Data Handling in Large-Scale Surface Reconstruction

Thomas Wiemann[1], Marcel Mrozinski[1], Dominik Feldschnieders[1], Kai Lingemann[2], and Joachim Hertzberg[1,2]

[1] Knowledge Bases Systems Group, Osnabrück University,
Albrechtstr. 28, 49076 Osnabrück, Germany
`twiemann|jhertzberg@uni-osnabrueck.de`
`http://www.informatik.uni-osnabrueck.de/kbs/`

[2] DFKI Robotics Innovation Center, Osnabrück Branch,
Albrechtstr. 28, 49076 Osnabrück, Germany,
`firstname.lastname@dfki.de`

Abstract. Using high resolution laser scanners, it is possible to create consistent 3D point clouds of large outdoor environments in short time. Mobile systems are able to measure whole cities efficiently and collect billions of data points. Such large amounts of data can usually not be processed on a mobile system. One approach to create a feasible environment representation that can be used on mobile robots is to compute a compact polygonal environment representation. This paper addresses problems and solutions when processing large point clouds for surface reconstruction.

Keywords: surface reconstruction, 3D laser scanning, polygonal mapping

1 Introduction

Laser scanners are commonly used to create 3D representations of outdoor scenes. When mounted on mobile systems, it is possible to measure large environments in short time. For robotic applications like localization, point clouds are an inefficient representation, since the collected 3D points are not topologically connected and even at high resolution, they do not deliver a continuous surface representation, just samples. Another problem is the large number of points that need to be collected and processed. Terrestrial laser scans are able to collect billions of points in a single scan. Therefore it is practically impossible to process such amounts of data on mobile systems.

A method to overcome these drawbacks is to compute polygonal meshes from the input data. These data structures deliver topologically connected continuous surface representations. Using suitable mesh optimization algorithms, very compact and geometrically correct environments representations are generated. The elements in such meshes, usually triangles, can be organized in efficient search structures like AABB trees for ray tracing and collision detection. Together with automatically generated textures [10], the geometric information of the laser scans can be fused with color information.

Polygonal meshes of large environments are typically processed offline due to hardware constraints. But the generated environment representations can be used on-board a mobile robot at least to represent the static parts of a scene. In this paper we present efficient methods to generate such meshes in short time. Emphasis of the presented procedures lies on minimizing the computation time and memory consumption during the reconstruction process.

The remainder of this text is organized as follows: First we will present the main features of state of the art reconstruction algorithms. In this section we will also discuss common problems that occur during reconstruction. The next section will present several solutions to these problems that are implemented in the Las Vegas Surface Reconstruction Toolkit (LVR) [13]. Section 4 will present experimental results, the final section concludes.

2 Reconstruction Methods

Surface reconstruction methods can roughly be categorized into methods based on direct triangulation [1, 5] and algorithms that compute a mathematical surface representation [4, 6, 8], commonly an iso surface, that is then reconstructed using suitable polygonalization algorithms like Marching Cubes [9]. An extensive evaluation of the features of the different algorithms can be found in [12]. Based on the results presented in [12], this paper focuses on Marching Cubes-based reconstructions. The bottleneck in terms of run time for this algorithms is the computation of surface normals to approximate an iso surface: Each point has to be associated with a normal that represents the orientation of the surface that point belongs to. Once the iso surface is computed, it is polygonized using a voxel grid. The resolution of this voxel grid mainly determines the amount of memory that is needed to compute the approximation. If the extent of a scene is large, as is the case in outdoor scenarios, the achievable resolution is limited due to memory constraints. For large environments even optimized triangle meshes may contain too many elements to be stored in the working memory of a mobile system. Therefore a smart management of the created data is needed, to swap relevant chunks of data from the complete map into the system's RAM.

3 Large-Scale Data Handling in Surface Reconstruction

In this section we present approaches to handle the collected point cloud data efficiently to overcome several of the problems sketched in the previous section.

3.1 Optimized Data Structures

The Marching Cubes Algorithm needs a discrete voxel grid of predefined resolution to compute a polygonal approximation of an iso surface. Usually octrees or hash-based approaches are used as grid representations. In LVR the default structure to represent occupied voxels is hash-based: Each data point is shifted to a discrete virtual grid, i.e., for a given voxel size v the indices that represent the affected voxels are calculated by:

$$i = \lfloor \frac{x - x_{min}}{v} \rfloor, \qquad j = \lfloor \frac{y - y_{min}}{v} \rfloor, \qquad k = \lfloor \frac{z - z_{min}}{v} \rfloor$$

where x_{min}, y_{min} and z_{min} are the minimal occurring coordinates values in the respective direction. Based on these indices and the known dimensions dim_x, dim_y and dim_z a hash function $\mathcal{H}(i, j, k)$ is defined to access the voxels stored in a hash table:

$$\mathcal{H}(i, j, k) = i \cdot dim_x \cdot dim_y + j \cdot dim_y + k$$

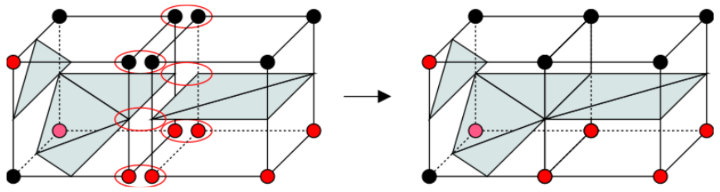


Fig. 1. Shared corners of voxels and triangle vertices in regular voxel grinds. Vertices that have already been created in adjacent cells have to be identified.

The main benefit of this representation is that the neighboring voxels of a given cell can be accessed easily and in constant look-up time by increasing or decreasing the grid indices. In LVR's Marching Cubes implementation, the neighbor relations are used to determine redundant vertices. In each cell it is locally checked if a needed triangle vertex was already interpolated in a neighbor cell (cf. Fig. 1). If such a vertex is found, it is reused by storing a reference to the already created vertex. By checking for redundant vertices, the meshes are topologically sound, which is important for later mesh optimization.

However, hash tables have a noticeable memory overhead. As presented in [3], octrees can be implemented very efficiently to represent voxels. The presented pattern reduces the needed memory per voxel significantly compared to other available implementations. We adapted and extended this structure in such a way, the created octrees can be used for Marching Cubes surface reconstruction. To be compatible with LVR's reconstruction pipeline, we implemented the following additional functionalities:

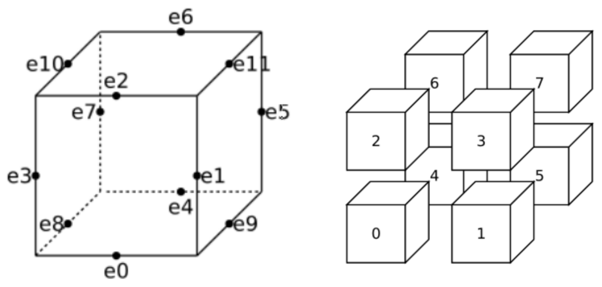
look-up of neighbor voxels. To search for redundant vertices, we added a search functionality similar to the hash structure described above.

Parallelization. In LVR the vertex interpolation for each cell is done in parallel using OpenMP. To achieve optimal performance, we made all search operations thread-safe.

Dynamic addition of new cells. LVR detects holes in the grid structure to fill up missing triangles on continuous surfaces. Therefore the new octree structure must support the inclusion of new cells at predefined positions in the voxel grid.

The first problem can be solved by exploiting the encoding presented in [3]: Each node contains a bit pattern that encodes whether a child exists and, if it does, whether it is a leaf. Every bit refers to a sub-octant with known relative position. A look-up table encodes for each leaf, where (i.e., in which child of the current leaf's parent node) to search for an existing neighbor.

If we do not find a leaf in the parent's node, we check recursively the next higher level, yet the local relations remain the same, hence we can search at predefined positions in the surrounding leafs. An example for the used encoding is shown in Fig 2. In this figure the search for possible neighbors containing the needed vertex is given. Generally, the internal positions according to the numeration can be expressed as well, but is left out here for sake of simplicity. Using this information, we can efficiently search for already created vertices without memory overhead, since all needed information can be computed from the internal encoding.



Vertex	Neighbors		
e_0	4	2	6
e_1	3	5	7
e_2	0	6	4
e_3	1	5	7
...			

Fig. 2. Calculation of look-up tables for voxel neighbor search. Given numeration of possible vertex positions (left), and relative voxel positions (middle), a table for possible duplicate vertex positions can be pre-computed. In the example the first row in the column means: Vertex e_0 can possibly be found in the relative neighbors 4, 2 and 6 in the given representation.

For parallel reconstruction, every child node in the octree is evaluated in parallel. Since the number of leaves is usually high, we use a thread pool pattern the reduce thread generation and termination overhead. Except checking for joined vertices, reconstruction on each leaf can be performed independently. Critical is the integration of new triangles into the global mesh representation: Insertion of vertices and triangles is done via the interface functions `addVertex` and `addTriangle`. Memory access in these functions is therefore protected using mutexes. Using these extension, we were able to achieve comparable run time

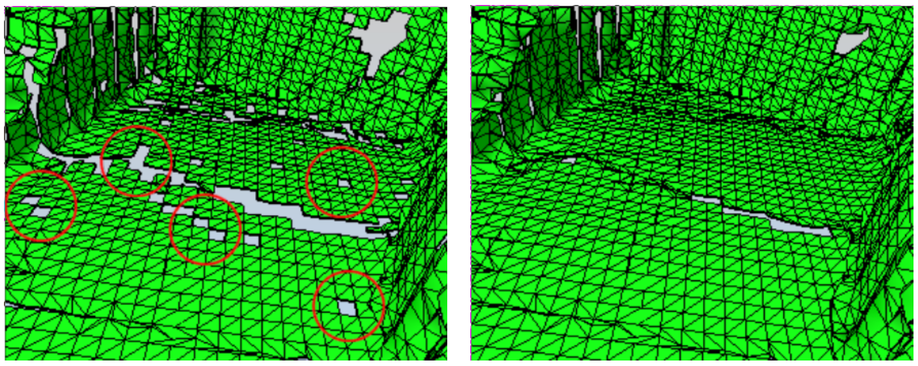


Fig. 3. Reconstruction results with and without grid extrusion. The obviously missing cells in the left image (marked red) are filled up for a continuous representation.

performance to the hash-based approach whilst reducing the memory overhead significantly as the evaluation in Section 4 demonstrates.

To get a complete reconstruction in sparse areas, LVR extends the grid around borders by additional surrounding cells, since in sparsely scanned areas it may happen that the grid shift of the data points excludes single voxels. If these empty voxels are surrounded by existing cells, we assume that we can fill up these holes. We call this grid extrusion. To implement this, we simply have to check for all leaves, if the respective neighbors are present using the neighborhood encoding. If a border leaf is found, the missing cells for extrusion are inserted into the tree. The influence of this extrusion to sparse scan data is shown in Fig 3. The kD tree-based segmentation delivers chunks of equal size with variable densities but locally suitable distributions to ensure stable normal estimation results. In Section 4 we will discuss the influence of the distribution on the computed normals.

3.2 Distributed Normal Calculation

Surface normal estimation is easy to parallelize, since it follows the SIMD scheme. In multi core systems, the process can be parallelized using a framework like OpenMP. For mapping of large areas it may be beneficial to outsource extensive calculations from the on-board hardware, especially if further infrastructure for data management is provided externally. In surface reconstruction the computationally most expensive part is the calculation of surface normals for the collected laser scan data. LVR therefore provides an MPI-based distributed normal estimation to distribute the data to a computing cluster that can be set up using standard desktop computers. The registered point clouds are divided into chunks of equal size that are sent from a master to several clients that perform the normal estimation. The calculated normals are then sent back to this master and integrated into the reconstruction pipeline.

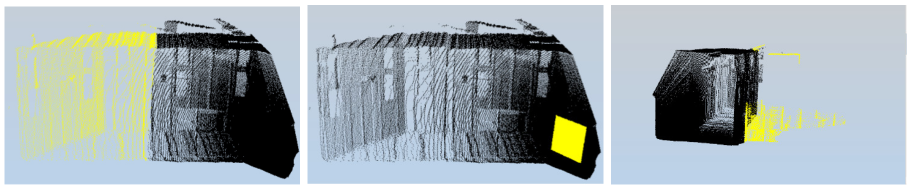


Fig. 4. Examples of data packages created from the kD tree (yellow). All consist of a similar number of points. Left: Region with low point density. Middle: Region with high point density. Right: Points highly distributed within the chunk.

The main challenge in this approach is to generate the data packages that are sent to the clients. To ease the scheduling for data distribution, these chunks should be of equal size. Furthermore, to produce accurate results it has to be ensured that points are clustered to chunks that contain points from the same surface. Therefore a randomized selection is not suitable. Octrees generate leaves of equal size, but it is not ensured that the number of points represented in them is more or less constant. We implemented a subdivision based on kD trees. The split rule we used is to sub-divide a leaf alternately along the longest mean axes of the point coordinates if a maximum number of points is reached. The maximum point criterion ensures that most chunks are filled up to the predefined chunk size. The alternating split along the mean axes ensures that points are clustered locally, which is required for normal estimation. To prevent splits that may produce undesirable point configurations, e.g., leaves with too few points or too low local density, we try to detect such configurations by analyzing the shape of regions covered by the leave points. If the bounding box of the stored points becomes too narrow, we shift the split axis until a reasonable density and point distribution is found. Some examples of the resulting local packages are shown in Fig. 4.

After the packages are computed, they are serialized and sent to the MPI clients which compute the normals. Since the results are usually not returned in the same order as they were sent, all points and computed normals are indexed with a unique ID to correlate the computed results with the point cloud data. After all normals are calculated, we globally average all normals with k neighbors to smooth the data.

3.3 Data Storage and Distribution

For safe navigation, robots need a suitable navigation map that contains all relevant obstacles. However, for large-scale applications even a reduced polygonal representation will usually be too large to be stored in the local memory of a robot. Real world scenarios are usually dynamic. Hence the computed maps have to be updated dynamically. We propose to use a Geometric Information System (GIS) – here we use PostGIS – to overcome these problems. PostGIS provides infrastructure to dynamically insert and query geometric information via a C++ interface. The computed maps are stored in a central data base. The needed navigational maps for different robots can be requested on demand, based on the robot's current position. This way, the size of the navigation maps can be limited according to the robot's memory size. If the robot leaves a known region, it can request a new map from the server. The complete structure of our system is shown in Fig. 5.

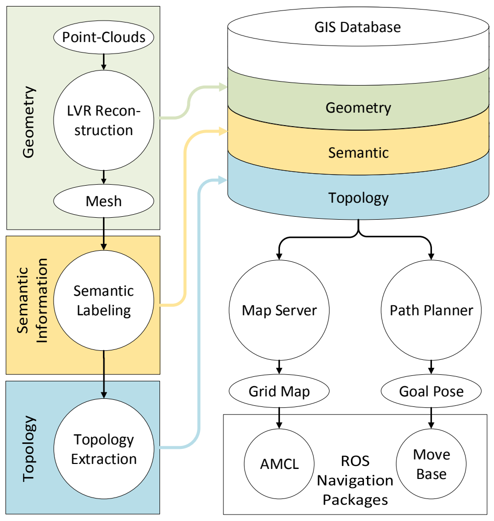


Fig. 5. Integration of the GIS into mapping and navigation

The incoming point clouds are processed using LVR to create an optimized polygonal mesh that is stored in the GIS database. In a post-processing step the meshes can be annotated with semantic information. This information is either computed automatically, e.g., in terms of walls, floors and ceilings in buildings [14], or driveable areas via clustering of planar patches in outdoor applications. If the stored map consists of several parts, the topological connections between single maps can be stored as well (e,g., elevators in a multi story building). More details concerning the semantic mapping system are presented in [2]. The main benefit of the GIS is that all relevant data for robotic task planning are stored in one single data base. The needed navigation maps are computed by clients that query the data base and fuse semantic information with geometric information into a ROS navigation map. GIS systems are able to handle large amounts of polygonal data. Hence it is possible to generate online maps of suitable size for different robots from a single data source.

The navigational maps are generated with an approach similar to the one presented by Hornung et al. [7]: Taking the geometry of the robot into account, we define several horizontal layers. All relevant obstacles for the robot's parts in each layer are projected into different 2D collision maps that are used for path planning. For instance, with this approach we are able to move the base of our robot under a table to bring its robotic arm closer to the working area.

4 Experimental Results

This section describes an evaluation of the methods presented above. First, we evaluate our octree-based reconstruction. Second, we demonstrate the speed up

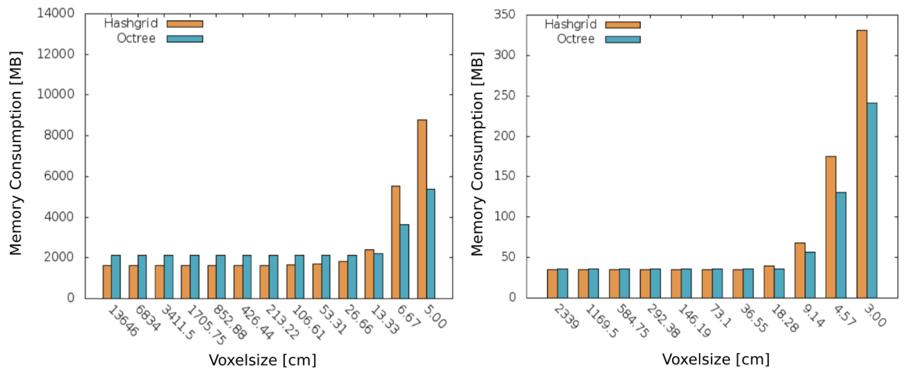


Fig. 6. Comparism of the memory consumption for a high resolution laser scan (18 million points) and a laser scan taken with a tilted laser scanner.

in normal estimation using the distributed approach. Third, we demonstrate an application example of automatic construction of navigation maps from data stored in the GIS data base.

4.1 Evaluation of Octree-Based Reconstruction

First, we compared the memory consumption of the octree-based reconstruction with the hash-based reconstruction. The results are presented in Fig. 6. We tested the two reconstruction methods on two different data sets: A high resolution laser scan consisting of 18 million points and a laser scan taken with a tilting SICK laser scanner with about 300.000 points. As the amount of memory needed theoretically increases cubically with the resolution, we varied the voxel sizes in the evaluation. For relevant voxel sizes between 5 cm and 10 cm the octree reconstruction saves about one third of total memory. Interestingly, for lower voxel sizes the memory consumption is nearly constant, with the octree needing more memory than the hash grid. This is due to the static internal data structures that are needed to manage the tree linking or the hash table respectively. For high resolution reconstructions the memory overhead becomes negligible and the octree-based reconstruction outperforms the hash-based one.

A comparism of reconstruction times between the hash grid structure and octree is shown in Fig 7. It is obvious that the octree reconstructions do not scale as well as the hash-based reconstruction with increasing number of threads. This is due to the fact that we have to search recursively in the parent nodes for neighbor voxels which implies that searches will have different run time, which in turn makes thread scheduling more difficult. Furthermore, the recursive search has to lock access to shared vertex data, which also reduces the performance gain. In general, as a rule of thumb, the octree needs about 25% more time than the hash grid-based reconstruction. To sum up: The octree reduces the memory overhead significantly for small voxel sizes at the cost of run time.

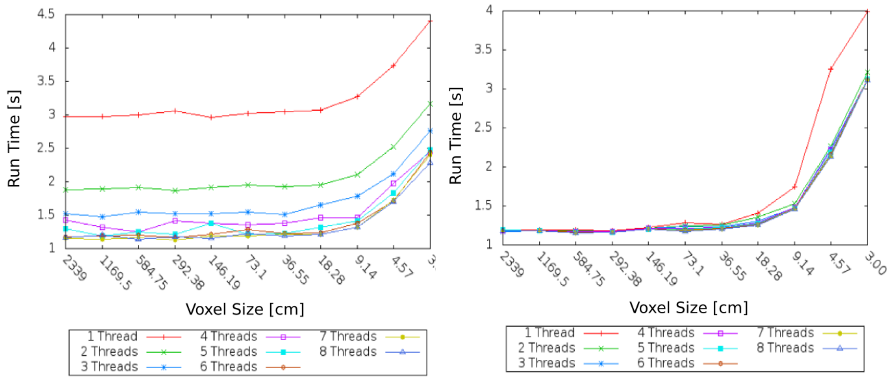


Fig. 7. Comparism of reconstruction run time on a SICK LMS laser scan between hash grid (left) and octree (right) for different voxel sizes in parallel reconstruction with multiple threads.

4.2 Evaluation of the Distributed Normal Computation

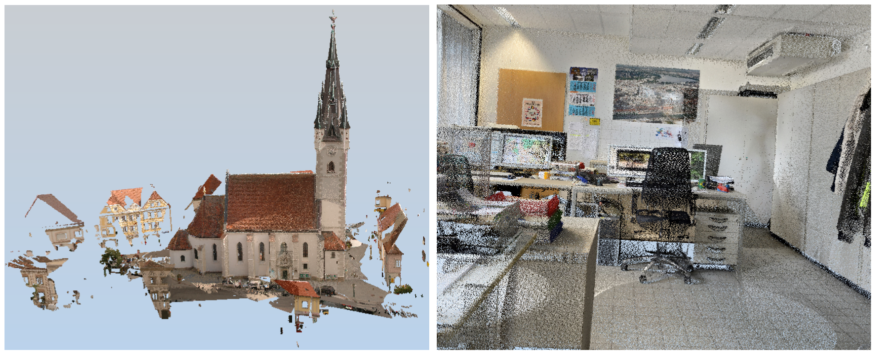


Fig. 8. The two data sets used for evaluation of distributed normal calculation. Left: outdoor data set consisting of 17 million points, right: indoor data set with 30 million points.

To evaluate the MPI-based distributed normal estimation, we used two different large-scale data sets that represent different geometries. One data set was an outdoor scan of on outdoor scene consisting of 17 million points, the second one was an indoor scan of several offices, consisting of 30 million data points, cf. Fig. 8. First, we evaluated the scaling ability of the presented approach on an pool of up to 25 standard Intel Core i3 desktop computers that were connected through our public university network. We deliberately used no closed setup to demonstrate that such clusters can be easily set up on demand using

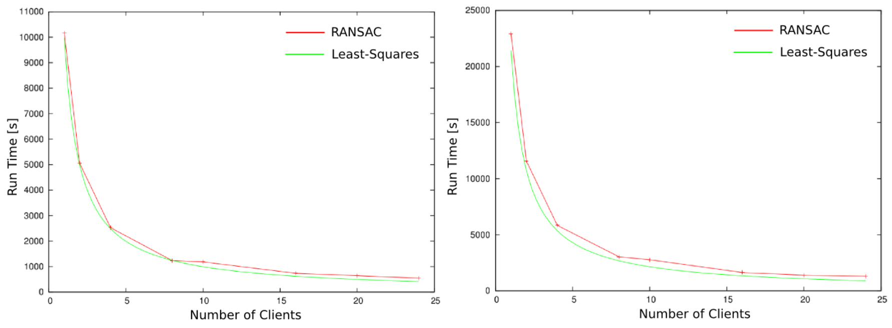


Fig. 9. Run time of distributed normal estimation with raising number of clients.

our software. Fig 9 shows the results of the experiments for two different normal estimation methods, namely regression plane calculation via least squares fit and RANSAC.

Both approaches scale very well with the number of available clients. Without distribution, the indoor data set needs about 25 minutes to compute, the outdoor data set 5 Minutes. The graph shows that the run time nearly halfs with doubled number of available clients. Using 24 clients, the reconstruction times drop down to 5 minutes indoor and 2 Minutes outdoor. That is a relative speedup of 17.7 on the indoor data and 18.6 on the outdoor data. This figures include data transfer and management overheads.

The evaluate the influence of the sub-division into data packets by the host process, we compared the estimated normals via MPI with the normals computed by a single process that had the complete point cloud available. Fig. 10 shows the results for RANSAC-based normal estimation. The histogram shows the number of normals whose orientation had a certain deviation of a certain degree, i.e., many matches with little angular deviation means a good correspondence to ground truth. On the indoor data set 75% of the computed normals were within

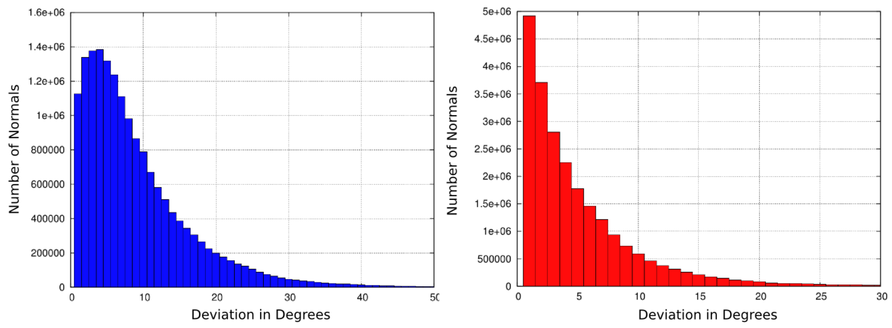


Fig. 10. Comparism of normal deviations for both data sets.

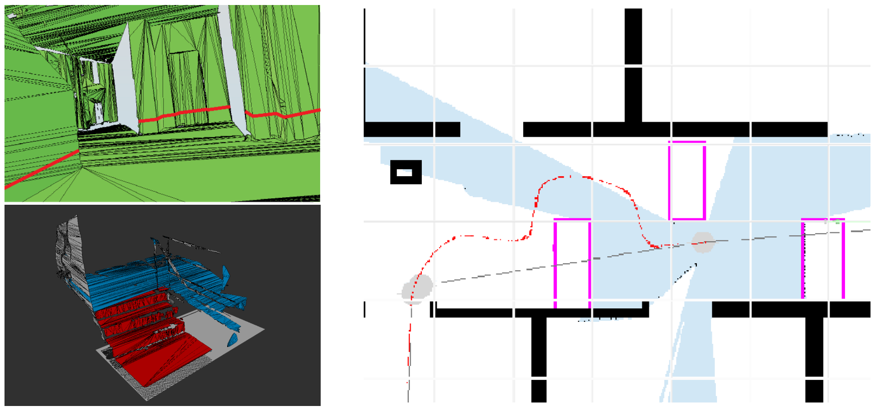


Fig. 11. Navigation map generation. Geometry is extracted from the reconstruction (top left) and fused with semantic annotation (bottom left) to generate individual navigation maps for two different sized robots (gray for a small robot, red for a large robot).

an error margin of 5°, 82 were within this limit the outdoor data set. Since LVR's Marching Cubes implementation averages the signed distance to the iso surface these deviation are negligible and had no significant impact on the mesh quality.

4.3 Application Example for Dynamically Computed Navigation Maps

Fig. 11 shows an application example of the navigation map generation presented in [2] for two different robots using a shared GIS data base. Via a parallel cut through the reconstruction according to the height of the robots, individual navigation maps are generated from a common 3D data source. The semantic classification of polygons, e.g., undrivable stair cases, can be integrated into the navigation map. The application example shows the calculated paths for a large and a small robot under the presence of tables as presented in [11]. The large robot computes a path around the tables according to its individually generated collision map, whilst the small robot can drive under the tables, that are not present as obstacles in its collision map. This example shows that the storage of 3D geometric information and semantic annotation in GIS can help to reuse the collected and post-processed data on different platforms.

5 Conclusion

This paper presented a family of approaches to improve the processing and management of data from large-scale laser scans in the context of polygonal reconstruction for mobile robots. The introduction of octrees for reconstruction can save memory in the presence of a huge amount of voxels. MPI-based data distribution can speed up the required normal estimation significantly. The central storage of the processed data in a central data base can be beneficial, if the data is reused on different platforms. Furthermore, the representation of semantic information in a relational data base can be easily fused with geometry using PostGIS.

References

1. N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proceedings of the 6th ACM Symposium on Solid Modeling and Applications (SMA '01)*, pages 249–266, New York, NY, USA, 2001. ACM.
2. H. Deeken, S. Pütz, T. Wiemann, K. Lingemann, and Hertzberg J. Integrating semantic information in navigational planning. In *Proc. ISR/Robotik 2014*, 2014.
3. J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 3(1):2–12, 2012.
4. G. Guennebaud and M. Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, 2007.
5. H. Edelsbrunner and E.P. Mücke. Three-Dimensional Alpha Shapes. *ACM Transactions on Graphics*, 13:43–72, Jan. 1994.
6. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2), 1992.
7. Armin Hornung, Mike Phillips, Edward Gil Jones, Maren Bennewitz, Maxim Likhachev, and Sachin Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Proc. ICRA 2012*, pages 423–429, 2012.
8. M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing (SGP '06)*, pages 61–70. Eurographics Association, 2006.
9. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH*, 1987.
10. K.O. Rinnewitz, T. Wiemann, K. Lingemann, and J. Hertzberg. Automatic creation and application of texture patterns to 3d polygon maps. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3691–3696, 2013.
11. S. Stiene and J. Hertzberg. Virtual range scan for avoiding 3d obstacles using 2d tools. In *Proc. ICAR 2009*, 2009.
12. T. Wiemann, J. Hertzberg, K. Lingemann, and H. Annuth. An evaluation of open source surface reconstruction software for robotic applications. In *Proceedings of International Conference On Advanced Robotics (ICAR 2013)*, 11 2013.
13. T. Wiemann, K. Lingemann, A. Nüchter, and J. Hertzberg. A toolkit for automatic generation of polygonal maps – las vegas reconstruction. In *Proceedings of the 7th German Conference on Robotics (ROBOTIK 2012)*, pages 446–451, München, 2012. VDE Verlag.
14. T. Wiemann, A. Nüchter, K. Lingemann, S. Stiene, and J. Hertzberg. Automatic construction of polygonal maps from point cloud data. In *IEEE International Workshop on Safety Security and Rescue Robotics (SSRR 2010)*, 2010.