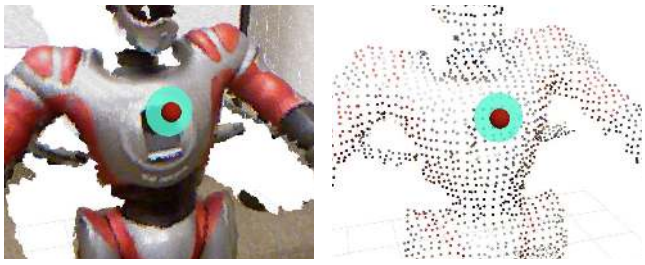
2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Point Clouds Can Be Represented as Implicit Surfaces for Constraint-Based Haptic Rendering

Adam Leeper*, Sonny Chan†, and Kenneth Salisbury†‡
*Department of Mechanical Engineering, †Department of Computer Science, ‡Department of Surgery
Stanford University, Stanford, CA 94305
Email: {aleeper, sonnycs}@stanford.edu, jks@robotics.stanford.edu

***Abstract*— We present a constraint-based strategy for haptic rendering of arbitrary point cloud data. With the recent proliferation of low-cost range sensors, dense 3D point cloud data is readily available at high update rates. Taking a cue from the graphics literature, we propose that point data should be represented as an implicit surface, which can be formulated to be mathematically smooth and efficient for computing interaction forces, and for which haptic constraint algorithms are already well-known. This method is resistant to sensor noise, makes no assumptions about surface connectivity or orientation, and data pre-processing is fast enough for use with streaming data. We compare the performance of two different implicit representations and discuss our strategy for handling time-varying point clouds from a depth camera. Applications of haptic point cloud rendering to remote sensing, as in robot telemanipulation, are also discussed.**



(a) (b)

Fig. 1. The algorithm presented in this paper works on arbitrary point cloud data. Clouds collected by one or more sensors, such as the scene in (a) viewed by an RGB-D camera, can be updated and rendered at interactive rates. (b) Lower point densities are handled well by the algorithm. The haptic interaction point is the dark sphere and the local constraint is the green disk.

I. Introduction

Haptic rendering of virtual or remote environments requires representing objects in a form that can be processed efficiently in the haptic rendering pipeline. In a virtual environment it is common to use simple mathematical representations such as potential fields, geometric primitives, and triangle meshes to represent objects and constraints. In a remote teleoperation situation, however, the environment may not be known in advance and can only be measured by sensors. The rise of widely-available, low-cost RGB-D cameras promises to dramatically improve remote sensing capabilities in robotics; our motivation for this work is to facilitate exploration and remote teleoperation in dynamic environments by finding a method to compute haptic interaction forces and constraints from 3D point data.

Sensors such as RGB-D cameras and laser scanners produce dense 3D point clouds (Figure 1). However, attempts to fit point data into traditional structures (shape primitives and polygon meshes) face significant limitations. Shape primitives are of limited use for arbitrary objects. Sensor noise reduces the effectiveness of mesh tesselation methods, and mesh representations make strong assumptions about the connectivity and topology of point data that are not suitable for the general case of multiple sensors viewing a dynamic scene with asynchronous data acquisition.

We propose that point data can be rendered directly as an implicit surface, allowing us to adapt the rendering algorithm described by Salisbury [10]. Pre-processing is minimized because the implicit surface can be computed locally near the haptic interaction point (HIP) at each servo cycle.

We use two approaches for computing the implicit surface, both of which are widely used in graphics but have seen little use in haptic rendering. In one approach the points are used to form a metaball surface (also known as a soft object [13] or a convolution surface [2]). This method treats each point as a representation of a probable obstacle, so it excels at rendering thin objects and sparse clouds with poorly defined outer surfaces. In the second approach, an implicit surface is defined using the weighted average of nearby points, as in [1]. It is to this definition we refer when we mention 'point-set surface' in this work. This method is better suited to point clouds that closely follow the surface of a closed object. For both methods we argue that the clouds can be downsampled significantly while retaining most of the necessary shape information, facilitating fast updates from sensors.

The contributions of this work are:

- Application to haptic rendering of two implicit surface representations of unorganized point data.
- Investigation of the qualities of these representations for rendering of static and time-varying point sets.

II. Related Work

Haptic interfaces provide a means for an operator to interact with physically-remote or entirely virtual objects and to feel the forces resulting from the interaction. The forces displayed to the operator are governed by haptic rendering algorithms. Constraint-based methods, such as the god-object [14] or virtual proxy [8] algorithms, are the most preferred for 3-DOF haptic rendering. In these methods the haptic

978-1-4673-1405-3/12/$31.00 ©2012 IEEE

device is typically represented by a virtual point or sphere, and this point is constrained to the surface of the object geometry when in contact. The force rendered to the user is proportional to the difference between the virtual and actual device positions. Unlike penalty-based methods, which compute contact forces from object penetration distance, constraint-based methods do not suffer from the problems of popping through thin objects or an incorrect force summation from overlapping objects. These algorithms were originally developed to render polygonal mesh geometry, as this was the most prevalent representation of virtual objects used for visual rendering at the time.

The advent of RGB-D camera technology provides us with a new kind of object representation to understand and explore. A natural path to allowing haptic interaction with RGB-D data is to tessellate the organized grid of depth values in 3D to form a special kind of triangle mesh called a *terrain*. This mesh can be rendered using the original constraint-based methods, although the tight density of mesh elements sometimes poses special problems in the simulation of the proxy's motion. Walker & Salisbury [11] developed the proxy graph algorithm to address these problems and greatly accelerated the simulation of the proxy's motion over this type of mesh structure. Cha *et al.* [3] proposed the use of this algorithm specifically for haptic rendering of depth image data and expanded the method to render surface properties, such as friction, on the mesh.

A straightforward tessellation of the depth image points, however, may not be the ideal representation of the geometry of the scene. Aside from generating an abundance of triangles, the tessellated mesh can suffer from a number of problems when rendered with a constraint-based algorithm. Firstly, the depth data is acquired through a physical sensor whose measurements are subject to noise. A small depth perturbation on a densely sampled surface that should be smooth can cause a peak or valley to form on the tessellated mesh, and we can see that it would be easy for the proxy to catch momentarily on a peak or in a valley as it moves over the triangles of the surface. Secondly, the sharp edges of the triangular mesh elements can be perceived, especially if a high-resolution haptic interface is used for rendering. These edges are an artifact of the tessellation, and not a feature on the object itself. Force shading [8] can mitigate this problem, but cannot completely eliminate it because the mesh geometry is unchanged.

El-Far *et al.* [4] proposed a method that allows direct haptic interaction with the point cloud data without first creating a tessellated mesh. The points are grown into overlapping regular hexahedral regions for the purposes of collision detection, and then the proxy moves from point to point to minimize the distance to the device position. In essence, their method approximates the original 3-DOF constrained motion simulation over a discretized surface formed by the point cloud. A drawback of this method is that the descretization in the proxy motion may lead to perceptible discontinuities in the rendered force.

A recent interest in point-based representations from the graphics community has led to the development of a rigorous theory on surfaces induced by point sets. Smooth, manifold surfaces can be defined to approximate unorganized sets of points, and computational tools were developed for various operations on such surfaces. One often-used definition of a surface is based on a moving least-squares (MLS) approximation of the points in a local, compact region of support [7]. However, evaluating or querying an MLS surface (e.g. testing for containment or intersection) can be computationally costly. Adamson & Alexa [1] presented another popular scheme, wherein they used a weighted average of points from a local region of support to derive functions which define a local implicit surface and a local normal direction.

With this in mind, we discuss other works that look at surface properties in a local region of a point cloud. Lee & Kim [5] described a penalty-based method for rendering a point set surface defined by an MLS approximation. They used MLS projection, accelerating the surface queries with a bounding volume hierarchy of swept-sphere volumes, to determine the nearest point on the surface from the haptic device position. The contact force is based on the computed penetration distance.

Most recently, Ryden et. al. [9] achieved results similar to ours using an iterative stepping algorithm for motion of the proxy. Advantages of our implicit surface approach are that surface effects such as friction can be applied [10] and the algorithm is readily extensible to 6-DOF rendering techniques. For example, the authors' work in [6] uses the surface representation described herein for a 6-DOF telerobotic grasping application.

III. Computing the Implicit Surface

This section explains the two methods for mathematical formulation of the implicit surface from a set of points. Our algorithm also relies on techniques to increase speed and robustness to noise, which are discussed in section IV.

The implicit rendering algorithm in [10] requires the ability to evaluate the magnitude and gradient of a 3D scalar field, $f(x, y, z) = 0$. It uses this function to find a local tangent plane constraint for the implicit surface; the haptic proxy is constrained to always lie on or above this plane, and the haptic force is computed by applying a virtual spring between the proxy and the haptic interaction point (HIP). The location of the tangent constraint and proxy are updated as the HIP moves inside the surface. The primary advantage of this algorithm is that it prevents pop-through, even if the HIP has penetrated through an object into free-space.

We use two methods for creating a scalar field equation from point data. As we describe below, the "metaball" approach treats each point as a piece of constructive geometry, letting us render points that do not represent an oriented surface. For example, a single line of points would render as a blobby cylinder encasing the points. The "point-set surface" approach approximates the local surface in a region of points, requiring point sets that more closely resemble surfaces, but producing an implicit surface that lies *within* rather than *around* the points.

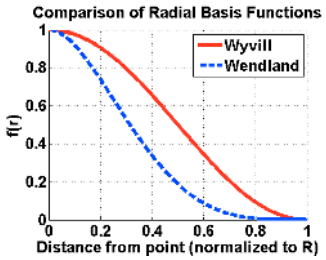


Fig. 2. The radial basis functions used to weight the strength of each point have the useful property that $f(r > R) = 0$, and $f'(0) = f'(R) = 0$.

In either case, we need a method to weight the contribution from local points to the computation of the implicit surface. For this purpose, we use a compactly supported radial function as described below.

A. Point Weighting Using Radial Basis Functions

At each haptic update we estimate the local surface near the haptic proxy. We assume that this can be estimated from points in a small neighborhood, and we weight the contribution from each point based on its distance to the query location. A physically motivated function with infinite support, such as an inverse-square, is generally avoided because it is computationally expensive and does not allow for efficient spatial partitioning. Instead, we use functions with compact support, that is, they are non-zero only on the range $[0, R]$, where R is picked as described in section III-D.

We explored the soft-objects function $C(r)$ described by Wyvill [13] and the Wendland function $\psi_{3,1}(r)$ in [12], referred to here as simply $\psi(r)$.

$$C(r) = 1 - \frac{4}{9}r^6 + \frac{17}{9}r^4 - \frac{22}{9}r^2 \quad \text{(Wyvill)} \quad (1)$$

$$\psi(r) = (1-r)^4(4r+1) \quad \text{(Wendland)} \quad (2)$$

As shown in Figure 2, these functions have the useful property that $f(r > R) = 0$, and $f'(0) = f'(R) = 0$. In this way we are assured that moving incrementally closer to new points will smoothly increase their weight and avoid any sharp jumps in the calculation of the scalar field. In the remainder of this paper we will use $w(\mathbf{x})$ as a generic reference to any weighting function at the position $\mathbf{x}$.

B. Metaball Surface Representation

We motivate a metaball surface representation by assuming that points in the cloud represent an occupied region of space, with some probability function $w(\mathbf{x})$. Regions in space near many points will have a high probability of being occupied, so we want to prevent movement through regions wherein the probability exceeds some chosen threshold. Formally, the haptic surface is an isosurface on probability; the size of the isosurface depends on the maximum radius R of the support function and the threshold value, T, as illustrated in Figure 3.

The gradient of a single metaball at location $\mathbf{p}_i$ is $\frac{\mathbf{x}-\mathbf{p}_i}{||\mathbf{x}-\mathbf{p}_i||} w_i(\mathbf{x})$, but we use the *negative* of this since, by convention, the gradient of an implicit surface points outward.

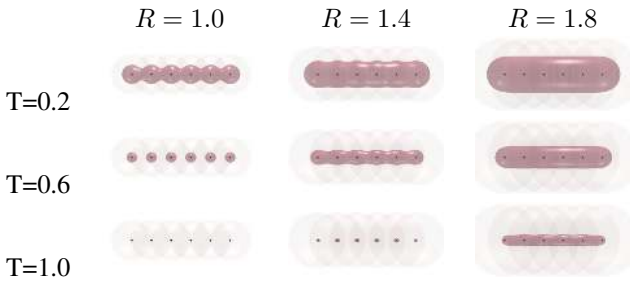


Fig. 3. A uniform line of points with the resultant metaball isosurface is shown for various values of radius R and threshold T. A mid-value threshold and a large radius provide the best smoothness at the cost of reduced sensitivity to sharp, high-frequency features in the cloud. The dark dots are points, the red region is the isosurface, and the light circles are the finite radii of the weighting function for each point.

The net implicit function, $f(\mathbf{x})$, and gradient, $\nabla f(\mathbf{x})$, are:

$$f(\mathbf{x}) = T - \sum_i w_i(\mathbf{x}) \quad (3)$$

$$\nabla f(\mathbf{x}) = -\sum_i \left(\frac{\mathbf{x} - \mathbf{p_i}}{||\mathbf{x} - \mathbf{p_i}||} w_i(\mathbf{x}) \right) \quad (4)$$

In the graphics literature this is frequently referred to as a metaball equation, and is often used in fluid simulations or other situations that call for smooth constructive geometry. This smoothness is highly desirable for haptic rendering, yet we note that the results can be somewhat wavy or bumpy, depending on parameters and noise. We regard this as a benefit for certain applications wherein the points are relatively sparse and are used as constructive geometry. However, if the point data are sampled from the surfaces of objects *and* we know or can reliably compute the appropriate surface normal, we use the representation described in the following section.

C. Point-Set Surface Representation

Raytracing of point data has been well explored for producing images, and we observe that the resulting equations can be used for haptics as well. We choose to use the method in [1] because it is simpler and more efficient than MLS.

For a given query location, $\mathbf{x}$, we define the weighted average $\mathbf{a}(\mathbf{x})$ of the point positions $\mathbf{p_i}$, and the weighted average $\mathbf{n}(\mathbf{x})$ of the point normals.

$$\mathbf{a}(\mathbf{x}) = \left(\sum_i \omega_i \mathbf{p}_i \right) / \left(\sum_i \omega_i \right) \quad (5)$$

$$\mathbf{n}(\mathbf{x}) = \left(\sum_i \omega_i \mathbf{n}_i \right) / \left(|| \sum_i \omega_i \mathbf{n}_i || \right) \quad (6)$$

Then the implicit equation and gradient are:

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x})^T (\mathbf{x} - \mathbf{a}(\mathbf{x})) \quad (7)$$

$$\nabla f(\mathbf{x}) \approx \mathbf{n}(\mathbf{x}) \quad (8)$$

In words, $f(\mathbf{x})$ tells us whether a given point in space lies 'above' or 'below' $\mathbf{a}(\mathbf{x})$ along the normal direction. This method acts as a better low-pass filter than the metaball approach, minimizing bumps in the rendered surface.

D. Parameter Selection

Selection of the radius of influence R (and surface threshold T for a metaball surface) has a strong influence on the surface computation. While each cloud can benefit from tweaking of the parameters by hand it is best to have an automated way to calculate the parameters. We observe that the value of R should be dependent on the density of the cloud. We compute the average separation s between points in the cloud, and R is set to be some multiple m of the spacing, $R = m \cdot s$. Small values of R tend to produce higher frequency features in the rendered surface but can also leave holes and result in pop-through. Large values of R undermine the efficiency of any spatial partition structure and, for the metaball surface, make a very "fat" surface around the points. We discuss selection of the value of m in section V-A.

We can support clouds with non-uniform densities by allowing the value of R to vary across points. Instead of computing a global value of s and R, we can assign a value of R to each point individually as a function of the average distance to its nearest k neighbors (we used $k = 3$). For rendering, we choose R of the point nearest to the proxy as the radius for neighbor searches. We assume that cloud density varies slowly, such that points in a neighborhood will have roughly uniform support radii. Although in theory this can produce minor geometric discontinuities as R jumps in value from point to point, in practice this is imperceivable because the basis function diminishes quickly near the outer edges.

IV. Methods for Real-Time Rendering

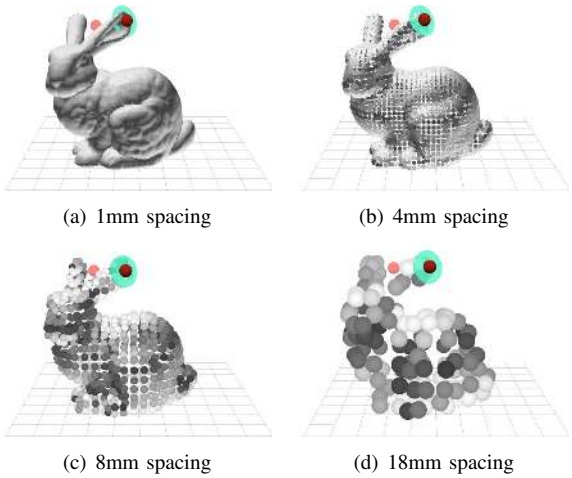
The methods described above will work for a point cloud of arbitrary size, but real-time performance is not feasible if the cloud is too large. We note that stable rendering of stiff surfaces is improved at high servo rates; the standard is 1kHz. In this section we describe key insights and methods that allow us to render a reaslistic haptic interaction at high rates and handle real-time updates to the environment.

A. Fast Collision Detection

An accelerated collision detection structure is essential for rendering the large data sets encountered, as there is insufficient time in a haptic rendering cycle to perform a linear time collision search. Point data has the favorable quality that it is easily partitioned using a kd-tree, which nominally has *O*(log n) search time. Note that we use a kd-tree because we want to avoid assumptions about the density of the point data. That said, since we downsample the data using a voxel grid filter (section IV-B) we could use a spatial hash or octree instead. Points further than a distance R from the proxy do not contribute to the surface computation at that location, so we set our kd-tree search radius to R.

B. Downsampling the Cloud

A single 640×480 depth image contains over 300,000 points, and using multiple sensors or a history of depth images could quickly give millions of points to process. We argue that clipping and downsampling the cloud are effective measures that do not impact the perception of the surfaces.



(a) 1mm spacing (b) 4mm spacing

(c) 8mm spacing (d) 18mm spacing

Fig. 4. The algorithm adapts easily to any cloud density, so we downsample the cloud based on the desired frequency content to improve preprocessing efficiency. The proxy is shown in black with the local tangent constraint disk in green; the HIP is in light red.

For clipping, consider that at any time step we need only consider the cloud points near the haptic proxy. If we assume a maximum velocity of the haptic tool and a given rate of cloud updates then we can calculate a bounding volume for the points that may be encountered before the next update. All other points can be clipped from any pre-processing steps, such as building the spatial partition structure.

For downsampling, we consider the spatial frequency content of the point cloud. As discussed previously, algorithms that rely on tesselation of a depth image may be introducing unwanted sharp edges into the haptic surface. Since we know the cloud is derived from a sensor with noise it is reasonable to assume that the desired signal (the surface) has a significantly different bandwidth than the noise so that a filter can be applied. In our case a spatial filter that reduces frequency content while downsampling the cloud is highly desireable as it can dramatically decrease the number of points while also enforcing a measure of uniform spacing in the cloud. For this purpose we use a voxel grid filter [1] with adjustable grid size, generally 2-5mm (see Figure 4).

C. Multiple Point Sources

One advantage of our rendering strategy is that it easily handles overlapping views of the same region, such as when the same scene is captured through multiple cameras. Collision detection can be done in parallel. That is, the nearby points from each source can be queried separately. The union of these points can be used to calculate the implicit surface equation with no modification to the algorithm.

We note that this case is not easily handled by previous explicit surface methods. Tesselating a depth image is fast

[1]This filter partitions space into a uniform rectilinear grid and downsamples a point cloud by replacing all points in a given cell with a single point whose position is the average of those points.

because the points are ordered. If two depth images overlap, there is a dilemma. If the points are combined before tesselation, they will no longer be ordered, making the operation prohibitively slow. If the images are tesselated separately, then collision detection and force rendering must be done on two meshes that may weave in and out of each other.

D. Transparent Cloud Updates

Since the typical 1 kHz haptic update rate is much faster than any other update rate in the system (e.g. 30 Hz camera), it is important to consider how data will be updated in the rendering pipeline. There are two main issues:

1) The sudden change in cloud composition tends to result in force discontinuities, which feel like vibration.
2) Pre-processing cannot interfere with the haptic update.

The first problem is solved by storing and using N point clouds for collision detection, essentially averaging the surface location to minimize vibration effects due to noise. We query the local neighbors from all point sources and compute the surface from the union of these points. For the metaball case, we avoid making the surface "fatter" by dividing the weighting function by N. For the point-set surface this is not needed, and the normals are not jeopardized by the additional noise covariance from multiple clouds if normal estimation is done per-cloud. We observe that this solution is most appropriate for a quasi-static scene. Future work will address a formulation that handles filtering in time at the correct bandwidth.

For the second issue, the time between receiving a new point cloud and when it is ready to use for rendering can vary between 15ms to hundreds of milliseconds, depending on the number of points. It is not acceptable to force the haptic update loop to wait while the new cloud is prepared. To solve this, we maintain $N + 1$ point clouds for each source, and up to N of them are active at any time. This requires more memory but is essential for fast updates. We prepare the new cloud in a separate thread while haptic interaction continues with the previous clouds. When the new data structure is ready, the pointer to the new cloud is swapped in before the start of the next haptic update.

V. Results and Discussion

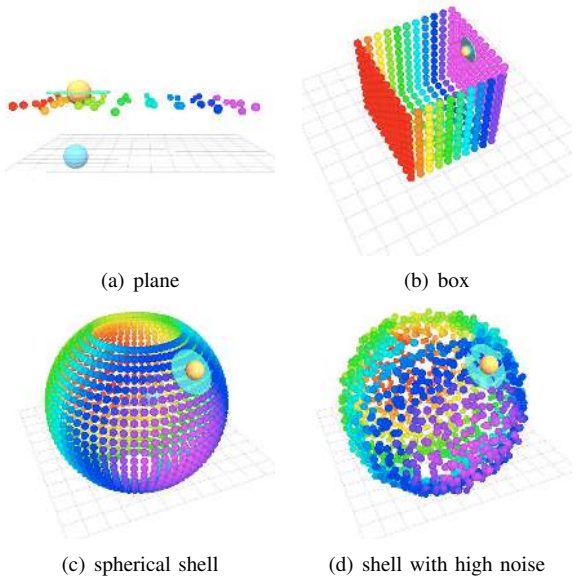
The proposed algorithm was implemented on an Intel PC (Core i7-950 3.06GHz, 6 GB RAM) under Ubuntu Linux. We used CHAI 3D (chai3d.org) to interface with a Phantom Omni haptic device (sensable.com). All visualization was done using the Rviz tool in ROS (ros.org). Our depth camera was the Microsoft Kinect™ using the OpenNI driver in ROS.

We profiled the speed of the costly parts of our cloud pre-processing pipeline. The results are shown in Figure 5. We observe that normal estimation is much more costly than creating the spatial partition and estimating cloud parameters; naturally, a multi-threaded or GPU-based normal estimation strategy would help.

The two implicit surface representations have different strengths. An advantage of the metaball surface representation is reduced pre-processing time since point normals are not needed. The metaball representation also works better on thin or sparse areas, such as the ear of the bunny in figure 4(d), as the point-set surface cannot handle cloud regions that do not closely resemble a surface. Conversely, the point-set surface is much smoother in cases where a surface is clearly intended but masked by noise, such as figure 6(a,d).

points	build kd-tree	estimate normals	find spacing
146,000	135ms	780ms	275ms
36,000	26ms	160ms	60ms
16,000	10ms	70ms	25ms
2,500	1.4ms	10ms	3.7ms

Fig. 5. Elapsed times for cloud pre-processing steps. The metaball method does not require normal estimation, while the point-set surface requires all three preprocessing steps.



(a) plane (b) box

(c) spherical shell (d) shell with high noise

Fig. 6. A variety of sythetic objects. The proxy is shown in yellow with the local tangent constraint disk in green; the HIP is in blue.

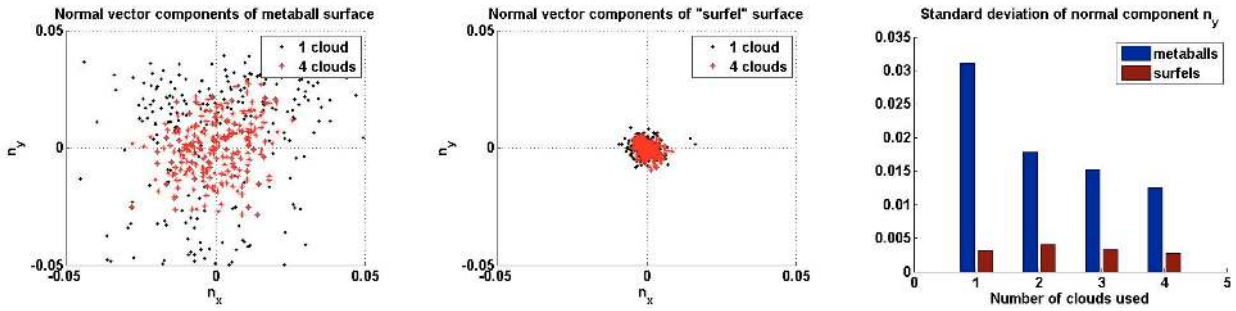
A. Static Point Sets

We tested the algorithm on a variety of static synthetic data sets (Figure 6), with and without noise. We also used some ubiquitious high-density point sets from the literature, such as the Stanford Bunny (Figure 4). We varied the density of the point sets to find a heuristic for selecting parameters for the implicit surface equation, namely, the value of R and T. We found the following, where s is the average spacing between points in the cloud and $R = m \cdot s$:

- For a roughly regular, low-noise point surface, we could select $m \approx 2.0$ and $T = 0.5$.
- For noise amplitudes on the order of the nominal point spacing, a smoother result was obtained for $m \approx 3.0$ to $m \approx 5.0$ and $T = 1.0$.

B. Streaming RGB-D Camera Data

We tested the algorithm and our full system on a live feed from a Microsoft Kinect RGB-D camera. For a quasi-static scene (e.g. Figure 1) the interaction force was very



(a) (b) (c)

Fig. 7. The tangential components (n_x, n_y) of the surface normal have high variability over time when using (a) the metaball surface, while (b) the point-set surface is smoother over time. (c) Keeping a history of previous clouds dramatically improves the performance of the metaball algorithm.

stable in most regions; regions at the edge of an object were susceptible to some vibration, but the effects were mitigated by the averaging technique described in IV-D.

To evaluate the stability of the haptic isosurface over time, we pointed the Kinect at a static, horizontal surface. To simulate a robot exploring a remote environment, the Kinect was angled down 27° from horizontal, and the test point was 70 cm from the Kinect. We pressed the haptic device into the surface at the fixed test point and measured the height and surface normal of the proxy over time. The isosurface was computed using $T = 0.5$ and $m = 2.5$. The number of clouds used for rendering varied from 1 to 4.

The results, shown in Figure 7, was that the stability of the surface normal was quite different between the two algorithms. The tangential components of the surface normal have much higher variablity using the metaball surface compared to the point-set surface when using only 1 cloud, though using 4 clouds reduces this effect dramatically. While slight changes in the 'height' of the surface over time are mostly imperceptible, the changes in direction of the haptic force can be rather unsettling when exploring the surface with a light touch.

VI. CONCLUSIONS AND FUTURE WORK

The ability to haptically render point cloud data has utility in multiple contexts such as enabling users to feel the shape of remote, inaccessable, or other noisily defined objects. It is also of value in remote manipulation contexts where operator motions need to be constrained by surfaces that are visually detected in realtime. We have presented an algorithm for interaction with arbitrary point clouds, like those acquired by one or more range scanners like an RGB-D camera. We have demonstrated the applicability of implicit rendering techniques to point data and described the advantages of two methods for calculating a surface equation. We have also demonstrated methods to improve efficiency and robustness to noisy data.

A natural application area for this work is settings where limited sensing results in noisy and incomplete environment data. In parallel work we are using the algorithms from this paper to enable 6-DOF interaction between a proxy object (e.g. a robot gripper) and a point cloud acquired by a robot in a remote manipulation scenario. We are also working to formalize our approach to spatial and temporal filtering, which will be the subject of future work.

ACKNOWLEDGMENTS

A. Leeper is supported in part by a National Science Foundation GRFP Fellowship. S. Chan is supported in part by a post-graduate scholarship from the National Science and Engineering Research Council (NSERC) of Canada.

We thank Reuben Brewer and Günter Niemeyer for discussions to refine ideas and for helping with revisions.

REFERENCES

[1] A. Adamson and M. Alexa. Approximating and intersecting surfaces from points. In *Proceedings of the 2003 Symposium on Geometry Processing*, pages 230–239. Eurographics Association, 2003.

[2] J. Bloomenthal and K. Shoemake. Convolution surfaces. In *Proceedings of SIGGRAPH '91*, pages 251–256, New York, New York, USA, 1991. ACM Press.

[3] J. Cha, M. Eid, and A. El Saddik. DIBHR: Depth Image-Based Haptic Rendering. *Haptics: Perception, Devices and Scenarios*, 2008.

[4] N. R. El-Far, N. D. Georganas, and A. El Saddick. An algorithm for haptically rendering objects described by point clouds. In *Canadian Conference on Electrical and Computer Engineering*, 2008.

[5] J.-K. Lee and Y. J. Kim. Haptic rendering of point set surfaces. *World Haptics Conference*, 0:513–518, 2007.

[6] A. Leeper, S. Chan, and K. Hsiao. Constraint-Based Haptic Rendering of Point Data for Teleoperated Robot Grasping. In *Proceedings of IEEE Haptics Symposium*, 2012.

[7] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1532, Oct. 1998.

[8] D. C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *In SIGGRAPH '97*, pages 345–352, 1997.

[9] F. Ryden, S. Nia Kosari, and H. J. Chizeck. Proxy method for fast haptic rendering from time varying point clouds. In *Proceedings of IROS, 2011*, pages 2614–2619. IEEE, Sept. 2011.

[10] K. Salisbury and C. Tarr. Haptic Rendering of Surfaces Defined by Implicit Functions. In *Proceedings of 6th Annual ASME Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 61–67, Dallas, TX, 1997.

[11] S. P. Walker and J. K. Salisbury. *Large haptic topographic maps*. ACM Press, New York, New York, USA, 2003.

[12] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, Dec. 1995.

[13] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, Aug. 1986.

[14] C. Zilles and J. Salisbury. A constraint-based god-object method for haptic display. In *Proceedings of IROS, 1995*, pages 146–151. IEEE, 1995.