# Digitizing automotive production lines without interrupting assembly operations through an automatic voxel-based removal of moving objects

Johannes Schauer[1] and Andreas Nüchter[1]

***Abstract*—We present an efficient method to partition a point cloud gathered through kinematic laser scanning into static and dynamic points. The presented algorithm utilizes a voxel grid data structure and uses a ray intersection test to mark voxels as dynamic. The algorithm does not require any ego-motion estimations, computationally expensive object recognition or tracking of moving objects over time. It is easy to implement and can be executed on many cores in parallel. We show the viability of this approach by applying our algorithm to a dataset that we gathered by mounting a FARO Focus3D Laser scanner onto a skid which was then sent along a production line for consumer car chassis in a factory of the Volkswagen corporation. Since factory operators are interested in acquiring digital models of their production lines without suspending factory operations, the resulting point cloud will contain many dynamic objects like humans or other car bodies. We show how our algorithm is able to successfully remove these dynamic objects from the resulting point cloud with minimal errors. Our implementation is published under a free license as part of 3DTK.**

## I. INTRODUCTION

Due to increasing demands, ever increasing expectations of today's consumers and growing competition, car manufacturers are faced with having to offer a larger variety of car models than ever and to innovate with new models in ever quicker succession. This increase in variety requires that the production facilities are flexible enough to remain compatible with new requirements and thus competitive. In consequence, car factory operators are forced to repeatedly assess the ability of existing production lines for their viability to manufacture a newly designed car body. The question essentially is: does the new car model fit through all steps of the production process without colliding with the environment and while keeping an appropriate safety margin to it?

These assessments were so far carried out through manual surveys and required the suspension of the running production [1]. Since manual measurements are error prone and halting production incurs monetary losses it would be desirable to carry out all required collision tests and distance measurements on a digital model of the factory.

But most existing factories were erected at a time where digital planning didn't exist yet and then grew organically from that point on. Thus, regular 3D scans are the best option for factory owners to create a digital model and to keep it up-to-date with the structural reality of the factory.

[1]Informatics VII: Robotics and Telematics, Julius-Maximilians-Universität Würzburg, Am Hubland, 97074 Würzburg, Germany, johannes.schauer@uni-wuerzburg.de, andreas.nuechter@uni-wuerzburg.de

While terrestrial scanning techniques have the advantage of higher precision over kinematic laser scans, terrestrial laser scans have the disadvantage, that operators have to find fitting scan locations. Oftentimes these do simply not exist because factories are built with maximum volume utilization in mind, so there is little free space. Terrestrial scans become completely impossible in situations where the car bodies are sent through narrow tunnels or elevator shafts.

Thus, in 2014, we presented a system that mounted a FARO Focus3D laser scanner on a skid that would otherwise transport a car body along the production line [2]. That system is sent along the production line, while continuously scanning its surroundings. This system is not only able to record all features along the production line that are significant for collision detection queries, but also automatically produces the trajectory new car bodies take. In 2016 we have shown the feasibility to carry out collision detection queries on the resulting datasets [3] on CPU and GPU platforms using state-of-the-art algorithms for each platform with comparable performance results.

The last fundamental building block for full automation of the pipeline is presented in this paper. So far, recorded scans included numerous points along the production line, that are not part of the static factory structure but indeed belong to dynamic objects like humans working close to the production line. Without removal of all dynamic objects, an operator would be faced with many false positive results during any collision detection query. Thus, now we present a method to clear the collected scans from any dynamic points.

This paper is organized as follows. After discussing related work in the following section, we describe the conceptual design of our algorithm in section III and implementation details in section IV. In section V we present results of our algorithm on scans that we took via mobile mapping along a production line of a Volkswagen factory for consumer car assembly. Section VI discusses the performance of our implementation of the algorithm. We conclude this paper with a section on future work in section **??** before we draw our conclusion in the last section.

## II. RELATED WORK

The authors of [4] and [5] also use a voxel data structure to distinguish between static and dynamic sections of the recorded point cloud but instead of recording free voxels, they count how often a voxel is occupied. Due to the restrictions of this approach as explained in the following section, they have

to make a number of assumptions about their environment and employ several heuristics that are not necessary with our algorithm. Furthermore, their approach requires a ground surface estimation — in contrast to our approach which works with any arbitrary environment.

The creators of OctoMap [6] also use the same algorithm as we do by Amanatides and Woo [7] to cast rays. But instead of voxels they use an oct-tree data structure to find free volumes. They also employ a similar approach to avoid marking volumes as free in situations where rays meet a surface at a very shallow angle by grouping multiple scan slices together.

Besides voxels and oct-trees other data structures to store occupation information in are elevation maps [8], [9], multi-level surface maps [10], Gaussian Mixture Model [11], [12].

Existing methods that require computation of free volume for robotic path planning are known to use a 3D Bresenham ray casting kernel [13] carrying out the ray casting in many parallel threads on the GPU. GPU-based ray casting techniques were first shown by [14] and [15] and are today often implemented using OpenGL and CUDA [16].

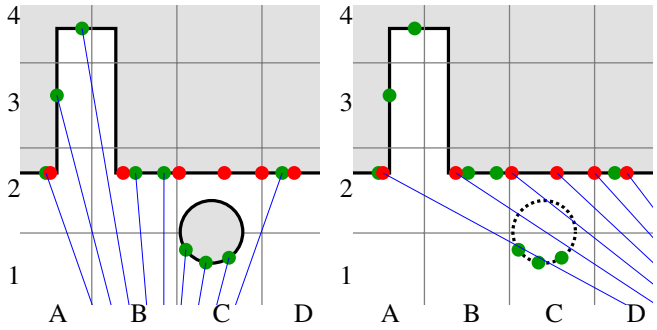
A related topic is 3D change detection. The authors of [17] project the points from one scan into the spherical coordinate system of another and then use angular neighbors to find those points in the first scan that the latter was able to see through and which can thus be classified as dynamic. While their approach is very efficient for a single pair of scans, runtime increases quadratically with increasing number of input scans. The authors of [18] also reason about the volume occupied by laser rays and fuse multiple rays into a larger volume using the Dempster-Shafer theory.

## III. DESIGN

Our initial approaches were inspired by how humans tend to distinguish between static and dynamic objects: If an object is seen as immobile for long enough, then we will classify it as static. While this approach would probably work well for a scanner with a static position (relative to the environment that we consider static), it seems to be an unfeasible approach in the mobile mapping scenario. Due to the scanner moving and the resulting variable occlusion of objects, it is hard to calculate how long an object *should* be visible (and not vanish because of an occlusion). Furthermore, without prior knowledge about whether the occluding object is actually static (and its occluded volume can thus be trusted) a chicken-and-egg problem is created where we need to know about which objects are static before we can consider them for occlusion testing.

Thus, instead of counting how long an object is seen as static, our algorithm does the opposite and instead tests whether any seen object was at any point in time in fact see-through. Since we want to avoid any higher-level processing like object recognition, our "objects" here are the voxels of a regular voxel grid. This data structure has the advantage, that it is computationally easy to enumerate all voxels along a ray (a line of sight).

Another advantage of this approach is, that we do not need to keep count of how often a given voxel has been seen with




(a) The scene as scanned from a center position. The scanner measures the green points.

(b) The scene as scanned from a position to the right. The circular object in areas C1 and C2 moved away and its former position is marked with dotted lines. The scanner measures the red points.

Fig. 1: The algorithm applied to a 2D scenario. The gray raster marks the 2D voxel (pixel) boundaries. Blue lines mark the scanner lines of sight. Dark lines are object boundaries. Gray areas mark solid space while white areas mark free space. Round dots represent the measured scan points of the first scan (top) in green and of the second scan (bottom) in red.

points in it (depending on the scanner position and possible occlusion) but that instead, seeing a voxel as free only during a single scanner rotation is indication enough that the points that were otherwise seen in it the rest of the time must in fact be dynamic. An additional advantage of this approach is, that it will automatically *not* remove points that were only seen very seldom.

Consider figure 1. It displays our algorithm as applied in a two-dimensional scenario. The circular object in areas C1 and C2 is dynamic and only seen by the first scan (figure 1a). Since the second scan (figure 1b) measures the red points in A2 and B2 with a line of sight that crosses area C1, it must mean that the points that were measured in C1 in the first scan can be classified as dynamic.

Figure 1 also shows how the algorithm does not remove points from areas that were only visible in a single scan. For example the green points in areas A3 and A4 are only seen from the central scanner position in figure 1a. Still, they are not removed because these areas are never marked as see-through by other scans (for example the second scan in figure 1b). The same holds for the red points in area C2. They are only seen by the second scan in figure 1b because the circular moving object in C1 and C2 occludes the points during the first scan in figure 1a. Still, the points remain classified as static because their containing areas are never marked as see-through.

## IV. IMPLEMENTATION

The implementation of the algorithm outlined in the last section is based on carrying out repeated voxel-walks along all voxels from each scanner position to the target points. The voxels that are traversed by this search up to the target

point are then marked as free up to conditions laid out in the rest of this section.

The voxel grid is implemented using a hash-map. Thus, reading and writing from and to the voxel grid is possible in $O(1)$. Furthermore, memory is only spent on occupied voxels.

Since the input data is acquired through mobile mapping, the input to the algorithm are the individual registered scan slices. Each scan slice comprises all measurements of a single mirror rotation. A single rotation of the scanner itself thus comprises a multitude of individual scan slices. Figure 1 can also be seen as a top-view on a three-dimensional scene that was recorded in this fashion (with the scanner being static in each figure). Then, each of the points in each figure would have been recorded in a different scan slice. Scan slices are indexed with unique integers.

The algorithm starts by reading all registered scan slices as well as the scanner positions that each slice was taken from into a global coordinate system. For each point, the scan slice index where the point was measured from is preserved.

In a next step, a voxel grid is created. We iterate through all points in the global coordinate system, find the voxel that they would fall into, and then store the scan slice index that the point belongs to in the voxel as a set data structure. Thus, the voxel grid does not store the point coordinates. Instead it only stores the set of scan slice indices of which points were seen in each voxel. This means, that we usually store less data points in each voxel than the number of points that would fall into that voxel because depending on the voxel size and density of the scan, it is likely, that a voxel contains many points of the same slice index.

Finally, the algorithm iterates over all scan slices and the associated scanner position that each slice was taken from. Starting from the corresponding scanner position of each slice, a ray is shot to each point recorded from that position. Each such ray is then traversed through the voxel grid at most up to the target point. All voxels up to the final voxel that are intersected by the ray are marked as free.

To do the traversal we use an algorithm that is inspired by the work shown in [7]. This algorithm in essence is a 3D variant of the Bresenham algorithm and is used by several software projects like the Point Cloud Library (PCL) [19] and OctoMap [20]. We enhanced the original voxel traversal algorithm presented in [7] to support negative voxel coordinates, casting rays exactly parallel to the voxel grid boundaries, casting rays exactly alongside voxel boundaries and casting a ray starting exactly from a voxel boundary. We also modified the algorithm such that instead of repeated addition of floating point numbers, we multiply a counter by the step size. This is to prevent escalation of small floating point errors after multiple additions ($0.1+0.1+0.1$ is unequal 3 times $0.1$) and is especially necessary in cases where the ray is nearly parallel to the voxel grid boundaries. The cost of this added exactness is just one multiplication instruction more per loop iteration.

One problem remains. To illustrate it, one can take for example the scan slice that resulted in measurement of the green point in area A4 of figure 1a. Clearly, the line of sight from the scanner towards this point (in blue) passes through area B2. Still, that area should not be marked as free. We solve the problem that partly occupied voxels impose by looking at the neighbor slices of the current scan slice. If the line of sight passes through voxels that also contain points from scan slices adjacent to the scan slice of the current target point, then these voxels are not marked as free and the voxel-walk aborts early without marking the any further voxels as free.

The idea here is, that while we are tracing lines of sight towards individual points, we also always take a sliding window of their neighborhood into account. This technique is similar to the one presented in [6] to avoid removal of points when scanning surfaces with a small incident angle of the laser beam. The size of the window is best chosen to be large enough such that the central slice does not share any voxels that we are interested in with the most outer slices. Obviously, the starting voxel of most rays will be shared by many slices, but having marked the starting voxel as free poses no problem as we can probably assume that the space that the scanner was moved through was free to begin with. At the same time, the window size should not be too large. For example it should not happen that the window contains slices that record points of the same object from two completely different scanner rotations. To satisfy both constraints, one input to our algorithm is the number of slices that the scanner records in one full rotation. The window size is then chosen as half that size. This ensures that the "field of view" is large enough to consider adjacent partly occupied voxels and that it is not so large as to have the same object twice in a single "field of view". An additional assumption here is also that the scanner is never turned quickly enough against its own orientation such that this constraint would be violated. In current practical setups in factory environments, this is not likely to happen.

At this point of the algorithm a ray has been shot toward each point in the scan and the traversed voxels have been marked as free. A last loop iterates through all points in the scan and marks each point either as dynamic or as static, depending on whether the point falls into a voxel that was marked as free or a voxel that was not, respectively.

The main part of the algorithm (shooting rays at every point and walking along all voxels traversed by the ray) can trivially be parallelized. This is, because this part of the algorithm only requires read-only operations (of point and voxel coordinates). The only write operation (marking voxels as see-through) can be done concurrently without individual threads interfering with each other as this part of the algorithm never queries whether a voxel was earlier marked as see-through or not.

The implementation of the algorithm was done in C++ with multithreading through OpenMP and using the library `std::unordered_map` to represent the voxel grid. It is published under the terms of the GNU General Public license 3 as part of *3DTK – The 3D Toolkit* [21].

## V. RESULTS

To test our method, we used a data set that we collected with a FARO Focus3D laser scanner in factories of the Volkswagen AG and with initial pose estimations coming from a setup supplied by the Measurement in Motion GmbH. The acquired data (comprised of individual scan slices) was registered using a semi-rigid SLAM approach [22] and then processed with our software to partition the point cloud into static and dynamic points. The dataset comprises 398 thousand scan slices with overall 350 million points. Since an operator is only interested in the immediate volume around the trajectory that a prospective new car model will take, we limited the search for free voxels to a volume with a 2.5 m radius (so a 5 m diameter) around the trajectory. This should be sufficient for most consumer car models which rarely exceed a width or height of 2 m. We performed our tests with a voxel size of 5 cm as this size is smaller than the typically imposed security margins.

Unfortunately, data collected through mobile mapping is by its nature more noisy than data collected through terrestrial scans. This means, that surfaces easily appear noisy in the global coordinate frame. This in turn leads to an affect where the presented approach would mark parts of that surface as "see-through" simply because there existed points "behind" the wall from the point of view of the laser scanner that were misplaced there during the registration phase. To dampen the effects of more noisy scans, we take three different measures.

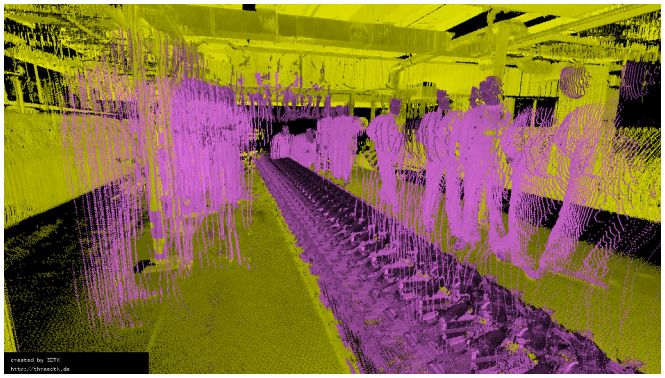
Firstly, a pre-filter is applied to the dataset. This filter examines each point of the dataset and removes it, if its direct neighborhood does not contain more than a certain number of points. For our dataset we applied a filter that required at least 3 points to lie within a radius of 3 cm. The application of this filter resulted in 11% of the former points to be removed.

Secondly, instead of walking each ray up to the target point we walk the ray up to a certain distance to the target point. This makes sure that the voxels that are marked as free are not accidentally part of the object that the target point belongs to. To account for small angles of a cast ray relative to a surface, we carried out the search up to 30 cm to each target point.

Thirdly and lastly, since scanner precision decreases with distance, we also decide not to follow the line of sight toward points at all if they are more than a certain distance away from the scanner. Since the factory environment is closely packed with objects, a distance of 10 m was enough for our setup.

Figure 2 shows a visualization of the result of our algorithm. We omit an image of the original point clouds for brevity. The topmost figure 2a shows the original point cloud partitioned into static (yellow) and dynamic (pink) points. As an additional example, figure 3 shows the resuls of the algorithm at another point of the production line. In the following we will use figure 2. One can make four observations.

Firstly, nearly all humans have been removed from the scene as they were detected as moving objects. This holds for humans walking alongside the scanner as well as for humans standing in the way of the scanner on the production line. One can though see that some artifacts remain. On the left hand side of figure 2b one can see some remaining points that originally belonged to humans. These were not removed because this group of humans remained standing there for the whole duration of the scan and was thus not detected as dynamic. Furthermore, there are some remains of scanned shoe soles left on the floor. The latter artifacts decrease with decreasing voxel size.



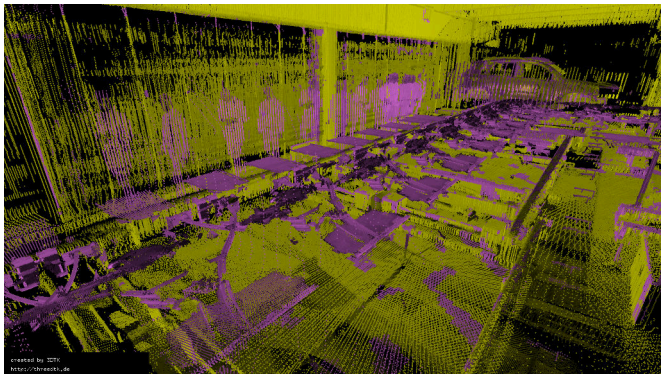
(a) The original scan partitioned into static (yellow) and dynamic (pink) points.

(b) The original scan without dynamic points.

Fig. 2: A scene from a Volkswagen consumer car factory with humans standing alongside or on the production line.

Secondly, the skid that the scanner was mounted on was scanned by the scanner during every single scan rotation and is thus part of the original scan in figure 2a and fills the whole production line. The points representing the skid were successfully classified as dynamic though without any manual intervention and were thus removed in figure 2b without any remains.

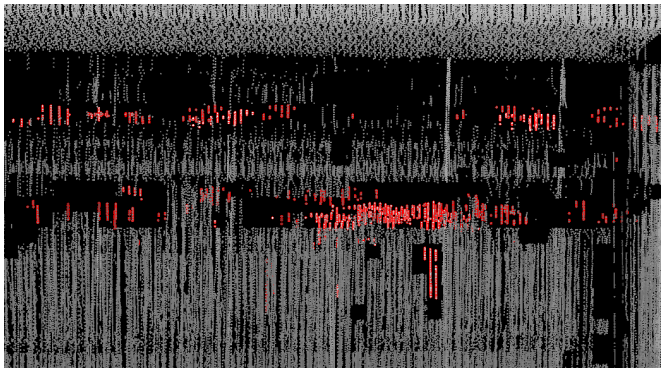
Thirdly, it is possible to see one fundamental drawback of our approach. Objects that are "see-through" for the scanner like glass, transparent plastic or thin lattices or meshes will be declared as dynamic because the scanner is able to see objects behind them. One can see this effect on the floor of the production line (which was a fine mesh that allowed to see the floor below) as well as in the background where transparent plastic foil hanging from the ceiling was completely removed in figure 2b.



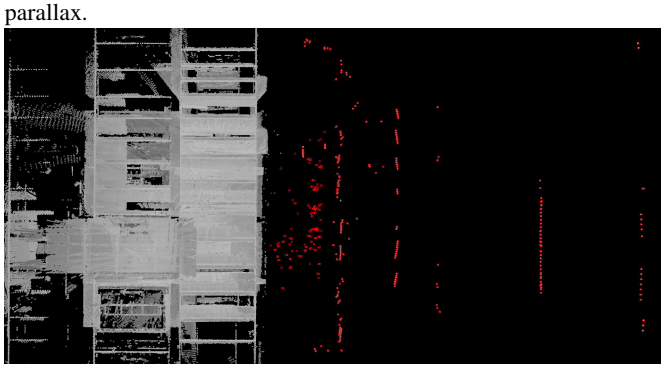
(a) The original scan partitioned into static (yellow) and dynamic (pink) points.

(b) The original scan without dynamic points.

Fig. 3: A scene from a Volkswagen consumer car factory with humans walking or standing behind the production line in front of a wall.



(a) "Holes" in the wall created by points (in red) recorded behind the wall. The points are not exactly aligned with the holes due to parallax.



(b) Top-view of the scene, showing the same points behind the wall (in red).

Fig. 4: The high reflectivity of surfaces commonly found in factory environments poses a great challenge. The wall in the top figure has holes because of reflected points "behind" the wall (in red). These points are false as the wall is solid and the area on the right in the bottom figure should be empty.

Fourthly, reflections can significantly alter the results. If objects with a high reflectivity exist in the scene, then the scanner will see objects as if they were "behind" the mirror. This not only leads to the mirror being marked as dynamic, but also static objects behind the mirror to become "see-through" and thus marked as dynamic. See figure 4 for a visualization of the problem. But mirrored points are a fundamental problem any 3D scan and that problem has to be solved at a different level.

Unfortunately, we didn't have the opportunity to take terrestrial scans of the scene because we didn't foresee ourselves using this data for the purposes of non-static point removal at the time that the scan was taken. Otherwise, a comparison of the result of our algorithm as shown in figures 2b and 3b with an actual human-free point cloud might have made it easy to quantify the quality of our approach. On the other hand, the quality of our approach is heavily dependent of how well the original scan registered. So it is likely that any such comparison to a terrestrial scan will instead become a comparison between the mobile mapping approach and the terrestrial approach.

## VI. PERFORMANCE

We tested our algorithm on an Intel Xeon e5-2630 v3 Desktop system with 8 physical cores with 2.4 GHz each. Due to HyperThreading we used 16 threads in our tests to make use of all virtual cores. As we expected the performance to depend on the input size, we recorded the run time of our algorithm for 100 different input sizes from just 1000 scan slices up to 100000 scan slices. Each scan slice contains 877 points on average, so we test our algorithm on input point clouds ranging from 877 thousand up to 87.7 million points in size. Since the performance of the algorithm heavily depends on the geometry of the point cloud, we took 130 samples from random locations in our test data for each of the 100 different input sizes and then used the median run time.

The results of our measurements can be seen in figure 5. The figure displays the performance of the algorithm as a measure of scan slices the algorithm is able to process per second. One can observe that the algorithm processes each slice faster if the overall input size is small. As the overall size of the input point cloud grows, the time our algorithm spends on each slice becomes more constant. Thus overall we can say that our algorithm scales linearly with the input size.

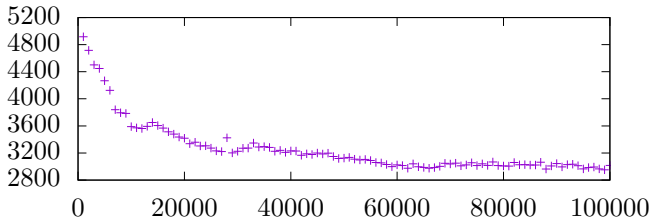


Fig. 5: The x-axis shows the number of scan slices given to the algorithm. The y-axis shows the performance of the algorithm as a measure of the number of slices that the algorithm is able to process per second. One can see that the algorithm is faster with less input but shows an asymptotic behaviour with larger input.

## VII. CONCLUSIONS

We presented an algorithm that is able to partition a point cloud into static and dynamic points in a simple and straight forward manner. Using the example of kinematic laser scans taken from an automotive production line, we have shown how our algorithm successfully removes dynamic objects like humans from the resulting point cloud with a minimal number of false positives. The algorithm doesn't make assumptions about the structure of the underlying point cloud data, is easy to implement and can trivially be executed in parallel.

Needless to say, a lot of work remains to be done. In the future we want to expand our method to terrestrial laser scans, further reduce the amount of false positives and explore more efficient data structures than a voxel grid as for example provided by OctoMap.

## ACKNOWLEDGMENT

We thank the Volkswagen corporation for their cooperation with us that made this research possible and especially our contacts Dirk Koriath and Ulrich Rautenberg for their continuous support. Furthermore, we thank the Measurement in Motion GmbH for providing us the datasets that the results in this paper are based on.

## REFERENCES

[1] E. Shellshear, R. Berlin, and J. S. Carlson, "Maximizing smart factory systems by incrementally updating point clouds," *IEEE computer graphics and applications*, vol. 35, no. 2, pp. 62–69, 2015.

[2] J. Elseberg, D. Borrmann, J. Schauer, A. Nüchter, D. Koriath, and U. Rautenberg, "A sensor skid for precise 3d modeling of production lines," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-5, pp. 117–122, 2014. [Online]. Available: http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-5/117/2014/

[3] J. Schauer, B. J., M. K., and A. Nüchter, "Performance comparison between state-of-the-art point-cloud based collision detection approaches on the cpu and gpu," in *Symposium on Telematics Application 2016*, 2016.

[4] A. Asvadi, P. Peixoto, and U. Nunes, "Two-stage static/dynamic environment modeling using voxel representation," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 465–476.

[5] A. Asvadi, C. Premebida, P. Peixoto, and U. Nunes, "3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes," *Robotics and Autonomous Systems*, vol. 83, pp. 299–311, 2016.

[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[7] J. Amanatides, A. Woo *et al.*, "A fast voxel traversal algorithm for ray tracing," in *Eurographics*, vol. 87, no. 3, 1987, pp. 3–10.

[8] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade, "Terrain mapping for a roving planetary explorer," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE, 1989, pp. 997–1002.

[9] P. Pfaff, R. Triebel, and W. Burgard, "An efficient extension to elevation maps for outdoor terrain mapping and loop closing," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 217–230, 2007.

[10] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2006, pp. 2276–2282.

[11] H. Andreasson, M. Magnusson, and A. Lilienthal, "Has somethong changed here? autonomous difference detection for security patrol robots," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 3429–3435.

[12] P. Núñez, P. Drews, A. Bandera, R. Rocha, M. Campos, and J. Dias, "Change detection in 3d environments based on gaussian mixture model and robust structural matching for autonomous robotic applications," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2633–2638.

[13] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified gpu voxel collision detection for mobile manipulation planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 4154–4160.

[14] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser, "Smart hardware-accelerated volume rendering," in *VisSym*, vol. 3. Citeseer, 2003, pp. 231–238.

[15] J. Kruger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. IEEE Computer Society, 2003, p. 38.

[16] A. Weinlich, B. Keck, H. Scherl, M. Kowarschik, and J. Hornegger, "Comparison of high-speed ray casting on gpu using cuda and opengl," in *Proceedings of the First International Workshop on New Frontiers in High-performance and Hardware-aware Computing*, vol. 1, 2008, pp. 25–30.

[17] J. P. Underwood, D. Gillsjö, T. Bailey, and V. Vlaskine, "Explicit 3d change detection using ray-tracing in spherical coordinates," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4735–4741.

[18] W. Xiao, B. Vallet, M. Brédif, and N. Paparoditis, "Street environment change detection from mobile laser scanning point clouds," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 107, pp. 38–49, 2015.

[19] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[20] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010.

[21] A. Nüchter and K. Lingemann, "6D SLAM software," http://slam6d.sourceforge.net/, 2011.

[22] J. Elseberg, D. Borrmann, and A. Nüchter, "6dof semi-rigid slam for mobile scanning," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1865–1870.