Plane Segmentation and Decimation of Point Clouds for 3D Environment Reconstruction

Lingni Ma, Raphael Favier, Luat Do, Egor Bondarev and Peter H. N. de With
Department of Electrical Engineering, Eindhoven University of Technology
{l.ma, r.raphael, q.l.do, e.bondarev, p.h.n.de.with}@tue.nl

***Abstract*—Three-dimensional (3D) models of environments are a promising technique for serious gaming and professional engineering applications. In this paper, we introduce a fast and memory-efficient system for the reconstruction of large-scale environments based on point clouds. Our main contribution is the emphasis on the data processing of large planes, for which two algorithms have been designed to improve the overall performance of the 3D reconstruction. First, a flatness-based segmentation algorithm is presented for plane detection in point clouds. Second, a quadtree-based algorithm is proposed for decimating the point cloud involved with the segmented plane and consequently improving the efficiency of triangulation. Our experimental results have shown that the proposed system and algorithms have a high efficiency in speed and memory for environment reconstruction. Depending on the amount of planes in the scene, the obtained efficiency gain varies between 20% and 50%.**

***Index Terms*—surface reconstruction, point cloud, triangulation, plane detection, decimation**

I. Introduction

The generation of three-dimensional (3D) models of environments are highly interesting for a number of applications, such as professional civil engineering and serious gaming based on real environments. In our case, we employ a hybrid system for 3D sensing of the environment using laser range finder and high-definition video imaging. The laser range finder produces 3D point clouds that can be used for surface reconstruction. In this paper, we will study the processing of point clouds of large-scale environments and how to extract meaningful semantic information from them to improve the efficiency of 3D reconstruction.

In literature, surface reconstruction using point cloud is a well-studied problem, for which many algorithms have been developed. These algorithms can be coarsely classified into implicit and explicit algorithms. *Implicit* algorithms first estimate a mathematical model for the surfaces and then perform triangulation based on the mathematical description [1]–[3]. These algorithms are robust to noise and always produce smooth meshes. However, such algorithms only apply to surfaces without holes. Therefore, they are not useful for environment reconstruction, where the continuity of the underlying surface is damaged due to uncovered areas from scanning. In contrast to this, *explicit* algorithms reconstruct meshes by connecting points directly. Most explicit algorithms are designed for reconstruction of single object [4], [5] and are less useful for environment. To process large and noisy point clouds, Gopi *et al.* proposed an incremental mesh-growing algorithm, known as Greedy Projection Triangulation (GPT) [6]. This algorithm was further improved by Marton *et al.* [7] with the KdTree searching technique to speed up triangulation. The GPT algorithm performs well for environment reconstruction in both accuracy and computational efficiency. However, GPT triangulates with all input points to preserve geometry, which is not always necessary for environment reconstruction.

In large-scale environment reconstruction, point cloud *segmentation* is a useful preprocessing technique. By recognizing shapes, surface can be better reconstructed according to the corresponding geometry. For environments, flat surfaces are quite common and therefore plane segmentation is likely to be beneficial. Point cloud segmentation is a recent research topic. In literature, a few algorithms extend the 2D graph cut theory towards 3D point cloud segmentation [8]–[10]. These algorithms are designed for general object classification and their complexity is too high for plane detection. A low-complexity algorithm is proposed by Rabbani *et al.* [11], which imposes a smoothness constraint on segmentation. However, this algorithm detects all types of smoothly connected shapes besides planes and therefore plane identification is required as post processing. To find predefined simple shapes (e.g. planes, spheres, cones, etc.), Schnabel *et at.* [12] have developed an algorithm based on RANdom SAmple Consensus (RANSAC). This algorithm can be used for plane detection. However, the RANSAC-based algorithm produces false detections in many situations, because it is designed to find clusters that best fit a plane model without considering the connectivity of points. The computational cost of the RANSAC-based algorithm is also relatively high for large point clouds, since it requires sufficient iterations to guarantee satisfying results.

To avoid massive point-based computations, we also study point cloud *decimation*. By removing the redundancy in points, the requirement for memory can be reduced and the mesh can also be simplified. Such processing also speeds up the related texture mapping and rendering. However, point cloud decimation is a very challenging task. In literature, most decimation algorithms are developed for multi-resolution rendering [13], [14], where points are downsampled to present different level of details. These algorithms are not suitable for environment reconstruction, because of the potential loss of details. Other possible algorithms are based on uniform voxel grid (*i.e.,* a 3D box). Such algorithms move a voxel over point clouds and replace all points inside the voxel with a single point. Obviously, the voxel size is a trade-off between the level of

978-1-4673-3133-3/13/$31.00 ©2013 IEEE

decimation and the maintained details. As a result, the voxel size is always comparable to the point density and therefore limits the performance of these algorithms for environment reconstruction.

Summarizing, the existing segmentation algorithms are not accurate enough for planes and relatively costly, while the decimation algorithms are not well suited for plane-based processing. In this paper, we aim at improving the the accuracy and efficiency of 3D reconstruction system for large-scale environments. Based on the observation that planes usually occupy a significant fraction in an environment, we focus on their optimization by introducing plane-based point cloud segmentation and decimation algorithms. Our full 3D reconstruction system has a hybrid processing architecture, which applies proper triangulation strategies according to different geometrical properties. In particular, we develop a flatness-based algorithm for plane detection and propose a QuadTree-Based (QTB) algorithm for plane decimation and triangulation. The experiment results will show that our proposed algorithms highly decimate point clouds and simplify mesh reconstruction, which is especially beneficial for environments with large flat surfaces.

The sequel of this paper is organized as follows. Section II gives a brief overview of our hybrid 3D reconstruction system for the point cloud processing. Section III describes the flatness-based algorithm for plane detection. Section IV explains the QTB algorithm for optimal plane reconstruction. Section V evaluates our algorithms with experiments. Section VI presents the final conclusions.

II. System Overview

Our proposed system architecture of hybrid point cloud processing is presented in Fig. 1. The processing starts with an unorganized point cloud as the input and generates a complete mesh as output. The processing pipeline consists of three main stages: *point cloud segmentation*, *object reconstruction* and *plane reconstruction*. Let us now briefly describe the function of each block.

1) Point Cloud Segmentation: This block segments point clouds for plane detection. Our early experiments show that large planes (e.g. floors, walls, ceilings, etc.) are quite common in environments. These planes have a simple and well-defined geometry, which allow them to be represented with fewer points and reconstructed with less triangles accordingly. Compared to planes, detailed objects have complex structures, which require a different reconstruction strategy. Therefore, it is reasonable to separate planes and objects and process them with algorithms specifically designed for their geometry. With such geometry-oriented processing, a high-quality mesh reconstruction can be achieved. The separate reconstruction of objects and each plane also enables parallelization, which speeds up the processing of large datasets. In this paper, we propose a flatness-based algorithm for plane segmentation in point clouds, which is further elaborated in Section III.

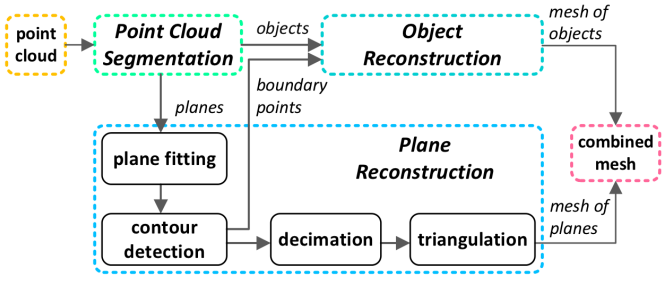


Fig. 1: Our proposed surface reconstruction system with hybrid point cloud processing.

2) Object Reconstruction: This stage reconstructs meshes for objects, which are segments of the input point cloud after plane detection. In order to generate a smoothly-connected mesh between objects and planes, boundary points of planes are also used in object reconstruction. In our system, we apply the GPT algorithm [6], [7] for object reconstruction, because of its accuracy and efficiency in processing large and noisy point clouds.

3) Plane Reconstruction: This stage reconstructs planes. Given a plane segment, a successive processing step is performed to reduce noise, decimate points and reconstruct meshes. A quadtree-based algorithm is developed for this block, which is presented in Section IV.

III. Plane Segmentation in Point Clouds

In this section, we introduce the flatness-based algorithm to segment planes from point clouds with low complexity and high accuracy. Our algorithm is designed for a large unorganized 3D point cloud.

A. Flatness-Based Segmentation Algorithm

Planes are characterized by their perfect flatness, which can be described as points that belong to planes with zero curvature. In practice, point clouds have noise, and points belonging to planes do not always have zero curvature. However, their curvature is still small enough to distinguish them from points belonging to other geometry. This observation motivates the design of a flatness-based algorithm for plane detection. Our work is partially developed from the algorithm in [11]. Two major modifications are made to detect only planes and speeds up the segmentation process.

Our flatness-based algorithm is an iterative process. Initially, the algorithm first estimates the normal of the plane to be detected. This is done by finding a point with the lowest curvature from remaining unsegmented points and uses its normal as the estimation. The algorithm then starts the plane detection in a *region growing* approach and the point is set as the first seed point. In each iteration, the k-nearest neighbors of the seed point are searched and their normals are compared to the estimated plane normal. A neighboring point is added to the current segment, if its normal does not deviate from the plane normal beyond an angle threshold. A qualified neighbor is also used as a new seed point for further region growing, if its curvature is sufficiently small. When no more points can be added to the current segment, a plane is considered

to be found. Afterwards, the whole process restarts with the remaining points, until the entire cloud is processed. The pseudo code of our algorithm is presented in Algorithm 1.

Algorithm 1 flatness-based plane detection in point cloud

Input: unorganized 3D point cloud: each point p_i has a normal $\mathbf{n_i}$ and curvature c_i with i as the indices of points; user-specified parameters: angle threshold θ_{th} and curvature threshold c_{th}.
Output: plane segments

```
mark all points unsegmented and empty the queue
while remain points unsegmented or the queue is not empty do
    if the queue is empty then
        pick a seed point p_s with the lowest curvature
        estimate the plane normal n_p with the normal of p_s
    else
        pop out a seed point from the queue
    end if
    mark the seed point segmented, search its k-nearest neighbors
    for every unsegmented neighbor p_i do
        if arccos(n_p, n_i) < θ_th then
            add p_i to the current segment, mark p_i segmented
            if c_i < c_th then
                add p_i to the queue
            end if
        end if
    end for
    if the queue is empty then
        output the current segment as a plane
    end if
end while
```

There are two major differences between our algorithm and the algorithm in [11]. The first difference is the verification of points into current segment. The original algorithm always updates the normal for verification with the normal of the latest seed point, which may give some limited bending to the detected shape. In our algorithm, we estimate the plane normal with the normal of the first seed point, and use this normal throughout the current plane detection. Since the first seed point has the lowest curvature, its normal is a good estimation for the normal of plane. With this modification, we avoid the detection of other smoothly-connected shapes, such as balls and cylinder-alike structures.

The second modification is involved with the estimation of curvature. The algorithm in [11] uses the residual of plane fitting as a substitute for curvature. In our algorithm, we directly estimate curvature using the original points and therefore enhance the computational efficiency. A necessary preprocessing step for our algorithm is normal estimation for point clouds. This is done by local plane fitting using Principal Component Analysis (PCA) [15]. The PCA for normal estimation also provides a curvature computation using the following equation [16]

$$c = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2},\ \lambda_0 \leq \lambda_1 \leq \lambda_2, \tag{1}$$

where λ_0, λ_1 and λ_2 are the eigenvalues from the PCA. These eigenvalues λ_0, λ_1 and λ_2 indicate the smallest, medium and largest variation along the directions specified by their corresponding eigenvectors. For points belonging to an ideal plane, we have $\lambda_0 = 0$ and hence $c = 0$. In the presence of noise, variable c becomes larger than zero.

The flatness-based algorithm works with two parameters. The first parameter θ_{th} specifies the maximum angle between the estimated plane normal and the normal of a potential point on the plane. Usually, a 10° angle works well for noisy point clouds. The second parameter c_{th} is the curvature threshold, which is used to verify whether a point should be employed as a seed point for region growing. Empirically, this threshold is set to a value below 0.1. Since we focus on processing of large planes, we also impose a threshold on the size of detected plane. The threshold for plane size also prevents over-segmentation.

B. Discussion

Fig. 2 portrays the plane detection with the flatness-based algorithm. For comparison, the result of the RANSAC-based plane detection is also presented. In both algorithms, we detect planes with a size larger than 5% of the total points. It can be seen that planes detected with the RANSAC-based algorithm can contain fractions from different disconnected objects, which are considered to best fit a plane model. However, the detected planes with our algorithm are complete and continuous, which have a physical representation, such as the wall, the floor, the table, etc. These features of our algorithm enables the further processing steps for 3D reconstruction more reliable and reasonable.

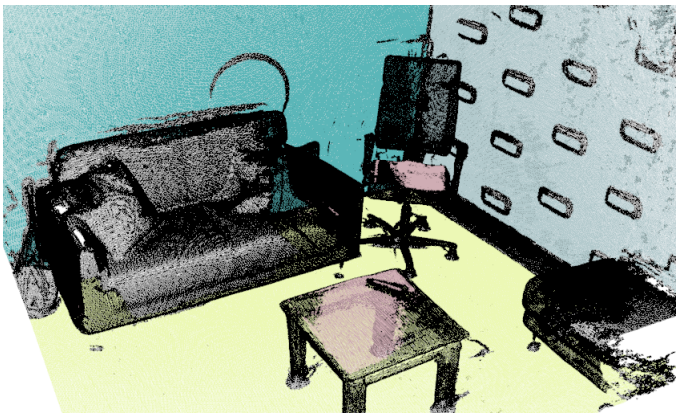
In addition to the accuracy of plane detection, the computational efficiency of our algorithm is also examined. In Table I, the execution times of the flatness-based algorithm and the RANSAC-based algorithm are presented. To ensure satisfying results, 100 iterations are used in RANSAC. The comparison shows that the flatness-based algorithm is in general 2 to 7 times faster in execution than the RANSAC-based algorithm. The computational efficiency is especially beneficial for large point clouds.

IV. Plane Reconstruction

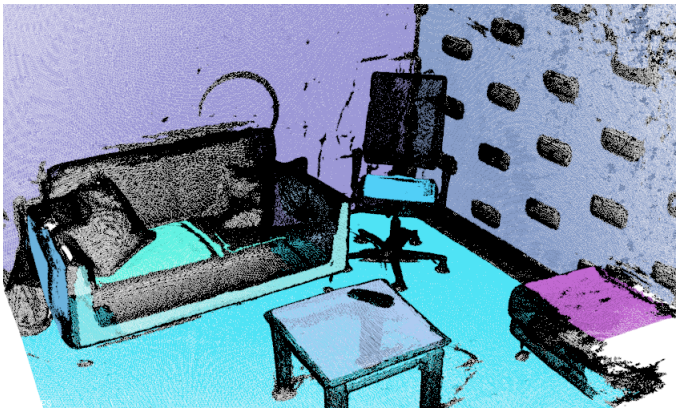
In this section, we propose our QuadTree-Based (QTB) algorithm for plane decimation and reconstruction. We aim for a simplified reconstruction with fewer points and triangles by exploiting the plane geometry. Such optimization is particularly beneficial for environments, where flat surfaces occur frequently. A simplified mesh also yields faster processing at further stages (e.g., texture mapping and rendering). In the remainder of this section, we first motivate the design of the QTB algorithm, then describe its implementation and finally present some comparisons and discussion.

A. Overview of QuadTree-Based Algorithm

The motivation of the QTB algorithm comes from two observations. First, boundary points of a plane contain all necessary information of its shape, while interior points do not carry extra geometrical information and only add redundancy to surface representation. This observation can be illustrated with an example to reconstruct a rectangle. Obviously, it is



(a) result of the RANSAC-based algorithm



(b) result of the flatness-based algorithm

Fig. 2: Plane detection with the RANSAC-based algorithm and our algorithm. Each detected plane is marked with a different color and points marked as black do not belong to any planes.

TABLE I: Comparison of the computational efficiency for the RANSAC-based algorithm and our algorithm.

number of input points	execution time of RANSAC (s)	execution time of our algorithm (s)
74,756	1.35	0.63
288,011	9.60	2.66
650,284	12.22	5.82
1,174,824	20.96	10.30
2,311,672	159.69	20.94

sufficient to create two triangles over its four corner vertices. Adding more points to the interior and creating more triangles with these points does not yield a better reconstruction. Our second observation is that although interior points do not provide valuable geometrical information of the plane, removing all of them also creates a serious difficulty for triangulation of the interior. An example is a plane with holes in its interior. With the complex boundary points, a proper triangulation strategy is difficult to design and will very likely generate skinny or overlapping triangles.

From the above observations, an adaptive decimation strategy is required, which preserves all boundary points and removes most of the interior points. Additionally, the decimation also needs to maintain a well-organized structure for remaining points, as to form a good base for triangulation. To satisfy these requirements, we propose to use *quadtree*, which is a well-studied tree structure developed by Finkel *et al.* [17]. It is a commonly used technique to partition a 2D region for locating objects and interesting points. The term quadtree refers to the division of the target region into four quadrants, where each quadrant can be further split into four if necessary. In our algorithm, the quadtree is applied to locate boundary points and adaptively generate grids for decimation and triangulation.

B. Preprocessing

Two preprocessing steps are required for the QTB algorithm to perform. First, noise in the points forming a plane is reduced to generate an ideal plane. Second, boundary points are detected for plane decimation.

1) Least-Squares Plane Fitting: Due to various types of noise, many points deviate from the real surfaces. This small fluctuation from real location is especially visible for planes, where their flatness becomes deteriorated. To reduce the noise of the plane, we formulate the plane equation as a least-squares optimization problem and project all points according to the equation afterwards. With this preprocessing, the flatness of the plane is restored.

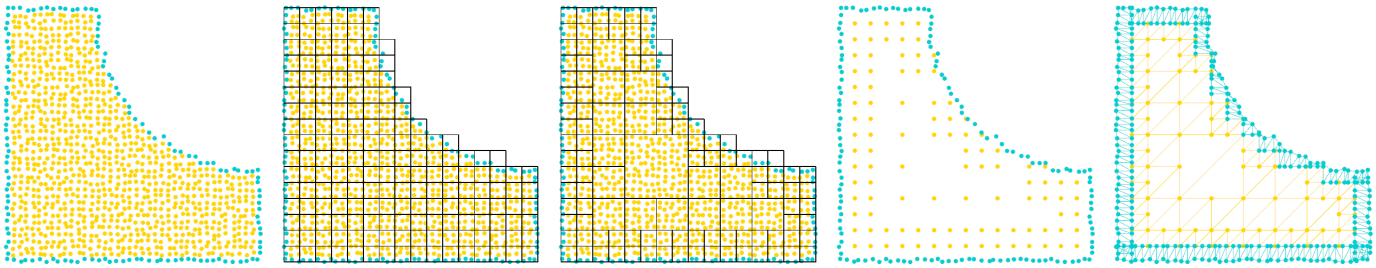
2) Boundary Detection: Detection of boundary points is crucial for the QTB algorithm, because these points contain all the necessary information to preserve the geometry. Boundary detection is also important for the overall reconstruction in our system. Since we reconstruct planes and objects separately, it is important to use boundary points in the triangulation of both planes and objects, such that a smoothly-connected mesh can be obtained.

In this paper, we use a concave-hull-based algorithm for boundary detection. Given a set of 2D points, their boundary can be well described by its concave hull. Since a 3D plane is an analogy of a 2D point set, the concave hull can also be used to detect the boundary of a 3D plane. Our implementation of the concave hull is based on the Point Cloud Library (PCL) [18], which gives a single detection for each boundary point.

C. QTB Algorithm

The QTB algorithm takes a plane segment with indices of its boundary points as input and generates a triangular mesh as output. The processing is a four-stage pipeline as shown in Fig. 3. First, a quadtree is initialized by regularly partitioning the plane. Second, the quadtree is optimized adaptively to create cells with variable size, which is small along boundary and large for interior. Third, the plane is decimated based on the quadtree grid. Finally, triangulation is performed for plane reconstruction. Let us now describe each stage in detail.

1) Initialization of quadtree: In this step the bounding box of the input plane is partitioned into small cells of uniform size. This is done by adding nodes to the quadtree using all points on the plane. When a point is considered, the bounding



(a) input plane (b) initialize quadtree (c) optimize quadtree (d) decimate plane (e) triangulate

Fig. 3: Incremental reconstruction with QTB algorithm. Boundary and interior points are marked blue and yellow, respectively.

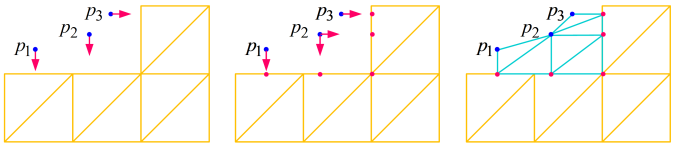
box of the plane is divided into four quadrants and the sub-quadrant, which encompasses the point is again split into four. The decomposition continues until the maximum tree depth is reached. The smallest cell that encloses the point is added as a new node to the quadtree, if it is not created earlier. The result of initialization is shown in Fig. 3b, where the bounding box of the plane is split into equally small cells. Each cell contains at least one point and no empty cell is created.

During initialization, every cell of the quadtree is classified into either *boundary* node or *interior* node. A *boundary* node represents a cell that contains at least one boundary point and an *interior* node represents a cell that contains only interior points. This node classification plays a crucial role in the following steps.

2) Optimization of quadtree: Optimization of the quadtree adaptively generates variable grids, which is one of the major advantages between our algorithm and voxel-based algorithms. During optimization, interior cells are enlarged by recursively merging *interior* nodes with only interior siblings. The merged nodes are also classified as *interior*, which allows them to be merged further. The optimization result of the quadtree in Fig. 3b is presented in Fig. 3c. It is shown that with optimization, the quadtree has large cells for the interior and gradually reduced cells when approaching to the boundary. Such adaptive grids benefit point decimation and also provide a good base for triangulation.

From Fig. 3c, it can be noticed that some neighboring cells can be further merged to generate a larger cell. However, further merging only provides a slight improvement for decimation, and only few cells can be further merged. Moreover, the computational cost to merge nodes from different parents is much higher than merging nodes from the same parent. The former requires searching throughout the quadtree to find possible merging, while the latter can be efficiently done with a recursive implementation. Given the high cost and minor improvement, we suggest further merging is not necessary.

3) Decimation: Based on the optimized quadtree and the classification of nodes, decimation is a simple process. For the *boundary* node, all boundary points inside the cell are preserved, while all interior points are removed. For the *interior* node, all points inside the cell are replaced with the four corner points of the cell. We do not represent the interior cell with its centroid, because the four corners of a cell give a better base for triangulation. The decimated plane of Fig. 3a is shown in Fig. 3d, where all boundary points are preserved and the number of interior points are reduced from 942 to 106, yielding a substantial 88.7% reduction.



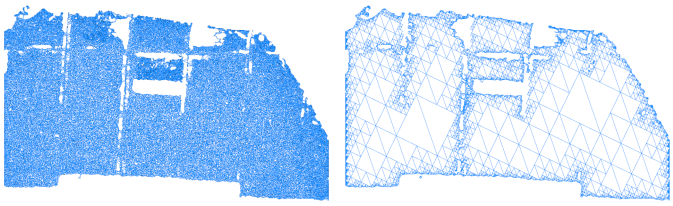
(a) boundary points with projection vectors (b) interpolate points for triangulation (c) add new triangles

Fig. 4: Triangulation for boundary points, where triangles are created to connect the existing interior mesh (yellow)

4) Triangulation: Finally, triangulation is performed for mesh reconstruction. In the QTB algorithm, triangulation for interior points and boundary points are processed differently. We first reconstruct the mesh for interior points, where two triangles are created for each interior cell based on its four corners. We then reconstruct the mesh for boundary points with the principle that boundary triangles need to be connected to the existing mesh. For this purpose, a projection vector is estimated for each boundary point, which indicates the direction of the interior region. This is done by analyzing the local interior points around boundary points and further classifying the projection vector into one of the four directions: up, down, left and right. Fig. 4a shows an example of three boundary points and their projection vectors. With projection vectors, mesh is reconstructed by connecting adjacent boundary points, where two possible situations can happen. For the first situation, the two adjacent points have the same projection vector. In this case, two new points are interpolated by projecting the boundary points along the projection vector onto the existing mesh. Afterwards, two triangles are connected with the boundary points and their projections. This scenario is demonstrated with point p_1 and p_2 in Fig. 4. For the second situation, the two adjacent points have different projection vectors that differs for 90°. In this case, one boundary point is projected along both projection vectors and the other point is projected only along its own projection vector. With the interpolated points, four triangles are created accordingly, as demonstrated with point p_2 and p_3 in Fig. 4.

TABLE II: Plane simplification with our QTB algorithm, in comparison to the mesh reconstruction using GPT algorithm.

no. of input points	no. of points with decimation	percentage of reduction	no. of triangles with GPT	no. of triangles with QTB	percentage of reduction	execution time with GPT (s)	execution time with QTB (s)
5,409	966	82.14%	6,155	998	83.79%	0.15	0.04
8,153	1,552	81.33%	13,997	1,534	89.04%	0.26	0.07
13,157	1,631	87.60%	20,434	1,690	91.73%	0.41	0.10
29,917	2,058	93.13%	50,687	2,136	95.79%	0.98	0.26
116,027	7,212	93.50%	177,032	7,436	95.80%	3.42	1.19
148,438	10,943	92.63%	245,611	11,294	95.41%	4.92	1.56
385,755	12,360	96.80%	546,901	12,624	97.70%	11.45	3.91



(a) GPT with 666,50 triangles (b) QTB with 3,198 triangles

Fig. 5: Triangulation comparison of GPT and QTB algorithm.

The mesh reconstruction for the decimated plane in Fig. 3d is presented in Fig. 3e. It shows that the mesh of interior points maintains the structure of the optimized quadtree. The mesh of the boundary points fully preserves the shape of the input plane. Few boundary triangles overlap with each other. However, this is acceptable for plane reconstruction.

D. Discussion

The result of mesh reconstruction with the QTB algorithm is shown in Fig. 5. For comparison, we apply the GPT algorithm to the same plane. The result shows that QTB generates large regular triangles for the interior of plane and refined triangles for the boundary. As a result, the same plane is reconstructed with much fewer triangles, while the plane shape is preserved. To further examine the simplification of plane reconstruction, some statistics are given in Table II. In the experiment, we compare the number of points before and after decimation with the QTB algorithm. In addition, we also compare the number of triangles and the execution time using QTB and GPT, respectively. It can be seen that the reduction ratio of both points and triangles is remarkable. The comparison of the execution time also demonstrates the computational efficiency of the QTB algorithm.

V. System Evaluations

In this section, we evaluate the overall performance of our system. We have developed our system on Linux Ubuntu 11.10, and all measurements are performed on an Intel Core i5 CPU 750 of 2.67GHz with 3.8GB memory. For the following experiments, only planes with more than 5,000 points are processed by the QTB algorithm for simplification.

The surface reconstruction results for some environment scans are shown in Fig. 6. It can be observed that our system reconstructs regular and simple meshes for large planes and refined meshes for objects with complex details. The geometry of both planes and objects is well preserved. Despite the separate reconstruction, the combined mesh is smoothly connected between objects and planes.

In the second set of experiments, we explore the possibilities for simplifying both point clouds and meshes with our system. For comparison, we have also implemented a system which only uses GPT for reconstruction. Seven data sets of different environments are tested and the results are presented in Table III. The percentage of reduced points is calculated by comparing the number of points before and after decimation. The percentage of removed triangles is computed by comparing meshes generated with our system and with the simple system using only GPT for processing. The statistics show that planes occupy large areas in most environments, and by applying our optimization these environments can be well reconstructed with fewer points and fewer triangles. The more points belonging to large planes, the higher reduction that is achieved.

Besides reduction of points and triangles, we also examine the computational cost of our system. The execution time for each processing block of the system is evaluated. The total execution time is also compared to a simple system that only uses the GPT algorithm for reconstruction. The experiment is performed with the same datasets as used in the second experiment, and the results are shown in Table IV. It can be seen that although our system contains more processing stages, the overall complexity is comparable to a simple system. Since the GPT algorithm is of $\mathcal{O}(n)$ complexity, we can estimate the same complexity for our proposed system. The computational efficiency of our system comes from plane simplification with the QTB algorithm. Therefore, for datasets with a small amount of planes, our system is less efficient than a GPT-only system, which is shown with the last dataset. It should be noted that in testing of the computational efficiency, we reconstruct objects and each plane in serial processing. Since our system architecture allows parallel reconstruction, the efficiency of our system can be further improved.

VI. Conclusion

In this paper, we have developed a hybrid system for point cloud processing to optimize 3D surface reconstruction of large-scale environments. For a high-performance reconstruction, we have proposed to segment point clouds into large

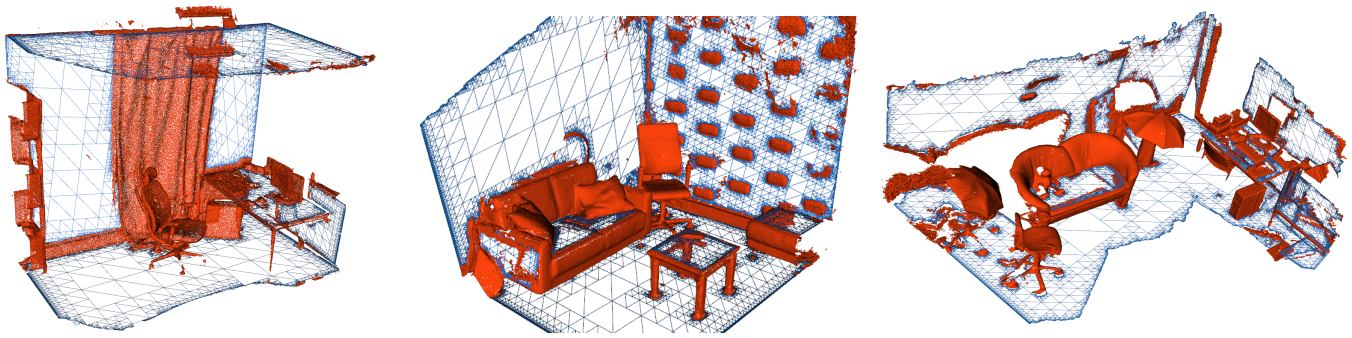


Fig. 6: Mesh reconstruction with our proposed system.

TABLE III: Efficiency improvement with our system, expressed with point reduction and mesh simplification, in comparison to a system using only GPT for reconstruction.

number of points	points belonging to planes	percentage of removed points	percentage of reduced triangles
288,011	31.4%	27.3%	33.2%
650,284	53.2%	47.2%	50.1%
936,469	44.1%	41.2%	48.2%
1,134,650	50.9%	48.6%	53.6%
1,514,722	29.1%	26.2%	30.4%
2,048,856	37.4%	33.4%	39.9%
3,205,086	20.8%	17.8%	20.6%

TABLE IV: Execution time with our system, in comparison to a system using only GPT for reconstruction.

no. of points	segment (s)	reconstruct objects (s)	reconstruct planes (s)	total time (s)	only with GPT (s)
288,011	2.68	6.20	0.81	6.69	9.37
650,284	5.77	9.64	4.02	19.43	21.01
936,469	8.75	17.82	4.06	30.63	33.50
1,134,650	10.41	17.05	5.79	33.25	36.17
1,514,722	14.61	33.79	4.37	52.77	48.63
2,048,856	19.51	40.88	8.47	68.68	68.72
3,205,086	32.29	80.85	6.90	120.04	104.31

planes and detailed objects and then apply proper triangulation strategies according to the geometrical properties of underlying targets. Our first contribution is a flatness-based algorithm for plane detection in point clouds, which features in a good initialization of the normal of the plane to be detected and a computationally efficient curvature estimation. Our second contribution is a QuadTree-Based algorithm for plane decimation and reconstruction, which yields more than 80% reduction for the used points and reconstructed triangles. The key to this algorithm is that it emphasizes the boundary points of planes to preserve geometry, while it also highly reduces the interior points to simplify mesh. The overall evaluation shows that our proposed system with the new algorithms has an efficiency gain varying between 20% to 50% depending on the amount of planes in the scene.

REFERENCES

[1] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proc. of the 4th Eurographics Symposium on Geometry Processing.*, 2006, pp. 61–70.

[2] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe, "Parallel poisson surface reconstruction," in *Proc. of the 5th Inter. Symposium on Advances in Visual Computing*, 2009, pp. 678–689.

[3] A. C. Jalba and J. B. T. M. Roerdink, "Efficient surface reconstruction from noisy data using regularized membrane potentials," *IEEE Trans. on Image Processing*, vol. 18, no. 5, pp. 1119–1134, May 2009.

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

[5] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust moving least-squares fitting with sharp features," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 544–552, Jul. 2005.

[6] M. Gopi and S. Krishnan, "A fast and efficient projection-based approach for surface reconstruction," in *Proc. Computer Graphics and Image Processing.*, 2002, pp. 179–186.

[7] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *Proc. IEEE Inter. Conf. Robotics and Automation ICRA '09*, 2009, pp. 3218–3223.

[8] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," *ICCV*, Sep. 2009.

[9] A. Golovinskiy and T. Funkhouser, "Min-cut based segmentation of point clouds," in *IEEE Workshop on Search in 3D and Video (S3DV) at ICCV*, Sep. 2009.

[10] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *Proc. of IEEE/RSJ IROS*, Oct. 2010.

[11] T. Rabbani, F. van Den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," *Inter. Archives of Photo. Remote Sensing and Spatial Info. Sciences*, vol. 36, no. 5, pp. 1–6, 2006.

[12] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection," *Computer Graphics Forum*, vol. 26, no. 2, pp. 214–226, Jun. 2007.

[13] S.-B. Park, S. U. Lee, and H. Choi, "Multiscale surface representation and rendering for point clouds." in *ICIP*, 2004, pp. 1939–1942.

[14] F. Duranleau, P. Beaudoin, and P. Poulin, "Multiresolution point-set surfaces," in *Proc. of Graphics Interface 2008*, ser. GI '08. Canadian Information Processing Society, 2008, pp. 211–218.

[15] N. J. Mitra, A. Nguyen, and L. Guibas, "Estimating surface normals in noisy point cloud data," in *special issue of International Journal of Computational Geometry and Applications*, vol. 14, no. 4–5, 2004, pp. 261–276.

[16] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proc. of the Conference on Visualization*, ser. VIS '02. IEEE Computer Society, 2002, pp. 163–170.

[17] R. A. Finkel and J. L. Bentley, "Quad trees: A data structure for retrieval on composite keys," *Acta Inf.*, vol. 4, pp. 1–9, 1974.

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