Constraint-Based Six Degree-Of-Freedom Haptic Rendering of Volume-Embedded Isosurfaces

Sonny Chan François Conti Nikolas H. Blevins Kenneth Salisbury*

Department of Computer Science
Stanford University

ABSTRACT

A method for 6-DOF haptic rendering of isosurface geometry embedded within sampled volume data is presented. The algorithm uses a quasi-static formulation of motion constrained by multiple contacts to simulate rigid-body interaction between a haptically controlled virtual tool (proxy), represented as a point-sampled surface, and volumetric isosurfaces. Unmodified volume data, such as computed tomography or magnetic resonance images, can be rendered directly with this approach, making it particularly suitable for applications in medical or surgical simulation.

The algorithm was implemented and tested on a variety of volume data sets using several virtual tools with different geometry. As the constraint-based algorithm permits simulation of a massless proxy, no artificial mass or inertia were needed nor observed. The speed and transparency of the algorithm allowed motion to be responsive to extremely stiff contacts with complex virtualized geometry. Despite rendering stiffnesses that approach the physical limits of the interfaces used, the simulation remained stable through haptic interactions that typically present a challenge to other rendering methods, including wedging, prying, and hooking.

Index Terms: H.5.2 [Information Interfaces and Presentation]: User Intefaces—Haptic I/O; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

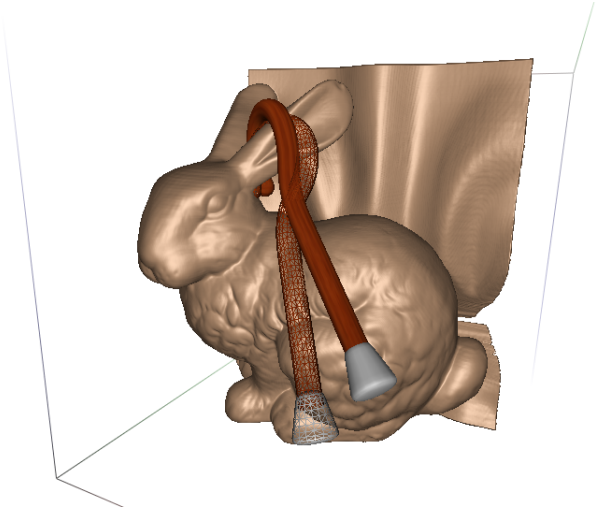


Figure 1: The 6-DOF haptic rendering algorithm allows a user to interact with a CT scan of the Stanford Bunny. The motion of the virtual tool or proxy (shown by a solid mesh) is constrained to the exterior of the volumetric isosurface, even when the configuration of the device (shown by a wire mesh) intersects the object.

1 INTRODUCTION

Physical interaction with the world using a rigid tool is inherently a six degree-of-freedom task. While 3-DOF haptic devices and rendering algorithms have been studied extensively and are readily available, the move to 6-DOF has presented quite a challenge in both the domains of hardware and software. However, in our experience, the addition of torque feedback has a dramatic effect on the perceived realism of interaction with virtual objects, provided that the rendering is done correctly. Our efforts are motivated primarily by our work in the simulation of bone surgery—an application that often demands this extra degree of realism.

This paper presents a new method for six degree-of-freedom haptic rendering of isosurfaces embedded within volumetric data. It allows real-time simulation of rigid-body contact between a geometrically complex virtual tool (represented as point-sampled surface) and isosurface geometry to a sub-voxel precision. The algorithm is designed to operate directly on sampled volume data so that medical images, such as computed tomography (CT) or magnetic resonance (MR) scans, can be haptically rendered and explored without any need for modification. We note that the intrinsic mass and inertia of most haptic interfaces already exceed those of a typical surgical instrument, and thus we adopt a quasi-static, constraint-based approach to simulate the motion of the proxy so that no additional virtual mass or inertia would be imposed on the tool. The passivity of the simulation permits us to render very stiff contacts representative of using a steel instrument to manipulate bone tissue, as often encountered in surgical procedures, up to the physical limits of the haptic device. Finally, the collision detection is designed to be fast (haptic rates) and accurate to the resolution of the isosurface within the volume data, so that the detail present in small, important structures can be represented correctly for applications requiring fine manipulation, such as microsurgery.

1.1 Previous Work

Volume data and volumetric representations of geometry have been no strangers to the field of haptic rendering. As early as 1996, Avila & Sobierajski [1] presented a haptic interaction method suitable for integration with a volume visualization system. Their method provides the ability to touch as well as to modify the volumetric geometry, and they demonstrate its suitability for applications ranging from virtual sculpting to medical simulation.

The first proxy-based haptic rendering algorithms were developed by Zilles & Salisbury [18] and Ruspini et al. [11]. These methods allow 3-DOF haptic interaction with a virtual environment through a point or a spherical "proxy" whose motion is constrained to the surface of the environment geometry, and never allowed to penetrate the objects. Haptic rendering using this constraint-based simulation of the proxy became highly popular because of its ability to render crisp, stable contacts without the "pop-through" problems

*e-mail:jks@robotics.stanford.edu (corresponding author). BioRobotics Laboratory, Suite E100, 318 Campus Dr., Stanford, CA 94305, U.S.A.

IEEE World Haptics Conference 2011
21-24 June, Istanbul, Turkey
978-1-4577-0298-3/11/$26.00 ©2011 IEEE

normally associated with direct rendering approaches. Although proxy-based methods were initially formulated for polygonal geometry, Salisbury & Tarr [13] followed shortly with a constraint-based rendering method for implicit surfaces, which, with a little adaptation, can be used to render geometry within sampled volume data equally well.

One of the first and most successful algorithms for six degree-of-freedom haptic rendering also employed a volumetric representation of the virtual environment geometry. Polygonal geometry was voxelized using 3-D scan conversion to produce a volumetric representation in the method presented by McNeely et al. [5], which facilitated much faster collision detection at the expense of geometric resolution and memory use. However, rather than using a constraint-based solution, they dynamically simulate the motion of the proxy using explicit Euler integration with penalty forces from object interpenetrations, although the algorithm was later improved with a quasi-static simulation [15]. With the advancement of collision detection algorithms, Otaduy & Lin [7] introduced a 6-DOF rendering algorithm for polygonal geometry that employs implicit integration to reduce the stability problems associated with rendering stiff contacts through a dynamic proxy.

Recently, Barbič & James extended some of the early ideas of [5] to introduce a 6-DOF haptic rendering algorithm capable of rendering contact between two geometrically complex deformable models. Finally, Ortega et al. [6] formulated the first direct extension of the constraint-based proxy simulation methods in [18] and [11] to six degrees of freedom for rendering complex polygonal meshes.

1.2 Key Contributions

The key contribution of this work is a new method for 6-DOF haptic rendering of isosurfaces embedded within sampled volume data. The following are distinguishing properties of the algorithm:

- The entire algorithm runs at a haptic update rate of 1000 Hz so that an asynchronous multi-rate simulation is not required.
- A constraint-based approach allows for simulation of distributed (multi-point) contact with a massless proxy, thereby eliminating unwanted inertia and enabling very stiff contact.
- The algorithm operates directly on volumetric data of any type and can render embedded isosurfaces to sub-voxel resolution without the need for any data preprocessing.

This last property yields significant benefits for applications in medical simulation, as entire three-dimensional CT or MR images can be loaded directly, and the isosurface level adjusted interactively, to enable haptic exploration of different anatomical structures that appear in the image data.

In essence, the algorithm presented in this paper is an extension of the implicit surface rendering method described by Salisbury & Tarr [13] to six degrees of freedom. A number of important developments from the past decade of research in 6-DOF haptic rendering are woven together to create a fast, high-fidelity rendering algorithm for volumetric isosurfaces.

2 The Algorithm

2.1 Overview

The role of a haptic rendering algorithm is to simulate the interaction between a haptic tool and a virtual environment, thereby generating contact forces so that virtual objects can be felt. In an impedance-control system, the rendering algorithm continually reads a device configuration (position and orientation) from the haptic device controller and outputs a force—the result of interactions with the environment—to be displayed to the user.

The haptic rendering algorithm maintains a configuration of the *proxy* or virtual tool, which may differ from the actual *device* configuration. In the general sense, impedance rendering presents an optimization problem where the objective is to find the best configuration of the proxy given a configuration of the haptic device, and to display to the user a force dependent on the configurations of both [8]. Figure 2 shows an overview of the haptic rendering algorithm, organized in the component structure introduced in [8].

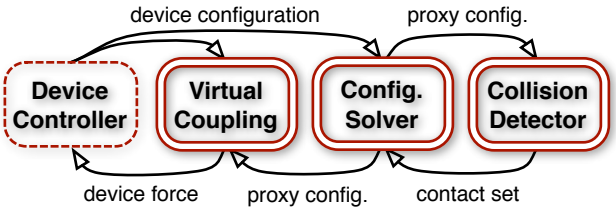


Figure 2: Architectural overview of the haptic rendering algorithm.

The three main components are largely independent from one another in the sense that they perform isolated functions on a set of well-defined inputs to generate a set of outputs suitable for the subsequent stage. The collision detector maintains the geometry of the objects in the virtual environment, and is responsible for detecting interferences between the proxy and the environment in virtual space. The configuration solver maintains the configuration of the proxy, and, given the configuration of the physical device and a set of environmental constraints, it must solve for an optimal configuration of the proxy. Finally, the virtual coupling uses the configurations of the proxy and the device to determine the force and torque displayed to the user. We have selected algorithms we deemed best for the task at hand for each of these components, each of which is described in detail in the subsections that follow.

The goal of this work was to develop a constraint-based 6-DOF algorithm suitable for rendering isosurface geometry within volume data. The initial inspiration for this approach was drawn from Salisbury & Tarr's original method for rendering point interactions with implicit surfaces [13], which can be adapted to render volume-embedded isosurfaces with only minor modifications. Observe that a sampled volume can be generated by evaluating an implicit function at grid locations, and its zero-level isosurface is a sampled version of original implicit surface geometry. Going the other direction, a sampled volume can be evaluated at arbitrary positions using any form of interpolation (e.g. trilinear), but it does lack one important property that most implicit functions possess: an analytic derivative. Instead, the gradient must be estimated by using a discrete approximation scheme such as central differencing.

The configuration solver tracks a 6-DOF proxy instead of a single point, and at each time step it considers a motion that would take the proxy to the device configuration or the next proxy configuration subject to the contact constraints. The rendering algorithm executes the following during every haptic update cycle:

1. Compute the **unconstrained** motion from the proxy configuration to the device configuration. (CS)
2. Verify and prune the current contact set. (CD)
3. Compute a **constrained** motion based on the current contact set and the unconstrained motion direction. (CS)
4. Detect collisions along the proxy's **constrained** path of motion. (CD)
5. Move the proxy along the **constrained** motion path to the first point of contact. (CS)
6. Compute and render the reflected force/torque from the proxy and device configurations. (VC)

Steps labelled (CS) are performed by the configuration solver, those labelled (CD) by the collision detector, and the one labelled (VC) by the virtual coupling.

Figure 3: An isocontour is drawn through grid-sampled voxels on the left. The proxy geometry is represented as a point shell, as shown in the center, which is derived from its polygonal mesh on the right.

2.2 Data Representation

The geometry of the virtual environment is embedded within a sampled volume, and the algorithm requires no modification to the volume data in order to render these embedded isosurfaces. As the sampled volume is taken to represent a continuous intensity field F, the only requirement is the ability to query the volume for an intensity value $F(\vec{x})$ and a gradient $\nabla F(\vec{x})$ at any position $\vec{x}$. Trilinear interpolation of 8 neighboring voxels is used to support querying of intensity values. Given a user-specified isosurface value s, the convention that values of $F(\vec{x}) > s$ is inside the surface and $F(\vec{x}) < s$ is outside is used (Figure 3, left). Sampling is performed with coordinates in physical space so that anisotropic volumes can be used.

A central difference is used to estimate the gradient as

$$\nabla F(\vec{x}) = \begin{pmatrix} F(\vec{x}+\delta_x) - F(\vec{x}-\delta_x) \\ F(\vec{x}+\delta_y) - F(\vec{x}-\delta_y) \\ F(\vec{x}+\delta_z) - F(\vec{x}-\delta_z) \end{pmatrix}, \quad (1)$$

where δ is typically taken to be the sampling interval (voxel spacing) along each axis. This scheme requires 6 interpolated samples per point but provides the best compromise between computation time and the estimate accuracy of the local intensity gradient required to obtain a smooth, continuous surface.

The geometry of the virtual tool is represented as a point-sampled surface (also known as a "point shell" [5]), as shown in Figure 3. This representation is simply a collection of points situated on the surface of the object. A point shell should ideally have its points as equally spaced as possible, although sometimes it is desirable to have a higher concentration of points in areas of greater geometric detail. A surface normal can optionally be associated with each point but is not necessary if the gradient of the isosurface is always assumed to be well-defined.

A virtual tool is typically described by a triangle or polygonal mesh, as that is the representation most suitable for visual rendering. Simply taking the vertices of such a mesh serves as a simple, but often effective, point shell for haptic interaction. However, because the edge and face geometry is lost, parts of the point-sampled surface may become too sparse to convey the shape of the tool effectively. In that case, certain parts of the mesh may need to be subdivided or manually filled before taking a vertex point shell. Remeshing [14] or particle repulsion [17] methods can be used to produce higher quality point shells at the desired level of detail, as shown in [2]. In this work, virtual tools were created for both visual and haptic rendering with a regular vertex spacing to the desired level of detail so that refinement techniques were not necessary.

2.3 Collision Detection

The role of collision detection is to find interferences between the objects in the virtual environment and the virtual tool as the tool is moved around by the haptic device. In a static configuration, with a sampled volume and point shell representation, this amounts to iterating over the points in the point shell and querying the volume at each location to determine if any of the points are within the isosurface. However, if we wish to prevent interpenetrations between tool and object, or otherwise constrain the tool to the object's surface, this simple technique is insufficient. Using only a static configuration, it is impossible to detect interference until tool-object interpenetration has already occurred. This problem manifests itself especially when the tool travels at a high velocity (translational or rotational) in a virtual environment containing very thin objects.

One method of addressing the interpenetration problem is to limit the velocity of the proxy so that the maximum distance it can travel during one time step is known, and the proxy can be stopped at the configuration before interference occurs. A more effective method is to consider both the initial and target configurations of the proxy for a given time step and to test the path swept by the proxy geometry between these configurations for interference. For a 3-DOF point proxy, this is akin to testing the line segment between the proxy position and the device position for interference [18].

The collision detection method employed here is similar in spirit to the continuous collision detection technique described by Redon et al. [9], but implemented for volume-embedded isosurfaces instead of polygonal geometry. The algorithm considers both the initial and target configurations of the proxy, or, equivalently, the current configuration of the proxy and a 6-dimensional velocity describing the rigid-body motion that would take the proxy to the target configuration in a unit time step. The goal is to find the earliest time at which interference occurs so that the proxy may be moved precisely to the instant in which it contacts the surface.

The motion of a point on the point shell sweeps out a curved path in space during each time step. In [9], the interference condition between geometric primitives is written as a time-parameterized equation, and interval arithmetic is used to isolate the earliest time interval in which a collision occurs. However, an isosurface cannot be exactly described by a set of polygonal primitives, and, hence, a time-parameterized interference equation cannot be formulated. Instead, we observe that an isosurface within a sampled volume has a smallest feature size, which we call f, that is a function of the sampling interval. In our case, we estimate f as half the smallest voxel side length in any cardinal direction. A series of static interference tests is used, with the motion path subdivided so that no segment exceeds f, to ensure that a collision is not missed.

Given an initial and target position of a point on the proxy, the constant velocity vector can be derived as $\mathbf{v} = (\vec{v}, \vec{\omega})$, where $\vec{v}$ is the linear component and $\vec{\omega}$ is the angular velocity about the proxy's center of rotation (typically the center of mass and virtual coupling point, as well). Note that the constant velocity assumption is made only to provide an even subdivision of the motion path and has no consequence on the actual motion of the proxy. The motion of the point through a unit time step is then

$$\vec{p}(t) = \vec{c_0} + \vec{v}t + \mathbf{R}(\vec{\omega}t)\vec{r_0}, \quad (2)$$

where $\vec{c_0}$ is the initial proxy's center of rotation, $\vec{r_0}$ is the point's radial vector from the center of rotation, and $\mathbf{R}$ is a rotation matrix about a given axis/angle. Although a screw motion can be used [6], we felt this formulation is more consistent with the motion that would be induced by the force of the virtual coupling spring.

The path length is also needed in order to subdivide this motion into feature-sized segments. Lacking a concise expression for the path length, s, a conservative estimate can be used. The distance travelled by the point in a unit time step is bounded by the sum of the linear and rotational arcs, or $s \leq |\vec{v}| + |\vec{r_0}||\vec{\omega}|$.

The motion for each point can then be divided into $\lceil s/f \rceil$ equal time intervals, and the configurations at the beginning and end of each time interval can be tested sequentially for interference. Once the first interval $[t_1, t_2]$ is found where $\vec{p}(t_1)$ is outside the surface and $\vec{p}(t_2)$ is inside, interval bisection is used to further refine the contact position to within a desired epsilon error, $\varepsilon < f$. Using the bisection method, the midpoint of the interval $[t_1, t_2]$ is tested, and the interval is halved accordingly until the travelled distance is less

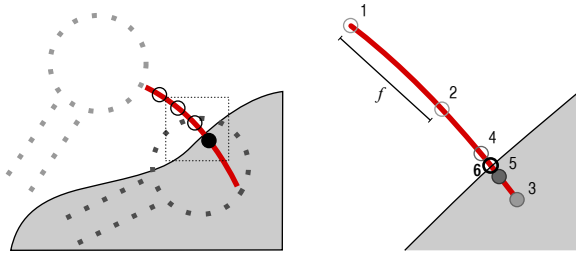


Figure 4: The motion arc is divided into feature-sized segments and each of the endpoints are tested for collision. The close-up on the right shows how interval bisection is used to refine the contact point.

than ε, thereby ensuring that the proxy will be moved to within a distance ε from the surface.

The full collision detection algorithm derives a rigid-body motion from initial and target configurations of the proxy, then performs an interference test for each point of the point shell using the subdivision and interval bisection methods described above. Rather than fully testing each point in turn, a time-prioritized queue of pending interference tests for all points is maintained. This type of priority-first search usually executes much faster than a depth-first search, especially when the objects are in contact, because the search can be terminated as soon as the first time of interference is found. This interference time, along with a contact set consisting of all points of the point shell within a distance ε of collision with the surface at that time, is reported as the result. A position within the volume, and the surface normal $\hat{n} = -\nabla F(\vec{p})/||\nabla F(\vec{p})||$ at that position, are also stored with each point in the contact set.

An additional task of the collision detector is to verify and prune the contact set in step 2 of the algorithm. This happens after the proxy and device have been moved during the previous time step and serves to ensure that every contact is still valid and should be considered for computing the constrained motion in the current time step. Given an initial proxy configuration and the current device configuration as the target, all points in the contact set are moved a distance ε from the initial configuration toward the target and then tested for interference. Points which lie outside the isosurface after applying the ε-motion are considered broken contacts and are discarded from the contact set. The remaining points are those that still, in a sense, lie between the proxy and the device, and are the ones that form active constraints on the proxy's motion.

2.4 Configuration Solver

The role of the configuration solver is to track the configuration of the proxy and to determine if, and how, it should move in response to motion of the haptic device. A constraint-based configuration solver much like that described by Ortega et al. [6] is employed for the rendering algorithm described in this paper. Each point of contact between the proxy and the environment establishes a constraint on the motion (translation and rotation) of the proxy. This differs from rendering algorithms that use a dynamic proxy, such as [2, 5, 7], where interpenetrations apply penalty forces to be integrated into the motion of the virtual tool.

The proxy is the representation of the haptic device within the virtual environment and will, therefore, attempt to move toward the device configuration whenever possible. Interferences with geometry in the virtual environment may prevent the proxy from moving to the target configuration, and the configuration solver's primary function is to compute the motion of the proxy given these constraints. With a 3-DOF point-based interaction, it is easy to picture the correct behavior of the proxy point. A single contact limit's the proxy's motion to a tangent plane perpendicular to the normal of the contact point (i.e. sliding on the surface). With the objective of moving the proxy toward the device, the configuration solver should move the proxy to the position on the constraint plane closest to the device position: its projection onto the plane. Simultaneous contacts imposing multiple constraints can likewise be solved using an iterative process or with Lagrange multipliers, as described in [18].

With the introduction of rotational motion for 6-DOF haptic interaction, the problem of solving for the optimal proxy configuration becomes significantly more complex. One elegant solution is provided by Gauss' principle of least constraint, which describes the motion of a body under external forces and constraints imposed on its motion. Of all allowable accelerations under the given constraints, Gauss' principle states that the true motion of the body will minimize the "acceleration energy" or "kinetic distance" [3, 10]:

$$\begin{aligned} E(\mathbf{a}) &= \tfrac{1}{2}(\mathbf{F}-\mathbf{Ma})^{\mathsf{T}}\mathbf{M}^{-1}(\mathbf{F}-\mathbf{Ma}) \\ &= \tfrac{1}{2}(\mathbf{Ma}_u-\mathbf{Ma})^{\mathsf{T}}\mathbf{M}^{-1}(\mathbf{Ma}_u-\mathbf{Ma}) \\ &= \tfrac{1}{2}(\mathbf{a}_u-\mathbf{a})^{\mathsf{T}}\mathbf{M}(\mathbf{a}_u-\mathbf{a}) \end{aligned} \tag{3}$$

The external force and torque applied to the body is denoted $\mathbf{F}$, resulting in an unconstrained accelerated $\mathbf{a}_u$ according to the body's mass matrix $\mathbf{M}$, and $\mathbf{a}$ is a generalized 6-dimensional acceleration vector compatible with the constraints.

In the configuration solver, the device exerts a force and torque on the proxy in the direction of the device configuration, and thus determines the direction of the proxy's unconstrained acceleration $\mathbf{a}_u$. The magnitude of $\mathbf{a}_u$ is chosen so that the motion would take the proxy from its initial configuration to the device configuration in a unit time step (i.e. $\Delta\mathbf{x} = \frac{1}{2}\mathbf{a}_u t^2 \Rightarrow \mathbf{a}_u = 2\Delta\mathbf{x}$). Each point of contact between the proxy and the environment reduces the proxy's motion by one degree of freedom, imposing a constraint on its acceleration $\mathbf{a} = (\vec{a}, \vec{\alpha})$ of the form

$$\vec{a}\cdot\hat{n} + \vec{\alpha}\cdot(\vec{r}\times\hat{n}) \geq 0, \tag{4}$$

where $\hat{n}$ is the surface normal at the point of contact and $\vec{r}$ is the radial vector from the proxy's center of rotation to the contact point.

The minimization of equation (3) subject to multiple constraints of the form of (4) is a quadratic optimization problem which, in the general case, can be solved using quadratic programming. However, careful examination of the problem yields a more efficient approach. Each constraint is a hyperplane passing through the origin in six-dimensional space. Together, the constraints form a convex polyhedral cone that bounds the allowable accelerations, and the constrained acceleration $\mathbf{a}_c$ that minimizes (3) is the (non-Euclidean) projection of the unconstrained acceleration onto this convex cone. Wilhelmsen's projection algorithm [16] is used to find the projected point, as was done in [6].

Once the constrained acceleration is computed, it can be applied directly to the initial proxy configuration to obtain a target proxy configuration (i.e. $\mathbf{x}_t = \mathbf{x}_i + \frac{1}{2}\mathbf{a}_c$). There is no numerical integration of the acceleration to find velocities or positions, as the solver treats the motion as a quasi-static system. The collision detector is then queried to find the first point of interference, if any, between the initial and target proxy configurations. Finally, the proxy is moved either along the constrained path of motion to the configuration of first contact or directly to the target configuration if no interference with the environment geometry occurred.

2.5 Virtual Coupling

The function of the virtual coupling component is to determine the physical force to be applied to the haptic device operator as a result of the tool's interaction with the virtual environment. A virtual spring between the proxy and device determines the force transmitted to the operator. The reflected force is opposite to that applied on the virtual tool and can be described as

$$\mathbf{F}_r = -k\mathbf{Ma}'_u, \tag{5}$$

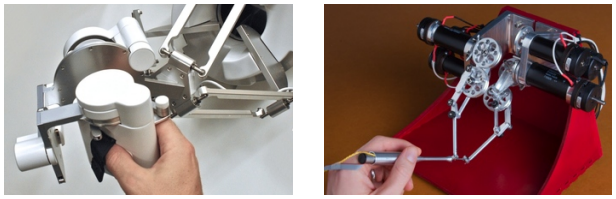
where k is a coupling constant (essentially the virtual spring constant), and $\mathbf{a}'_u$ is the acceleration vector that would take the proxy to the device configuration *after* it has moved. Different coupling constants for the linear and torsional components of the reflected force may be used if, for example, they would better match the output capabilities of the device, as long as the mismatch between the applied and reflected force directions is tolerable. In this case, the scalar k is replaced by a diagonal matrix $\mathbf{K}$ containing the constants.

3 Results

3.1 Experimental Setup

The haptic rendering algorithm was implemented on top of the open-source CHAI 3D framework [4] in order to maximize compatibility with a variety of haptic devices. Two haptic devices with different characteristics and at least six actuated degrees of freedom (shown in Figure 5) were used to evaluate the algorithm: the sigma.7 (Force Dimension, Nyon, Switzerland) and the μHaptic Device [12], a custom interface designed in our laboratory for microsurgical applications. The latter device has the workspace, force output, and inertia properties similar to a powered microsurgical instrument, such as a surgical drill.

The rendering algorithm is also compatible with asymmetric haptic devices that have six sensed, but fewer (typically three) actuated degrees of freedom. With such an interface, the 6-DOF method still provides an advantage in that it allows interaction using an arbitrarily shaped virtual tool, although a passive degree of freedom may contribute to overly active feedback on an actuated one if the tool has a large moment arm. Two asymmetric devices, an omega.6 (Force Dimension) and a PHANTOM Omni (SensAble Technologies, Wilmington, MA), were also used to test the algorithm.



(a) (b)

Figure 5: Haptic devices used to evaluate the rendering algorithm: (a) Force Dimension sigma.7 and (b) μHaptic Device.

Four sampled volume data sets were collected and prepared for evaluating the rendering algorithm: a CT scan of the original Stanford Bunny, a preoperative image of the temporal bone, a synthesized block with a hole through it, and the highly popular engine block volume. Additionally, polygonal models of four virtual tools were created to interact with the volume data: a cane, a surgical drill, a peg that fits the hole in the block, and a wrench.

3.2 Performance and Observations

The performance characteristics of the algorithm on the various data sets were measured and summarized in Table 1. The mean execution time required for the algorithm fits comfortably within the 1000 μs window available for a haptic servo rate of 1000 Hz, with the larger point shells approaching the limits of our computational resources. Although the algorithm is fast, it cannot be guaranteed to terminate within the allotted time for all possible configurations of the tool, as shown by the performance timings. If a hard 1000 Hz servo rate is required, the virtual coupling component can be run on a separate real-time thread that synchronizes with the other components as the proxy configuration is updated. This may also allow larger point shells to be used for describing the virtual tool.

Volume	Resolution	Tool	Points	Exec. Time (μs)
Bunny	$512 \times 512 \times 361$	Cane	403	209 (900)
Engine	$256 \times 256 \times 110$	Wrench	2126	776 (4380)
Hole	$256 \times 256 \times 256$	Peg	1307	414 (2135)
T. Bone	$768 \times 768 \times 70$	Drill	518	255 (1181)

Table 1: Performance characteristics of the rendering algorithm on a 2.67 GHz Intel Core i7 CPU. Mean execution time per haptic frame is reported, with the highest recorded single time in parentheses.

Our implementation of the configuration solver has a time complexity of $O(n^2)$ in the number of contacts, but the linear complexity of collision detection in the size of the point shell is the primary time bottleneck. Because each contact adds a 1-DOF constraint to the proxy's motion, 6 contacts would in theory fully constrain the motion of the proxy. Even allowing for redundant contacts due to approximations in collision detection, we observed that typical tool-object interactions do not exceed about 10 contacts. Contact between flat surfaces with densely-sampled coplanar points represents the pathological case, as demonstrated in our peg-in-hole benchmark, where up to 40 simultaneous contacts were observed.

Figure 6 shows selected frames during the haptic interaction on each of the four test volumes. The volumetric isosurfaces were visually rendered in real-time using a GPU-accelerated ray casting approach. The video supplement accompanying this paper shows the entire sequence for each of these interactions.

The virtual tool was wedged into crevices, pried while in holes, and hooked around protrusions on the isosurface geometry as shown in Figure 6 and in the video. The virtual coupling stiffness was set to the highest permissible virtual wall stiffness as rated by the device manufacturer. The simulation was stable throughout the entire duration of the haptic interaction, even with a coupling stiffness as high as 5000 N/m on the Force Dimension devices. The tool could be moved quickly in free space or in contact, and absolutely no effects of virtual mass, inertia, or viscosity could be felt.

4 Conclusion

A six degree-of-freedom haptic rendering algorithm for isosurface geometry embedded within sampled volume data was presented in this paper. It uses a quasi-static simulation of a geometrically complex proxy represented as a point-sampled surface. A constraint-based formulation of the configuration solver allows the proxy to be simulated without virtual mass or inertia and ensures that it never penetrates into other objects. Its ability to render geometry within unmodified volume data, such as computed tomography or magnetic resonance images, makes it particularly suitable to applications in medical or surgical simulation.

The algorithm was tested on a collection of four volume data sets with different tools controlled by a variety of haptic devices. In all cases, the simulation allowed quick movements of the tool without conveying artificial mass or inertia and was responsive to stiff contacts. The simulation remained stable while exercising interactions that included wedging, prying, and hooking, even with the coupling stiffness set as high as the devices' physical limits allowed.

As future work, we plan to explore optimizations that would allow the algorithm to use significantly more complex virtual tools without compromising its ability to render arbitrary volume data. Two possibilities are described in [2]: the use of a signed distance field for representating the environment, and a hierarchical organization of the points in the proxy. Although these techniques can accelerate collision detection by an order of magnitude or more, the primary disadvantage is that the signed distance field required can only embed a single isosurface. It is our ultimate goal to extend the ideas discussed in this paper to address haptic rendering of deformable volumetric geometry.

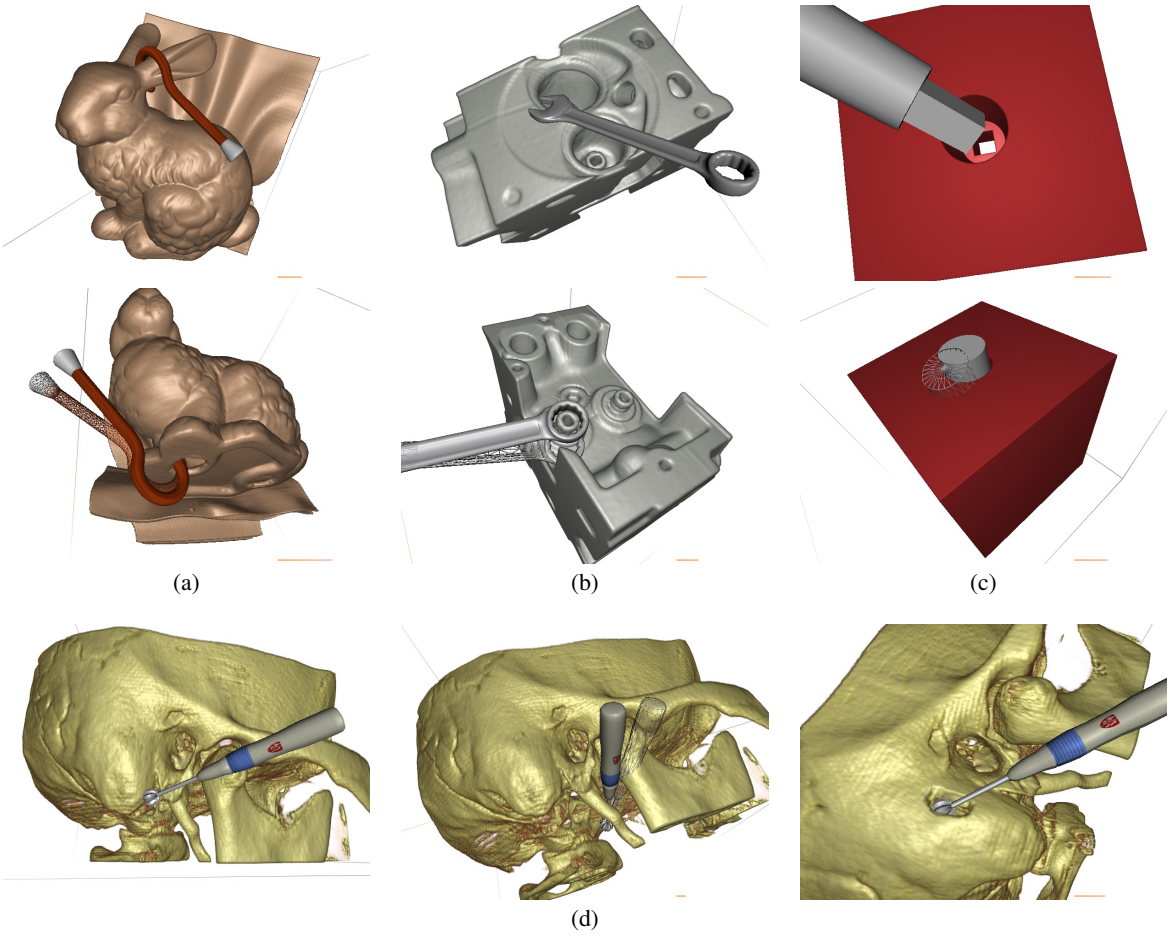


Figure 6: Results of haptic interaction between a complex virtual tool and isosurfaces within sampled volume data: (a) Stanford bunny with a cane, (b) engine block with a wrench, (c) synthetic hole with a snugly fitting peg, and (d) CT scan of the temporal bone with a surgical drill. The proxy (shown by the solid model) never penetrates the isosurface geometry, even when the device (shown by the wireframe model) does.

REFERENCES

[1] R. Avila and L. Sobierajski. A haptic interaction method for volume visualization. In *Proc. IEEE Visualization*, pages 197–204, 1996.

[2] J. Barbič and D. L. James. Six-DoF haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, 1(1):39–52, 2008.

[3] H. Bruyninckx and O. Khatib. Gauss' principle and the dynamics of redundant and constrained manipulators. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2563–2568, 2000.

[4] F. Conti, F. Barbagli, D. Morris, and C. Sewell. CHAI 3D: An open-source library for the rapid development of haptic scenes. In *Proc. IEEE World Haptics*, 2005.

[5] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. ACM SIGGRAPH*, pages 401–408, 1999.

[6] 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 Transactions on Visualization and Computer Graphics*, 13(3):458–69, 2007.

[7] M. A. Otaduy and M. C. Lin. Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. In *Proc. IEEE World Haptics*, pages 247–256, 2005.

[8] M. A. Otaduy and M. C. Lin. Introduction to haptic rendering algorithms. In M. C. Lin and M. A. Otaduy, editors, *Haptic Rendering*, chapter 8, pages 159–180. A K Peters, Wellesley, MA, 2008.

[9] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–287, Sept. 2002.

[10] S. Redon, A. Kheddar, and S. Coquillart. Gauss' least constraints principle and rigid body simulations. In *Proc. IEEE International Conference on Robotics and Automation*, pages 517–522, 2002.

[11] D. C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. In *Proc. ACM SIGGRAPH*, pages 345–352, 1997.

[12] C. M. Salisbury, J. K. Salisbury, N. H. Blevins, and R. B. Gillespie. A microsurgery-specific haptic device for telerobotic medical treatment. In *Proc. ANS International Joint Topical Meeting on Emergency Preparedness and Response and Robotic and Remote Systems*, 2008.

[13] K. Salisbury and C. Tarr. Haptic rendering of surfaces defined by implicit functions. In *Proc. ASME Dynamic Systems and Control Division*, volume 61, pages 61–67, 1997.

[14] G. Turk. Re-tiling polygonal surfaces. In *Proc. ACM SIGGRAPH*, pages 55–64, 1992.

[15] M. Wan and W. A. McNeely. Quasi-static approximation for 6 degrees-of-freedom haptic rendering. In *Proc. IEEE Visualization*, pages 257–262, 2003.

[16] D. R. Wilhelmsen. A nearest point algorithm for convex polyhedral cones and applications to positive linear approximation. *Mathematics of Computation*, 30(133):48–57, Jan. 1976.

[17] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proc. ACM SIGGRAPH*, pages 269–277, 1994.

[18] C. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 146–151, 1995.