A VOXEL-BASED METADATA STRUCTURE FOR CHANGE DETECTION IN POINT CLOUDS OF LARGE-SCALE URBAN AREAS

Joachim Gehrung[ab*], Marcus Hebel[a], Michael Arens[a], Uwe Stilla[b]

[a] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB,
76275 Ettlingen, Germany - (joachim.gehrung, marcus.hebel, michael.arens)@iosb.fraunhofer.de
[b] Photogrammetry and Remote Sensing, Technische Universitaet Muenchen, 80333 Muenchen, Germany - stilla@tum.de

Commission II, WG II/10

KEY WORDS: Ray Casting, Discretization Artifacts, Volumetric Environment Representation, Change Detection

ABSTRACT:

Mobile laser scanning has not only the potential to create detailed representations of urban environments, but also to determine changes up to a very detailed level. An environment representation for change detection in large scale urban environments based on point clouds has drawbacks in terms of memory scalability. Volumes, however, are a promising building block for memory efficient change detection methods. The challenge of working with 3D occupancy grids is that the usual raycasting-based methods applied for their generation lead to artifacts caused by the traversal of unfavorable discretized space. These artifacts have the potential to distort the state of voxels in close proximity to planar structures. In this work we propose a raycasting approach that utilizes knowledge about planar surfaces to completely prevent this kind of artifacts. To demonstrate the capabilities of our approach, a method for the iterative volumetric approximation of point clouds that allows to speed up the raycasting by 36 percent is proposed.

1. INTRODUCTION

The generation of reliable 3D environment data is playing an increasingly important role in todays mobile mapping systems. Therefore more and more of these apply mobile laser scanning, since this technology is usually able to produce more dense and accurate measurements than camera systems and airborne laser scanning. Change detection based on such data offers the possibility to identify developments of urban landscapes such as the construction and tear-down of buildings as well as more subtle ones like missing or moved city furniture. The task of change detection in large scale urban environments raises two research questions. The first one is considered with data cleansing, since measurements of moving objects and short-term stable structures such as parked cars would lead to changes which are irrelevant in the context of city development. The second research question deals with the efficient representation and handling of large environments. An extrapolation based on the *TUM City Campus*[1] dataset shows that an epoch for a city like Munich would require more than 100 TiB of data. Since multiple epochs are required for change detection, a massive amount of data needs to be stored and processed efficiently.

A suitable form of representation that fits both tasks - change detection and the handling of large urban environments - are volumes like voxels. Due to the space discretization, these are very memory efficient for reasonable resolutions. Compared to point clouds, they also have an upper memory boundary. Furthermore, when utilizing an octree based data structure for volume organization, it is not only possible to store information about free and occupied space, but also to represent unseen areas. This is valuable information for change detection, since the appearance or disappearance of a structure can only be evaluated correctly in its context. The downside of using such a representation is the sensitivity to artifacts, especially the ones caused by rays traversing a voxel otherwise considered as solid. These are particularly significant in the vicinity of planar structures, whenever an adverse combination of the voxel resolution and the angle between plane and ray appears. Correcting this effect is necessary, since a volume classification based on voxel statistics is strongly influenced by this effect.

As a first step towards a volumetric representation suitable for change detection in large scale urban environments, this paper proposes a raycasting algorithm that utilizes a plane-based filter to suppress the artifacts mentioned above. To demonstrate the capabilities of this method, we also propose an iterative algorithm for the volumetric approximation of a point cloud. In addition to that, the suitability of voxel statistics for moving object detection is investigated.

2. RELATED WORK

Change detection is a topic of interest in various research fields and therefore has different names depending on the context. Many research regarding this topic is done in 2D and 3D computer vision, where the *Detection and Tracking of moving Objects (DATMO)* is applied to solve problems from robotics, autonomous driving and SLAM. Litomisky and Bhanu (2013) for example remove moving objects from point clouds utilizing a clustering approach in order to make SLAM more robust. Since this paper deals with data collected by mobile laser scanning, the focus of this chapter will be on point cloud based approaches to change detection.

2.1 Change Detection on Point Clouds

Point based approaches are often used, since change detection on object level is affected by the quality of the object detection and classification. Girardeau-Montaut et al. (2005) utilize an

*Corresponding author
[1] http://s.fhg.de/mls1

octree-based structure to organize a point cloud and present several simple cloud-to-cloud comparison algorithms. Zeibak and Filin (2008) compare depths images from a stationary terrestrial laser scanner in order to compare multiple epochs recoded from the same position. Both approaches incorporate no information about free or unseen space.

2.2 Ray-based Change Detection

Point clouds can also be interpreted as a set of rays originating from the sensor position. Underwood et al. (2013) proposed an approach to 3D change detection by comparing LiDAR scans with ray casting in spherical coordinates. Hebel et al. (2013) combine voxel-indexed rays by applying the Dempster–Shafer theory of evidence to determine changes in airborne laser scans. Xiao et al. (2015) use a similar approach based on mobile LiDAR data. Both approaches consider free and unknown space. However, the downside of this kind of approach is that every single ray within the region of interest needs to be evaluated, which is not feasible for mobile laser data consisting of millions of single scans per second.

2.3 Change Detection based on Segments or Objects

This class of change detection methods is characterized by the segmentation of point clouds in order to get clusters or instances of known object classes. Schachtschneider et al. (2017) extract clusters from point clouds of urban environments using a region growing algorithm and assess the temporal behavior of each cluster utilizing an occupancy grid. Aijazi et al. (2013) also apply a segmentation step to the point cloud, but then classify the resulting clusters into permanent and known temporary classes. A similarity map created from an evidence grid is then either used for multiple epoch data fusion to build a 3D urban map or to do change detection. Both approaches require the performance of segmentation and classification to be reliable in order to generate good results. Applying classification also requires specialization on known object classes.

2.4 Change Detection using Occupancy Grids

Occupancy grids approximate the environment by storing information in a discretized space. Pagac et al. (1996) utilize a 2D occupancy grid based on the Dempster–Shafer theory of evidence to represent an autonomous vehicle’s environment. A similar approach has been proposed by Wolf and Sukhatme (2004). The authors use two grids to model static and dynamic parts of a scene and are therefore able to detect dynamic objects. Other approaches to change detection utilize an octree-based 3D occupancy grid referred to as Octomap (Hornung et al., 2013). It is used by Azim and Aycard (2012) in an online approach for moving object detection. A voxel list updated by conflict search between the known environment and LiDAR measurements is clustered and filtered for object candidates. The theoretical framework behind Octomap has also been used in our previous works, where we showed that the classification of voxels based on voxel statistics leads to the artifacts mentioned above and therefore to an inaccurate approximation of the environment (Gehrung et al., 2016, 2017).

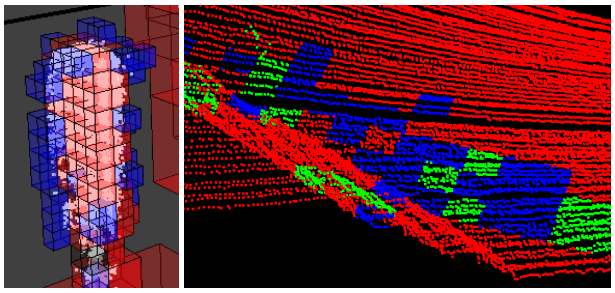
3. ARTIFACTS OF VOLUMETRIC REPRESENTATIONS

3.1 Volumetric Representation Framework

In this work, a modified version of the volumetric framework proposed by Gehrung et al. (2016) is used. Spatial data is organized in a Cartesian reference frame such as ECEF (”earth-centered, earth-fixed”). A grid is used to subdivide the data into cubic chunks called *tiles*. Among other things, each tile comprises multiple data containers storing the rays and the volumetric representation. The latter consists of an octree for volume organization. Each octree node contains statistics of the rays traversing and penetrating it, the ray indices, as well as a probability table describing the affiliation to one of multiple classes. For memory management and visualization reasons the edge length of a tile is set to 32 m.

3.2 Discretization based Artifacts

As mentioned above, one of the primary problems in terms of volumetric representations are artifacts that are caused by rays that traverse voxels which are considered to be solid. Since a voxel is a more or less arbitrary discretization of space, it is likely that measurements implying multiple classes of space are combined within a single voxel. For example this happens at object borders such as building corners, doors or windows. While a border voxel contains parts of the object, it is also traversed by the rays which belong to the free space around it. This particular artifact is characterized by small clusters of improperly classified voxels at object borders (cf. Figure 1(a)).



(a) (b)

Figure 1. Traversal artifacts due to ray casting within discretized space. (a) Artifact caused at object borders. (b) Improper voxel classification near plane-like structures.

A more severe class of artifact is caused by rays traversing a plane or planar surface at a relatively flat angle. In this case voxels otherwise considered to be occupied are traversed by a ray implying they are free. This leads to conflicting observations and therefore to large patches of incorrectly classified voxels. We showed in (Gehrung et al., 2017) that this effect appears when determining the voxel class on volume statistics only, without including information of higher levels such as the local neighborhood (cf. Figure 1(b)).

4. PLANE-BASED ARTIFACT REMOVAL

4.1 Plane Extraction

The algorithm proposed in this paper is intended to solve artifacts caused by plane-like structures. The idea behind it is simple. For each plane-like surface patch found in the point cloud, a second virtual plane shifted in direction of the ray origin is used to truncate each ray. To make sure that no voxel belonging to the surface patch is traversed, the distance between both planes is the diagonal of a voxel. Outside the space between both planes, all voxels are traversed as usual. Since a plane is infinite by definition, it is better to represent the surface patch with an oriented bounding

box (OBB). When adapting its thickness (which corresponds to its dimension with the lowest eigenvalue) to the voxel size of interest, it acts like the second plane mentioned above, but with the same spatial boundaries as the original surface patch.

Surface patches containing planes are extracted by using the normal based region growing algorithm proposed by Rabbani et al. (2006). Neighboring points which are close enough in terms of a smoothness constraint are merged to clusters. The constraint is the angle between the point normals. Seed points are selected from regions with minimum curvature, since these ones are located within the most flat areas. This reduces the total number of segments. Since lots of very small surface patches are found by the algorithm, candidate filtering is applied. Therefore the extracted clusters are sorted by their number of points, the lowest quarter on the list is dropped. This appears to be a good practice.

Oriented bounding boxes are preferred to axis aligned bounding boxes (AABB), since the latter ones are only a crude representation of the plane. To generate an oriented bounding box, principal component analysis is used to determine the coordinate frame of the point cloud cluster. Based on this, the eigenvectors and the cluster’s centroid are used to compute the transformation into the cluster’s coordinate frame. Once all points have been transformed, the AABB is determined. Due to the applied clustering procedure, the surface patch is not entirely flat. Therefore the extend of the AABB in the direction of the eigenvector with the smallest eigenvalue is set to zero. This is required to prevent volume approximation errors during raycasting. The AABB and the transformation from the global coordinate frame to the cluster coordinate frame are stored and used for ray intersection during the *plane-filtered raycasting*.

4.2 Plane-filtered Raycasting

Raycasting is used to generate volumes from point clouds. Each point is considered to be a ray originating at the sensor position and ending at the location of the measured surface point. Whenever a ray ends within a voxel or traverses it, the voxel’s corresponding counter is incremented. Casting a ray consists of two steps, traversing the free space and inserting the endpoint (which may correspond to occupied space but also to free space, if the ray has been truncated at the tile border due to performance reasons).

To avoid artifacts, each ray needs to be checked against all oriented bounding boxes for intersections. The test requires the ray to be projected into the box’s coordinate frame by applying the transformation calculated in the *plane extraction* step. At this point, intersection can be tested against an AABB, for example by using an algorithm such as the one proposed by Williams et al. (2003). Before the test is carried out, the bounding box is expanded in all dimensions by the diagonal of a voxel. This needs to be done at this point and not in the *plane extraction* step, since the bounding box needs to be adapted to the current voxel size (which may vary during refinement). If the test is positive, it returns the offset t along the ray at which the intersection occurred. This information can be used to determine the new endpoint of the untransformed ray.

An overview of the plane-filtered raycasting algorithm is given in Algorithm 1. In a preliminary *filter phase* the current ray is truncated based on intersection tests with all bounding boxes. Since there are multiple kinds of intersections, the case differentiation illustrated in Figure 2 needs to be applied. The following enumeration explains the nature of the different cases and how they need to be handled.

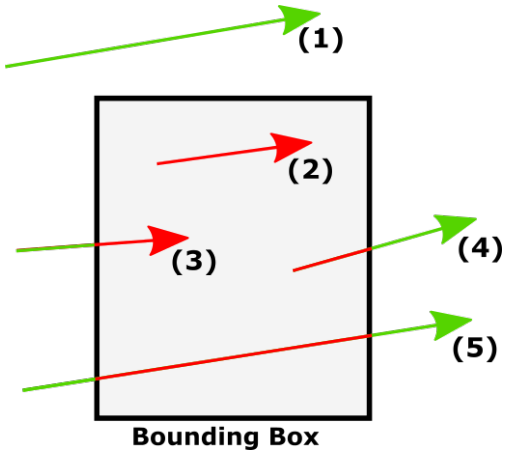


Figure 2. Intersecting ray and bounding box leads to five cases that need to be handled. Valid parts of the ray are in green, invalid ones in red. (1) No intersection with ray. (2) Ray is inside. (3) Ray starts outside but ends inside. (4) Ray starts inside, but ends outside. (5) Ray starts and ends outside, thereby traversing the bounding box.

1. No intersection between the ray and any bounding box. Add the full ray and its endpoint.

2. The ray is completely inside the bounding box. No traversing of the ray needs to be done. The endpoint is only to be considered in case it is occupied.

3. The ray starts outside the bounding box and ends inside. Traverse the part of the ray outside the bounding box. Only an occupied endpoint is to be inserted.

4. The ray starts inside the bounding box and ends outside. Traverse only the latter part of the ray. The endpoint is to be inserted, no matter whether or not it is free or occupied.

5. The ray starts and ends outside a bounding box it traverses. The two parts of the ray are handled separately. The first part outside the bounding box is treated as free space without endpoint. The second part is handled separately by adding it as an independent ray to the ray list.

An overview of the filter phase, which implements the case differentiation for each ray, is given in Algorithm 2. Both offsets used to truncate the ray after intersection with all bounding boxes are set to $t_{\text{start}} = 0.0$ and $t_{\text{end}} = 1.0$, which corresponds to the full ray. For each oriented bounding box, the intersection offsets t_{near} and t_{far} along the ray are calculated. If an intersection occurred, the ray offsets t_{start} and t_{end} are updated according to the case differentiation. The if-statements within most of the cases ensure that previous intersections with other bounding boxes are not revoked.

Prior to the actual raycasting step, ray consistency needs to be verified by evaluating whether or not $t_{\text{end}} \geq t_{\text{start}}$. A valid ray is truncated accordingly to t_{start} and t_{end}, raycasting along the remaining part is used to select the voxels that are to be marked as free space. During the plane-filtering step it was decided whether or not to insert an endpoint (endpoint handling policy). In case that an endpoint is to be inserted, the according voxel at the ray’s endpoint is determined and the counter for either free or occupied space is incremented.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-2-97-2018 | © Authors 2018. CC BY 4.0 License.

Algorithm 1: Overview of the plane-filtered raycasting algorithm.

Data: List of rays $R = \{r_1, \ldots, r_n\}$,
volumetric representation V.
Result: Updated volumetric representation V.
begin
  $B \longleftarrow getBoundingBoxes(R)$
  for $r_i \in R$ **do**
    $t_{\text{start}}, t_{\text{end}}, e, R \longleftarrow filterPhase(B, R, r_i)$
    if $t_{end} >= t_{start}$ **then**
      continue

    if $t_{start} >= 0.0 \wedge t_{end} <= 1.0$ **then**
      $d \longleftarrow r.start - r.end$
      $s.start \longleftarrow r.start + t_{\text{start}} * d$
      $s.end \longleftarrow r.start + t_{\text{end}} * d$
      $V \longleftarrow addTraversedSpace(V, s.start, s.end)$

    if $isOccupied(r.end)$ **then**
      if $e.insertEndpointOccupied$ **then**
        $V \longleftarrow addEndpointOccupied(V, r.end)$
    else if $isFree(r.end)$ **then**
      if $e.insertEndpointFree$ **then**
        $V \longleftarrow addEndpointFree(V, r.end)$
  return V

5. VOLUMETRIC APPROXIMATION AND VOXEL STATE ESTIMATION

5.1 Iterative Volumetric Approximation

By utilizing the plane-filtered raycasting algorithm described in Section 4, it is possible to generate an artifact-free volumetric approximation of a given point cloud. This section describes the iterative algorithm used for the approximation as well as the simple voxel labeling scheme.

A summary is given in Algorithm 3. The volumetric representation is initialized by applying *plane-filtered raycasting* at the octree depth d_{init}. The result is a crude approximation of the point cloud. In order to improve this initial guess, each voxel is expanded into a subtree of voxels by utilizing iterative refinement. This process is based on a queue that is initialized with the voxel in question. The first step is to extract the next voxel from the queue and check if a sufficient number of rays has been assigned to it. If this number is lower than a predefined threshold n, the voxel is classified as *residual* and not refined any further. This is required since the voxel state estimation based on statistics requires a minimum amount of points.

In case there is a sufficient number of rays and the maximum octree depth d_{max} is not yet reached, the voxel is labeled by the $estimateState$ function. The labeling step at this point is based on a simple heuristic. If all observations of a voxel consist of traversing rays, the voxel is labeled as *free*. In case all observations are rays ending within a voxel, it must be occupied. Marking this kind of voxel right away as *occupied* would lead to a very crude approximation of static structures. Therefore it is classified as *occupied split*, meaning that it is an occupied voxel that can only be split into child voxels which are either *free* or *occupied*. If a voxel contains both types of rays, it is marked to be *split*. A

Algorithm 2: Filter phase of the plane-filtered raycasting. The ray in question is intersected with all OBBs.

Data: List of bounding boxes $B = \{b_1, \ldots, b_n\}$,
list of rays $R = \{r_1, \ldots, r_n\}$,
ray r.
Result: Ray offsets t_{start} and t_{end}.
Expanded list of rays $R = \{r_1, \ldots, r_n\}$,
boolean vector e with endpoint handling policy.
begin
  $t_{\text{start}} \longleftarrow 0.0$
  $t_{\text{end}} \longleftarrow 1.0$
  $e.insertFreeEndpoint \longleftarrow true$
  $e.insertOccupiedEndpoint \longleftarrow true$
  for $b_i \in B$ **do**
    if $intersection(r, b_i)$ **then**
      $t_{\text{near}}, t_{\text{far}} \longleftarrow getIntersection(r, b_i)$
      if $t_{near} <= 0.0 \wedge t_{far} >= 1.0$ **then**
        $t_{\text{start}} \longleftarrow -\infty$
        $t_{\text{end}} \longleftarrow \infty$
        $e.insertEndpointFree \longleftarrow false$
      else if $t_{near} > 0.0 \wedge t_{far} > 1.0$ **then**
        if $t_{near} < t_{end}$ **then**
          $t_{\text{end}} \longleftarrow t_{\text{near}}$
          $e.insertEndpointFree \longleftarrow false$
      else if $t_{near} < 0.0 \wedge t_{far} < 1.0$ **then**
        if $t_{far} > t_{start}$ **then**
          $t_{\text{start}} \longleftarrow t_{\text{far}}$
      else if $t_{near} > 0.0 \wedge t_{far} < 1.0$ **then**
        if $t_{near} < t_{end}$ **then**
          $t_{\text{end}} \longleftarrow t_{\text{near}}$
          $e.insertEndpointFree \longleftarrow false$
          $e.insertEndpointOccupied \longleftarrow false$
          $R \longleftarrow R \cup createRay(r, t_{\text{far}}, 1.0)$
  return $t_{\text{start}}, t_{\text{end}}, R, e$

result of the iterative approximation without the final state estimation can be seen in Figure 3.

If the states *split* or *occupied split* have been assigned to the voxel, it is subdivided. The number of traversing and penetrating rays of each of its child voxels are calculated by using *plane-filtered raycasting*. The speed of this process can significantly be increased if only the ray indices are stored within each voxel. Each child voxel is then added to the queue.

5.2 Final Labeling

The final labeling step $estimateFinalState$ is executed for each voxel that reaches the maximum octree depth d_{max}. Since the problem of voxel state estimation is not a simple one, the approach described here is to be considered only as a simple approximate solution in order to investigate the suitability of the volumetric representation for moving object detection. The voxel is labeled as either *occupied*, *moving object* or *residual* based on statistics, considering only the number of traversing rays and rays ending within the voxel. At this point, it is ruled out that the state *free* can be assigned to the voxel in question. This is because it is hard to distinguish the states *free* and *moving object* as soon as a voxel contains observations that imply that it has been occupied at some point in time. For this reason, the free space has

Algorithm 3: Overview of the volumetric point cloud approximation algorithm.

Data: List of rays $R = \{r_1, \ldots, r_n\}$,
octree depths d_{init} and d_{max},
minimum number of point threshold n.
Result: Volumetric representation V.

```
begin
    V ⟵ planeFilteredRaycasting(R, d_init)
    for v_i ∈ V do
        Q ⟵ {v_i}
        while Q ≠ ∅ do
            q ⟵ Q[0]
            if q.numRays < n then
                q.state = 'residual'
            else if q.depth < d_max then
                q ⟵ estimateState(q)
                if q.state = 'split' then
                    for c_i ∈ getChildren(q) do
                        c_i ⟵ recastRays(R, q.depth + 1)
                        Q ⟵ Q ∪ c_i
            else
                q ⟵ estimateFinalState(q)
            Q ⟵ Q \ q
    return V
```

already be carved out during the iterative refinement using the simple heuristic mentioned above.

The approach to labeling is based on the authors observation that a voxel that belongs to a moving object contains up to 5 orders of magnitude more traversing rays than rays implying occupancy. An occupied voxel in which the artifacts introduced by objects borders as described in Section 3.2 are present contains only 1 or 2 orders of magnitude more traversing rays. Equations 1 exploits this fact to calculate the probability for an occupied voxel p_{occ}.

$$p_{\text{occ}} = \frac{\log(o)}{\log(o+f)} \tag{1}$$

Equation 2 shows the probability p_{mov} for the voxel to be part of a moving object.

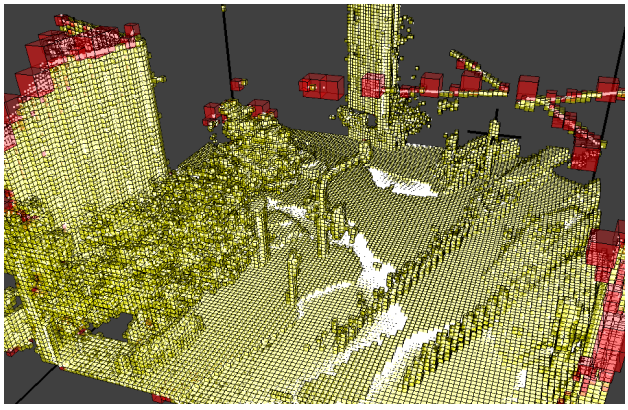


Figure 3. Iterative approximation of a street without the final state estimation (red=*occupied*, yellow=*split*; free space is not shown.

$$p_{\text{mov}} = 1.0 - p_{\text{occ}} \tag{2}$$

The probability p_{res} required to distinguish signal from noise is shown in Equation 3.

$$p_{\text{res}} = \frac{1.0}{(o+f)} \tag{3}$$

The variables f and o are the numbers of rays traversing the voxel and ending within it, therefore implying it to be either free or occupied respectively.

6. EVALUATION

6.1 Experimental Setup

The evaluation is based on the *MLS 1 - TUM City Campus*[2] dataset, which has been recorded by the measurement vehicle *MODISSA* of the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB). The dataset consists of about 2.25 billion georeferenced mobile laser scans (MLS) and covers an area of 0.17 km^2, which encompasses the TU Munich inner city campus and the surrounding streets. Measurements were recorded in 2016 utilizing two Velodyne HDL-64E LiDAR sensors mounted at an angle of 25 ° on the front roof of the vehicle. The georeferencing is based on navigational data provided by an Applanix POS LV navigation system that utilizes two GNSS antennas installed at the front and back of the car, an inertial measuring unit and an additional distance rotary encoder installed on one of the back wheels. The navigation data has been postprocessed to increase accuracy. The dataset is publicly available under a Creative Commons License.

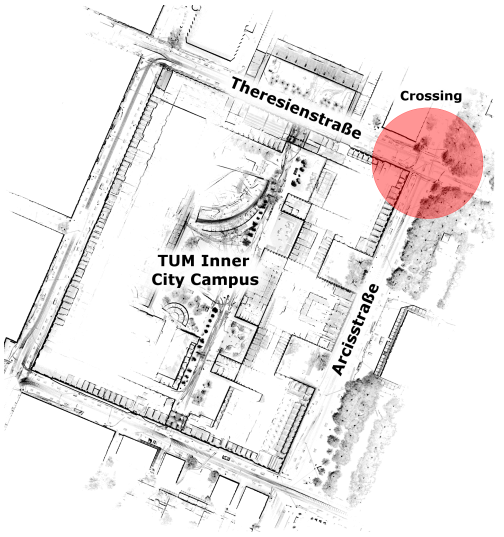
A crossing at the corner of Arcisstraße and Theresienstraße northeast of the TUM city campus has been chosen for evaluation (cf. Figure 4). The corresponding data contains 84 million range measurements. One side of the crossing is bordered by large buildings, while the other side consists of natural terrain in form of a meadow with large trees. The streets contain signs and traffic lights. Pedestrians and cars are participating in the traffic. The measurement vehicle approaches the crossing, waits there shortly and then turns left.

6.2 Qualitative Evaluation

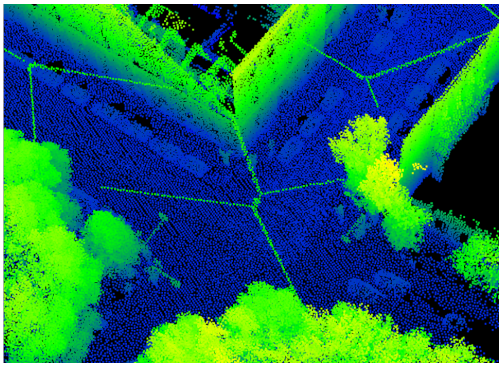
A labeled ground truth was not yet available at the time this paper has been submitted, therefore an evaluation of the plane-filtered raycasting algorithm is done by a human observer. Since the correct labeling is of minor interest apart from planar structures, a comparison to a ground truth would be of limited use for this work. The main focus of the evaluation is on the performance of the plane-filtered raycasting algorithm and the volumetric approximation of the point cloud. The following criteria are checked:

- What is the impact of the plane-filtered raycasting?
- Are all major planar surfaces correctly recognized? Are some missing or not correctly determined?

[2] http://s.fhg.de/mls1



(a)



(b)

Figure 4. The scene used for evaluation. (a) Overview. (b) Accumulated Point cloud of the crossing.

- Are planar surfaces approximated correctly by both segmentation and bounding box estimation?

And for the sake of completeness, the results of the simple voxel state estimation are also evaluated:

- Are solid structures and free space correctly classified?
- Which quality has the estimation of voxels that belong to moving objects?

The questions are answered by investigating a color-coded volumetric representation of the scene in combination with the point cloud the former one has been generated from. In addition to the above questions, the results are investigated for anomalies.

6.3 Runtime

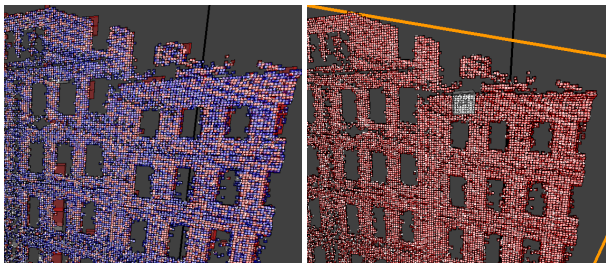
The evaluation is completed by a short discussion regarding the runtime of the plane-filtered raycasting and the iterative volumetric approximation.

7. RESULTS AND DISCUSSION

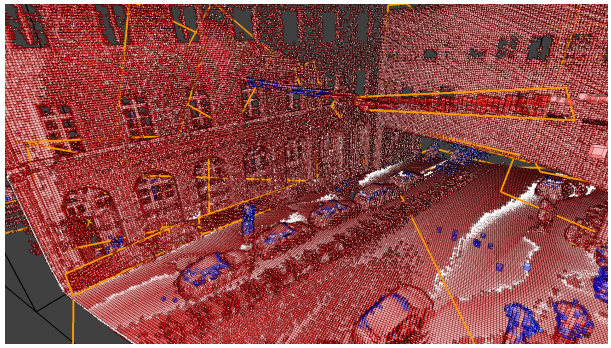
7.1 Results of the Qualitative Evaluation

Impact of Plane-filtered Raycasting – To illustrate the effect of the approach presented in this paper, the same scene has been generated with both normal raycasting and *plane-filtered raycasting*. In the first case, large parts of the walls are traversed by rays, which waters down the state of most of the voxels. This causes a mislabeling into the *moving object* class (cf. Figure 5(a)). Utilizing our raycasting approach in combination with the iterative refinement leads to a result in which every single voxel in proximity to a planar surface is classified correctly as occupied (cf. Figures 5(b)).

The capability of the iterative refinement approach is shown in Figure 5(c). Although the scene is quite complex from a geometric point of view, the approximation of the point cloud is very accurate, for example this can be seen at the ornaments of the windows, the cars and the powerline between the buildings. Even the cyclist in the middle of the street that is incorrectly classified as solid structure can easily be recognized by a human observer.



(a) (b)



(c)

Figure 5. The volumetric representation (a) without and (b,c) with plane-filtered raycasting (red=solid, blue=moving object).

Planar Surface Segmentation and Approximation – The method described in Section 4.1 is able to extract the most dominant planar structures such as the ground plane and building facades (cf. Figure 6), however there are anomalies caused by some planes which are either small in terms of their geometry or contain only a low number of points. In some cases arbitrary point clusters are recognized as planes.

In some cases these planes also accidentally mark moving objects. This happens especially in case of a moving car; when accumulating multiple laser scans, the object creates a long tube-like structure. Due to its planar roof, this also leads to a very large planar patch discovered by the plane detection.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-2-97-2018 | © Authors 2018. CC BY 4.0 License.

Very rare cases of improperly oriented bounding boxes were observed. It is assumed that this could be handled in future work by changing the way the oriented bounding box is determined.

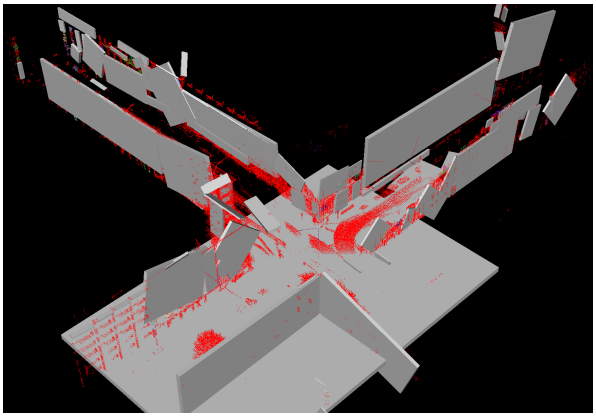


Figure 6. Planar surfaces (gray) extracted from an examplary point cloud (red). Most planar surfaces are correctly approximated. Some oriented bounding boxes are not aligned correctly.

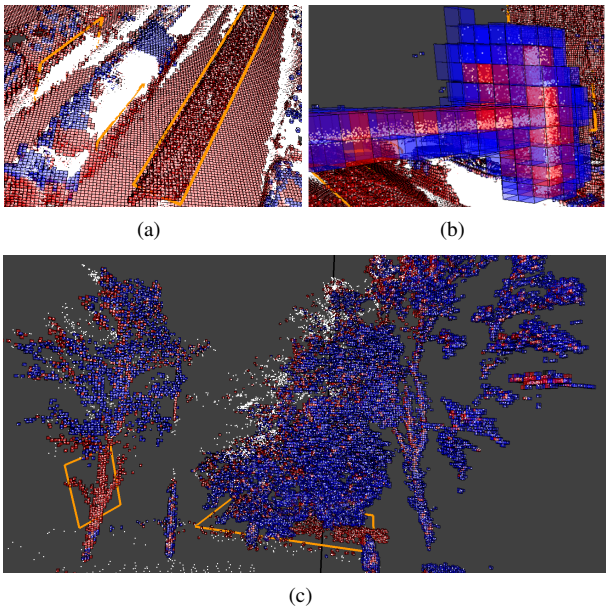
Voxel Labeling – Although a very simple heuristic has been applied to estimate free space as well as occupied space along planar structures, the results are persuasive. Free space is reliably carved out and thanks to the plane-filtered raycasting planar structures are artifact-free.

As one would expect, the naive state estimation method for the labeling of moving objects works only in well observed situations which correspond to the heuristic used for labeling. Apart from the direct vicinity of the sensor vehicle, this assumption is not always given. A further investigation led to the conclusion that labeling based on a single voxel is not a feasible approach. Utilizing the direct neighborhood of a voxel may be a way to increase the robustness of the labeling. But since the raw range measurements supply not only a much higher spatial resolution, but also a temporal order of range measurements, the authors came to the conclusion that moving objects are best handled before generating the volumetric representation.

Anomalies – As mentioned previously, there are planar structures created by moving objects which mislead the plane extraction algorithm (cf. Figure 7(a)). In addition, there are moving objects which have been mislabeled partly or as a whole. As can be seen in Figure 7(b), the artifacts induced by borders is still remaining, as the traffic light in the example is coated with voxels implying movement. This has been expected due to experiences from our previous works and the applied simple state estimation. The same applies to trees. The tree trunk is mostly recognized as solid structure, but the leaves and branches are almost completely estimated to be part of a moving object (cf. Figure 7(c)). This effect appears since only sometimes rays hit a leaf or another solid structure; which is exactly the effect used to determine whether or not a voxel contains a moving object. Since this applies pretty well to most parts of the tree, a detection algorithm based on this effect may be possible.

7.2 Discussion of the Runtime

The runtime per ray is 0.19 ms for normal raycasting. If the plane-based filter is applied, the runtime per ray grows up to 0.24 ms. This is an increase in runtime of about 26 percent. Both values are calculated by averaging over about 39000 points.



(a) (b)

(c)

Figure 7. Anomalies caused by the plane-filtered raycasting. (a) A moving car classified as solid due to the plane fitted onto its roof. (b) Object border traversal artifacts. (c) Tree trunks are solid, but leafs are recognized as moving objects (red=*solid*, blue=*moving object*).

Although the runtime per ray has grown, the overall time per scan has decreased in comparison to our previous work (Gehrung et al., 2016). Raycasting within a fixed resolution grid requires 31.3 seconds, whereas the iterative refinement-based approach presented in this paper only requires 19.9 seconds. This is due to the fact that during iterative refinement, most of the free space is only handled at a very crude resolution. No refinement is applied to a voxel that is believed to contain only free space. Since a ray mostly traverses free space, this leads to the speedup of 36 percent observed here.

8. CONCLUSION AND FUTURE WORK

This work shows that a plane filter for raycasting can be used to completely remove artifacts caused by discretization in the vicinity to planar structures. It was also shown that our iterative refinement algorithm leads to a good approximation of a given point cloud with respect to the geometric reproduction of its structure. Due to the iterative nature of the approach, it was possible to speedup the raycasting by 36 percent.

In order to either mark or completely remove moving objects, there are multiple possibilities. The simple state estimation investigated in this work shows that voxel labeling without considering the local neighborhood is not a feasible way. While it would also be possible to solve the problem by using geometrical features or scene knowledge based in context of the volumetric representation, the authors belief that handling the problem based on the rays directly would lead to a more simple and robust approach. This is due to the fact that working on rays permits the exploitation of the higher spatial resolution and temporal order of the measurements.

Once a reliable voxel representation without moving objects has been generated, another topic of interest is the actual change detection in large scale urban environments.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-2-97-2018 | © Authors 2018. CC BY 4.0 License.

References

Aijazi, A. K., Checchin, P. and Trassoudaine, L., 2013. Detecting and Updating Changes in Lidar Point Clouds for Automatic 3D Urban Cartography. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-5/W2, pp. 7–12.

Azim, A. and Aycard, O., 2012. Detection, classification and tracking of moving objects in a 3D environment. In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pp. 802–807.

Gehrung, J., Hebel, M., Arens, M. and Stilla, U., 2016. A Framework for Voxel-based Global Scale Modeling of Urban Environments. In: *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. XLII-2/W1, pp. 45–51.

Gehrung, J., Hebel, M., Arens, M. and Stilla, U., 2017. An approach to extract moving objects from MLS data using a volumetric background representation. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. IV-1/W1, pp. 107–114.

Girardeau-Montaut, D., Roux, M., Marc, R. and Thibault, G., 2005. Change detection on point cloud data acquired with a ground laser scanner. In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 36-3/W19, pp. 30–35.

Hebel, M., Arens, M. and Stilla, U., 2013. Change detection in urban areas by object-based analysis and on-the-fly comparison of multi-view als data. *ISPRS Journal of Photogrammetry and Remote Sensing* 86, pp. 52–64.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. and Burgard, W., 2013. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots* 34(3), pp. 189–206.

Litomisky, K. and Bhanu, B., 2013. *Removing Moving Objects from Point Cloud Scenes*. Vol. 7854, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 50–58.

Pagac, D., Nebot, E. M. and Durrant-Whyte, H., 1996. An evidential approach to probabilistic map-building. In: *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 745–750 vol.1.

Rabbani, T., van den Heuvel, F. A. and Vosselman, G., 2006. Segmentation of Point Clouds Using Smoothness Constraint. In: *ISPRS 2006 : Proceedings of the ISPRS Commission V Symposium*, Vol. 35, International Society for Photogrammetry and Remote Sensing (ISPRS), pp. 248–253.

Schachtschneider, J., Schlichting, A. and Brenner, C., 2017. Assessing temporal behavior in LIDAR point clouds of urban environments. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. XLII-1/W1, pp. 543–550.

Underwood, J. P., Gillsjö, D., Bailey, T. and Vlaskine, V., 2013. Explicit 3D change detection using ray-tracing in spherical coordinates. In: *2013 IEEE International Conference on Robotics and Automation*, pp. 4735–4741.

Williams, A., Barrus, S., Keith, R. and Shirley, M. P., 2003. An efficient and robust ray-box intersection algorithm. *Journal of Graphics Tools* 10, pp. 54.

Wolf, D. and Sukhatme, G. S., 2004. Online simultaneous localization and mapping in dynamic environments. In: *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04*, Vol. 2, pp. 1301–1307.

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

Zeibak, R. and Filin, S., 2008. Change detection via terrestrial laser scanning. In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 36, pp. 430–435.