Poster: Integration of a Haptic Rendering Algorithm Based on Voxelized and Point-Sampled Structures into the Physics Engine Bullet

Mikel Sagardia* German Aerospace Center (DLR)
Theodoros Stouraitis† German Aerospace Center (DLR)
João Lopes e Silva‡ German Aerospace Center (DLR)

Abstract

We present the evaluation of a haptic rendering algorithm based on the Voxmap-Pointshell algorithm which was integrated in the physics engine Bullet. Our algorithm uses voxelized distance fields and multi-resolution point-sampled representations and it is able to compute collision forces between arbitrary complex bodies at 1 kHz. Current experiments show that the integrated algorithm outperforms standard collision detection engines in Bullet in terms of speed if high resolutions and accuracies are required, enabling physically more realistic behaviors.

Index Terms: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms, Object hierarchies; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Animation, Virtual reality.

1 Introduction

Collision detection and force computation are essential for virtual reality applications that require six-DoF haptic interaction, such as interactive gaming, assembly simulations, or virtual prototyping. However, many available solutions have to find a trade-off between the high computational speed required by haptics (1 kHz) and the obtained accuracy by using simplified geometries. This might result in unrealistic simulations.

The Physics Library Bullet [2] provides several collision detection implementations able to handle simple geometries, and a powerful rigid body dynamics framework. Yet, the performance of the library decreases when realistic complex objects are in the virtual scene. Therefore, the implementation of a fast collision detection algorithm would be useful for enabling haptic interactions and simulating realistic multi-body environments.

An extensive state-of-the-art compilation of collision detection and force computation methods is given in [5]. One well-known approach for collision detection is the Gilbert-Johnson-Keerthi (GJK) Algorithm [3], which computes distances or penetrations between convex shapes using their Minkowski sums. This method is implemented in Bullet and tested in our experiments. Some other algorithms build bounding volume hierarchies [4] or segment the objects in convex patches [6]. A convex decomposition approach is also implemented in Bullet and tested in our experiments.

Our collision detection approach is based on the Voxmap-Pointshell (VPS) Algorithm [7], which enables six-DoF haptic rendering between complex geometries. For each colliding object pair, (*i*) *voxelmaps* or voxelized representations and (*ii*) *pointshells* or point-sampled structures are used (see Figure 1). The penetration of the points in the voxelized object can be computed for evaluating the collision force.

The VPS Algorithm was improved in [1] to support deformable objects. In that work, hierarchical data structures and distance fields were used. Similarly, we developed a haptic rendering algorithm for rigid bodies based on the VPS Algorithm which also uses hierarchies and distance fields. However, our data structures are optimized for fast and accurate collision and proximity queries rather than for deformation simulations. Moreover, we additionally compute the contact manifold information as required by Bullet, extending this physics engine with a collision detection method able to perform at haptic rates with complex objects.

*e-mail: mikel.sagardia@dlr.de
†e-mail: theodoros.stouraitis@dlr.de
‡e-mail: jpedro.e.silva@gmail.com

2 Haptic Rendering Algorithm

Given a triangle mesh, layered voxelmap and plain *point-soup* representations of objects are precomputed in the order of magnitude of seconds using the methods presented in [8]. We additionally extend these data structures during the offline precomputation: real distance-field values are stored in the voxels close to the surface and a point-sphere tree is built (down-top).Neighbor points are grouped into clusters in our hierarchies, starting from the leaves of the tree, which are all the points of the initial *point-soup*. The point in the cluster which is closest to its center of mass is set to be the parent point of the cluster; this point belongs to the upper level in the tree, which is also clustered. All points and children points within a cluster are enclosed with a minimally bounding sphere. This hierarchy allows for fast collision area localization by means of the sphere tree and using high point sampling resolutions in a time-critical manner.

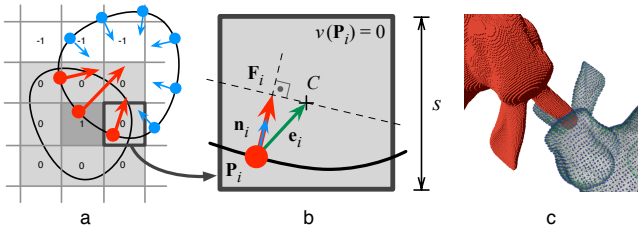

Figure 1: (a) Voxelized and point-sampled objects in collision; Each voxel has its voxel layer value (v) related to its penetration in the voxelmap, and each point ($\mathbf{P}_i$) its inwards pointing normal vector ($\mathbf{n}_i$). (b) Single point force ($\mathbf{F}_i$) can be computed scaling the normal vector with its penetration in the voxelmap in the classical VPS. The cross products of forces and points yield torques. In our implementation, we instead provide the contact manifold ($\{\mathbf{P}_i, \mathbf{n}_i, v(\mathbf{P}_i)\}$) to Bullet. (c) Voxelized and point-sampled Stanford Bunnies colliding.

The realtime collision detection algorithm computes the penetration of the colliding points in the voxelmap, as shown in Figure 1. At the begining of each haptic cycle, the uppermost cluster with the sphere that encloses all points is pushed to a FIFO-queue. Then, the clusters of the queue are iteratively popped in breadth-first manner. In case the popped cluster sphere is colliding, the parent point of the cluster is checked for collision and the children clusters are pushed to the queue. In the classical approach, the single collision force that corresponds to each colliding point is the normal vector scaled with the distance to the surface, as explained in Figure 1. How-

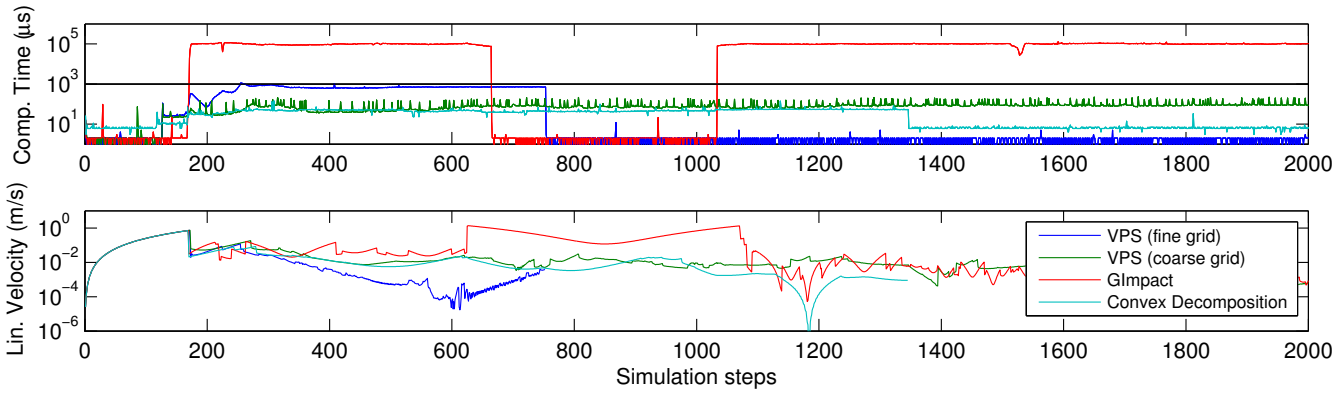

Figure 2: Computation time (μs) and linear velocity (m/s) curves in logarithmic scale for the testing scenario in which a Stanford Bunny with 35606 vertices is dropped onto a plane. The pointshell of the bunny is composed of 799 (coarse) and 35596 (fine) points, and the voxelmap with $306 \times 305 \times 282$ voxels. The decomposed bunny consists of 8 convex hulls with 100 vertices each. Note that Bullet de-activates collision detection under certain kinetic conditions causing sudden steps in the computation time curves. All the tests were carried out using a PC running SUSE Linux Enterprise Edition 11 with an Intel Xeon CPU at 4x2.80 GHz.

ever, we provide Bullet with geometrical contact information (i.e. the contact manifold $\{\mathbf{P}_i, \mathbf{n}_i, v(\mathbf{P}_i)\}$) and let this engine compute the corresponding forces and motion subject to the constraints.

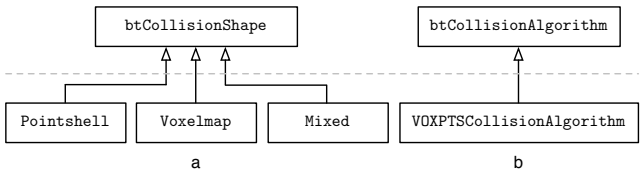

Figure 3: (a) Integration of the three new collision shapes into the pool of collision shapes provided by Bullet: `Pointshell`, `Voxelmap`, and `Mixed`, which contains the previous two. (b) Integration of the `VOXPTSCollisionAlgorithm` by inheriting from Bullet's superclass `btCollisionAlgorithm`. Native Bullet classes (with prefix `bt-`) are above the dashed line. All the implementation was performed in C++.

3 Integration into the Bullet Physics Engine

As shown in Figure 3, the first step in the integration was to extend the supported collision shapes to contain the data structures introduced in the previous section. Since our algorithm requires two different data structures for each colliding pair, we defined a mixed structure containing both structures for each object. The structure which is used is selected online.

The collision detection is divided in two phases in Bullet: (*i*) the broadphase, in which pairs of objects are quickly rejected based on the overlap between their axis aligned bounding boxes, (*ii*) and the narrowphase, in which the appropriate collision detection algorithm is called. Our algorithm is called during the narrow phase, checking first the bounding sphere and successively refining the collision regions.

Finally, we augmented the Bullet collision table by incorporating our collision detection algorithm into the list of collision detection algorithms. Note also that simple manifolds are used in our current implementation rather than persistent manifolds.

4 Experiment Results and Conclusions

Figure 2 shows computation time (μs) and linear velocity (m/s) diagrams produced by our algorithm, the Bullet's convex decomposition, and the Bullet's GImpact, which is used with arbitrary non-convex triangle meshes. During the experiment a Stanford Bunny (35606 vertices) was dropped onto a table (see attached video).

During full operation (steps 250 to 600), our algorithm is 137$\times$ faster than GImpact and requires 0.71 ms for each check on average using the fine resolution (34892 points). The bunny is simplified to a convex hull composed of only 42 vertices in the Bullet's GJK implementation and to 8 convex hulls with 100 vertices each in the convex decomposition approach. With these conditions, GJK is 339$\times$ faster than our algorithm with a fine resolution (34892 points), but the convex decomposition is only 1.3$\times$ faster than our algorithm with a coarse resolution (799 points).

Given these results, we can conclude that, while achieving a higher accuracy, this first integration of our presented haptic rendering algorithm presents similar computation times as the tested ones with low resolutions, whereas it outperforms them when the resolution is increased.

Future work will address, among other topics, modifying the Bullet force constraint solver and extending the time-critical approach of our algorithm to work with the Bullet framework.

References

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

[2] E. Coumans. Bullet physics library 2.82, `http://bulletphysics.org/`. Accessed on December 2nd, 2013.

[3] E. G. Gilbert, D. W. Johnson, and S. S. Keehrthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 1988.

[4] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH '96*, 1996.

[5] M. Lin and M. Otaduy. *Haptic Rendering: Foundations, Algorithms, and Applications*. A K Peters, Ltd., 2008.

[6] K. Mamou and F. Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *IEEE International Conference on Image Processing (ICIP)*, pages 3501–3504, 2009.

[7] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Voxel-based 6-dof haptic rendering improvements. *Haptics-e: The Electronic Journal of Haptics Research*, 3, 2006.

[8] M. Sagardia, T. Hulin, C. Preusche, and G. Hirzinger. Improvements of the voxmap-pointshell algorithm - fast generation of haptic data-structures. In *53. IWK - TU Ilmenau*, 2008.