Quality Preserving Fusion of 3D Triangle Meshes

Sebastian Wuttke, Dominik Perpeet, and Wolfgang Middelmann
Fraunhofer IOSB
76275 Ettlingen, Germany
{sebastian.wuttke, dominik.perpeet, wolfgang.middelmann}@iosb.fraunhofer.de

***Abstract*—Surface representations in the shape of 3D triangle meshes of heterogeneous quality are readily generated using sensors such as laser scanners. We propose an algorithm to fuse two registered, i.e. aligned in rotation and translation, meshes while using only vertices contained in the input data. Mesh quality is determined for each input mesh depending on the assumed sensor model. In the first step, redundant faces are removed using local comparison of mesh quality. Finally, the meshes are fused by inserting new faces, using the remaining vertices, resulting in a mesh with locally optimal quality regarding both input datasets. Since no new vertices are introduced, further processing can rely on the original measurements and any associated meta data such as sensor specific information. Results are shown for fusing overlapping parts of datasets representing a ballroom and the Stanford Bunny.**

I. Introduction

A variety of sensors deliver measurements which are in turn used to generate surface models, represented as 3D triangle meshes. Often multiple measurement sequences, such as laser scans, or even data from different sensors need to be merged. In the following we will present a method to fuse two 3D triangular meshes of different quality or even from different sensors. Vertices of the final mesh are original measurements, with no added point data, redundant faces are removed. Since all steps take the local mesh quality based on topology and sensor models into consideration, information from individual measurement sequences is preserved in the final, merged mesh.

Previous methods based on the *marching cubes* [1] algorithm achieve mesh fusion via union of their points and creation of a new mesh. Space is partitioned into voxels of the same size, where every such cube either remains empty or a triangle is fitted to its contained points. An important advantage of this approach is that the resulting 3D surface model is of constant density. While this can simplify further processing, it comes at the cost of discarding original measurements. Since algorithms based on marching cubes only need to access independent subsections of the data, they are inherently well suited for parallelization. Great care must be taken to choose voxel orientation and size correctly. Flat surfaces not aligned with voxel edges can result in artificial alignment differences between neighboring triangles even where the original surface was continuous. Choice of an appropriate voxel size is crucial: if too small, sensor noise and alignment issues might have a severe impact. If the voxel is too great, surface details are lost during the fitting of triangles. The need for exact parametrization can be alleviated by methods such as *adaptive marching cubes* [2], yet the principal method of using local point data to calculate approximating surface triangles remains.

An alternative approach consists of using implicit modeling [3], where the data is approximated by an implicit surface using a function. There are many possible functions and approximation methods, such as *moving least squares* methods [4] or even multi-resolution techniques [5]. All variants depend on identifying appropriate parameters for fitting them to the data correctly. If successful, noise and missing data can be countered, resulting in a smoothed and filtered surface model. *Implicit fusion* [6] transforms existing meshes into an implicit representation, creates a transition surface and tessellates the result to form the combined mesh. Surface approximation can also be achieved using Support Vector (SV) machines [7]. Support Vector representation of surfaces can prove to be beneficial in further processing such as physics simulations. On the other hand they prove challenging to calculate locally. Furthermore, quality criteria and special exception decisions affect the core of a SV machine and can therefore not be added or changed without extensive changes to the algorithm.

A similar approach to our mesh fusion, called *zipper*, was already described by Turk and Levoy [8]. They use a confidence measure given by their structured light scanner to determine the quality of mesh vertices. In overlapping areas, they introduce new vertices to allow for a smoother combined mesh. Although these new vertices lie on the edges of existing triangles, they do not correspond to an actual measurement.

All previously mentioned methods either perform a surface reconstruction using only points or introduce new vertices to existing meshes in order to fuse them. Our novel approach guarantees that non-overlapping regions of the meshes to be merged remain unaffected. Additionally, no artificial vertices are introduced during the process. This ensures that each vertex of the resulting mesh can be attributed to exactly one vertex of an input mesh, preserving optional meta information such as sensor parameters. Since no generalization or model-based approximation is performed, no assumptions about geometry are made save that the input meshes be triangular and represent 2D-manifolds. Especially there are neither requirements on density and distribution, as is helpful when constructing an implicit representation, nor on axis alignment, as impacts voxel based algorithms based on marching cubes. Furthermore, since meshes are only inspected and compared locally, there is no need for homogeneity. Instead, heterogeneous density, e.g. from a terrestrial laser scan, is preserved.

II. Quality Preserving Mesh Fusion

The general goal of sensor data fusion is to improve the data quality regarding a set of criteria by combining different sensor inputs. The possible improvement greatly depends on the quality of the source data and assertions about the used sensor model. Sensor data fusion combines different input data sets and generates output optimized regarding a set of criteria, referred to as quality in the following. Assertions about used sensors, e.g. choosing certain sensor models, and the signal to noise ratio of input data are important factors in the quality of the output.

In a laser scanning system, the laser beam is subject to divergence. Because of this, the size of the laser footprint on a scanned surface increases quadratically with the distance. This results in the smoothing of detailed surface structures which in turn has a negative impact on the quality of the measurement. In general the quality is worse if the measurement is farther away from the scanner.

A simple fusion approach, using only geometrical information, would result in a reduced level of detail, i.e. even already acquired surface structures would be smoothed. Our approach instead uses sensor model based meta data and decides locally which mesh is better suited to preserve detailed structures.

Fusion of two meshes is achieved in two separate stages. First a reduction of the meshes is performed by removing overlap. In the second step, resulting gaps between the two meshes are closed by adding new triangles along common borders of the meshes.

In the following, these two steps are described in detail and demonstrated with data from the Stanford Bunny. A triangle mesh was already generated with the *zipper* algorithm [8]. Using this data, two overlapping partial models are generated and the steps of the mesh fusion algorithm are shown. Surface models of the two parts are depicted in Fig. 1, overlapping triangles are marked red.

A. Mesh reduction

Regions from both meshes that are geometrically close enough, taking the used sensor into consideration, describe the same surface area. Triangles from these areas are called *overlapping*. Quality is calculated locally using interchangeable quality criteria. The algorithm preserves local quality by only replacing those parts of one mesh in overlapping regions if the corresponding parts in the other mesh have higher quality.

If the two input meshes were simply merged without removing any faces, overlap would usually result in intersecting faces. For many further processing steps this would prove to be problematic, since a single surface could be ambiguously represented by multiple intersecting triangles. In order to remove potentially problematic faces, we consider for each triangle t_i all triangles of the respectively other mesh within *minGapSize* distance. For these we have to decide whether they represent the same surface as t_i and thus are redundant. In our sample case of data from a terrestrial laser scanner these criteria include the direction from which the surface was scanned as well as orientation via comparison of the

Fig. 1. Two partial models of the Stanford Bunny to be combined using the fusion algorithm. Overlapping areas marked red. Parts are shown separate for visualization purposes only, the data is aligned.

triangle's normal. For each pair of redundant triangles the one with lower quality is deleted. Quality criteria depend on the intended application. The ones we use fall into two categories: sensor model and mesh topology based.

Sensor model based criteria utilize meta information linked to the measurements:

- Distance to the scanner: Usually, and especially for optical systems such as laser scanners, sensor accuracy deteriorates with increasing distance from the measurement unit.
- Surface orientation: Reflective properties have a different impact on measurements depending on the angle at which the sensor detected a surface [9].
- Signal strength: Some sensors deliver meta data which can be used to gauge accuracy of measurements. Several laser scanners report the measured intensity of captured reflected light, allowing a comparison to calibrated ranges.

On the other hand, quality criteria derived only from mesh topology allow for a more sensor independent fusion. These can vary greatly depending on the intended application:

- Triangle area: If a single larger triangle can replace several smaller ones without loss of accuracy, the larger triangle is usually preferable. A lower triangle count implies less computational burden and eases further processing in many cases.
- "Pointedness": Degenerate triangles lead to numerical problems and should be avoided. In our case we use the ratio of longest side to shortest side. Also possible is the ratio of largest to smallest angle [10].

The presented mesh fusion results are achieved using a linear combination of the above mentioned quality criteria. In case of the ballroom, the distance to the scanner is strongly weighted because it is one of the best suited criteria to gauge

the quality of the measurements. The proposed mesh fusion algorithm focuses more on the measurement quality than on the topological mesh quality. Therefore weighting the triangle area and pointedness less.

The used model of the Stanford Bunny contains no meta information about the sensor model, thus all derived quality criteria based on it are weighted with zero. Accordingly the topology based criteria are weighted more strongly.

It should be noted that, once all overlap has been removed, most implementations can be optimized for merging the input meshes into a combined dataset, since the mesh reduction step guarantees that the two meshes do not intersect each other. The original parts are separated by a gap wherever they previously overlapped. The result for the Stanford Bunny is presented in Fig. 2.

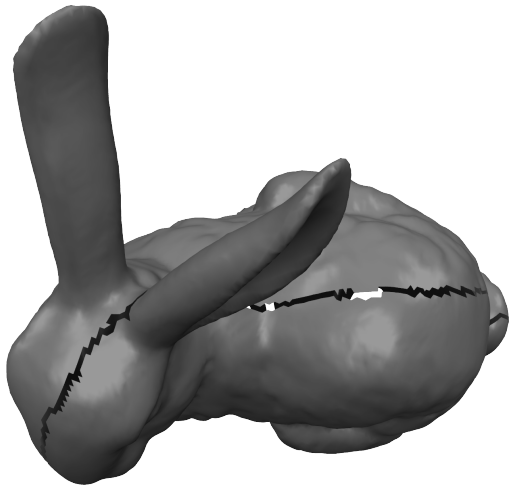

Fig. 2. Combined meshes after the *reduce* step removed all overlap. The *merge* step will close the remaining gap while only using existing vertices.

B. Mesh merging

Border vertices are vertices with at least one outgoing edge that does not belong to two triangles. During the combining process, lists with all border vertices of each mesh can be generated. The mesh *merge* step has to connect these border vertices with new triangles where appropriate. Important criteria for these are that they introduce neither overlap nor other conditions that lead to a non-2D-manifold mesh.

Merging is performed in two steps that are repeated until each initial border vertex has been reviewed once:

1. Choose a vertex X, not previously used during *merge*, from one border vertex list and perform the following:

- Consider all vertices from the other border vertex list within a distance of *maxGapSize*, beginning with the geometrically closest vertex Y.
- If Y has already been used, choose the next closest vertex.
- If X and Y have been measured from (nearly) opposite directions, they probably do not represent the same surface; choose the next closest vertex.
- If X is not the nearest unused vertex as seen from Y, i.e. they are not each other's closest neighbors, choose the next closest vertex.
- If one such Y is found, there are now four possible triangles that can be inserted, see Fig. 3. Of these, the one with the best quality regarding the previously mentioned criteria (cf. section II-A) is added.

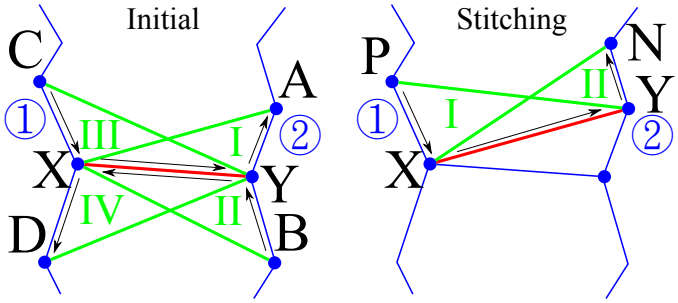

Fig. 3. Visualization of all triangle candidates for both steps of the *merge* process. 1 and 2 are the two meshes which should be merged. X and Y are the vertices from the currently *active* edge (marked in red). I to IV are all possible triangle candidates which contain the active edge. They are evaluated and the one with the best quality is added to the fused mesh.

2. Once a suitable triangle has been found, there are now two additional edges connecting the two meshes. Perform the following for each in turn respectively referred to as the *active* edge:

- P (previous border vertex of X) and N (next border vertex of Y) provide the two possible new triangles that can be inserted to further close the gap (*stitching*), as depicted in Fig. 3.
- If a candidate would lead to undesired results like non-2D-manifolds or inconsistent topologies (cf. search for initial triangle) mark it as invalid.
- If only one candidate is valid insert it, or if both are valid, insert the one with the higher quality.
- Choose the newly created edge between the meshes as the new *active* edge and repeat these steps to close the gap or until no valid triangle candidates can be found.

After these steps are finished and no further candidates can be found, all gaps up to *maxGapSize* are closed. The results for the Stanford Bunny are depicted in Fig. 4, with added edges marked red.

C. Computational complexity

The *reduce* step performs a nearest neighbor (NN) query for each triangle. With N the number of triangles in mesh 1 and M the number of triangles in mesh 2, these queries can be performed in $N * \mathcal{O}(logM)$ and $M * \mathcal{O}(logN)$ by using a k-d tree. In the following, we will consider only N and assume $M \leq N$. When including initial k-d tree construction, overall computational complexity for the first step is in $\mathcal{O}(NlogN)$.

Merge is comprised of two tasks: find an initial triangle to connect the two meshes and stitching the gap. Each potential candidate (border vertex) requires a NN query into the other mesh, which is performed in $\mathcal{O}(logN)$. In order to verify

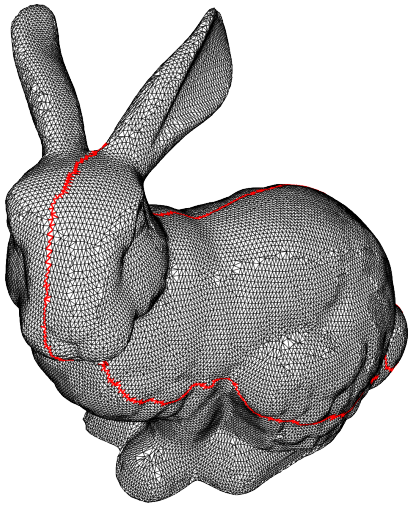

Fig. 4. Merged meshes of the overlapping Stanford Bunny parts. Added edges are marked red.

a candidate the algorithm requires a NN query in the other direction and four topology queries. With all potential candidates this again results in a computational complexity upper bound of $\mathcal{O}(NlogN)$. Yet if one considers average use cases, where meshes only have a partial overlap, one can expect better performance due to the fact that only border vertices close enough to the other mesh are potential candidates. Once an initial triangle has been found and the gap between the input meshes is large enough, the *stitching* step only requires two topology queries for each newly added triangle. With a datastructure such as the half-edge mesh used by our implementation these are performed in constant time, which means that N border nodes can be stitched in $\mathcal{O}(N)$.

While the overall computational intensity for the *merge* step is in $\mathcal{O}(NlogN)$, the actual performance depends on the ratio of number of initial triangles placed to the number of triangles added via stitching. Ideally the gap between meshes resembles a single seam, resulting in long stretches of stitched triangles added in an efficient manner. With sufficient overlap, terrestrial laser scans with a large number of triangles can be expected to have fewer, continuous gaps. In our datasets the ratio varies between 5 and 10%, meaning that no more than a tenth of added triangles required computational expensive NN queries.

III. Results

The algorithm presented in this paper can be used to fuse triangle meshes of objects scanned from the outside, here represented by the Stanford Bunny, as well as objects scanned from the inside, here a ballroom. An evaluation of the execution times using our implementation is presented in the third section.

A. Model 1 - Stanford Bunny

As described in the previous sections, the original model of the Stanford Bunny [8] is used. It is split into two parts which have an overlap of 16.4% (11,385 triangles). Additionally, artificial noise is added to one of the parts to simulate different dataset quality. The average side length of the triangles in the original part is 1.45 mm. By adding uniformly distributed noise each vertex is moved on average by a distance of 0.39 mm with a standard deviation of 0.11 mm. Because this is less than a third of the average triangle side length, no inconsistencies, such as triangle intersections, are expected due to the noise. Used parameters and weights are listed in table I.

TABLE I
Execution parameters and weights for the Stanford Bunny.

Quality Criteria Weights		
Sensor Model	Topology	Parameters [mm]
distance: 0	area: $\frac{1}{3}$	minGap: 1
orientation: 0	pointedness: $\frac{2}{3}$	maxGap: 5

During the *merge* step nearest neighbor queries were only performed for 0.7% of the created triangles. The rest could be created with topology queries via *stitching*. Nearest neighbor queries using our k-d tree implementation which did not return an empty result had an average result size of 11.4 elements.

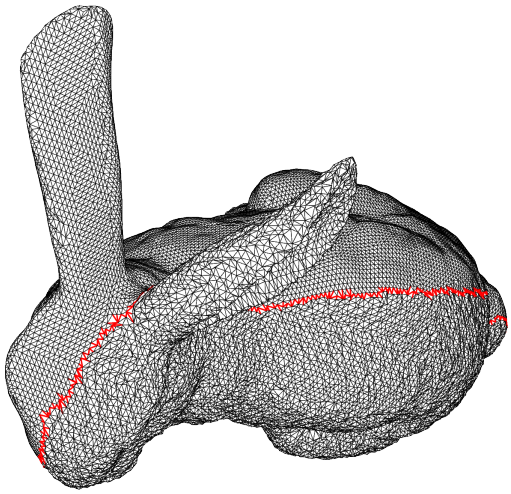

Fig. 5. Final mesh after fusing the two parts. Artificial noise is added to the bottom part of the original meshes. Edges of added triangles between the meshes are shown in red.

Fig. 5 presents the final result of the algorithm for this model. The noise is clearly visible but does not affect the algorithm. Not all added triangles are identical to the version without noise (cf. Fig. 4). This is to be expected, since the added noise leads to a changed gap between the two parts to be fused. Execution times are shown in Table III.

B. Model 2 - Ballroom

For the second model data was acquired with a terrestrial laser scanner. The ballroom scanned is the “Salle Mozart” in Strasbourg, France. The data captured by the scanner can be represented as a 2.5-dimensional range image. In this format the data can be easily and reliable transformed into a 3D triangle mesh.

Fig. 6. Grayscale image of the Salle Mozart in Strasbourg, France, colored using the measured laser intensity. The two parts which will be fused are marked red and green.

The two overlapping meshes used for the algorithm are generated by using different areas from the range image. Fig. 6 depicts these areas surrounded by red and green boundaries, they have an overlap of 6.8% (9.4 M triangles). During the merge step only 4.2% of the triangles are added using nearest neighbor queries. The average size of the not empty results for the nearest neighbor queries is 19.6. Used parameters and weights are listed in table II.

TABLE II
EXECUTION PARAMETERS AND WEIGHTS FOR THE SALLE MOZART.

Quality Criteria Weights		
Sensor Model	Topology	Parameters [mm]
distance: $\frac{10}{17}$	area: $\frac{1}{17}$	minGap: 1
orientation: $\frac{5}{17}$	pointedness: $\frac{1}{17}$	maxGap: 5

To demonstrate the high resolution of the scan Fig. 7 depicts two selected small areas and the result of the fusing algorithm.

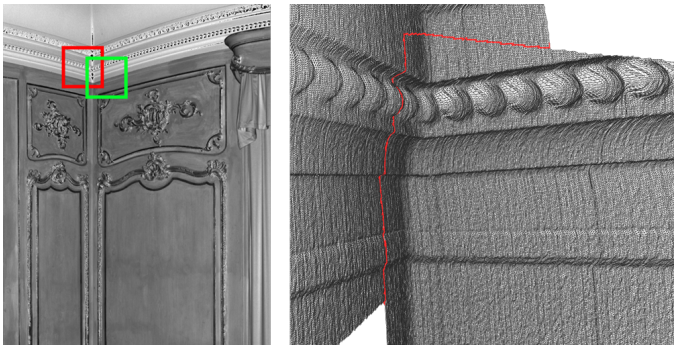

Fig. 7. Left: Selection of stucco from the scan of the Salle Mozart. Right: Fused mesh in highest resolution. Added triangles are colored in red.

C. Performance evaluation

Since the presented algorithm depends on nearest neighbor queries in the *reduce* and *merge* steps, a k-d tree datastructure was chosen. These need to be constructed initially for the triangles of each input mesh and once again after the *reduce* steps for the vertices in each list of border vertices. The implementation is based on the parallel construction algorithm detailed by Choi et al. [11]. With N triangles, construction is performed in $\mathcal{O}(NlogN)$, nearest neighbor queries are executed in $\mathcal{O}(logN)$.

Runtime measurements were performed on a pc with the following configuration:

- Windows Server 2008 R2 Enterprise (64 bit)
- 4 CPUs: Intel®Xeon®X7560 @ 2.27 GHz
- RAM: 128 GB

The results shown in Table III contain only the execution time of the *reduce* and *merge* steps. File I/O and generation of the k-d trees for the nearest neighbor queries are not included.

TABLE III
RUNTIME STATISTICS FOR THE TWO MODELS. GENERATION OF THE K-D TREES AND FILE I/O ARE NOT INCLUDED IN THE GIVEN TIMES.

Model	Number of Triangles				Time [s]
	Mesh 1	Mesh 2	Deleted	Created	
Stanford Bunny	33,097	47,739	12,491	939	1.28
Ballroom	98.6 M	44.0 M	9.4 M	14,613	183.96

For further performance evaluation regarding scalability, models with an equal amount of overlap but different number of triangles are needed. Because of its low triangle count, the Stanford Bunny is not included. Instead, the ballroom model is used with different resolutions. This is achieved by using only each n-th point in each dimension[1] of the range image. The resulting meshes have 0.4 M to 142 M triangles.

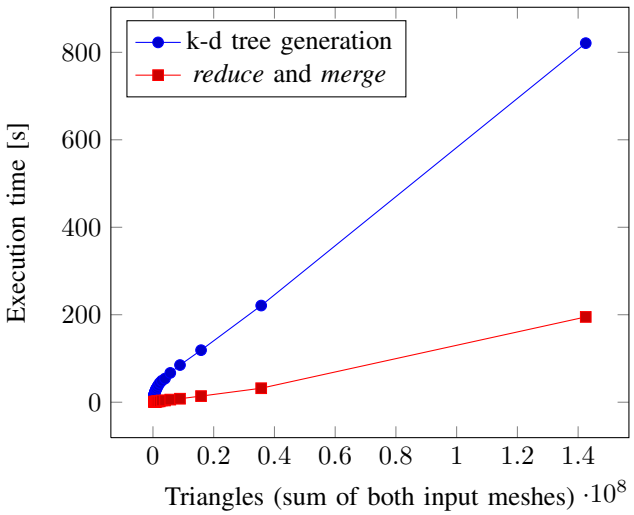

Fig. 8. Average execution times without I/O.

The timing for each resolution, i.e. number of triangles, is the mean of nine processing runs. Fig. 8 shows a diagram of the time needed for k-d tree generation compared to *reduce*

[1]These dimensions in the 2D range image correspond to azimuth and elevation angles of the used terrestrial laser scanner.

and *merge* steps for these models with different number of triangles. While both aspects have a computational complexity upper bound of $\mathcal{O}(NlogN)$ regarding the number of triangles N, measured times are almost linear regarding N. This is partially due to the fact that only border vertices are considered as potential vertices to link the input meshes. Yet the number of border vertices in overlapping regions usually scales sub-linearly compared to the overall number of vertices in a mesh. K-d tree generation appears to scale worse with growing N than the presented steps do. In order to optimize runtime performance, a faster k-d tree generation could be used, while comparing the gained performance to the probably costlier nearest neighbor queries during the *reduce* and *merge* steps.

IV. Discussion

While many different algorithms to generate a combined triangle mesh from 3D input meshes exist, as described in section I, none are optimized for merging 2D-manifold surface representations without adding any new vertices. Using our presented algorithm, any meta data about vertices, such as detailed information on measurements from the input sensor, is preserved throughout the fusing process.

Removal of redundant faces is achieved through easily configurable quality criteria which can be custom tailored to suit different sensors and requirements on the data during further processing steps. When fusing data from terrestrial laser scanners, it is advisable to let the quality measure reflect decreasing accuracy with growing distance of the scanned surface from the sensor. Another possibility is to penalize certain types of triangles, or even triangle configurations, that can lead to problems in following processing steps such as generalization.

Gaps between the input meshes, once redundancy is removed, are closed using two different methods. The first finds an initial face connecting the two meshes. The best triangle in this step is the one which connects the two vertices which are closest to each other. Using two-way nearest neighbor queries guarantees to find this triangle. This generates the best suited starting edges for the second method, *stitching*. This process is an approximation to quickly close gaps once good initial edges have been found, using only information on mesh topology. Due to this, merging meshes with a higher vertex density in overlapping regions gives better results. If triangle size differs significantly between the two input meshes, this approach may create very pointed triangles. This is often undesired, as it can lead to numerical instability. For angles and lengths near zero small perturbations result in large normal vector variations. Care should be taken not to increase the parameter *maxGapSize* past the desired level of detail, as it can lead to actual holes being filled and loss of object features such as 3D corners.

Our approach favors the local quality of the measurements over the global quality of the topology. This is achieved by using very flexible quality criteria. The overall performance is enhanced by re-using large areas of the original meshes. Therefore the computational expensive re-meshing occurs only in small local areas. Also the use of topological information instead of k-d tree queries during the *stitching* step improves runtime performance at the cost of optimal mesh generation. The topology queries can be answered very efficiently by the use of a half-edge datastructure [12].

The algorithm was implemented and used in a production system to fuse data from over 600 terrestrial laser scans of the interior of a large building, with a combined total of $>$60 billion measured points. Individual scans were merged sequentially using the presented mesh fusion. As a result, final vertices retained can be exactly matched to their original measurements. One application of this is to estimate vertex accuracy using distance from the scanner during measurement.

It seems viable to examine the use of k-d trees for nearest neighbor queries in the algorithm. Since most queries are similar to other queries in their vicinity, a first step would be to explore caching of the query results. Furthermore, analysis of query patterns may yield an improved heuristic for a k-d tree better suited for mesh fusion. One aspect could be to take the parameters *minGapSize* and *maxGapSize* into account. Finally, such an analysis might conclude that a different datastructure could improve performance, perhaps even reduce the computational complexity.

References

[1] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 163–169, August 1987.

[2] R. Shu, C. Zhou, and M. S. Kankanhalli, "Adaptive marching cubes," *The Visual Computer*, vol. 11, pp. 202–217, 1995, 10.1007/BF01901516.

[3] B. C. Bloomenthal, J., Ed., *Introduction to Implicit Surfaces*. San Francisco: Kaufmann, 1997.

[4] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust moving least-squares fitting with sharp features," *ACM Trans. Graph.*, vol. 24, pp. 544–552, July 2005.

[5] I. Tobor, P. Reuter, and C. Schlick, "Multi-scale reconstruction of implicit surfaces with attributes from large unorganized point sets," in *Shape Modeling Applications, 2004. Proceedings*, june 2004, pp. 19 – 30.

[6] X. Jin, J. Lin, C. C. Wang, J. Feng, and H. Sun, "Mesh fusion using functional blending on topologically incompatible sections," *The Visual Computer*, vol. 22, pp. 266–275, 2006, 10.1007/s00371-006-0004-8.

[7] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2004, 10.1023/B:STCO.0000035301.49549.88.

[8] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 311–318.

[9] T. P. Kersten, K. Mechelke, M. Lindstaedt, and H. Sternberg, "Methods for geometric accuracy investigations of terrestrial laser scanning systems," *PFG Photogrammetrie, Fernerkundung, Geoinformation*, vol. 2009, no. 4, pp. 301–315, 10 2009.

[10] J. R. Shewchuk, "Delaunay refinement mesh generation," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997, available as Technical Report CMU-CS-97-137.

[11] B. Choi, R. Komuravelli, V. Lu, H. Sung, R. L. Bocchino, S. V. Adve, and J. C. Hart, "Parallel sah k-d tree construction," in *Proceedings of the Conference on High Performance Graphics*, ser. HPG '10. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2010, pp. 77–86.

[12] L. Kettner, "Using generic programming for designing a data structure for polyhedral surfaces," *Comput. Geom. Theory Appl.*, vol. 13, no. 1, pp. 65–90, May 1999.