2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Voxel Planes: Rapid Visualization and Meshification of Point Cloud Ensembles

Julian Ryde, Vikas Dhiman and Robert Platt Jr.

Abstract— **Conversion of unorganized point clouds to surface reconstructions is increasingly required in the mobile robotics perception processing pipeline, particularly with the rapid adoption of RGB-D (color and depth) image sensors. Many contemporary methods stem from the work in the computer graphics community in order to handle the point clouds generated by tabletop scanners in a batch-like manner. The requirements for mobile robotics are different and include support for real-time processing, incremental update, localization, mapping, path planning, obstacle avoidance, ray-tracing, terrain traversability assessment, grasping/manipulation and visualization for effective human-robot interaction.**

We carry out a quantitative comparison of Greedy Projection and Marching cubes along with our voxel planes method. The execution speed, error, compression and visualization appearance of these are assessed. Our voxel planes approach first computes the PCA over the points inside a voxel, combining these PCA results across 2x2x2 voxel neighborhoods in a sliding window. Second, the smallest eigenvector and voxel centroid define a plane which is intersected with the voxel to reconstruct the surface patch (3-6 sided convex polygon) within that voxel. By nature of their construction these surface patches tessellate to produce a surface representation of the underlying points.

In experiments on public datasets the voxel planes method is 3 times faster than marching cubes, offers 300 times better compression than Greedy Projection, 10 fold lower error than marching cubes whilst allowing incremental map updates.

I. Introduction

We propose an efficient data structure and corresponding mechanisms for the classification and meshification of unorganized points cloud that are typically produced by range sensors such as laser scanners and image based depth sensors. By meshification we are referring to the whole process of surface reconstruction followed by triangulation. Our application focus is for mobile robotics and generating 3D maps that are an improvement of occupancy grids but like occupancy grids can support incremental update and other applications necessary for mobile robotics. We refer to this representation as voxel planes.

These voxel planes have all the advantages of occupancy grids which are a well tested representation for mobile robotics but render better and contain more information helpful to robotic applications. They are an effective lossy compression technique suitable for containing the pertinent information from ensembles of point cloud data typically collected by mobile robot range sensors.

We refer to a collection of point clouds that can be collectively co-registered to produce a map as an ensemble of point clouds, these are then added to a data structure that we refer to as the map. It is this map that the robot queries for information about the world and updates by aligning and adding new points as they arrive from the sensors. We formulate the mechanism for supporting incremental updates upon this voxel planes map by ensuring that each voxel has sufficient statistics. This means that as new points are observed in each map cell they can be quickly incorporated which is vital for continuous real-time operation. It also means sliding window processing of the voxels can be done efficiently whereby the results from each voxel in a 2x2x2 neighborhood can be combined to produce the result for the overall neighborhood. The sliding window voxel neighborhood analysis along with our meshification procedure based on plane voxel intersection improves the visual rendering. This is not purely an aesthetic consideration but also a practical one important for human robot interaction and appearance based localization. It is easier for humans to interact with robots if they can readily see a rendering of the robots' internal map. A voxel rendering can be difficult to interpret whereas a voxel planes representation is much easier to visually process for the human, Fig. 2 and Fig. 5.

The majority of current methods for representing point clouds in some mesh-like manner do not support incremental update. As new points arrive the entire mesh has to be recalculated. Marching cubes and voxel planes are the only methods that have the ability to incorporate new data points as they arrive without having to recalculate the entire structure. However as is shown in the experiments voxel planes has significant advantages over marching cubes in speed, accuracy and visual appeal.

We compare our proposed approach voxel planes to several existing state of the art methods on publicly available 3D laser scan datasets referred to as *thermolab*, [1] and *thermogauss*, [2]. A quantitative comparison is undertaken of Greedy Projection and Marching cubes. Execution speed, error and compression of these are assessed.

While voxel data structures like occupancy grids, octrees, occupancy lists [3] have proved useful they have some limitations when in comes to the various needs of robotics. These in fact differ from the needs of the graphics community which has done the bulk of the research on generating meshes from unorganized point clouds usually generated by tabletop scanning systems. visualization of the resulting data structure is important as well for robotics applications further considerations to those of the graphics community need to be accommodated. The representation should be amenable to iterative closest point aligning multiple incoming point clouds from a moving platform. The structure should allow

Computer Science and Engineering, SUNY at Buffalo, Buffalo NY, USA
{jryde,vikasdhi,robplatt}@buffalo.edu

the search for grasp affordances and traversability analysis for robot locomotion.

II. Related Work

The graphics community has explored many meshification approaches. However, they have primarily been motivated to produce water tight, high quality meshes, irrespective of the computing power. Such algorithms include the Power Crust [4], Ball Pivoting [5], Spectral Surface Reconstruction [6], Smooth Signed Distance [7] and the Poisson Surface Reconstruction [8], [9].

There are classical meshification algorithms like Marching Cubes [10], Delaunay Triangulation [11] and their extensions like Faster Delaunay triangulation, [12], Marching triangles [13] which follow similar guiding principles.

The benefits of a surface representation for 3D data like point clouds or occupancy grids have been acknowledged in the robotics community particularly in applications like grasping, terrain traversability assessment, mapping and visualization. Wang et al. [14] use full surface reconstruction by Hoppe's algorithm [15] of the object for grasping unknown objects. Surface reconstruction is among the established approaches in terrain traversability analysis. For example, De cubber et al. [16] establish traversability by the distance of the observed ground plane from the expected ground plane. While Soundrapandian et al [17] use feature extraction from image data with a crisp rule based (CRB) classifier to classify terrain's traversability. Vandapel et al. [18] use the principal component analysis (PCA) of points in a neighborhood, to determine the "point-ness", "curve-ness" and "surface-ness" of the neighborhood. These features are fed into a classifier for traversability analysis. With applications to mapping, surface reconstruction has been used by Klass et al. [19]. They create 3D surfel Grid maps by estimating the "surface normals by the eigenvector to the smallest eigenvalue of the 3D sample covariance." Though our approach is similar, we improve on their work taking a sliding window approach over PCA, formulating a mixture of Gaussians update model for the voxel neighborhood, and demonstrating how to generate a good mesh from the voxel planes. We also perform a quantitative comparison of numerous contemporary methods evaluating the accuracy, execution speed and compression factors. Stuckler and Behnke [20] extend surfel representation to multi resolution surfel representation.

III. Background

We compare our approach with Marching Cubes, Greedy projection and Poisson Surface Reconstruction.

A. Marching Cubes

Marching Cubes [10] is a surface reconstruction technique that extracts an isosurface from a scalar field. Most of the time in robotics the input data arrives as point clouds rather than scalar fields. Various transformations can be applied to convert point clouds to a scalar field. For example, for faster computation one can compute the point density in each voxel to create the required scalar field. Another way is to generate a distance field from the points which is slower but generates smoother surfaces. However, both of these methods produce a pair of surfaces with a non-zero error greater than the size of the voxel. In order to create a single surface from Marching Cubes, we can compute the signed distance function from the point cloud, but this needs oriented point clouds. For our experiments, we compute the distance field from the points and run an implementation of Marching Cubes provided by the Visualization Toolkit (VTK) [21] over the resultant scalar field.

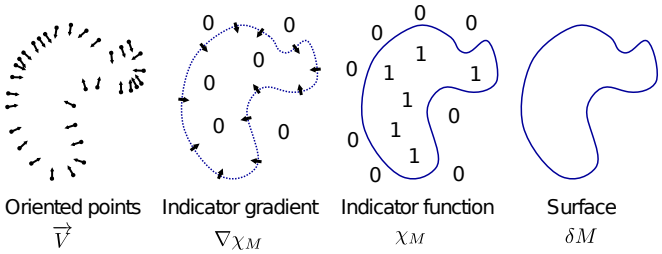

Fig. 1. Intuitive illustration of Poisson reconstruction in 2D (from [8]).

B. Greedy Projection

Marton et al. [22] implemented a fast reconstruction method in the Point Cloud Library [23], that creates a surface by triangulating points after projecting them onto a tangential surface in an adaptive local neighborhood. This method is fast and has mechanisms to ease the addition of new points as the robot explores new spatial scenes. However, greedy triangulation keeps all the points and edges and they are never deleted causing it to be memory and bandwidth inefficient for transmitting over the network.

C. Poisson Surface Reconstruction

Poisson surface reconstruction [8], [9] relies on an implicit function approach for reconstruction. The reconstruction is designed to work with oriented point clouds where each point has an associated normal vector indicating local orientation.

Their main insight is that oriented points, say $\vec{V}$, may be viewed as samples of the gradients of the model's indicator function, say χ_M, (see Fig. 1) and the model's surface is an isosurface over the indicator function. On applying the divergence operator, the problem takes the form of the Poisson problem.

$$\Delta\chi_M \equiv \nabla \cdot \nabla\chi_M = \nabla \cdot \vec{V} \tag{1}$$

The problem is then discretized over the space using octrees, such that each sample point falls into a leaf node at depth D of an octree, where D is a user provided parameter. The solution of discretized poisson equation is then fed into an octree based Marching cubes framework to generate surfaces.

Poisson reconstruction produces smooth, closed, water tight surfaces that are excellent for reconstructing models once we have scans covering the model from all directions. However, in robotics we usually acquire scans of a scene that are not closed, for example a room with doors and windows.

Poisson reconstruction draws closure surfaces around doors and windows which is not good for visualizing scenes in which robots usually operate.

IV. Technical Details

We need to represent point clouds in a compressed format, yet be able to create aesthetically pleasing visualizations for human operators. In order to do this, we discretize the point cloud as voxels, and retain only sufficient statistics of the points within that voxel. At each voxel is stored the number of points (1 integer), centroid of the points (3 floats) and the 3×3 symmetric covariance matrix (6 floats).

This representation can be used to render surface elements (surfels) in the voxels, by the eigenvector corresponding to the smallest eigenvalue. This is considered as the normal to the surface and planar patch can be fitted inside the voxel for visualization. Moreover, this representation supports incremental update as we receive more data points. However, if we follow this approach many spurious surfels arise as shown in Fig. 2.

The sliding window augmentation gives a smoother transition in plane fitting compared to the per voxel approach. It produces a more continuous mesh which is better suited for visualization. The case when a plane of points intersects the corner of a voxel is illustrated in Fig. 3. When analyzed independently the points lying within the corner of a voxel will be poorly fit and classified as insufficiently planer. This problem is alleviated by considering the neighborhood around each voxel vertex via the efficient sliding window implementation.

A. Incremental update and sliding window computation

The per voxel sufficient statistics are used for two primary reasons, to enable the incremental addition of new co-registered points to the data structure and for accelerating the sliding window processing. Rather than considering a collection of all the points in a 2x2x2 neighborhood and performing PCA on those points, instead PCA is performed separately on the points within each voxel and the sufficient statistics stored. The PCA of each 2x2x2 neighborhood is then calculated as the composition of the Gaussian distributions (2) and (3) in each voxel within that neighborhood. Because fewer points need to be considered for each PCA operation and the PCA is not being repeated for each of the 8 sliding windows that include a particular voxel it is much faster.

For a combination of multiple Gaussian distributions the resulting mean $\bar{X}$ and covariance $\operatorname{cov} X$ may be calculated as

$$\bar{X} = \sum_i p_i \bar{X}_i \tag{2}$$

and

$$\operatorname{cov} X = \sum_i p_i \left(\operatorname{cov} X_i + \bar{X}_i \bar{X}_i^\top - \bar{X} \bar{X}^\top \right) \tag{3}$$

where $p_i = N_i / \sum_i N_i$ and N_i is the number of points associated with each normal.

B. Meshification

The underlying representation for each voxel is defined by the number of points, the centroid and the covariance matrix. Now we discuss conversion of this underlying representation into a form suitable for visualization.

Graphics hardware generally expects a mesh for hardware accelerated rendering and so conversion of unorganized points into meshes, a process known as meshification or triangulation, is an active research area in the computer graphics community where they strive for high fidelity and accurate meshes. The requirements for mobile robotics are different and include support for real-time processing, incremental update, localization, mapping, path planning, obstacle avoidance, ray tracing and manipulation.

First voxels are classified into planar and non-planar voxels by inspecting their eigenvalues as follows. Planar voxels must satisfy the constraint on the eigenvalues within their 2x2x2 neighborhood that two are large and one small. The value of the large eigenvalue is ultimately limited by 2x2x2 window. Rather than training a classifier as in [18] we establish a viable threshold based on the meaning of the eigenvalue, the voxel size and the anticipated point error. This point error is predominantly due to noise in our range sensors and registration errors. The eigenvalues are the variances along the corresponding eigenvectors. As such for planar voxels across the sliding window we expect the eigenvalues to satisfy, $\lambda_0 > \lambda_t, \lambda_1 > \lambda_t, \lambda_2 < \lambda_t$ where $\lambda_t = \epsilon^2$ with epsilon the anticipated point noise and in our case ϵ is 0.02m.

Once voxel neighborhoods have been classified then the different elements can be converted to a mesh for accelerated rendering on the GPU. For planar voxels the surrounding 8 voxels (the neighborhood) are considered and the infinite plane is intersected with an inner voxel that surrounds the vertex at the center of the 2x2x2 block. The plane normal is given by the eigenvector with the smallest eigenvalue and the point in the plane is the centroid of the points.

To generate the mesh that corresponds to this plane voxel intersection the intersection points of the plane with each edge of the voxel are determined. For a line defined by $\mathbf{P}_1$, $\mathbf{P}_2$ and plane with normal, $\hat{\mathbf{n}}$ and point $\mathbf{P}_0$ intersection point, $\mathbf{U}$, is

$$\mathbf{U} = \frac{\hat{\mathbf{n}}.(\mathbf{P}_0 - \mathbf{P}_1)}{\hat{\mathbf{n}}.(\mathbf{P}_2 - \mathbf{P}_1)}. \tag{4}$$

This line plane intersection is done for each of the 12 edges of the voxel cube and there can be 0-6 intersection points. The intersection points then have triangles formed between them to flesh out the surface element contained within the voxel. This triangulation is straightforward because the points form a planar convex polygon.

By generating the mesh in this fashion a contiguous surface of tessellating surface elements can be constructed when multiple voxels contain parts of a larger surface. This looks better visually and the resulting composite mesh can be used for ray tracing operations which are often necessary for predicting sensor data given a particular pose.

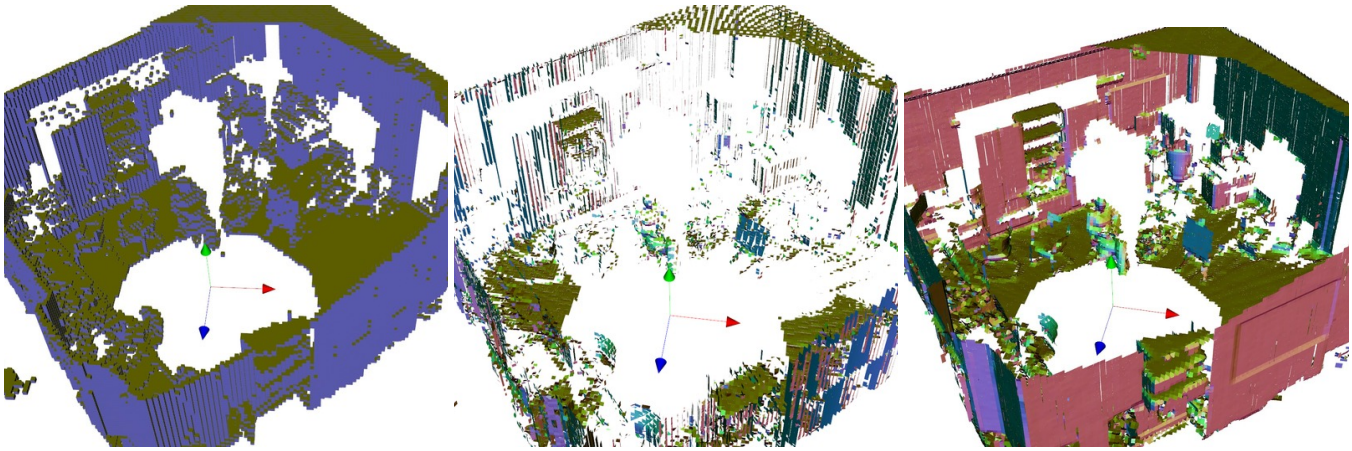

Fig. 2. Example visualization approaches. Visualizing the voxels of an occupancy grid as cubes (Left). Drawing planes within each voxel by estimated surface normal without using sliding window (Center). Final visualization by our method (Right).

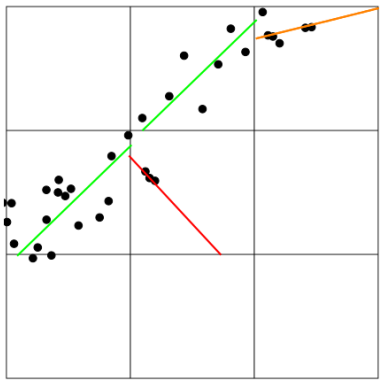

Fig. 3. Illustration of the problem with performing PCA on each voxel independently which is alleviated by the presented modified sliding window approach.

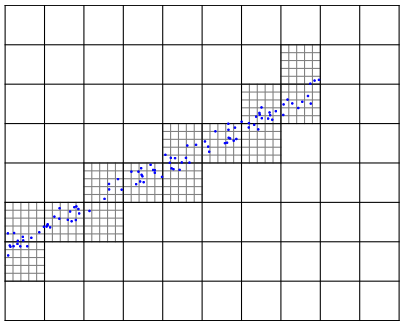

Fig. 4. 2D illustration of the semi-sparse blockwise data structure which enables fast access whilst reducing memory requirements for processing large datasets. It is a hybrid structure consist of a dense coarse (1m) grid of pointers to fine grids (0.05m) for occupied coarse grid voxels.

C. Semi-Sparse blockwise implementation

In practice, voxel plane estimation is performed by quantizing 3D sensor data at a chosen resolution and storing it in a semi-sparse blockwise data structure.

This blockwise structure consists of a coarse resolution (1m) dense grid that contains pointers for non-empty coarse voxels. The pointers at each occupied coarse voxel point to the corresponding fine resolution (0.05m) dense occupancy grid for that coarse voxel. Fig. 4 is a 2D illustration of this. This is similar in ethos to an octree but with only one level of indirection is much faster to access a particular voxel at the cost of an increase in required memory. The memory consumption is vastly reduced compared to a dense occupancy grid at the fine resolution. If occupied voxels were distributed randomly throughout space the semi-sparse approach would offer less memory savings, however due to the inherently clustered nature of typical real 3D data the blockwise data structure is suitable. Often not just a single voxel is required but a group of voxels in a neighborhood need retrieving which can be done quickly.

D. Coloring unoriented planes by normals

To help visualize texture-less meshes the surface elements need to be colored by their normals. To visualize the smoothness of a surface it is vital for the coloring scheme to have two characteristics. Firstly, the colors should be the same for the planes with normal vectors that are opposite in direction as it is impossible to determine the orientation of plane in unoriented point clouds. Secondly, the color space should be continuous, i.e. no sharp changes in color when two planes are closely oriented.

The first condition can be satisfied by flipping the normals that point on one side of a chosen plane (say X-Y plane).

$$\hat{n}' = \begin{cases} -\hat{n} & \text{if } \hat{n}_z < 0 \\ \hat{n} & \text{if } \hat{n}_z \geq 0 \end{cases} \tag{5}$$

This flipping creates a discontinuity in the coloring of normals if each component in the normals is mapped to RGB components, since the domain of normals is just a hemisphere instead of a sphere. To solve this problem, we *stretch* the hemisphere to complete a sphere. This stretching

is done in spherical coordinates. Let (r, θ, ϕ) be the spherical coordinates of flipped normals $\hat{n}'$, then the vector in the stretched sphere is given by spherical coordinates $(r, 2\theta, \phi)$ (where $r = 1$ since $\hat{n}'$ is a unit vector). The coordinates of the resultant vector can be directly mapped to RGB colors (after appropriate scaling and shifting) for a continuous color space over the normals.

$$[\theta, \phi] = \left[\arccos \frac{\hat{n}'_z}{r}, \arctan \frac{\hat{n}'_y}{\hat{n}'_x}\right] \tag{6}$$

$$[R, G, B] = \frac{1}{2}\left[\sin 2\theta \cos \phi + 1, \sin 2\theta \sin \phi + 1, \cos 2\theta + 1\right] \tag{7}$$

V. Experiments

We compare our voxel planes algorithm with three other surface reconstruction algorithms. The chosen algorithms are Greedy Projection [22], Poisson Surface Reconstruction [8], and Marching Cubes [10]. We do only qualitative comparison with Poisson Surface Reconstruction, while we do quantitative comparisons by comparing the remaining algorithms in terms of their CPU runtime and error to representation size. Fig. 6 shows a CPU runtime comparison as we increase the number of points. It is clear from Fig. 6, that Marching cubes has a much higher growth rate than Voxel Planes or Greedy projection. It should be noted that Greedy projection is faster than Voxel Planes but the growth rates of both algorithms are similar. The speed of Greedy projection can be attributed to fast triangulation of points while Voxel Planes depends upon PCA in each voxel of points. Moreover, Greedy projection is implemented in C++, while Voxel Planes is implemented in Python. Fig. 7 plots a comparison in terms of error and representation size. For Robotics requirements like transmitting data over network, the representation with acceptable error and minimum representation size is the best. Hence, results towards the lower left corner of the graph are better.

The error metric in Fig. 7 is computed by cross-validation technique. The input point cloud is divided into test and meshification samples in the ratio 1:9. We use the meshification sample to perform surface reconstruction by the different algorithms and then compute average distance of test points from the generated mesh. The representation size for Voxel Planes is 10 floats per voxel while for Marching Cubes it is 1 float per voxel. For Greedy projection, the representation size is given by 3 floats per point and 3 integers per triangle. For these calculations we assume floats and integers to be 4 bytes each. Also note that we vary the representation size of Marching Cubes and Voxel Planes by changing the voxel size but the Greedy projection representation is changed by sampling a smaller number of points from the dataset. For Voxel planes, the error per points decreases with increase in representation size but, interestingly, after a certain point it starts to increase again. This behaviour can be explained by too few points per voxel at a finer resolution. In a noisy dataset, too few points per voxel cause improper plane fitting and hence greater errors. This can be eliminated by increasing the sliding window size for smaller voxels.

Experiments are undertaken on two datasets. The first the *thermolab* dataset, [1], was recorded with a Riegl VZ-400 Laser Scanner and consists of 8 scans captured inside a building at Jacobs University Bremen. The second the *thermogauss* dataset, [2], an outdoor large-scale urban environment. Visualization of these datasets are displayed in Fig. 5 and Fig. 10 respectively.

It has been observed that for point clouds that come from laser range scanners there is anisotropic sampling on planes that are orientated obliquely to the scanner. This results in columns of points clamped to the same horizontal angle but due to range error they appear to be distributed in a plane aligned with the rays from the range sensor. These point columns give rise to spurious fit planes which are eliminated by the sliding window approach.

VI. Conclusion

If one needs to agglomerate a continuous stream of range data either from range images or range scans in an online manner then most conventional methods for the meshification of unorganized points are unsuitable. Marching cubes and the voxel planes method proposed herein are two options. Whilst the marching cubes algorithm has good performance we show, through experiments on both indoor and outdoor urban environments where flat surfaces are prevalent, that it is significantly outperformed by our voxel planes method. Improvements are observed in terms of execution speed, scalability in the number points and compression ratio.

We have presented a procedure for converting the point clouds from laser and depth sensors that is suitable for robotics applications especially as it is amenable to continuous incremental updates. This novel process starts with storing the number of points, centroid and covariance matrix for each voxel. Then in order to calculate the eigenvalues and vectors of the 2x2x2 voxel region the voxel statistics are combined via a mixture of Gaussians and PCA performed to classify and determine the best fit plane for voxels of the neighborhood. This plane is then intersected with the voxel encompassing the central point of the neighborhood to generate a surface patch which meshes with those of surrounding voxels to assemble a surface suitable for rendering on contemporary graphics processing hardware. Our metrics provide information that allows an informed choice between the various families of unorganized point cloud meshification and surface extraction methods.

Acknowledgements

This material is based upon work supported by the Federal Highway Administration through CUBRC's Cooperative Agreement No. DTFH61-07-H-00023 and NASA under grants NNX12AQ69G and NNX12AM06G. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the FHWA or NASA.

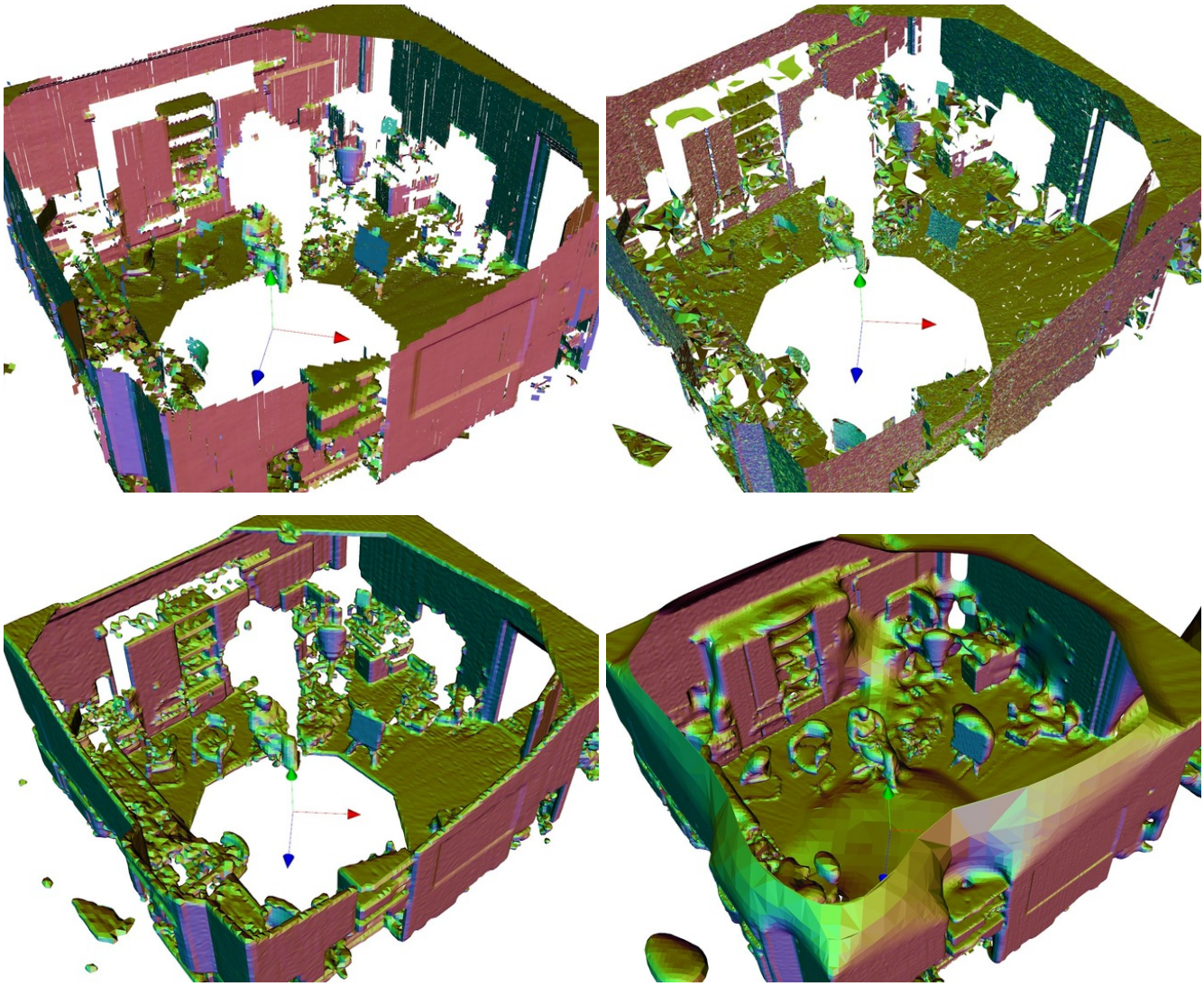

Fig. 5. Visualization of the four meshification methods compared: Voxel Planes (top-left), Greedy Projection (top-right), Marching Cubes (bottom-left) and Poisson (bottom-right). The Poisson visualization has been clipped from above to visualize the interior of the scene as it produces water tight closed surfaces.

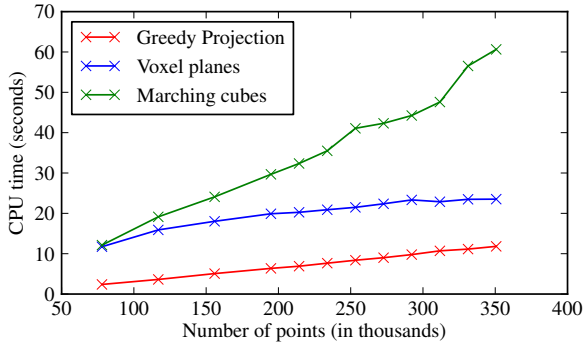

Fig. 6. Runtime comparison of the algorithms on the *thermolab* dataset [1].

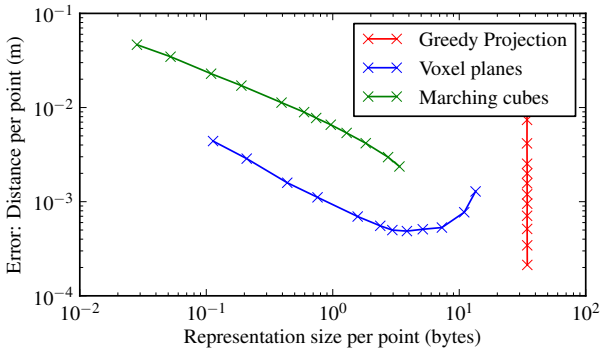

Fig. 7. Graph showing the change in accuracy as a function of representation size for each method on *thermolab* dataset [1]. Lower error at a small representation size, i.e. lower left corner in the plot, is better. To compute error metric we perform surface reconstruction on only 90% of the points while using the remaining points to calculate the average per point distance to the mesh.

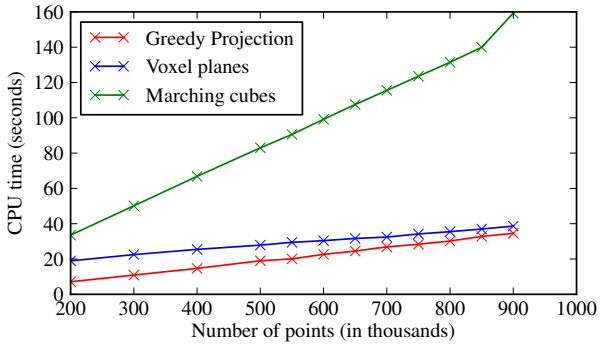

Fig. 8. CPU runtime comparison of different algorithms on *thermogauss* [2] dataset.

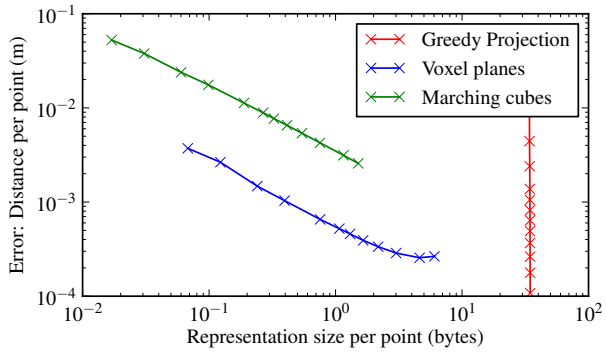

Fig. 9. Error versus representation size comparison on *thermogauss* [2] dataset.

REFERENCES

[1] D. Borrman, "Automation labs, Bremen dataset," 2009. [Online]. Available: http://kos.informatik.uni-osnabrueck.de/3Dscans/

[2] D. Borrmann and A. Nüchter, "Bremen city Guassian point dataset," 2009. [Online]. Available: http://kos.informatik.uni-osnabrueck.de/3Dscans/

[3] J. Ryde and H. Hu, "Mutual localization and 3D mapping by cooperative mobile robots," in *Proceedings of International Conference on Intelligent Autonomous Systems (IAS)*, The University of Tokyo, Tokyo, Japan, Mar. 2006.

[4] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust," in *Proceedings of the sixth ACM symposium on Solid modeling and applications*. ACM, 2001, pp. 249–266.

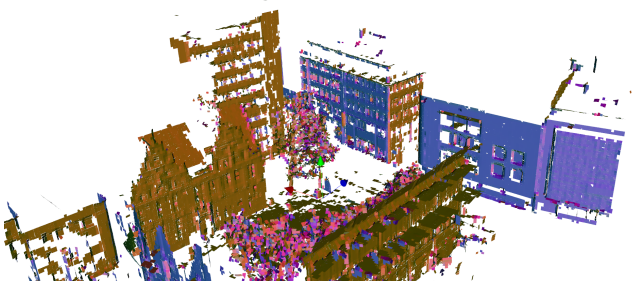

Fig. 10. Visualization of surface reconstruction done by our algorithm on thermogauss dataset.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 4, pp. 349–359, 1999.

[6] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral surface reconstruction from noisy point clouds," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM, 2004, pp. 11–21.

[7] F. Calakli and G. Taubin, "SSD: Smooth signed distance surface reconstruction," in *Computer Graphics Forum*, vol. 30. Wiley Online Library, 2011, pp. 1993–2002.

[8] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[9] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Trans. Graph.*, vol. 32, no. 1, 2013.

[10] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *ACM Siggraph Computer Graphics*, vol. 21. ACM, 1987, pp. 163–169.

[11] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[12] P. Cignoni, C. Montani, and R. Scopigno, "DeWall: A fast divide & conquer delaunay triangulation algorithm in E^d," *Computer-Aided Design*, vol. 30, pp. 333–341, 1998.

[13] A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt, "Marching triangles: Range image fusion for complex object modelling," in *Image Processing, 1996. Proceedings., International Conference on*, vol. 1. IEEE, 1996, pp. 381–384.

[14] B. Wang, L. Jiang, J. Li, and H. Cai, "Grasping unknown objects based on 3D model reconstruction," in *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*. IEEE, 2005, pp. 461–466.

[15] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '92. New York, NY, USA: ACM, 1992, pp. 71–78. [Online]. Available: http://doi.acm.org/10.1145/133994.134011

[16] G. De Cubber, D. Doroftei, H. Sahli, and Y. Baudoin, "Outdoor terrain traversability analysis for robot navigation using a time-of-flight camera," in *Proc. RGB-D Workshop on 3D Perception in Robotics, Vasteras, Sweden*, 2011.

[17] K. SoundraPandian and P. Mathur, "Traversability assessment of terrain for autonomous robot navigation," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 2, 2010.

[18] N. Vandapel, D. F. Huber, A. Kapuria, and M. Hebert, "Natural terrain classification using 3-D ladar data," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 5117–5122.

[19] J. Kläß, J. Stückler, and S. Behnke, "Efficient mobile robot navigation using 3D surfel grid maps," *ROBOTIK 2012*, 2012.

[20] J. Stückler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3d modeling and tracking," in *Journal of Visual Communication and Image Representation*, 2013.

[21] W. J. Schroeder, L. S. Avila, K. M. Martin, W. A. Hoffman, and C. C. Law, "The visualization toolkit-user's guide," *Kitware, Inc*, 2001. [Online]. Available: http://www.vtk.org/

[22] Z. C. Marton, R. B. Rusu, and M. Beetz, "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.

[23] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.