Proxy Method for Fast Haptic Rendering from Time Varying Point Clouds

Fredrik Rydén, Sina Nia Kosari, Howard Chizeck

Abstract—This paper proposes a novel algorithm for haptic rendering from time varying point clouds captured using an Xbox Kinect RGB-D camera. Existing methods for point-based haptic rendering using a proxy can not directly be applied since no information about the underlying objects is given. This paper extends the notion of proxy to point clouds. The resulting haptic algorithm can successfully render haptic forces from point clouds captured in real-time representing both static and dynamic objects.

I. INTRODUCTION

Haptic interaction is the translation of forces in a virtual environment to a physical device. Haptic devices can be categorized as impedance and admittance type. In this paper we will consider impedance type haptic devices. The generation of forces is referred to as Haptic Rendering.

Recent advancements in RGB-D cameras have resulted in low cost and low latency cameras with good resolution. An example of this is Xbox Kinect. This camera gives a depth value for each pixel and using these values one can build a set of points in space, referred to as a point cloud. A point cloud derived from Kinect appears as a swarm of time varying points in space, updated in real-time at 30 Hz.

The depth information captured by the camera makes it possible to perform haptic rendering on the point cloud. While there exists a rich body of literature on haptic rendering for virtual objects with defined surfaces, little work has been done on haptic rendering of virtual environments defined by real-time point clouds.

Haptic rendering from points was first developed for virtual environments with complex meshes, with the goal of improving the computational performance [7]. In that work, the point map, also known as the Voxmap PointShell is created by sampling of meshes. As a consequence, each point contains additional information regarding its corresponding mesh, such as the surface normal. This information is not available in point clouds captured by RGB-D cameras.

Haptic rendering from RGB-D data was first investigated by Cha et al. [2]. Their method requires pre-processing of recorded camera data and therefore is not suitable for real-time rendering of dynamic environments.

One could argue that the simplest solution would be to construct surfaces and meshes of the whole point cloud at every frame, and then use traditional haptic rendering algorithms. This would require a large number of computations. The method developed in this paper avoids that requirement.

A similar idea was presented in Ruspini et al. [8] where the massless point was replaced with a massless sphere with some volume. This sphere is referred to as the proxy and serves the same purpose as the god-object as shown in Fig. 1.

II. BACKGROUND

A. Haptic Interaction Point and Proxy

A Haptic Interaction Point (HIP) is to a haptic device what a mouse pointer is to a computer mouse. Ideally the HIP should not be able to penetrate objects. However, since we only have control over the force applied to the HIP (and thus the user), and not over its position (which is generated by the user) this is not possible. For this reason, consider instead a virtual massless point that represents the ideal position of the HIP on the surface, i.e. it can not penetrate objects. Such a point was suggested by Zilles and Salisbury [9] and was named a god-object even though it actually is a point. This massless point is useful for haptic rendering in environments where shapes are well-defined. A common way to render haptic forces is to attach a spring between the HIP and the ideal point on the surface. The spring constant can then be set to an appropriate value.

A similar idea was presented in Ruspini et al. [8] where the massless point was replaced with a massless sphere with some volume. This sphere is referred to as the proxy and serves the same purpose as the god-object as shown in Fig. 1.

Fig. 1. Illustration of the proxy method in two dimensions. Left: The proxy (big circle) and HIP (small filled circle) in free motion. Middle: At the moment of contact, the HIP continues to penetrate the object while the proxy is constrained by the surface. Right: The HIP is inside the object and proxy remains on the surface. F indicates the force on the HIP.

B. RGB-D cameras

Recent advancements in the gaming industry have reduced the cost for RGB-D cameras by more than an order of magnitude while providing acceptable latency and depth resolution. RGB-D cameras capture images similar to a regular video
camera and in addition also capture a depth image containing depth values for each pixel. This can be done using different techniques and the most common are active stereo, time-of-flight and projected pattern.

The camera used in this paper is Xbox Kinect. The four important parts of the Kinect for this work are:

1) The infrared (IR) projector that projects a defined dot pattern (see Fig. 2).
2) The IR camera that captures the reflections of the projected IR dot pattern. This is then used to create a depth mapping.
3) A regular (RGB) camera that captures an image that can be superimposed onto the depth mapping.
4) Accelerometers that obtain the X-Z-orientation of the camera.

Processing is performed on the captured dot pattern resulting in a 640x480 matrix $M(u,v)$ containing a depth value for each pixel. The depth values range between 0 and 2047 (11 bits) with 1 cm depth resolution and 3 mm horizontal/vertical resolution at 2 m distance. Capturing of data and signal processing is done in real-time at the rate of 30 Hz.

C. Interpretation of Kinect Depth Data

In order to render haptic forces, depth data needs to be represented in a Cartesian coordinate system. We denote Kinect’s coordinate system by $\{0\}$ and the Cartesian coordinate system by $\{C\}$. The transformation between the two coordinate systems is done by first by rotating, translating and scaling the data with respect to the camera intrinsics. Then the data is rotated with respect to the cameras X-Z-orientation as given by the accelerometers. Note that the two transformations are invertible.

III. HAPTIC RENDERING FOR POINT CLOUDS

In this paper, the term point cloud is used to denote a set of points (representing physical objects) captured by an RGB-D camera. No other information about the underlying objects is given such as surface normals or information about what object a specific point in the cloud corresponds to.

This is different from the established haptic rendering methods for Voxmap Pointshells where known meshes are sampled and therefore surface normal information is available.

Fig. 3 shows a Kinect derived point cloud visualized in OpenGL. The black spots corresponds to regions without depth information.

In this section a haptic rendering algorithm is developed to extend the notion of proxy to time varying point cloud data derived from an RGB-D camera. The goal of this algorithm is illustrated in Fig. 4.

In order to render the haptic forces the following is done:

1) Depth data from the RGB-D camera is captured.
2) The position of the haptic device is read and the HIP moves accordingly.
3) The proxy is moved towards the HIP with respect to point cloud constraints using the Proxy Movement Algorithm.
4) Forces are rendered based on the distance between the proxy and the HIP.

The camera captures depth data at a rate of 30 Hz. The haptic device sends position and receives forces at a rate of 1000 Hz. The rate of the Proxy Movement Algorithm is variable.

Fig. 4. Two dimensional illustration of haptic rendering from time varying point clouds (where haptic rendering is based on points rather than surfaces as in Fig. 1). Left: The proxy (big circle) and HIP (small filled circle) in free motion. Middle: At the moment of contact, the HIP continues to move through the points while the proxy does not. Right: The HIP is behind the points and the proxy remains on an estimated surface. F indicates the force on the HIP.
A. The Proxy

The proxy has three possible states: free motion, contact and entrenchment (the proxy digging into the point cloud). Consider the following three radii \( R_1, R_2, R_3 \) extending out from the center of the proxy \( P_{proxy} \) (see Fig. 5). If there are any points within \( R_1 \) of \( P_{proxy} \), the proxy is said to be entrenched. If there are any points between \( R_1 \) and \( R_2 \) (but not within \( R_1 \)) the proxy is said to be in contact. If there are no points within \( R_2 \) the proxy is said to be in free motion. Points within \( R_3 \) will be used to estimate surface normals.

![Fig. 5. Proxy regions defined by \( R_1 < R_2 < R_3 \).](image)

The quantity \( R_2 - R_1 \) is chosen to be just larger than the anticipated noise level. Large values will degrade contact detection. Radius \( R_3 \) is chosen sufficiently large to get good surface normal estimation. If \( R_3 \) is chosen too large, the sharpness of the surface normal estimation will be reduced.

B. Selection of Points for Surface Normal Estimation

Having defined regions of points around the proxy we now consider how to move the proxy in virtual 3-space. The surface represented by the points needs to be locally estimated in order to constrain the proxy movement while in contact, so that only movements perpendicular to (or away from) the estimated surface are allowed. To do the surface normal estimation, one could simply search the entire point cloud. However this requires excessive computations and slows down the haptic rendering. Instead we will select a subset of points \( \{C P_{sub}\} \) containing only points in the proximity of the proxy.

This point selection is done by performing the following three steps on every Kinect frame at 30 Hz.

1) Create a bounding cube with sides of length \( L \) around the proxy (in the Cartesian coordinate system) and transform this cube to Kinect’s coordinate system.
2) The transformed cube can now be projected onto the depth data matrix \( M(u,v) \). Then, select \( \{0 P_{sub}\} \) as those points in the \( M(u,v) \) within the boundaries of the projected cube.
3) Finally, create the set \( \{C P_{sub}\} \) by transforming all the points in \( \{0 P_{sub}\} \) to the Cartesian coordinate system.

This method of selection will reduce the number of points used for surface normal estimation.

C. Surface Normal Estimation

The haptic rendering process is at 1kHz. During each cycle (1ms) the proxy is moved iteratively in steps of \( \delta > 0 \). The number of iterations depends on the speed of the computer. Initially the proxy and the HIP coincide and the proxy is in free motion. The surface normal \( \hat{n} \) is estimated at each iteration as follows:

\[
\hat{n} = \frac{1}{N} \sum_{k=1}^{N} \frac{P_{proxy} - P_k}{||P_{proxy} - P_k||}
\]

D. Proxy Movement Algorithm

The goal of the Proxy Movement Algorithm is to move the proxy such that it tracks the HIP with respect to the estimated surface defined by \( \hat{n} \).

The algorithm is implemented using the following steps. \( v \) denotes the vector between the proxy and the HIP.

1) If the proxy is in free motion (as described in Section III-A), move the proxy one step of length \( \delta \) along \( v \) (the direction of the HIP).
2) If the proxy is entrenched, move one step in the direction of \( \hat{n} \) (see Fig. 6).
3) If the proxy is in contact or entrenched and the HIP is outside of the estimated surface, move one step along \( v \) (see Fig. 7).
4) If the proxy is in contact or entrenched and the HIP is inside the estimated surface, project \( v \) on the plane defined by \( \hat{n} \) \((v_{plane})\). Then move one step in this direction (see Fig. 8).
5) Iterate steps 1 to 5.

![Fig. 6. Estimation of surface normal.](image)

This iteration is continued until the end of current haptic cycle (1ms). This gives the proxy position for next haptic cycle.

E. Force Rendering

The resulting proxy position is used to render the force to the user. A virtual spring is attached between the HIP and the midpoint between \( R_1 \) and \( R_2 \) (see Fig. 9).

![Fig. 7. The proxy is in contact. Since the HIP is moving out of the surface the proxy will move in the direction of \( v \).](image)

![Fig. 8. The proxy is in contact and the HIP is inside the estimated surface, in this case the proxy will move in the direction of \( v_{plane} \).](image)
The force acts only on the HIP and only when the HIP is outside the midpoint between \( R_1 \) and \( R_2 \) as given by Eq. (2)

\[
F_{\text{HIP}} = -v \frac{||v||}{||v||} K_{\text{Spring}} (||v|| - \frac{R_1 + R_2}{2})
\]  

Fig. 9. Illustration of force acting on the HIP. Note that the left side of the virtual spring is attached to the midpoint between \( R_1 \) and \( R_2 \).

Fig. 9 is theoretically correct but in practice the computations are in discrete time for this marginally stable system. An artifact of the discretization, since the spring has no accompanying damping, is a jittery rendering of force during contact [3]. This can be compensated for by adding a damping \( B \) during contact to the force calculation. Essentially this damping ties the HIP to the reference frame.

IV. EXPERIMENTAL EVALUATION

In this section, we demonstrate the performance of the haptic rendering.

A. The Setup

The haptic rendering is tested using an XBox Kinect\textsuperscript{TM} depth camera, a Phantom Omni (Sensable Inc.) haptic device and two desktop PC’s (see figure 10).

The Virtual Environment consists of software developed to perform the haptic rendering and the graphic visualization that runs on a desktop computer (Intel Core 2 Quad, 4 GB RAM, Nvidia Geforce 7100). The other computer is connected to the Virtual Environment through a network connection.

All the points in the matrix \( M(u, v) \) are transformed (as in Section II-C) and visualized in the Virtual Environment. Only points selected in Section III-B are transformed and used in the haptic rendering. The transformation in the first case is performed on the graphics card while the latter is performed on the CPU.

![Diagram of experimental setup](image)

Fig. 10. Experimental setup.

B. Evaluation of Haptic Rendering

The proxy in Fig. 11 has a radius of 5\textit{mm} and moves in steps of \( \delta = 1\textit{mm} \). Table I contains a list of parameters used.

In a typical haptic scene the Proxy Movement Algorithm runs between 10 and 100kHz with an average of 80Hz. It used roughly 0.5% of the point cloud (about 1500 points).

If all points in the point cloud were used for surface normal estimation (instead of just a subset as described in Section III-B) the rate of the Proxy Movement Algorithm drops to 200Hz.

This 400-fold improvement allows us to achieve the haptic rendering rate of 1kHz recommended by Basdogan and Srinivasan [1].

<table>
<thead>
<tr>
<th>Table I: Parameters used in Experimental Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
</tr>
<tr>
<td>( R_2 )</td>
</tr>
<tr>
<td>( R_3 )</td>
</tr>
<tr>
<td>Proxy Movement Step Size (( \delta ))</td>
</tr>
<tr>
<td>Bounding Cube Side Length (( L = R_2 \times 4 ))</td>
</tr>
<tr>
<td>Spring Constant (( K_{\text{Spring}} ))</td>
</tr>
<tr>
<td>Damping Constant (( B ))</td>
</tr>
</tbody>
</table>

![Image of proxy and HIP](image)

Fig. 11. The proxy and the HIP next to a point representation of an apple for size comparison. The pin sticking out of the proxy indicates the estimated normal \( \hat{n} \).

V. RESULTS

Evaluation of the accuracy and the correctness of the haptic rendering from vision is difficult since true forces that represent physical contact can not be defined.

Fig. 12 compares the position of the proxy developed in this paper with the HIP as the user moves the HIP into a virtual surface. The HIP penetrates the surface, the proxy does not. That is, the “pop-through” problem for point cloud haptic rendering does not appear here. The bottom trace of Fig. 12 shows the force applied to the user. The three peaks are caused by the user trying to push through the surface.

The movement of proxy at a very high rate causes noisy force signals. In order to suppress this noise a 26 point FIR lowpass filter is applied. The other source of noise is the noise...
on the depth signal with a standard deviation of 0.82mm (at 0.45m depth).

The haptic rendering algorithm allows interaction with convex and concave surfaces as well as sharp corners (see Fig. 13). Fig. 14 shows the y-position of the proxy while it is moving off a 5mm ledge.

This approach to haptic rendering allows interaction with moving point clouds. To illustrate this, the HIP was remained at rest while a flat surface was moved in positive y-direction (up) in front of the Kinect, forcing the proxy away from the HIP. The distance between the proxy and the HIP was measured continuously. An average velocity of 0.08m/s was achieved before the proxy popped-through the object (see Fig. 15).

VI. Conclusion

By extending the proxy method, this paper developed and implemented a novel method for haptic rendering from time varying point clouds where surface normals are unknown. The ability to remotely feel objects in real-time using non-contact devices can potentially be useful in a wide range of applications, such as tele-operation. This paper demonstrates feasibility for these applications.