SURFACE RECONSTRUCTION BY RESTRICTED AND ORIENTED PROPAGATION

Xavier Suau *Josep R. Casas* *Javier Ruiz-Hidalgo*

Signal Theory and Communications Department, Technical University of Catalonia
UPC-Campus Nord, C/ Jordi Girona, 1-3, 08034, Barcelona
E-Mail: {xavier.suau, josep.ramon.casas, j.ruiz}@upc.edu

ABSTRACT

This paper considers the problem of reconstructing a surface from a point set. More specifically, we propose a method which focuses on obtaining fast surface reconstructions for visual purposes. The proposed scheme is based on propagation in a voxelized space, which is performed in the directions defined by a propagation pattern, during an optimal number of iterations. Real-time applications are conceivable thanks to a low execution time and computational cost, keeping an acceptable visual quality of the reconstruction.

Index Terms— Surface Reconstruction, Mesh generation, Point Sets, Propagation

I. INTRODUCTION

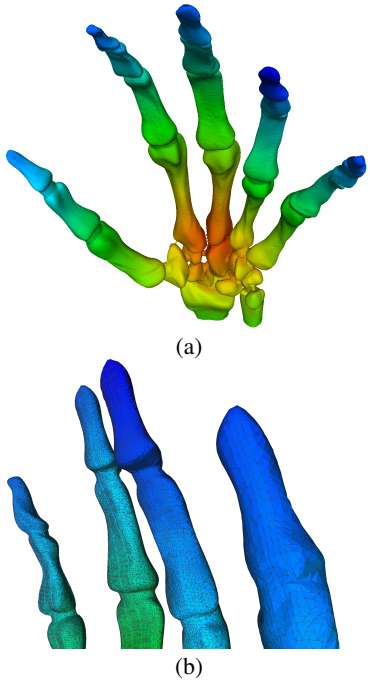
Point sets defining surfaces are very common in many recent applications due to technical improvements in scanners and z-cams. The size of such point sets (PS) may vary from some hundreds to some millions of points. Therefore, computation time becomes an important bottleneck of any PS processing algorithm. We present in this article a novel method to obtain meshed surfaces from point sets in a very fast, up to real-time, way.

The proposed algorithm may perform in applications where a low computation time is essential. For example, one may think of scene reconstruction in a multicamera environment, providing a 3D meshed version of the scene close to real-time. Surface reconstruction of huge models (millions of points) may also be performed in a reasonable, user-friendly time. In general, most applications derived from reconstructed surfaces are related to visualization of the obtained 3D object. Thus, mathematical properties of the meshes are not as important as calculation time or first glance appearance, for example.

Many surface reconstruction techniques have been developped over the past years. Bernardini *et al.* [1] proposed the ball-pivoting algorithm where a virtual ball of a user-specified radius scans the PS. When the ball touches three points without containing any other point, a triangular face is found. Poisson Reconstruction proposed by Kazhdan *et al.* in [2] solves a Poisson equation trying to best approximate the vector field defined by the samples. In [3], Guennebaud and Gross exploit Algebraic Point Set Surfaces (APSS), which are an extension of the Point Set Surfaces representation technique developped by Alexa *et al.* in [4] and [5]. A technique to render APSS is presented by Guennebaud *et al.* in [6]. The Poisson Reconstruction scheme leads to a very smooth surface which is not able to deal with sharp features. APSS Rendering and Ball-Pivoting are precise with sharp details but their computational cost is prohibitive for applications close to real-time. These three methods are implemented for the MeshLab software presented by Cignoni *et al.* in [7].

Our proposal relies on a prior voxelization of the subspace in which the PS is confined. Spatial resolution, or voxel size, is a parameter of the algorithm, which is related to the reconstruction accuracy. Spatial resolution may be arbitrarily chosen by the user depending on the required reconstruction quality and memory resources. A second parameter of our scheme is called propagation pattern. We will see how a smart choice for the propagation pattern may strongly reduce the computation time while preserving reconstruction quality.

This work has been partly supported by the project TEC2007-66858/TCM PROVEC of the Spanish Government



(a) (b)

Fig. 1. (a) Meshed surface of the *hand* PS (327K points) obtained in $5.44\,s$ with the proposed method. Ball-Pivoting, Poisson Reconstruction and APSS Rendering took $1513\,s$, $30\,s$ and $413\,s$ respectively to mesh this PS. (b) Finger tips detail.

Several PS of different sizes are used to test the proposed algorithm for surface reconstruction. Experimental results (meshed surfaces) obtained with our method are evaluated and compared with the state-of-the-art methods cited above. Accuracy of the resulting surface is measured as the Hausdorff Distance to a groundtruth surface as proposed in [8] by Aspert *et al.* for their M.E.S.H. software.

The remainder of this paper is organized as follows. In the next section, voxelization details, the propagation algorithm and a propagation stop threshold are presented. In Section III, the results and performance obtained by our system are discussed and compared to the other existing methods. Finally, in the last section, conclusions are drawn and the direction of our future work in this subject is presented.

II. PROPAGATION ALGORITHM

The proposed method is divided into 3 steps; voxelization, propagation and stop, which are depicted hereafter.

II-A. Voxel Filling

The subspace $\mathcal{B}$, of size $[\mathcal{B}_x, \mathcal{B}_y, \mathcal{B}_z] \in \mathbb{R}^3$, which contains the point set $\mathcal{S}$ is divided into $N_x \times N_y \times N_z$ voxels as defined in (1). Therefore, the resulting voxels are cubes with edges of length dV, which is a parameter of our system. Voxel size dV is directly our spatial resolution, its choice being crucial since it states a compromise between resolution and computational load. Indeed, reducing dV will increase spatial resolution, which offers more texture detail and sharp features preservation. On the other hand, high resolution implies a high computational load and memory requirements.

The choice of dV is done by the user and should take into account many terms such as the dimensions of the 3D model, the density of $\mathcal{S}$, the demanded resolution or the available memory amongst others.

$$N_i = \left\lceil \frac{\mathcal{B}_i}{dV} \right\rceil \qquad i = x, y, z \tag{1}$$

Once $\mathcal{B}$ has been voxelized, one may proceed to associate each 3D point in $\mathcal{S}$ to a voxel. As usual, every point $p_i \in \mathcal{S}$ consists of a position $\mathcal{P}_i = (x, y, z)$ and a color $\mathcal{C}_i = (r, g, b)$. Three situations may be considered depending on the number of points which are associated to a given voxel υ_k:

0 points in voxel :
$\nexists p_i \in \upsilon_k \Rightarrow \upsilon_k \leftarrow \varnothing$

1 point in voxel :
$p_i \in \upsilon_k \Rightarrow \upsilon_k \leftarrow (\mathcal{P}_i, \mathcal{C}_i)$

m points in voxel :
We may calculate the mean position $\overline{\mathcal{P}}$ and mean color $\overline{\mathcal{C}}$ as $(\overline{\mathcal{P}}, \overline{\mathcal{C}}) = \frac{1}{m}\sum_{i=1}^{m}(\mathcal{P}_i, \mathcal{C}_i)$ of the m points associated to υ_k. Then : $\upsilon_k \leftarrow (\overline{\mathcal{P}}, \overline{\mathcal{C}})$

After the filling step, every voxel containing at least one point is labeled as *seed voxel* or υ^S. Such voxels are characterized by a pair $(\mathcal{P}, \mathcal{C})$ containing the interpolated position and color of the points $p_i \in \upsilon^S$. Seed voxels will perform as propagation sources during the propagation phase.

II-B. Propagation steps

The objective of the propagation algorithm is to find the closest points of every point in $\mathcal{S}$. In order to do so, an iterative algorithm which propagates our seed voxels is proposed.

A *propagation pattern* is a set of positions relative to a propagation source or seed voxel. The most common and intuitive patterns are the 6, 18 and 26 omnidirectional (OMNI) patterns shown in Figure 2.(a-c), which propagate seed voxels in all directions. A more intelligent pattern may perform propagation in a 3D octant of the space, like our 6-OCT pattern presented in Figure 2.(d).

As a matter of fact, we are trying to find pairs of neighbours, which present a reciprocity property. Indeed, direction of neighbour finding is of no importance. For example, when a point P is propagated and encounters its neighbour Q, both know their respective neighbour (noted P $\rightsquigarrow$ Q); so the result is the same than when point Q finds point P. Knowing this, we rapidly see that 6, 18 and 26 OMNI patterns are redundant since they propagate in all directions.

Every pair of found neighbours will define a mesh edge, a post-processing step being necessary to extract triangular faces from the resulting edges. The proposed 6-OCT pattern will be used in the experiments presented in Section III since it limits the number redundant edges and faces of the output mesh and avoids unnecessary calculations.

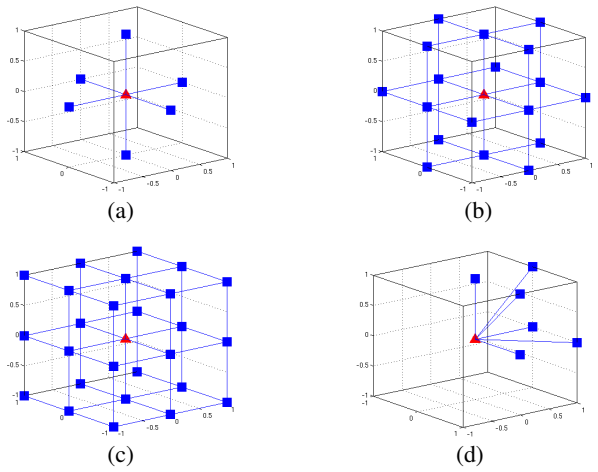


Fig. 2. The red triangle at the origin represents the source voxel, while blue squares shape the propagation pattern. (a) 6-OMNI pattern. (b) 18-OMNI pattern. (c) 26-OMNI voxel pattern. (d) 6-OCT pattern.

During the first propagation iteration, those voxels surrounding (pattern dependant) every seed voxel υ^S are filled. These recently checked voxels are saved as new propagation sources for the next iteration. The group of voxels relative to a same υ_k^S are called *seed volume* or $\mathcal{V}_k$. As iterations increase, seed volumes grow, eventually intersecting other seed volumes. Two seed voxels υ_i^S and υ_j^S are neighbours when their respective seed volumes $\mathcal{V}_i$ and $\mathcal{V}_j$ intersect. Thus, points p_i and p_j, which originated both seed voxels are neighbours too (2), a mesh edge has been found between these two points.

$$\mathcal{V}_i \cap \mathcal{V}_j \neq \varnothing \quad \Rightarrow \quad p_i \rightsquigarrow p_j \tag{2}$$

II-C. Stop Threshold

Propagation iterations should be stopped at the appropriate moment to avoid finding far neighbours which may create redundant edges. The number of created edges per iteration is called edge density or D_e. Such magnitude presents a maximum $D_{e_{\max}}$ at a low number of iterations $\kappa_{\max}$, so that $D_e(\kappa_{\max}) = D_{e_{\max}}$. We choose to stop propagation when the two conditions in (3) are verified (see the example in Figure 3).

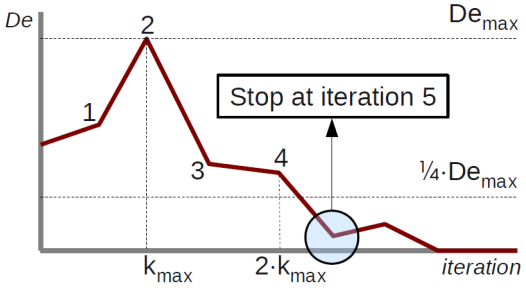
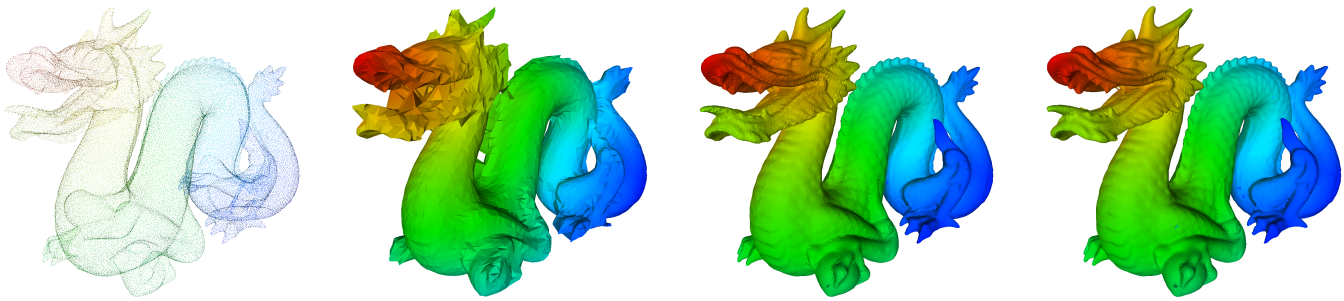


Fig. 3. Example of automatic propagation stop. Propagation stops at iteration 5 where the two conditions in (3) are verified.



(a) (b) (c) (d)

Fig. 4. Surface reconstruction with the proposed ReOP method with (b) $dV = 0.03$, (c) $dV = 0.01$, (d) $dV = 0.006$. (a) Initial *Dragon* data set consisting of 437,645 points.

$$\left(\kappa \geq 2\kappa_{\max}\right) \wedge \left(D_e(\kappa) < \frac{1}{4} D_{e_{\max}}\right) \Rightarrow stop\, at\, iteration\, \kappa \quad (3)$$

As a result, since iterations are Restricted and Propagation is directionally Oriented, the proposed method is named *ReOP*.

III. EXPERIMENTAL RESULTS

Experiments are carried out with four 3D models provided by the Stanford 3D Scanning Repository: *Bunny* (35,947 points), *Hand* (327,323 points), *Dragon* (437,645 points) and *Buddha* (543,652 points). A groundtruth meshed surface is provided along with the PS, which is used for comparison purposes. The proposed ReOP algorithm is evaluated against the Ball-Pivoting [1], Poisson Reconstruction[1] [2] and APSS Rendering [6] algorithms. Experiments are executed on a 64-bit Intel Xeon CPU @ 3.00GHz processor.

The Hausdorff distance (δ_H) metric is used to evaluate the similarity between the obtained surface and a groundtruth surface, giving an idea of the accuracy of the reconstruction. A second parameter taken into account is the overall calculation time (t_O), which includes memory allocation and mesh writing. Other evaluated features are the number of obtained faces or the visual quality.

Results are summarized in Figure 5, where all the algorithms are presented on the (δ_H, t_O) plane. Thus, the further right an algorithm is placed, the more accurate the reconstruction is. The same way, the further up, the faster. Therefore, algorithms placed close to the upper-right corner are the ones which best accomplish the accuracy-speed compromise. Remark that t_O is shown in a logarithmic scale.

Several implementations of the algorithms are evaluated by tunning the associated precision parameters (octree levels for Poisson Reconstruction, voxel grid for APSS Rendering and dV for ReOP). Figure 4 shows three different implementations of the propagation algorithm.

The proposed ReOP algorithm clearly overtakes the other tested methods in the (δ_H, t_O) plane evaluation. APSS rendering achieves a similar quality, however its processing time is usually higher (up to ten times slower than ReOP). Poisson Reconstruction is slightly faster than APSS but provides a lower accuracy. The Ball-Pivoting algorithm is enormously time consuming when dealing with large data sets like those in Figure 5.(b-d).

Our method is the most suitable one to perform for real-time applications. Furthermore, it has been tested that between 70% and 80% of the overall time is due to memory allocation. Thus, getting rid of such task will speed-up our system $\times 5$ approximately. We may think of applications with constant voxelization (memory allocation is done once at initialization), a room or a stage in a multicamera environment for example, where a meshed reconstruction of the scene could be done at some frames per second. For the moment, the current processor and memory set-up allow processing models of 200^3 voxels at 2 frames per second with an acceptable visual quality. Optimization of the code (parallelized execution on GPU's as in [9], [10]) and better machines will lead to performant real-time applications.

[1]MeshLab's Poisson Reconstruction implementation fails with the *Hand* data set.

Method	$\mathbf{t_O}$ [s]	$\delta_\mathbf{H}$	Faces
Poisson 8 levels	30.8	0.000595	162,270
Poisson 9 levels	65.1	0.000184	631,480
Poisson 10 levels	109	0.000192	1,202,802
APSS Grid 600	143	0.000201	1,034,902
APSS Grid 800	298	0.000061	1,612,044
APSS Grid 1000	520	0.000046	2,641,481
ReOP $dV = 0.0015$	2.38	0.000086	252,329
ReOP $dV = 0.0008$	7.23	0.000033	735,881
ReOP $dV = 0.0005$	22.2	0.000025	1,367,336

Table I. Results on the *Buddha* data set (values correspond to those in Figure 5.(d))

Table I summarizes the number of faces obtained with different reconstructions of the *Buddha* PS. The processing time for the Poisson and APSS algorithms include the PS normal extraction step, since these methods require points equipped with normals. Normal extraction takes about 10 to 20 seconds with our hardware.

Poisson reconstruction provides its best mesh at about 1 million faces with $\delta_H = 0.000184$. These results are improved by APSS rendering at the expense of a bigger output mesh of up to 2.5 million faces with $\delta_H = 0.000046$. The proposed ReOP approach both improves the observed results in time and accuracy, fournishing a mesh of about 1.3 million faces for an accuracy of $\delta_H = 0.000025$.

Since APSS rendering needs a prior voxelization, memory requirements may be compared to those of the proposed method. APSS with a grid of 1000 requires 10^9 voxels, while ReOP with a $dV = 0.0008$ (similar δ_H accuracy) requires $90 \times 200 \times 90 = 1.62 \cdot 10^6$ voxels, which is a significant memory reduction. Such difference in memory requirements may partly explain the variation in the execution times of both methods.

Manifoldness cannot be ensured for the mesh obtained with ReOP, which is the main drawback of the proposed scheme. The other methods here presented do ensure such property. Nevertheless, our main objective is to create fast meshes for visual applications, mathematical properties being of second order in this paper.

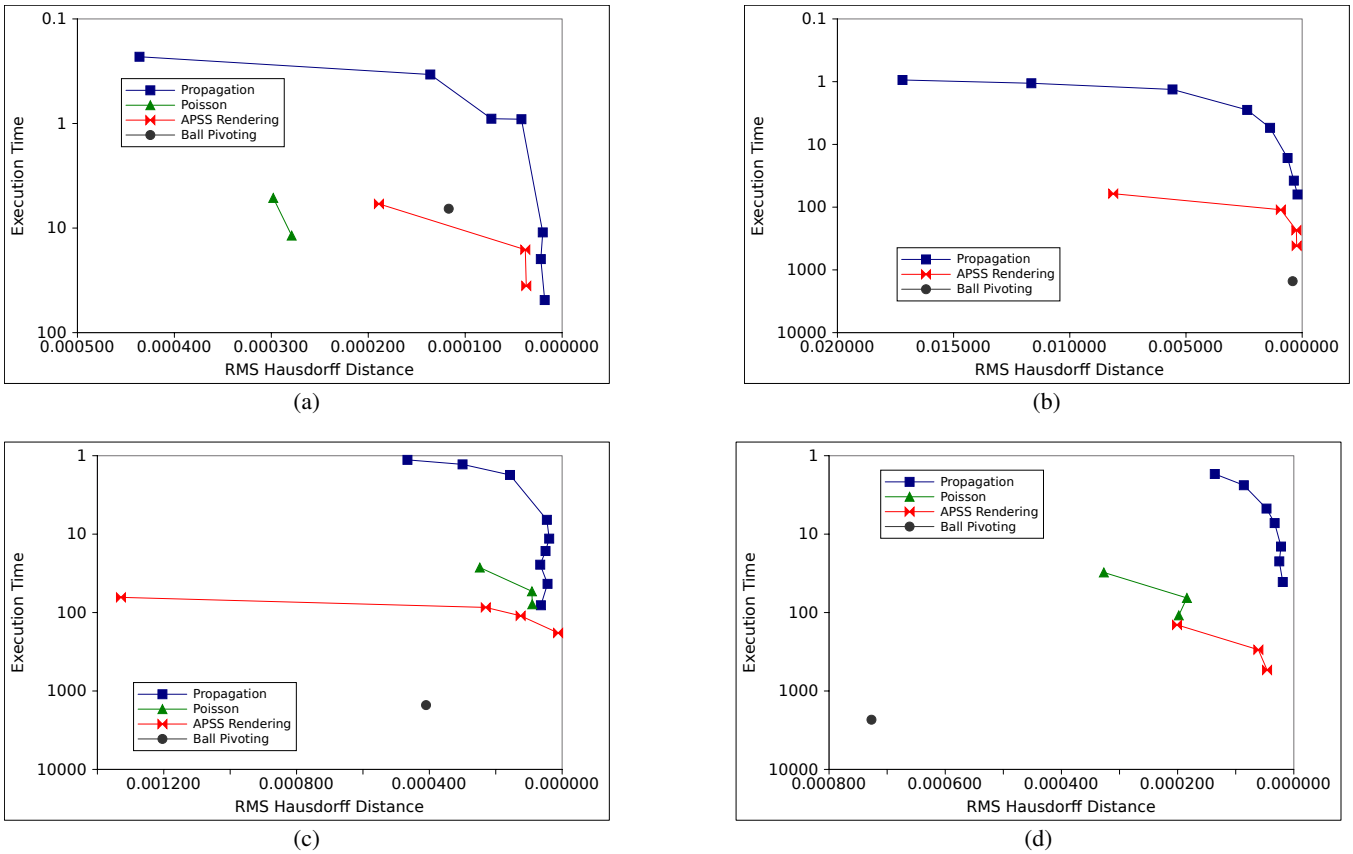


Fig. 5. Positioning of different implementations of the studied algortihms on the Accuracy VS. Time (δ_H, t_O) plane. (a) *Bunny*. (b) *Hand*. (c) *Dragon*. (d) *Buddha*.

IV. CONCLUSION AND FUTURE WORK

A new method to reconstruct meshed surfaces from point sets has been presented. A propagation through a voxelized space is performed to find the closest neighbours of every data point. Propagation is done following a specific propagation pattern which exploits reciprocity in neighbour finding. Seed volumes are utilized to find the closest neighbours of every data point in a fast way. Every pair of neighbours found defines an edge of the output mesh.

Propagation stops after the first maximum of the edge creation density, which indicates when the closest surface points have been found. Such threshold limits the number of redundant faces in the reconstructed surface and also avoids performing superfluous operations.

The proposed approach provides a reconstructed surface of a 3D model in a very fast way, up to ten times faster than the best state-of-the-art methods tested in the experiments. Regarding the observed execution time, real-time applications are conceivable using the proposed propagation approach. A moderate size mesh is obtained, its dimension being comparable to the other existing methods. Visual assessment, as well as measured quality by Hausdorff distance, is similar or better than that provided by other state-of-the-art solutions.

Adapting the propagation pattern to the topology and density of every point set is one of the major issues of our future work. Other tasks will focus on reducing the number of redundant faces and optimizing the code to obtain faster reconstructions.

V. REFERENCES

[1] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, pp. 349–359, 1999.

[2] M. Kazhdan, M. Bolitho, , and H. Hoppe, "Poisson surface reconstruction," in *Eurographics Symposium on Geometry Processing*, 2006.

[3] G. Guennebaud, , and M. Gross, "Algebraic point set surfaces," in *Siggraph*, 2007.

[4] M. Marc Alexa, J. Johannes Behr, D. Daniel Cohen-Or, S. Fleishman, D. Levin, and C.T. Silva, "Point set surfaces," in *Conference on Visualization*, 2001.

[5] M. Marc Alexa, J. Johannes Behr, D. Daniel Cohen-Or, S. Fleishman, D. Levin, and C.T. Silva, "Computing and rendering point set surfaces," in *IEEE TVCG*, 2003.

[6] G. Guennebaud, M. Germann, and M. Gross, "Dynamic sampling and rendering of algebraic point set surfaces," in *EUROGRAPHICS 2008*, 2008, vol. 27.

[7] P. Cignoni, M. Corsini, and G. Ranzuglia, "Meshlab: an open-source 3d mesh processing system," *ERCIM*, pp. 47–48, 2008.

[8] N. Aspert, D. Santa-Cruz, and T. Ebrahimi, "Mesh : Measuring errors between surfaces using the hausdorff distance," in *IEEE International Conference in Multimedia and Expo (ICME)*, 705-708, 2002, number 1.

[9] T. R. Halfhill, "Parallel processing with cuda," Tech. Rep., Reed Electronics Group, 2008.

[10] L. Nyland, M. Harris, and J. Prins, "Fast n-body simulation with cuda," Tech. Rep., NVIDIA, 2008.