Planar Simplification and Texturing of Dense Point Cloud Maps

Lingni Ma[1], Thomas Whelan[2], Egor Bondarev[1], Peter H. N. de With[1] and John McDonald[2]

***Abstract*— Dense RGB-D based SLAM techniques and high-fidelity LIDAR scanners are examples from an abundant set of systems capable of providing multi-million point datasets. These large datasets quickly become difficult to process and work with due to the sheer volume of data, which typically contains significant redundant information, such as the representation of planar surfaces with hundreds of thousands of points. In order to exploit the richness of information provided by dense methods in real-time robotics, techniques are required to reduce the inherent redundancy of the data. In this paper we present a method for efficient triangulation and texturing of planar surfaces in large point clouds. Experimental results show that our algorithm removes more than 90% of the input planar points, leading to a triangulation with only 10% of the original amount of triangles per planar segment, improving upon an existing planar simplification algorithm. Despite the large reduction in vertex count, the principal geometric features of each segment are well preserved. In addition to this, our texture generation algorithm preserves all color information contained within planar segments, resulting in a visually appealing and geometrically accurate simplified representation.**

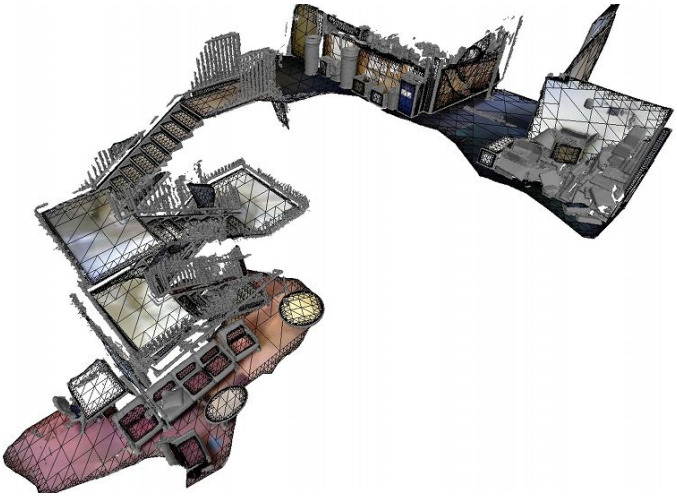


Fig. 1: Scene triangulation showing a simplified mesh for planar segments with non-planar features highlighted.

I. Introduction

The generation of 3D models of real-world environments is of high interest to many application fields including professional civil engineering, environment-based game design, 3D printing and robotics. Industrial Light Detection And Ranging (LIDAR) platforms and extended scale RGB-D mapping systems can output dense high-quality point clouds, spanning large areas that contain millions of points [1], [2]. Key issues with such large-scale multi-million point datasets include difficulties in processing the data within reasonable time and a high memory requirement. In addition to this, some features of real-world maps, such as walls and floors, end up being over-represented by thousands of points when they could be more efficiently and intelligently represented with geometric primitives. In particular the use of geometric primitives to represent a large 3D map to localise against is emerging as a feasible means of robot localisation [3]. In this paper, we examine the problem of planar surface simplification in large-scale point clouds with a focus on quality and computational efficiency.

II. Related Work

In the literature triangular meshing of 3D point clouds is a well-studied problem with many existing solutions. One class of triangulation algorithms computes a mathematical model prior to triangulation to ensure a smooth mesh while being robust to noise [4], [5]. This type of algorithm assumes surfaces are continuous without holes, which is usually not the case in open scene scans or maps acquired with typical robotic sensors. Another class of algorithms connects points directly, mostly being optimized for high-quality point clouds with low noise and uniform density. While these algorithms retain fine details in objects [6], [7], they are again less applicable to noisy datasets captured with an RGB-D or LIDAR sensor, where occlusions create large discontinuities.

With real-world environment triangulation in mind, the Greedy Projection Triangulation (GPT) algorithm has been developed [8], [9]. The algorithm creates triangles in an incremental mesh-growing approach, yielding fast and accurate triangulations. However, the GPT algorithm keeps all available points to preserve geometry, which is not always necessary for point clouds containing surfaces that are easily approximated by geometric primitives. To solve this problem a hybrid triangulation method was developed in [10], where point clouds are segmented into planar and non-planar regions for separate triangulation. The QuadTree-Based (QTB) algorithm was developed to decimate planar segments prior to triangulation. The QTB algorithm significantly reduces the amount of redundant points, although a number of limitations degrade its performance. For example, the algorithm does not guarantee that final planar points will lie inside the original planar region, which can lead to noticeable shape distortion. The algorithm also produces duplicate vertices, overlapping triangles and artificial holes along the boundary.

To summarize, existing triangulation algorithms perform

[1]Department of Electrical Engineering, Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands. l.ma at tue.nl

[2]Department of Computer Science, National University of Ireland Maynooth, Co. Kildare, Ireland. thomas.j.whelan at nuim.ie

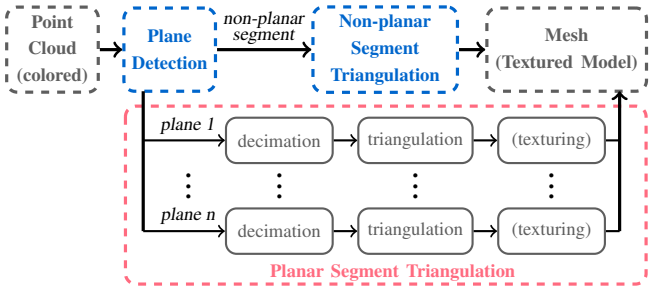


Fig. 2: Parallel system architecture to process point clouds of large-scale open scene scans or maps.

poorly in removing redundancy in dense point clouds, or are not suited to the kind of data typically acquired with common robotic sensors. In this paper we address these problems with two main contributions based on the work of [10]. Firstly we present an accurate and robust algorithm for planar segment decimation and triangulation. In comparison to the existing QTB algorithm, our algorithm guarantees geometrical accuracy during simplification with fewer triangles, without duplicate points, artificial holes or overlapping faces. Secondly we present a method to automatically generate textures for the simplified planar mesh based on dense colored vertices. Our experimental results show that the presented solutions are efficient in processing large datasets, through the use of a multi-threaded parallel architecture.

III. System Overview

A. Building Blocks

Our system architecture is shown in Figure 2. It takes a point cloud as input and generates a triangular mesh as output. If the input is a colored point cloud, the output can also be a textured 3D model. The processing pipeline consists of three main blocks.

Plane Detection segments the input into planar and non-planar regions to enable separate triangulation and parallel processing. This design is especially beneficial for real-world environments, where multiple independent planar surfaces occur frequently. In our system we apply a local curvature-based region growing algorithm for plane segmentation [10].

Non-Planar Segment Triangulation generates a triangular mesh for non-planar segments using the GPT algorithm [9]. Given a colored point cloud, we preserve the color information for each vertex in the output mesh. Dense triangular meshes with colored vertices can be rendered (with Phong interpolation) to appear similar to textured models. Additionally, as opposed to using textures, maintaining color in vertices of non-planar segments provides easier access to appearance information for point cloud based object recognition systems.

Planar Segment Triangulation triangulates planar segments and textures the mesh afterwards, if given a colored point cloud. In our system we improve the decimation algorithm in [10] and further develop a more accurate and robust solution for triangulation. A detailed description of our algorithm is provided in Section IV. Our method for planar segment texture generation is described in Section V.

B. Computationally Efficient Architecture

To improve computational performance, a multi-threaded architecture is adopted, exploiting the common availability of multi-core CPUs in modern hardware. We apply a coarse-grained parallelization strategy, following the Single Program Multiple Data (SPMD) model [11]. Parallel triangulation of planar segments is easily accomplished by dividing the set of segments into subsets that are distributed across a pool of threads. For maximum throughput of the entire pipeline, segmentation and triangulation overlap in execution. With an n-core CPU, a single thread is used for segmentation and the remaining $n-1$ threads are used for triangulation, each with a queue of planar segments to be processed. Upon segmentation of a new planar region, the segmentation thread checks all triangulation threads and assigns the latest segment to the thread with the lowest number of points to be processed. This strategy ensures even task distribution among all threads. When plane segmentation is finished, the segmentation thread begins the non-planar triangulation in parallel to the other triangulation threads.

IV. Triangulation of Planar Segments

In this section, our algorithm for planar segment decimation and triangulation is described. A simplified mesh of a planar segment is generated by removing redundant points that fall within the boundary of the segment. In the following text the input planar segment is denoted as $\mathcal{P}$, made up of points $\mathbf{p} \in \mathbb{R}^3$. With colored point clouds, each point $\mathbf{p}$ also contains (R, G, B) color components.

A. QuadTree-Based Decimation

Planar segments have a simple shape which can be well described by points on the boundary of segment. Interior points only add redundancy to the surface representation and complicate the triangulation results. Figure 3 shows such an example, where the planar segment is over-represented with thousands of triangles generated with the GPT algorithm using all planar points. However, a naïve solution to remove all interior points and triangulate only with boundary points normally leads to skinny triangles, again shown in Figure 3. With these observations in mind, the quadtree proves to be a useful structure to decimate the interior points of a segment while preserving all boundary points for shape recovery [10].

1) Preprocessing: To prepare a planar segment for decimation it is first denoised and aligned to the x-y axes. We employ Principal Component Analysis (PCA) over the planar segment to compute a least-squares plane fit as well as an affine transformation T for x-y axes alignment. The aligned planar segment is denoted as $\mathcal{P}_t$. Afterwards, the boundary points of $\mathcal{P}_t$ are extracted as an α-shape [12], [13]. We denote the boundary as the concave hull $\mathcal{H}$ of the planar segment, which is an ordered list of vertices describing a polygon for which $\forall \mathbf{p} \in \mathcal{P}_t$ and $\mathbf{p} \notin \mathcal{H}$, $\mathbf{p}$ is inside the polygon.

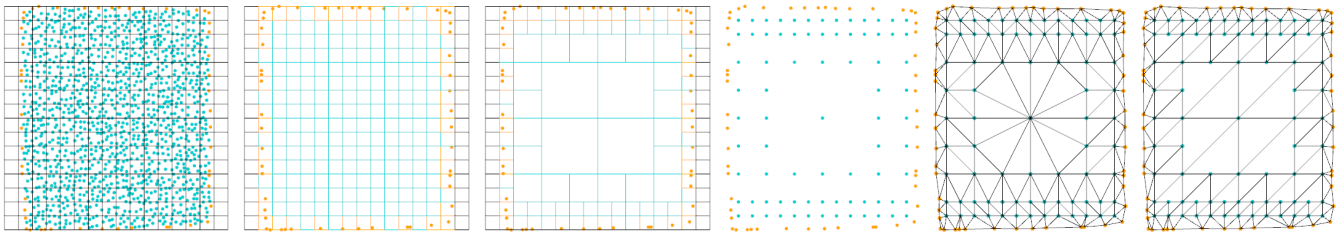


Fig. 4: Planar decimation and triangulation (boundary and interior points are orange and teal, respectively), from left to right: (a) initialize by subdividing the quadtree bounding box; (b) classify nodes into interior (teal), boundary (orange) and exterior (black); (c) merge interior nodes; (d) generate vertices; (e) point-based triangulation; (f) polygon-based triangulation.

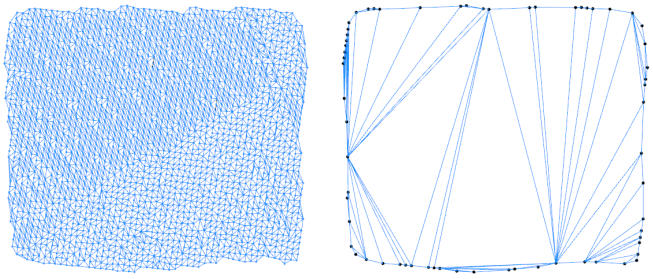


Fig. 3: Undesirable planar triangulation: the left GPT mesh over-represents the shape while the right boundary-based Delaunay triangulation produces unnatural skinny triangles.

1	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**2**	2	**1**
2		4		4		4		4		4		4		4		4		4		4		2
2	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**4**	4	**2**
2				4				4				4				4				4		2
2				4				4				4				4				4		2
2				4				4				4				4				4		2
2	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**4**	4	4	4	**2**
2				4								4								4				2
2				4								4								4				2
2				4								4								4				2
2	4	4	4	**4**								4								**4**	4	4	4	**2**

Fig. 5: Degree grid of the upper planar segment in Figure 4 (0-valued cells hidden). The underlined bold values are the degrees of the inner vertices $\mathcal{I}$.

2) Decimation: Planar segment point decimation consists of four steps as shown in Figure 4.

Firstly, a quadtree is constructed by subdividing the bounding box of $\mathcal{P}_t$ into a uniform grid of small cells. Typically the 2D bounding box is non-square, in which case the smallest side is extended to equalize width and height. The resulting bounding box $\mathbf{b}$ is composed of a minimum point $\mathbf{b}_{min}$ and a maximum point $\mathbf{b}_{max}$, with a dimension $\mathbf{s} = \mathbf{b}_{max} - \mathbf{b}_{min}$. Secondly, the quadtree nodes are classified as either interior, boundary or exterior. An *interior* node is fully contained within the polygon $\mathcal{H}$, while an *exterior* node is fully outside. All others are *boundary* nodes, which intersect $\mathcal{H}$. Thirdly, the interior nodes of the quadtree are merged to create nodes of variable size, typically largest around the center and becoming increasingly fine-grained when approaching the boundary. When a parent node contains only interior children, the four children nodes are merged into one. The merged node is then also classified as interior, allowing further recursive merging with its siblings. Finally, the corner points of the remaining interior nodes are extracted as the new internal vertices $\mathcal{I}$ of $\mathcal{P}_t$, while all boundary points $\mathcal{H}$ are preserved.

B. Triangulation

We provide two methods for triangulation of a simplified planar segment: 1) a low-complexity Point-Based Triangulation and 2) an alternative Polygon-Based Triangulation. Both methods make use of the Constrained Delaunay Triangulation (CDT) [14].

1) Point-Based Triangulation: The point-based approach is a low-complexity triangulation method, where CDT is directly applied to the decimated segment. The ordered boundary vertices $\mathcal{H}$ serve as constraining edges and the inner vertices $\mathcal{I}$ are used as input points. An example output is shown in Figure 4. Point-based triangulation has all of the advantages of Delaunay triangulation but does produce more triangles than the polygon-based approach described next.

2) Polygon-Based Triangulation: The regular grid pattern of the inner vertices $\mathcal{I}$ immediately lends itself to a simple triangulation strategy, where two right-angled triangles are created over each interior node of the merged quadtree. To complete the triangulation, the space between the interior right-angled triangles and the boundary points $\mathcal{H}$ is triangulated using CDT. Two sets of constraining edges are input to the CDT, one being $\mathcal{H}$ and the other being a rectilinear isothetic polygon that bounds interior triangles. This two-step triangulation is similar to the QTB algorithm of [10]. However, a major difference lies in how the boundary points are connected. With our CDT-based approach, we avoid overlapping triangles and artificial holes that would normally be produced by the QTB algorithm.

Efficient computation of the polygon which exactly bounds the interior vertices $\mathcal{I}$ is non-trivial, since the interior nodes provide only sparse spatial information for geometric operations. We invoke a solution that maps the interior vertices onto a binary image, where the bounding polygon can be easily extracted using a greedy nearest-neighbour tracing algorithm normally used in image processing [15].

The binary image is represented by an $n \times n$ array, where $n = 2^{d+1} + 1$ and d is the quadtree depth. This provides a 2D grid large enough to represent empty space between the two vertices of any edge. To project a vertex $\mathbf{v} \in \mathcal{I}$ onto the

array, a mapping function $f : \mathbb{R}^3 \rightarrow \mathbb{N}^2$ is defined by

$$f(\mathbf{v}) = \frac{n(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}, \tag{1}$$

where $\mathbf{b}$ is the bounding box and $\mathbf{s}$ is its dimension. The division is performed on an element-by-element basis. Given that $\mathcal{I}$ is aligned to the x-y axes, function f effectively maps from $\mathbb{R}^2$ to $\mathbb{N}^2$. We associate two elements with each array cell: a reference to the mapped vertex (effectively implementing f^{-1}) and a degree value to quantify vertex connectivity. Initially, the degree is zero for all cells. During the triangulation of $\mathcal{I}$, the degree grid is populated. When a vertex is extracted from the merged quadtree, the reference of the corresponding cell is updated and its degree is increased by 1. This policy alone cannot fully recover the degree of a given vertex, since only the two ends of an edge are obtained from quadtree vertices. To overcome this problem, all cells between the two ends of an edge also have their degree increased by 2. Figure 5 shows a part of the degree grid of the planar segment in Figure 4. If we consider the interior triangulation to be a graph, the 2D degree grid resolves the degree of each vertex. All non-zero cells are treated as "1-valued" foreground pixels and the rest as "0-valued" background pixels in the binary image representation.

V. Texture Generation

In this section we present our texture generation algorithm for planar segments using dense colored point clouds. Due to the significant loss of colored vertices during decimation, the appearance of a simplified planar segment is greatly diminished. We therefore generate textures prior to decimation for the purpose of texture mapping the simplifed planar mesh.

We generate textures by projecting the vertex colors of the dense planar segment onto a 2D RGB texture $\mathcal{E}(x, y) \in \mathbb{N}^3$. We define a texture resolution $\mathbf{d}$ as some resolution factor r times $\mathbf{s}$, where $\mathbf{s}$ assumes the dimension of the bounding box $\mathbf{b}$. In our experiments a value of $r = 100$ provides good-quality textures. The resolution factor can also be automatically computed based on point cloud density. Each pixel $\mathbf{a} \in \mathcal{E}$ is first mapped to a 3D point $\mathbf{v}$ by a mapping function $g : \mathbb{N}^2 \rightarrow \mathbb{R}^3$, defined as

$$g(\mathbf{a}) = \frac{\mathbf{as}}{\mathbf{d}} + \mathbf{b}_{min}, \tag{2}$$

with an element-by-element calculation. Since $\mathcal{P}_t$ is aligned to the x-y axes, the function g effectively maps to $\mathbb{R}^2$. A corresponding colored point to $\mathbf{v}$ in $\mathcal{P}_t$ is found by a nearest neighbor search using a kd-tree. We have chosen this approach as it produces good-quality textures while being computationally inexpensive. However, it can be easily extended to produce even higher-quality textures by averaging a number of k-nearest neighbours. Algorithm 1 describes the texture generation process. Figure 6 shows an input planar segment and the output texture.

When texture mapping the final planar mesh, the uv texture coordinates $\mathcal{U}$ for the vertices $\mathcal{O}$ of each face are computed with the inverse function $g^{-1} : \mathbb{R}^3 \rightarrow \mathbb{N}^2$, derived from Equation (2) as

$$g^{-1}(\mathbf{v}) = \frac{\mathbf{d}(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}. \tag{3}$$

With x-y axes aligned points, g^{-1} is actually mapping from $\mathbb{R}^2$. Algorithm 2 describes the uv-coordinates computation. The list $\mathcal{U}$ guarantees a 1-to-1 mapping to the set $\mathcal{O}$.

Any objects lying on a planar segment are completely excluded from the texture and not projected onto the plane. In fact, the generated texture implicitly provides the Voronoi diagram of the face of the object lying on any plane, which in turn provides position and orientation information of any object lying on a segmented plane, as shown in Figure 7.

Fig. 6: Texture generation, from left to right: (a) plane segment from a colored point cloud; (b) generated texture.

Algorithm 1: Vertex color to texture.

Input: $\mathcal{P}_t$ set of transformed input vertices
Input: $\mathcal{H}$ concave hull of $\mathcal{P}_t$
Output: $\mathcal{E}$ 2D RGB texture
foreach *pixel* $\mathbf{p}$ *in* $\mathcal{E}$ **do**
  $\mathbf{v} \leftarrow g(\mathbf{p})$;
  if $\mathbf{v}$ *is inside* $\mathcal{H}$ **then**
    $\mathbf{n} \leftarrow$ nearest-neighbour of $\mathbf{v}$ in $\mathcal{P}_t$;
    $\mathbf{p} \leftarrow (\mathbf{n}_R, \mathbf{n}_G, \mathbf{n}_B)$;
  else
    $\mathbf{p} \leftarrow (0, 0, 0)$;

Algorithm 2: uv texture coordinate calculation.

Input: $\mathcal{O}$ set of final face vertices
Output: $\mathcal{U}$ uv texture coordinates for $\mathcal{O}$
foreach *vertex* $\mathbf{v}$ *in* $\mathcal{O}$ **do**
  $\mathbf{a} \leftarrow g^{-1}(\mathbf{v})$;
  $u \leftarrow \frac{\mathbf{a}_x}{\mathbf{d}_x}$;
  $v \leftarrow 1.0 - \frac{\mathbf{a}_y}{\mathbf{d}_y}$;
  Add (u, v) to $\mathcal{U}$;

VI. Evaluation and Results

In this section we evaluate our work with a series of experiments. We ran our C++ implementation on Ubuntu Linux 12.04 with an Intel Core i7-3930K CPU at 3.20 GHz with 16 GB of RAM. Four colored point clouds of real-world environments were used in the experiments, as shown in Figure 10a. These datasets encompass a wide variation in the number of points, planar segments and their geometry. All four datasets have been acquired with an implementation of the Kintinuous dense RGB-D mapping system [16].

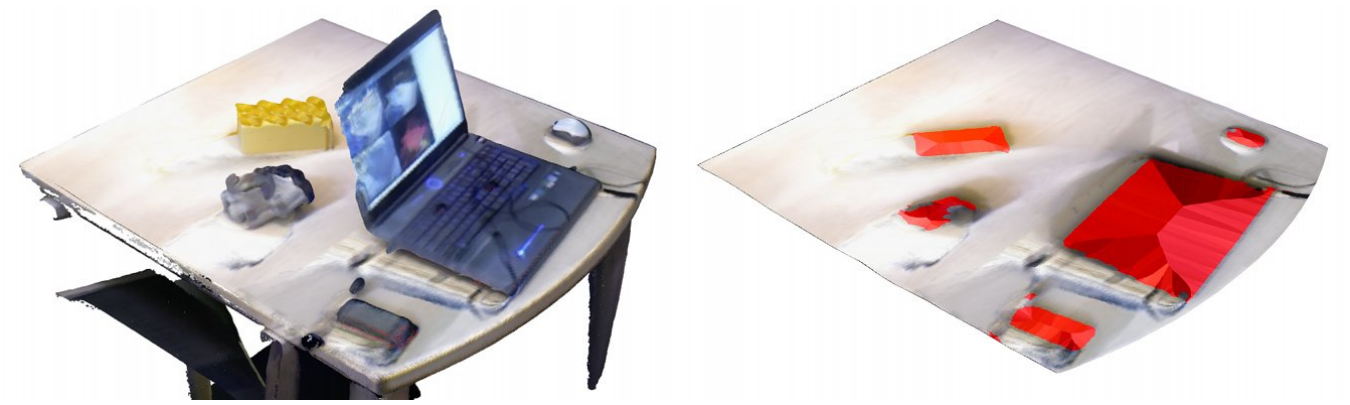


Fig. 7: Implicit object information from texture generation, from left to right: (a) input colored point cloud; (b) generated texture with implicit Voronoi diagrams and locations of objects resting on the plane highlighted.

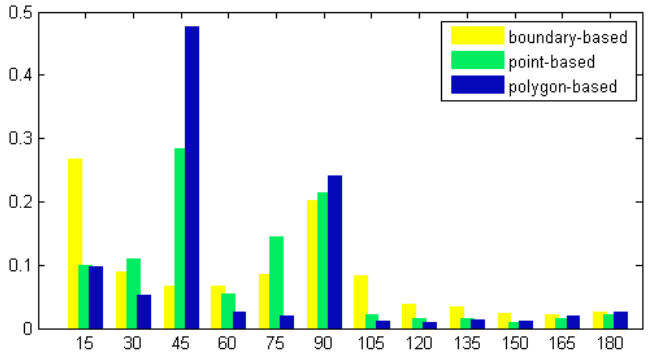


Fig. 8: Triangulation quality measured with the angle distribution of planar meshes.

A. *Triangulation Performance*

To assess the triangulation performance, qualitative and quantitative evaluations are presented. A comparison of the triangulation algorithms is shown in Figure 10. Additionally we present map fly-throughs in a video submission available at http://www.youtube.com/watch?v=dn6ccmFXSzQ . It can be seen that both algorithms produce a highly simplified triangulation, while preserving the principal geometry of the segment.

Further assessment of mesh quality is done by measuring the angle distribution across meshes. A naïve simplified planar mesh is set as a baseline, which applies Delaunay triangulation to only the boundary points of a planar segment. The normalized distribution is shown in Figure 8, collected from the 400 planar segments of the four datasets. It can be seen that approximately 80% of the triangles from the polygon-based triangulation are isosceles right-angle triangles, resulting from the quadtree-based triangulation. With point-based triangulation, the angles spread over 30°-90°, whereas the naïve boundary-based triangulation shows an even more random distribution. Defining a skinny triangle as one with a minimum angle $<15°$, the percentages of skinny triangles with boundary-based, point-based and polygon-based triangulation are 28%, 10% and 10%, respectively.

TABLE I: Planar point reduction with our decimation algorithm in comparison to the QTB algorithm.

	data 1	data 2	data 3	data 4
Total points	890,207	1,094,910	2,799,744	5,641,599
Planar points	540,230	708,653	1,303,012	2,430,743
QTB decimation	105,663	303,348	189,772	457,678
Our decimation	47,457	84,711	43,257	76,624

TABLE II: Planar mesh simplification with our triangulation algorithms measured with triangle counts, in comparison to GPT and the QTB algorithm.

	data 1	data 2	data 3	data 4
GPT	1,020,241	1,350,004	2,293,599	4,665,067
QTB	90,087	288,833	182,648	433,953
Point-based	85,653	161,270	79,988	143,396
Polygon-based	76,367	130,642	66,413	118,275

The effectiveness of planar segment decimation is also evaluated. Table I shows the point count for planar point decimation. Approximately 90% of the redundant points are removed with our algorithm, which is 15% more than the QTB algorithm, despite the fact that both algorithms are based on a quadtree. Part of this reduction gain comes from our triangulation methods, which add no extra points once decimation is done, unlike the QTB algorithm. In Table II, the mesh simplification statistics with triangle counts are also given. We take the triangle count of GPT for non-decimated planar segments as the baseline. In accordance with the point count reduction, both of our algorithms require no more than 10% of the amount of triangles of a non-decimated triangulation, and both perform better than the QTB algorithm.

B. *Texture Generation Performance*

In Figure 10d, generated textures are shown. The output textures incorporate almost all visual information contained in the original dense point cloud, enabling a photo-realistic and aesthetically-pleasing textured 3D model.

TABLE III: Efficiency of triangulation and the parallel architecture, measured in seconds. The 1:x ratio denotes 1 segmentation thread with x triangulation threads.

	data 1	data 2	data 3	data 4
Number of planar segments	101	116	66	117
Serial GPT	18.6	24.3	44.2	91.1
Serial QTB	16.7	18.7	38.3	73.1
Serial point-based	6.9	9.8	17.7	40.2
Serial polygon-based	6.9	9.5	17.8	40.0
Serial polygon-based (texture)	8.3	10.0	20.3	41.4
1:1 Polygon-based	6.4	8.1	15.1	33.8
1:1 Polygon-based (texture)	7.6	8.5	17.4	35.2
1:3 Polygon-based	3.6	4.2	8.3	19.2
1:3 Polygon-based (texture)	4.4	4.1	9.2	19.6
1:5 Polygon-based	3.7	3.5	7.9	16.1
1:5 Polygon-based (texture)	4.7	3.5	8.7	16.2

C. Computational Performance

Lastly, we evaluate the computational efficiency of our algorithms and the parallel system for large-scale data processing. The baseline for comparison is standard serial processing with the GPT and QTB algorithms. Table III shows the execution times. The point-based and polygon-based triangulations are approximately of the same speed, both 2 to 3 times faster than the GPT and QTB algorithms. The results also show that the texture generation algorithm is fast in execution, processing multi-million point datasets in less than 2 seconds. Examining the bottom half of Table III, it is clear that the parallel system architecture has a profound effect on the overall performance. The execution time decreases with an increasing number of triangulation threads. An effect of diminishing returns becomes apparent as the number of triangulation threads increases, due to the overhead associated with the parallel implementation. However, as the per-thread workload increases, such as inclusion of texture generation, the overhead of parallelization becomes overshadowed.

D. Discussion

Both point-based and polygon-based triangulation yield accurate and computationally efficient planar segment triangulations with significant point and triangle count reductions, both exceeding the performance of the QTB algorithm. The point-based approach is of low complexity and maintains good triangular mesh properties that are desirable for lighting and computer graphics operations. The polygon-based approach yields higher point and triangle count reductions with a more regularized mesh pattern, capturing information about the scene in the form of principal geometric features, such as the principal orientation of a planar segment. While the polygon-based method produces less triangles, it does generate T-joints in the mesh. Such feature is detrimental when employing Gouraud shading and other lighting techniques to render a mesh with colored vertices. The polygon-based and point-based methods offer a trade-off depending on the desired number of triangles or the intended use of the final triangulation. With robot navigation in mind, the low polygon-count models achieved with our system are suitable for use in a primitives-based localization system, such as the KMCL system of Fallon *et al.* [3].

The gaps between planar and non-planar triangulations are apparent. The gap can also be closed by including the boundary vertices of the segmented planes into the non-planar segment GPT triangulation, as shown in Figure 9. The number of boundary vertices can be increased with a smaller alpha value when computing the concave hull of each segment or by linearly interpolating between boundary vertices. Extra vertices can also be extracted from the vertex degree grid used in polygon-based triangulation. In our system we chose to leave these gaps open, as this separation gives an easier visual understanding of any map, implicitly providing a separation between structural features (like walls, table tops) and "object" features, useful in automatic scene understanding, manipulation and surface classification.

VII. Conclusions

In this paper we have studied the problem of triangulation of planar segments from dense point clouds with a focus on quality and efficiency. Three significant contributions are made. Firstly, we have made a strong improvement on planar segment reconstruction. Both of the presented point-based and polygon-based triangulation methods produce a more accurate, simpler and robust planar triangulation than the existing QTB algorithm. With these two algorithms approximately 90% of input planar points are removed, and the planar segments are triangulated with no more than 10% of the amount of triangles required without decimation. Secondly, we have developed a computationally inexpensive algorithm to automatically generate high-quality textures for planar segments based on colored point clouds. Last, our parallel system multi-threaded architecture enhances the efficiency in processing large-scale datasets. The results show that our system provides a computationally manageable map representation for real-world environment maps and also generates a visually appealing textured model in a format useful for real-time robotic systems.

References

[1] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The Int. Journal of Robotics Research*, 2012.

[2] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Jul 2012.

[3] M. F. Fallon, H. Johannsson, and J. J. Leonard, "Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2012.

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

[5] 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, pp. 1119–1134, May 2009.

Fig. 9: Joining of GPT mesh with planar segment triangulations. Left shows unjoined segments and right shows segments joined with interpolated boundary vertices.

[6] 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, 1999.

[7] C. E. Scheidegger, S. Fleishman, and C. T. Silva, "Triangulating point set surfaces with bounded error," in *Proc. of the 3rd Eurographics symposium on Geometry Proc.*, Eurographics Association, 2005.

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

[9] 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*, pp. 3218–3223, 2009.

[10] L. Ma, R. Favier, L. Do, E. Bondarev, and P. H. N. de With, "Plane segmentation and decimation of point clouds for 3d environment reconstruction," in *Proc. the 10th Annual IEEE Consumer Communications & Networking Conference*, Jan. 2013.

[11] F. Darema, D. George, V. Norton, and G. Pfister, "A single-program-multiple-data computational model for epex/fortran," *Parallel Computing*, vol. 7, no. 1, pp. 11 – 24, 1988.

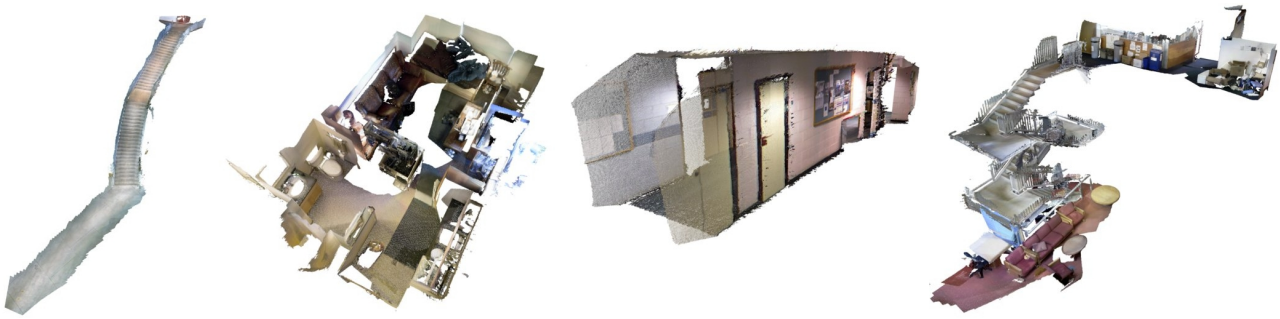
[12] M. Duckham, L. Kulik, M. Worboys, and A. Galton, "Efficient generation of simple polygons for characterizing the shape of a set of points in the plane," *Pattern Recogn.*, vol. 41, no. 10, pp. 3224–3236, 2008.

[13] B. Pateiro-López and A. Rodríguez-Casal, "Generalizing the convex hull of a sample: The r package alphahull," *Journal of Statistical Software*, vol. 34, no. i05, 2010.

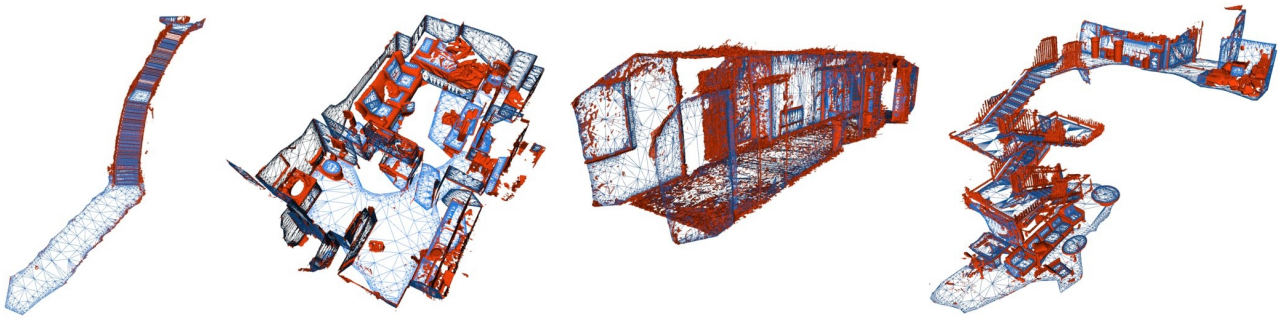
[14] V. Domiter and B. Zalik, "Sweep-line algorithm for constrained delaunay triangulation," *Int. J. Geogr. Inf. Sci.*, vol. 22, pp. 449–462, Jan. 2008.

[15] J. Marquegnies, "Document layout analysis in SCRIBO," Tech. Rep. CSI Seminar 1102, Research and Development Laboratory, EPITA, http://www.lrde.epita.fr/dload/20110704-Seminar/1102.pdf, July 2011.

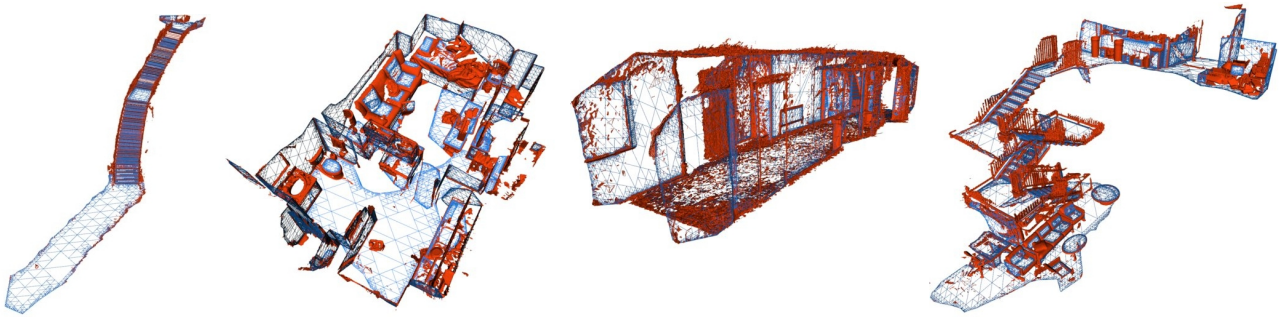
[16] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, (Karlsruhe, Germany), May 2013. To appear.



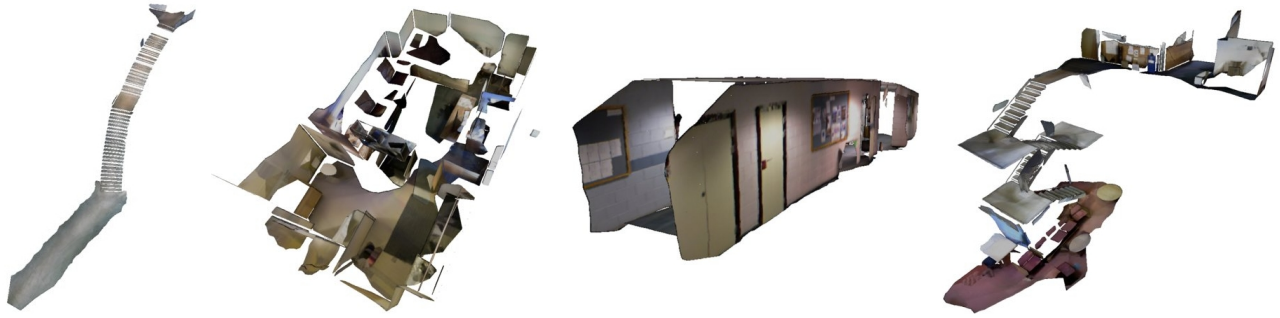
(a) Four dense colored point cloud datasets used for evaluation, numbered 1, 2, 3 and 4 from left to right.



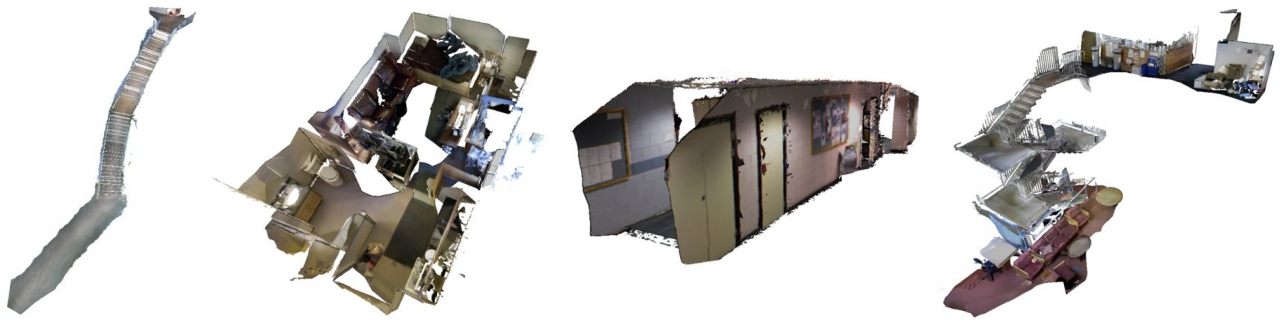
(b) Point-based triangulation, with planar and non-planar meshes highlighted in blue and orange, respectively.



(c) Polygon-based triangulation with planar and non-planar meshes highlighted in blue and orange, respectively.



(d) Textured simplified planar segments from each dataset.



(e) Complete 3D model with our proposed system.

Fig. 10: Four evaluated datasets with various triangulation results and texturing results.