Voxel-Based Haptic Rendering Using Implicit Sphere Trees

Emanuele Ruffaldi[1] Dan Morris[2] Federico Barbagli[3] Ken Salisbury[3] Massimo Bergamasco[1]

[1]Scuola Superiore S. Anna [2]Microsoft Research [3]Stanford University

Abstract

Haptic interaction in six degrees of freedom is critical to numerous applications, but is still prohibitively complex for realistic environments. This paper presents an approach to rendering six-degree-of-freedom contact among virtual objects using a novel data structure referred to as an *implicit sphere tree*. This data structure allows an extremely compact representation of volumetric objects and extremely rapid intersection testing among objects, which broadens the scope of virtual environments that can be rendered in six degrees of freedom at interactive rates. We introduce this data structure, along with appropriate techniques for collision detection and haptic rendering, and demonstrate its efficiency in representing and manipulating complex models.

Keywords: haptics, collision detection, volumetric models

Index Terms: H.5.2 [Information Interfaces and Presentation]: User Interfaces — Haptic I/O; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — Geometric algorithms, languages, and systems

1 Introduction

As haptic applications in virtual prototyping, medical simulation, and entertainment demand increasing immersiveness and realism, the perceptual limitations of three-degree-of-freedom (3-DOF) haptic rendering become increasingly problematic. Six-degree-of-freedom (6-DOF) haptic rendering intrinsically provides increased realism for such applications (since real-world interactions typically use at least six degrees of freedom) and leverages the full capacity of haptic devices that can render force and torque simultaneously (e.g. [18],[5]). However, this increased realism comes at a cost: 6-DOF rendering requires significantly more complex collision detection among virtual objects and fundamentally prevents the frequently-used approach of representing a haptic tool as a single point or as a small cluster of points [42].

To cope with this increased computational cost, many approaches to 6-DOF haptic rendering leverage voxel-based models as a fundamental representation, in contrast to the surface-based (and generally polygon-based) models used to represent most objects in computer graphics and 3-DOF haptic rendering. Voxel-based rendering offers inherently more rapid collision tests among primitives. But again, this comes at a cost: in this case, voxel-based models are less accurate in representing object boundaries than surface models. However, this loss in accuracy can be mitigated to a nearly-arbitrary degree by maximizing the resolution of voxel-based models. Therefore, the goal of the present work, and previous work in 6-DOF haptic rendering, is to maximize the resolution of voxel-based objects that can be rendered interactively.

Address correspondence to emanuele.ruffaldi@sssup.it .

This goal – processing increasingly complex volumetric geometry – also supports the growth in complexity and resolution of application-specific data sources that are supplying increasingly complex models to virtual environments. For example, the resolution of medical imaging devices continues to improve, so voxel-based anatomical models used for haptic surgical simulation continue to increase in complexity. Similarly, as haptic feedback becomes increasingly relevant for virtual prototyping and CAD applications [16], designers will need to represent increasingly complex parts – made up of numerous sub-parts – if they are to make use of haptic simulation tools.

Thus the goal of maximizing the complexity of voxel-based models that can be manipulated interactively improves both the general realism of haptic environments and the suitability of haptic simulation for specific applications.

This paper therefore addresses the problem of 6-DOF manipulation of unconstrained rigid bodies represented as volumetric models. Our primary contributions are threefold:

1) We introduce the *implicit sphere tree*, a novel data structure for representing volumetric models, and describe an optimized approach for building this data structure.

2) We introduce collision-detection and force-rendering schemes that are suitable for interactive use of the implicit sphere tree.

3) We present benchmarking results that demonstrate the computational efficiency of this data structure.

In Section 2, we survey previous work related to 6-DOF haptic rendering, volumetric representations, and dynamic simulation. In Section 3, we describe the implicit sphere tree and our approach to interactive rendering. In Section 4, we present results demonstrating the computational efficiency of our approach. In Section 5, we discuss the relevance of this work and discuss future extensions.

2 Related Work

In this section, we survey previous work on volumetric representation of objects (Section 2.1) and six-degree-of-freedom haptic simulation (Section 2.2), and place our own work within the context of the existing literature.

2.1 Volumetric Models

As we discuss above, volumetric models present an advantage over surface models for rapid intersection testing in complex virtual environments. In addition, volumetric models present an inherent advantage for representing objects when the interior of an object has information associated directly with it, e.g. vector fields or medical image data.

2.1.1 Basic voxel representations

The most basic representation for a volume is the classic voxel array, in which each discrete spatial location has a one-bit label indicating the presence or absence of material. A slightly more detailed representation used in [25] includes a "Surface Descriptor" at each voxel, labeling each voxel as "empty", "full", "surface", or "proximity". A "surface voxel" is a full voxel that is near one or more empty voxels, and a "proximity voxel" is an empty voxel that is near a surface voxel. This additional representation, which can be encoded using only two bits per voxel, allows more sophisticated handling of the volume's surface for collision-detection.

Another frequently-used adaptation of the classic voxel array is the distance field, in which each voxel is labeled with the distance between that voxel and the nearest surface point. This allows rapid computation of local gradients, which can be used for optimizing collision detection and computing collision response forces [13].

2.1.2 Voxel storage mechanisms

Independent of the data stored with each location in a voxel representation, the *storage* of voxel data can be classified roughly into approaches based on a *uniform grid* and approaches based on *volume hierarchies*.

The *uniform grid* stores every voxel in the volume of an object using a tri-dimensional matrix or a hash table. Matrices are suitable for small objects and allow for extremely rapid data access, good spatial locality, and extremely rapid point-volume intersection tests. Hash tables or related indirect-access structures allow for a much more compact representation of sparse voxel arrays, but are more complex to address and manipulate, and can result in decreased cache performance relative to dense arrays when nearby voxels are accessed sequentially.

Approaches based on *volume hierarchies* store volume information at multiple levels of detail for compactness and rapid intersection-testing. These approaches are generally adaptations of the classic octree, itself an adaptation of the classic n-ary tree, in which each node represents a cube in space and each child of a node, if present, represents additional detail within a subspace of that cube [41]. Although voxel-based representations typically suffer from inadequate detail around high-frequency surface features, octrees can be optimized with adaptive sampling [12] to provide additional information in such regions. Another variant of the octree is the generalized spatial tree, in which each cube is partitioned into a larger number of subspaces (typically 64, 256, etc.), which in some applications reduces access time by flattening the hierarchy [25].

The implicit sphere tree presented in this paper builds upon the Surface Descriptor and generalized octree presented in [25] and the octree-based collision detection tree presented in [39].

2.2 6-DOF Haptic Rendering

Previous approaches to six-degree-of-freedom (6-DOF) haptic rendering can be classified into two main categories – *direct rendering* and *virtual coupling* – depending on the way they relate the physical position of the haptic device with the virtual position of the haptic interface point.

2.2.1 Direct Rendering

Direct rendering approaches (e.g. [14],[22],[20],[30]) to 6-DOF haptic rendering do not de-couple the physical device and virtual probe positions: the virtual haptic interface point is a simple linear transformation of the physical haptic device position. This guarantees that a user's control of a virtual haptic probe is direct, intuitive, and without latency, but allows for a large penetration depth between the probe and objects in the virtual environment, potentially resulting in perceptual inaccuracy, reduced frame rate, and instability.

2.2.2 Virtual Coupling

In contrast, approaches to 6-DOF haptic rendering based on *virtual coupling* [6] use a dynamic simulation to compute the position of the virtual haptic probe based on the position of the physical haptic device. For example, a bi-directional spring is frequently used to simulate this coupling. This solution provides a much more stable response and maintains perceptual accuracy for graphic rendering, but it has the often-undesirable effect of smoothing the haptic feedback forces provided to the user.

There are several more sophisticated dynamic simulation methods used for this coupling; we can classify them loosely as *penalty-based*, *constraint-based*, and *impulse-based* methods.

Penalty-based methods (e.g. [7],[14],[24],[25],[29],[32],[39]) identify two discrete simulation states: *contact* and *non-contact*. Such approaches respond to the *contact* state – in which the virtual probe is immersed in another object in the virtual environment – with a force that is proportional to the penetration depth between the object and the stiffness of the materials. These approaches are suitable for haptics because they are computationally efficient, but limit the perceived stiffness of haptic interactions. The approach described in [17] uses the *volume* of intersection – instead of the penetration depth – for computing penalty forces. In [32], penalty-based approaches were extended to a multi-rate computation scheme to maintain haptic fidelity even during variable-rate collision detection. Penalty-based approaches are also used for non-haptic dynamic simulation (e.g. [21],[24],[37]).

Constraint-based methods (e.g. [3],[35]) represent objects or other environmental phenomena as analytic constraints, and typically integrate forces as necessary to ensure that those constraints are not violated. Work in this area has focused on schemes for smooth and variable-time integration and on real-time translation of analytic constraints into computationally-efficient penalty-based rendering schemes at the level of the haptic controller (e.g. [3],[33],[34],[35],[42]). Again, constraint-based methods for dynamic simulation have been used extensively outside of haptics, particularly in computer graphics (e.g. [1],[2]).

Finally, *impulse-based* methods (e.g. [4],[8]) respond to collisions between a haptic probe and other objects in the virtual environment with an impulsive force intended to both simulate the interaction between rigid objects and eliminate penetration among objects. This problem has been explicitly addressed by the use of braking forces ([36],[25]), by an open-loop or event-based response ([11],[23]), or by a hybrid approach [9] that generates force pulses at the initial contact but uses a penalty-based response for the resting contact.

In this work, an impulse-based method has been used for the resolution of contacts, as presented in Section 3.3.

2.3 Summary of Related Work

Figure 1 presents a subset of the broad hierarchy of techniques used for 6-DOF haptic rendering and volume modeling, particularly focusing on those approaches discussed in this section. This figure is intended to situate our work within this increasingly-complex research space and highlight the highest-level design choices that guide our methods.

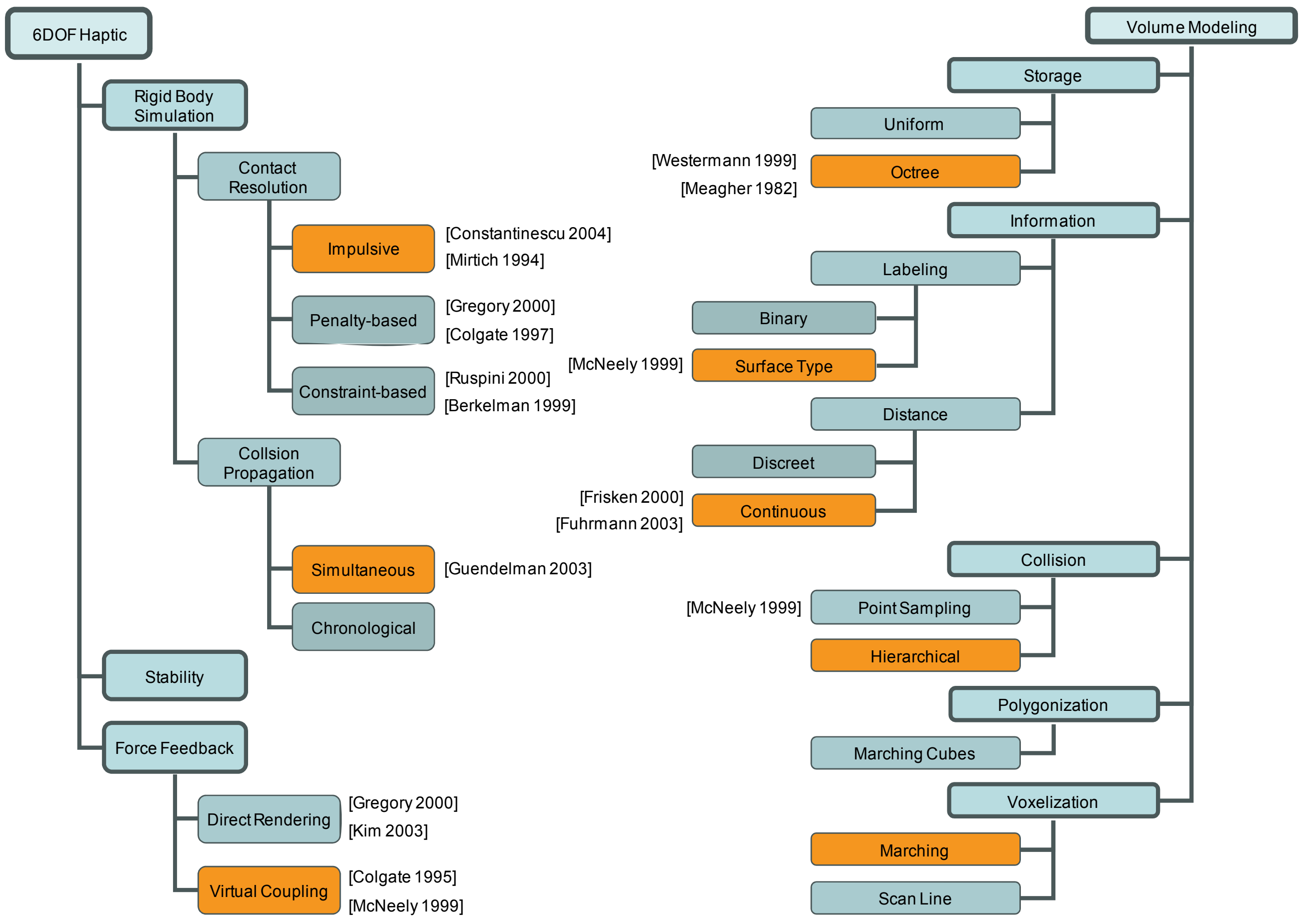

Figure 1. Situating our work (orange) among the methods and representations used in the literature for 6DOF haptic rendering (left) and volumetric modeling (right). Work referenced elsewhere in this paper is cited here for context.

3 Methods

This section introduces the *implicit sphere tree* (Section 3.1), the central data structure in the present work, then describes appropriate collision-detection (Section 3.2) and force computation (Section 3.3) approaches for working with implicit sphere trees.

3.1 The Implicit Sphere Tree

In section 2.1.2, we discuss the octree, a hierarchical series of cubes traditionally used for compactly representing complex objects. While this data structure is efficient, intersection testing among cubes is computationally expensive, relative to spheres, when working with objects at arbitrary rotations. The rotational invariance of the sphere makes it a particularly desirable geometry for bounding-volume hierarchies. Consequently, hierarchical *sphere trees* [19] have been explored in previous work as an alternative to octrees. However, sphere trees are much more complex to construct and manipulate than octrees, and suffer from much less efficient bounding of voxel arrays than octrees.

The data structure presented here – the *implicit sphere tree* – combines the intersection-testing advantages of the sphere tree with the spatial efficiency of octrees by building a hierarchy of spheres directly from the nodes of an octree while traversing the tree for collision-detection, with minimal additional storage.

The use of the implicit sphere tree begins with the construction of a traditional octree representation for an object of interest; each octree node stores – in addition to the traditional list of child-node pointers – the level L of the tree at which this node sits. A leaf node (typically a single voxel) is assigned a level L=0. Each node is assigned a level one greater than the level of its children, so – for example – the root of an octree containing 256 voxels per side has a level L=8. In practice, we need only store the level L of the *root* node of the octree.

We build a standard octree [27] (enhanced by these node-level labels) and compute – from the known dimensions of the octree – the radius of the bounding sphere around the *root* of the tree (bounding the entire object). We note that for a cube of side x, this bounding sphere has a radius of $x\sqrt{3}/2$, and we can thus compute the radius of the sphere that bounds the root of the octree as:

$$r_0 = s2^{L-1}/\sqrt{3}$$

…where r_0 is the radius of the bounding sphere at the root of the tree, s is the edge length of an individual voxel, and L is the node level of the root octree node. In the case of the generalized N-tree, this radius is $s2^{N\,(L-1)}/\sqrt{3}$, where N is 1 for the octree, 2 for the 64-tree and 3 for the 512-tree.

During the collision-detection process, as with traditional octree-based collision-detection, we will be descending the tree from the root to determine regions that merit further intersection testing (Section 3.2 will provide more detail on collision detection). As we descend the tree, choosing to descend to certain nodes in the octree, we compute the bounding spheres of each

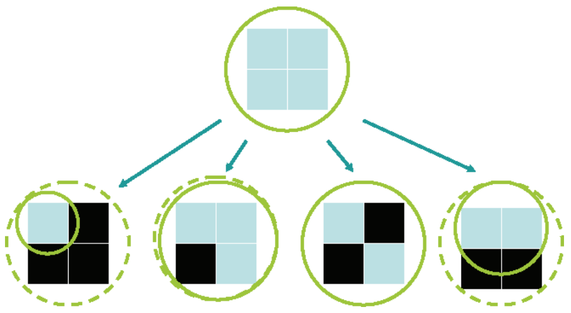

Figure 2. A two-dimensional quadtree is used to illustrate the finite set of bounding circles that exist for an octree node, and the dependence of those bounding circles on the empty/full state of child nodes. Dotted lines indicate the largest possible bounding circle; solid lines indicate optimized bounding circles. Dark child nodes are empty, light (blue) nodes are full.

child of a given octree node by simply scaling the radius of the parent node's bounding sphere by ½ and offsetting the current node's bounding sphere center by $s2^{L-2}$ along each axis, where s is the size of a voxel and L is the current octree level (which we know based on how many levels we've descended so far). We thus implicitly compute the bounding sphere for each child node based only on very limited global storage (the number of levels in the octree) and on our real-time information about this node's parent node.

We can further optimize the computation of our implicit sphere tree when the octree (or the generalized n-tree) is not full, by computing a bounding sphere at each node that takes into account the *real* distribution of the children of the octree node. This optimization has the objective of reducing the volume of each sphere and thus reducing the number of collision tests necessary to eliminate intersection candidates. We can do this without complex computation at each node, leveraging the small set of possible bounding configurations that can exist at each level of an octree. In other words, there are only so many possible configurations of *non-empty* child nodes within an octree node, and therefore there are only so many possible bounding spheres needed to represent all possible child-node configurations. By pre-computing this very limited set, and using a simple integer representation for each node in our octree, where each bit represents the presence or absence of a child node, we can compute a *tight* bounding sphere for an octree node by taking this integer representation and directly indexing into a table containing scale/offset information for all possible bounding spheres.

Figure 2 demonstrates this principle in two dimensions for the quadtree (the two-dimensional equivalent of the octree). In the quadtree case, we would use four bits to represent the set of child nodes within each quadtree node. The case where no child nodes are present does not require further computation, since this node will never be a candidate for object intersections, so we have only fifteen possible arrangements of present/absent child nodes. As illustrated in Figure 2, three of these cases yield a bounding circle that is equivalent to the circle bounding the complete square, and each of the other twelve cases yields one of three other possible (smaller) bounding circles. For these twelve reduced-bounding-circle cases, we can store a single, global table that contains the relative offset and scale of these bounding circles, which can be quickly looked up with a single integer-indexing operation.

When we move to the octree we have 255 combinations, among which 85 yield bounding spheres that are smaller than the largest possible bounding sphere. Looking up the appropriate, optimized bounding sphere for an octree node proceeds exactly as in the quadtree case, making use of a single, global lookup table containing a scale and three-dimensional offset (relative to node centers) for each possible bounding sphere.

3.2 Collision Detection for Implicit Sphere Trees

Collision detection using the implicit sphere tree is similar to traditional collision detection using octrees; the sphere tree is used to greatly accelerate intersection testing by making all intersection tests rotationally-invariant.

When we wish to determine whether two objects are intersecting, we begin with the root node of each object's octree and, as described in Section 3.1, compute the two global bounding spheres in global coordinates, testing for intersection between those spheres using a simple distance/radius comparison. If the two root spheres intersect, we compute the bounding spheres at the next level of each octree, as described in Section 3.1. We note that computing the next level of bounding spheres *is* dependent on rotational state, since it involves offsetting each parent node's center along each axis; however, we can take advantage of the fact that within an object, these axes do not change from level to level within the tree, and we can therefore simply compute these primary axes *once* per iteration of our program's main simulation loop and scale them appropriately each time we descend the tree. Once we have computed these second-level bounding spheres for each root octree node, we test for intersections between the child-node spheres of one octree with the root-node sphere of the other, and vice versa. Again, all computations can be performed without respect to the rotational state of each object using the rotationally-invariant implicit sphere tree.

If intersections are detected, we continue this process, descending the tree as needed until either all possible intersections are eliminated or leaf nodes (voxels) from each tree are found to intersect. As with any bounding volume hierarchy, the descent of the collision tree can be interrupted at a certain level when the number of generated collisions is too large or an application-specific resolution limit is reached. Also, for applications that are interested specifically in intersections between surfaces and are able to assume a limited penetration depth among objects, it is possible to simplify collision detection by using only voxels labelled as "surface" voxels (using the Surface Descriptor approach presented in [25]), and building the octree (and thus the implicit sphere tree) such that it only contains nodes whose children ultimately contain surface voxels.

In order to maximize the parallelism of our approach in environments with multiple processors, a breadth-first recursion is used. This allows each level to be assigned to an independent processor. This approach also allows the descent of the collision tree to be interrupted at a certain level when the number of generated collisions is too large. Otaduy and Lin [31] present a perceptually-derived metric for early termination of a collision search that maximally preserves haptic fidelity.

3.3 Collision Response: Haptic Rendering with Implicit Sphere Trees

This section describes our collision response algorithm, suitable for use with the implicit sphere tree presented above, and the overall structure of our 6-DOF haptic rendering algorithm. We follow the approach of [25] in assuming that the virtual environment is represented as a voxel model (the "world voxel model"). A finite set of samples points (the "point shell") is used to represent the surface of a probe object (being controlled by a haptic device); this point shell is treated as a voxel array for purposes of collision detection. As in [25], we assume that the

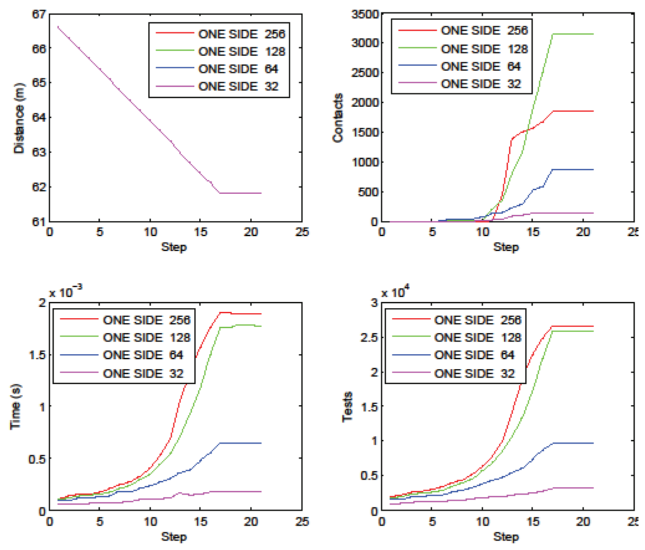

Figure 3. The relationship between collision detection performance and the resolution of a volumetric model (collision response disabled). Graphs show the distance between the centers of the two voxel models being tested (upper-left), the number of contacts detected between these objects (upper-right), the computation time required for each simulation time step (lower-left), and the number of intersection tests required at each simulation time step (lower-right).

world voxel model is static and the probe object is dynamic.

Although the present work uses the voxel representation presented by McNeely et al., [25], our work differs from [25] both in terms of the collision detection scheme and the force computation scheme. In particular, our work has been designed to support multibody dynamics.

3.3.1 Contact Resolution

This work uses the implicit sphere tree described above to detect collisions between points in the point shell (the dynamic object) and voxels in the (static) world voxel model. Collisions detected in a single simulation time step are treated as having occurred simultaneously, as detecting the precise sequence of contact events would require a prohibitively-complex rewinding of the simulation whenever collisions occur.

When collisions are detected, an impulse is applied to the probe object to eliminate penetration between the probe object and the world voxel model. We adapt the methods of [28] and [15] to compute this impulse, and we describe this adaptation here.

The contact response system receives a list of intersecting voxel pairs; each pair is described by the two voxel centers, the normal at each voxel, the relative velocity of the intersecting voxels, and the penetration depth between the two voxels. The contact response system selects the colliding voxel pair that has the largest penetration depth, ignoring any contact pairs whose velocities would result in a resolution of the intersection in the next integration step. In other words, we do not apply impulses to resolve contacts that would be resolved in the next time step by inertia alone. When an intersecting voxel pair is selected, the contact response system resolves the contact using an impulse that imposes a separating velocity condition in the next integration step [2].

The impulse computed above will be applied to the object in a subsequent integration step, but other intersecting voxel pairs may still require resolution. Instead of immediately applying the computed impulse to the body and re-computing the set of intersecting voxel pairs, we continue to use the same set of intersecting voxel pairs but use the computed impulse to update the velocity of the body. This update allows us to discard most of the intersecting voxel pairs that are geometrically close to the one computed in the previous step, since those will now be resolved in the next integration step without an additional impulse, as described above. The system thus proceeds to the next intersecting voxel pair that would *not* now be separated in the subsequent integration step. This operation is repeated until there are no eligible intersecting voxel pairs, or until a maximum number of iterations is reached. The result of the collision response is the cumulative impulse, which is then applied to the probe object.

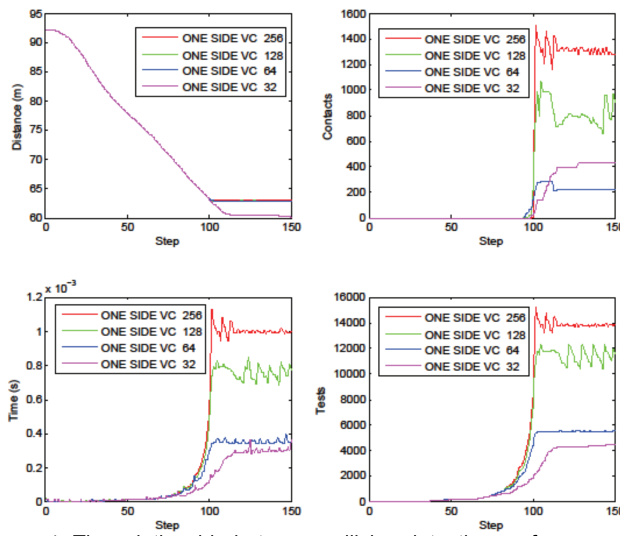

Figure 4. The relationship between collision detection performance and the resolution of a volumetric model (collision response enabled). Graphs show the distance between the centers of the two voxel models being tested (upper-left), the number of contacts detected between these objects (upper-right), the computation time required for each simulation time step (lower-left), and the number of intersection tests required at each simulation time step (lower-right).

The resulting impulsive forces are applied to the haptic device through virtual coupling [6]. At the cost of some damping, virtual coupling provides smoother, more stable haptic feedback than direct coupling, and allows force feedback computation to proceed at haptic rates even when collision detection and dynamic simulation are slowed by scene complexity.

4 Benchmarking

The algorithms described above have been implemented in C++ using the CHAI3D [10] open source haptics library. In this section, we present the results a series of benchmark tests applied to this implementation. Tests have been conducted with a Phantom Desktop haptic device on a Intel Core2 Quad running at 2.4GHz with 4GB of memory under Windows XP.

Our first benchmark assesses the impact of voxelization resolution on collision detection performance. For this evaluation, a simulation was repeated several times, identical in each case other than the resolution of the world voxel model. In this simulation, a model is moved along a trajectory toward another object; collision response forces are disabled. Figure 3 presents an analysis of this simulation. We highlight in particular that for a relatively high number of collisions (around 1000) the algorithm requires less than 1ms of computation time, allowing it to run within a typical haptic simulation timestep. We also note that the more complex simulation timesteps occurring toward the end of

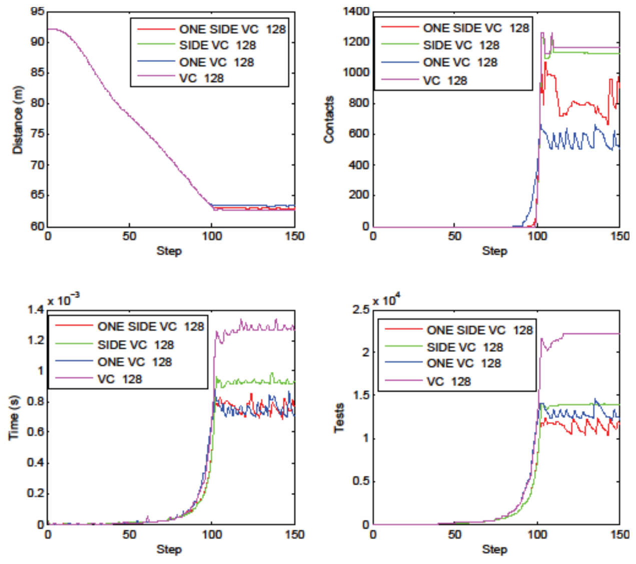

Figure 5. The impact of the "one-child" and "side" optimizations (discussed in Section 3) on collision detection performance. Graphs show the distance between the centers of the two voxel models being tested (upper-left), the number of contacts detected between these objects (upper-right), the computation time required for each simulation time step (lower-left), and the number of intersection tests required at each simulation time step (lower-right).

this simulation, which require more computation time, are not representative of typical interactive simulation timesteps, as collision response was disabled for this evaluation. Figure 4 summarizes the results of the same analysis performed with collision response enabled.

It is interesting to understand the effect of the optimizations discussed above, in particular the use of the one-child optimization for skipping intersection tests and the side optimization for adjusting the position. To assess the utility of these optimizations, we again simulate an approach between two voxel objects, in this case with a fixed size. The results of this analysis are illustrated in Figure 5. We highlight that the one-child optimization provides a significant performance benefit.

The overall performance of the algorithms presented in this paper has been tested by generating random trajectories of collision between two objects. The large 6DOF search space has been randomly sampled using a sphere method [38]. One of the two models is fixed, and in each simulation the other object is moved along the randomly-generated trajectory. Each trajectory is a translation from an initial position randomly generated in spherical coordinates to a final position that has the same angular coordinates of the starting point but a radius that is known to result in contact between the simulated objects. The translating object is moved with constant velocity according to the method of [38]. The result of this evaluation is summarized in Figure 6.

5 Conclusions

The use of volumetric models for collision detection allows more complex objects to be simulated interactively, but requires specific collision detection schemes. In this paper, we have introduced the *implicit sphere tree*, a data structure that allows rapid intersection testing among complex voxel-based objects.

Future work will focus on more sophisticated parallelization of the proposed approaches, particularly on implementing the proposed approach in massively-parallel computing environments

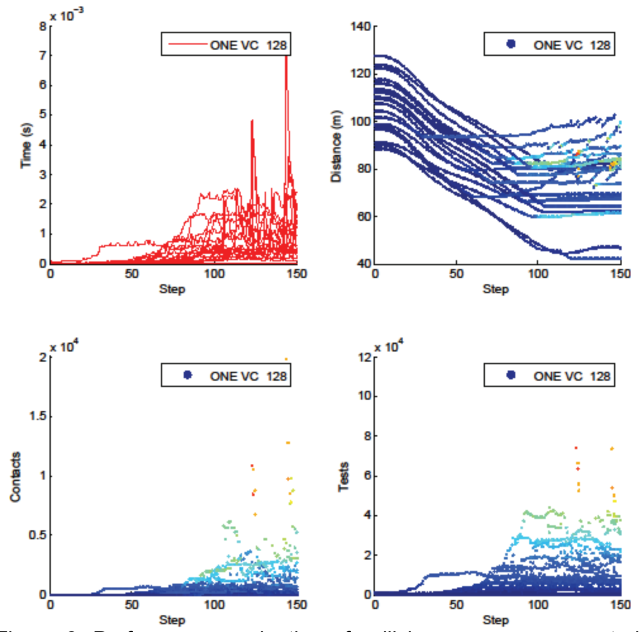

Figure 6. Performance evaluation of collision response generated by random trajectories. Individual simulations are represented with a constant color. Graphs show the distance between the centers of the two voxel models being tested (upper-left), the number of contacts detected between these objects (upper-right), the computation time required for each simulation time step (lower-left), and the number of intersection tests required at each simulation time step (lower-right).

such as GPUs. Additional work will apply the sensation-preserving optimizations presented in [31] to the implicit sphere tree.

Acknowledgments

This work was supported by NIH LM07295, BioX 2DMA178, and by the European Commission within the Enactive Network of Excellence.

References

[1] D. Baraff. Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies. *Proc SIGGRAPH 1989*.

[2] D. Baraff. Fast Contact Force Computation for Non-penetrating Rigid Bodies. *Proc SIGGRAPH 1994*.

[3] P. Berkelman, R. Hollis, and D. Baraff. Interaction with a Realtime Dynamic Environment Simulation using a Magnetic Levitation Haptic Interface Device. *Proc ICRA 1999*.

[4] B. Chang and J. Colgate. Real-time Impulse-Based Simulation of Rigid Body Systems for Haptic Display. *Proc ASME Dynamic Systems and Control Division*, 1997.

[5] E. Chen. Six Degree-of-Freedom Haptic System for Desktop Virtual Prototyping Applications. *Proc First Intl Workshop on Virtual Reality and Prototyping*, 1999.

[6] J. Colgate, M. Stanley, and J. Brown. Issues in the haptic display of tool use. *Proc Intl Conf on Intelligent Robots and Systems*, 1995.

[7] J. Colgate and G. Schenkel. Passivity of a class of sampled-data systems: application to haptic interfaces. *Journal of Robotic Systems,* v14.1, 1997.

[8] D. Constantinescu, S. Salcudean, and E. Croft. Impulsive Forces for Haptic Rendering of Rigid Contacts. *Proc Intl Symp on Robotics*, 2004.

[9] D. Constantinescu, S. Salcudean, and E. Croft. Haptic Rendering of Rigid Contacts Using Impulsive and Penalty Forces. *IEEE Trans on Robotics*, 21(3):309–323, 2005.

[10] F. Conti, F. Barbagli, D. Morris, and C. Sewell. CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes. Demo paper presented at IEEE World Haptics, March 2005.

[11] J. Fiene, K. J. Kuchenbecker, and G. Niemeyer. Event-based haptics with grip force compensation. *Proc IEEE Haptic Symposium*, 2006.

[12] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proc Intl conf on Computer Graphics and Interactive Techniques*, 2000.

[13] A. Fuhrmann, G. Sobottka, and C. Gross. Distance fields for rapid collision detection in physically based modeling. *GraphiCon 2003*.

[14] A. Gregory A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha. Six Degree-of-Freedom Haptic Display of Polygonal Models. In *Proc IEEE Visualization 2000*.

[15] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex Rigid Bodies with Stacking. *ACM Trans on Graphics*, 22(3):871–878, 2003.

[16] R. Gupta, D. Whitney, and D. Zeltzer. Prototyping and Design for Assembly Analysis using Multimodal Virtual Environments. *Computer-Aided Design*, 29(8):585–597, 1997.

[17] S. Hasegawa, N. Fujii, Y. Koike, and M. Sato. Real-time Rigid Body Simulation Based on Volumetric Penalty Method. *Proc IEEE Haptics Symposium*, 2003.

[18] V. Hayward et al. Freedom-7: A high fidelity seven axis haptic device with application to surgical training. In *Experimental Robotics V, Lecture Notes in Control and Information Science 232*, pages 445–456, 1998.

[19] P. M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Trans on Graphics*, 15(3):179–210, 1996.

[20] D. E. Johnson and P. Willemsen. Accelerated Haptic Rendering of Polygonal Models through Local Descent. *Proc IEEE Haptics Symposium*, 2004.

[21] H. Keller, H. Stolz, A. Ziegler, and T. Bräunl. Virtual Mechanics - Simulation and Animation of Rigid Body Systems. Technical Report University of Stuttgart Technical Report TR-1993-08, 1993.

[22] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Six-degree-of-freedom Haptic Rendering using Incremental and Localized Computations. *Presence*, 12(3):277–295, 2003.

[23] K. J. Kuchenbecker, J. Fiene, and G. Niemeyer. Event-based Haptics and Acceleration Matching: Portraying and Assessing the Realism of Contact. *Proc Word Haptics 2005*.

[24] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. *Proc SIGGRAPH 1990*.

[25] W. McNeely, K. Puterbaugh, and J.J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. *Proc SIGGRAPH 1999*.

[26] W. McNeely , K. Puterbaugh, and J.J. Troy. Voxel-Based 6-DOF Haptic Rendering Improvements. *Haptics-e*, 3,7, 2006.

[27] D. Meagher. Geometric Modeling using Octree Encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.

[28] B. Mirtich and J. F. Canny. Impulse-based Dynamic Simulation. *Proc Workshop on Algorithmic Foundations of Robotics*, 1994.

[29] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. *Proc SIGGRAPH 1988*.

[30] D. Nelson, D. Johnson, and E. Cohen. Haptic rendering of surface-to-surface sculpted model interaction. *Proc IEEE Haptics Symposium*, 1999.

[31] M. Otaduy and M. Lin. Sensation preserving simplification for haptic rendering. *Proc SIGGRAPH* 2003.

[32] M. Otaduy and M. Lin. Stable and Responsive Six-Degree-of-Freedom Haptic Manipulation Using Implicit Integration. *Proc IEEE WorldHaptics 2005*.

[33] D. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *Proc SIGGRAPH 1997*.

[34] D. Ruspini and O. Khatib. Dynamic Models for Haptic Rendering Systems. In *Advances in Robot Kinematics (ARK) 1998*.

[35] D. Ruspini and O. Khatib. A framework for multi-contact multi-body dynamic simulation and haptic display. *Proc Intl Conf on Intelligent Robots and Systems*, 2000.

[36] S. Salcudean and T. Vlaar. On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input/Output Device. *Proc ASME Haptics Symposium, 1997*.

[37] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically Deformable Models. *Proc SIGGRAPH 1987*.

[38] S. Trenkel, R. Weller, and G. Zachmann. A Benchmarking Suite for Static Collision Detection Algorithms. In *Proc of the Intl Conf in Central Europe on Computer Graphics, Visualization and Computer Visiion (WSCG)*, January 2007.

[39] C. Tzafestas and P. Coiffet. Real-Time Collision Detection using Spherical Octrees: Virtual Reality Application. 5th IEEE Intl Workshop on Robot and Human Communication, 1996.

[40] M. Wan and W.A. McNeely. Quasi-static approximation for 6 degrees-of-freedom haptic rendering. *Proc IEEE Visualization 2003*.

[41] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[42] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. *Proc Intl Conf on Intelligent Robots and Systems*, 1995.