A Proxy Method for Real-Time 3-DOF Haptic Rendering of Streaming Point Cloud Data

Fredrik Rydén, *Student Member*, *IEEE*, and Howard Jay Chizeck, *Fellow*, *IEEE*

Abstract—This paper presents a new haptic rendering method for streaming point cloud data. It provides haptic rendering of moving physical objects using data obtained from RGB-D cameras. Thus, real-time haptic interaction with moving objects can be achieved using noncontact sensors. This method extends "virtual coupling"-based proxy methods in a way that does not require preprocessing of points and allows for spatial point cloud discontinuities. The key ideas of the algorithm are iterative motion of the proxy with respect to the points, and the use of a variable proxy step size that results in better accuracy for short proxy movements and faster convergence for longer movements. This method provides highly accurate haptic interaction for geometries in which the proxy can physically fit. Another advantage is a significant reduction in the risk of "pop through" during haptic interaction with dynamic point clouds, even in the presence of noise. This haptic rendering method is computationally efficient; it can run in real time on available personal computers without the need for downsampling of point clouds from commercially available depth cameras.

Index Terms—Haptic rendering, streaming point cloud data, point cloud velocity estimation

1 Introduction

This paper presents a method for real-time 3-DOF haptic rendering of streaming point clouds. Using point clouds to represent physical objects has recently gained popularity because of the availability of inexpensive RGB-D cameras such as Kinect (Microsoft Corp.). The point clouds representing physical objects are captured and streamed to a computer. Using haptic interacting with these point clouds, "one-way remote touch" is achieved. "Two-way remote touch" would be possible using the haptic rendering method presented here, in combination with a teleoperated robot to implement virtual fixtures.

There exist several challenges in haptic interaction with streaming point cloud data. The most critical challenge is the latency induced in the haptic rendering. This has to be kept to a minimum in order for the haptic rendering to be perceived by the user as being in real time. This imposes a computational speed requirement on any haptic rendering method for moving objects, used in real-time interactions. A second challenge is to correctly render the smallest possible geometries for a given point cloud density (spatial sampling rate). The haptic resolution (the level of detail the user will be able to feel) is highly dependent on the density of the point cloud. Haptic interaction with point cloud representations of physical objects is nontrivial because no information is given regarding which object a point belongs to or surface normals. A third challenge in the design of a haptic rendering algorithm is to be robust to point cloud movements as well as noise.

• *The authors are with the Department of Electrical Engineering, University of Washington, 185 Stevens Way, Paul Allen Center, Room AE100R, Campus Box 352500, Seattle, WA 98195-2500.*
E-mail: {ryden, chizeck}@uw.edu.

Manuscript received 25 Apr. 2012; revised 8 Feb. 2013; accepted 25 Feb. 2013; published online 7 Mar. 2013.
Recommended for acceptance by M. Otaduy.
For information on obtaining reprints of this article, please send e-mail to: toh@computer.org, and reference IEEECS Log Number TH-2012-04-0032.
Digital Object Identifier no. 10.1109/TOH.2013.20.

In this paper, we present a method for haptic rendering that addresses the aforementioned challenges by extending the idea of a haptic proxy to streaming point clouds. These challenges were first addressed in [1], where an iterative proxy method was implemented. This paper extends that work by presenting a new method for choosing a proxy step size as well as a novel method for estimating point cloud velocity. The use of a variable proxy step size yields improved haptic accuracy as well as fewer iterations (and hence faster computation). The point cloud velocity estimation reduces the risk for incorrect haptic rendering, and "pop through" when in contact with a moving point cloud, by correcting the proxy position at every point cloud update.

2 Related Work

There exist several well-established methods for three degree-of-freedom haptic rendering. Most of them are based upon haptic rendering of polygons, mainly because this traditionally has been the format for shapes in computer graphics. The increased availability of RGB-D cameras and laser scanners has changed this because the output from these devices are point clouds.

This work builds upon prior work in haptic rendering and point set surface estimation. An innovation in this work is the extension of traditional haptic rendering methods to real-time interaction with streaming point clouds without the need for tesselation or preprocessing of the point cloud.

2.1 Traditional Methods for Haptic Rendering

In haptic interaction, an operator uses a physical *haptic device* to interact with objects in a virtual space. This virtual space can be a representation of physical objects, as represented by sensor data. Or it could be purely virtual—produced by computer simulation. The *Haptic Interface Point* (or *HIP*) can be thought of as the position of an end effector in virtual space. It matches the position of

the haptic device. When the user moves the haptic device, the HIP moves accordingly in the virtual environment.

Forces can then be exerted by the environment to simulate touch sensations with objects. The simplest method for haptic rendering is the *direct method*, where the force on the haptic device is based on the distance between the HIP and the surface. In some cases this works well, but upon interaction with thin shapes these methods fail and the HIP *pops through* the surface. That is, the HIP moves through a surface when it actually should be constrained by the surface.

In the mid-1990s, several researchers developed alternatives to direct methods for haptic rendering. Most of these are classified as *virtual coupling* methods [2]. The god-object method [3] and the proxy method [4] are two widely used virtual coupling methods for 3-DOF haptic rendering of polygons. By using these methods, the HIP did not pop through thin shapes. The main idea was to construct a virtual object (sphere or point) that could not penetrate shapes. This object was variously denoted as a *proxy*, a *god-object* or an *ideal HIP* (IHIP). The virtual object was then tied to the HIP by a virtual spring (this is the virtual coupling), thus specifying the force exerted on the user by the haptic device. The movement of the object relative to the HIP was then determined using either a *constraint-* or *penalty-based* simulation.

2.2 Point Clouds and RGB-D Data

A point cloud can be thought of as an unstructured set of points representing the boundary of physical objects. It can be obtained using stereo cameras, laser scanners or depth cameras [5] in which case the data will be sorted in the image plane. RGB-D data are an extension of the point cloud structure by augmenting the data with color information.

2.3 Haptic Rendering of Static Point Clouds

Haptic rendering of point clouds has been investigated by several groups. In [6], the first method for haptic rendering explicitly from a point set was presented. This point set was sampled from highly detailed meshes and, therefore, contained surface normal information, which is different from discontinuous point clouds without surface normals. A direct rendering method for point clouds was presented in [7]. Their method required construction of a surface using a moving least square (MLS) method [8]. In [9], a method for haptic rendering of point clouds was implemented by placing a small bounding box around each point. They then used a tree-structure for rapid collision detection.

2.4 Haptic Rendering of Dynamic Point Clouds

Haptic rendering of recorded of point cloud data was first done in [10] and [11]. In those works, RGB-D data were recorded, after which each frame was tessellated. A god-object haptic rendering method was then used.

The first haptic rendering method for streaming point cloud data was presented in [1], where a constraint-based method for interaction with streaming RGB-D data was presented. This method was then used in rendering of forbidden-region virtual fixtures in [12]. Another method for 3-DOF haptic rendering of streaming point cloud data was presented in [13], where a point cloud was converted to an implicit surface "on-the-fly" (a gradient to an implicit surface was estimated at every time step). Two different methods were presented for doing this. In the first method, the implicit surface was obtained by radially expanding each point and smoothing the resulting volume. In the second method, the surface normal vector for any given point was calculated using total least-squares.

A quite different direction is taken in [14], where 2D images were augmented with haptic features such as texture feedback and geometry feedback (from a depth camera). In that work, the depth data were overlaid directly onto the image without first converting the data to a point cloud.

3 Challenges in Haptic Interaction with Streaming Point Cloud Data

The use of streaming point cloud data in haptic rendering imposes some new challenges as well as opportunities. The main opportunity is that it allows us to interact with dynamic and/or moving objects in real time. The goal is to design a haptic rendering method for streaming point clouds with performance comparable to methods for polygons while meeting the timing constraints for real-time applications.

3.1 Challenge 1: Timing/Latency

The haptic rendering method should not induce excessive latency if it is to be considered real time. It has been shown that 99 percent of the population is unable to perceive a delay between visual and haptic feedback if the delay is less than 54 ms [15]. For our purpose, we denote our haptic rendering method to be real time when it is perceived as real time. The main sources of latency are the processing on the RGB-D camera and the computational effort in the haptic rendering. In this work, the challenge of minimizing the latency involved in the haptic rendering method is addressed.

3.2 Challenge 2: Interaction with Moving Objects

A physical object moving at a constant velocity is represented in a streaming point cloud as points moving with discrete jumps. At higher velocities, this might cause a problem as the jumps become larger. In addition to this, the point cloud is typically subject to measurement noise. The challenge is to construct a proxy method that is robust to both movements and noise in the point cloud.

3.3 Challenge 3: Haptic Resolution

The resolution for point cloud haptic rendering is highly dependent on the density of the captured point cloud as well as the size and geometry of the proxy. The goal is to design a haptic rendering method that allows the user to distinguish the smallest possible features for a given spatial sampling rate. This work focuses on rendering the correct geometry of objects given that the point cloud is dense (no up-sampling is needed). The representation of surface mechanical properties, such as friction, are beyond the scope of this paper.

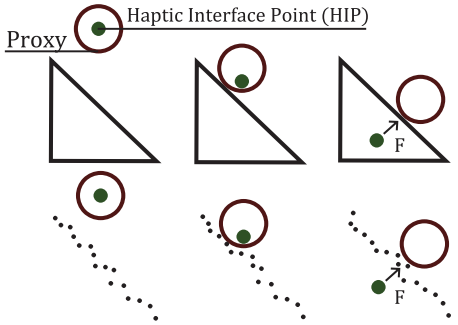

Fig. 1. A two-dimensional illustration of the similarities between traditional haptic rendering methods and our proposed method for haptic rendering of streaming point clouds. Just as the proxy is constrained by the polygon (top), the proxy is constrained by the point cloud (bottom) [1].

4 The Haptic Rendering Method for Streaming Point Clouds

We have developed a method for haptic rendering that addresses the challenges mentioned in the previous section. Our initial approach was presented in [1] where the main idea was an adaptation of the proxy method to point clouds by iteratively moving the proxy toward the HIP while estimating surface normals at every proxy step (see Fig. 1). This was done to prevent the proxy from moving through points and instead remain on an estimated surface. The force sent to the haptic device was calculated as a virtual spring-damper coupling between the HIP and the proxy. An extended and improved version of that haptic rendering method is presented in this section.

The adaptation of prior proxy method to point clouds is done by extending the traditional spherical proxy with three different regions. These regions are used to detect the current *proxy state* while moving in the point cloud.

TABLE 1
Variables and Definitions

f_c	The frame rate of the point cloud source
$j = 0, 1, 2...$	Numbering of point cloud frames
$k = 0, 1, 2...$	Numbering of proxy steps
$l = 0, 1, 2...$	Numbering of force frames
$\mathbf{P}_k$	The proxy position (at step k)
$r_{\{1,2,3,c\}}$	Radii defining proxy regions
$\mathbf{u}_k$	The vector between the proxy and the HIP (in the beginning of step k)
$\mathbf{s}_k$	The change in proxy position (at step k)
d_k	The step size calculated in Section 4.4
ξ	A function determining the step size when moving large steps in contact
γ	A scaling factor for small proxy steps when the proxy is in contact
θ	A constant to determine proxy convergence
$\hat{\mathbf{n}}_k$	The estimated normal vector at $\mathbf{P}_k$
$\psi(r)$	The modified Wendland weighting function
$\mathbf{s}_j$	The correction in proxy position due to point cloud velocity (at frame j)
$\hat{\mathbf{v}}_\mathbf{j}$	The estimated point cloud velocity vector
$v_j, \hat{v}_j$	The true and estimated point cloud velocity respectively
$\mathbf{F}_l$	The force sent to the haptic device (at frame l)
$\mathbf{u}_l$	The most recent $\mathbf{u}_k$ at force frame l

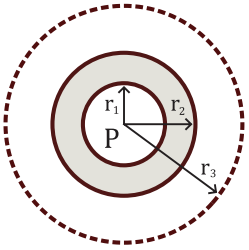

Fig. 2. Two-dimensional cross section of the proxy. Proxy regions are defined by $r_1 < r_2 < r_3$.

The proxy is moved iteratively by a proxy movement algorithm that starts when a new point cloud is captured or when the haptic device moves. The proxy movement algorithm moves the proxy in small steps until it converges (i.e., reaches the desired position in the point cloud). The force sent to the haptic device is calculated as a virtual spring-damper between the HIP position and the proxy position (as given by the proxy movement algorithm).

In prior work, the proxy would "pop through" if the point cloud moved too fast. In this section, we present a method for preventing pop through by estimating and correcting for the point cloud velocity at each point cloud update. This essentially enforces an upper bound on point cloud acceleration rather than velocity. However, the proxy might pop through at lower accelerations due to the effects of noise.

The presented method involves many variables and definitions, and the most important ones are listed in Table 1.

4.1 A Proxy in a Point Cloud Environment

The classic proxy as developed in [4] is defined as a sphere that perfectly serves the purpose in a virtual environment consisting of polygons with well-defined surfaces. For the proxy to move in a noisy point cloud environment, the proxy is defined here to have three radii $r_1 < r_2 < r_3$ extending out from its center (see Fig. 2). The region inside r_1 is used to detect when the proxy is about to pop through a point cloud. The region between r_1 and r_2 is used to detect contact with a point cloud, since the proxy most likely will be in contact with noisy point clouds rather than well-defined surfaces. The contact point is in the middle of this region; that is, $r_c = \frac{r_1+r_2}{2}$. All the points within r_3 are used for the normal estimation (see Fig. 3).

Thus during haptic interaction with a point cloud the proxy will be in one of three possible states [1]:

1. In *free motion* if there are no points within r_2 from the proxy.

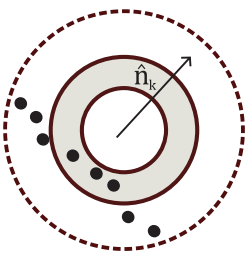

Fig. 3. The normal vector estimate $\hat{\mathbf{n}}_k$.

2. In *contact* if there are points between r_1 and r_2 but no points within r_1 of the proxy.
3. In *entrenchment* if there are points within r_1 of the proxy. This can be thought of as the proxy digging into the point cloud.

4.2 Local Surface Normal Estimation

The proxy should ideally never move through points to achieve correct haptic rendering. This can be enforced by using the points to form a linear constraint on the proxy movement and then moving the proxy iteratively perpendicular to this constraint. Note that when a spherical object (such as a proxy) comes into contact with any continuous surface, the normal vector at any contact point points toward the center of the sphere. That is, the geometry of the proxy can be used to estimate a normal vector to the surface. The alternative to this approach would be to compute a normal vector for each point using total least squares, as done for point cloud surface reconstruction in [16], which would involve an eigenvalue decomposition for each point.

Our approach avoids this repeated eigenvalue decomposition. It also produces the expected result when in contact with a single point. To further improve speed, the constraints from the points inside radius r_3 are weighted to form a single planar constraint to the proxy; this weighting assumes that the points approximately are uniformly sampled within r_3. In many situations, it is sufficient to group the movement constraints into a single constraint as long as the step size is small enough (which it will have to be anyway due to linearization of the constraints). Another implication of this approach is that the point cloud can be touched from any direction.

Every time before the proxy moves, this normal vector estimate $\hat{\mathbf{n}}_k$ is found as follows:

1. Let $\mathbf{x}_i,\ i = 1, 2, \ldots, N$ be the points within radius r_3 of the proxy.
2. Let

$$\hat{\mathbf{n}}'_k = \sum_{i=1}^{N} \frac{\mathbf{P}_k - \mathbf{x}_i}{\|\mathbf{P}_k - \mathbf{x}_i\|_2} \psi(\|\mathbf{P}_k - \mathbf{x}_i\|_2), \tag{1}$$

$\hat{\mathbf{n}}_k$ is then found by normalizing $\hat{\mathbf{n}}'_k$.

In (1), $\psi(r)$ is a modified version of the Wendland function [17] such that

$$\psi(r) = \begin{cases} 1 & \text{if } r \leq r_1 \\ \left(1 - \dfrac{r - r_1}{r_3 - r_1}\right)^4 \left(\dfrac{4(r - r_1)}{r_3 - r_1} + 1\right) & \text{if } r_1 < r < r_3 \\ 0 & \text{if } r \geq r_3. \end{cases} \tag{2}$$

This smooth weighting function is monotonically decreasing between $r = r_1$ and $r = r_3$. As a result, new points are introduced gradually (starting with low weight) as the proxy moves in the point cloud. The Wendland function was suggested for point cloud haptic rendering in [13].

The transformation of points from an RGB-D frame to a Cartesian coordinate is done in parallel utilizing general purpose GPU computing. Consequently, timing issues of concern in [1] are resolved. This method decreases CPU load and does not increase GPU load because all points would have to be transformed for visualization anyway.

For a large point cloud, it can be computationally expensive to find the points that are in the neighborhood of the proxy. Luckily, most point clouds derived from RGB-D cameras are sorted horizontally and vertically in an array according to some coordinate frame. By transforming the proxy position to this frame, the neighboring points can be found. Details regarding finding points in the neighborhood of the proxy can be found in [1].

If the number of points in the neighborhood around the proxy is relatively small, a brute-force search of this neighborhood is sufficient. That is what we have done in this paper. For very dense point clouds, the points in a (larger) neighborhood can be ordered in a tree-structure at every point cloud update. Unfortunately, this will bound the distance that the proxy can move until the next point cloud update, as collision outside the scope of this tree structure will be missed.

4.3 Proxy Movement Algorithm

This algorithm is based on the normal vector estimate and the state of the proxy moving the proxy to its new position in small steps of varying size (this step size is described in Section 4.4). When the point cloud updates (and potentially moves), the algorithm corrects for the point cloud velocity by moving the proxy along an estimated velocity vector (this velocity is estimated in Section 4.5). The algorithm starts when either the haptic device moves or when the point cloud updates, and it iterates until the proxy has converged:

1. If the point cloud was updated since the last iteration, correct for this by moving the proxy one step

$$\mathbf{s}_j = \frac{1}{f_c} \hat{v}_j \hat{\mathbf{n}}_{k-1}, \tag{3}$$

where $\hat{v}_j$ is the point cloud velocity to be estimated in Section 4.5. This velocity is assumed to be in the direction of the previous normal vector.
2. Determine the proxy state.
3. If the proxy is in free motion (as described in Section 4.1), move the proxy one step

$$\mathbf{s}_k = d_k \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|_2} \tag{4}$$

along the vector between the proxy and the HIP (d_k is the step size determined in Section 4.4).
4. If the proxy is entrenched, move the proxy one step

$$\mathbf{s}_k = d_k \hat{\mathbf{n}}_k \tag{5}$$

in the direction of the normal vector estimate.
5. If the proxy is in contact and the HIP is inside the estimated surface (see Fig. 4), project $\mathbf{u}_k$ on the plane defined by $\hat{\mathbf{n}}$ (denote this vector $\mathbf{u}_{k,p}$). Then, move the proxy one step

$$\mathbf{s}_k = d_k \frac{\mathbf{u}_{k,p}}{\|\mathbf{u}_{k,p}\|_2} \tag{6}$$

in the direction of $\mathbf{u}_{k,p}$.

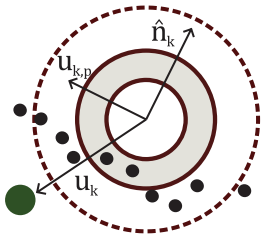

Fig. 4. The proxy is in contact and the HIP is inside the estimated surface. In this case, the proxy will move in the direction of $\mathbf{u}_{k,p}$.

6. If the proxy is in contact and the HIP is outside the estimated surface, move the proxy one step

$$\mathbf{s}_k = d_k \hat{\mathbf{u}}_k, \tag{7}$$

which moves the proxy toward the HIP and eventually out of contact (see Fig. 5).

7. Iterate steps 1 to 7 until the proxy has converged. If the proxy is in contact, it has converged if

$$\cos^{-1}\left(\frac{-\mathbf{u}_k^T \hat{\mathbf{n}}_k}{\|\mathbf{u}_k\|_2}\right) < \theta, \tag{8}$$

where θ is a constant representing a small angle between $-\mathbf{u}_k$ and $\hat{\mathbf{n}}_k$. If the proxy is in free motion, it has converged if the HIP position equals the proxy position.

There will most likely be several iterations of this algorithm for every haptic frame (indexed with $l = 0, 1, 2 \ldots$) and even more iterations for every point cloud update (indexed with $j = 0, 1, 2 \ldots$). One iteration of the proxy movement algorithm is referred to as a *proxy step*, and $\mathbf{P}_k$ should therefore be interpreted as the proxy position at step $k = 0, 1, 2 \ldots$

4.4 Variable Step Size for Improved Performance

In [1], the proxy was moved with fixed size steps. To achieve good haptic resolution, the step size had to be small. Because of this, many iterations were required for the proxy to move longer distances even if no points were constraining the proxy's path. It also caused the proxy to jitter up and down because it almost never made perfect contact. In this paper, we address these problems by considering a variable step size d_k (associated with proxy step k) determined by the points in the neighborhood of the proxy.

4.4.1 Step Size in Free Motion

The desired step size for a proxy in free motion is the largest distance the proxy can move in a straight line toward the HIP without passing through any points, see Fig. 6. This step size is found by considering each point as an individual constraint on the proxy movement. For each point, the distance that the proxy could move toward the HIP without passing through the point is calculated. The desired step size is chosen as the smallest of those distances (which takes all constraints into account).

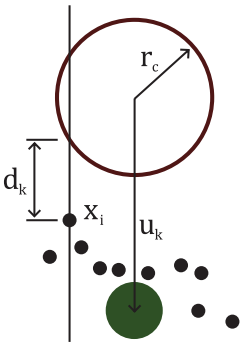

Fig. 6. d_k is the largest distance the proxy can move (in the direction of $\mathbf{u}_k$) without passing through any points.

Let $\mathbf{x}_i$, $i = 1, 2 \ldots M$ be the set of M points that could potentially constrain the proxy movement. The step size associated with the ith point (d_i) is then found by solving

$$r_c - \left\| \mathbf{x}_i - \mathbf{P}_k - d_i \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|_2} \right\|_2 = 0 \tag{9}$$

for the solution with the smallest magnitude (this has to be done M times). Step size is then chosen as $d_k = \min_i d_i$. If d_k is larger than the distance, then the proxy would have to move to perfectly match the position of the HIP, simply set $d_k = \|\mathbf{u}_k\|_2$.

4.4.2 Step Size in Entrenchment

For an entrenched proxy, the desired step size is the smallest distance the proxy has to move to get all points out of entrenchment. Let $\mathbf{x}_i$, $i = 1, 2 \ldots M$ be the entrenched points (all points within region 1 of the proxy). The step size associated with the ith point (d_i) can be found as the positive solution of

$$r_c - \|\mathbf{x}_i - \mathbf{P}_k - d_i \hat{\mathbf{n}}_k\|_2 = 0. \tag{10}$$

Then, choose $d_k = \max_i d_i$ (see Fig. 7).

4.4.3 Step Size in Contact

When the proxy is in contact the goal becomes to move the proxy toward the ideal position on the surface. This

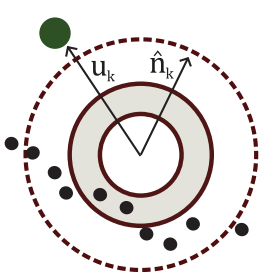

Fig. 5. The proxy is in contact. Since the HIP is moving out of the surface, the proxy will move in the direction of $\mathbf{u}_k$.

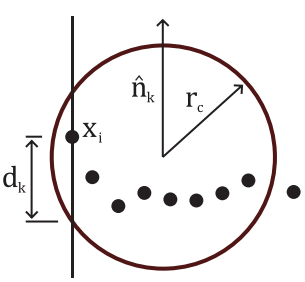

Fig. 7. d_k is the shortest distance the proxy has to move (in the direction of $\hat{\mathbf{n}}_k$) to get all the points out of entrenchment.

ideal position can be thought of as the position locally minimizing the distance between the proxy and HIP, or finding a position such that the angle between $-\mathbf{u}_k$ and $\hat{\mathbf{n}}_k$ is 0 (in practice this angle is set to a small constant θ). Consider thinking of $\mathbf{u}_{k,p}$ as a gradient estimate to the distance between the proxy and the HIP (which it is for some coordinate frame). The movements toward the ideal position could then be found using numerical optimization by approximating a Hessian at every proxy step. However, this is not a good idea for haptic rendering with a streaming point cloud because the point cloud does not represent an underlying smooth function.

Instead, when a proxy is in contact with the surface and closer than r_1 to the ideal position (as determined by the projection $\mathbf{u}_{k,p}$), the step size d_k should equal the magnitude of $\mathbf{u}_{k,p}$ scaled by a factor $\gamma \leq 1$ to suppress errors in the normal vector estimate. That is, when $\|\mathbf{u}_{k,p}\|_2 < r_1$ choose

$$d_k = \gamma \|\mathbf{u}_{k,p}\|_2, \tag{11}$$

where γ can be chosen as a function of the number of proxy steps taken while closer than r_1 to the ideal position. This way the proxy steps become smaller as the number of iterations increase.

When the proxy is in contact and farther away than r_1 from the ideal position on the point cloud, the step size is chosen as

$$d_k = \xi r_1, \tag{12}$$

where $\xi < 1$ is a constant. This step size is chosen after observing that a proxy in contact might become entrenched after a movement of size $<r_1$ but realizing that this will be corrected for in the next proxy step. *The important thing is that the proxy never will be able to pass through a point cloud in a single step.*

4.5 Point Cloud Velocity Estimation

A physical object moving at a constant velocity v is represented in a point cloud as a set of points moving in discrete steps of size v/f_c. Upon haptic interaction with a point cloud moving toward the proxy at a low velocity, the proxy becomes entrenched at every point cloud update. This is then corrected for in the next proxy step when the proxy moves in the direction of the estimated normal vector. In this setting, point cloud movements larger than r_1 per update will cause the subsequent normal estimate (if any) to point in wrong (opposite) direction. As a result, the proxy will move out of entrenchment in the wrong direction. This is referred to as a *pop through*.

One could argue that a solution to this would be to keep track of the points where the proxy is in contact and then constrain the proxy to always be on the correct side of these points. However, this is not feasible in a point cloud without any information regarding which physical object or surface a point belongs to.

Instead, consider correcting for the point cloud velocity at every point cloud update by moving the proxy along a local velocity vector estimate (as done in step 1 of Section 4.3). Even though it is called a velocity vector, only the velocity component toward the proxy is estimated, because this is the only component that could potentially cause a pop through. That is, it is perfectly fine for a point cloud to "slide" sideways under the proxy, as the velocity component toward the proxy is zero in that case. If the proxy becomes entrenched at the point cloud update (but after the velocity correction), it will be interpreted as a positive point cloud acceleration (in a finite difference sense). Conversely, if the proxy is in free motion after the point cloud update, it will be interpreted as a negative point cloud acceleration. In either case, the velocity estimate is corrected using the length of proxy step taken right after the point cloud update ($\mathbf{s}_k$), projected onto the normal vector that was estimated just before point cloud update ($\hat{\mathbf{n}}_{k-1}$), scaled by the point cloud update rate (f_c). These quantities are illustrated in Fig. 8. Given this local velocity, estimated at point cloud update $j-1$ (assume $v_0 = 0$), the velocity estimate at point cloud update j becomes

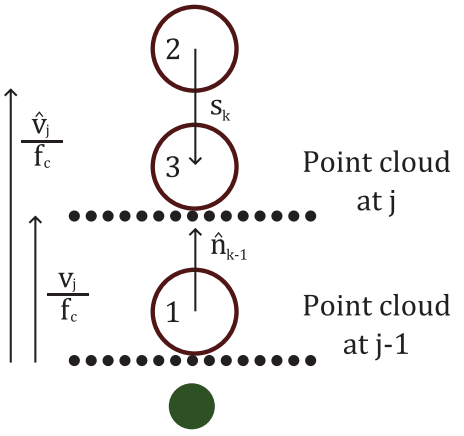

Fig. 8. 1) The proxy in contact at proxy step $k-1$. 2) Before proxy step k, the point cloud updates and the velocity vector $\mathbf{vj}/fc$ is added to the proxy position. This causes the proxy to move too far because the magnitude of the true velocity was less than the estimated velocity. 3) One iteration of the proxy movement algorithm brings the proxy down to the surface. The velocity estimate is then updated using the projection of s_k onto $\hat{\mathbf{n}}_{k-1}$ as given by (13).

$$\hat{v}_j = \hat{v}_{j-1} + \hat{\mathbf{n}}_{k-1}^T \mathbf{s}_k f_c. \tag{13}$$

If the estimated velocity becomes negative, set $\hat{v}_j = 0$ because there is no risk for pop through when a point cloud is moving away from the proxy.

For haptic interaction with a dense and perfectly flat point cloud surface moving toward the proxy, the upper bound on point cloud acceleration becomes

$$a < r_1 f_c^2. \tag{14}$$

This way of estimating the point cloud velocity assumes small deviations in the point cloud trajectory between each frame. This assumption is valid for point clouds sampled at a sufficient rate.

4.6 Force Rendering

The force on the haptic device is calculated once every $1\ \mathrm{ms}$ as a virtual spring-damper coupling between the HIP and the contact region of the proxy. Let $\mathbf{u}_l$ be the vector between the most recent proxy and the HIP (the most recent $\mathbf{u}_k$ at the time of force rendering). If this vector would be used in the force rendering, a stuttering force would be sent to the haptic device upon interacting with

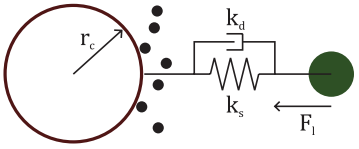

Fig. 9. Illustration of the force acting on the HIP that is being sent to the haptic device. The force is calculated as if there was a virtual spring-damper coupling between the HIP and the proxy.

point clouds moving with higher velocities. This is because the proxy moves along the estimated velocity vector at every point cloud update (f_c times per second). By adjusting $\mathbf{u}_l$ for the estimated velocity, the estimated vector between the proxy and the HIP can be written as

$$\hat{\mathbf{u}}_l = \mathbf{u}_l + (t - t_j)\hat{\mathbf{v}}_j, \tag{15}$$

where t_j is the time at which the most recent point cloud was updated. The force on the haptic device can then be calculated as

$$\mathbf{F}_\mathrm{l} = -\frac{\hat{\mathbf{u}}_l}{\|\hat{\mathbf{u}}_l\|_2}(k_s(\|\hat{\mathbf{u}}_l\|_2 - r_c) - k_d f_h(\|\hat{\mathbf{u}}_l\|_2 - \|\hat{\mathbf{u}}_{l-1}\|_2)), \tag{16}$$

where f_h is the rate of which the force is rendered and the following term should be interpreted as the one-sided finite difference of the distance between the proxy and the HIP (a velocity). This force is sent to the haptic device *only* when the HIP is outside the radius r_c of the proxy and only when the proxy is in contact (see Fig. 9). As a result, the user will feel a force only after the HIP has passed through the points rather than directly when the proxy becomes constrained by the points. In practice, the computation of 16 is done in a separate thread running at a fixed 1,000-Hz rate. This has to be done since convergence cannot be guaranteed in an arbitrary point cloud environment.

5 Results

This haptic rendering method successfully renders haptic forces upon haptic interaction with streaming point clouds. The efficient structure allows the haptic rendering to be executed on a common desktop computer. The point cloud velocity estimation presented in Section 4.5 significantly reduces the risk for pop through at point cloud updates. The use of the velocity estimate for force rendering in Section 4.6 smoothens the force sent to the haptic device. The method successfully renders haptic forces for interaction with sharp corners. Haptic interaction with concave shapes can be accurately rendered if the curvature is less than the curvature of the proxy.

The performance of this haptic rendering method is evaluated in a virtual 3d environment simulated on a computer (Intel i7-2675QM CPU with AMD Radeon HD 6750M GPU) running Windows 7. The HIP in this environment was controlled by either a SensAble PHANToM Omni haptic device or by automating the HIP movement (for evaluation). The point cloud can either be imported from a RGB-D camera or generated as samples of an implicit function.

The correctness of our haptic rendering method is evaluated by constructing a synthetic point cloud with grid spacing $s_{xz} = 0.3r_c = 0.0015$ m (which is roughly what we would expect at 1-m distance from an RGB-D camera), which is updated at $f_c = 30$ Hz. These values were chosen to match typical properties of point clouds derived from commercially available RGB-D cameras. Since these synthetic data sets are not derived from a depth camera, a brute-force search was used to find a normal vector, proxy state, and step size. The HIP is automated in this point cloud to simulate haptic interaction for purposes of observing the proxy movement under different point cloud conditions. These evaluations focus on the proxy movement rather than the force sent to the haptic device with the understanding that correct proxy movements result in a correct force on the haptic device. Finally, the feasibility of haptic rendering with an RGB-D camera is shown using point clouds from an Xbox Kinect.

TABLE 2
Parameters Used in Experimental Evaluation

r_c	$5 \cdot 10^{-3}m$
Proxy radius r_1	$r_c - 0.01 \cdot 10^{-3}$
Proxy radius r_2	$r_c + 0.01 \cdot 10^{-3}$
Proxy radius r_3	$2r_c$
Convergence constant θ	$0.1\pi/180$
Step size scaling factor γ	$3/10$
Step size scaling factor ξ	$1/10$
Spring constant k_s	$600N/m$
Damping constant k_d	$0Ns/m$
Point grid spacing s_{xz}	$0.3r_c$
Point cloud update rate f_c	$30Hz$

The parameters used in the evaluation are shown in Table 2. The results in this section are dependent on these parameters and, in particular, dependent on the three radii associated with the proxy. The results are, therefore, normalized using $r_c = 0.005$ m. That is, the results for the synthetic data set hold for varying r_c as long as the radii $r_{1,2,3}$ as well as the grid spacing are scaled accordingly. In experiments, it was found that no damping was needed in the virtual coupling, because the damping in the haptic device was sufficient. In this test implementation, the proxy movement algorithm does not stop upon convergence, this since it is not feasible to start and stop threads at the required rate on a non real-time operating system.

5.1 Flat Point Cloud Surface

The HIP is first automated on a 100×100 point cloud (see Fig. 10) such that it moves with a constant velocity of $2r_c$ per second ($v = 0.01$ m/s) in positive tangential direction to the plane. The proxy is first moved 1 s above the points, followed by 3 s below the points and then again 1 s above the points. The y-coordinates of the proxy and the HIP are shown in the top trace of in Fig. 11. The bottom trace shows a closer look at the proxy trajectory; with this zoom level a periodic pattern can be observed. This is expected since the proxy movement algorithm moves the proxy with respect to the points and not to an assumed underlying surface. This can be thought of as the proxy "squeezing" in between the points. The amplitude of this periodic pattern is $0.01r_c = 5 \cdot 10^{-5}$ m. The step-like trajectory arises from the fact that the contact region allows the proxy to become slightly entrenched (or in free motion) before correcting and moving up (or down). Upon haptic interaction using a haptic device, the surface feels like a frictionless plane.

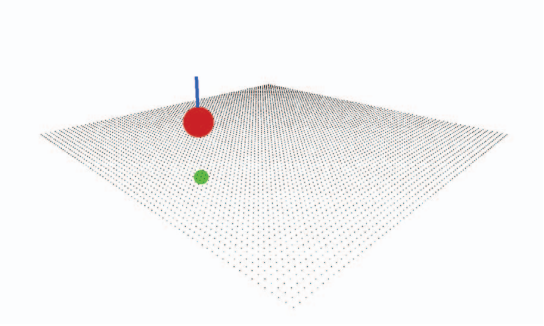

Fig. 10. Haptic interaction with a 100 × 100 point cloud grid with equal spacing. The larger sphere representing the proxy and the smaller sphere below the point cloud representing the position of the HIP. The pin sticking out of the proxy illustrates the estimated normal vector.

When the HIP makes a discrete jump at 1,000 ms, it takes the proxy less than 1 ms to move down to the point set surface. Hence, a correct force can be sent to the haptic device in the next haptic cycle (after 1 ms). This means that this method of haptic rendering meets the timing goal of *Challenge 1* in Section 3.1.

5.2 Moving Point Cloud Surface

The performance of the proxy in contact with moving point clouds is evaluated by constructing a 200 × 200 flat point cloud with varying acceleration and noise. Initially, the proxy and the HIP are placed at the origin and the point cloud is centered $10r_c = 0.05$ m below the origin. The point cloud is then lifted (in the direction of the normal vector) with constant velocity $20r_c$ per second ($v_0 = 0.1$ m/s) until it reaches the origin, after which a constant acceleration a is applied for 10 frames (1/3 second) (see Fig. 12). To simulate measurement noise from an RGB-D camera, zero-mean Gaussian noise (standard deviation σ) is applied to each Cartesian coordinate of each point. During the acceleration, the HIP is kept $6r_c = 0.03$ m below the point cloud surface. If the proxy is located below the point cloud after the lift is finished, we denote it a pop through. This procedure was repeated 5,396 times for different values of point cloud

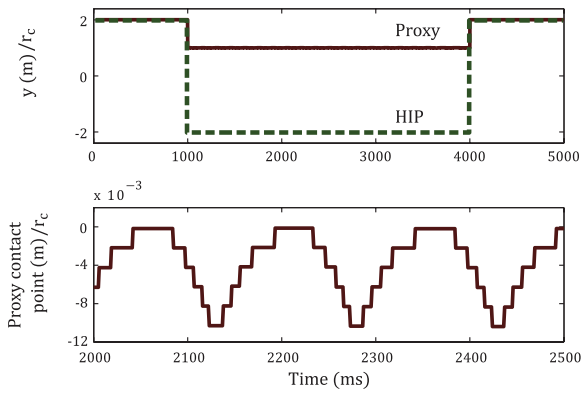

Fig. 11. To validate that the presented method results in a correct haptic proxy, the HIP is automated on an 100 × 100 point cloud grid. The top trace shows the y-coordinates of the proxy and the HIP as the HIP are moved at a constant velocity of $2r_c$ per second ($v = 0.01$ m/s) in positive tangential direction to the plane. After 1,000 ms, the HIP moves below the surface and then after 4,000 ms, it moves back up again. The bottom trace shows the y-coordinate of the proxy contact point which shows a periodic pattern with amplitude $0.01r_c = 5 \cdot 10^{-5}$m.

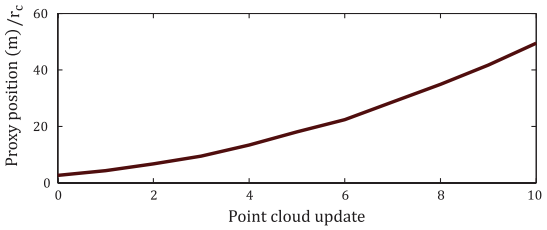

Fig. 12. The proxy y-coordinate for each point cloud update when the point cloud is lifted with constant acceleration of $600r_c$ ($a = 3$ m/s^2) and under the influence of noise with $\sigma = 0.6r_c = 0.003$.

acceleration and point cloud noise. The result is shown in Fig. 13, where a filled square indicates that a pop through *did not* occur. These results are a major improvement compared to [1] where velocities above $16r_c$ per second ($v = 0.08$ m/s) resulted in pop through. The goals regarding moving point clouds in *Challenge 2* in Section 3.2 are, therefore, met.

5.3 Point Cloud Representing Sharp Corners

The accuracy of this haptic rendering method is evaluated by moving the HIP in a 300 × 100 grid with the shape of a triangle wave with increasing amplitude and observing how well the proxy can track this surface. Fig. 14 shows how the proxy tracks the surface over a few convex and concave corners. As can be seen, the convex part is rendered very well as a sharp edge. For the concave part of the wave, the rendering is ultimately dependent on the size of the proxy. When moving into a narrow concave shape, the proxy becomes constrained by the points on the sides, and therefore, the proxy cannot move to the bottom of the triangle wave. The goals of *Challenge 3* in Section 3.3 are met because narrow features and sharp corners can be distinguished.

5.4 Streaming Point Cloud from an RGB-D Camera

Haptic interaction with a streaming point cloud is evaluated using an Xbox Kinect RGB-D camera capturing point clouds

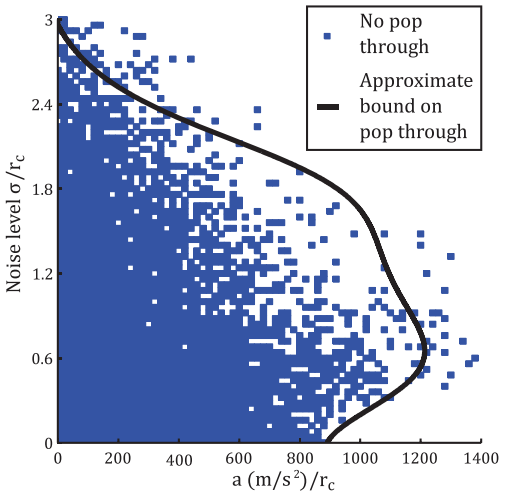

Fig. 13. The result of 5,396 point cloud lifts for varying values of noise level σ and acceleration a. The filled squares indicates a successful lift, the white portion of the plot illustrates the values for which the proxy popped through the point cloud. The solid line illustrates the mathematically derived approximate bound on pop through obtained in Section 6.2. The axes are scaled by r_c.

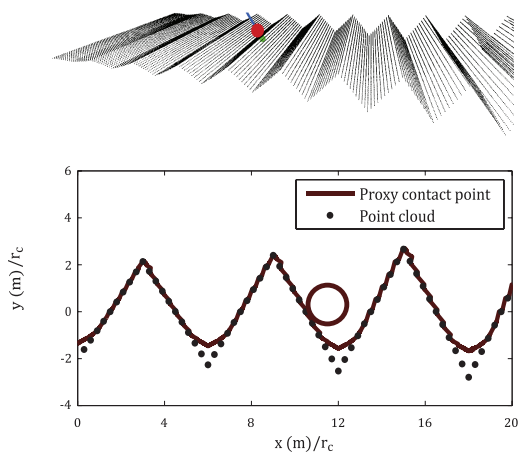

Fig. 14. A 300 × 100 point set surface shaped as a triangle wave with increasing amplitude (top). A two dimensional cross section of the point cloud (ldots) and the contact point of the proxy (line) for a small portion of this triangle wave (bottom). The line ultimately illustrates the surface the user will feel when interacting with this point cloud. The circle has radius r_c.

at $f_c = 30$ Hz. Fig. 15 shows a point cloud representation of a physical object on a table, where the RGB data are superimposed on the depth data. For purposes of evaluating the accuracy of haptic interaction with a noisy point cloud, the HIP was moved at constant velocity in positive z-direction. The top trace of Fig. 16 shows the proxy trajectory in the y-z plane as the proxy moves with respect to the point cloud representation of the physical object. The bottom trace shows the magnitude of the force sent to the haptic device for each HIP position.

Upon haptic interaction using a SensAble PHANToM Omni haptic device, the force sent to the haptic device remains stable and without oscillations despite of the evident measurement noise in the point cloud data. The use of the velocity estimate in Section 4.6 results in smooth haptic interaction with moving point clouds even when the proxy moves in large discrete steps at every point cloud update. The method successfully renders forces at different locations in the scene (with different point cloud resolutions) as long as the point cloud is dense enough for the proxy not to fall in between points. The proxy typically converges in 1-2 proxy steps each time the haptic device moves. The proxy movement algorithm typically runs at a maximum rate of 30 kHz (this includes finding proxy state, normal vector as well as step size).

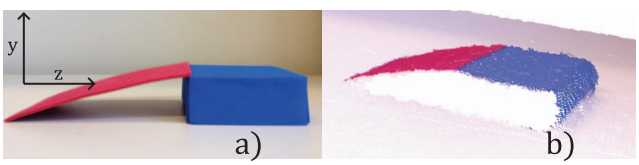

Fig. 15. A physical object consisting of a ramp and a box is captured by the Xbox Kinect camera for purposes of evaluating haptic interaction with streaming point clouds. a) A photo of the physical objects. The camera was located outside the right upper corner of this photo. b) The point cloud representation of the same scene as captured by the Kinect camera.

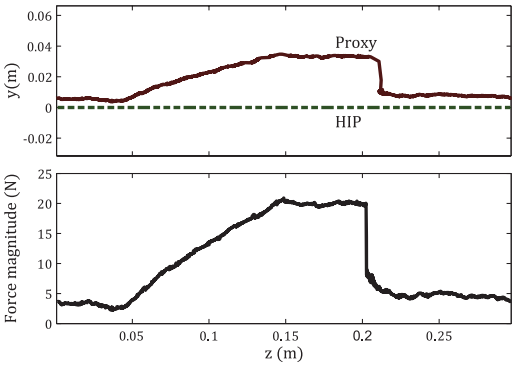

Fig. 16. The top trace shows the z-coordinates plotted versus the y-coordinates for the proxy and the HIP when moving in the point cloud representation of the physical object. The bottom trace shows the magnitude of the force sent to the haptic device.

The speed of the algorithm scales approximately linearly with the number of points in the neighborhood of the proxy as collision detection, normal vector calculation, and the search for step size are the computationally most intensive parts of the algorithm. In actual implementation, these three are performed in the same loop, where each individual task involves a brute-force search of the neighborhood around the proxy. This will probably be the main limitation of a 6-DOF version as the virtual tool typically will be larger than the small spherical proxy in this work. If the virtual tool is larger than the spherical proxy used here, the computational load of this 6-DOF haptic rendering algorithm will increase. The force is rendered independently of the proxy algorithm as the rate is fixed at 1,000 Hz.

6 Analysis

6.1 A Bound on Pop through

In Section 5.2, it was shown that our haptic rendering method significantly reduces the risk for pop through upon haptic interaction with noisy point clouds moving at high accelerations. Under the influence of noise, the condition for pop through becomes a bit more complex than described by (14). Noise affects the pop through problem mainly in two ways. First because it results in an erroneous normal vector estimate. An error that propagates to the proxy movement algorithm, where it results in an incorrect point cloud velocity correction. Second, it might cause the point cloud to become thicker, which reduces the risk for pop through. In fact, for a given estimated velocity vector $\hat{\mathbf{v}}$ and a point cloud thickness at $j-1$ (denoted h_{j-1}), correct haptic interaction with respect to pop through can be expected if

$$\frac{\hat{\mathbf{v}}_j^T \mathbf{v}_j}{f_c \|\mathbf{v}_j\|_2} + r_1 + h_{j-1} > \frac{v_j}{f_c}. \tag{17}$$

This inequality is illustrated in Fig. 17.

For a point cloud moving in the direction of the true normal vector, (17) can be rewritten in terms of the true and estimated normal vector as well as the point cloud acceleration a_j (since $v_j = v_{j-1} + \frac{a_j}{f_c}$), which yields

$$r_1 + h_{j-1} > \frac{a_j}{f_c^2} + \frac{1}{f_c}\left(v_{j-1} - \hat{v}_j \mathbf{n}_{k-1}^T \hat{\mathbf{n}}_{k-1}\right). \tag{18}$$

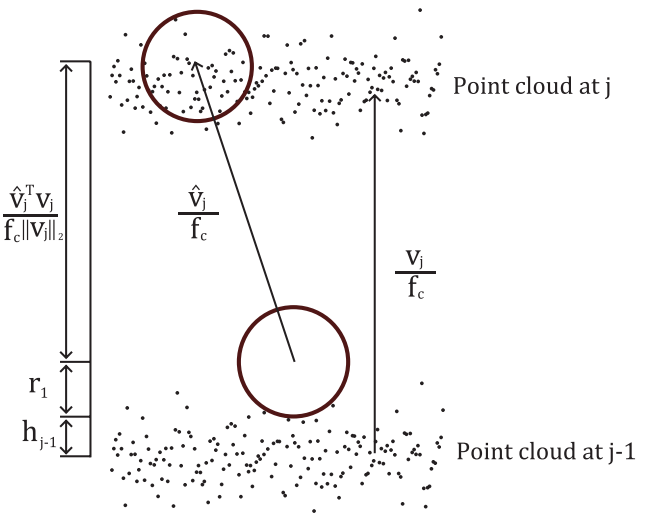

Fig. 17. The proxy will pop through the point cloud if the sum of the true velocity is greater than the sum of the estimated velocity projected onto the true velocity vector, the inner radius r_1 of the proxy and the thickness of the point cloud (denoted h_{j-1}).

6.2 Validation of the Pop through Bound

To validate (17), we compare it with the experimental results from Section 5.2 after making some simplifying assumptions. First, note that (18) gives a bound on pop through for a given point cloud update j rather than a sequence of point cloud updates as in Section 5.2 (acceleration during 10 frames). By observing (18), one can argue that the risk for pop through increases with velocity as the error in the normal vector estimate becomes dominant. Therefore, only the risk for pop through at the point cloud update with the highest velocity (the 10th frame) will be considered in this analysis.

Second, the quantities of h_{j-1}, $\hat{\mathbf{n}}_{k-1}$, and $\hat{v}_j$ are unknown and values of these have to be estimated or chosen.

To get some intuition on how to choose them, consider the following sequence of events leading up to a pop through:

1. The normal vector estimate calculated just before point cloud update $j-1$ is close to the actual normal vector and the proxy does not pop through at this point cloud update.
2. The resulting velocity estimate is very good, that is
$$\hat{v}_j \approx v_{j-1}. \tag{19}$$
3. Due to noise, the normal vector estimate right before point cloud update j is poor and points in an erroneous direction.
4. The resulting proxy step at point cloud update j ($\mathbf{s}_j$) will, therefore, be in the wrong direction. As a result, the proxy will not be moved to the right position and a pop through will most likely occur.

With this in mind, $\hat{v}$ at the 10th frame should for a given constant acceleration a be chosen as

$$\hat{v} = \max_j v_{j-1} = v_9 = \frac{1}{10} + \frac{9a}{30}, \tag{20}$$

since $v_0 = 0.1$ m/s was used in Section 5.2.

By rewriting (18) using (19), we get (by dropping the subscripts j and k because only pop through at the 10th point cloud update is considered)

$$r_1 + h > \frac{a}{f_c^2} + \frac{\hat{v}}{f_c}\left(1 - \mathbf{n}^T\hat{\mathbf{n}}\right). \tag{21}$$

In (21), $\mathbf{n}^T\hat{\mathbf{n}}$ should be interpreted as the length of the estimated normal vector projected onto the true normal vector, which makes it convenient to define

$$e_{\hat{n}} = 1 - \mathbf{n}^T\hat{\mathbf{n}}, \tag{22}$$

where h and $e_{\hat{n}}$ can be thought of as functions of the random variables associated with the noise on each point. The expected values of h and $e_{\hat{n}}$ can be obtained as functions of σ by simulating point cloud lifts with zero acceleration and varying noise levels between $\sigma = 0$ and $\sigma = 2r_c = 0.01$ (since pop through never occurred in this interval in the experiment). By simulating point cloud lifts in this way and applying least-square fits to the expected values, the estimates are found to be

$$\begin{aligned}\hat{h}(\sigma) = {} & (5.5 \cdot 10^4)\sigma^4 - (6.7 \cdot 10^3)\sigma^3 \\ & + 63\sigma^2 + \sigma - 4.1 \cdot 10^{-5},\end{aligned} \tag{23}$$

and

$$\begin{aligned}\hat{e}_{\hat{n}}(\sigma) = {} & (2.9 \cdot 10^{10})\sigma^5 - (6.4 \cdot 10^8)\sigma^4 + (4.8 \cdot 10^6)\sigma^3 \\ & - (1.2 \cdot 10^4)\sigma^2 + 16\sigma.\end{aligned} \tag{24}$$

Using these estimates, we can approximately say that a pop through in Section 5.2 should not have occurred if

$$r_1 + \hat{h}(\sigma) > \frac{a}{f_c^2} + \frac{\hat{v}}{f_c}\hat{e}_{\hat{n}}(\sigma). \tag{25}$$

We can evaluate the accuracy of this analysis and of (17) in Fig. 13 by comparing the approximate pop through bound on the results from the experimental results. The approximate bound is a bit loose partly because of noise as well as errors in the least square estimates $\hat{h}(\sigma)$ and $\hat{e}_{\hat{n}}(\sigma)$, but mainly because our simplified model only takes into account pop throughs that occur at the 10th frame (and not in frames leading up to the 10th frame). Note that the bound above $\sigma = 2r_c = 0.01$ is a bit more conservative because $\hat{h}(\sigma)$ and $\hat{e}_{\hat{n}}(\sigma)$ were obtained using measurements only up to $\sigma = 0.01$. The risk for pop through increases for increasing velocities as $\hat{e}_{\hat{n}}(\sigma)$ gets magnified. The risk for pop through decreases as f_c increases because the discontinuities between each frame becomes smaller. An exact bound on pop through could theoretically be obtained using (18) if the quantities of h_{j-1}, $\hat{\mathbf{n}}_{k-1}$, and $\hat{v}$ were known exactly for each point cloud update.

7 Conclusion

We have presented a haptic rendering method for streaming point cloud data by extending a traditional *virtual coupling*-based proxy method. The proxy moves iteratively with respect to the points without any need to assume a sufficiently sampled point cloud or preprocessing of points. The use of a variable proxy step size results in better

accuracy for short proxy movements and faster convergence for longer movements. The method presented provides highly accurate haptic interaction for geometries in which the proxy can fit. This means that the challenges of timing as well as accuracy are resolved. This novel point cloud velocity estimation significantly reduces the risk for pop through upon haptic interaction with point clouds moving under the influence of noise. Thus, this method resolves the challenge of haptic interaction with point clouds moving at high velocities. The haptic rendering method presented is very computationally efficient and can run on a Intel Core i7 or equivalent personal computer.

ACKNOWLEDGMENTS

The authors thank Professor Blake Hannaford and Sina Nia Kosari for their assistance. This work was funded by the National Science Foundation (grant no. 0930930) and SERDP 13 MRSEED01-006/MR-2323.

REFERENCES

[1] F. Rydén, S. Nia Kosari, and H. Chizeck, "Proxy Method for Fast Haptic Rendering from Time Varying Point Clouds," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS '12)*, pp. 2614-2619, 2011.

[2] J. Colgate, M. Stanley, and J. Brown, "Issues in the Haptic Display of Tool Use," *Proc. IEEE/RSJ Int'l Conf. Human Robot Interaction and Cooperative Robots*, vol. 3, pp. 140-145, 1995.

[3] C. Zilles and J. Salisbury, "A Constraint-Based God-Object Method for Haptic Display," *Proc. IEEE/RSJ Int'l Conf. Human Robot Interaction and Cooperative Robots*, vol. 3, pp. 146-151, 1995.

[4] D. Ruspini, K. Kolarov, and O. Khatib, "The Haptic Display of Complex Graphical Environments," *Proc. 24th Ann. Conf. Computer Graphics and Interactive Techniques*, pp. 345-352, 1997.

[5] L. Linsen, "Point Cloud Representation," technical report, Faculty of Informatics, Univ. of Karlsruhe, Germany, 2001.

[6] W. McNeely, K. Puterbaugh, and J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *Proc. 26th Ann. Conf. Computer Graphics and Interactive Techniques*, pp. 401-408, 1999.

[7] J. Lee and Y. Kim, "Haptic Rendering of Point Set Surfaces," *Proc. Second Joint EuroHaptics Conf. and Symp. Haptic Interfaces Virtual Environment and Teleoperator Systems*, pp. 513-518, 2007.

[8] D. Levin, "Mesh-Independent Surface Interpolation," *Geometric Modeling for Scientific Visualization*, vol. 3, pp. 37-49, 2003.

[9] N. El-Far, N. Georganas, and A. El Saddik, "An Algorithm for Haptically Rendering Objects Described by Point Clouds," *Proc. Canadian Conf. Electrical and Computer Engineering (CCECE '08)*, pp. 001443-001448, May 2008.

[10] J. Cha, S. Kim, I. Oakley, J. Ryu, and K. Lee, "Haptic Interaction with Depth Video Media," *Proc. Sixth Pacific-Rim Conf. Advances in Multimedia Information Processing (PCM '05)*, pp. 420-430, 2005.

[11] J. Cha, M. Eid, and A.E. Saddik, "DIBHR: Depth Image-Based Haptic Rendering," *Proc. Sixth Int'l Conf. Haptics: Perception, Devices and Scenarios*, pp. 640-650, 2008.

[12] F. Ryden and H. Chizeck, "Forbidden-Region Virtual Fixtures from Streaming Point Clouds: Remotely Touching and Protecting a Beating Heart," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS '12)*, pp. 3308-3313, 2012.

[13] A. Leeper, S. Chan, and K. Salisbury, "Point Clouds Can be Represented as Implicit Surfaces for Constraint-Based Haptic Rendering," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '12)*, pp. 5000-5005, 2012.

[14] S. Rasool and A. Sourin, "Haptic Interaction with 2D Images," *Proc. 10th Int'l Conf. Virtual Reality Continuum and Its Applications Industry*, pp. 13-22, 2011.

[15] L. Batteau, A. Liu, J. Maintz, Y. Bhasin, and M. Bowyer, "A Study on the Perception of Haptics in Surgical Simulation," *Proc. Int'l Symp. Medical Simulation*, pp. 185-192, 2004.

[16] N. Mitra and A. Nguyen, "Estimating Surface Normals in Noisy Point Cloud Data," *Proc. 19th Ann. Symp. Computational Geometry*, pp. 322-328, 2003.

[17] H. Wendland, "Piecewise Polynomial, Positive Definite and Compactly Supported Radial Functions of Minimal Degree," *Advances in computational Math.*, vol. 4, no. 1, pp. 389-396, 1995.

Fredrik Rydén received the BS degree in 2010 in automation and mechatronics from the Chalmers University of Technology and the MS degree in 2012 in electrical engineering from the University of Washington. He is currently working toward the PhD degree in the BioRobotics Lab, Department of Electrical Engineering, University of Washington. His research interests include haptics, surgical robotics, marine robotics, and teleoperation. He is a student member of the IEEE.

Howard Jay Chizeck received the ScD degree in electrical engineering and computer science from MIT in 1982. From 1981 to 1998, he was a faculty member at Case Western Reserve University, serving as the chair of the Department of Systems, Control, and Industrial Engineering from 1995 to 1998. He was the chair of the Electrical Engineering Department at the University of Washington from 1998 to 2003. Currently, he is a research thrust co-leader for the US National Science Foundation Engineering Research Center for Sensorimotor Neural Engineering and also the codirector of the University of Washington BioRobotics Laboratory. He is a fellow of the IEEE and the AIMBE.