensional shape parameterization for time-critical collision and force processing. Regardless of the particular reduced deformation model used, our approach addresses stable haptic display of distributed contact between two objects *each* with non-trivial geometry.

As in VPS, one of our objects is modeled as a pointshell. We explore the relationship between point-sampling and haptic stability, and propose pointshells that near-uniformly sample a *smooth offset surface* to the input triangle ("polygon soup") mesh. Instead of a VPS voxmap, we use a floating-point signed-distance field to (a) make the contact force continuous (improving stability), (b) support time-critical contact estimation, and (c) exploit temporal coherence. At every haptic cycle, pointshell points are queried against the distance field to determine contact and compute contact forces (see Figure 4). We support contact between a rigid and a reduced deformable model using a rigid distance field and a deformable pointshell. We introduce output-sensitive signed-distance field oracles for deformable models to support contact between two reduced deformable models. Our approach scales with increasing geometric detail. To the best of our knowledge, we are first to demonstrate haptic rendering of deformable contact, with several simultaneous contact sites, on a 6-DoF haptic device, with force and torque stiffnesses at hardware device limits, between such geometrically complex models.

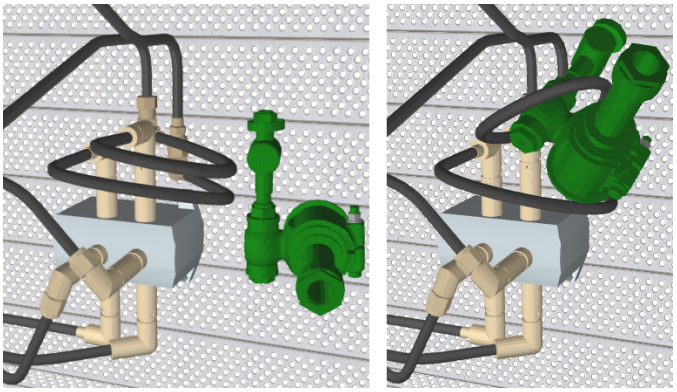

Fig. 3. **Real-time 6-DoF haptic rendering** of rigid and deformable objects in contact. The operator is navigating the green rigid part in between two deformable hoses.

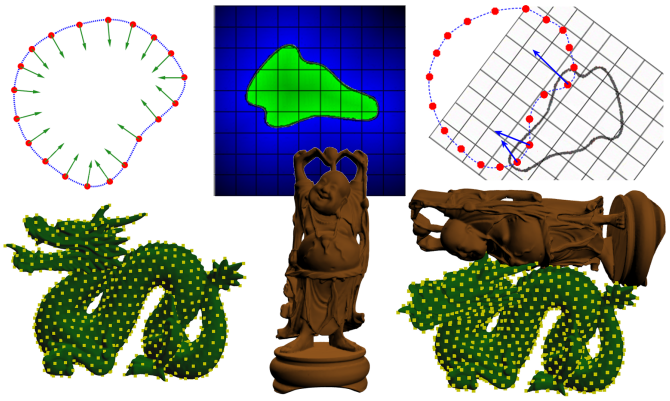

Fig. 4. **Point-contact model:** (**Left**) Reduced-deformable pointshell with inward normals; (**Middle**) the distance field; (**Right**) deformable contact and contact forces.

To support haptic display of geometrically detailed models, e.g., involving a million pointshell points, we propose organizing the pointshell into a *nested* multi-resolution hierarchy. The hierarchy is constructed by fitting a desired number of points (particles) onto an appropriately constructed smooth manifold (offset) surface, using multi-resolution particle repulsion. Our sampling is near-uniform which provides economic pointshells and improves haptic stability. We use a precomputed sphere-tree hierarchy to bound the pointshell points, *with sphere centers located at (non-leaf) pointshell points*. For deformable models, this gives a point-based modification of the Bounded Deformation Tree (BD-Tree) [5]. Such a hierarchy permits large pointshells, bigger than what could be examined point by point during one haptic cycle: if a point on some coarse hierarchical level is

sufficiently far from contact, none of the points in the subtree can be in contact, and then those points need not be checked. In contrast, if a large part of the pointshell is in contact or close to contact, few subtrees will be culled and a large part of the pointshell will need to be examined point by point. We address this problem by providing *graceful degradation of contact*: if there is not enough computation time to fully complete the tree traversal, the algorithm still returns a reasonable answer, with accuracy dependent on contact-configuration difficulty and available processing power. Graceful degradation is achieved by traversing the nested hierarchy in a breadth-first order, rendering deeper and deeper tree levels until out of computation time. We use two separate activation thresholds to avoid abrupt changes in the rendered depth during consecutive haptic cycles. The simulation also uses temporal coherence to time-sample the individual points at rates depending on distance to contact (temporal adaptivity).

We compute both contact forces and torques, and haptically render them using static virtual coupling [2]. We propose a quasi-static damping scheme to improve the passivity of penalty-based contacts. In haptics, higher update rates generally enable stable rendering of stiffer forces and torques; but reaching such high rates is challenging for models with detailed geometry. In our system, the deformations, collision detection and contact force computations run together in one loop at haptic rates (1000 Hz). We do not need to extrapolate from lower-rate signals. A multi-rate simulation using the contact gradients of Section V could exploit user's limited sensing bandwidth to further increase example complexity [6], [7].

The technical sections of this paper are organized as follows. Section II gives our proposed contact model for computing contact forces between a point and a distance field. It also explains how we generate our pointshells and distance fields from input "polygon soup" triangle meshes, and how points are organized into a multi-resolution hierarchy. In Section III, we give our run-time algorithm: we explain how we traverse the hierarchy at run-time, how we exploit temporal coherence, and how we adjust the rendered level of detail to support graceful degradation of contact. In Section IV, we extend the algorithm to deformable simulation; first, by making the pointshell deformable (§ IV-A), next by adding deformations to the distance field (§ IV-B).

B. Related Work

Haptic rendering has received increasing attention over the last decade [8], and significant challenges remain especially for rendering deformable systems. Early methods were developed for tissue simulations during virtual surgery [9], and mostly used mass-spring deformable objects. Space-time adaptive and multi-rate methods have been successful for real-time deformable object simulation [10], [11], but time-critical distributed contact algorithms have received less attention (c.f. [12]). A variety of preprocessing techniques, such as precomputed Green's functions, exist for accelerating linear elastostatic models to support haptic force-feedback rendering of point-like contact, or modest multi-point contacts via low-rank updates [13]–[17]. Unfortunately, most such haptic rendering papers assume interactions based on pulling on (or constraining) certain vertices, not performing real-time deformable-rigid object collision detection and distributed contact force and torque computations with complex geometry.

As mentioned, the Voxmap-PointShell (VPS) method was introduced by [1] to support rigid-rigid contact rendering. Pointshell accuracy and virtual coupling stability were improved in [18]. In [2], they replaced the original VPS mass-spring-damper with a quasi-static spring. The recent paper of [3] introduces wider voxmaps (4-bit), which now have more room to store distance-to-nearest-contact information for free-space voxels. Since the haptic device is user controlled, there exists an upper bound on the maximum possible velocity of a point shell point, and therefore points located in a voxel sufficiently far from contact need not be checked for several haptic cycles (temporal coherence). Also, the pointshell is traversed on two levels: a coarse level shell is tested against a coarse level voxmap, followed by finer tests if necessary. The hierarchy only consists of two levels, the simulation is rigid, and graceful degradation of contact was not addressed.

Our progressive interruptible point-based contact-sampling approach is analogous to the QSplat point-based rendering algorithm [19] which represented the point cloud hierarchically to enable level-of-detail control for *graphical* rendering. Our oriented pointshell can be seen as a surfel point cloud. A single-resolution contact algorithm for two deformable point-sampled objects has been presented in [20], reaching interactive, but not haptic rates. Hierarchical point-contact handling was explored in [21], where they demonstrate near-interactive (a few frames per second) contact of point-based quasi-rigid (quasi-static and linear) deformable objects. In this paper, we present a hierarchical contact rendering method designed for haptic rendering. We demonstrate how to maintain upper bounds on the amount of computation per haptic cycle, and how to minimize contact force artifacts when run-time constraints force a transition to a different contact resolution level.

Bounding volume hierarchies can be used for *interruptible* collision detection for rigid [22] and embedded deformable models [12] by traversing the hierarchy in a breadth-first manner. Time-critical contact can be approximated by applying contact forces between contacting (sphere) primitives at the deepest level traversed in a particular frame. Unfortunately, such approaches can have trouble resolving close-proximity and conforming contact configurations in high-rate scenarios (as in haptic rendering): finer bounding volumes more closely approximate the object, and as such the effective contact surface (and therefore forces) depend on the level of detail rendered at any instant. In our work, the points at different resolution levels progressively sample the same well-defined contact surface, thereby making contact at different levels of detail more consistent.

Six-DoF haptic rendering of complex *rigid* objects can be achieved using a number of techniques, in addition to VPS, such as multi-resolution collision detection on convex decompositions, followed by a clustering of contacts [23]. A generalization of the god-object method to rigid objects has been presented in [24], producing rigid haptic simulations with tens of thousands of triangles. Multi-resolution contact haptics has been demonstrated for polygonal geometry by appropriately simplifying the objects at different hierarchical levels, to preserve the user haptic *sensation* [6], [25]. The authors suggest that for contact sites of large area, resolving the contact at a coarser resolution will not result in significant sensation degradation; this has motivated our graceful degradation algorithm. These approaches, however, do not directly apply to point-sampled objects or deformable objects.

Linear complementary problem (LCP) solvers can be used to solve the Signorini contact problem between a rigid object and a linear deformable object [26], at haptic rates. LCP solvers are

computationally demanding which limits the approach to models of modest geometric complexity. Haptic interaction between two objects with detailed surface deformations can be achieved using geometry proxies combined with GPU image space techniques, in a multi-rate simulation with collision detection running at graphics rates [27]. Haptic results were demonstrated using a 3-DoF device, with one large contact area between a rigid object and the deformable surface of the other object.

A deformable pointshell haptic rendering example has been demonstrated in [4]; however, one of the two contacting objects (a small rigid ball) had trivial geometry. In this paper, *both* contacting objects have non-trivial geometry, resulting in a much more difficult collision detection problem; also the contact model of analytical gradients (as used with the ball) gives discontinuities if naively extended to non-trivial distance field geometry. We published a conference version of our work in [28]. In this paper, we extend the technique to handle deformable vs deformable contact. We give convergence properties of our contact model. We also present experimental results on rendering fine geometric detail, and asymmetry of the contact model.

II. Multi-resolution Point-contact Model

In this section, we describe our point-contact model; its components were selected to enable stable haptic rendering of large models at fast update rates. Our point-contact models for rigid-rigid, rigid-deformable, or defo-defo contact are conceptually identical: during every haptic cycle, the contact penalty forces are determined by querying the points of the pointshell object against the signed distance field of the other object (see Figure 4). The user interactively manipulates one of the two objects (the *haptic object*). In our implementation, the distance-field object is manipulated; the distance field then translates and rotates with the haptic object. The roles could be reversed. Both pointshell and the distance field are computed in a pre-processing stage from input ("polygon soup") triangle meshes. Pointshell is fitted onto a smooth manifold surface, using a multi-resolution construction that permits incremental construction of contact forces at runtime.

A. Contact Forces

Negative distance field values (by convention) denote the interior of the distance field object, and therefore points in contact. Penalty forces are computed for points in contact. We choose the direction of the force to equal the inward normal of the pointshell point in contact (and as such the force acts to resolve contact).The magnitude is determined from the amount of penetration. The force is (see also Figure 4, Right):

$$\mathbf{F} = -k_{\mathrm{C}}\, d\, \mathbf{N}, \tag{1}$$

where k_{C} is the contact penalty force stiffness, $d < 0$ is the signed distance field value, and $\mathbf{N}$ is the point's inward normal in the world coordinate system. The point world-coordinate position and normal computation are discussed in later sections. Note that the total force and torque can be computed easily using this model, and the trilinearly interpolated distance field.

This particular contact model was selected because of its useful approximation properties for haptics. Our motivation was to improve the VPS contact model. VPS force directions are continuous, but magnitudes can be discontinuous when crossing voxmap voxel boundaries. It is then up to virtual coupling to remedy

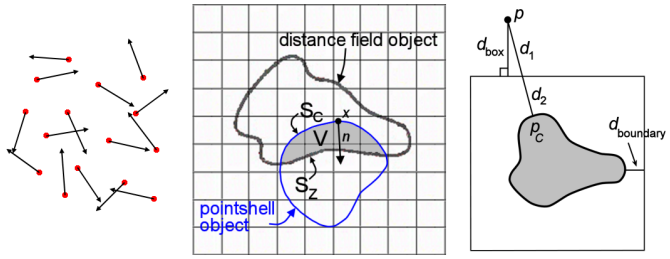

Fig. 5. (**Left**) A low-quality pointshell; haptic signal will be chaotic; (**Middle**) analytical contact limit (Equation 2); (**Right**) conservative extension of the distance field outside of the distance field box.

these discontinuities; lower stiffness levels might be required to preserve stability. In our first contact model attempt, we obtained both force direction and magnitude from the distance field object, by using distance field gradients. The gradient, however, is discontinuous across voxel boundaries and across interior object medial axes. The resulting force direction discontinuities gave poor haptic signals. Therefore, we designed our contact model to only use distances and not distance gradients. Our contact forces are continuous functions of positions, orientations and deformations of the two contacting objects, which permits stable stiffer contacts. We found our contact model to provide more stable haptic feedback than the VPS voxmap model, especially at higher stiffness settings.

In our model, only the normal of one of the two contacting objects (that of the pointshell object) determines force direction. This is both an approximation used for speed, and robustness; also note that for smooth surfaces in non-penetrating contact, the normals of the two opposing surfaces at the contact point are colinear. Our contact forces are temporally coherent in that contact sites (points in contact) vary smoothly from one haptic cycle to the next, improving stability. The contact force can start decreasing (but does not invert) under large penetrations, e.g., when crossing a medial axis of the distance field object. However, this issue is remedied by keeping the contacts shallow (using virtual coupling saturation, and schemes that limit haptic object velocities, see Section V). Our contact model is asymmetric: different contact forces will be obtained if the roles of the pointshell and distance field object are reversed. However, due to shallow penetrations, the two contact forces are very similar (see Figure 15 in the Results section).

B. Pointshells sampling closed manifold surfaces

We found that the haptic signal quality greatly depends on the quality of the pointshell; a good pointshell can mean the difference between a very stable signal and a very chaotic signal (see Figure 5, Left). At first, we considered pointshells sampling "polygon soup" geometry directly, but stable haptic rendering can not be guaranteed due to unpredictable normals and the non-closed and non-oriented nature of the surface sampling. Consequently, we designed our pointshells to sample a *closed manifold oriented surface*. When combined with virtual coupling saturation (§ V), this restricts the distance field object to outside of the pointshell object (or at most a shallow penetration), where the pointshell normals point in the direction that resolves contact. It is critical that pointshells resolve surfaces well and that normals are set orthogonal to the surface; isolated points can get rotated arbitrarily and can even push the object deeper into contact.

Another problem happens if points are too far apart, allowing a small feature in the distance field object to slip undetected in between two points; we address this by keeping the pointshell resolution equal or finer than the distance field resolution. The original VPS pointshell was obtained by voxelizing the pointshell geometry, and using centers of surface voxels as points of the pointshell. Such an approach is simple and fast, however we found that it can result in a relatively large (redundant) number of points. Also, the contact surface is sampled on an axis-aligned grid, which can result in staircase-like point positions. We found that pointshells positioned on *smooth closed manifold surfaces* tend to be stable; high frequency components, either in point locations or point normals tend to cause instabilities. Sharp corners can still be rendered, albeit not aggressively.

We note that our contact model converges to a well-defined limit with progressively finer (near-regular) pointshells, and this limit has an intuitive volume integral interpretation (obtained using the divergence theorem):

$$\mathbf{F}_{\text{limit}} = \frac{-k_{\text{C}}\, L \int_{S_C} d(x)\mathbf{n}(x) dS}{\int_{S_C} dS} = \frac{k_{\text{C}}\, L \int_V \text{grad}\, d(x) dV}{\int_{S_C} dS}, \tag{2}$$

where $d(x)$ and $\mathbf{n}(x)$ are the signed distance value and inward normal at x, respectively (see Figure 5, Middle), and the integer L is the scaling threshold for many-points contacts (we use $L = 10$; see Section V). Our discrete contact model of Equation 1 can therefore be seen as resolving contact between a surface and a distance field object, by approximating the surface integral of Equation 2 with a discrete Riemann sum evaluated at the locations of pointshell points. The norm of the distance field gradient is typically close to 1. For contacts where the gradient does not significantly change direction within the contact volume (e.g., in locally planar regions), the limit contact force is proportional to the penetration volume, divided by the surface area of penetration.

C. Distance field generation and evaluation

Computing a *signed* distance field is straightforward for closed manifold meshes. For "polygon soups", we first compute an *unsigned* distance field, then extract an isosurface for some small distance value using marching cubes to obtain a closed manifold mesh [29] (see [30] for an alternative approach). Next we remove any mesh components completely contained inside another component, and then compute a *signed* distance field with respect to the remaining closed manifold mesh(es). At runtime, we trilinearly interpolate the distance field grid values to the query location when inside the distance field box. If outside the box, we propose using the conservative estimate $d(p) \geq d_{\text{box}} + d_{\text{boundary}}$, where d_{box} is the shortest distance from the query location p to the distance field box, and d_{boundary} is the query-independent minimum distance field value on the surface of the distance field box (we give proof in Figure 5, Right).

D. Multiresolution nested pointshell

At every haptic cycle, our algorithm can traverse the pointshell linearly (point by point), evaluating the contact model of Equation 1, and rendering the sum to the user (via virtual coupling, § V). However, only small pointshells ($\sim$ 4000 points on our hardware) fit into the computational budget of one haptic cycle. To some extent, larger pointshells are possible by using temporal coherence and wider voxels [3]. To be able to simulate much larger pointshells (rigid or deformable), we propose organizing the pointshell into a *nested tree hierarchy* (see Figure 6). We

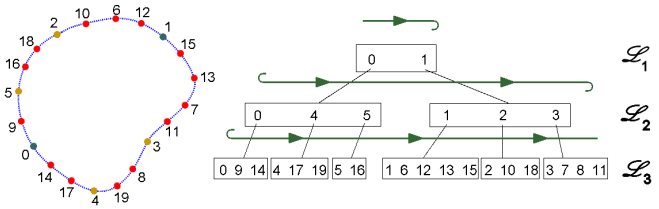

Fig. 6. **Nested point-tree:** (**Left**) The pointshell; (**Right**) the hierarchy, the traversal order and tree levels $\mathcal{L}_i$. Particle-repulsion levels are 0-1, 2-5, 6-19.

generate our multi-resolution pointshells by fitting a set of *particles* (points) onto a smooth manifold surface, using particle repulsion [31], [32]. Initially, we simply tried sampling surfaces randomly, or we used vertices of input triangle meshes, with poor results. Repulsion gives near-regular point distributions at all hierarchical levels (see Figure 7), which improves haptic stability. It works by randomly positioning a desired number of points (imagine electrons) to the surface, followed by a simulation where the points repel each other while constrained to the surface. The nested point hierarchy is constructed by freezing converged points, then injecting the points of the next level (we use a $4\times$ branching factor). Our construction is related to [33]; however, to support non-manifold input geometry, we constrain points on a level set of a distance field as opposed to a triangle mesh. Repulsion requires tuning two parameters (repulsion force strength and repulsion force kernel width). We manually (interactively) tune the two parameters on a small pointshell, then run the multi-resolution repulsion simulation offline, decreasing force strength and kernel width by a factor of two for each new level. Running times were on the order of one to two hours for our 256K pointshells. Repulsion is very parallel and could be accelerated using multiple cores or a GPU. Details are available in [34].

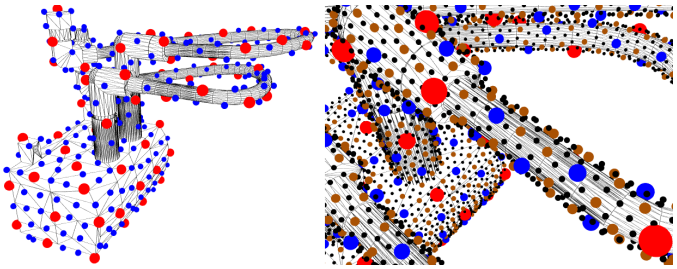

Fig. 7. **Multi-resolution oriented pointshell:** (**Left**) The first two levels of a part of the hose scene; (**Right**) detail with four levels shown. Points are fitted on a small-distance offset oriented manifold surface to support "polygon soup" input geometry, and provide oriented surface normals for contact.

After particle repulsion, each point is positioned, and assigned to one of the disjoint *particle-repulsion* levels. It is now necessary to organize the points into a *nested* hierarchical datastructure (see Figure 6, Right), so that contact forces can be computed progressively by a breadth-first hierarchy traversal (Section III). We first generate *tree levels* $\mathcal{L}_i$: the set $\mathcal{L}_i$ consists of an instance of every point of particle-repulsion levels 1 through i. Therefore, if a point belongs to particle-repulsion level i, an instance (a copy) of this point will appear in all tree levels $\mathcal{L}_j$, for $j \geq i$. Instances can share common memory data. The deepest tree level consists of a single instance of all the points in the pointshell. The elements of

$\mathcal{L}_i$ are our tree *nodes* on tree level i. This construction was chosen so that all nodes at all levels lie on the actual contact surface $\mathcal{T}$, and that each $\mathcal{L}_i$ samples $\mathcal{T}$ about uniformly (without gaps), with progressively finer resolution with level index i. We establish the tree parent-child relationships by traversing nodes at every level $\mathcal{L}_{i+1}$, and assigning each node to the nearest node from level $\mathcal{L}_i$. A bounding sphere radius is computed for each non-leaf tree node, centered at the location of the node, and covering all nodes in the subtree, in the undeformed configuration. The radius of the bounding sphere gets progressively smaller for instances of the same point at different levels. The nested point tree can be augmented to support deformable pointshells (§ IV-A).

III. Time-critical Progressive Contact Forces

We now describe the core approach for time-critical evaluation of contact forces when one object is represented as a rigid or deformable pointshell, and the other object provides a signed-distance oracle, $d(p)$.

Runtime tree traversal: At every haptic cycle, the algorithm traverses the pointshell hierarchy in breadth-first order. For every tree node, the algorithm looks up the value of the signed distance field at the node's location. If the distance field value is greater than the radius, no point in the subtree can collide, and the subtree is not traversed further (see Figure 8 (Middle, e.g., node 2)). Otherwise, all children of the node are added to a list for future traversal (see Figure 8 (Middle, e.g., nodes 0, 3)). If the distance field value is negative (contact), a penalty force is assigned to the node, just like in the single-resolution case. If child copies of the node are visited during later traversal, no additional force is accumulated (but the copy is not skipped from traversal since its proper children may be in contact).

Temporal coherence: Each time a tree node is processed, an estimate of the earliest possible haptic cycle when the bounding ball at this node can enter contact is determined and stored with the node. This estimate is computed based on node's current distance to contact, sphere radius, and maximum point velocity that can occur in the simulation. It can be the immediate next cycle if the node is close to contact (or in contact), or it can be hundreds of cycles into the future for nodes far from contact. Next time a node is about to be processed, we can safely skip it if the current haptic cycle is earlier than what stored with the node. Temporal coherence reduced our simulation times 20%-45%.

Graceful degradation: If the contact scenario is not very involved, much of the tree will be quickly culled and only a small fraction of the pointshell will be traversed. In contrast, if there are many different contact sites, such as when the pointshell object is positioned close to the distance field object, a large fraction of the pointshell will need to be traversed, preventing a complete tree traversal within one haptic cycle. To support graceful degradation in such cases, we traverse the tree progressively level by level (see Figure 8). While traversing the list of nodes that need to be visited on level i (as determined by level $i-1$), we build the traversal list for level $i+1$. At the end of level i, we compare the size of this list to a measure of the remaining computation time for the haptic cycle. Only if enough time remains, we render the next level. The algorithm always either completes a level or does not start it. The resulting contact forces are identical to those obtained under an exhaustive tree-less traversal of points up to a given level. Note that such a scheme is of course approximate: only geometric features resolved by points on the deepest traversed level will be rendered. If the surface contains detail beyond what the particular CPU can render, some points in contact (deep in the hierarchy) could remain undetected; no force will be applied to such points. The benefit of our construction is that it produces a reasonable stable contact force given the available CPU power. The rendered surface resolution and contact force accuracy both increase with available computing power.

LOD control: The amount of work required to process a node is predictable and it is not necessary to time the execution explicitly. Instead, we count the total number of tree nodes visited during the current haptic cycle. We use two thresholds to determine whether to render the next tree level. All the levels up to and including the deepest level rendered in the previous haptic cycle use the *warm threshold* V_W, whereas all deeper levels use the *cold threshold* $V_C < V_W$. A level i is rendered if the total number of tree nodes visited before entering level i plus the size of traversal queue for level i (as determined by the just completed level $i-1$) is less than the threshold for level i. Cold threshold is stricter to prevent popping back and forth among two levels during consecutive haptic cycles; we set $V_C = 0.8V_W$. Our LOD control therefore has a hysteresis, similar to, say, a thermostat controlling a heater. In practice, we have seen LOD changes at most about 2-3× per second, and did not feel any popping haptic rendering artifacts. LOD changes can be made fewer by reducing V_C. We selected V_W manually by running the simulation and observing computation times per haptic cycle. The selection was not difficult and could be automated.

IV. Deformations

In this section, we describe the necessary algorithmic modifications to support deformations. In our rigid vs deformable simulations, we render a deformable pointshell against a rigid distance field. For deformable vs deformable simulations, we render a deformable pointshell against a deformed distance field.

A. Deformable pointshell

Our pointshell is embedded into a deformable object; as the object deforms, so does the pointshell. Our method is not specific to a particular deformable object algorithm, as long as this algorithm has the following properties: (1) it runs at haptic rates, (2) given any particular tree node queried by our runtime nested tree traversal, it must be able to (quickly) provide the current point world-coordinate location, and a bounding radius for the node subtree, and (3) it must provide the (deformed) normal for points in contact. For small pointshells, these requirements can be satisfied by running any fast deformable object algorithm, and computing the positions, normals, and bounding spheres for all tree nodes at the beginning of every haptic cycle. With large pointshells, it is wasteful (and computationally infeasible) to always compute the positions and normals for all the points, because only a small fraction of the points will usually be queried during the tree traversal. Instead, the requirements (1-3) can be satisfied by running a deformable object algorithm where some simplified dynamics drives the deformations of the (detailed) pointshell. Examples of suitable deformable models are: any coarse deformable model (mass-spring system, FEM, etc.), model-reduced (linear or nonlinear) FEM models, multi-resolution models, articulated rigid or flexible models, and skinned deformable models. We use reduced nonlinear FEM models in our examples; the specifics

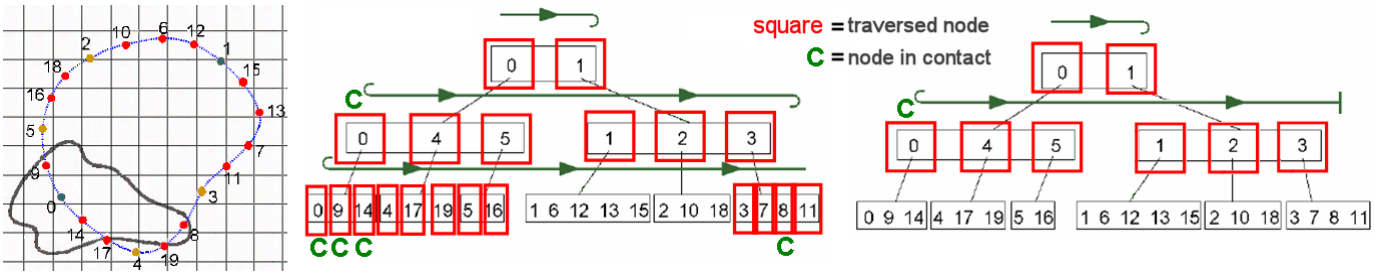

Fig. 8. **Graceful degradation:** With involved contact scenarios (**Left**), many tree nodes need to be traversed during a complete hierarchy traversal (**Middle**). Under graceful degradation (**Right**), tree is traversed only up to a shallow depth, as permitted by available CPU power.

of this model are given in [34]. Regardless of the particular deformable model, the simplified dynamics runs quickly at haptic rates, and individual deformable point positions and normals are computed adaptively as needed by the tree traversal. In particular, the displacement $u \in \mathbb{R}^3$ of a point in any of the listed deformable models can be expressed as $u(t) = U\, q(t)$, where $U \in \mathbb{R}^{3\times r}$ is a certain point-specific time-independent matrix (e.g., FEM shape functions, or model reduction basis functions evaluated at point's material mesh location), $q = q(t) \in \mathbb{R}^r$ is the (relevant part of) the low-dimensional deformation state, and r is the (point-dependent in some deformable models) number of shape functions affecting the particular point. Deforming the normals is important for simulation quality. Continuously deformed normals can be approximated quickly using deformation gradient push-forward of undeformed normals. Such normals can be evaluated using a point-specific normal correction matrix $N \in \mathbb{R}^{3\times r}$, similarly to point positions (details in [34]).

The contact model and the deformable model are linked as follows. During the nested tree traversal, the current deformation is held fixed. The computed contact forces are then set as external forces to the deformable object integrator. The integration timestep is performed at the end of the haptic cycle, producing the deformations for the next haptic cycle.

The bounding spheres of tree node subtrees need to be refitted under deformations. The centers of our spheres always coincide with current point locations; this is computationally convenient so that, in order to determine if a subtree needs to be traversed, the distance field value queried at the point location need only be compared to the bounding sphere radius. For small pointshells, we can refit the entire sphere hierarchy before every haptic cycle. For large pointshells, we refit (conservatively) only the sphere radii of traversed nodes, using a point-based modification of the Bounded Deformation Tree (BD-Tree) [5], [34]. The subtle but algorithmically key point is that our point-based BD-Tree uses nested pointshell points as deformed sphere centers. This seemingly trivial choice enables time-critical testing of the point-based BD-Tree against a distance field: updating a BD-Tree sphere also updates a deformed pointshell point, p, which can in turn be immediately evaluated against the distance field, and if in contact ($d(p) < 0$) will contribute to the progressive accumulation of contact forces in a multi-resolution manner. Therefore, even in the expected case where there is insufficient time to completely test the BD-Tree against the distance field, approximate contact forces can still be accumulated in a time-critical fashion.

B. Deformed distance field

Our real-time contact algorithm performs over a thousand signed-distance-field evaluations per millisecond-long haptic frame. Distance fields for complex rigid objects can be preprocessed and runtime-sampled quickly, however the same is not true of the deformable distance fields needed to support "defo-defo" contact. In this section, we introduce an algorithm which can approximate signed distance values to deformed complex geometry very quickly, provided that the deformations themselves are reasonably coarse (low-frequency). Our approach is similar to material-space methods [35], but does not require a bounding volume hierarchy and can report distance field approximations outside of the object.

If distance field computation (evaluation) costs were zero or negligible, one could simply treat the deformed distance field as an oracle to provide the exact distance $d(p)$. On today's hardware, however, distance field evaluation for deformable models is expensive and can not be performed at haptic rates for non-trivial geometry [36], [37]. Even if an acceleration datastructure such as a bounding volume hierarchy was computed (re-fitted) to the deformed triangle mesh before each haptic cycle, each individual point distance query requires $O(\log(n))$ search time, where n is the number of triangles in the model. Deformed distance fields can be computed on *grids* in appropriate regions of interest using methods such as Fast Marching [38], or 3D scan conversion [39] (also see [40] for a survey). These methods recompute the distance field from scratch, i.e., they are general-purpose distance field computation algorithms, invoked separately with each deformed configuration. They can not update the distance field at hard real-time rates for complex geometry as their computation times depend on the number of model triangles and the distance field resolution. Fast marching has also been applied to unstructured locations inside the object [41], however marching still requires near-uniform spatial distribution of the distance field query locations, known at the beginning of marching. In our work, we need the distance only at the sparse locations of traversed tree nodes, most of which will be outside the object, and all of which only become known progressively during the traversal.

Our method is related to the "material-space distance" method [35] which we briefly describe next. In [35], a deformed distance field is approximated by re-using a rest configuration precomputed distance field. No particular structure in query locations is required, enabling convenient output-sensitive random access to the signed distance field. The deformable object carries a simulation mesh, such as a tetrahedral mesh or a hexahedral mesh. Given the current mesh deformation, a bounding volume hierarchy is (re-)fitted to the tetrahedra. Then, for a query location p, one

can use the hierarchy to quickly locate the tetrahedron containing p, or detect that p is not contained in any tetrahedron. If inside a tetrahedron, one uses p's barycentric weights to pull p back to the rest configuration, obtaining the corresponding material location P. A distance value estimate is then obtained by looking up P into the precomputed signed distance field. Note that the tet mesh is typically coarser than the surface triangle mesh, or else it is advantageous to compute the distances simply by maintaining a hierarchy on the surface triangle mesh. The method can not report distances outside of the tetrahedral mesh, and was only used in offline simulations. It requires maintaining a bounding volume hierarchy on the deforming tet mesh, and a traversal of the hierarchy for each distance field query.

We propose a modification of the material-space distance method that does not use a volumetric (tetrahedral) mesh and does not require a deformable bounding volume hierarchy. It can also compute distance approximations outside of the object. During pre-process, we compute a signed distance field with respect to the undeformed configuration. At runtime, we compute fast approximations to the deformed distance field values $d(p)$ at the locations of traversed pointshell points. We do so by using first-order local deformation models at a small number of selected locations (the "proxies"). We interpolate these models to pull pointshell point positions back to the undeformed configuration of the distance field object, followed by a lookup into the precomputed distance field.

The specifics of the method are as follows. During pre-process, we fit a small pointshell (typically $\sim$40 points) to the surface of the *distance field object*. We call this pointshell the *proxyshell* and its points the *proxies*. We fit the proxies using the same algorithm that was used to fit the points to the surface of the pointshell object. Proxies are embedded into (deform together with) the deformable distance field object. To evaluate the deformed distance field at some query pointshell point location x, we first perform a k-nearest neighbor search to locate the k current closest proxies $p_1, \ldots, p_k$ to x (see Figure 9, Left, Middle). We then assign non-negative weights to p_i; closer proxies are given larger weights, and the weights sum to one; we use the scheme described in [42] to assign weight w_i to proxy i,

$$w_i = \frac{w'_i}{W}, \qquad w'_i = \frac{1/d_i - 1/d_{\max}}{1/d_{\min} - 1/d_{\max}}, \qquad W = \sum_{i=1}^{k} w'_i, \quad (3)$$

where $d_i = \|x - p_i\|_2$ and $d_{\max}$ ($d_{\min}$) is the max (min) of d_i over $i = 1, \ldots, k$. We typically set $k = 5$ in our implementations to avoid singular configurations where $d_{\min} = d_{\max}$ (five points in general do not lie on the same sphere in 3D).

We use local first-order deformation models at p_i to generate an approximation to the deformed distance field at x, as follows. Let P_i be the material position of p_i; then the deformation transformation in the vicinity of P_i can be approximated as

$$X \mapsto x = p_i + F_i(X - P_i), \quad (4)$$

where X is an arbitrary material point sufficiently close to P_i, and $F_i \in \mathbb{R}^{3\times 3}$ is the deformation gradient at P_i. Deformation gradients can be evaluated quickly. For example, for (coarse) FEM models or reduced FEM models this can be achieved in $9r$ flops, by taking analytical gradients of the shape (or basis) functions, and precomputing deformation gradient mode matrices, $(F_i)_j$, for $j = 1, \ldots, r$, where r is the number of shape functions affecting the deformation at P_i [34]. We then pull the query position x back to X_i, by inverting equation 4 (see Figure 9, Right). Next, we obtain d_i, the distance with respect to proxy i, by looking up the undeformed field at X_i. Our distance approximation is $d = \sum_{i=1}^{k} w_i d_i$. Note that the weights are continuous, and vanish at the furthest nearest neighbor, ensuring C^0 continuity of d both with respect to x and the deformations of the distance field object.

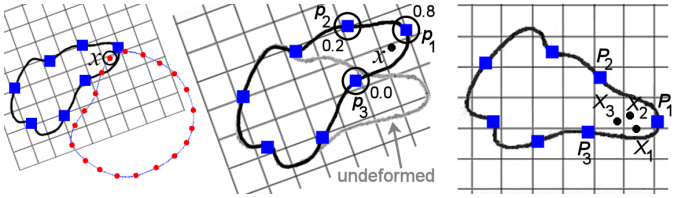

Fig. 9. **Deformed distance field approximation:** (**Left**) Proxies (squares), the query pointshell point at x; (**Middle**) 3-nearest neighbors and their weights; (**Right**) material configuration pull-backs X_i of x.

The resulting contact forces are applied (with opposite sign) to both the pointshell deformable object and the distance field deformable object. For the pointshell object, the force is applied at the location of the point. For the distance field object, the material location of the force is known; however, the simulation mesh element (or nearest simulation mesh vertex) containing this material location is not known a priori. We apply the force to the locations of the nearest proxy which had already been identified during the k-nearest neighbor query. Force location could be improved by a bounding volume hierarchy search for each point in contact, exploiting the fact that only a small number of all the traversed points will be in contact at any moment of time.

Our scheme can be seen as approximating the global deformation transformation (both inside and outside of the object) by interpolating (and extrapolating) local first-order deformation models. It is capable of computing a fast approximation to the deformed distance field with respect to *complex* geometry. This is possible because the complex geometry is encoded into the distance field in the undeformed configuration. The k-nearest neighbor search is fast since it is performed only on a small number of proxy points. Our approximation benefits from spatially smooth deformable models, and becomes progressively less accurate under stretching/compression. Our proxies are uniformly distributed across the model. Proxy positions could be chosen adaptively such that more proxies are used in regions with localized deformations or larger changes in deformation gradients.

Multiple domains: The basic deformable distance field scheme can suffer from spurious contact regions when two nearby parts deform away from each other: "mirror" images of one part appear in estimates d_i under proxies on the other part (see Figure 10, Middle). Similar "ghosting" has also been reported with feature-based 2D image morphing techniques [43], where a morph between two images is computed from a collection of transformations of pairs of line segments. These artifacts can be reduced by dividing the distance field object into several domains, precomputing a separate distance field for each domain, performing the single-domain scheme with respect to each domain, and taking the minimum (see Figure 10, Right). Mirroring essentially introduces "extra" geometry, which typically pushes the isocontours of the approximate field further out than the true isocontours, leading to conservative distance estimates. While we can not *guarantee* conservative distance estimates, we

did not encounter problems with our examples. Although more costly, deformable distance field accuracy might be improved under compression deformations by pushing forward material-space closest-point information [44].

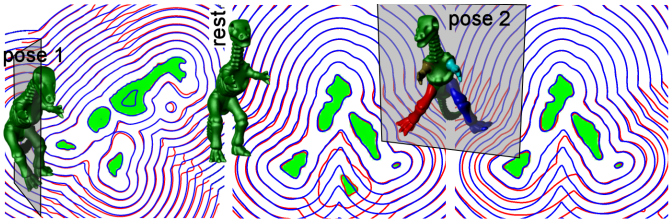

Fig. 10. **Isocontours of the exact and approximate deformed distance field:(Left)** Pose 1, single-domain approximation; (**Middle**) pose 2, single-domain approximation with the two legs mirroring; (**Right**) pose 2, multiple domain approximation without mirroring. Green denotes the interior of the object, with respect to the approximated field.

V. Haptic Display of Distributed Contact

Until now we have abstracted 6-DoF haptic rendering as simply a matter of computing contact forces and torques at high rates. However, penalty forces and torques can not simply be sent to the haptic device: each point in contact adds to the overall stiffness of the system and if enough points are added, maximum renderable stiffness of the haptic device is easily exceeded. Instead, it is customary to separate the *simulation* position and orientation of the haptic object from the position and orientation imposed by the haptic manipulandum, and connect the two with a generalized spring(*virtual coupling* [45]). This penalty spring of stiffness k_{VC} tries to align the simulation position and orientation of the haptic object to the manipulandum. We adopt *static* virtual coupling of [2] since it needs a relatively small number of parameters (unlike [1]), and does not need device velocity estimates. We extend static coupling by introducing quasi-static damping. We also include large rotation terms in the static equilibrium equations (details in [34]).

In every haptic cycle, we first read the position and orientation of the haptic device manipulandum. We then compute contact penalty forces and torques (by traversing the nested point-tree), virtual coupling forces and torques, *and their gradients* with respect to the *simulation* position and orientation of the haptic object (see Figure 4). The net force and torque on the simulation object, under a (small) incremental displacement Δx and (small) incremental rotation $\Delta\omega \in \mathbb{R}^3$, are then

$$F_{\mathrm{net}} = F_{\mathrm{VC}} + F_{\mathrm{C}} + \Big(\frac{\partial F_{\mathrm{VC}}}{\partial x} + \frac{\partial F_{\mathrm{C}}}{\partial x}\Big)\Delta x + \Big(\frac{\partial F_{\mathrm{VC}}}{\partial \omega} + \frac{\partial F_{\mathrm{C}}}{\partial \omega}\Big)\Delta\omega \tag{5}$$

$$T_{\mathrm{net}} = T_{\mathrm{VC}} + T_{\mathrm{C}} + \Big(\frac{\partial T_{\mathrm{VC}}}{\partial x} + \frac{\partial T_{\mathrm{C}}}{\partial x}\Big)\Delta x + \Big(\frac{\partial T_{\mathrm{VC}}}{\partial \omega} + \frac{\partial T_{\mathrm{C}}}{\partial \omega}\Big)\Delta\omega \tag{6}$$

where $F_{\mathrm{VC}}, T_{\mathrm{VC}}$ denote current virtual coupling force and torque [2], and $F_{\mathrm{C}}, T_{\mathrm{C}}$ denote the sum of all contact forces and torques. We displace the virtual object such that the net force and torque on the object vanish under this first-order model, by solving the non-symmetric 6x6 linear system of Equations 5,6 for $(\Delta x, \Delta\omega)$ using LU decomposition (fast, $\sim 5\mu s$ per solve). Finally, we compute the virtual coupling force and torque with respect to the new simulation position, and render them to the user. Virtual coupling gradients are determined through analytical differentiation of the virtual coupling forces and torques. Gradients of contact forces and torques are computed by treating each contact point as undergoing contact with an infinite halfspace (wall), oriented according to the contact normal (details in [34]).

In our deformable distance field simulations, there are three haptic object configurations at any moment of time: the manipulandum position and orientation, the rigid simulation position and orientation (blue color in Figure 13), and the complete simulation configuration where the haptic object's deformations are added to the rigid simulation position and orientation. The virtual coupling forces and torques are computed with respect to the manipulandum and rigid simulation positions and orientations. Rigid body motion of the deformable haptic object is set to the static equilibrium between virtual coupling forces and the defo-defo contact forces determined during the tree traversal, in the same way as with rigid haptic objects. The deformation of the haptic object's distance field is held fixed during the tree traversal. One timestep of haptic object's deformations is performed at the end of each haptic cycle, using the defo-defo contact forces as external forces. Note that our haptic object deformations are decoupled from rigid body motion; this approximation is very common in computer graphics for its simplicity and plausible results [5].

A simple calculation shows that the maximum stiffness felt by the user if ℓ points are in contact is $1/(1/(\ell \cdot k_{\mathrm{C}}) + 1/k_{\mathrm{VC}})$; therefore k_{VC} presents an upper bound on the stiffness ever rendered to the haptic device, regardless of ℓ. Since time is sampled discretely, simulation stability is improved if k_{C} is scaled down when there are multiple points in contact; as in VPS, we replace k_{C} with $k_{\mathrm{C}}L/\ell$ whenever $\ell \geq L$ (we use $L = 10$). The displacement-force relationship of virtual coupling is designed to be linear only up to a certain displacement, after which it saturates to some maximum force value (c.f. virtual proxy [46], [47]).

Quasi-static damping: The lack of dissipation in the static virtual coupling model can lead to slight instabilities, such as during fast sliding contact. We augmented the static virtual coupling model by introducing a quasi-static damping, or a damped state update: after $\Delta x, \Delta\omega$ are computed, we only change the simulation position and rotation by $(1-\alpha)\Delta x, (1-\alpha)\Delta\omega$, where $\alpha \in [0, 1)$ controls the amount of static damping. Such damping exponentially restores the simulation position to the manipulandum position, similar to critical damping in mass-spring-damper dynamic virtual coupling models [1]. We set $\alpha = 0.5$ in all our simulations; high values of α are avoided since they introduce a surface stickiness effect.

VI. Results

Statistics on all haptic rendering examples are provided in Table I. Haptic demos (executables for Sensable's Phantom Premium 6DOF, 3DOF, and Omni devices) are freely available at: http://graphics.cs.cmu.edu/projects/defoContact/demo/index.html.

In our first 6-DoF haptic rendering example, we simulate a detailed deformable bridge in contact with a detailed rigid dinosaur (see Figure 2). We recorded a characteristic manipulandum trajectory, and report simulation data under this trajectory computed offline under different force approximations (see Figure 11). We also present a detailed deformable dragon in contact with a rigid Buddha mesh. Figure 12 gives a haptic gaming example demonstrating that pointshells with a million points can be simulated when our method is used for rigid simulations only.

TABLE I

MODEL STATISTICS FOR REPRESENTATIVE CONTACT CONFIGURATIONS

Contact case; pointshell+distance field	Contact depicted in	Pointshell		Dist. field resolution	Traversed nodes		Points in contact	LOD	Timings [μs]				
		levels	points		with TC	TC off			VC	StVK	TT	total	TL
knight(r) + axe(r)	Results	6	1.02 M	256	2050	3400	820	6	90	0	780	870	80,000
bridge(d) + dinosaur(r)	Figure 2	4	85K	256	1989	5200	426	4	90	45	710	845	12,200
CAD scene 1(d) + metal part(r)	Figure 3	4	80K	128	530	1520	50	4	90	45	125	260	6,600
dragon(d) + Buddha(r)	Figure 4	5	256K	256	1750	2900	650	5	90	45	615	750	29,800
dragon(r) + dragon hole(r)	Figure 16	5	256K	256	1701	2642	843	2	90	0	722	812	45,000
CAD scene 2(d) + hose(d)	Figure 13	5	256K	256	537	1720	40	5	90	190	507	787	210,000
dragon(d) + dinosaur(d)	Figure 13	5	256K	256	245	1390	24	3	90	90	531	711	590,000

Timings generated on an Intel Core 2 Duo 2.66 GHz processor with 2 Gb RAM. All deformable models use $r = 15$ deformation modes, except *CAD scene 2 + hose* ($r = 20$ for both models). LEGEND: r=rigid, d=deformable, VC=virtual coupling and manipulandum transformations, StVK=deformable FEM dynamics, TT=tree traversal, TC=temporal coherence, LOD=level of detail rendered (in the particular configuration), TL=time under a tree-less pointshell traversal.

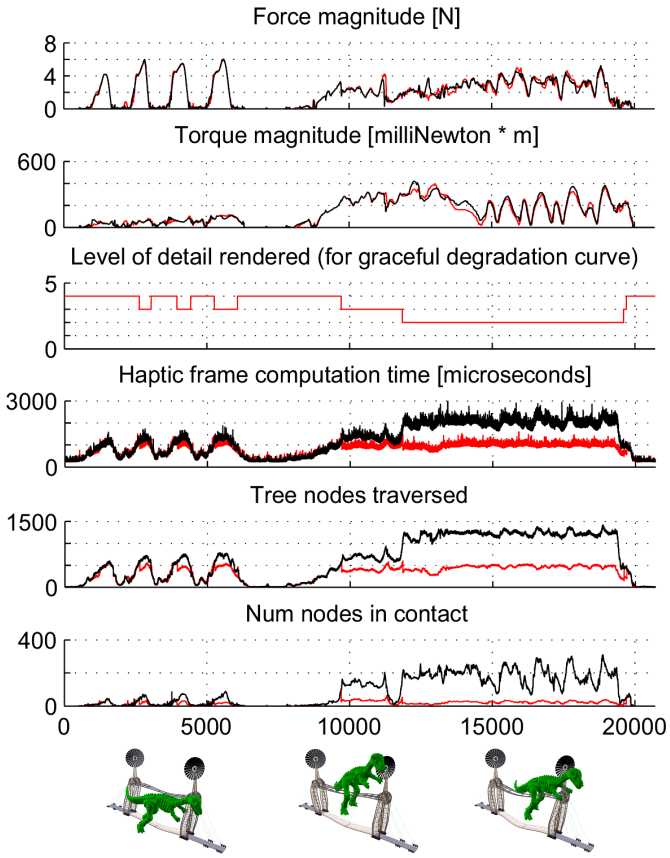

Fig. 11. **Illustration of graceful degradation (GD):** Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 600 nodes; without GD (in black) with all four tree levels. The pointshell has $85,018$ points total. The common x-axis corresponds to haptic frames. This data was recorded on a three-year-old 3.0 GHz dual Intel Xeon processor with 2.75 Gb of memory, illustrating algorithm's adaptivity to different computer speeds. On the (faster) machine of Table I, the black timing curve reaches a maximum of $\sim$800 μs.

We demonstrate "defo-defo" contact by simulating a deformable dragon in contact with a deformable dinosaur, and a deformable mechanical structure in contact with a deformable hose (Figure 13). Figure 16 demonstrates the benefits of graceful degradation to simulate very large contact areas. Interactive (dis)assembly and path planning are illustrated in Figure 3 by example where the user can manipulate a metallic rigid component in contact with deformable hose geometry typical of modern airplanes. Large hose deformations allow the operator to interactively navigate the component in between the hoses.

Fig. 12. **Interactive haptic gaming** is an emerging application area. In this example, the user can swing a detailed rigid axe to "interact" with an armored rigid knight. This example demonstrates crisp contact with detailed million-point pointshells possible when our method is used for rigid-only simulations. The pointshell is resolved at the deepest level when poking the knight with the blade of the axe. Our graceful degradation algorithm reduces the LOD to maintain haptic update rates if the axe is positioned in flat contact against the knight.

VII. CONCLUSION

We have demonstrated that real-time time-critical haptic rendering of distributed contact between geometrically complex models is possible for both rigid and reduced deformable models. A key ingredient has been the exploitation of low-dimensional parameterizations of deformable models for fast dynamics, point-based deformation bounds, and output-sensitive evaluation of pointshell points and contact normals.

Discussion: Our deformations are reduced and not fully general; in the future, fully general complex FEM deformable objects might run at haptics rates on commodity workstations, but this is not the case today. Our deformable model is adaptive: more basis vectors can be added to the basis as processor speeds will increase. While our current demonstrations run on standard Windows or Linux platforms, simulation on operating systems and parallel architectures that can guarantee hard real-time scheduling is a natural area of exploration. The simulation can be simplified if the signed distance field is replaced by a simpler implicit function with an analytical formula. Our offset surfaces are not reaching down to the exact geometry in case of "polygon soup" input geometry, when the contact surface is not well-defined. Quality of the offset surface degrades under offsets too small compared to distance field resolution. Large deformations can potentially introduce undersampled pointshell surface regions; however this was not problematic in our case of deformations mostly consisting of large rotations and small strain.

Distance fields can be memory-consuming; for example, a 256x256x256 32-bit floating point distance field occupies about 64 Mb of memory. This was not a limiting factor in our implementations where the manipulated object carries a distance field. In virtual assembly simulations, the manipulated object is often

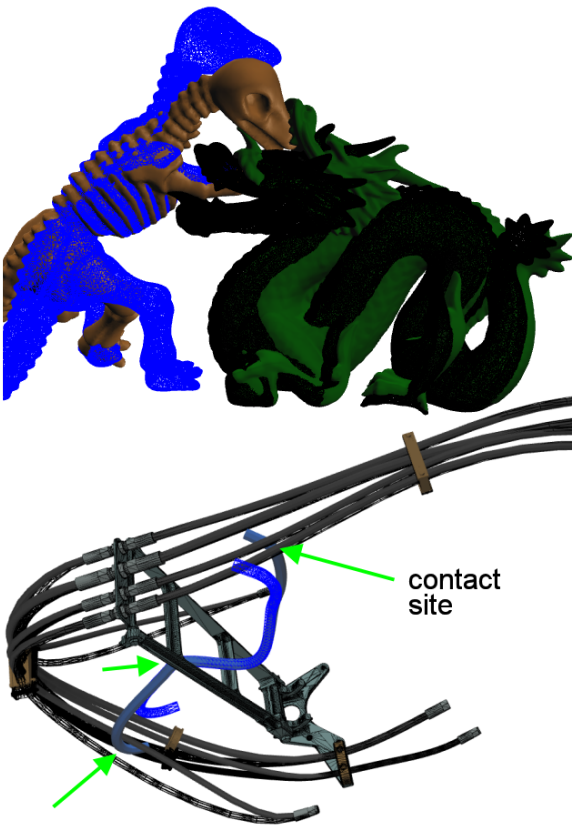

Fig. 13. **Deformable vs deformable contact:** (**Top**) A deformable dinosaur (haptic object; deformable distance field; five domains; $k = 5$) in contact with a deformable dragon (rooted to ground; deformable pointshell). (**Bottom**) A deformed hose (haptic object; deformable distance field; one domain; $k = 5$) in contact (at three separate locations, indicated by green arrows) with a deformed mechanical structure (rooted to ground; deformable pointshell). Blue=deformable distance field object rigid configuration; black=pointshell object rigid configuration.

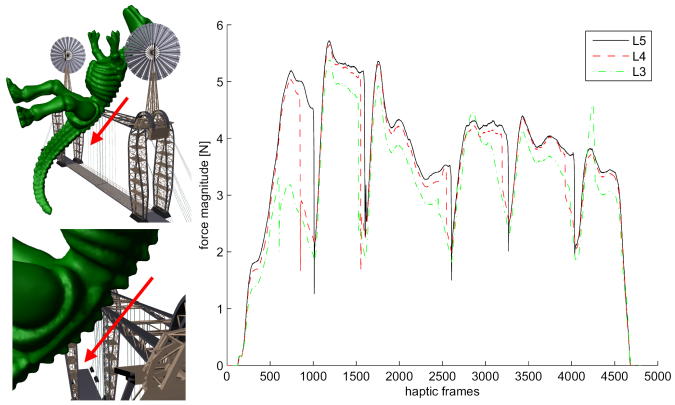

Fig. 14. **Our method can render small features:** (**Left (top and bottom)**) Resolution in our method is high enough for the user to feel the bumps on the back of the rigid dinosaur (distance field object, 256x256x256), by sliding the back of the dinosaur against the upper shelf of the bridge (pointshell object). The red arrows are indicating the direction of the dinosaur's sliding. Both objects are rigid in this experiment. Bridge pointshell has five levels of detail with 256,000 points. (**Right**) The force magnitude rendered to the user, under progressively finer pointshells and a pre-recorded manipulandum trajectory. Sampling rate is 1000 frames per second. The user caused six bumps total to touch the bridge (one after another, in tangential motion, while simultaneously also pushing the dinosaur in a normal direction against the bridge). The impact of each individual bump can be clearly seen in the haptic signal, and felt in the haptic simulation.

the tool; greater accuracy is typically needed to represent the virtual environment (pointshell object in our case). While we did not pursue out-of-core approaches that would enable one, say, to render the interior of an entire airplane, we believe our nested point hierarchy would be very effective in quickly ruling much of the airplane out of collision. The point hierarchy could be preceded with a few coarse standard hierarchy levels to cover major individual airplane parts before diving into the individual point hierarchies. To reduce distance-field memory, a hybrid data structure would be possible, using a wide-bit voxmap in free space, and a distance field accessed via a hash table in the shallow contact force layer surrounding the object. Distance field could also be sampled adaptively [48].

Our rendered level of detail is selected globally for the entire object. In some situations, it might be advantageous to be able to select LODs locally, for example, if the haptic object touches a large flat panel on one side, and a thin wire on the other side. Such an algorithm would need to minimize the number of LOD transitions and could employ a perception error metric [6] to balance accuracy against performance. Although our implementation is not highly optimized, we did implement some time-critical routines in assembly (SSE2 multimedia instructions), and carefully laid out data in memory to optimize cache performance, resulting in a system speedup of about $2\times$. Perceptual studies may be useful in determining speed-accuracy trade-offs, such as maximum necessary contact resolution or perception of deformation complexity, similar to other studies done in rigid-rigid contact rendering [25]. Another interesting area of future work is simulating friction, which can play an important role in applications such as analysis of mechanical part removability and accessibility. With reduced nonlinear FEM models, some applications may be limited by the fact that modification or cutting of the models is prohibited by the preprocessing employed for performance. Although unaddressed in our implementation, large deformations can necessitate self-collision processing, and this remains a significant computational challenge for hard real-time *haptic* rendering applications (c.f. [49]).

Acknowledgment

This research was sponsored by the Boeing Company, the National Science Foundation (CAREER-0430528), the Link Foundation, Pixar, Nvidia, Intel, and donation of Maya licenses from Autodesk.

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. 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.

[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, "Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models," *ACM Transactions on Graphics (SIGGRAPH 2005)*, vol. 24, no. 3, Aug. 2005.

[5] D. L. James and D. K. Pai, "BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 393–398, 2004.

[6] M. A. Otaduy and M. C. Lin, "A Modular Haptic Rendering Algorithm for Stable and Transparent 6-DoF Manipulation," *IEEE Trans. on Robotics*, vol. 22, no. 4, pp. 751–762, 2006.

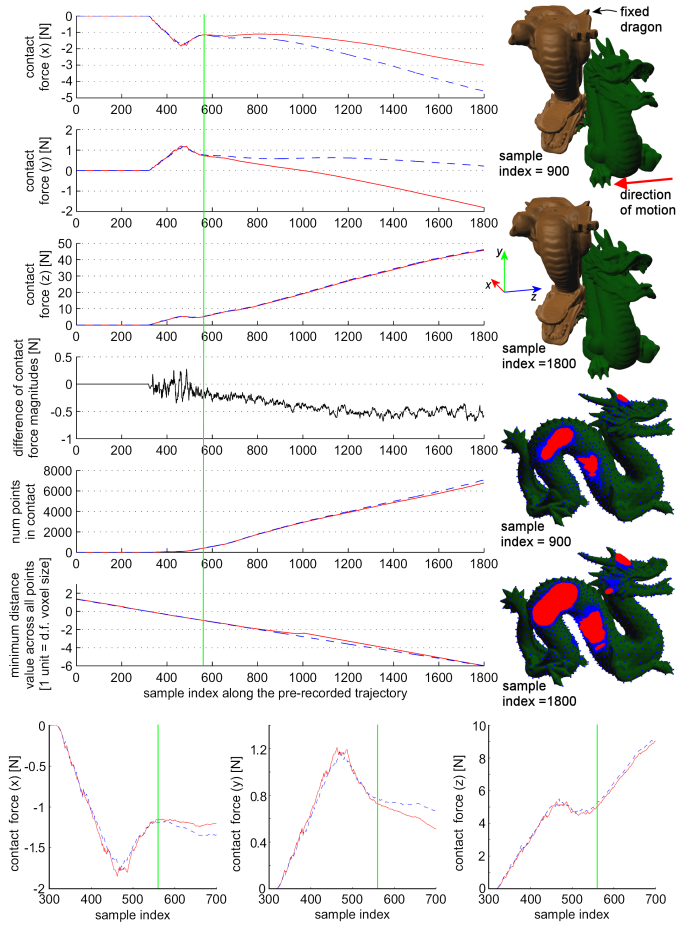

Fig. 15. **Asymmetric contact model:** The contact force changes under role reversal, however the difference is reasonably small. The first contact case (rigid green dragon=pointshell, rigid brown dragon=signed distance field) is depicted with a solid red line, the reverse case (rigid green dragon=signed distance field, rigid brown dragon=pointshell) is shown in dashed blue. Both lines show the contact force (without virtual coupling) on the green (i.e., moving) dragon. The top-right image shows the configurations at samples 900 and 1800, respectively. The bottom-right images depict points in contact (red=in contact, blue=traversed by tree, but not in contact). Note that the z-axis is aligned with the direction of motion, and is (about) normal to the contact areas; z-force is therefore the normal contact force and is as such substantially larger than tangential forces. Penetration depths are large in this experiment; in our simulations, virtual coupling coupling saturation would prevent penetrations deeper than about 1 voxel. The vertical green line denotes the sample index where penetration reaches 1 voxel. The bottom image shows the shallow penetration part of the top image in greater detail. Contact force differences can be seen to be small.

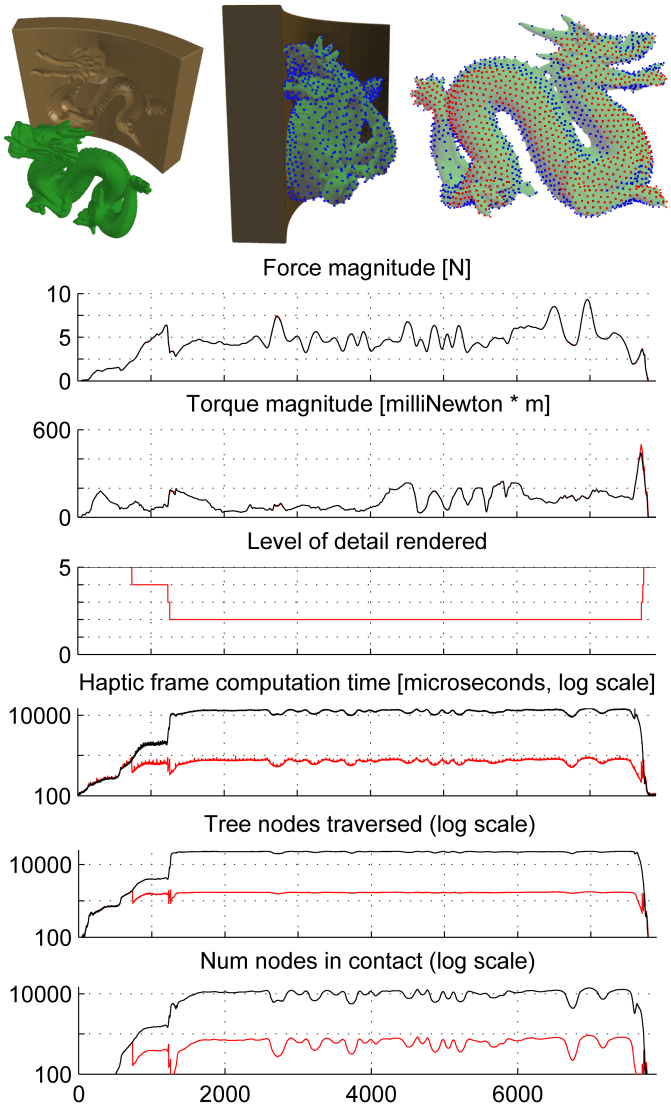

Fig. 16. **Graceful degradation supports large conforming contacts** to allow time-critical approximation of very difficult contact configurations (here under 1 millisecond). (**Top Left**) A dragon "peg" inserting into a matching dragon "hole" obtained via CSG difference. (**Top Middle**) Dragon manipulated haptically into the hole. (**Top Right**) Large contact area with traversed points in blue (43% of all L2 points), and contacting points in red (21% of all L2 points). (**Bottom Graphs**) Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 2000 nodes; without GD (in black) with all five tree levels. Same machine as in Table I. The common x-axis corresponds to haptic frames.

[7] M. A. Otaduy and M. Gross, "Transparent Rendering of Tool Contact with Compliant Environments," in *Proc. of the World Haptics Conference*, 2007, pp. 225–230.

[8] S. Laycock and A. Day, "A Survey of Haptic Rendering Techniques," *Computer Graphics Forum*, vol. 26, pp. 50–65, 2007.

[9] U. Kühnapfel, H. Çakmak, and H. Maaß, "Endoscopic surgery training using virtual reality and deformable tissue simulation," *Computers & Graphics*, vol. 24, pp. 671–682, 2000.

[10] O. Astley and V. Hayward, "Multirate Haptic Simulation Achieved by Coupling Finite Element Meshes Through Norton Equivalents," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1998.

[11] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr, "Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling," in *Proc. of ACM SIGGRAPH 2001*, August 2001, pp. 31–36.

[12] C. Mendoza and C. O'Sullivan, "Interruptible collision detection for deformable objects," *Computer & Graphics*, vol. 30, no. 2, 2006.

[13] M. Bro-Nielsen and S. Cotin, "Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation," *Comp. Graphics Forum*, vol. 15, no. 3, pp. 57–66, 1996.

[14] S. Cotin, H. Delingette, and N. Ayache, "Realtime Elastic Deformations of Soft Tissues for Surgery Simulation," *IEEE Trans. on Vis. and Comp. Graphics*, vol. 5, no. 1, pp. 62–73, 1999.

[15] D. L. James and D. K. Pai, "A Unified Treatment of Elastostatic Contact Simulation for Real Time Haptics," *Haptics-e, The Electronic J. of Haptics Research (www.haptics-e.org)*, vol. 2, no. 1, September 2001.

[16] ——, "Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects," *ACM Trans. on Graphics*, vol. 22, no. 1, pp. 47–82, 2003.

[17] S. Jun, J. Choi, and M. Cho, "Physics-based s-adaptive haptic simulation for deformable object," in *Proc. of the Symp. on Haptic Interfaces For Virtual Environment and Teleoperator Systems*, 2006, pp. 72–78.

[18] 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.

[19] S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *Proceedings of ACM SIGGRAPH 2000*, ser. Computer Graphics Proceedings, Annual Conference Series, Jul. 2000, pp. 343–352.

[20] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Contact Handling for Deformable Point-Based Objects," in *Proc. of Vision, Modeling, Visualization 2004*, 2004, pp. 339–346.

[21] M. Pauly, D. K. Pai, and L. Guibas, "Quasi-Rigid Objects in Contact," in *Proc. of the Symp. on Comp. Animation 2004*, 2004.

[22] P. M. Hubbard, "Collision Detection for Interactive Graphics Applications," Ph.D. dissertation, Department of Comp. Science, Brown University, 1995.

[23] 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.

[24] M. Ortega, S. Redon, and S. Coquillart, "A Six Degree-of-Freedom God-Object Method for Haptic Display of Rigid Bodies with Surface Properties," *IEEE Trans. on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 458–469, 2007.

[25] M. A. Otaduy and M. C. Lin, "Sensation Preserving Simplification for Haptic Rendering," in *Proc. of ACM SIGGRAPH 2003*. ACM, 2003, pp. 543–553.

[26] 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.

[27] N. Galoppo, S. Tekin, M. A. Otaduy, , M. Gross, and M. C. Lin, "Haptic Rendering of High-Resolution Deformable Objects," in *Proc. of 12th International Conference on Human-Computer Interaction*, 2007.

[28] J. Barbič and D. L. James, "Time-critical distributed contact for 6-DoF haptic rendering of adaptively sampled reduced deformable models," in *Proc. of the Symp. on Comp. Animation 2007*, July 2007, pp. 171–180.

[29] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, "Efficient implementation of Marching Cubes' cases with topological guarantees," *Journal of Graphics Tools*, vol. 8, no. 2, pp. 1–15, 2003.

[30] C. Shen, J. F. O'Brien, and J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup," in *Proceedings of ACM SIGGRAPH 2004*. ACM Press, Aug. 2004, pp. 896–904.

[31] G. Turk, "Re-Tiling Polygonal Surfaces," *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, vol. 26, no. 2, pp. 55–64, 1992.

[32] A. P. Witkin and P. S. Heckbert, "Using particles to sample and control implicit surfaces," in *Proceedings of SIGGRAPH 94*, ser. Computer Graphics Proceedings, Annual Conference Series, Jul. 1994, pp. 269–278.

[33] G. Turk, "Texture Synthesis on Surfaces," in *Proceedings of ACM SIGGRAPH 2001*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 2001, pp. 347–354.

[34] J. Barbič, "Real-time Reduced Nonlinear Deformable Objects for Computer Graphics and Haptics," Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, Aug. 2007.

[35] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs, "An implicit finite element method for elastic solids in contact," in *Proc. of Computer Animation, Seoul, South Korea, 2001*, 2001.

[36] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha, "Interactive 3D Distance Field Computation using Linear Factorization," in *Proc. ACM Symposium on Interactive 3D Graphics and Games (I3D)*, 2006.

[37] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha, "Fast Proximity Computation Among Deformable Models using Discrete Voronoi Diagrams," in *Proc. of ACM SIGGRAPH 2006*, 2006.

[38] S.Fisher and M.C.Lin, "Fast penetration depth estimation for elastic bodies using deformed distance fields," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001, pp. 330–336.

[39] C. Sigg, R. Peikert, and M. Gross, "Signed Distance Transform using Graphics Hardware," in *Proceedings of IEEE Visualization Conference*, 2003, pp. 83–90.

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

[41] D. Marchal, F. Aubert, and C. Chaillou, "Collision Between Deformable Objects Using Fast-Marching on Tetrahedral Models," in *Proc. of the Symp. on Comp. Animation 2004*, Aug 2004.

[42] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen, "Unstructured lumigraph rendering," in *Proceedings of ACM SIGGRAPH 2001*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 2001, pp. 425–432.

[43] T. Beier and S. Neely, "Feature-Based Image Metamorphosis," *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, vol. 26(2), pp. 35–42, July 1992.

[44] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw, "Robust Quasistatic Finite Elements and Flesh Simulation," in *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Jul. 2005, pp. 181–190.

[45] J. Colgate, P. Grafing, M. Stanley, and G. Schenkel, "Implementation of Stiff Virtual Walls in Force-Reflecting Interfaces," in *Proc. of IEEE Virtual Reality Annual Int. Symp.* IEEE, 1993, pp. 202–208.

[46] 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.

[47] D. C. Ruspini, K. Kolarov, and O. Khatib, "The Haptic Display of Complex Graphical Environments," in *Proc. of ACM SIGGRAPH 97*, T. Whitted, Ed. Addison Wesley, Aug. 1997, pp. 345–352.

[48] S. Frisken, R. Perry, A. Rockwood, and T. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," in *Proc. of ACM SIGGRAPH 2000*. ACM, 2000, pp. 249–254.

[49] N. K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin, and D. Manocha, "Interactive collision detection between deformable models using chromatic decomposition," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 991–999, Aug. 2005.

Jernej Barbič obtained his PhD from the Computer Science Department at Carnegie Mellon University (2007), and is now a post-doc at the Massachusetts Institute of Technology. Jernej's research interests include real-time simulation, FEM deformable objects, haptic rendering of contact for geometrically rich scenarios, sound simulation, and model reduction and control of nonlinear systems.

Doug L. James obtained his PhD from the Univ. of British Columbia. Doug has been a professor of Computer Science at Cornell since 2006; prior to that, he was a Computer Science and Robotics professor at Carnegie Mellon University (2002-2006). Doug's research interests include computer graphics, physically based animation, computational geometry, scientific computing, dimensional model reduction, computational robotics, and haptic force-feedback rendering.