Automatic Map Creation for Environment Modelling in Robotic Simulators

Thomas Wiemann, Kai Lingemann and Joachim Hertzberg

Knowledge Based Systems Group
Osnabrück University
Albrechtstr. 28, Osnabrück, Germany
twiemann@uni-osnabrueck.de

DFKI GmbH
Robotics Innovation Center Bremen
Albrechtstr. 28, Osnabrück, Germany
{kai.lingemann|joachim.hertzberg}@dfki.de

KEYWORDS

Environment Modeling, Model Generation, Robot Simulators

ABSTRACT

This paper presents an approach to automatically create polygonal maps for environment modeling in simulators based on 3D point cloud data gathered from 3D sensors like laser scanners or RGB-D cameras. The input point clouds are polygonalized using a modified Marching Cubes algorithm and optimized using a pipeline of mesh reduction and filtering steps. Optionally color information from the point clouds can be used to generate textures for the reconstructed geometry.

INTRODUCTION

Simulators play an important role in the development of algorithms for robotic applications. They allow to test new procedures on synthetically generated data before they are evaluated on the real hardware. In simulators, the emulated data is generated based on physical models and descriptions of the used hardware and the environment the robot is interacting with. Usually polygonal models are used to represent robot and environment. Manual modeling is a time consuming and tedious job, especially for complex geometries.

With the rapid development of 3D scanning technology, real world objects can be scanned faster, more precisely and at higher resolution. State of the art laser scanners are able to acquire in the order of a hundred million points with one single scan. Besides laser scanners, 3D or RGB-D cameras like Microsoft Kinect can be used to digitize complex scenes using specialized SLAM methods like (Henry et al., 2010). Compared to a 3D laser scan, an RGB-D point cloud lacks density and provides a low opening angle; however, as the cameras deliver up to 30 frames per second, the data accumulates to very large registered point clouds as well.

For the use in simulation raw point clouds are clumsy for several reasons. First, large environments would require a huge amount of points to be represented, and, consequently, much memory is needed just to store the raw data. This problem gets worse if search data structures like *kd* trees are used to optimize queries like nearest neighbor searches, needing additional memory. Second, even with optimized search structures, run time is critical. Third and maybe most important: Point clouds, by definition, only contain discrete scan points. Even dense, high resolution point clouds are no substitute for continuous surface representations that are needed to use them as environment models for simulators like Gazebo (Koenig and Howard, 2004).

A common solution to overcome these disadvantages is to compute a polygonal mesh representation that approximates the point cloud data. Being a standard data structure for object modeling in computer graphics, optimal polygonal approximations are memory efficient, and efficient algorithms for rendering and ray tracing are available. In the context of mobile robotics, polygonal environment maps offer great potential for applications ranging from usage in simulators, virtual environment modeling for tele operation to robot localization by generating virtual scans via ray tracing. However, creating polygonal environment maps based on laser scan data *manually* is a tedious job, hence quite a number of automatic surface reconstruction algorithms have been developed over the past years.

The available surface reconstruction algorithms are usually optimized for special use cases and input data, but generally deliver acceptable results. However, when using them in robotic applications, several practical issues arise. Most general surface reconstruction procedures would represent sharp features poorly and produce more triangles than needed to approximate a surface, especially on planes. Other problems occur when the input data is incomplete due to scan shadows that yield holes in the reconstructed meshes. Furthermore, many of reconstruction algorithms like Power Crust (Amenta et al., 2001) and Poisson Reconstruction (Kazhdan et al., 2006) rely on closed geometries, which is generally not the case for arbitrary environments.

In this paper we present a software we call the Las Vegas Surface Reconstruction Toolkit (LVR) (Las Vegas Surface Reconstruction) (Wiemann et al., 2012) to automatically create polygonal environment representations from point cloud data that can be used for the simulation of typical environments in robotic applications. The reconstruction procedure basically consists of three steps: Initial mesh generation using Marching Cubes, mesh optimization and texture generation. All steps are performed automatically without user interaction. The software can use the results of the provided, efficient Marching Cubes implementations as input, or, alternatively, can be used together with other surface reconstruction implementations like the ones in CGAL and PCL (Rusu and Cousins, 2011). We show the practical usability of the software by demonstrating the whole re-

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

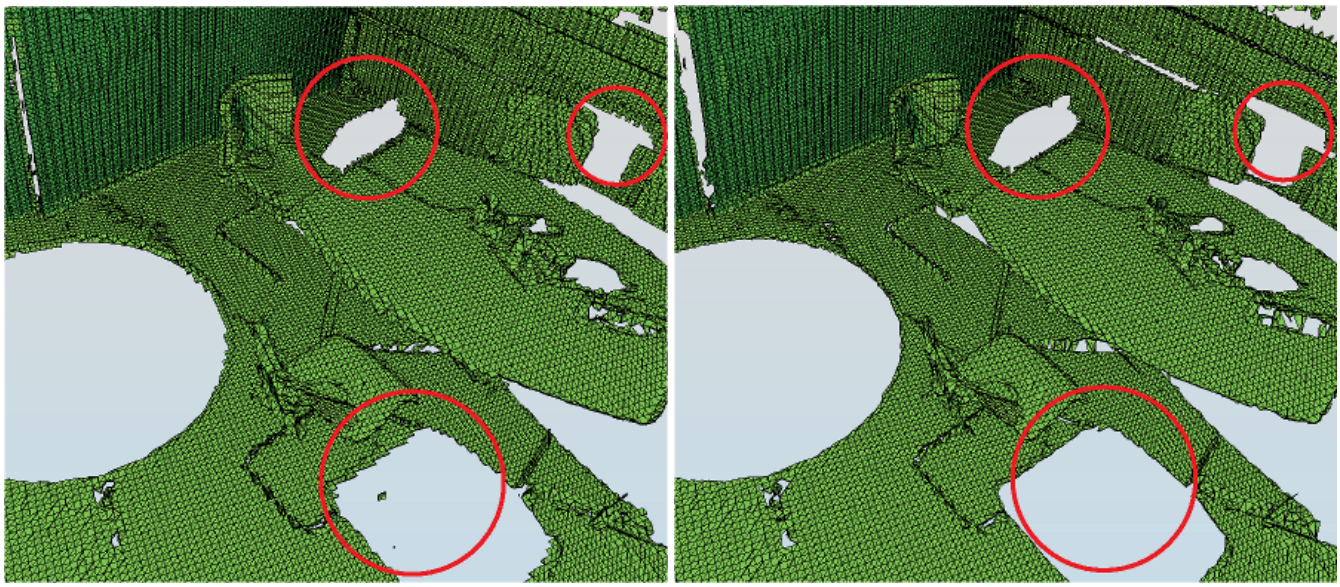


Fig. 1. Comparison of standard Marching Cubes (left) and Planar Marching Cubes (right). The contours of the marked areas show considerable discretization artifacts when the original Marching Cubes algorithm is used. Planar Marching Cubes on the other side produces smooth contours.

construction process from point cloud to a textured polygon map that is loaded into Gazebo and used to generate simulated laser scan data.

The remainder of this paper is organized as follows: The next section presents the state of the art in polygonal surface reconstruction from point cloud data. The next part presents the implemented mesh generation and optimization techniques followed by an application example. The last section concludes.

RELATED WORK

Automatic construction of meshes from point cloud data has received much interest in computer graphics. The defacto standard method to generate triangle meshes is the Marching Cubes algorithm (Lorensen and Cline, 1987), which comes in a number of variants. (Newman and Yi, 2006) provides a comprehensive review. Marching Cubes is an iso surface extraction algorithm, which means, a continuous mathematical scalar field description of the underlying surface is needed. Typically, the iso value of a surface within this field is 0. The first method to approximate a suitable representation for unorganized point cloud data was developed by Hoppe et al (Hoppe et al., 1992). It uses local approximations of so-called tangent planes to represent a signed distance function that defines a zero surface. A variant of this method together with a GPU implementation of Marching Cubes is used in Kinect Fusion (Izadi et al.) to create meshes of Kinect data in real time, but this method is limited to a predefined maximum reconstruction volume. Other methods to create iso surfaces are variants of Moving Least Squares like APSS (Guennebaud and Gross, 2007) and RIMLS (Öztireli et al., 2009), which are both integrated in Meshlab. While there are reconstruction methods based on Marching Cubes, a number of algorithms exist that directly triangulate point cloud data. One example is the "Growing Cell Structures" (Annuth and Bohn, 2010) that uses a neural network together with unsupervised learning. Other direct methods are based on Delaunay triangulations (Devillers, 2002) (Amenta et al., 2001) or generate greedy triangulations like the one implemented in PCL (Rusu and Cousins, 2011). Another successful approach for closed geometries is Poisson Reconstruction (Kazhdan et al., 2006). The drawback of these approaches is that they are sensitive to the quality of the input data, since every data point is used in the triangulation, resulting in uneven meshes in the presence of noise.

Surface reconstruction algorithms usually produce more triangles than strictly needed to represent a surface. Therefore after creating an initial triangulation of the input data, meshes are usually optimized. Most of the respective algorithms rely on calculating the cost of removing vertices or faces (Garland and Heckbert) (Melax, 1998) and iteratively remove the elements that cause the lowest geometric error. (Hoppe et al., 1992) uses a similar approach, but instead of measuring the local error, a global error sum is used to determine the redundant triangles. These methods work well for curved models, but in meshes with many sharp features, the sharp edges will ultimately blur out. The mesh optimization steps presented in this paper aim at optimizing planar patches bounded by sharp edges, to get an improved triangle mesh.

MAP GENERATION

Our map generation procedure consists of three main parts: Mesh generation, mesh optimization and texturing. The details of the single processing steps are described in the following sections.

Initial Mesh Generation

Many robotic environments are mostly planar. This fact has to be taken into account when choosing the mesh generation method. Our experience has shown, that Marching Cubes based approaches deliver the best results for arbitrary geometries since they do not rely on special properties like convexity, can handle holes in the input data correctly and are very fast and memory efficient. For the task at hand

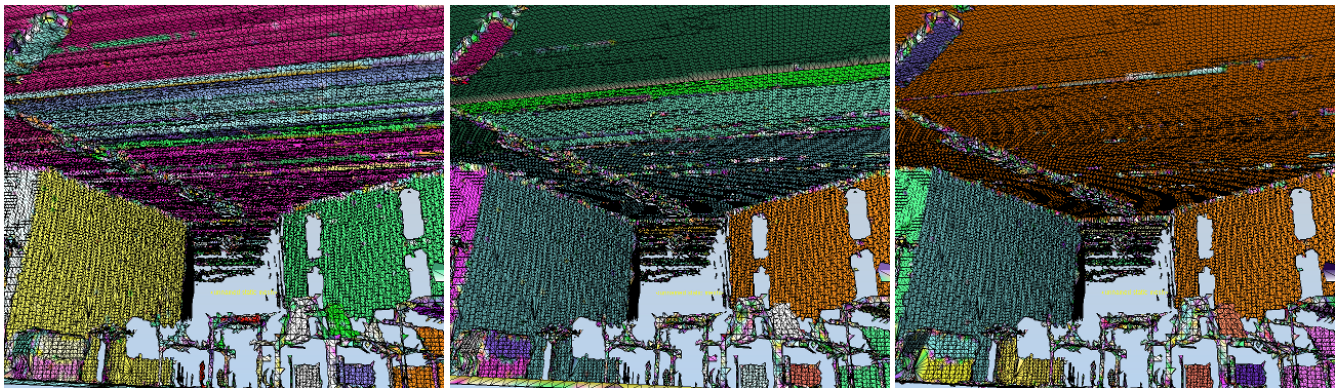


Fig. 2. Region growing based plane detection in the initial reconstructions. Different colors signal different planes. Note that the regions on the ceiling are separated in the first clustering step due to noise (left). Applying the algorithm repeatedly, clusters can be fused (middle) until the whole ceiling plane is finally extracted (right).

we use a modified implementation we call *Planar Marching Cubes*, that is optimized for planar reconstructions. It uses Hoppe's distance function (Hoppe et al., 1992) together with an optimized normal estimation procedure (Wiemann et al., 2012).

The basic idea of Marching Cubes is to calculate the intersection of a surface with a cell within a voxel grid and use precomputed approximation patterns to generate a triangular approximation of the surface's course within the cell. The main problem when approximating planar surfaces is that the classic Marching Cubes approach cannot detect if a surface ends within a cell, due to the fact, that interpolation is only done in one dimension. This results in a noticeable discretization of the reconstruction of planar regions. Our Marching Cubes implementation is optimized to represent the contours of planar objects correctly to create realistic environment models. To model the ending of a surface correctly, we project the created vertices in planar configuration on the nearest data point. This way, we achieve a surface approximation in two dimensions, which results in noticeably better reconstruction, as shown in Fig. 1.

Mesh Optimization

The most noticeable drawback when using Marching Cubes for mesh generation is that the algorithm produces far more triangles than necessary to approximate planar regions, a fact that can also be witnessed in Fig. 1. Since the reconstructions shall be used in simulators, it is mandatory to keep the number of triangles as low as possible to ensure a high update rate of the simulated sensor data. On the other hand, for accurate results, the used polygon map has to be geometrically correct. The Marching Cubes algorithm also is very sensitive to noise, too, resulting in unevenness of the reconstructed planes, which can be seen in the lower left corner of the presented figure. Another problem is the approximation of sharp features. These are usually blurred out due to the used approximation patters. Kobbelt (Kobbelt et al., 2001) describes a technique to reconstruct sharp features, but the used heuristics to identify sharp bends and corners do not work in noisy data. Therefore we implemented several mesh optimization steps to make the reconstruction feasible for the use in simulators.

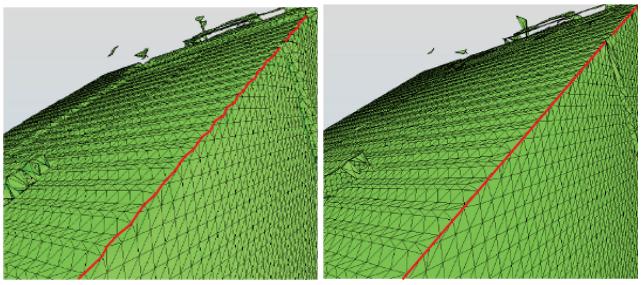


Fig. 3. Optimization of plane intersections. The left image shows the original reconstruction of a bend between two walls (red). The right picture shows the triangle mesh after optimizing the the intersection vertices.

The first step is the optimization of planar regions. To detect planes, we use a region growing based approach. Region-growing is done by checking if the surrounding triangles of an arbitrarily chosen start triangle have a similar surface normal. As long as the normal of a neighbor triangle does not differ more than a user defined threshold from the start triangle, a new search is started recursively from this triangle. This process is carried on, until a bend in the surface is detected. To reduce the noise in the planar regions, we calculate the plane equation using a RANSAC based fit to all vertices and project them into the common plane. Since this process changes the initial geometry, we re-start the region growing afterwards, to fuse regions that may have been separated due to the normal criterion in the first place. This process is repeated until convergence. For the results of this interactive plane detection, see Fig. 2.

After detecting the planes in the mesh, we calculate the exact intersections between them to approximate sharp features between walls, floor and ceiling. To enhance the mesh, we stretch the vertices of triangles near a calculated straight line onto it. The effect on the reconstruction is in Fig. 3.

To reduce the number of triangles in the mesh, we extract and fuse the contour edges of the found planes and re-triangulate them using the OpenGL tesselator. To create an optimal 2D polygon representation of the boundary of a region with a minimal number of vertices we use the Douglas Peucker optimization algorithm (Douglas and Peucker, 1973). Fig. 4 shows an example. The left picture shows the initial mesh. The right picture displays the effect of intersection optimization and mesh reduction via re-triangulation.

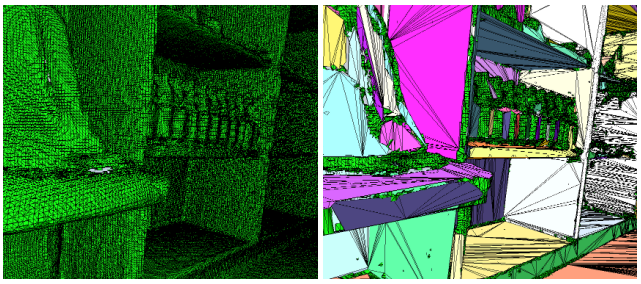


Fig. 4. Mesh reduction via re-triangulation.

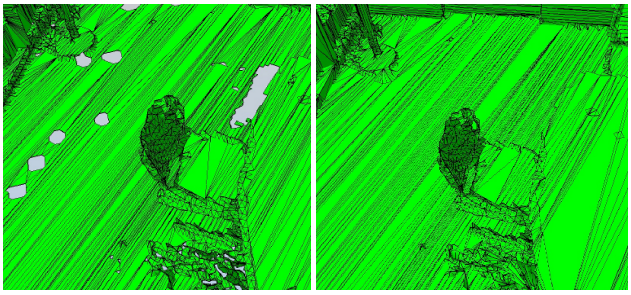


Fig. 5. Exemplary result of the "Hole Filling" function. Before (left) and after (right) application.

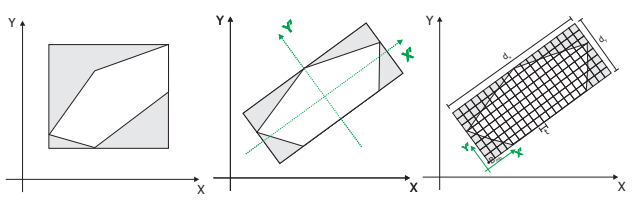
The optimized mesh represents sharp features correctly and the number of triangles in the planar regions is significantly reduced, while curved surfaces (bottles) are still represented with the original geometry.

Another optimization feature is hole filling. The hole filling algorithm performs a contour tracking on the mesh and collects all holes up to a certain size which is given by the number of edges in the contour. In a second step the edges of each hole are collapsed using an edge collapse operation until there are only three edges left per hole. The remaining triangle holes are closed by adding new faces to the mesh which close those holes. Fig. 5 displays an application example.

Besides the presented optimization features, we have implemented a number of several other filters to reduce artifacts due to sensor noise and outliers. A full overview of available features can be found on the LVR website (Las Vegas Surface Reconstruction).

Texture Generation

If the input data contains any information that can be mapped to color values, e.g. RGB information, reflectivity values, thermal data etc., LVR can convert these values into textures for the generated polygons. Texture generation is done by putting a rectangular grid of fixed cell size t over each polygon. The voxelsize of the grid determines the resolution of the texture image. For each cell in the grid, a color value is calculated by averaging the colors of the k nearest points in the given point cloud. To keep the textures small, it is necessary to find a grid alignment that maximizes the used area (cf. Fig. 6). In the presented example, an axis aligned pixel grid would contain a large amount of unused pixels (gray) while an image in the green coordinate system would be significantly smaller (only 70% of the axis aligned version). To determine the best alignment we compute the Principle Component Analysis (PCA) of the polygon vertices. The first two eigenvectors of the result deliver a good estimation for the alignment of the polygon in the global coordinate system, thus we define the pixel matrix by determining the bounding box of the polygon in this local system defined by X' and Y'.



(a)Axis parallel alignment (b)Alignment in local system (c)Calculation of texture pixels

Fig. 6. Finding the right coordinate system for texture generation. In the presented case, an axis aligned pixel grid (a) would result in a lot of unused space (gray). An aligned bounding box would reduce this space (b). The actual pixel map as calculated as shown in (c).

For practical reasons, textures are only calculated for regions above a certain size. Small regions and single triangles are assigned with a single color value to save texture memory and speed up the meshing process. An example of a textured reconstruction is shown in Fig. 7. The left picture shows the input point cloud taken with a Faro laser scanner. The picture in the middle shows the optimized reconstruction. The textured mesh is shown on the right. The large areas on floor, wall and ceiling are textured while the small regions are represented by single colored triangles. It is obvious, that the colored and textured mesh encodes far more information than the pure geometry like the representation of cobblestones on the floor and panels below the ceiling, which are difficult to model geometrically.

The achievable texture quality strongly depends on the quality of the input data. Low point densities will cause blurry textures while dense point cloud data will allow to choose a high resolution for texture generation. For fast rendering the texture resolution shouldn't be too high, since the planar regions can become quite large after reconstruction and the maximum texture size is limited by the graphics hardware.

APPLICATION EXAMPLE

With the methods present above it is possible create high quality textured reconstructions from point cloud data. To prove their usability in simulators we will present an actual application example, where the point cloud data acquired by a 3D laser is automatically processed into a polygonal map that in turn is used in Gazebo to simulate the environment. We will then spawn a robot into the simulation and collect simulated 2D laser scans. The reconstruction will be fully automated without any user interaction or manual editing.

Input Data and Environment Description

As input data we will use a point cloud that was taken at a lecture hall at Osnabrück University called "Reithalle". The data set consists of 6 high resolution laser scans captured with a Leica HDS 6000 laser scanner. The scans were aligned automatically using slam6d (Nüchter et al., 2012) using special markers for initial pose estimation. The laser scanner delivers no color information about the environ-

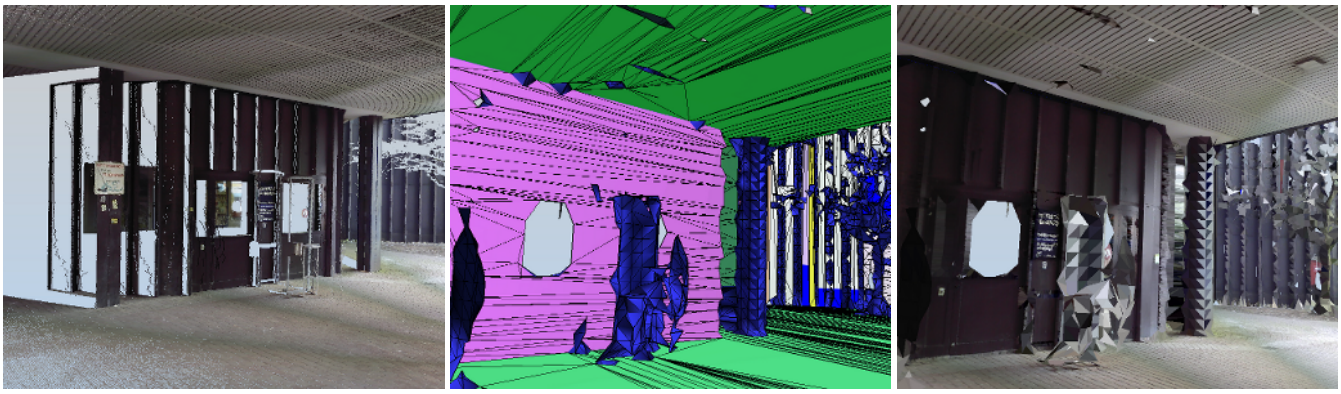


Fig. 7. Textured mesh generation from colored point cloud data: The input point cloud (left), the optimized mesh (middle) and the textured reconstruction (right).

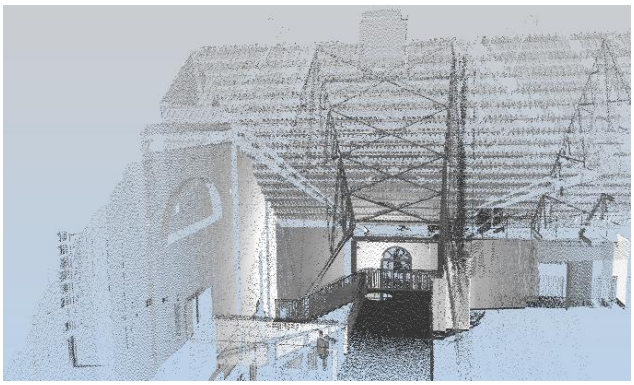


Fig. 8. The input point cloud used in the experiment.

ment, but it returns a reflectance value for each point, that can be used to generate gray scale textures. A rendering of the point cloud is shown in Fig. 8. The environment was chosen to demonstrate the ability to handle large data sets and extensive environments. The geometry of the area is mostly planar and shows drivable surfaces, so that it makes sense to use it in a simulation.

Reconstruction Results

Fig. 9 shows the results of the reconstruction process. The left image shows the initial Marching Cubes reconstruction from the input data. Please note the unevenness on the floor plane due to sensor noise and the smooth transitions between walls and ceilings. The picture in the middle shows the optimized mesh after plane detection, intersection calculation and hole filling. The reconstructed planes show no noise, the intersections are sharp. Due to the plane optimization, some details have been erased from the initial mesh like the door on the right front wall. With texturing this is usually no problem, since these features are represented in the calculated textures, as the picture on the right shows.

The input point cloud contained about 14 million data points. The initial Marching Cubes Reconstruction generated a mesh consisting of 371.640 triangles. This number was reduced down to 98.464 triangles after mesh optimization. The computation time for initial mesh generation was 1:14 minutes on an Intel Core i7 system with 16 GB RAM. Automatic mesh optimization only took 14 s, texture generation another 2:23 minutes. The main bottle neck in the reconstruction is the nearest neighbor search, that has to be performed to estimate the normals for distance function evaluation and to calculate the textures. Currently we are using FLANN (Muja and Lowe, 2009) and OpenMP to parallelize the process. The example demonstrates that our mesh optimization procedure is very efficient and can reduce the number of triangles in the reconstructions significantly. We have tested the optimization with meshes from other mesh generation software (actually the initial mesh in Fig. 4 was created with Kinect Fusion) as well and are currently planning to integrate some algorithms into PCL.

Using the Reconstruction in Gazebo

In our test we have exported the reconstruction into the Collada file format and loaded the mesh into Gazebo. Via the ROS connection of this simulation environment we were able to spawn a model of a Kurt robot (Albrecht et al., 2006) into the reconstructed environment. In our experiments it showed that the rendering front end of Gazebo experienced problems to render the mesh, although the physics simulation was working correctly. The problem here was that the generated environment model was not generated via CSG (Constructive Solid Geometry). Models generated using that technique are always close, i.e. a face is alway pointing towards the spectated. Our meshes on the other hand are single sided, so consequently faces that were not pointing the camera were removed by the renderer ("Back Face Culling"), as can been seen Fig. 10. A quick fix for that problem was to insert each triangle twice with opposite normal directions. We admit that this is merely a hack, but for the time being it solved the rendering problems we experienced.

To check the simulation we navigated the Kurt robot via a ROS node. The model of the robot included the simulation of the robot chassis with differential drive and a Sick LMS200 2D Laserscanner. Visualization of the robot position and the laser scan were done in RVIZ. Our tests proved that the simulation behaved as expected including gravity and collision detection between robot model and environment. Using the double sided reference model, we experienced no performance issues while navigating the robot. An example of an simulated 2D laser scan together with the corresponding geometry is shown in Fig. 12. The left image shows a rendering of the reconstructed model from the

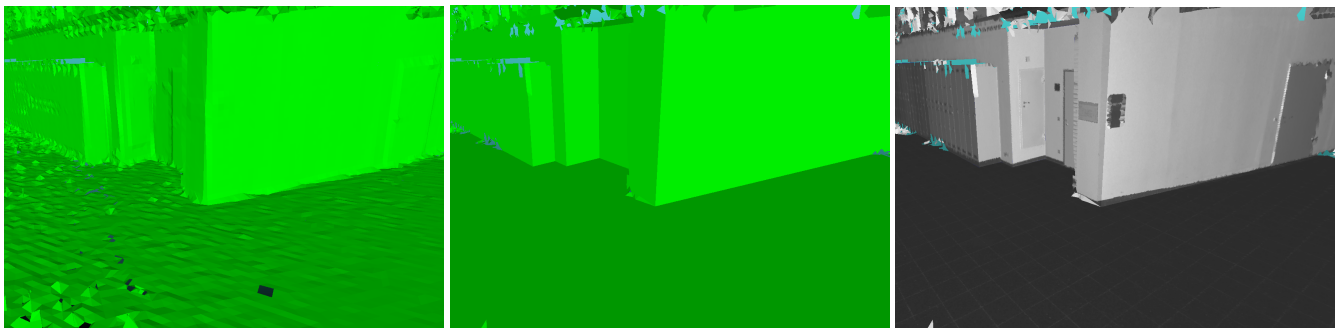


Fig. 9. Reconstruction of the "Reithalle" data set. The images show the initial reconstruction (left), the optimized mesh (middle) and the textured model visualized the reflectance values of the laser scanner.

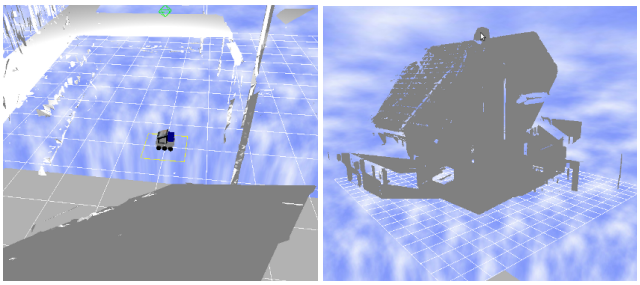


Fig. 10. Rendering of the reconstruction in Gazebo. Without double inserting all triangles the ones that point away from the camera are removed (left). If we double insert the faces with flipped normals the rendering is correct (right).

Fig. 11. Textured reconstruction from a Kinect point cloud. The quality of the reconstructed geometry is acceptable, the quality of the textures is poor due to the relatively sparse point density.

point of view of the robot. The red line indicates the height of the simulated laser scan. The image in the right shows the visualization of the robot's pose and the simulated scan data. From the image it becomes clear that simulated data and loaded geometry match as expected.

DISCUSSION

The presented experiment has shown that LVR reconstructions can be used as environment models in robotic simulation environments. The main problem here is to export the generated polygon models into a data format that can be interpreted by the selected simulator. In case of the Gazebo example we were able to generate a Collada export that was correctly rendered and handled as expected by the simulator. Other simulators like USARSim use proprietary file formats which are not that easy to implement. In principle it may be possible to export the generated meshes to other file formats and re-import them into the proprietary editors, but there is always the risk of loss of information when converting file formats.

In the presented experiment we focused on pure geometry, i.e., we did not evaluate the benefits of the generated textures in the simulator. As shown in Fig. 9, the inclusion of texture information can add more information about the environment then bare geometry, namely the position of the door and signs on the walls. Currently we are not convinced that the quality of textures is high enough to use simulated camera data for vision based robotic applications, so we did not include textures in the simulation. For other purposes like virtual reality or human robot interaction textured models can enhance the human perception of the reconstructed environment significantly.

The presented reconstruction was created using a high resolution laser scanner. Since this kind of sensor is not commonly available on robotic platforms we have also evaluated our reconstruction software on other 3D data like rotating and tilted 2D SICK laser scanners and Kinect data. An example reconstruction for the latter shows Fig. 11. This demonstrates that in principle these sensors can be used as well, but noise and the need of registering a lot of frames make handling uncomfortable. A full evaluation of geometric precision of the generated models from different sensors is beyond the scope of this paper and will be subject of forthcoming publications. For the proof of concept we concentrated on high quality data.

CONCLUSION AND FUTURE WORK

In this paper we have presented an application example where an automatically created polygon model from point cloud data was used as environment model in an robotic simulator. The reconstruction was computed automatically without user interaction. Reconstruction time was less than 4 minutes, which is, especially for complex geometries, quite small compared to the expected effort that is needed to build a model manually.

Future research will concentrate on integration of out-of-core data handling to process even larger data sets. Another interesting field of research is the improvement of texture usage. Currently we generate potentially large textures. By searching for regular patterns it will be possible to reduce the amount of pixel data needed.

REFERENCES

S. Albrecht, J. Hertzberg, K. Lingemann, A. Nüchter, J. Sprickerhof, and S. Stiene. Device level simulation of

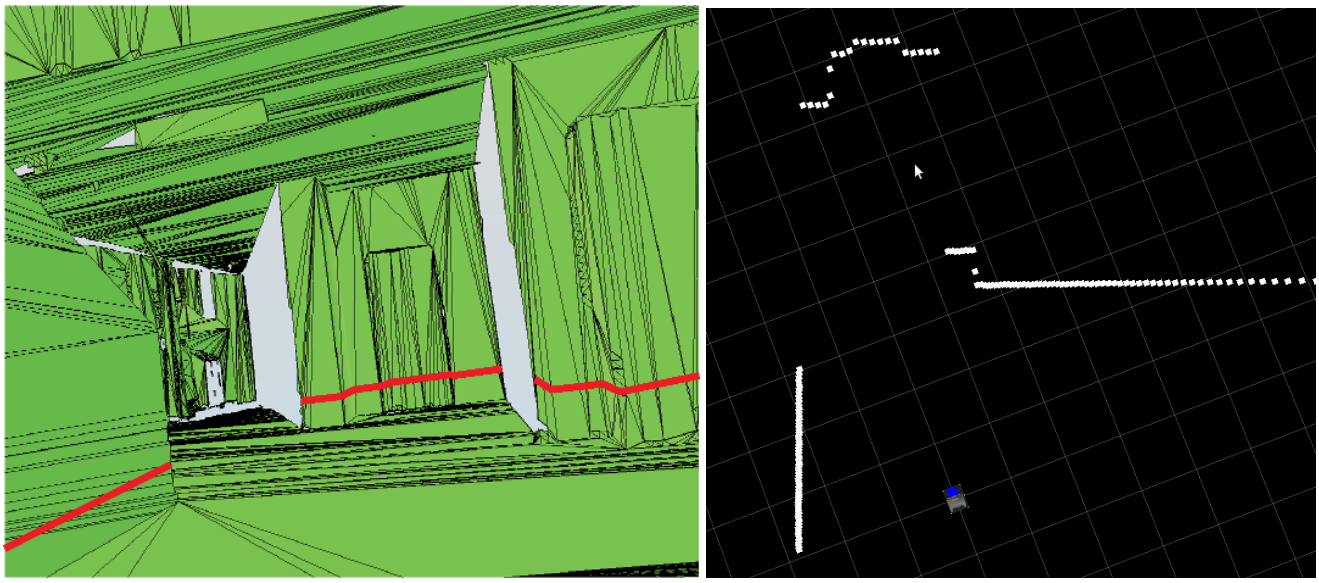


Fig. 12. Simulation of a 2D laser scan in the reconstructed environment. The scanned line in the polygon model is indicated in red (left). The picture on the right shows the robot model at scanning pose and the simulated scan data in RVIZ.

kurt3d rescue robots. In *Proc. SRMED 2006)*, 2006.

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.

H. Annuth and C.-A. Bohn. Smart growing cells. In Joaquim Filipe and Janusz Kacprzyk, editors, *IJCCI (ICFC-ICNC)*. SciTePress, 2010.

O. Devillers. The Delaunay Hierarchy. *International Journal of Foundations of Computer Science*, 13, 2002.

D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, (10):112–122, 1973.

M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH '97*. ACM Press.

G. Guennebaud and M. Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, 2007.

P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGBD mapping: Using depth cameras for dense 3d modeling of indoor environments. In *RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2), 1992.

S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, and et al. Kinectfusion: real-time dynamic 3d surface reconstruction and interaction. In *ACM SIGGRAPH 2011 Talks*. ACM. ISBN 978-1-4503-0974-5.

M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. SGP '06)*, pages 61–70. Eurographics Association, 2006.

L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH '01*, pages 57–66. ACM, 2001.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proc. IROS '04*, pages 2149–2154, 2004.

Las Vegas Surface Reconstruction. `http://www.las-vegas.uni-osnabrueck.de`.

W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH*, 1987.

S. Melax. A Simple, Fast, and Effective Polygon Reduction Algorithm. *Game Developer Magazine*, November 1998.

M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

T Newman and H Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5), 2006.

A. Nüchter, K. Lingemann, J. Elseberg, and D. Borrmann. slam6d, 2012. http://slam6d.sourceforge.net.

A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2), 2009. ISSN 1467-8659.

R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Conference on Robotics and Automation*, 2011.

T. Wiemann, K. Lingemann, A. Nüchter, and J. Hertzberg. A toolkit for automatic generation of polygonal maps – las vegas reconstruction. In *Proc. ROBOTIK*, München, 2012.

AUTHOR'S BIOGRAPHIES

Thomas Wiemann is currently finishing his PhD theses on automatic generation of polygonal maps for robotic applications at the Knowledge Based Systems Research group at Osnabrueck University. An important aspect of his research is generating semantic scene interpretations from point cloud data.

Kai Lingemann is a researcher at the Osnabrueck branch of DFKI's Robotics Innovation Center. His research topics include mobile robotics, where he has published over 60 papers and has been involved in a number of research and industrial projects.

Joachim Hertzberg is a full professor for computer science at Osnabrueck University, heading the Knowledge-Based Systems group; he is also head of the Osnabrueck branch of DFKI's Robotocs Innovation Center. His areas of interest, where he has published over 130 papers, are AI and Mobile Robotics, with a focus on plan-based robot control.