Continuously-Adaptive Haptic Rendering

Jihad El-Sana[1] and Amitabh Varshney[2]

[1] Department of Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel
`jihad@cs.bgu.ac.il`
[2] Department of Computer Science, University of Maryland at College Park,
College Park, MD 20742, USA

Abstract. Haptic display with force feedback is often necessary in several virtual environments. To enable haptic rendering of large datasets we introduce Continuously-Adaptive Haptic Rendering, a novel approach to reduce the complexity of the rendered dataset. We construct a continuous, multiresolution hierarchy of the model during the pre-processing and then at run time we use high-detail representation for regions around the probe pointer and coarser representation farther away. We achieve this by using a bell-shaped filter centered at the position of the probe pointer. Using our algorithm we are able to haptically render one to two orders of magnitude larger datasets than otherwise possible. Our approach is orthogonal to the previous work done in accelerating haptic rendering and thus can be used with them.

1 Introduction

Haptic displays with force and tactile feedback are essential to realism in virtual environments and can be used in various applications such as medicine (virtual surgery for medical training, molecular docking for drug design), entertainment (video games), education (studying nano, macro, or astronomical scale natural and science phenomena), and virtual design and prototyping (nanomanipulation, integrating haptics into CAD systems). Humans can sense touch in two ways: tactile and kinesthetic. Tactile refers to the sensation caused by stimulating the skin nerves such as by vibration, pressure, and temperature. Kinesthetic refers to the sensation from motion and forces, which trigger the nerve receptors in the muscles, joints, and tendons.

Computer haptics is concerned with generating and rendering haptic stimuli to a computer user, just as computer graphics deals with visual stimuli. In haptic interface interaction, the user conveys a desired motor action by physically manipulating the interface, which in turn provides tactual sensory feedback to the user by appropriately stimulating her/his tactile and kinesthetic sensory systems. Figure 1 shows the basic process of haptically rendering objects in a virtual environment. As the user manipulates the generic probe of the haptic device, the haptic system keeps track of the position and the orientation of the probe. When a probe collides with an object, the mechanistic model calculates the reaction force based on the depth of the probe into the virtual object.

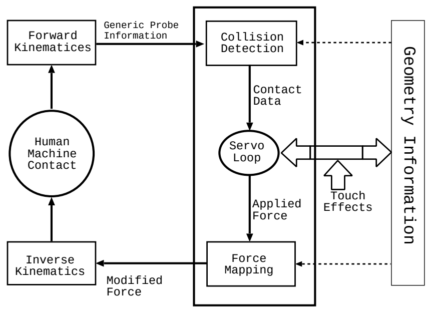

Fig. 1. The haptic rendering processes

Several haptic techniques have been developed to haptically render 3D objects which can have either surface-based or volume-based representation. Haptic interaction in virtual environment could be *Point-based* or *Ray-based*. In point-based haptic interaction only the end-point of the haptic device, known as Haptic Interface Point (HIP), interacts with the objects. In ray-based haptic interaction, the generic probe of the haptic device is modeled as a finite ray. The collision is detected between this ray and the object, and the orientation of the ray is used in computing the haptic force feedback. In this approach the force reflection is calculated using a linear spring law $F = kx$.

In visual rendering several techniques are used to enhance the interactivity and improve the realism of the rendered objects. Smooth shading and texture mapping are good examples of these techniques. Similar algorithms are used in haptic rendering to convey the tactual feeling of the inspected objects. Some of these approaches have been adapted from graphics rendering while others have been developed exclusively for haptic rendering.

2 Previous Work

In this section we overview some of the work done in haptics rendering for virtual environment applications. Haptic rendering has been found to be particularly useful in molecular docking [3] and nanomanipulation [19].

Randolph *et al.* [13] have developed an approach for point-based haptic rendering of a surface by using an intermediate representation. A local planar approximation to the surface is computed at the collision point for each cycle of the force loop. The reaction force vector is computed with respect to the tangent plane. This approach has one major drawback – undesirable force discontinuities may appear if the generic probe of the haptic device is moved over large distances before the new tangent plane is updated. An improvement to this method has been presented by Salisbury and Tarr [17].

Basdogan *et al.*[2] have developed a ray-based rendering approach. The generic probe of the haptic device is modeled as a line segment. They update the simulated generic probe of the haptic device (stylus) as the user manipulates the actual one. They detect collisions between the simulated stylus and the virtual objects in three progressively nested checks: (a) the bounding boxes of the virtual objects, (b) the bounding box of appropriate triangular elements, and (c) appropriate triangular elements. They estimate the reaction force by using a linear spring law model.

Ruspini *et al.* [16] introduce the notion of proxy on the haptic system as a massless sphere that moves among the objects in the environment. They assumed that all the obstacles in the environment could be divided into a finite set of convex components. During the update process, the proxy attempts to move to the goal configuration using direct linear motion.

Gregory *et al.* [8] have developed an efficient system, *H-Collide*, for computing contact(s) between the probe of the force-feedback device and objects in the virtual environment. Their system uses spatial decomposition, a bounding volume hierarchy, and exploits frame-to-frame coherence to achieve a factor of 3 to 20 in speed improvement.

Polygonal or polyhedral descriptions are often used to represent objects in virtual environments. The straightforward haptic rendering of these objects often does not convey the desired shape for the user. Morgenbesser and Srinivasan [15] have developed *force shading*, in which the force vector is interpolated over the polygonal surfaces. Haptic rendering has also been successfully pursued for volumetric datasets [1] and for NURBS surfaces [4]. The sensations of touch have been conveyed to the human tactile system using textures generated by – *force perturbation* and *displacement mapping*. Force perturbation refers to the technique of modifying direction and magnitude of the force vector to generate surface effects such as roughness [14]. In displacement mapping the actual geometry of the object is modified to display the surface details. To improve the realism of the haptic interaction such as the push of a button or the turn of a switch, friction effects have been introduced. Friction can be simulated by applying static and dynamic forces in a direction tangential to the normal force.

2.1 Haptic Rendering

The haptic rendering process involves the following three steps:

- Initializing the haptic device interface and transferring the dataset representation from the user data buffers to the haptic device drivers or API buffers. This step may require translating the data from the user representation to match the haptic API representation.
- Collision detection between the elements representing virtual objects and the probe of the haptic device. Such detection becomes much more complex when the probe has multiple dynamic fingers.
- Estimating the force that the haptic device needs to apply to the user's hand or finger. This force is fed to the generic probe.

We would like to reduce the overhead for the above three steps. Different approaches could be used to achieve this goal. A simple way could be subdivide the dataset into disjoint cells (using octree or any other spatial subdivision) during pre-processing. Then at run-time the cells which are within some threshold distance from the probe pointer are considered in the collision detection and force feedback estimation. This approach has two drawbacks. First, the selected cells may eliminate part of the force field that affects the user. For example, when haptically rendering a surface as in Figure 2 the user may sense incorrect force when using spatial subdivision. Second, if the user moves the probe pointer too fast for the application to update the cells, the user could perceive rough (and incorrect) force feedback. Another approach to reduce the above overhead could be to reduce the complexity of the dataset through simplification. Several different levels of detail could then be constructed off-line. At run time, an appropriate level is selected for each object. However, switching between the different levels of detail at run time may lead to noticeable changes in the force feedback which is distracting. Also, if the objects being studied are very large, this method will provide only one level of detail across the entire object.

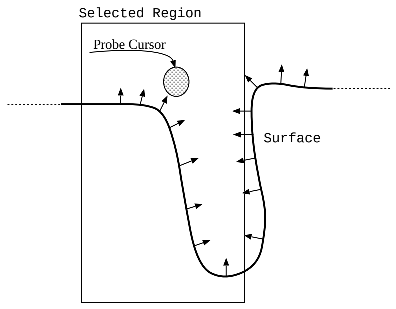

Fig. 2. The use of spatial subdivision may result in incorrect sense of the force field

In this paper we introduce *Continuously-Adaptive Haptic Rendering* – a novel approach to reduce the complexity of the rendered dataset – which is based on the *View-Dependence Tree* introduced by El-Sana and Varshney [6]. We use the same off-line constructed tree and at run time we use a different policy to determine the various levels of detail at the different regions of the surface.

2.2 View-Dependent Rendering

View-dependent simplifications using the edge-collapse/vertex-split primitives include work by Xia *et al.* [20], Hoppe [10], Guéziec *et al.* [9], and El-Sana and Varshney [6]. View-dependent simplifications by Luebke and Erikson [12], and De Floriani *et al.* [5] do not rely on the edge-collapse primitive. Klein *et al.* [11] have developed an illumination-dependent refinement algorithm for multiresolution meshes. Schilling and Klein [18] have introduced a refinement algorithm that is texture dependent. Gieng *et al.* [7] produce a hierarchy of triangle meshes that can be used to blend different levels of detail in a smooth fashion.

View-dependence tree [6] is a compact multiresolution hierarchical data-structure that supports view-dependent rendering. In fact, for a given input dataset, the view-dependence tree construction often leads to a forest (set of trees) since not all the nodes can be merged together to form one tree. The view-dependence trees are able to adapt to various levels of detail. Coarse details are associated with nodes that are close to the top of the tree (roots) and high details are associated with the nodes that are close to the bottom of the tree (leaves). The reconstruction of a real-time adaptive mesh requires the determination of the list of vertices of this adaptive mesh and the list of triangles that connect these vertices. Following [6], we refer to these lists as the list of *active nodes* and the list of *active triangles*.

3 Our Approach

We have integrated view-dependent simplification with haptic rendering to allow faster and more efficient force feedback. We refer to this as as *continuously-adaptive haptic rendering*. Similar to graphics rendering, Continuously-adaptive haptic rendering speeds up the overall performance of haptic rendering by reducing the number of triangles representing the dataset. In our approach we do not need to send the complete surface to the haptic system. Instead, we send a surface with high details in the region close to the generic probe pointer and coarser representation as the region gets far from the generic probe.

3.1 Localizing the View-Dependence Tree

The the construction of view-dependence trees results in dependencies between the nodes of the tree. These dependencies are used to avoid foldovers at run time by preventing the collapse or merge of nodes before others. Therefore, these dependencies may restrict the refinement of nodes, might have otherwise refined to comply with the visual fidelity or error metric. In order to reduce such restrictions we reduce the dependencies between the nodes of the tree. We can reduce the dependencies by *localizing the tree*, which refers to constructing the tree to minimize the distance between the nodes of the tree.

We define the *radius* of a subtree as the maximum distance between the root of the subtree and any of its children. We are currently using the Euclidean distance metric to measure the distance. We can localize a view-dependence tree by minimizing the radius of each subtree. Since we construct the tree bottom-up, the algorithm starts by initializing each subtree radius to zero (each subtree has only one node). Each collapse operation results in a merge of two subtrees. We collapse a node to the neighbor which result in the minimum radius. It is important to note that our algorithm does not guarantee optimal radius for the final tree. In practice, it results in fairly acceptable small radius.

3.2 Levels of Detail

When haptically rendering a polygonal dataset we need to detect collisions between the probe and the dataset and compute the force that the probe supplies the user at very high rates (more that 1000 Hz). The triangles close to the probe contribute more to the force feedback and have a higher probability of collision with the probe. The triangles far from the probe have little effect on the force-feedback and have a smaller probability of collision with the probe.

In our approach we use high-detail representation for regions near the probe and coarser representation farther away. We achieve this by using a bell-shaped filter as in Figure 3(a). In our filter, the distance from the haptic probe pointer dictates the level of detail of each region. This filter could be seen as a mapping of distance from the probe pointer to the switch value (switch value is the value of the simplification metric at which two vertices had collapsed at the tree construction time). The surface close to the probe should be displayed in its highest possible resolution in order to convey the best estimation of the force feedback. In addition, regions far enough from the probe can not be displayed at less than the coarsest level. We were able to achieve further speed-up by changing the shape of our filter from bell-shaped to multiple-frustums shape. This reduces the time to compute the switch value of an active node, which needs to be executed for each node at each frame. Figure 3(b) shows the shape of the optimized fileter. This change reduces the computation of distance (from the probe) and cubic function (which we use to estimate the bell-shaped filter) to find the maximum difference along any of the three axes x, y, and z. We also allow the user to change some of the filter attributes that determine the relation between the level of detail and the distance between the probe pointer.

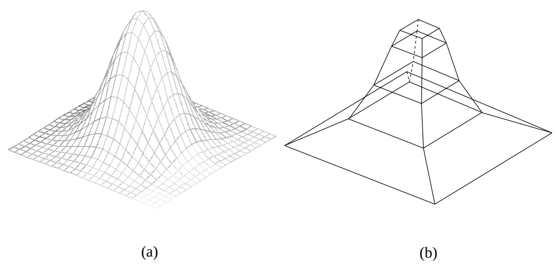

Fig. 3. Ideal verse optimized filter

At run time we load the view-dependence trees and initialize the roots as the active vertices. Then at each frame we repeat the following steps. First, we query the position and the orientation of the probe. Then we scan the list of active vertices. For each vertex we compute the distance from the probe position, determine the mapping to the switch value domain, and then compare the resulting value with the switch value stored at the node. The node splits if the computed value is less than the switch value and the node satisfies the implicit dependencies for split. The node merges with its sibling if the computed value is larger than the switch value stored at the parent of this node and the node satisfies the implicit dependencies for merge.

After each split, we remove the node from the active-nodes list and insert its two children into the active-nodes list. Then we update the adjacent triangle list to match the change; and insert the *PAT* triangles into the adjacent list of the newly inserted nodes. The merge operation is carried out in two steps, first we remove the two merged nodes from the active-nodes list, and then we insert the parent node into the active-nodes list. Finally, we update the adjacent-triangles list by removing the *PAT* triangles list and merging the two merged nodes triangles (the interested reader may refer to the details in [6]). The resulting set of active triangles is sent to the haptic interface.

4 Further Optimizations

We were able to achieve further speedups by fine-tuning specific sections of our implementation. For instance, when updating the active-nodes list and active-triangles list after each step we replace pointers instead of removing and inserting them. For example after split, we replace the node with one of its children (the left one) and insert the second child. Similarly in merge we replace the left child with its parent and remove the other child. Even though the active lists are lists of pointer to the actual nodes of the tree, still their allocation and deallocation requires more time because it relies on the operating system.

The haptic and graphics buffers are updated in an incremental fashion. Since the change between consecutive frames tends to be small, this results in small changes in the haptic and graphics buffers. Therefore, we replace the vertices and triangles that do not need to be rendered in the next frame with the newly added vertices and triangles. This requires very small update time that is not noticeable by the user.

Since the graphics rendering and the haptic rendering run at different frequencies we have decided to maintain them through different processes (which run on different processors for a multi-processor machine). The display runs at low update rates of about 20 Hz, while the haptic process runs at higher rates of about 1000 Hz. We also use another process to maintain the active lists and the view-dependence tree structure. At 20 Hz frequency we query the haptic probe for its position and orientation, then update the active lists to reflect the change of the probe pointer. Finally, we update the graphics and the haptic buffers. In this scenario, the graphics component is updated at 20 Hz and runs at this rate while the haptic component runs at 1000 Hz and is updated at only 20 Hz. To better approximate the level of detail when the user is moving the probe fast, we use the estimated motion trajectory of the probe pointer, and the distance it has traveled since the previous frame to perform look-ahead estimation of the probe's likely location.

5 Results

We have implemented our algorithm in C++ on an SGI ONYX2 with infinite reality. For haptic rendering we have used the *PHANToM* haptic device from SensAble Techologies with six degrees of input and three degree of output freedom. The haptic interface is handled through the GHOST API library (from SensAble Techologies). This haptic device fails (the servo loop breaks) when it is pushed to run at less that 1000 Hz frequency.

Dataset	Triangles	CHR OFF		CHR ON	
		Average Frame rate(Hz)	Average Quality	Average Frame rate (Hz)	Average Quality
CAD-Obj1	2 K	1500	good	1500	good
CAD-Obj2	8 K	600	bad	1200	good
Molecule	20 K	—	breaks	1000	good
Terrain	92 K	—	breaks	1000	good

Table 1. Results of our approach

We have conducted several tests on various datasets and have received encouraging results. Table 1 shows some of our results. It shows results of haptic rendering with (CHR ON) and without (CHR OFF) the use of continuously-adpative haptic rendering. For medium size datasets the haptic device works for some time then fails. When the datasets become larger the haptic device fails almost immediately because it was not able to run at the minimum required frequency. This failure could be the result of failing to finish the collision detection process or the failure to finish the force field estimation process. Reducing the dataset size using our algorithm enables successful haptic rendering of these datasets. Figure 4 shows the system configuration we used in our testing. In our system the mouse and the haptic probe pointer is used simultaneously to change and update the viewed position of dataset. Figure 5 shows high level of detail around the probe pointer (shown as a bright sphere in the center).

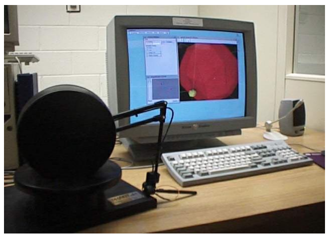

Fig. 4. Our system configuration

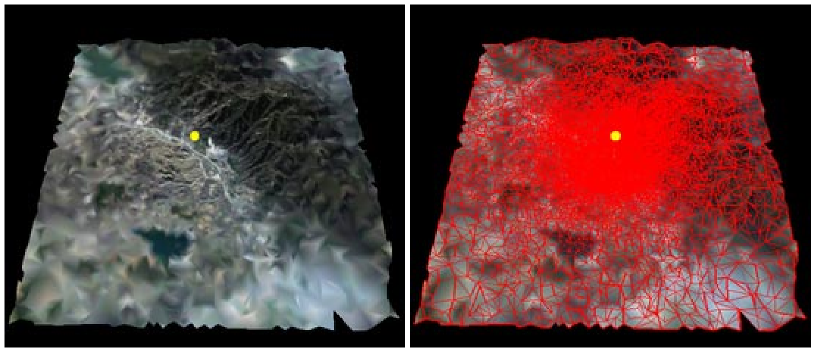

Fig. 5. Haptic rendering of terrain dataset, the yellow sphere is the haptic probe pointer

6 Conclusions

We have presented the continuously-adaptive haptic rendering algorithm, which enables haptic rendering of datasets that are beyond the capability of the current haptic systems. Our approach is based upon dynamic, frame-to-frame changes in the geometry of the surface and thus can be used with any of the prior schemes, such as bounding volume hierarchies, to achieve superior acceleration of haptic rendering. Haptic interfaces are being used in several real-life applications such as molecular docking, nanomanipulation, virtual design and prototyping, virtual surgery, and medical training. We anticipate that our work highlighted in this paper will achieve accelerated haptics rendering for all of these applications.

Acknowledgements

This work has been supported in part by the NSF grants: DMI-9800690, ACR-9812572, and a DURIP award N00014970362. Jihad El-Sana has been supported in part by the Fulbright/Israeli Arab Scholarship Program and the Catacosinos Fellowship for Excellence in Computer Science. We would like to thank the reviewers for their insightful comments which led to several improvements in the presentation of this paper. We would also like to thank our colleagues at the Center for Visual Computing at Stony Brook for their encouragement and suggestions related to this paper.

References

1. R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. In *Proceedings, IEEE Visualization*, pages 197–204, Los Alamitos, October 27–November 1 1996. IEEE.
2. C. Basdogan, C. Ho, and M. Srinivasan. A ray-based haptic rendering technique for displaying shape and texure of 3-d objects in virtual environment. In *ASME Dynamic Systems and Control Division*, November 1997.

3. F. P. Brooks Jr., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick. Project GROPE — haptic displays for scientific visualization. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 177–185, August 1990.
4. F. Dachille IX, H. Qin, A. Kaufman, and J. El-Sana. Haptic sculpting of dynamic surfaces (color plate S. 227). In Stephen N. Spencer, editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 103–110, New York, April 26–28 1999. ACM Press.
5. L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulation. In H. Rushmeier D. Elbert and H. Hagen, editors, *Proceedings Visualization '98*, pages 43–50, October 1998.
6. J. El-Sana and A. Varshney. Generalized view-dependent simplification. In *Computer Graphics Forum*, volume 18, pages C83–C94. Eurographics Association and Blackwell Publishers Ltd 1999, 1999.
7. T. Gieng, B. Hamann, K. Joy, G. Schussman, and I. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, 1998.
8. A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction. Technical Report TR98-032, Department of Computer Science, University of North Carolina - Chapel Hill, November 03 1998. Tue, 3 Nov 1998 17:27:33 GMT.
9. A. Guéziec, G. Taubin, B. Horn, and F. Lazarus. A framework for streaming geometry in VRML. *IEEE CG&A*, 19(2):68–78, 1999.
10. H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 189 – 197. ACM Press, August 1997.
11. R. Klein, A. Schilling, and W. Straßer. Illumination dependent refinement of multiresolution meshes. In *Computer Graphics Intl*, pages 680–687, June 1998.
12. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 198 – 208. ACM SIGGRAPH, ACM Press, August 1997.
13. W. Mark, S. Randolph, M. Finch, J. Van Verth, and R. Taylor II. Adding force feedback to graphics systems: Issues and solutions. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, pages 447 – 452. ACM Press, 1996.
14. M. Minsky, M. Ouh-young, O. Steele, F. P. Brooks, Jr., and M. Behensky. Feeling and seeing: Issues in force display. In *1990 Symposium on Interactive 3D Graphics*, pages 235–243, March 1990.
15. H. Morgenbesser and M. Srinivasan. Force shading for haptic shape preception. In *ASME Dynamic Systems and Control Division*, volume 58, pages 407 – 412, 1996.
16. D. Ruspini., K. Kolarov, and O. Khatib. The haptic display of complex graphical environment. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 345 – 352. ACM SIGGRAPH, ACM Press, August 1997.
17. J. Salisbury and C. Tarr. Haptic rendering of surface defined by implicit functions. In *ASME Dynamic Systems and Control Division*, November 1997.
18. A. Schilling and R. Klein. Graphics in/for digital libraries — rendering of multiresolution models with texture. *Computers and Graphics*, 22(6):667–679, 1998.
19. R. Taylor, W. Robinett, V. Chi, F. Brooks, Jr., W. Wright, R. Williams, and E. Snyder. The nanomanipulator: A virtual-reality interface for a scanning tunnelling microscope. In *Proceedings, SIGGRAPH 93*, pages 127–134, 1993.
20. J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, pages 171 – 183, June 1997.