DIBHR: Depth Image-Based Haptic Rendering

Jongeun Cha[1], Mohamad Eid[2], and Abdulmotaleb El Saddik[2]

Multimedia Communications Research Laboratory
University of Ottawa, Canada
[1] jcha@discover.uottawa.ca,[2] {eid,abed}@mcrlab.uottawa.ca
http://www.mcrlab.uottawa.ca

Abstract. This paper presents Depth Image-Based Haptic Rendering (DIBHR), a haptic rendering algorithm that enables users to haptically explore 3D video media based on depth image-based representation (DIBR). The algorithm computes the shortest proxy (god-object) path along which the proxy goes into the local distance minimum to the goal (haptic interaction point) constrained on surface in order to obtain correct friction force when the friction cone algorithm is applied. This algorithm is based on the god-object [3] concept and adopted the neighborhood search algorithm [5][6]. The experiments compare DIBHR with two previous algorithms [5][6] as per computation time and smoothness of resultant force rendering. The results show slower computation time yet within 1 millisecond and smoother force with friction.

Keywords: Haptic rendering algorithm, depth image.

1 Introduction

The research of multimedia systems has reached the limit with what can be done with audio and video information. Nowadays researchers have fostered their interest to integrate the sense of touch in networked multimedia systems, fueled by several motivations. For instance, haptics is crucial for interpersonal communication as a means to express affection, intention or emotion; such as a handshake, a hug or physical contact [11]. Furthermore, the potential of haptics as a new way of learning (tele-mentoring) over a network has been acknowledged by several studies [12][13]. One limiting factor of using haptics in networked multimedia systems is the critical bandwidth requirements, especially with non dedicated networks such as the Internet.

In most haptic applications, polygonal meshes are widely used to represent 3D virtual objects. However, in multimedia applications, those representation methods are not appropriate because of massive amount of data with limited network bandwidth, progressive transmission, and compression. In order to bridge the gap between image-based modeling [1] and full 3D modeling, a depth image-based representation was proposed [2]. The 3D video media are the combination of general color video and synchronized grey-scale depth video (depth image sequences) containing per-pixel depth information, as shown in Fig. 1. The grey-level of each pixel in the depth image indicates the distance from a camera to the pixel. The higher the level is, the closer the distance to the camera. This modeling is referred to as 2.5D representation since the depth image has incomplete 3D geometrical information and is complete only from the view of the camera. Nonetheless, the viewers can at least touch what they can see.

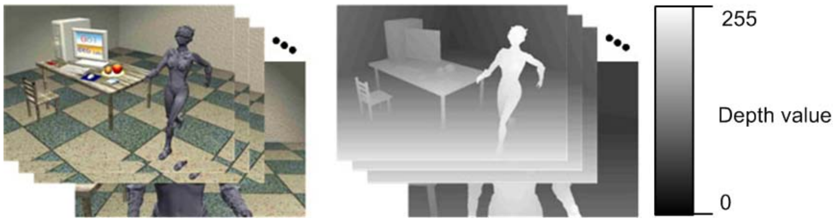

Fig. 1. Depth image-based representation with color and synchronized depth images

In this paper, we propose a 3 degree-of-freedom (DOF) haptic rendering algorithm to touch the depth image. The algorithm is formulated from the constrained based proxy (god-object) concept [3][4] with local search and transition method for updating the proxy position [5][6][7]. Abstractly, the algorithm is explained based on general triangular meshes and then optimized to fit the depth image by using the characteristics of the depth image: (1) each point above or below the depth image is projected on only one unique surface point of it and (2) the depth image itself encodes the topology information of each triangle. In order to provide surface properties, friction is applied by using a friction cone algorithm [7] and roughness is implemented by perturbing the depth values based on a bump map.

2 Background and Motivation

Haptic rendering algorithm is the process of computing and generating forces in response to interaction between the haptic device and the virtual environment. 3-DOF haptic rendering algorithm restricts the user's avatar to a single point of

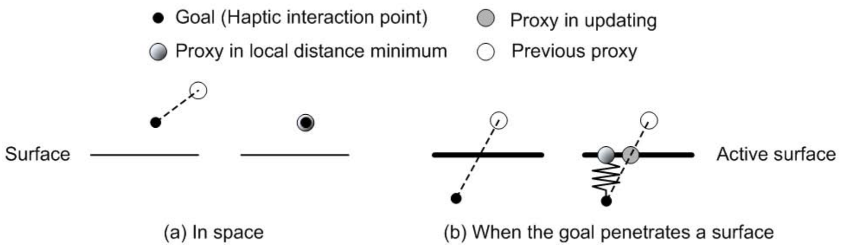
Goal (Haptic interaction point)
Proxy in updating
Proxy in local distance minimum
Previous proxy
Surface
Active surface
(a) In space
(b) When the goal penetrates a surface

Fig. 2. Proxy-based algorithm

interaction. Several algorithms were proposed, for instance a proxy based 3-DOF algorithm for polygonal meshes was first introduced by Zilles and Salisbury [3]. A proxy or god-object, that is an ideal massless point that can not penetrate any surface, is connected to a goal point that represents the position of the haptic device in a virtual environment. The proxy and goal objects are connected through an ideal spring that has zero to infinity length. In a haptic loop, when the goal is moved, the proxy is updated to a location with minimum local distance to the goal (because of the ideal spring). In other words, if the goal is in a space or the path of the goal (a line segment to previous proxy to the goal) does not collide with any object, the proxy coincides with the goal as shown in Fig. 2(a), and if the goal object penetrates a surface or the line segment collides with a surface, the collided surface is set as active and the proxy is updated to the closest location to the goal constrained on a plane that contains the active surface as shown Fig. 2(b).

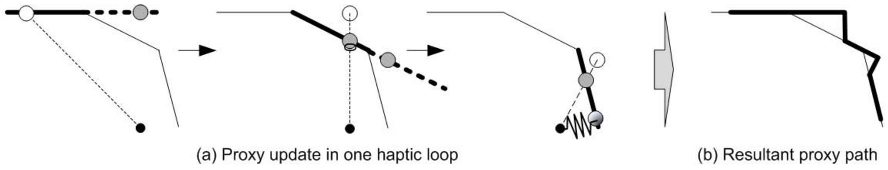

Fig. 3. Transitioning active surfaces in god-object algorithm

However, the god-object algorithm causes floating and sticking problems when a user explores many surfaces in one haptic loop (see [6] for detailed discussion). These problems can be resolved by updating the active constraints in one haptic loop as shown in Fig. 3(a), which in turn makes the algorithm time-expensive. In addition, on the convex surface, the path of the proxy can be in free space as shown in Fig. 3(b). This can result in a final proxy location in free space and distort the resultant force when the friction cone algorithm is applied.

Ho et al. [5] resolved these problems by adopting local neighborhood search. The first step is to construct a hierarchical database to store the geometrical properties of the 3D object and the information of neighboring primitives. When a collision is detected, the contacted primitive is set to be active and then its neighbors are searched to find nearest primitive to the goal. This time, the nearest neighboring primitive is set as active and this process is repeated until the active primitive is nearest to the goal than any other neighboring primitives as shown in Fig. 4(a). As a result, the proxy path in updating remains on object's surface as shown in Fig. 4(b). In this algorithm, the proxy path connects the nearest points on active primitives.

Walker and Salisbury [6] restricted the primitives into vertices to speed up computation time for large model such as topographic map, namely Proxy Graph Algorithm (PGA). However, both algorithms induce distortions in the proxy path. Assume there is a flat surface with a goal penetrating the surface from the

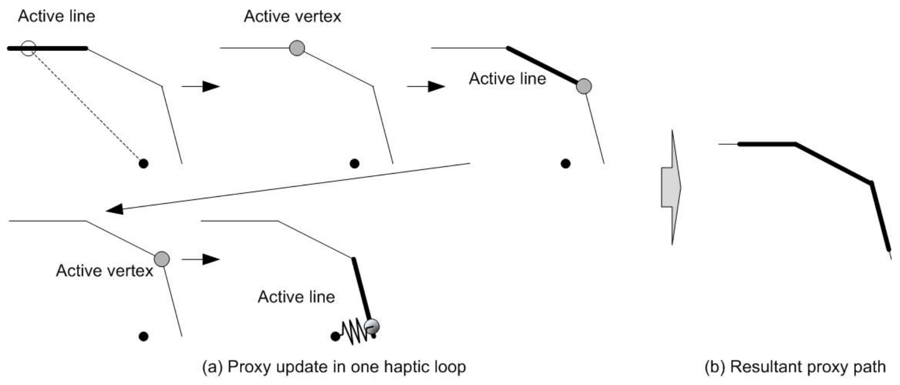

Fig. 4. Neighborhood search and transitioning active primitives

top view as shown in Fig. 5. The proxy location is determined on the flat surface just above the goal location although the proxy paths obtained from each algorithm are different. If the surface has friction, the proxy should stop where the proxy get into the friction cone on the proxy path. In this case, the proxy paths computed from the neighborhood search and the PGA are not accurate and thus the resultant forces can be distorted as shown in Fig. 5(b)(c). In order to render the friction properly, the correct proxy path should be computed (Fig. 5(a)). This paper proposes DIBHR, a haptic rendering algorithm that computes the correct proxy path.

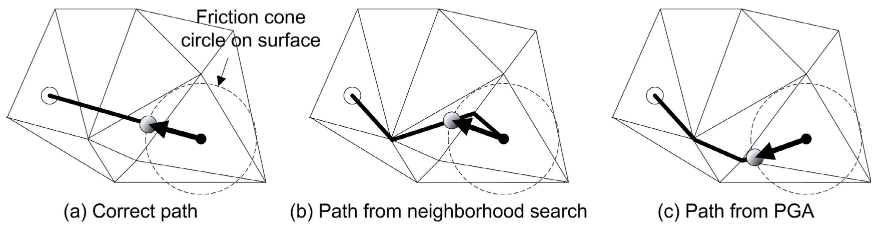
Friction cone circle on surface
(a) Correct path
(b) Path from neighborhood search
(c) Path from PGA

Fig. 5. Proxy paths computed from different algorithms and resultant force directions

3 Algorithm Description

3.1 Overview of Algorithm

In order to simplify the haptic rendering process, three types of primitives are defined: TRIANGLE, EDGE and VERTEX. Each primitive contains its geometrical information and neighborhood primitives' information. TRIANGLE has three vertices and normal as geometry information and three EDGEs as neighbors. EDGE has two vertices and two TRIANGLEs. VERTEX has a vertex and six EDGEs. In order to avoid redundant overlap, the TRIANGLE and the VERTEX are related through EDGE.

The core of DIBHR is to search for new proxy location that minimizes the distance to the haptic interaction endpoint and eventually find out the shortest path along which the proxy traces to the new proxy location. When a collision is detected between a triangle and the line segment that connects the goal and the proxy, the proxy is moved on the obstructing triangle at the collided position and the current triangle is set to an active primitive as TRIANGLE.

Once a primitive is active, the neighborhood search algorithm is started. The first procedure is to determine whether the proxy will go into the space or not. In order to avoid redundant overlapping computation, this procedure is performed at TRIANGLE only. It means that the proxy can go into free space through TRIANGLE only. Then, the algorithm computes the candidate of the new proxy location. If the candidate location is not on the primitive and goes over any neighbor, the active primitive is updated to the neighbor primitive and the proxy will be located at the collided position. If the candidate location is on the active primitive, it becomes a new proxy location in local minimum. These processes are repeated until the proxy location is obtained at local minimum. Fig. 6 depicts a complete flow chart outlining the algorithm. Following subsection explains the detailed procedures on each primitive.

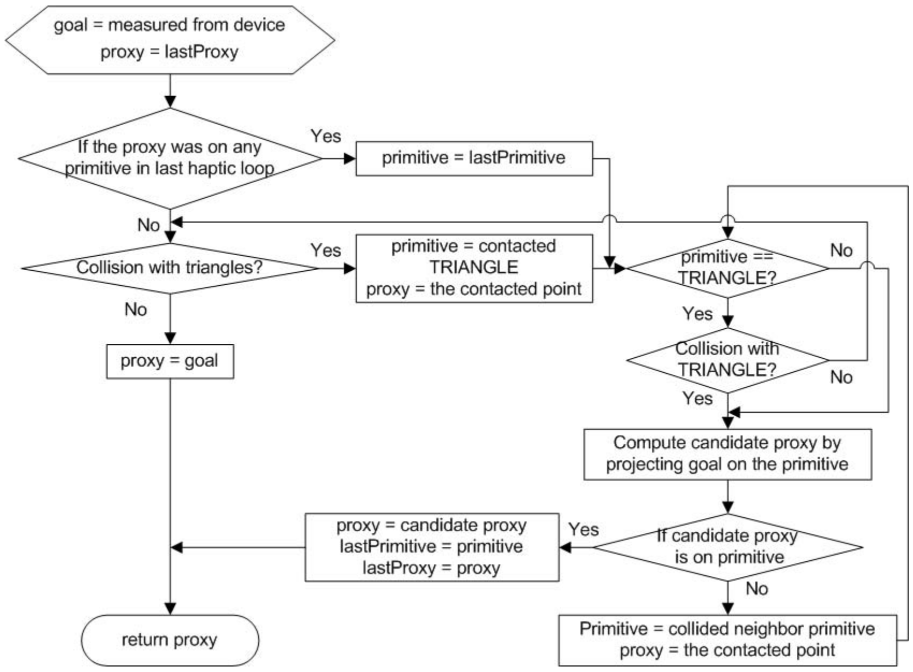
goal = measured from device
proxy = lastProxy
If the proxy was on any primitive in last haptic loop
Yes
primitive = lastPrimitive
No
Collision with triangles?
Yes
primitive = contacted TRIANGLE
proxy = the contacted point
No
proxy = goal
primitive == TRIANGLE?
No
Yes
Collision with TRIANGLE?
No
Yes
Compute candidate proxy by projecting goal on the primitive
If candidate proxy is on primitive
Yes
proxy = candidate proxy
lastPrimitive = primitive
lastProxy = proxy
No
Primitive = collided neighbor primitive
proxy = the contacted point
return proxy

Fig. 6. Complete flow chart outlining DIBHR

3.2 Updating Proxy Location on Each Primitive

On TRIANGLE, we first check whether the goal is inside object's surface by computing the dot product between the vector from the proxy to the goal and the normal of the TRIANGLE. If it is larger than zero, the proxy goes into free space.

When the goal is inside the surface, a candidate proxy is determined by projecting the goal position on the plane that includes TRIANGLE. If the candidate proxy is inside TRIANGLE, it becomes the new proxy in local minimum as shown Fig. 7(a). Otherwise, the proxy should stop at the EDGE that the proxy path collides with as shown in Fig. 7(b). This procedure is performed by checking the collision between the line segment, PC, and three neighbor EDGEs, respectively. If there is no collision, the proxy will stay in TRIANGLE. Otherwise, active primitive is transitioned to the colliding EDGE.

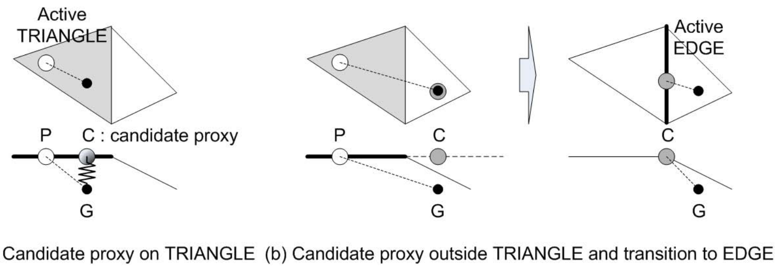
Active TRIANGLE
P
C : candidate proxy
G
P
C
G
Active EDGE
C
G
(a) Candidate proxy on TRIANGLE
(b) Candidate proxy outside TRIANGLE and transition to EDGE

Fig. 7. Updating candidate proxy and transition to EDGE on TRIANGLE

On EDGE, the proxy can go onto four neighbors, two TRIANGLEs and two VERTEXs. First, we check if the proxy can go onto the TRIANGLEs. Each EDGE has two normal vectors (m1 and m2 as shown in Fig. 8) that points towards each neighboring TRIANGLE perpendicular to the EDGE and parallel to each TRIANGLE. In order to determine which TRIANGLE decreases the distance from the proxy to the goal at a faster rate, the distance gradients of the normals (m1, m2) and the normalized vectors from the goal to the proxy are compared. The TRIANGLE that has smaller gradient is set to be active (Fig. 8(a)). Since the proxy is at the TRIANGLE already, it does not change in this transition.

If the two gradients are positive, two TRIANGLEs can not decrease the distance as shown in Fig. 8(b) and then the proxy slides along the EDGE. The candidate proxy location is obtained by projecting the goal onto the line that contains the EDGE. If the candidate is on the EDGE, the candidate location becomes a new proxy location in local minimum. Otherwise, the VERTEX on the path from the proxy to candidate location becomes active and the new proxy becomes the VERTEX as shown in Fig. 8(b).

On VERTEX, the distance gradients for each EDGE are compared. The EDGE that has the smallest gradient becomes active and the proxy does not

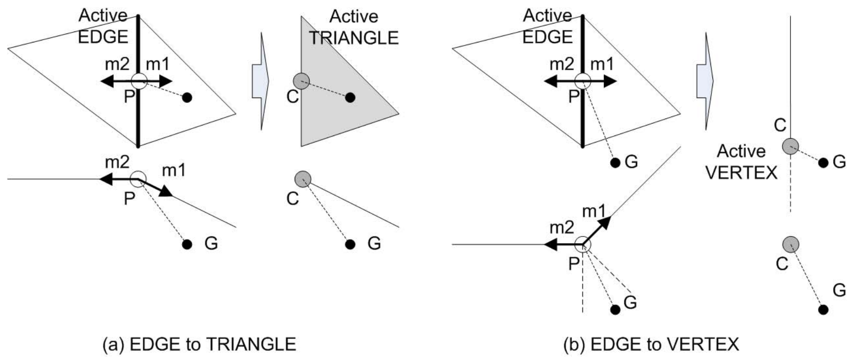

Fig. 8. Updating candidate proxy and transition to TRIANGLE and VERTEX on EDGE

change because the proxy is already at the EDGE, as shown in Fig. 10(a). If all gradients are positive, the point of the VERTEX becomes a new proxy location in local minimum as shown in Fig. 9(b).

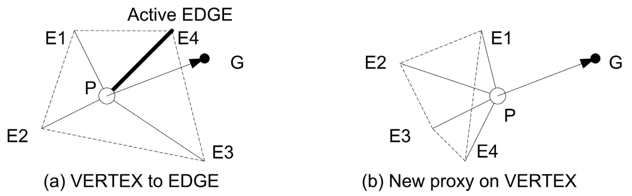

Fig. 9. Updating candidate proxy and transition to EDGE on VERTEX

3.3 Friction

The friction force is computed by using the friction cone algorithm [7]. As described earlier, the proxy is updated to new location during transitions between primitives except for transitions between EDGE and TRIANGLE. Before the proxy is actually updated in each transition, we check whether the path of the proxy meet with a friction cone. If so, the proxy stops at the intersection of the path and the friction cone and it becomes the new proxy location in this haptic loop.

3.4 Application to Depth Image

In this section, the proposed algorithm is applied to the depth image. The collision detection is optimized using the characteristics of the data structure of the depth image as described in [6]. The depth image dataset consists of an evenly spaced 2D array of elevations which correspond to gray-scale pixels. A triangle-based surface can be derived from this array by adding horizontal, vertical and

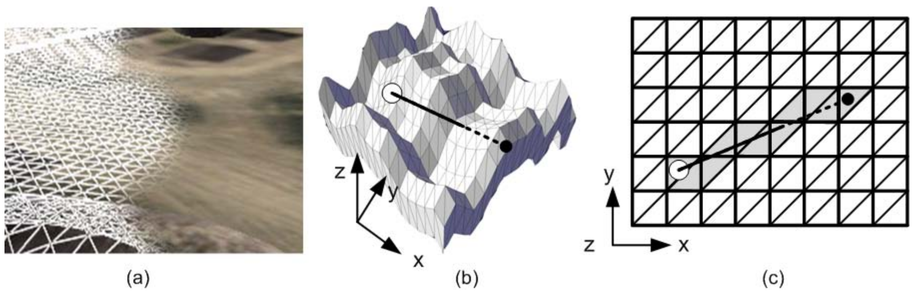

Fig. 10. Triangulation of depth image and optimized collision detection

diagonal lines between each element as shown in Fig. 10(a). In the collision detection process, the line segment between the goal and the previous proxy is projected onto the 2D representation of the depth image in order to generate a list of candidate triangles which should be checked for collisions (shaded area in Fig. 10(c)). Then, cells within this candidate list possessing elevation values below that of the line segment can be discarded. These optimizations yield an algorithm which executes rapidly when applied to a depth image. In addition, the optimized collision detection does not need bounding boxes that should be pre-computed and memory consuming.

When the proposed algorithm is applied to general polygonal meshes, the neighborhood information of each primitive should be pre-processed. However, since the triangles of the depth image are uniformly located from the top view as shown in Fig. 10(c), the neighborhood information is already encoded in the depth image itself. As shown in Fig. 11 (a)(b)(c), there are two types of TRIANGLEs, three types of EDGEs and one type of VERTEX in a depth image. If an active primitive is set, its neighboring primitives can be set without any processing (Fig. 11(d)(e)(f)). Therefore, the neighboring primitives are set in real-time preventing pre-processing and additional memory usage.

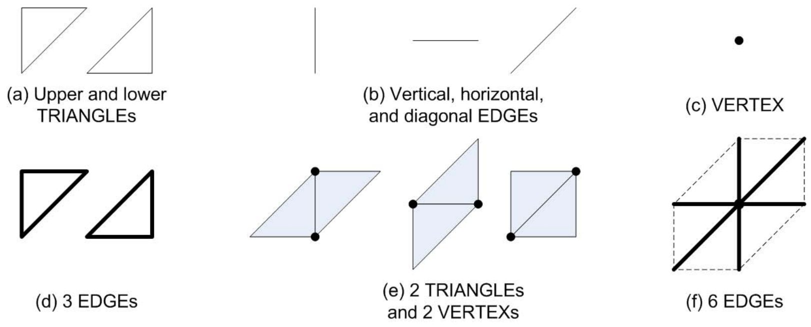
(a) Upper and lower TRIANGLEs
(b) Vertical, horizontal, and diagonal EDGEs
(c) VERTEX
(d) 3 EDGEs
(e) 2 TRIANGLEs and 2 VERTEXs
(f) 6 EDGEs

Fig. 11. Each primitive and its neighbors in depth image

4 Performance Evaluation

The proposed algorithm and the previous algorithms were implemented on an Intel based PC (Pentium®D 3.4GHz, 1 GB RAM, Intel®946GZ Express Chipset) under Windows XP. As a haptic device, the Novint Falcon was used [8]. Three experiments were conducted to evaluate and compare the performance (computation time and friction force) of DIBHR with the PGA [6] and the Neighborhood algorithms [5]. The computational time for each haptic update was measured using a high-resolution timer provided in the Windows XP.

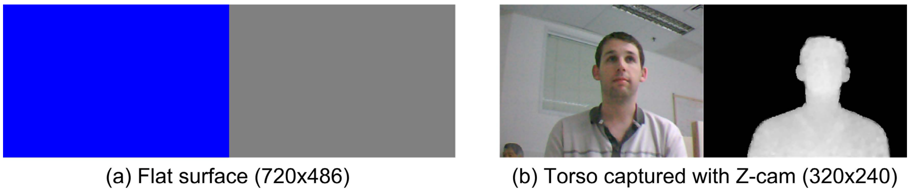
(a) Flat surface (720x486)
(b) Torso captured with Z-cam (320x240)

Fig. 12. Color and depth images for experiment

In the first and second experiments, the goal location is simulated to move across the flat depth image in Fig. 12(a) along linear (back and forth along x-direction) and circular paths (200 pixels radius). The speed of movement was set to 4961 pixel/s based on a normal male adult's hand speed, 700 mm/s, Falcon's workspace (40"×40"×40") and standard definition's depth image resolution (720×486×256). In order to span the whole workspace, 720 pixels along x-direction were mapped to 40". In the third experiment, a human subject is asked to grab the Falcon device and freely explore a depth image (320×240) captured by Z-Cam [9] in Fig. 12(b) based on DIBHR. The position data of the human subject were stored and then applied to the other two algorithms to simulate same user exploration. The computation time and the resultant force were measured for 20 seconds and the force variation (the direction difference between the current force and the previous force) was calculated.

Table 1 shows computation time, average and standard deviation of force variation for each algorithm. As expected, the PGA was the fastest because the proxy moves across the vertices only. However, DIBHR operates comfortably within a 1 millisecond range producing a stable force. Note that the reason why PGA show slower computation in human manipulation is that sometimes the line segment between the proxy and the goal became long compared to other algorithms and it made collision detection process slower. As for force variation, DIBHR has the lowest values. This implies that DIBHR can render significantly smoother friction forces. Note that, in linear path experiments, although the force variations were zero in all algorithms, PGA and Neighborhood algorithms computed wrong friction force directions.

Table 1. Computation time(μs) and force variation($^\circ$, average(standard deviation))

	Computation time			Force variation		
Algorithms	Linear	Circle	Human	Linear	Circle	Human
DIBHR	188.61	207.31	94.21	0(0)	0.53(0.85)	1.81(4.13)
PGA	43.38	47.40	80.97	0(0)	4.13(6.77)	3.01(7.97)
Neighborhood	159.22	161.31	81.74	0(0)	7.45(10.48)	4.74(6.39)

5 Conclusion and Future Work

This paper presented a haptic rendering algorithm to touch 3D video contents. The algorithm computes the correct proxy path that minimizes the distance from the proxy and the goal and then produces smooth friction forces. The performance of DIBHR was compared with two other algorithms. Even though DIBHR is more computationally expensive than the other two algorithms, the smoothness of the rendered friction forces has significantly improved. This implies that the haptic rendering is more stable and transparent.

As per future work, we are planning to test the performance of DIBHR using real-time video contents based on our previous work [10]. Furthermore, a usability analysis will be conducted to see whether users would be able to recognize the differences in the rendering quality between DIBHR and the others.

References

1. Shum, H., Kang, S., Chan, S.: Survey of Image-Based Representations and Compression Techniques. IEEE Trans. Circuits and Systems for Video Technology 13(11), 1020–1037 (2003)
2. Kauff, P., Cooke, E., Fehn, C., Schreer, O.: Advanced Incomplete 3D Representation of Video Objects using Trilinear Warping for Novel View Synthesis. In: Proc. Picture Coding Symp., pp. 429–432 (2001)
3. Zilles, C.B., Salisbury, J.K.: A Constraint-Based God-Object Method For Haptic Display. In: Proc. IEE/RSJ Int. Conf. Intelligent Robots and Systems, vol. 3, pp. 146–151 (1995)
4. Ruspini, D.C., Kolarov, K., Khatib, O.: The Haptic Display of Complex Graphical Environments. In: Proc. ACM SIGGRAPH, pp. 345–352 (1997)
5. Ho, C., Basdogan, C., Srinivasan, M.A.: Efficient Point-Based Rendering Techniques for Haptic Display of Virtual Objects. Presence: Teleoperations and Virtual Environments 8(5), 477–491 (1999)
6. Walker, S.P., Salisbury, J.K.: Large Haptic Topographic Maps: MarsView and the Proxy Graph Algorithm. In: Proc. ACM SIGGRAPH, pp. 83–92 (2003)
7. Melder, N., Harwin, W.S.: Extending the Friction Cone Algorithm for Arbitrary Polygon Based Haptic Objects. In: Proc. Int. Symp. Haptic Interfaces for Virtual Environment and Teleoperator Systems, pp. 234–241 (2004)
8. Novint Technologies, `http://home.novint.com/`
9. 3DV Systems, `http://www.3dvsystems.com/`

10. Cha, J., Kim, S., Ho, Y., Ryu, J.: 3D Video Player System with Haptic Interaction Based on Depth Image-Based Representation. IEEE Trans. Consumer Electronics 52(2), 477–484 (2006)
11. Brave, S., Dahley, A.: inTouch: a medium for haptic interpersonal communication. In: Proc. of ACM CHI 1997, pp. 363–364. ACM Press, Atlanta (1997)
12. Basdogan, C., De, S., Kim, J., Manivannan, M., Kim, H., Srinivasan, M.A.: Haptics in minimally invasive surgical simulation and training. IEEE Computer Graphics and Application 24(2), 56–64 (2004)
13. Eid, M., Mansour, M., Iglesias, R., El Saddik, A.: Haptic Multimedia Handwriting Learning System. In: Proc. of EMME 2007, Augsburg, Germany (September 2007)