Hardware Based 2.5D Haptic Rendering Algorithm using Localized Occupancy Map Instance

Jong-Phil Kim and Jeha Ryu

Human-Machine-Computer Interface Lab., Dept. of Mechatronics,
Gwangju Institute of Science & Technology(GIST), Gwangju, 500-712 Korea
{lowtar, ryu}@gist.ac.kr

Abstract

In this paper, we propose a novel 2.5D haptic rendering algorithm for a background scene. The 2.5D haptic rendering is basically same as the full 3D haptic rendering except that it doesn't provide the haptic feedback against the reverse face or hidden surface. It is likely that haptic rendering for the reverse side of the background scene is hardly required. In the proposed algorithm, collision detection and response are performed by using depth values provided by a graphic rendering hardware. The collision is detected by comparing the haptic interaction point with the associated depth value of the background scene. In the collision response, a localized occupancy map instance (LOMI) is used to determine the rendering force. The LOMI is a small-size temporal occupancy map and updated at every haptic rendering rate by referring the depth values around the ideal haptic interaction point. The proposed algorithm provides kinesthetic feels as long as the background scene is rendered by graphic rendering hardware and thus any type of object data is available. It also supports dynamically changing background scene without any additional efforts. Moreover, it uses small amount of physical memory and computation load and is not affected by the complexity of the background scene.

Key words: Haptic Rendering, Virtual Reality

1. Introduction

Haptic rendering is a process to provide a user with tactual sensory information of environments. This allows exploration and interactive manipulation of virtual objects through a haptic interface. Environments may include real, virtual, augmented or multimedia contents and interactions may be accompanied by visual and/or auditory modalities. The kinesthetic or tactile feedback provides very effective interaction tool with the contents, assists to have the full recognition of the environments, and enhances a sense of immersion in the implementations.

Most of the previous works in the haptic rendering algorithm have focused on the full force feedback with well-established object data. The geometric haptic rendering algorithms [1-6] use the surface data representation and the volumetric haptic rendering algorithms [7-9] use the volumetric data representation. Implicit object data [10, 11] is also used.

Early haptic rendering algorithms using penalty methods focused on simple object shapes such as spheres, cubes and planes since it is easy to determine the direction and penetration depth. The penalty method was extended to a method applicable to the more complex objects that can be subdivided into the internal volume. Each sub-volume is correlated with a surface toward which reflection force is exerted [1]. However, the penalty method has some drawbacks: lack of locality, force discontinuity, and apparent penetration through thin objects.

Later, more sophisticated haptic rendering algorithms had been proposed using the constraint-based methods in order to overcome limitations of the penalty method [2]. These methods used a god-object that is constrained by the surfaces of the objects. The god-object method was extended to the virtual proxy [3, 4] that is represented by an object substituting for the physical finger or probe in virtual environments. This method reduced tasks in the haptic servo loop for minimizing error between probe and proxy position.

In the haptic rendering algorithms, that use surface-based representation, pre-computation for a hierarchical data structure is required to perform appropriate collision detection and collision response computation. Moreover, haptic rendering speed depends on the number of polygons. A volume-based data representation can provide much faster and more informative rendering than the surface-based representation. It needs, however, also pre-computation generating voxel data (voxelization) and requires a large physical memory depending on the environment size.

Environments may contain diverse and enormous number of objects depending on each development purpose. However, it is not quite likely that all the objects require full haptic feedback. A picture frame on a wall or a curtain at a window in a virtual environment, for example, the force display or manipulation for its back-face is hardly demanded for general implementations. The objects can be classified into object models and background scene by the required assignment and their importance level. The *object models* are targeted objects which a user focuses on and tries to touch, feel, and manipulate, and thus *full haptic feedback* should be provided through their 3D object data with surface-based or volume-based representation. On the contrary, the *background scene* is less important set of objects which improve realism of the application and aid to perceive circumstantial states of surroundings and vicinities in the environment. In general, the background scene occupies very wide portion of the environment and contains huge number of objects. Figure 1 shows an example of the background scene. The full haptic feedback of the background scene requires excessive amounts of offline processing, on-line computation, and physical memory. However, it looks ineffective considering its role and significance. It is highly likely that haptic rendering for the reverse side of the background scene is hardly required. Thus, it seems that *2.5D haptic feedback* is sufficient for common applications. The 2.5D haptic feedback is same with the full 3D haptic feedback except that it doesn't provide the tactual feels on the reverse or hidden surfaces.

In addition, background scene may be represented by 2.5D data achieved by a z-cam or stereo camera in order to enhance the realism or simplify the modeling process. The 2.5D data is the most commonly recognized data structure where a z value (normally depth or elevation) is recorded as an attribute for each data point (x, y). These z values can be used in a perspective plot to create the 3D appearance or to cull the hidden surfaces. In these applications, the full haptic feedback is impossible only with the raw 2.5D object data.

In this paper, we propose a novel 2.5D haptic rendering algorithm for the background scene. In the proposed algorithm, collision detection and response are performed by using depth values in a graphic rendering hardware. The collision is detected by comparing the haptic interaction point with the associated depth value. In the collision response, a localized occupancy map instance (LOMI) is used to determine the rendering

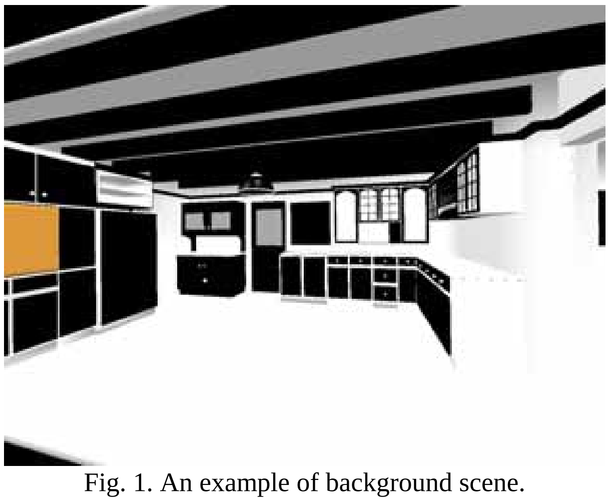


Fig. 1. An example of background scene.

force. The LOMI is a temporal small-size occupancy map and updated at every haptic rendering rate by referring the depth values around the ideal haptic interaction point.

The proposed algorithm provides kinesthetic feels once the background scene is rendered by graphic rendering hardware since the only graphic rendering contexts are referred for the collision detection and response. Thus any type of object data is available such as surface-based and volume-based 3D representation as well as 2.5D data. It also supports dynamically changing background scene, that is usually found in the multimedia contents, without any additional efforts since it does not use any pre-computed data hierarchy such as bounding boxes or voxmap. Moreover, it requires small amount of physical memory and computation load and is not affected by the complexity of the background scene. Thus, we can assign much computation resources to the haptic rendering for object models.

2. Collision Detection

In the proposed haptic rendering algorithm, collision detection is performed by using depth buffer values assigned at each pixel in rendering contexts of a graphic hardware. Each pixel has not only RGB but also depth information that is usually used to cull the hidden surfaces and to create the 3D appearance in a perspective plot. The full 3D haptic rendering requires depth information from six sides. In contrast, the 2.5D haptic rendering for the background scene requires the depth values from only one side since it discards the reverse faces or the hidden surfaces. Thus, the required depth values are achieved by the pixel information without any additional graphic rendering function.

Once the background scene is rendered by the graphic hardware, the location of its front face along the z direction can be achieved by the depth information. If a haptic interaction point (HIP) is located beyond the front face of the background scene, then the collision is detected. Thus, the collision detection algorithm performs the following procedure.

1) Calculate the HIP position by a haptic interface in the object coordinates.
2) Find the pixel corresponding to the x-y coordinate of the HIP.
3) Get the depth value assigned at the pixel.
4) Convert the depth value to object coordinates. It gives the position of the background scene along z at the given x and y position.
5) Compare the converted depth value and z value of the HIP.

Figure 2 shows the collision detection algorithm using graphic hardware in detail. Once the background scene is rendered, the depth values are stored by OpenGL function `glReadPixels` with `GL_DEPTH_COMPONENT` type-command, which reads the frame buffer on the graphic hardware. Then, we find the pixel corresponding to the HIP. The pixel is described in window coordinates and the HIP is described in object coordinates. OpenGL provides a mapping function `gluProject` which transforms the specified the object coordinates into window coordinates using current modelview and projection matrices and current viewport. The transformation matrices and viewport can be obtained by the function `glGetDoublev` and `glGetIntegerv` respectively.

The depth values are normalized values such that the minimum depth value maps to 0.0 and the maximum value maps to 1.0. The near clip plane is mapped to 0.0 and the far clip plane is mapped to 1.0 in default. The normalized depth value is converted to the object coordinates. The relationship between the depth value and the distance in the object coordinates is linear in an orthographic projection but not in a perspective projection. In the case of a perspective projection, the amount of the nonlinearity is proportional to the ratio of far to near. The nonlinearity increases the resolution of the depth value when they are close to the near clip plane. Let d be the normalized depth value and Z_{near} and Z_{far} be the position of the near and the far clipping plane respectively, then the converted depth value (Z) in the object coordinates is calculated by

$$Z = c - \frac{Z_{far} Z_{near}}{Z_{far} - d(Z_{far} - Z_{near})} \tag{1}$$

where c is the position of the eye or the virtual camera. When the z value of the HIP is less than the converted depth value, the collision between the HIP and the background scene is detected. The corresponding code is:

```
   -----------  graphic rendering loops  ----------
glPushAttrib(GL_DEPTH_BUFFER_BIT);
// background scene rendering
BackgroundScene.Render();

// get transformation matrices
glGetDoublev(GL_MODELVIEW_MATRIX, ModelViewMatix);
glGetDoublev(GL_PROJECTION_MATRIX, ProjectionMatix);
glGetIntegerv(GL_VIEWPORT, Viewport);

// get the depth information from corresponding pixel
glReadPixels(pHIP.x, pHIP.y, w, h,
             GL_DEPTH_COMPONENT, GL_FLOAT, BSd);
glPopAttrib();
   ----------------------------------------------


   -----------  haptic rendering loops  -----------
// get the pixel position corresponding to Haptic In-
teraction Point
gluProject(HIP.x, HIP.y, HIP.z, ModelViewMatix,
ProjectionMatix, Viewport, &pHIP.x, &pHIP.y, &pHIP.z);

// convert the depth information to object coordinates
BSz=(Cpos-(Znear*Zfar)/(Zfar-BSd*(Zfar-Znear)));

// collision detection
if(HIP.z > BSz)
        IsCollide=false;
else
        IsCollide=true;
   ----------------------------------------------
```

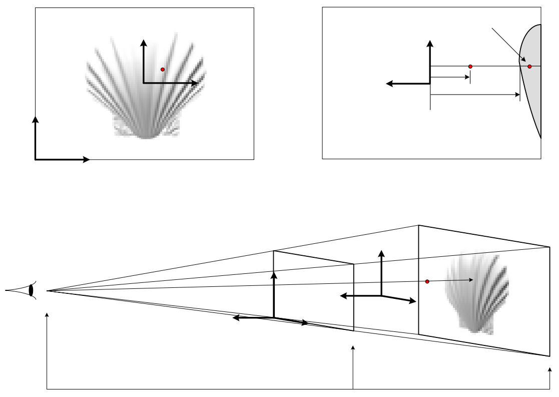


Fig. 2. The collision detection algorithm using graphic hardware. (a) Once the background scene is rendered by graphic hardware, read the depth value corresponding to the HIP. (b) When the HIP is located beyond the front face of the background scene, the collision is detected. (c) The front face position along z direction is achieved by the converted depth value (Z) in the object coordinates.

Note that the collision detection algorithm is proposed for the 2.5D haptic rendering and thus it does not detect the collision between the HIP and the reverse face or the hidden surface of the background scene.

3. Collision Response

Once the collision is detected, appropriate forces are generated as a collision response. The previous haptic rendering algorithms such as penalty methods or constraint-based methods require object data such as surface equation parameters or surface normal vectors. In other words, they require full 3D description of the object data. In the case of the 2.5D background scene acquired by the z-cam or stereo-camera, it requires an enormous amount of the off-line processing for converting the 2.5D data to 3D data. This may introduce difficulties for the use of the dynamically changing background scene. To avoid the shortcomings the proposed algorithm uses a localized occupancy map instance (LOMI) which is derived by the graphic rendering context. The LOMI is updated at every haptic rendering rate. Based on the LOMI, the position of the ideal haptic interaction point (IHIP; also called god-object, proxy point, or surface contact point) is determined. Then the direction and the magnitude of the rendering force are determined by the HIP and the IHIP. The following section goes into details.

In general, an occupancy map (also called voxmap) is used to the haptic rendering algorithm for the voxel-based data representation. The occupancy map is a voxelized (or discretized) rectangular 3D grid which is composed of memory cells. Each memory cell is assigned to each voxel and contains type or address of the assigned voxel. In the proposed algorithm, we create a localized occupancy map instance (LOMI) which is a temporal small-size occupancy map and centered on the IHIP. It follows the IHIP and is updated at every haptic rendering rate. The LOMI is composed of memory cells containing voxel type which is classified into free space, surface, and interior. Figure 3 shows the LOMI in detail.

In order to update the LOMI, we perform a voxelization procedure similar to the depth buffer based voxelization algorithm [12] which uses depth values associated to the whole objects by six virtual cameras. However, the LOMI is created by the depth values from one side and thus it does not require any additional graphic rendering function. Moreover, it does not require much computation time since it is performed at the localized region around the IHIP instead of the whole objects. Therefore, it can be updated at every haptic rendering rate in real-time.

The proposed LOMI is created and updated by following procedures;

1) Decide the LOMI size which is the number of voxels and allocate physical memory for the LOMI.
2) Initialize the LOMI. It makes all the values in the assigned physical memory be zero.
3) Find the pixel corresponding to the IHIP using `gluUnproject`.
4) Get the depth values of the rectangular region. The center of the rectangle is the IHIP and the dimension is the LOMI size. Note that we don't need to execute `glReadPixels` since we already read the depth values for the collision detection.
5) Determine the voxel types through the depth values. The voxels inside the object are assigned as the interior voxel.

Note that step2 to step5 are performed with the updated IHIP at every haptic rendering rate.

4. Implementation

The proposed algorithm discussed in this paper is implemented by 6DOF PHANToM device. The haptic and the graphic rendering rate are 1 KHz and 60 Hz respectively. In this implementation, the size of LOMI is 5 and thus the LOMI is composed of 5x5x5 voxels. It enables to use very small physical memory and computation load regardless of the background scene complexity. Since the proposed algorithm refers to the only graphic rendering contexts, any object data is available for the haptic rendering without any pre-computation. Thus, the implementation is performed with following three types of objects.

4.1 Primitive objects

First, we apply it to primitive objects supported by the OpenGL, which are glutSolidCube and glutSolidSphere. In CSG applications, complex models are constructed by combining primitive objects. Once the Cube and Sphere is graphically rendered, then a user can touch them without any additional efforts. In figure 4, the small sphere on the big sphere indicates the IHIP and the red line expresses the rendered force by the haptic device. The generated force is normal to its contact surface.

4.2 Background Scene with 3D Representation

Second, we apply it to the background scene with 3D objects. In figure 5, the background scene is composed of triangular meshes. The haptic rendering does not refer to the object data. Only graphic rendering uses them. The small sphere on the bed indicates the IHIP and the red line expresses the rendered force by the haptic device. The proposed algorithm provides kinesthetic feels on visible surfaces.

4.3 Background Scene with 2.5D Representation

Finally, we apply it to the background scene with 2.5D data. In figure 7, the background scene is composed of RGB data and depth data which are shown in figure 6. The small sphere on the floor and the wall indicates the IHIP and the red line expresses the rendered force by the haptic device. In augmented reality applications, 2.5D data is commonly used. Thus, we can expect the proposed algorithm can be used in AR environment to provide haptic feedback without any additional computation.

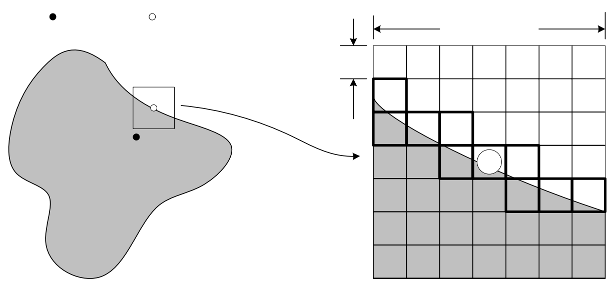


Fig. 3. The localized occupancy map instance which is voxelized data structure. (a) The LOMI is small-size occupancy map centered at the IHIP. (b) The LOMI uses 2-bit voxel types.

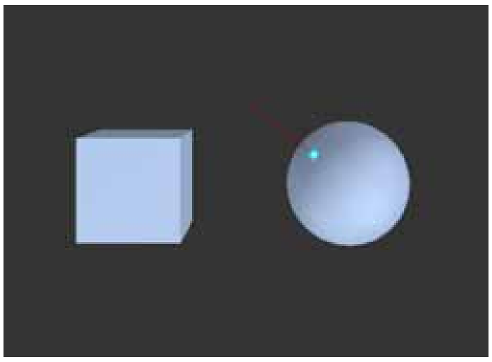


Fig. 4. Haptic rendering with primitive objects. The objects is created by OpenGL commands such as glutSolidCube and glutSolidSphere.

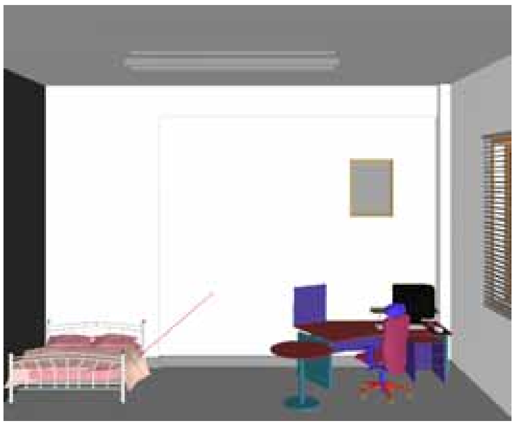
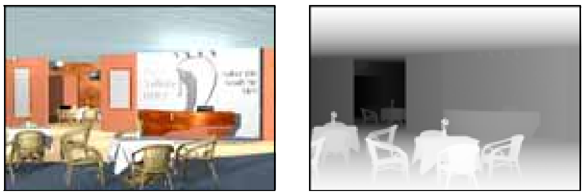


Fig. 5. Haptic rendering with background scene composed of 3D objects. The objects are represented by triangular mesh structure. Only graphic rendering refers to the object data.


(a) (b)
Fig. 6. 2.5D data sets. (a) color map, (b) depth map

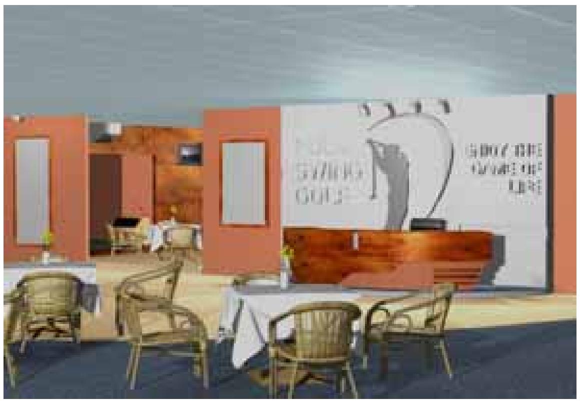


Fig. 7. Haptic rendering with background scene composed of 2.5D data sets.

5. Conclusion and Future Work

This paper introduces a novel 2.5D haptic rendering algorithm for the background scene. In this, a collision is detected by the graphic hardware and thus very fast collision detection with no computational intensity is possible. It also uses very small memory by grace of the constraint-based haptic rendering algorithm with very small-size localized occupancy map instance. Thus, we can assign much computation power and memory to the haptic rendering for object models.

Besides, any type of object data is available such as surface-based and volume-based 3D representation as well as 2.5D object data since the only graphic rendering contexts are referred for the collision detection and the force generation. Once the background scene is rendered by graphic device, therefore, we can touch it.

In addition, it does not use any pre-computed hierarchy of object data such as bounding boxes or voxmap. Thus, we can reduce time and effort assigned to the off-line processing and moreover we can achieve excellent ability for the haptic rendering of the static as well as dynamically changing background scene which is usually found in the multimedia contents. In future, we will use it for the 3D broadcast application.

Acknowledgement

This work was supported by the Ministry of Information and Communication (MIC) through the Realistic Broadcasting IT Research Center (RBRC) at Gwangju Institute of Science and Technology (GIST).

References

1. Massie, T.M., Salisbury, J.K., “The PHANToM Haptic Interface: A Device for Probing Virtual Objects.” ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems, Volume 1, pp.295-301, 1994.

2. Zilles, C.B., Salisbury, J.K., **“**A constraint-based god-object method for haptic display**”,** Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on, Volume 3, pp. 146-151, 1995.

3. K. Salisbury, et al., “Haptic rendering: programming touch interaction with virtual objects” Symposium on Interactive 3D Graphics, pp. 123-130, 1995.

4. Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib., “The haptic display of complex graphical environment”, In SIGGRAPH 97 conference proceedings, Volumn 1, pp. 295-301, 1997.

5. Chih-Hao Ho, Cagatay Basdogan, mandayam A. Srinivasan, "Efficient Point-Based Rendering Techniques for Haptic Display of Virtual Object", Presence, Volume. 8, No. 5, 1999

6. A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin and D. Manocha. “Six Degree-of-Freedom Haptic Display of Polygonal Models”, Proc. Visualization 2000, pp139-146, October 2000, Utah, United States.

7. Sarah Frisken, "Beyond Volume Rendering: Visualization, Haptic Exploration, and Physical Modeling of Voxel-based Objects", Mitsubish Electric Research Laboratories, TR95-04, 1995

8. R. S. Avila, L. M. Sobierajski, “A Haptic Interaction Method for Volume Visualization” IEEE Visualization, 1996.

9. William A. McNeely, Kevin D. Puterbaugh, James J. Troy, “Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling”, SIGGRAPH ’99 Proceeding of the 26th Annual Conference on Computer Graphics, ACM, pp401-408, 1999

10. T. Maneewarn, B. Hannaford, D. Storti, M. Ganter, “Haptic Rendering For Internal Content Of An Implicit Object,” ASME Winter Annual Meeting Haptics Symposium, Nashville, TN, Nov. 1999

11. Laehyun Kim; Kyrikou, A.; Sukhatme, G.S.; Desbrun, M., “An implicit-based haptic rendering technique”**,** Intelligent Robots and System, 2002. IEEE/RSJ International Conference , Volume 3, pp. 2943-2948, 2002.

12. Karabassi, Evaggelia-Aggeliki, Papaioannou, Georgios and Theoharis Theoharis, “A fast depth-buffer-based voxelization algorithm”, Journal of graphics tools, Volume 4, number 4, pp. 5- 22, 1999.

13. Bayakovski, L. Levkovich-Maslyuk, A.Ignatenko, A. Konushin, D. Timasov, A. Zhirkov, M. Han, I. K. Park, “Depth Image-based Representations for Static and Animated 3D Objects**”**, International Conference on Image Processing (ICIP02), Vol. 3, pp. 25-28, 2002.