STAG: Smart Tools and Apps in computer Graphics (2016)
Giovanni Pintore and Filippo Stanco (Editors)

Surface Reconstruction from Range Images

G. Mariosa [1], N. Fioraio[1], A. Franchi[1], L. Di Stefano[2]

[1]Datalogic Automation, Bologna, Italy
[2]DISI, University of Bologna, Italy

Abstract

In this work we propose an algorithm for fast and detailed surface reconstruction from range images. Our method extends the well-known Ball Pivoting Algorithm (BPA) so to exploit the inherent structure of range images, outperforming the state-of-the-art both in terms of reconstruction quality and computation time. Moreover, we have investigated on automatic estimation of the algorithm's parameters and, in particular, we propose a novel robust approach to estimate the sequence of radii required by BPA, thus demanding less effort by the user compared to existing solutions.

1. Introduction

Reconstruction of 3D objects is one of the most challenging problems in computer vision research. Traditionally, 3D measurements are collected from a sequence of acquisitions through Lidar/Ladar systems or reconstructed from multiple views [AFS*11]. Then, a mesh is built by connecting the 3D points with triangles. Over the years, many advanced techniques have been proposed for estimating a surface from a Delaunay triangulation [EM94, BMR*99, AB98, ACDL00, ACK01], such as *Alpha-shapes*, *Ball Pivoting Algorithm*, *Crust* and its variants. Most of these methods make few assumptions on the data source and thus often require minutes or hours to reconstruct a scene at high details.

Recently, the increasing availability of low-cost 3D cameras, such as passive stereo heads, structured light systems and Time-of-Flight cameras, has facilitated acquisition of 3D point clouds. These systems often runs at the rate of 15Hz or more and return 3D scans in the form of range images, i.e., point clouds organized on a 2D lattice. However, a few surface reconstruction approaches are designed to exploit such inherent organization and they are either fast but inaccurate [ZRB09] or high quality but slow [ABK98, BMR*99]. In this work we investigate on how to produce a high quality mesh from a single range image at a fraction of the time required by state-of-the-art methods [BMR*99, Dig14, ZRB09, ACDL00, ACK01]. Purposely, we extended the well-know Ball Pivoting algorithm [BMR*99, Dig14] to support fast windowed-based computation on range maps. Moreover, we propose a novel hyper-parameter setting to compute the best algorithm setup by specifying the level of detail of the mesh instead of requiring the user to input known metric values.

The rest of the paper is organized as follows: in Sec. 2 we discuss related work on surface reconstruction, in Sec. 3 we describe the original BPA algorithm and our novel contribution, then experimental results are given in the Sec. 4, while in Sec. 5 we analyze open issues and highlight future directions of work.

2. Related Work

Generally speaking, surface reconstruction has been addressed according to two main approaches: *the combinatorial approach* and the *models fitting approach*. The former is based on the theoretical foundations of computational geometry and it works by establishing adjacency relations between input points. The latter, instead, finds the best fitting of a given model into the given point cloud. One of the most important class of algorithms deploying the combinatorial approach is the Delaunay-based group. Indeed, a facet subset of the 3D Delaunay triangulation is a good approximation of the sampled surface [Boi84]. Based on this concept and extending the 2D *alpha-shapes* definition [EDR83], Edelsbrunner and Mücke [EM94] introduced a formal geometric notion of the *shape* of a point set in the three-dimensional space. *Alpha-shapes* are a generalization of the convex hull and a sub-complex of the Delaunay triangulation. Given a triangulation of the input point cloud, a real parameter, *alpha*, controls the level of detail through *alpha-balls*. Similarly, the *Ball Pivoting Algorithm* [BMR*99] builds a triangular mesh starting from a seed triangle and extending the triangulation insofar as possible. Finally, the Crust algorithm and its variants [AB98, ABK98, ACDL00, ACK01] combines Voronoi diagrams and Delaunay triangulation by using Voronoi vertices to remove triangles from the Delunay triangulation.

As for the model-fitting approach, most methods try to fit a *global* or *local* surface model on the 3D points by reducing the gap between the model and the data [Sal10]. Typically, the model is defined through the implicit form of a specific basis function. A prominent member of this group is the Marching Cubes algorithm [LC87], which extracts an iso-surface from an implicit representation of the data as a distance function. The algorithm locates

DOI: 10.2312/stag.20161371

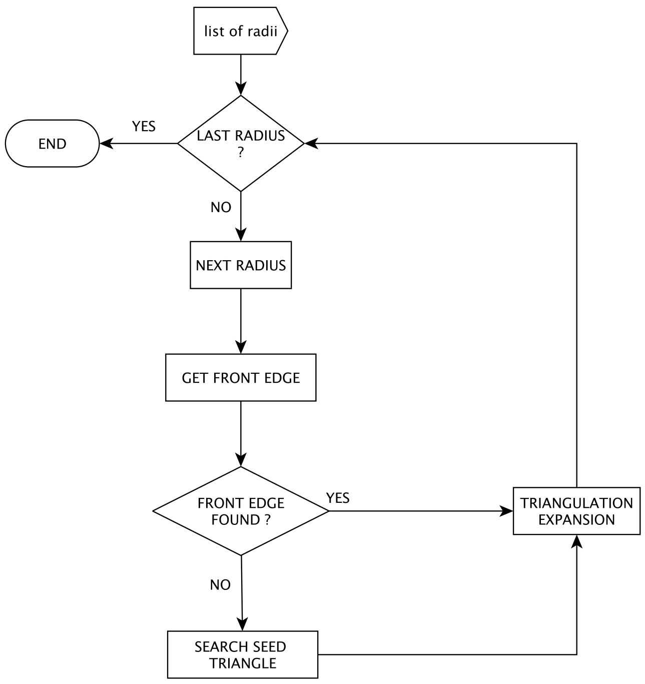

Figure 1: *Flow chart of the Ball Pivoting Algorithm.*

the iso-surface in a voxel grid and finds through linear interpolation how the surface intersects each voxels. The Poisson Surface Reconstruction algorithm [KBH06], rather, is based on a basis function belonging to the *indicator functions* family. Given a set of oriented input points, such a function is valued one inside the object and zero outside. The algorithm starts by computing as accurately as possible the indicator function from the input points. Then it reconstructs a triangular mesh by approximating the indicator function of the model and extracting the iso-surface.

In this paper we consider the Ball Pivoting Algorithm (BPA) for its efficiency and robustness: given a sufficiently dense point cloud, the algorithm is guaranteed to reconstruct a surface as close as possible to the original one. Furthermore, it is based on sound theoretical foundations, as it is linked to *Alpha shape*. Indeed, the BPA computes a mesh containing triangles that are a subset of the 2-skeleton of the 3D Delaunay triangulation of the input point cloud. This important feature is key to our purposes.

3. BPA for Range Images

Many surface reconstruction methods, like the original BPA and its implementation [Dig14], do not make assumptions on the structure of point cloud data. As such, they use three-dimensional data structures to make the search for neighbor points faster. In such structures, the 3D space is split into a regular grid of cubic cells. Each cubic cell has a reference to the point subset located in that specific space portion. Differently, other algorithms, such as our BPA extension (hereinafter Range BPA or RBPA), exploit the inherent structure of range images to find quickly neighbor points.

BPA builds the mesh incrementally by *pivoting* a ball of fixed radius on the input point cloud. It starts from a seed triangle and then tries to expand the triangulation. This process continues until all input points have been considered, taking advantage also of point normals to handle cases of missing or noisy data. As shown in Fig. 1, the reconstruction workflow follows two main steps: the first runs a *seed triangle search* by looking for three orphan vertexes such that a ball of a given user-specified radius r touches them without including any other point; the second step runs an *expansion of the triangulation front* by adding one triangle at a time to the mesh by pivoting a ball around *front edges*. When the front is empty a new seed triangle searching is needed.

Therefore, BPA requires fast lookup of the neighborhood of a given 3D point. The original approach [BMR*99, Dig14] implements spatial queries using a regular voxel grid. The user defines as input parameters a single radius or a list of increasing radii that the algorithm will then use to perform the reconstruction. If more than one radius is provided, the algorithm runs one iteration, i.e., *Finding seed triangle* and *Expanding triangulation*, for each radius. Indeed, using multiple radii is useful to fill holes in non-uniform point clouds and to reconstruct a scene at different levels of details.

The BPA implementation proposed in [Dig14] assumes that input point clouds are already endowed with point normals. It uses an octree data structure containing vertexes (a 3D point and its normal) to run neighborhood queries. Therefore, the 3D space is split into cells with size defined starting from the minimum input radius. More precisely, neighborhood queries are implemented through *locational codes* that return the path, starting from the octree root, to a specific leaf cell containing a given point. During our tests we have noticed two main limitations of the BPA implementation. Firstly, a time and computational overhead dependent on the octree depth, due to the traversing of the octree index for each neighborhood query. Secondly, the difficult choice of a suitable radius value, since a small radius tends to generate more holes in the final mesh, whereas a large radius may cause losing fine details. Therefore, our contribution aims at extending the original BPA implementation so as obtain:

- a fast BPA implementation for range images by exploiting adjacencies on the image plane;
- an automatic and robust estimation of the radius.

Normal Computation As above-stated, the algorithm requires a normal at each 3D point. To this aim, we performed the *Principal Component Analysis* (PCA) of the covariance matrix computed at each point neighborhood. Then, we assign to the point normal the eigenvector of the covariance matrix corresponding to the minimum eigenvalue. Moreover, for a more robust estimation we calculate, for each point neighborhood, the median distance m between the central point and its neighbors. Then, to compute the covariance matrix, we consider only the subsets of neighbors whose distance to the central point is less or equal than a multiple of m.

Radii Estimation The BPA implementation [Dig14] deploys a heuristic rule to estimate the radius value:

$$r = \sqrt{\frac{20}{N}S} \tag{1}$$

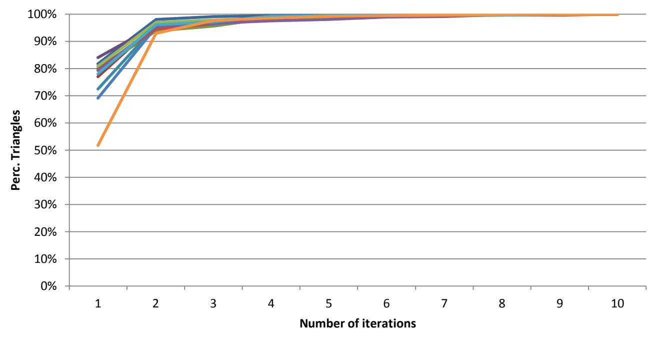

Figure 2: *Percentage of mesh triangles vs number of iterations for different point clouds. Each colored curve corresponds to a different mesh. All meshes reach 90% of triangles after one iteration and above 99% after seven iterations.*

where *20* is the number of points per range neigborhoods, heuristically estimated on a subset of point clouds, N is the number of points and S is the size of the point cloud bounding box. Instead, we propose a more robust and automatic estimation of the radius. First, we measure the minimum and maximum distances between each input point and its neighbors within a defined squared image window. Then, we generate a radius list starting from the global minimum distance found on the whole range image and iteratively adding a *radius step* until we get a value less than or equal to the global maximum distance. We use the global minimum distance value as *radius step* as well. However, such distances should be computed between points lying on the same surface, i.e., points that likely will be eventually triangulated. Therefore, we measure such distances only on the planar regions of the point cloud. To this aim, we compute a score based on the eigenvalues of the covariance matrix of the current point's neighborhood, calculated during the normal computation, and sort these scores for each point in the point cloud. Then, only the points yielding the highest scores are considered for radius estimation. As highlighted in Fig. 2, promising tests on automatic radius estimation suggest the possibility to know in advance the number of radii needed to achieve a given level of detail. Thus, it is possible to define an upper bound on the number of the radii, corresponding to the number of iterations of the algorithm. Above this upper bound we are confident that further iterations would not significantly increase the quality of the mesh.

Neighborhood Queries The octree data structure used in the original BPA resolves neighborhood queries in the 3D space. Instead of building and traversing an octree, we exploit the range image structure by deploying a sliding window approach. Therefore, when looking for new triangles, for each point we select only the neighbor points falling within a squared window centered at such point. Accordingly, we always focus on the subset of points that most likely will form triangles. Instead, the original BPA would always traverse the whole octree index. In practice, we create a matrix of vertexes of the same size as the range image and we use it as data structure to slide our window. Moreover, each vertex stores its relative image coordinates on the range image plane.

Camera Projection Model Since the algorithm works with range images, we have to consider the mapping between a 3D point and its projection onto the range image plane, i.e. we have to take into account the projection model adopted by the camera for the image formation process. There are two cases where this issue arises, that is, (a) accessing the neighborhood of an *edge midpoint*, during the front expansion step, and (b) accessing the neighborhood of a given *triangle circumcenter*, before passing to the next radius when trying to add new front edges. Indeed, in both cases we do not have a direct relationship between a 3D point and a location onto the range image, so that we have to find their corresponding image coordinates to get a suitable reference point on the range image. Generally speaking, we can find the image coordinates of a 3D point by applying *perspective* or *orthogonal* projection.

The *perspective* projection is defined by two equations mapping scene points into image points:

$$\begin{cases} u = x\frac{f}{z} + c_x \\ \\ v = y\frac{f}{z} + c_y \end{cases} \tag{2}$$

(x,y,z) denoting the 3D point coordinates, (u,v) the corresponding image coordinates, f the camera focal lenght and (c_x,c_y) the principal point. In case (a) mentioned above, we wish to find the image point $M_{2\text{D}}$, corresponding to the 3D midpoint $M_{3\text{D}}$, of an edge. We know that the 3D edge midpoint is given by:

$$M_{3\text{D}} = \begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = \begin{cases} \frac{1}{2}\,(x_t + x_s) \\ \frac{1}{2}\,(y_t + y_s) \\ \frac{1}{2}\,(z_t + z_s) \end{cases} \tag{3}$$

where (x_s,y_s,z_s) and (x_t,y_t,z_t) are the 3D coordinates of the edge source and target, respectively. We also know the corresponding image coordinates of such endpoints, i.e. (u_s,v_s) and (u_t,v_t). From (2) we get:

$$\begin{cases} x = (u - c_x)\,\frac{z}{f} \\ y = (v - c_y)\,\frac{z}{f}\,. \end{cases} \tag{4}$$

By replacing (4) in the two first equations of (3)

$$\begin{cases} x_m = \frac{x_t + x_s}{2} \\ y_m = \frac{y_t + y_s}{2} \end{cases} \tag{5}$$

the corresponding projection $M_{2\text{D}}$ can be found as follows:

$$M_{2\text{D}} = \begin{pmatrix} u_m \\ v_m \end{pmatrix} = \begin{cases} \frac{u_s z_s - c_x z_s + u_t z_t - c_x z_t}{2 z_m} + c_x \\ \\ \frac{v_s z_s - c_y z_s + v_t z_t - c_y z_t}{2 z_m} + c_y \end{cases} \tag{6}$$

Similarly, in case (b) we want to find the image point, $C_{2\text{D}}$, corresponding to the 3D circumcenter, $C_{3\text{D}}$, of a generic triangle. We know that the 3D barycentric coordinates of the circumcenter of a generic triangle ABC, with edge lengths a, b and c are:

$$C_{3\text{D}} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \begin{cases} \alpha x_1 + \beta x_2 + \gamma x_3 \\ \alpha y_1 + \beta y_2 + \gamma y_3 \\ \alpha z_1 + \beta z_2 + \gamma z_3 \end{cases} \tag{7}$$

where

$$\begin{cases} \alpha = a^2\left(b^2+c^2-a^2\right) \\ \beta = b^2\left(a^2+c^2-b^2\right) \\ \gamma = c^2\left(a^2+b^2-c^2\right) \end{cases} \tag{8}$$

and (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3) are the 3D coordinates of the triangle vertexes. We also know the image coordinates corresponding to such vertexes, i.e. (u_a,v_a), (u_b,v_b), (u_c,v_c). By replacing (4) in the first two equations of (7)

$$\begin{cases} x_c = \alpha x_1 + \beta x_2 + \gamma x_3 \\ y_c = \alpha y_1 + \beta y_2 + \gamma y_3 \end{cases} \tag{9}$$

the corresponding projection C_{2D} can be found as follows:

$$C_{2D} = \begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{cases} \frac{\alpha(u_a-c_x)z_1+\beta(u_b-c_x)z_2+\gamma(u_c-c_x)z_3}{z_c} + c_x \\ \frac{\alpha(v_a-c_y)z_1+\beta(v_b-c_y)z_2+\gamma(v_c-c_y)z_3}{z_c} + c_y \end{cases} \tag{10}$$

Perspective effects may be not so evident, this occurring whenever the acquired object is much thinner than the distance from the camera. In these cases, perspective projection can be approximated by *orthogonal* projection:

$$\begin{cases} u = x \\ v = y \end{cases} \tag{11}$$

Accordingly, in case (a), the midpoint image coordinates, (u_m, v_m), are equal to the corresponding (x_m, y_m) coordinates of the 3D edge midpoint:

$$M_{2D} = \begin{pmatrix} u_m \\ v_m \end{pmatrix} = \begin{cases} \frac{x_t+x_s}{2} \\ \frac{y_t+y_s}{2} \end{cases} \tag{12}$$

Similarly, in case (b) the image coordinates of the triangle circumcenter, (u_c, v_c), are equal to the corresponding (x_c, y_c) coordinates of the 3D circumcenter:

$$C_{2D} = \begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{cases} \alpha x_1 + \beta x_2 + \gamma x_3 \\ \alpha y_1 + \beta y_2 + \gamma y_3 \end{cases} \tag{13}$$

4. Experimental Results

We compared our proposal against the original BPA implementation, the BPA proposed by *MeshLab* tool [Cig] and *Organized Fast Mesh* (OFM) [HB14], a fast meshing algorithm for range images available in the Point Cloud Library [RC11]. As for the data, we used a publicly available dataset acquired with a Kinect RGB-D sensor [PDS15]. We deployed a 11x11 sliding window to search point neighbors and orthogonal projection formulas to solve the 3D-2D mapping problem. It should be recalled that, for all tests, the RBPA pre-processing time comprises the point normal computation. The BPA time, instead, includes the octree initialization time. We ran three different tests on the same dataset composed of 82 point clouds:

1. first, we compared the BPA and RBPA algorithms using our proposed estimation of the sequence of radii;
2. then, we triangulate again the data using the radius estimated as in [Dig14];
3. then, we compared our RBPA and BPA proposed by *MeshLab*;
4. finally, we compute the meshes through *Organized Fast Mesh* [HB14]

As for the first test, we set the number of iterations to seven in order to get a reconstruction percentage above 99% (cfr. Fig. 2). Tab. 1 and Fig. 3 highlight a lower processing time for our approach compared to BPA, while Fig. 8 shows the meshes reconstructed by RBPA and BPA using our radius estimation algorithm. The level of detail is preserved by our method, though at a reduced computation cost.

Tab. 2 and Fig. 4 refer to the second test session. As shown, the BPA reconstruction time is much larger than RBPA's. This is due to the estimated sequence of radii, which starts from a value greater than the one estimated by our approach, so affecting the octree cell

	RBPA	BPA
Dataset cardinality	82 point clouds	
Average num. pts.	24859 points	
Average Preproc. Time	3.9 s.	3.9 + 0.9 s.
Average Reconstruction Time	0.32 s.	0.94 s.
Max Reconstruction Time	0.98 s.	2.52 s.
Average Number of triangles	48620	48603

Table 1: *BPA and RBPA statistics using our automatic radius estimation.*

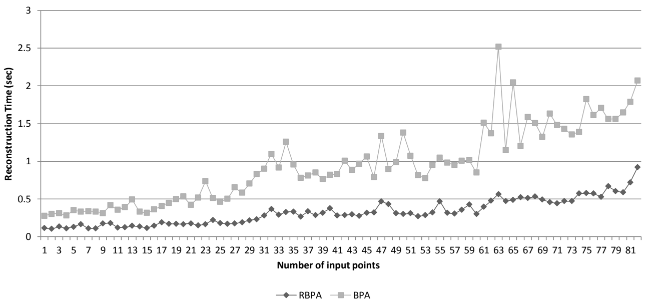

Figure 3: *BPA and RBPA reconstruction time using our radius estimation. The range images (x-axis) are sorted by increasing number of points.*

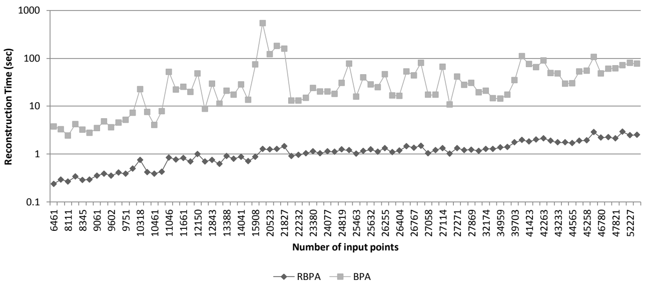

Figure 4: *BPA and RBPA reconstruction time using the radius estimation proposed in [Dig14]. The range images (x-axis) are sorted by increasing number of points.*

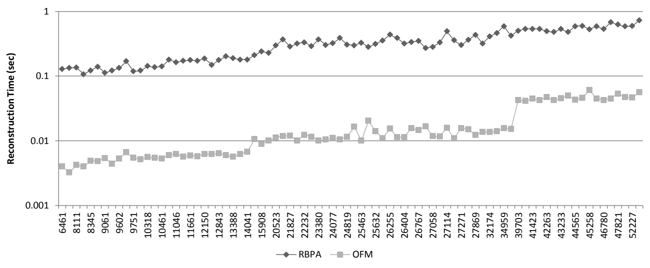

Figure 5: *RBPA and Organized Fast Mesh (OFM) [HB14] reconstruction time. The range images (x-axis) are sorted by increasing number of points.*

	RBPA	BPA
Dataset cardinality	82 point clouds	
Average num. pts.	24859 points	
Average Preproc. Time	3.9 s.	3.9 + 0.9 s.
Average Reconstruction Time	1.18 s.	41.5 s.
Max Reconstruction Time	2.92 s.	540.0 s.
Average Number of triangles	41782	41286

Table 2: *BPA and RBPA statistics using the radius estimation proposed in [Dig14].*

size. As shown in Fig. 9, the radius estimation affects the final mesh quality as well, which lose finer details compared to the mesh we got using our proposed radius estimation algorithm, both for RBPA and BPA.

As for the third test, we compared RBPA against the BPA implementation provided by *MeshLab* [Cig]. This algorithm implementation allows both to specify a radius value and to estimate its value through an autoguessing method. Furthermore, it is possible to define a clustering radius, as a percentage of the ball radius, in order to avoid the creation of too small triangles: if a vertex is found too close to a previous one it is merged with it. We compared RBPA against *MeshLab* BPA using the radius autoguessing both without clustering and with clustering radius fixed to the default value (20% of ball radius). It is important to mention that the computational time of *MeshLab* BPA includes an overhead due to I/O operations. Therefore, we tested RBPA in the same conditions by including the I/O operation time. Tab. 3 and Fig. 6 show that *MeshLab* BPA, with radius autoguessing and clustering, is a little bit faster than RBPA. Tab. 4 and Fig. 7, instead, show that *MeshLab* BPA without radius

	RBPA	MeshLab BPA
Dataset cardinality	82 point clouds	
Average num. pts.	24859 points	
Average Preproc. Time	3.9 s.	3.9 s.
Average Reconstruction Time	0.81 s.	0.72 s.
Max Reconstruction Time	1.75 s.	1.69 s.
Average Number of triangles	48620	40644

Table 3: *RBPA and MeshLab BPA (with clustering) satistics. The reconstruction time includes the I/O overhead.*

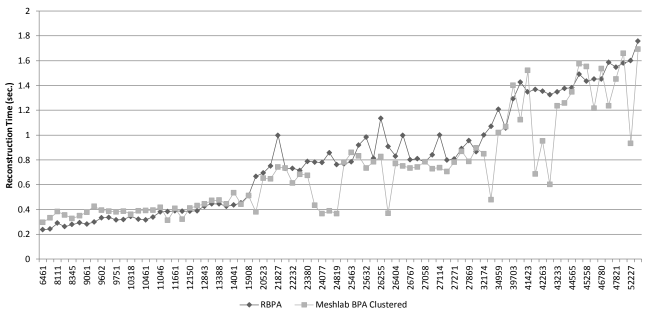

Figure 6: *RBPA (with our radius estimation) and MeshLab BPA (with radius autoguessing and clustering) reconstruction time. The range images (x-axis) are sorted by increasing number of points.*

	RBPA	MeshLab BPA
Dataset cardinality	82 point clouds	
Average num. pts.	24859 points	
Average Preproc. Time	3.9 s.	3.9 s.
Average Reconstruction Time	0.81 s.	7.17 s.
Max Reconstruction Time	1.75 s.	34.2 s.
Average Number of triangles	48620	48446

Table 4: *RBPA and MeshLab BPA (without clustering) satistics. The reconstruction time includes the I/O overhead.*

clustering is much slower than RBPA. Fig. 11 shows reconstructed meshes by *MeshLab* BPA.

Finally, we compared our method to *Organized Fast Mesh* [HB14], which tries to build the best triangle for each point in the range image by considering only its adjacent points. Tab. 5 and Fig. 5 show a faster runtime for *Organized Fast Mesh*, mainly due to the simpler workflow and the much smaller neighborhood size. However, our approach is able to link points even if they are separated by a few invalid range measurements and to cope with clear outliers, as shown in Fig. 10.

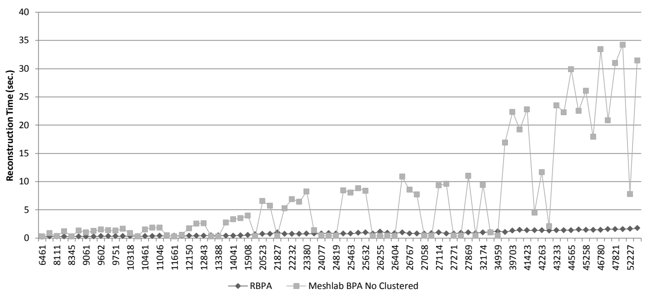

Figure 7: *RBPA (with our radius estimation) and MeshLab BPA (with radius autoguessing and without clustering) reconstruction time. The range images (x-axis) are sorted by increasing number of points.*

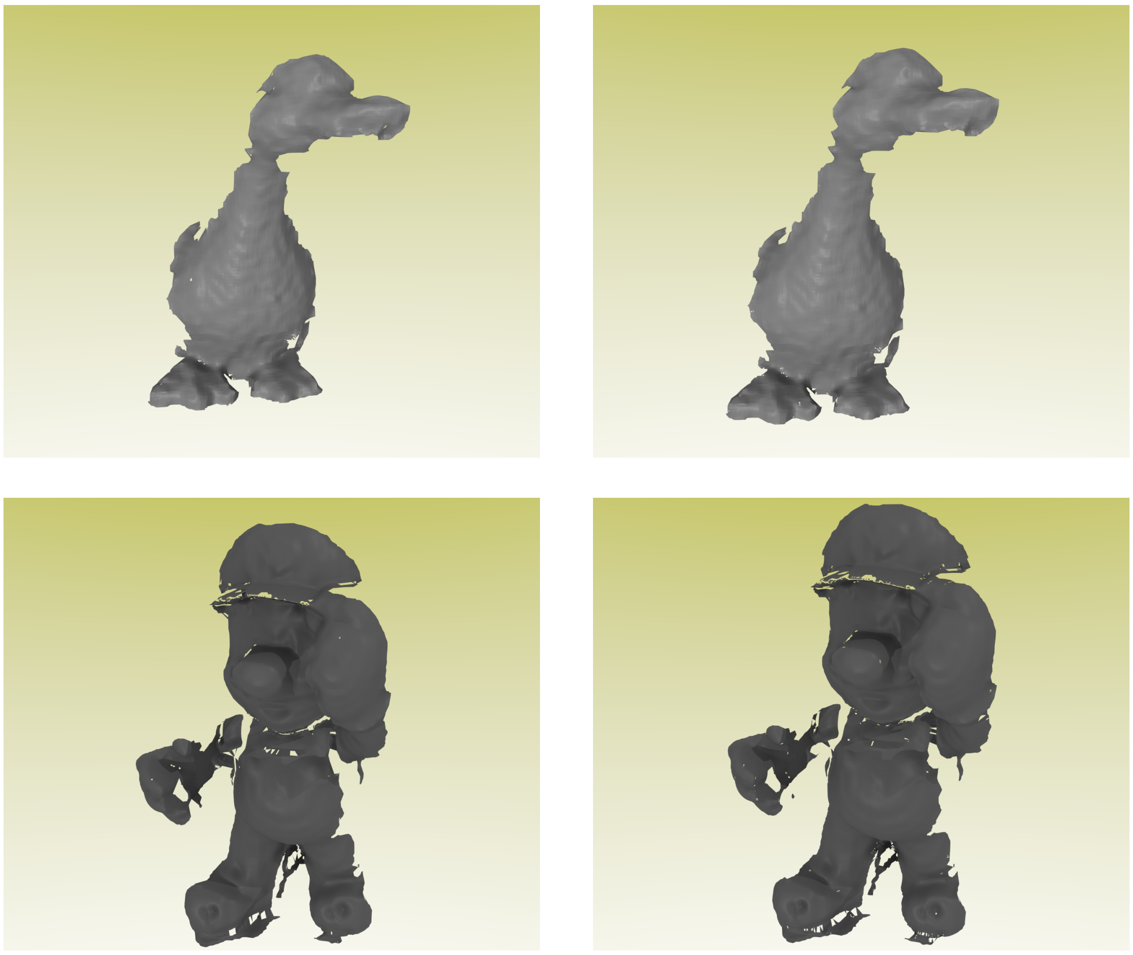

Figure 8: *BPA (left) and RBPA (right) reconstruction using our radius estimation. Note the equivalent level of detail.*

	RBPA	Fast Org. Mesh
Dataset cardinality	82 point clouds	
Average num. pts.	24859 points	
Average Preproc. Time	3.9 s.	-
Average Reconstruction Time	0.32 s.	0.018 s.
Max Reconstruction Time	0.92 s.	0.06 s.
Average Number of triangles	48577	47320

Table 5: *RBPA and Organized Fast Mesh [HB14].*

5. Conclusions

In this work we have investigated a novel BPA implementation suited for range images. We have shown that by exploiting pixel adjacencies on the range image and by considering a constant number of neighbors, our RBPA builds meshes in much less processing time than BPA, while generating a comparable or higher number of triangles. Moreover, our robust and automatic radius estimation algorithm allows for finding the best sequence of radius values only by specifying the number of iterations of the reconstruction algorithm. This allows to prevent to specify a user-defined metric radius value as input parameter. One limitation of our approach is the computational time overhead in case of very large range images since the sliding window scans the image with one pixel step. We fixed the sliding window size in an heuristically manner.

The guidelines for future work should be estimating the best sliding window size. Eventually, we noticed slight differences between meshes reconstructed by orthogonal projection and those yielded by perspective projection. Therefore, another direction for future work concerns assessing and analyzing the mesh differences both quantitatively and qualitatively.

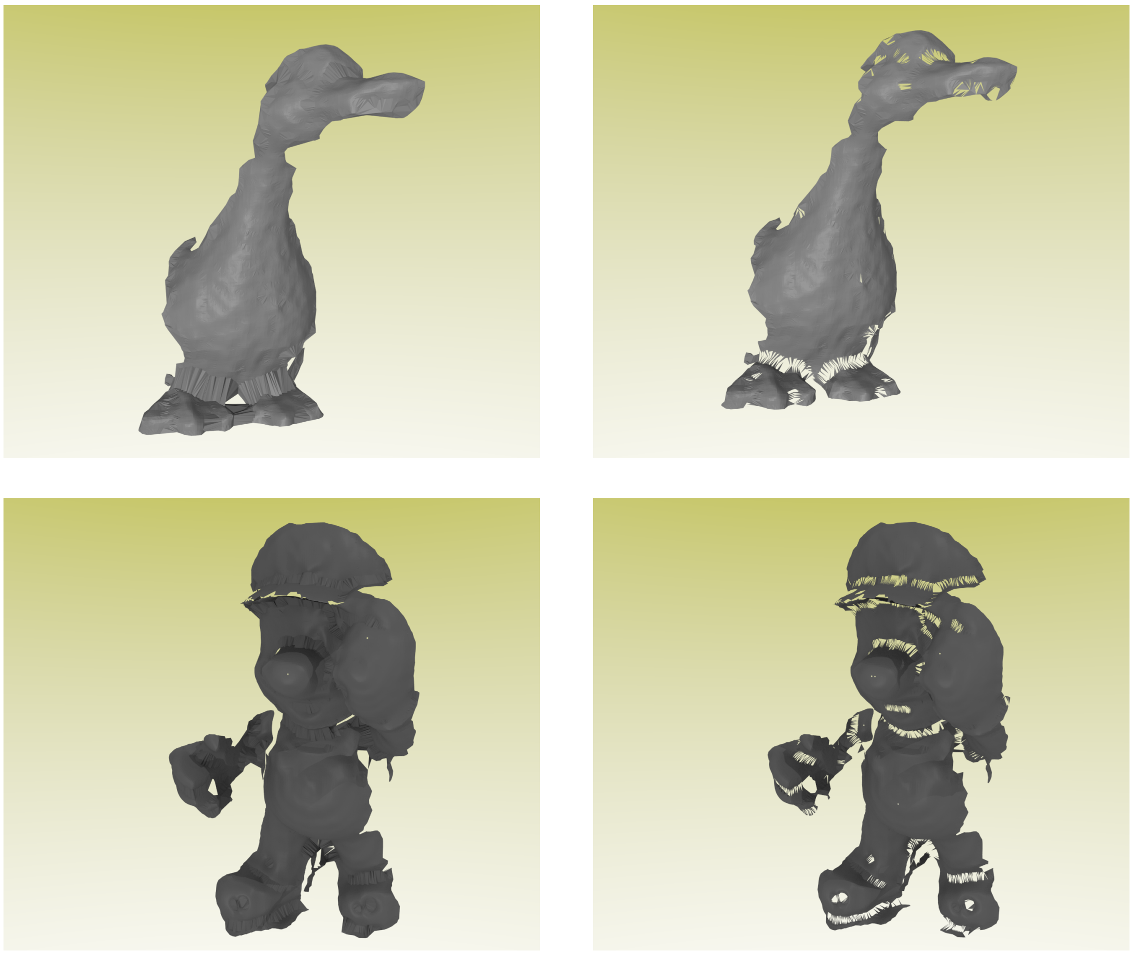

Figure 9: *BPA (left) and RBPA (right) reconstruction using the radius estimation proposed in [Dig14]. Note the lower quality compared to Fig. 8.*

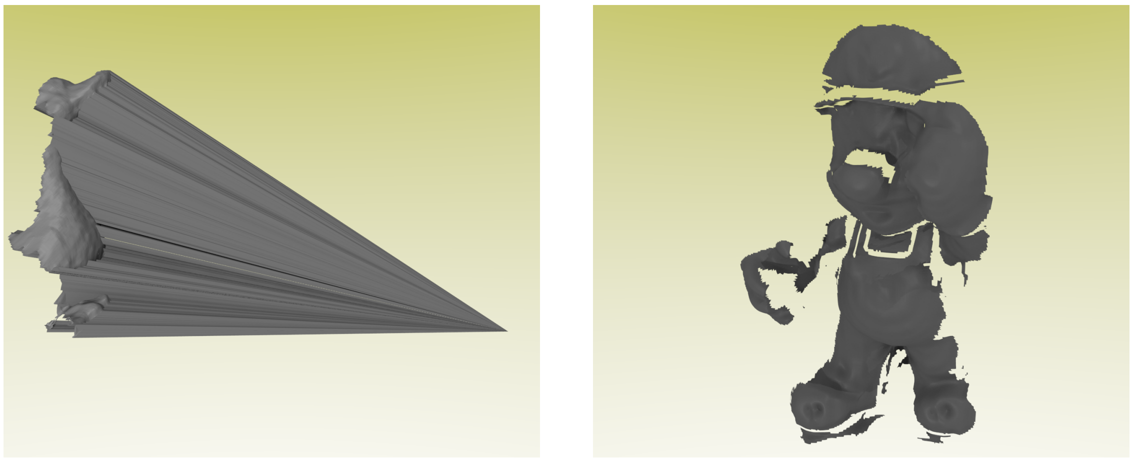

Figure 10: *Reconstruction through Organized Fast Mesh [HB14] reconstruction. On the left, the algorithm fails by linking points to a clear outlier. On the right, the reconstruction quality is quite low, especially on the nose, the right hand and the hat.*

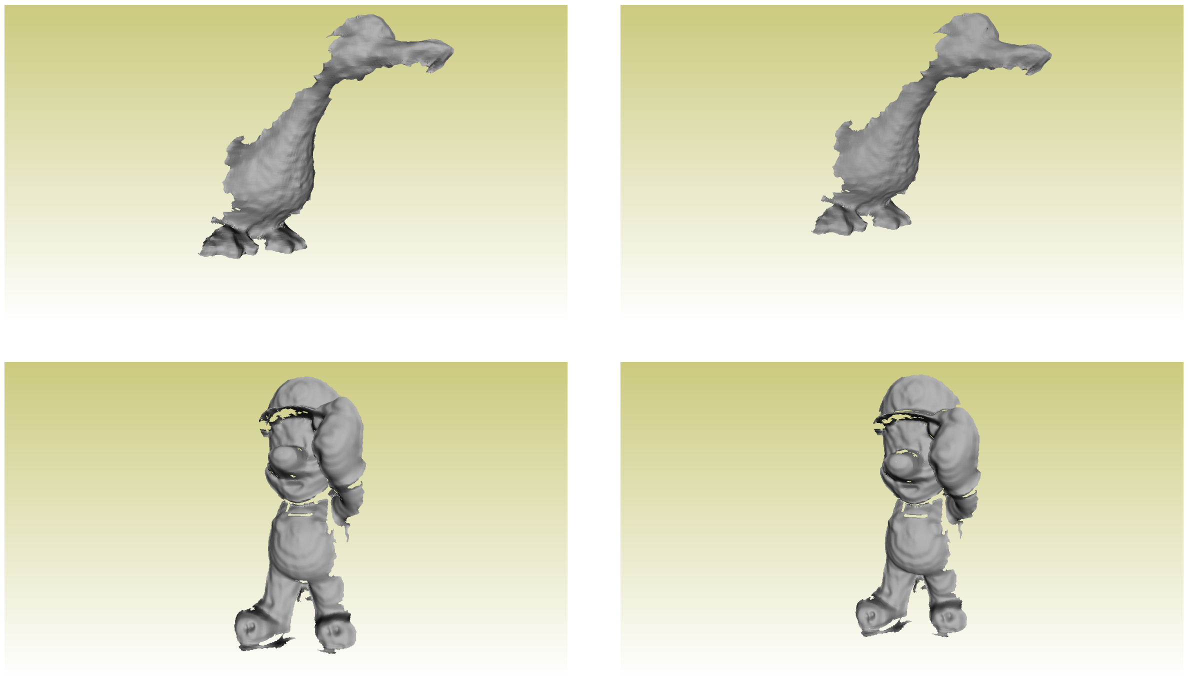

Figure 11: *MeshLab BPA using radius autoguessing with clustering (right) and without clustering (left). Note, although the level of detail is similar to the RBPA reconstructions of Fig. 8, the meshes at the bottom for which MeshLab can't reconstruct part of the original surface (right arm).*

References

[AB98] Amenta N., Bern M.: Surface reconstruction by voronoi filtering. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 1998), SCG '98, ACM, pp. 39–48. 1

[ABK98] Amenta N., Bern M., Kamvysselis M.: A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 415–421. 1

[ACDL00] Amenta N., Choi S., Dey T. K., Leekha N.: A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 2000), SCG '00, ACM, pp. 213–222. 1

[ACK01] Amenta N., Choi S., Kolluri R. K.: The power crust. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications* (New York, NY, USA, 2001), SMA '01, ACM, pp. 249–266. 1

[AFS*11] Agarwal S., Furukawa Y., Snavely N., Simon I., Curless B., Seitz S. M., Szeliski R.: Building rome in a day. *Commun. ACM 54*, 10 (Oct. 2011), 105–112. 1

[BMR*99] Bernardini F., Mittleman J., Rushmeier H., Silva C., Taubin G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (Oct. 1999), 349–359. 1, 2

[Boi84] Boissonnat J.-D.: Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3*, 4 (Oct. 1984), 266–286. 1

[Cig] Cignoni P.: MeshLab. URL: `http://meshlab.sourceforge.net/`. 4, 5

[Dig14] Digne J.: An Analysis and Implementation of a Parallel Ball Pivoting Algorithm. *Image Processing On Line 4* (2014), 149–168. 1, 2, 4, 5, 8

[EDR83] Edelsbrunner H., David K., Raimund S.: On the shape of a set of points in the plane. *IEEE Transactions on information theory 29*, 4 (July 1983), 551–559. 1

[EM94] Edelsbrunner H., Mücke E. P.: Three-dimensional alpha shapes. *ACM Trans. Graph. 13*, 1 (Jan. 1994), 43–72. 1

[HB14] Holz D., Behnke S.: Approximate triangulation and region growing for efficient segmentation and smoothing of range images. *Robot. Auton. Syst. 62*, 9 (Sept. 2014), 1282–1293. 4, 5, 7, 8

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), SGP '06, Eurographics Association, pp. 61–70. 2

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 163–169. 1

[PDS15] Petrelli A., Di Stefano L.: Pairwise registration by local orientation cues. *Computer Graphics Forum* (2015). 4

[RC11] Rusu R. B., Cousins S.: 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)* (Shanghai, China, May 9-13 2011). 4

[Sal10] Salman N.: *From 3D point clouds to feature preserving meshes*. Theses, Université Nice Sophia Antipolis, Dec. 2010. 1

[ZRB09] Zoltan C. M., Radu B. R., Beetz M.: On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (Kobe, Japan, May 12-17 2009). 1