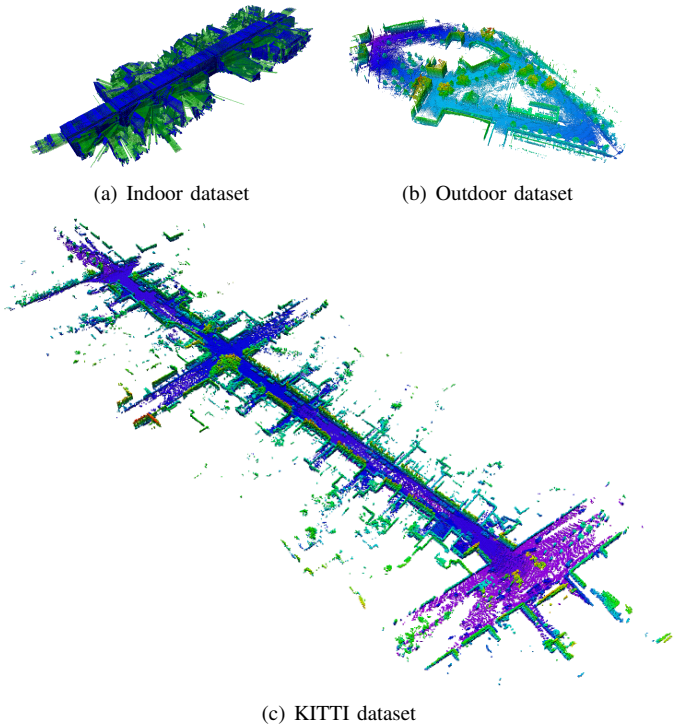
Super Rays and Culling Region for Real-Time Updates on Grid-based Occupancy Maps

Youngsun Kwon[1], Donghyuk Kim[1], Inkyu An[1], and Sung-eui Yoon[2]

***Abstract*—In this paper, we present two novel approaches, Super Rays and Culling Region, for efficiently updating grid-based occupancy maps with point clouds. Rays, which traverse from the sensor origin to the sensor data, update the occupancy probabilities of a map representing an environment. Based on the ray model, we define a *super ray* as a representative ray to multiple rays having the same traversal patterns during the map updates. Our super rays utilize the geometric information of rays and reduce the number of points used for updating the map. For constructing super rays efficiently, we propose mapping lines for handling 2D and 3D cases from an observation that edges or grid points branch out the traversal patterns on the map. Furthermore, we introduce a *culling region* using the occupancy states of the updated map for reducing redundant computations occurred in updates. The super rays perform the update process in a single traversal, and the culling region reduces the number of unnecessary traversals for updating the map. As a result, our combined method improves the update performance without compromising any representation accuracy of a grid-based map.**

We test the update performance of the proposed method using public indoor and outdoor datasets. Our combined approach shows up to 11.8 times and 2.8 times performance improvement over the state-of-the-art update methods of grid-based maps in the indoor and outdoor scenes, respectively. Also, we compare the update speed and the representation accuracy of our method using KITTI dataset over the state-of-the-art learning based occupancy maps. In a navigation scenario that raw point clouds are acquired in 10 Hz, our method shows the best performance on the update speed and thus the highest representation accuracy within a given time.



(a) Indoor dataset (b) Outdoor dataset

(c) KITTI dataset

Fig. 1. These figures visualize the map representations for three public datasets. a) Blue and green cubes represent occupied and free spaces, respectively. b, c) We use heat colors to represent relative heights for visualizing the occupied cells of the maps.

I. INTRODUCTION

MANY robotic systems use various sensor data for understanding their environments. Point clouds have been known as an effective representation of the environment around robots, and are easily captured in recently emerging, inexpensive consumer-level depth sensors as well as laser scanners. The point clouds are represented by a large number of points representing geometric information of environments in high resolutions, yet with various levels of sensor noise. In applications such as path planning or collision detection, it is difficult to use such point clouds directly because of the sheer amount of generated data themselves as well as the noise.

To address these issues, various occupancy map representations such as grid-based maps [1], [2] and continuous maps [3], [4] have been proposed to represent occupancy states of an environment, for reducing the memory requirement and considering the uncertainty of point clouds. Grid-based maps reflect geometric information of the data with high performance on updates by representing the environment as a set of independent cells. On the other hand, continuous maps learn geometric correlation of the data for classifying occupancy state of the environment, which results in accurate representations.

Recent applications use such occupancy maps to achieve better performance for their goals. Furthermore, real-time applications require to fast process the sensor data acquired over time for reacting to dynamic environment such as avoiding a suddenly appeared obstacle. Grid-based maps are appropriate for these applications thanks to the high update speed.

Unfortunately, constructing such occupancy maps out of point clouds can still take a high computation overhead on sensors with high-frequency. When we use a lower resolution for the grid-based map, we can achieve a high update speed but comes with a low representation accuracy, which may result in serious problems for various robotic operations; for example, inaccurate collision detection in the map with low solution may result in the collision of robot with surrounding obstacles.

[1]Youngsun Kwon, Donghyuk Kim, and Inkyu An are with School of Computing, KAIST, Daejeon, South Korea. e-mail: youngsun.kwon@kaist.ac.kr, donghyuk.kim@kaist.ac.kr, inkyu.an@kaist.ac.kr

[2]Sung-eui Yoon is with Faculty of School of Computing, KAIST, Deajeon, South Korea. e-mail: sungeui@kaist.edu

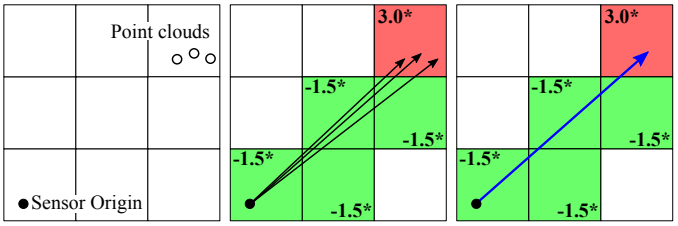


Fig. 2. This figure represents an overview of super ray when we have the new measurements as shown in *a*). *b*) and *c*) represent occupancy probabilities of cells after updating the 2D grid map with different methods. The green and red cells have the free and occupied states, respectively. The bold numbers with * notation in cells indicate that those cells are classified into fully occupied or fully free state. In *b*), the state-of-the-art method updates the same set of cells for three different rays, which causes redundant computation on overlapped traversals on the cells. The blue ray in *c*) is a super ray computed out of those three rays in *b*). The super ray updates the map with a single traversal on the cells. In this figure, we use $l_{occ} = 1.7$, $l_{free} = -0.8$, $l_{max} = 3.0$, and $l_{min} = -1.5$.

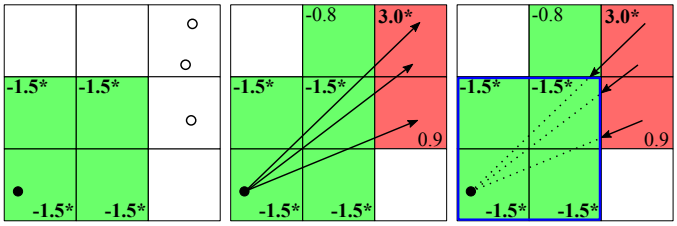


Fig. 3. This figure shows an overview of our culling region, given the new measurements as shown in *a*). In *b*), the prior method causes redundant computation on traversals on the cells having fully-free states. The blue box in *c*) is a culling region that prevents the three rays to traverse the fully-free cells for updates. In this figure, we use the same setting with Fig. 2.

Main contributions. In this paper, we present novel, efficient map update methods based on super rays as well as culling region, while achieving high performance without compromising the representation accuracy of grid-based maps. Specifically, we propose to use super rays of points as our main update method for maps, where the ray is a common model for extracting geometric information from point clouds; for example, the space between a sensor origin and a point of point cloud has non-collision. A super ray is a representative ray for a set of rays, and is constructed in a way that updating the map with those super rays traverses the same set of cells with original rays. To generate such super rays given input points, we propose to use a mapping line for updating 2D maps (Sec. IV-A and Sec. IV-B) , and generalize it to 3D maps using the 2D approach (Sec. IV-C). Furthermore, we propose a culling region that uses occupancy correlation between scans and reduces redundant computations by stopping unnecessary traversals of rays in advance (Sec. IV-E).

To demonstrate benefits and robustness of our methods, we test our methods and prior works with a variety of cases. We first test the update performance with two public datasets (Fig. 1-(a) and (b)) for the grid-based maps such as uniform 3D grid map and octree map. We found that our method combined with super rays and culling region is robust enough to show the performance improvement, 6.3 and 1.8 times improvement across a diverse set of configurations, compared to prior works in the indoor and outdoor scenes, respectively (Sec. V-A). Furthermore, we provide the update speed and the representation accuracy of various mapping algorithms using a public KITTI dataset [5], for discussing practical benefits of real-time updates. In this test, we found that our combined method can give positive effects to such navigation in practice by showing the best frequency of updates (Sec. V-B).

A preliminary version of this paper has appeared in ICRA 2016 [6]. In this extended paper, we provide a new methodology of generating super rays in complex 3D case. Our new concept, mapping plane in 3D, generates super rays out of given point clouds with our similar approach, mapping line in 2D. In addition, we propose a new update method, culling region based updates, for utilizing the occupancy states of updated maps, while the super rays use geometry property of maps. We also perform various tests to demonstrate the improvement on update performance as well as the practical benefits of real-time updates.

II. RELATED WORK

In this section, we discuss prior works on map representations modeling environments and their updating methods.

A. Occupancy Maps

A point cloud is one of the most common sensor data captured by a depth sensor or a laser range finder. Point clouds themselves can serve as a map representation for the environment under the study. Recently, many affordable consumer-level depth sensors become available. Some of the recent works use point clouds and apply them directly to applications (e.g., collision detection [7]). Nonetheless, point clouds can have an excessive amount of points especially for large-scale scenes, and more severely they can have inherent sensor noise. Due to the aforementioned issues, many prior approaches [8], [9] convert point clouds to other representations (e.g., triangular meshes) in order to process them in a simple and efficient way.

Learning-based occupancy maps [3], [4] have been studied to estimate occupancy states of the environment by learning an implicit correlation of occupancies through sensor data. These methods can predict occupancy states of unmeasured regions, resulting in high representation accuracy. A well-known drawback of such maps, however, the is low performance in the online updates for learning. Recent works [10], [11] handled the issue on updating continuous maps, but the performances are not sufficient to deal with a high amount of point clouds generated by a sensor in real time. We compare our method against these approach in Sec. V-B

Grid-based maps partition environment into cells representing the occupancy probabilities and shows high performance on updates. While simple uniform grid maps [1], [12] are proposed early and succeed in various robotic applications, it has certain limitations. Its main drawback is that it requires a tremendous size of memory, when we handle large-scale environments or require high resolutions for accurate representations.

Tree-based representations such as quadtree maps in 2D and octree maps in 3D have been studied in order to overcome the problem. The octree map divides a 3D space into 8 sub-spaces that have the same volume, and represents a space with a cell having an occupancy state. When all the children cells have the same states, this map results in a compact representation than the grid map. Thanks to this useful property, tree-based representations have been used for modeling environments [13], [14]. Payeur et al. [15] suggested to augment octrees with probabilistic occupancy states for considering sensor noise. Recently, OctoMap [2], [16] adopted unknown states for representing regions occluded by obstacles.

Coenen et al. [17] considered unknown states as regions with a high probability having collision. Many applications such as navigation [18] and point cloud compression [19] have been developed based on this octree map representation.

In this paper, we assume that a robotic application uses grid-based occupancy map representations to deal with point clouds efficiently. For such applications, we develop an accurate, yet efficient update method for these maps.

B. Real-Time Updates on Grid-based Maps

A point captured from a sensor means that we do not have any collision along the line segment connecting the sensor origin and the point. We need to reflect this information on grid or tree based occupancy maps. This process can be very slow, especially when we have many points in large-scale environments and applications requiring high-resolution maps.

A useful approach to accelerate the speed of updating tree based maps from point clouds is to decide an adequate resolution of the tree based map, instead of updating the full resolution of the map. Along these lines, different methods have been proposed for using various resolutions depending on objects [20] or statistics of updated states of each cell [21]. While it uses adaptive resolutions, its performance can vary depending on parameters related to the resolutions, and the updated maps can be significantly different from the original results.

In graphics literature, various techniques traversing grids have been studied for ray tracing, a specialized form of collision detection [22], [23]. Wald et al. [24] proposed a method to traverse a grid with coherent rays. This work finds a set of cells by packetizing rays that traverse similar space in the grid for intersection tests on ray-tracing. This work is not applicable to grid-based occupancy map because of the lack of information that how many rays traverse a grid cell for updating occupancy of the map. Nonetheless, we are inspired by this approach for efficiently computing our super rays out of point clouds.

Voxel filtering of PCL [25] is used frequently to accelerate speed of processing point clouds in the robotics literature. This method decreases the processing time by reducing the number of points using voxels, while sacrificing the representation accuracy of maps in the same spirit of using adaptive resolutions. Departing from these prior approaches, our method maintains the original representation accuracy of occupancy maps and improves the overall update performance by utilizing access pattern of updating maps with point clouds.

The well-known occupancy map representation, OctoMap library [2], uses the 3DDDA (3-Dimensional Digital Difference Analyzer) based algorithm [22] as an exact method to update the map with point clouds. Furthermore, the library provides a batch based updates specialized for tree-based map representations. Our approaches can be applied to the 3DDDA as well as the batch based updates, and can accelerate the update speeds of both prior methods.

III. Overview

We give backgrounds on grid-based occupancy maps and provide an overview of our methods.

A. Backgrounds on Grid-based Occupancy Maps

Point clouds consist of points captured by a depth sensor or laser range finder. When a point is reported by the sensor, it implies that the space between the sensor origin and the point is empty. As a result, we associate a ray with the point starting from the sensor origin. Thus, the problem can be transformed into map traversal along the ray from the sensor origin toward the reported end point.

Such a ray provides two kinds of state information about space under the study: occupied and free states. The end point of the ray has the occupied state, since the sensor reports an object on that particular point. On the other hand, other space that the ray passes through has the free state. This information is critical for various applications such as motion planning. As a result, it is very important to construct an occupancy map accommodating this information acquired from sensors. Unfortunately, data captured by sensors are plagued by various levels of noise. To consider such noise, map representations commonly use an occupancy probability, instead of simple boolean occupancy states of occupied or free.

Grid-based maps such as uniform grid-map or octree-map partition an environment into grid cells representing the occupancy probabilities, and update them to maintain the recent occupancy states of the environment through sensor measurements. Such maps use a ray-casting algorithm to find cells where rays modeled by point clouds traverse on grid from the sensor origin to the end points.

A ray-casting algorithm such as 3DDDA [22] computes adjacent cells on the uniform grid where a ray traverses from the start to the end cells containing the sensor origin and end point, respectively. The cells traversed by rays are updated to have free states, and the end cells are updated to have occupied states.

On the assumption that cells of a map are independent of occupancy states, the likelihood of occupancy, $P(x|z_{1:t})$, represents the occupied state of a cell, x, given sensor measurements, $z_{1:t}$, from the initial time step 1 to the current time step t, and can be modeled by the Bayes rule and Markov assumption [26] as follows:

$$\frac{P(x|z_{1:t})}{1-P(x|z_{1:t})} = \frac{P(x|z_t)}{1-P(x|z_t)} \frac{P(x|z_{1:t-1})}{1-P(x|z_{1:t-1})} \frac{1-P(x)}{P(x)}. \quad (1)$$

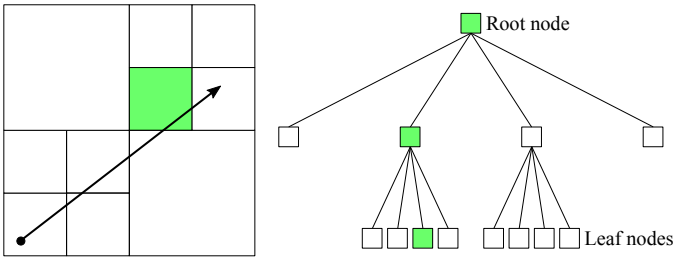


Fig. 4. This figure shows an example of updating a cell in the 2D quadtree map. The ray traverses the cell colored by green in the left figure. To maintain the tree structure of the quadtree, we update the occupancy probabilities of all nodes from the leaf to the root, colored by green in the right figure.

For the fast update to the map, a well-known approach using the log-odd notation $\mathrm{L}(x) = \log\left[\frac{P(x)}{1-P(x)}\right]$ transforms the prior equation into:

$$\mathrm{L}(x\,|\,z_{1:t}) = \mathrm{L}(x\,|\,z_{1:t-1}) + \mathrm{L}(x\,|\,z_t) - L(x). \tag{2}$$

Based on this equation, the OctoMap framework [2] uses a prior probability $P(x) = 0.5$ representing the unknown state and the simple inverse sensor model on the log-odd notation $L(x\,|\,z_t)$ defined as the following:

$$L(x\,|\,z_t) = \begin{cases} l_{occ}, & \text{if the end point of a ray is in the cell,} \\ l_{free}, & \text{if a ray passes through the cell.} \end{cases}$$

As a result, the simple form of Eq. 2 results in the efficient update of occupancy probability at a cell using fast addition operations.

When a cell has an occupancy probability that has been accumulated over long time steps, a new input data that conflicts with the current state of the cell cannot change the state immediately. This over-confidence problem can frequently occur in dynamic environments. Yguel et al. [27] solve the problem by using a clamping policy that limits the occupancy probability of a cell based on the minimum and maximum state bounds: l_{min} and l_{max} for free and occupied states, respectively. The state of a cell limited by either one of those two bounds is considered to be fully free or fully occupied with a high occupancy probability. Fig. 2 and Fig. 3 show illustrations of updating the grid map in 2D given point clouds.

A uniform grid map consists of cells having the same size determined by a user-defined resolution. We find a set of traversed cells of rays using a ray-casting algorithm, and update their occupancy probabilities using the update rule (Eq. 2).

Hierarchical representations. Using the uniform grid with the ray-casting updating method is simple and intuitive to represent occupancy states of an environment, but can require a huge size of memory, especially when we have a high resolution. To overcome the problem, quadtree maps in 2D or octree maps in 3D are proposed to have a tree data structure for providing various resolutions, resulting in more compact representations. Such maps merge 4 or 8 sub-divided children nodes that have the same occupancy probabilities into one parent cell representing a space in a coarser level (Fig. 4). The properties of the tree structure can be used efficiently for collision detection or motion planning, although it generates the hierarchical update from the leaf to all the parent cells unlike the grid map. The recent OctoMap [2] framework avoids the duplicated updates generated from updating the tree structure. This is done by batching leaf cells that all rays traverse on the uniform grid with the maximum resolution, before updating the maps. The batching-based method updates tree-based maps in a single time using the batched cells, resulting in the performance improvement.

B. Motivations

Grid based occupancy maps have been widely used for various applications. We, however, found that updating these maps can take a huge amount of computation time compared to the frequency of sensor measurement. Furthermore, we have identified that the original update method for occupancy maps has redundant computations, because of the discrete nature of grid based representations. For example, Fig. 2-*b*) shows three different rays traverse the same set of cells in 2D grid, while these rays have different end points. When we update the map with these rays one-by-one, duplicate computations are made on traversal and updating through the exact same set of cells, resulting in lower performance.

Additionally, certain traversals do not contribute at all to cells whose occupancy probabilities are out of range of the min and max bound values due to the clamping policy, as shown in Fig. 3-*b*). We define the traversal as an unnecessary traversal. These problems frequently occur because the original update method does not consider the discrete nature and occupancy states of map representations.

C. Overview of Our Approaches

To overcome the aforementioned problems, we first propose an update method utilizing super rays for occupancy maps. We define a super ray as a representative ray to rays associated with given points (Fig. 2). The super ray is constructed in a way that traversing those rays for updating the map requires to access the same set of cells in the map. We then update the map by traversing those cells with the super ray only a single time, while considering the number of points associated with the super ray, thus removing redundant computation and achieving higher performance.

On top of super rays, we propose a culling region based method by culling out traversal on cells having already saturated occupancy probabilities to fully-free states. We define a culling region as a set of cells where traversals from those cells to the sensor origin are unnecessary (Fig. 3). If rays encounter the culling region while traversing from their end points to the sensor origin, our method stops the remaining traversals from a cell entering the culling region to the sensor origin. As a result, we can achieve a higher performance thanks to removing unnecessary traversals.

Our two approaches are orthogonal and easily applied to 3DDDA and batch based methods on grid based occupancy maps. Ours improve the performance of updates compared to the prior work, and these two methods complement each other, as shown in the result section (Sec. V). Overall, our method combined both with super rays and the culling region shows the best performance on average.

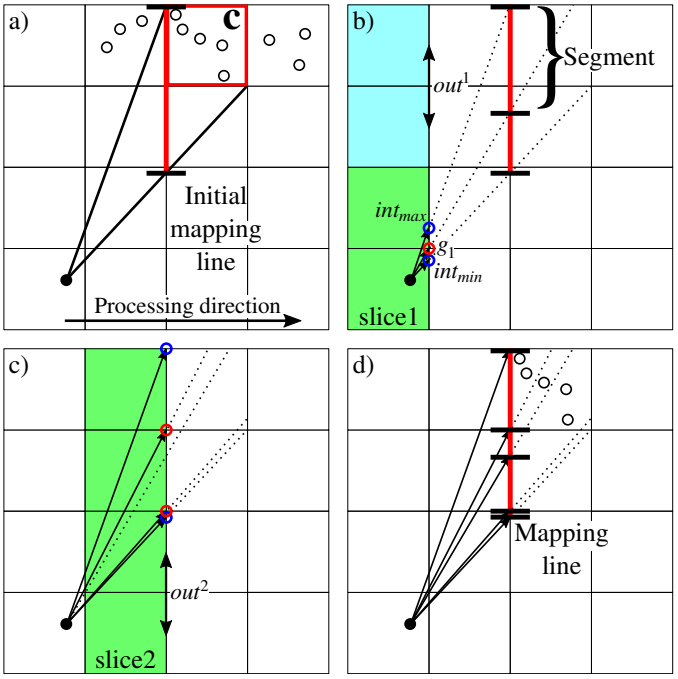


Fig. 5. This figure shows an example of updating a mapping line for a cell c. The red grid point g_1 in b) divides the seed frustum into two sub-frustums, and its projected point generates two segments on the mapping line. In c), two grid points in out^2 in the slice 2 also generate two more segments in the mapping line shown in d).

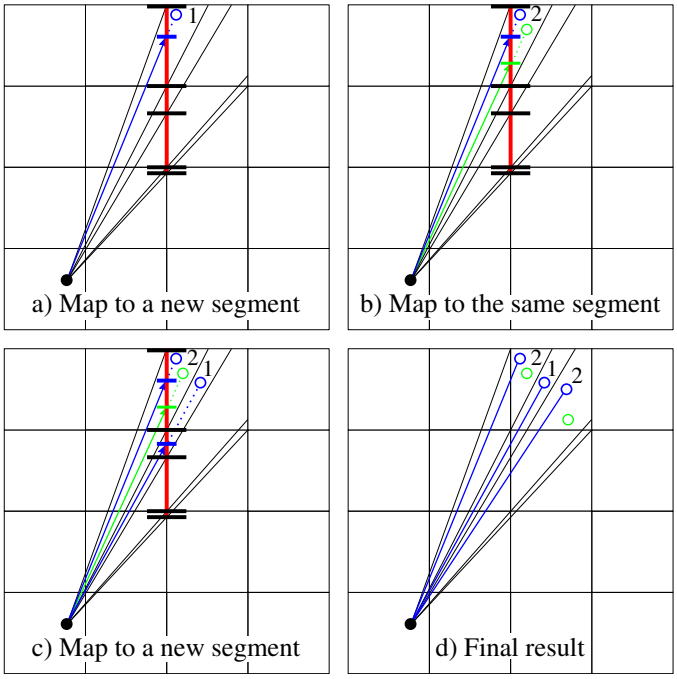


Fig. 6. This figure shows how we generate three different super rays out of five rays using the generated mapping line. a) A new ray maps to a new segment, and we treat it as a new super ray with a weight of one. b) Another ray maps to the prior segment, and we increase its weight to two. c) The new ray maps to a new segment and a new super ray is assigned to it. d) shows the final, three super rays with their weights.

IV. Updates using Super Rays and Culling Region

In this section, we explain our approaches in detail. We first propose a mapping line and explain how to use it for generating super rays starting from a single, seed frustum containing all the points of a cell in the map (Sec. IV-A). We then identify which points in a cell have the same set of traversed cells for updating the map based on the mapping line (Sec. IV-B). To extend the concepts of a 2D case into a 3D case, we introduce a mapping plane conceptually used for generating super rays in the 3D case, and then explain how to solve the 3D problem efficiently using mapping lines of the 2D case (Sec. IV-C). We then explain how to update cells that each super ray passes without compromising the representation accuracy of maps (Sec. IV-D). In the end, we also propose to use a culling region for improving performance on maps by considering occupancy states of the maps (Sec. IV-E).

A. Generating a Mapping Line

In general, point clouds are defined in the sensor coordinate system, while occupancy maps model them in the world coordinate. Based on the assumption that we know the position and orientation for the sensor in the world coordinate, we transform point clouds from the sensor coordinate to the world coordinate, and update the map with them.

For each cell, c, in the map, we conceptually construct a seed frustum starting from the sensor origin to the cell box containing all the points in the cell c, the red box shown in Fig. 5-a). Starting from the seed frustum, we partition it into multiple ones, each of which traverses the same cells of the map. To do this, we design our algorithm to access grid cells slice-by-slice, where a slice contains cells in a line for the 2D data. For this process, we pick an axis, i.e., X or Y axis, for computing such slices, and treat it as a processing direction. Fig. 5 shows a case that X axis is the processing direction.

For identifying which points are mapped to the same super ray, we introduce a *mapping line*, which is a line segment that overlaps between the cell c and the slice containing the cell c. Fig. 5-a) shows an initial mapping line. Each segment of the mapping line corresponds to one of the super rays, while we also use the terms of frustums or super rays conceptually to explain our geometric concepts. The initial mapping line starts with a single line segment representing a super ray, but can be divided into multiple segments corresponding to multiple super rays.

One key observation for generating the mapping line to represent different frustums is that the traversal patterns of cells differ along grid points, when we consider cells slice-by-slice. Fig. 5-b) shows a grid point, shown in the red circle within the initial frustum. Given the grid point, the traversed cells differ, and thus we need to partition the seed frustum into two different ones, resulting in two segments on the mapping line (Fig. 5-b)). Based on this observation, the key operations are how we efficiently generate the mapping line within the frustum.

Let out^i of i-th slice to denote the faraway line of the slice along the processing direction. Fig. 5-b) shows an example of out^1 for slice 1. We can then compute the two intersection points, int_{min} and int_{max}, of the seed frustum for each i-th slice like the blue circles in Fig. 5.

Algorithm 1: BUILD MAPPING LINE

Input: C_{box}, a cell box in 2D, O, a sensor origin in 2D
Output: M_{line}, a mapping line

$M_{line} \leftarrow InitMappingLine(C_{box})$
$S_{slice} \leftarrow InitSlices(O, C_{box})$
for i *in* $1 : length(S_{slice}) - 1$ **do**
  $g \leftarrow ComputeGridPoints(S_{slice}[i], C_{box})$
  for j *in* $1 : length(g)$ **do**
    // project onto mapping line
    $M_{line}.insert(Projection(g_j))$
return M_{line}

Suppose that the first slice containing the sensor origin is slice 1 and the last slice containing point clouds is slice N. Our algorithm of generating a mapping line works in an iterative manner from slice 1 to slice $N-1$. To find grid points that differentiate the traversal pattern, we first compute two points int_{min} and int_{max} in out^i of each slice starting from slice 1. We then project all the grid points between int_{min} and int_{max} onto the mapping line. Suppose that there are m grid points, $g = \{g_1, g_2, \cdots, g_m\}$. These grid points partition the current frustums into at most $m+1$ sub-frustums, resulting $m+1$ corresponding segments on the mapping line. Each pair of two consecutive points projected onto the mapping line implicitly defines a segment and its associated frustum (and its super ray). Note that we can easily find these grid points and compute segments of the mapping line thanks to the discrete nature of occupancy maps, resulting in a fast update method. The pseudocode of generating a mapping line for a seed frustum is shown in Alg. 1.

B. Generating Super Rays using the Mapping Line

After we generate the mapping line of each cell at the prior step, we use it for computing which rays should be merged into the same super ray. To perform this process, we map all the input rays onto the mapping line and count how many rays are assigned to each segment of the mapping line (Fig. 6).

The rays assigned to the same segment of the mapping line have the same traversal patterns in terms of cells traversed for updating the map. We can merge the rays into a single super ray with a weight as the number of merged rays.

We use the weight information associated with a super ray to efficiently update the occupancy map without losing the representation accuracy, because the super ray updates all the same set of cells where its associated rays traverse. We skip here proving the completeness of using the mapping line to classify rays traversing the same set of cells. Instead, we show the completeness of our algorithm as Theorem 1 in Sec. VII.

C. Extension to the 3D Case

In this section, we explain how we extend our prior 2D approach into handling 3D point clouds. We first introduce a concept of *mapping plane* for generating super rays for the 3D case. Different regions in the mapping plan indicate different traversal patterns. We then propose to use three orthogonal mapping lines defining such different regions in the mapping plane, in order to efficiently handle the 3D case. For the sake of clarity, these corresponding concepts in 2D and 3D cases are summarized in Table I.

TABLE I
CORRESPONDING CONCEPTS OR 2D AND 3D CASES ARE SUMMARIZED.

Case	2D	3D
Data structure for generating super rays	Mapping line	Mapping plane
Element of the mapping data structure	Segment	Region
Geometric entity differentiating traversal patterns of rays	Grid points	Edges of cells

Similar to the 2D case, we first compute a bounding volume containing point clouds in the map representation. We also construct a seed frustum traversing to the volume, and then partition the frustum into sub-frustums, each of which traverses the same set of cells.

The key observation for the 3D case is that traversal patterns of cells differ along edges of cells, not just at the grid points. Based on the observation, we can partition the seed frustum into sub-frustums, each of which has the same traversal pattern and thus is constructed as a super ray. Fig. 7 shows an example of the mapping plane consisting of regions associated with super rays. In this example, we pick one of the planes that are orthogonal to axes, e.g., $z = d$, and explain our approach based on this example plane for the sake of clarity; other mapping planes can be treated in a similar manner. As you can see in Fig. 7, those projected lines, i.e., red, green, and blue lines, in the mapping plane creates many complex regions.

Conceptually, we can use the mapping plane for generating super rays in a similar manner to using the mapping line for the 2D case described in Sec. IV-B. Simply speaking, we map all input rays onto the mapping plane and count how many rays are assigned to each region of the mapping plane. The rays assigned to the same region of the mapping plane have the same traversal pattern in terms of cells traversed for updating the map. Using the mapping plane, we can merge multiple rays into a single super ray. The completeness of our algorithm using the mapping plain to generate super rays is given as Theorem 2 in Sec. VII.

We now go into details of our 3D approach using mapping plane to generate super rays, starting from introducing geometric values shown in Fig. 7. Let (x_i, y_j) be a 2D point, where the 3D grid point (g_x, g_y, g_z) is projected onto the plane $z = d$. They are expressed as follows:

$$x_i = \left(\frac{d - o_z}{g_z - o_z}\right)(g_x - o_x) + o_x, \tag{3}$$

$$y_j = \left(\frac{d - o_z}{g_z - o_z}\right)(g_y - o_y) + o_y, \tag{4}$$

where the line $x = x_i$ partitions the plane along the X axis, since a ray mapped on the left side of the line has a different traversal pattern to another ray mapped on the right side. The line $y = y_j$ partitions the plane along the Y axis, in the same

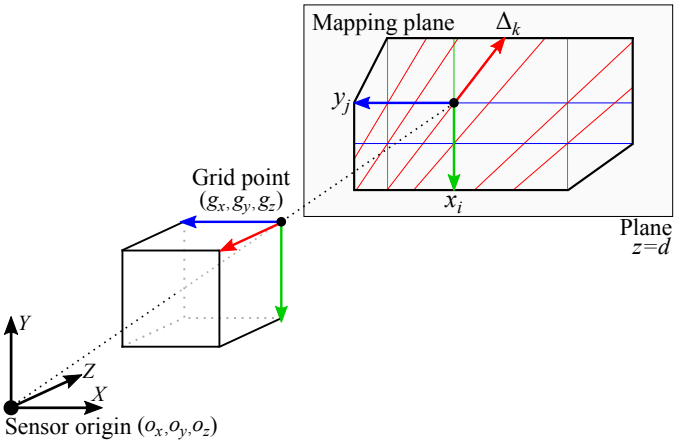


Fig. 7. This figure shows an example of a mapping plane on a plane $z = d$ in the 3D case. The projected lines, which three edges (red, green, and blue) of all grid points are projected to, partition the plane into regions, each of which is associated with a unique traversal pattern.

manner. Note that the value x_i is computed only with X and Z coordinates of the grid point, without the Y coordinate. As a result, the line $x = x_i$ can be computed in the 2D $X-Z$ space. Another partitioning line $y = y_j$ is treated similarly in the $Y-Z$ space.

For the red edge projected in the mapping plane $z = d$ shown in Fig. 7, it partitions the plane in a slant line. Its gradient, Δ_k, is computed by the slope from the sensor origin (o_x, o_y) and the projected point (x_i, y_j):

$$\Delta_k = \frac{y_i - o_y}{x_j - o_x} = \frac{g_y - o_y}{g_x - o_x}. \tag{5}$$

While the slant line seems to behave differently from other orthogonal lines, we can know that the gradient of the slant line consists of only X and Y coordinates without the Z coordinate, indicating that this can be treated in the 2D $X-Y$ space.

We now explain the geometric interpretation of utilizing the mapping plane to generate super rays, shown in Fig. 8. Let x_i and x_{i+1}, y_j and y_{j+1}, and Δ_k and Δ_{k+1} be the X, Y, and gradient values of lines, respectively, which construct the smallest region of the mapping plane. To classify rays of point clouds having the same traversal patterns and then generate super rays, we map all the rays into one of the regions on the mapping plane. The classification task of a ray can be expressed as:

$$\begin{cases} (a)\, x_i \le x_p < x_{i+1}, \\ (b)\, y_j \le y_p < y_{j+1}, \\ (c)\, \Delta_k \le \Delta_p < \Delta_{k+1}, \end{cases} \tag{6}$$

where (x_p, y_p) is a projected point of a ray onto the mapping plane and Δ_p is a gradient between the sensor origin and the projected point.

Implementation of generating super rays. We use Eq. 6 to classify rays with the same traversal patterns and generate super rays. Eq. 6 consists of three sub-tests that each of them is computed with two of three coordinates, as we mention in Eq. 3, Eq. 4 and Eq. 5. Each sub-test is identical to process finding segments of mapping line for generating super rays in 2D. For example, Eq. 6-(a) uses X and Z coordinates

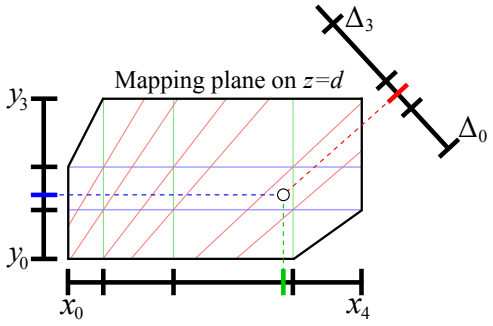


Fig. 8. This figure shows a process to generate super rays using mapping plane. In this example, we project a ray on a region of the mapping plane for finding the traversal pattern of a ray.

Algorithm 2: GENERATE SUPER RAYS

Input: P, a set of points in a cell, O, a sensor origin
Output: S_{ray}, a set of super rays

$C_{box} \leftarrow ComputeCellBox(P)$
$M_{xy} \leftarrow BuildMappingLine(C_{box}(X,Y), O(X,Y))$
$M_{yz} \leftarrow BuildMappingLine(C_{box}(Y,Z), O(Y,Z))$
$M_{zx} \leftarrow BuildMappingLine(C_{box}(Z,X), O(Z,X))$
$S_{ray} \leftarrow GenerateSuperRays(M_{xy}, M_{yz}, M_{zx}, P)$
return S_{ray}

without Y coordinate as can be seen in Eq. 3, and has the same formulation to finding segments of mapping line in $X-Z$ space. As a result, we can classify rays with the same traversal patterns by finding rays projected into all the same segments of three mapping lines in sub-spaces. Using this fact, we can implement our method to generate super rays in 3D using our 2D approach we introduce in Sec. IV-A. The pseudocode of generating super rays for a cell in the 3D case is shown in Alg. 2.

D. Updating Occupancy Map using Super Rays

To update occupancy maps with computed super rays, we use existing update methods with a minor modification. To determine cells needed for the update, we use the 3DDDA based algorithm [22]. Because all the points in a super ray update the same set of cells of a map, we traverse and update those cells in only a single traversal. Since a super ray is generated for multiple points, we take account of the weight of the super ray w (the number of contained points), and use the following, modified inverse sensor model:

$$L(x\,|\,z_t) = \begin{cases} w\, l_{occ}, & \text{if the end point of a super ray is in the cell} \\ w\, l_{free}, & \text{if a super ray passes through the cell.} \end{cases}$$

It is then guaranteed that we achieve the same occupancy map to that computed by processing points individually with multiple traversals.

Batching based updates. For a high performance of updating tree-based occupancy maps, we use a batching technique implemented in OctoMap [2]. The batching method reduces the number of repeated accesses to cells from a leaf to the root for updating the occupancy probability of the leaf cell.

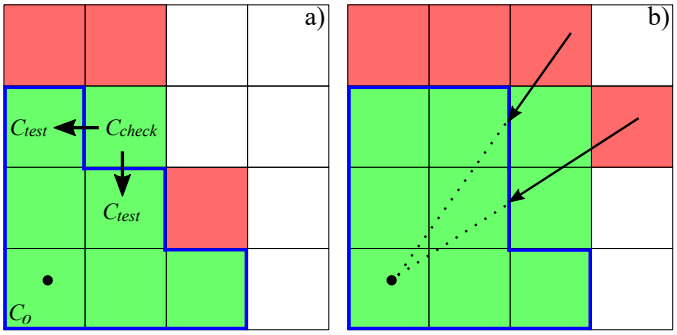


Fig. 9. These figures show the examples of building and using our culling region. The blue outline represents a culling region and the fully free cells are shown as green cells. In a), we show a test to check whether a cell C_{check} can be inserted into the culling region or not, during the process to build the region. We insert the cell C_{check} into the culling region, because both two neighbor cells C_{test} are in the culling region and C_{check} itself is in the fully free state. In b), we show the updated map with new measurements of points, while the generated culling region allows to skip the remaining traversals of the rays represented by the dotted lines.

The method batches cells that rays traverse, and then updates tree-based maps in a single time using the batched cells. This technique shows good performance in tree-based maps, but the time for batching the cells depends on the overhead of finding such cells. Fortunately, super rays can reduce the overhead of batching process thanks to a single traversal of super ray, instead of multiple traversals of points (Sec. V-C).

E. Culling Region based Updates

In this section, we propose another approach, culling region based update method, which utilizes occupancy states of maps updated by previous scans. The method increases the performance of super rays when we use an occupancy map with a high resolution. The number of generated super rays can be similar to the number of point clouds. Therefore, we cannot maximize the benefits of using super rays at such high-resolution maps. To overcome the issue, we propose its complementing method, a culling region based update method, for achieving a robust performance even with high resolutions.

As we mentioned in Sec. III-A, the clamping policy prevents occupancy maps from having the over-confidence problem. This thresholding technique makes the maps to support dynamically changing environments. Given this clamping policy, we observe that some traversals of a ray do not affect occupancy probabilities of cells, when those probability updates are limited by the thresholds (Fig. 3). Before a new update, cells of the map can have occupancy states updated during previous time steps. For updating the map with a new point cloud, the rays generated from the new sensor data traverse and then update occupancy probabilities of the map. However, some of the traversals do not change the occupancy states of the map. We aim to reduce such unnecessary traversals of a ray for achieving higher update performance.

Based on the observation, we define a culling region as a set of cells, where traversals from a cell in the region to the sensor origin are unnecessary. By utilizing the property of the culling region, we propose an efficient method to build and use the culling region for map updates without redundant computation. Our culling region based method consists of two steps per scan: 1) building the culling region by using the occupancy states of the map as described in Alg. 3 (Fig. 9-a)), and 2) updating the map by utilizing the generated region that reduces the unnecessary traversals (Fig. 9-b)). Note that for supporting a dynamic environment, our approach efficiently constructs and re-initializes a new culling region per scan.

Algorithm 3: BUILD CULLING REGION

Input: C_O, the cell containing a sensor origin
Output: CR, a culling region

```
1  CR = ∅, queue = ∅
2  if C_O is fully free then
3      CR.insert(C_O)
4      queue.push(neighbor cells of C_O)
5  while queue is not empty do
6      C_check ← queue.pop()
7      C_test ← Compute Test Cells(C_O, C_check)
8      if C_check is fully free ∧ all the C_test are in CR then
9          CR.insert(C_check)
10         queue.push(neighbor cells except C_test)
11 return CR
```

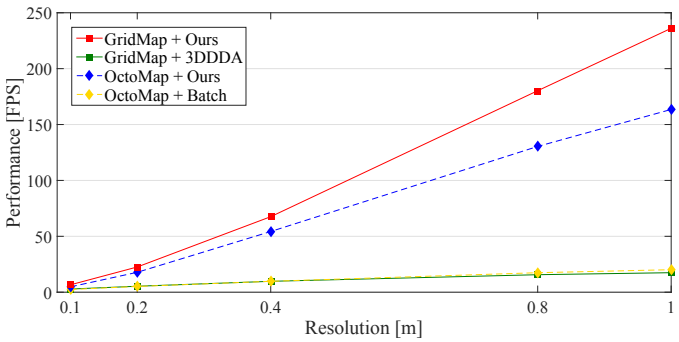
Building the culling region. At the given map, we first initialize and construct the culling region by checking whether a cell satisfies the properties of culling region or not. A naive approach would be to consider a frustum generated from a cell to the sensor origin and to check whether all the cells of the frustum are in the fully free states. Nonetheless, this naive approach may require a high computational overhead, which can even lower down the overall performance at the worst case.

We therefore propose a method of identifying the culling region that incrementally utilizes our simple tests, instead of the naive and time-consuming method. Our method makes a culling region by incrementally extending the region from the given C_O cell containing the sensor origin at new measurements. If the origin cell has a fully free state, we insert the cell into culling region because the origin cell guarantees the properties of the region. After inserting the origin cell into the culling region, we then prepare to extend the region using our simple tests on neighbor cells of the culling region (line 2:4 in Alg. 3).

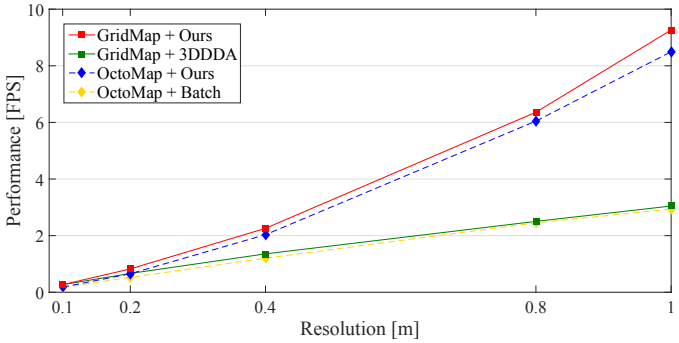
In our approach, we define a test cell, C_{test}, to be one of the neighbor cells of the current cell, C_{check}, which is located right outside of the current culling region. As a result, the test cell has the shorter Manhattan distance to the origin cell than the distance between the current cell and the origin cell. These cells are shown in Fig. 9 and thus we have: $dist(C_{test}, C_O) \equiv dist(C_{check}, C_O) - 1$ and $dist(C_{check}, C_{test}) \equiv 1$. In other words, the neighbor cells C_{test} for the test are candidate cells that a ray passing through the current cell C_{check} can traverse in the next step toward the origin cell (Fig. 9-a)). According to the definition of test cells C_{test}, we can have up to two and three test cells in the 2D and 3D cases, respectively (line 7 in Alg. 3).

When C_{check} does not have the fully free state, we should update the cell C_{check}. When C_{check} is in the fully free state, we check whether it can be included in the current culling region or not. For doing that, we can simply check whether its test cells C_{test} are in the culling region or not. When C_{test} are in the culling region, it is guaranteed that all the traversal from C_{test} are unnecessary given the definition of the culling region. If the current cell C_{check} passes such simple tests (line 8 in Alg. 3), traversals of a ray from the current cell to the origin cell are guaranteed to be unnecessary under the clamping policy. We therefore insert the C_{check} into the culling region and then continue the simple tests on neighbor cells that have not been tested (line 9:10 in Alg. 3). Finally we make the culling region efficiently through such simple tests in the 2D as well as 3D cases. Note that the culling region constructed by our method has a convex shape, surrounded by cells that do not have the fully free states.

Updating the map using the culling region. Our approach uses the generated culling region to remove the unnecessary traversals of rays. We make a ray traverse from its end point to the sensor origin, which is different from the common direction of traversal. If a ray encounters a cell of the culling region, our method can stop and terminate the remaining traversals from those cells to the sensor origin, as shown in Fig. 9-b). We therefore skip the updates for remaining traversals which do not affect the occupancy probabilities of map, and only update the traversed cells outside of the culling region. As a result, our approach achieves the high update performance with preserving the occupancy representation of the map.



(a) Indoor Dataset



(b) Outdoor Dataset

Fig. 10. These figures show the average performance, Frame Per Second (FPS), in two scenes according to various resolutions. Note that *Ours* represents the combination of our two methods using both super rays and the culling region. The solid and dashed lines represent the performance of method on GridMap and OctoMap, respectively.

V. Results and Discussions

In this section, we provide various results and discussions of our methods and other prior methods using a machine that has i7-8700K CPU with 12-cores and 32GB memory.

We mainly test our update methods and others on grid-based maps against two datasets, indoor and outdoor datasets, used in OctoMap [2]. The indoor dataset consists of 66 scans captured in a corridor (Fig. 1-(a)), and the outdoor dataset consists of 81 scans captured in a campus (Fig. 1-(b)). Scans of the indoor and outdoor datasets have point clouds consisting of 89,446 points and 247,817 points on average, respectively. We first give the update performances of various methods on grid-based maps in the indoor and outdoor scenes (Sec. V-A), and then discuss the issues related to our methods in the following sections (Sec. V-C and Sec. V-D).

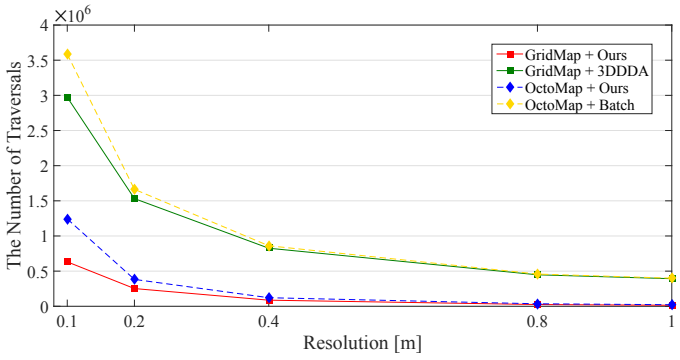
Furthermore, we test the grid-based maps as well as learning-based maps by using a new public dataset, KITTI [5], to show the performance of updates as well as the accuracy of mapping (Sec. V-B). The KITTI dataset[1] that we use consists of 395 scans captured by 3D laser scanner and has 119,801 points in a scan on average (Fig. 1-(c)).

We first introduce implementation details of optimizing our methods, followed by comparisons over prior methods. Note that in our earlier version of this work, we had tested super rays with the datasets using a single core. In this paper, we implement the test methods by applying parallel computation with 12-threads to updating maps as well as generating super rays.

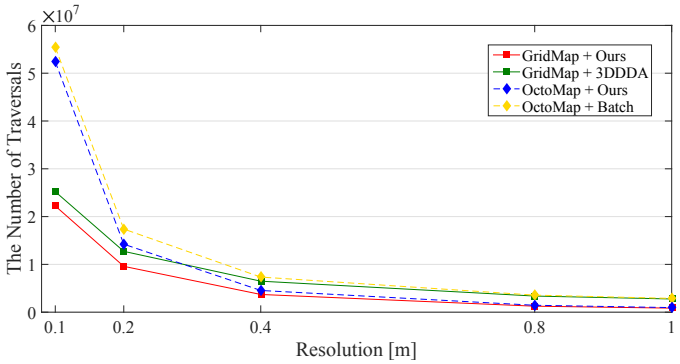
Implementation detail of super rays. Our super ray based method has a preprocessing cost induced by generating super rays, while it is designed for the efficient process. At the worst case, each super ray can have only a single point, demonstrating only the overhead of our method without any benefits. To prevent such a case, we use a threshold value, k, as the minimum number of points in a cell for generating super rays. We set the value to 20 for all experiments, and found that the threshold is enough to handle the problem. The detailed discussion about super rays is shown in Sec. V-C.

Implementation detail of the culling region. Our culling region based method finds fully-free cells on the updated map, and checks whether those cells can be inserted into the culling region or not. At the worst case in this process, we can insert some cells that do not trigger culling, and thus spend unnecessary time on computing those cells. This redundant computation occurs outside cells of the sensor's field-of-view. Therefore we limit a range of the culling region using the input sensor origin and point clouds.

[1]This dataset has the name, 2011_09_26_drive_0039, in the residential category

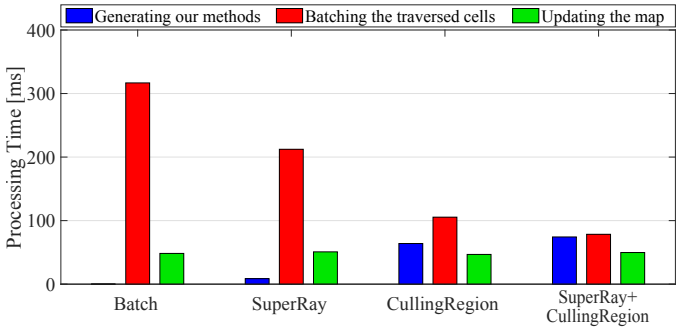


(a) Indoor Dataset

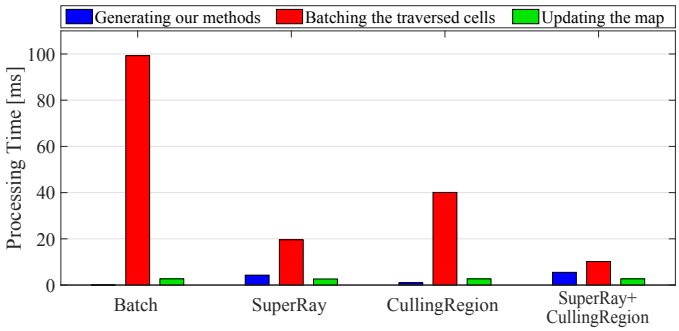


(b) Outdoor Dataset

Fig. 11. These figures show the number of traversals on average of available scans in two datasets according to various resolutions. The reported results are related to the performance graph, Fig. 10.



(a) Indoor dataset with 0.1 m resolution



(b) Indoor dataset with 0.4 m resolution

Fig. 12. These figures show the processing time that four different methods spend for each process on OctoMap with 0.1 m and 0.4 m resolutions. Overall, our methods give the high performance improvement compared to the batching based method, despite the computation time to generate super rays or culling region. The blue label

A. *Performance comparison for update methods*

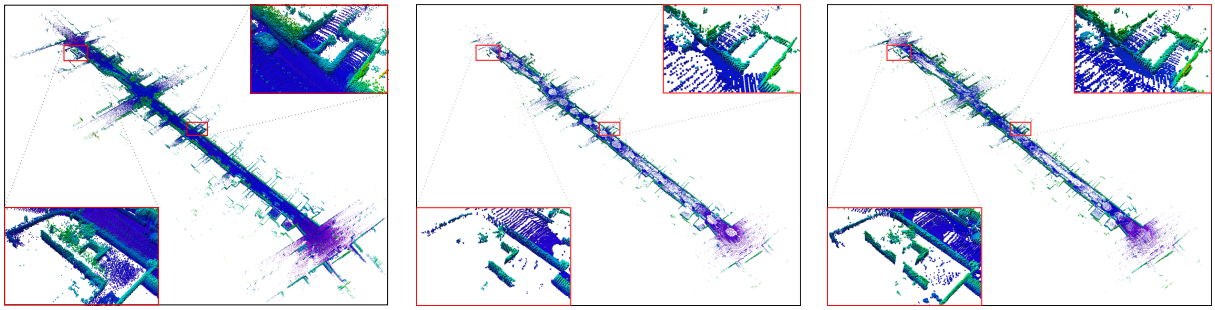
In the following experiments, we compare our method against the 3DDDA and batch based update methods on GridMap, the uniform grid map in 3D, and OctoMap, the octree based occupancy map, respectively. The overall performance of our methods includes all the processing time for updates such as the generation time of both super rays and the culling region; we analyze the performance of each component, super rays and culling region, of our method in the following sections. For all the experiments, we use the same settings of resolutions and parameters used in the prior work [2]; $l_{occ} = 0.85$ and $l_{free} = -0.4$, which are the log-odds values of occupancy probabilities to update cells of maps, and $l_{min} = -2$ and $l_{max} = 3.5$, the log-odds values of the clamping policy.

We measure all the computation time of generating super rays, building the culling region, and updating the maps for both indoor and outdoor datasets with various resolutions, and report the average frame (scan) per second (FPS) computed with all the available scans in Fig. 10. As shown in the graph, our method shows the highest performance in most of the tested cases. In the indoor scene, we achieve 7.7 times and 5.3 times faster performance compared to the 3DDDA based method on GridMap and the batch based method on OctoMap, respectively. In the case using outdoor dataset, our method shows the 1.9 times performance improvement for updates on average across resolutions, compared to the prior works on both GridMap and OctoMap.

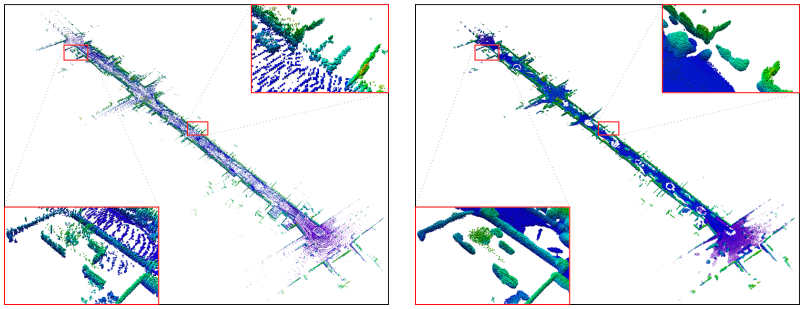
To analyze reasons of achieving such overall performance improvements, we also measure the number of traversed cells for updates. In the case of the tree structure, OctoMap, we count the number of cells updated from the leaf to all the parent nodes. As shown in Fig. 11-(a), our proposed method reduces the number of traversals by a factor of 13.0 times and 9.0 times on average in the indoor scene, compared to the 3DDDA based method in GridMap and the batch based method in OctoMap respectively. In the outdoor scene, our method removes about half of traversals for updates compared to the prior works, as reported by 2.0 times and 1.9 times reductions on average in GridMap and OctoMap (Fig. 11-(b)). As a result, it enables a significant decrease in the update time of our method. The detailed results are reported in Table IV.

Note that our methods provide the same maps to those computed by the 3DDDA or batch based updates, since our methods do not sacrifice any accuracy of the grid-based maps. Additionally, we also measure numerically how well our methods update occupancy probabilities compared to the prior methods. For this purpose, we measure mean squared errors of occupancy probabilities between our occupancy map and the map updated by the prior methods. We verify that the numerical errors turn out to be zero across all the tested settings.

For the specific analysis of our methods, we report the time breakdown of processes for each method on OctoMap with 0.1 m and 0.4 m resolutions in the indoor scene. Fig. 12-a) shows that our method using the culling region shows the



(a) Occupied test points (ground-truth) (b) OctoMap (83.6%, 2.19 sec/scan) [2] (c) Ours (87.1%, 1.10 sec/scan)



(d) BGKOctoMap (81.2%, 2.63 sec/scan) [10] (e) LARD-HM (86.0%, 3.59 sec/scan) [11]

Fig. 13. These figures visualize the points that each map classifies the test points to be occupied in our navigation scenario. We do not visualize the free points in this figure to avoid cluttered visualization, but consider them to compute the representation accuracy. The color represents the relative height of points, and the number in parenthesis is the representation accuracy and the update speed of a map. Our method shows the fastest update performance resulting in the highest representation accuracy.

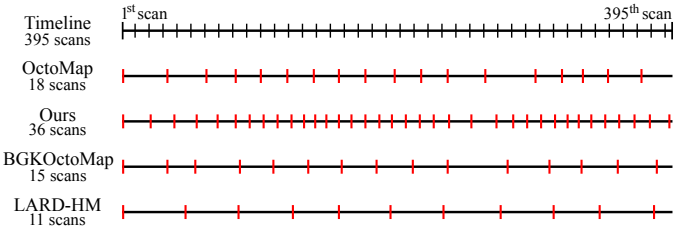


Fig. 14. This figure shows timestamps, represented by red bars, when each map uses point clouds in a scan for updates. A faster method can process more scans. The timeline of 395 scans captured by a 3D laser scanner at 10 Hz represents each of 10 scans as a black bar.

better performance than using super rays in the map with the high resolution (0.4 m). As shown in the blue bars of the figure, the culling approach spends 63.9 ms to generate the culling region, while the method using super rays reports less time 8.7 ms to generate the super rays. However, the culling region achieves a much larger benefit, 211 ms decrease on the batching process compared to the prior work, than the 104 ms decrease of using super rays.

Unlike the case of the high resolution, the super ray based method shows the better performance than the culling region based method for the low resolution case (Fig. 12-(b)). In this case, the culling region achieves 59 ms time reduction on the batching process despite the 0.9 ms time consumption to generate the culling region. Using super rays, on the other hand, reduces 80 ms on the batching process with 4.3 ms time spent on generating super rays.

As shown in Fig. 12, our combined method using both super rays and culling region shows the best performance compared to the prior work and our methods using only one of them. Our two update methods using super rays and culling region improve the update performance utilizing different types of information. Super ray based updates consider geometric relations among point clouds in the current scan. In other words, this method generates a super ray by grouping points associated with rays traversing the same set of cells. On the other hands, the culling region based method utilizes occupancy states of the updated map. This culling method skips rays whose updates on related cells are conservatively identified to be unnecessary. As a result, these two different methods complement to each other and combining them together shows the best performance.

B. Comparison of mapping algorithms

Recently, learning based approaches have been proposed to learn a correlation of occupancies from sensor measurements and to predict occupancy states of unobserved regions. These methods handle the mapping problem using classification and regression. In this section, we compare the performance in an aspect of the update speed as well as the representation accuracy of the grid-based maps, OctoMap [2], ours, and the learning-based maps, BGKOctoMap [10] and LARD-HM [11]. For the test, ours represents the octree map using the combined method of super rays and culling region for updates. We use the public KITTI dataset shown in Fig. 1-(c) for all points and Fig. 13-(a) for test points.

Each update method of map trains its representation with 80% of point clouds in 395 scans, and uses the remaining 20% points for testing the accuracy of mapping. We prepare the test points with occupancy states using the remaining points for

TABLE II
THE NUMBER OF GENERATED SUPER RAYS WITH DIFFERENT RESOLUTIONS.

# of points	Indoor [89,446]		Outdoor [247,817]	
Evaluation	# of super rays	# of points / super ray	# of super rays	# of points / super ray
0.1m	43605	2.1	186504	1.3
0.2m	25064	3.6	150453	1.6
0.4m	10668	8.3	102076	2.4
0.8m	3072	29.1	52906	4.7
1.0m	2073	43.1	40833	6.1

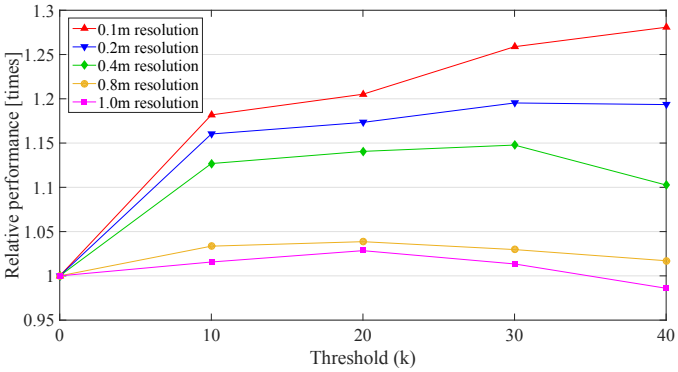


Fig. 15. This figure shows the relative performance to the case computed without using the threshold, i.e., $k = 0$; a higher value indicates faster performance. We report the performance of batch based updates using super rays on OctoMap with various resolutions. We pick $k = 20$ for all the other tests.

computing the representation accuracy; free points in the test set are selected randomly along rays traversing from the sensor origin to end points which have occupied states. For computing the representation accuracy, we consider points in the test set with occupancy probabilities less than 0.3 and larger than 0.7 values to have free and occupied states, respectively. In such test setting, we measure the rate of correct prediction of occupancy states, i.e., the rate of prediction showing the same occupancy states to the pre-computed states at all the test points, and report the measurement as the representation accuracy for each map. Note that OctoMap and ours use 0.2 m resolution to represent the environment, and BGKOctoMap and LARD-HM use all the same values of parameters reported in their corresponding papers.

The KITTI dataset we used in the test consists of raw point clouds captured by a 3D laser scanner at 10 Hz and the recorded configuration of a vehicle that had driven in a residential area. Using a playback tool *bag* on ROS (Robot Operating System), we can simulate a navigation scenario that a vehicle navigates the region and builds a map using captured sensor data in real-time. On the simulation, we let each map to discard point clouds of scans acquired during processing another scan, s, and process an available scan right after finishing the update process of the scan s. The playback of this scenario runs about 40 seconds because the KITTI dataset consists of 395 scans captured at 10 Hz.

In such a navigation problem, the ability of more frequently updating an accurate map can be considered as a better reaction ability to avoid suddenly appeared obstacles. For demonstrating such importance of real-time performance, we measure the time stamps of scans used for updating a map during the simulation (Fig. 14), and report the representation accuracy of the updated map using the test points (Fig. 13). As shown in Fig. 14, ours handles raw sensor data most frequently and uses the most number of scans for updating the map compared to the other mapping algorithms. As a result, the combined method using super rays and culling region deals with the number of scans by a factor of two over the OctoMap.

Such a high update speed results in a high mapping accuracy, since our map uses occupancy information of many sensor measurements. Fig. 13 shows the visualization of test points that each map classifies to have occupied states. As shown in the figure, our map has the highest representation accuracy, 87.1%, over the other occupancy maps, mainly thanks to the best performance on the update speed. Compared to OctoMap, ours represents the environment in more details as reporting a high accuracy of mapping. BGKOctoMap makes the sharp representation despite relatively a small number of scans. On the other hand, LARD-HM shows the dense representation for occupied points. However, the map has relatively low accuracy of mapping for free states. In this test, our combined method using super rays and culling region shows the closest update speed to the scanning speed of sensor compared to other methods. As a result, we achieve the high representation accuracy, while utilizing a high number of point clouds in many scans.

C. Analysis of super ray based updates

We analyze the performance of our super ray based updates in terms of two factors; the grouping ratio depending on various resolutions and the user-defined threshold limiting to generate super rays.

Super rays improve the update performance by reducing the number of rays used for updating the maps. To analyze such performance improvement, we measure the number of super rays with its grouping ratio in Table II for the test settings. As can be seen in this table, our method shows varying grouping ratios depending on resolutions. Intuitively a higher grouping ratio of our super rays leads the update methods to reduce the number of traversals more, which results in the high performance improvement.

Overall, our super rays give the high performance improvement to the update methods despite the computation time for generating super rays. They, however, show slightly lower performance gain in the maps with a very high resolution, e.g., 0.1 m, due to a low grouping ratio and its overhead for generating super rays in the high resolution. To lower the overhead, we use only a simple heuristic identifying based on the factor, the number of points in a cell. We use a simple threshold value, k, as the minimum number of points in a cell for generating super rays. In other words, for a cell with points less than k, we simply consider each point in the cell as a super ray with weight 1. For the rest of other cells, we apply our method using super rays (Sec. IV-C). In practice,

TABLE III
THE NUMBER OF UNNECESSARY TRAVERSALS OCCURRED BY BATCHING AND CULLING REGION BASED METHODS.

# of unnecessary traversals	Indoor		Outdoor	
Evaluation	Batch	Culling region	Batch	Culling region
0.1m	2649K	473K	11.0M	9.8M
0.2m	1277K	230K	7.2M	6.1M
0.4m	608K	109K	3.8M	2.9M
0.8m	240K	39K	1.7M	1.2M
1.0m	215K	27K	1.3M	0.8M

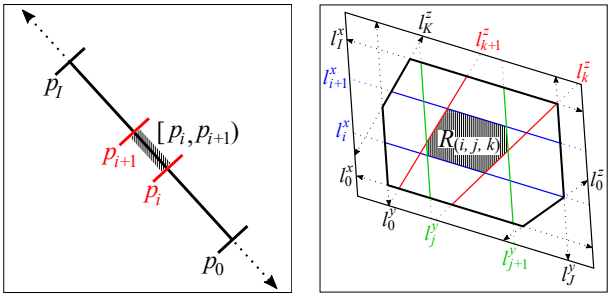
we found that 10 to 30 for k work reasonably well, and pick 20, since this setting shows robust performance gain across different tested resolutions over the case of $k = 0$ (Fig 15).

D. Analysis of culling region based updates

Our culling region improves the update performance by reducing the number of unnecessary traversals that do not affect the occupancy probabilities of maps. To analysis such performance improvement, we measure the number of unnecessary traversals processed in the prior work and our culling region based method, as shown in Table III. The culling approach reduces a huge amount of such traversals, 83.4% on average, in the indoor scene. As a result, this method enables the performance improvement for updating a map without sacrificing the occupancy information of sensor data. For example in the test shown in Fig. 12-(a), the culling region consists of 57,842 cells that removes the 82.1% unnecessary traversals accounting for 60.7% of entire traversals of the batching based method. As a result, our culling region based method improves 1.7 times performance over the prior work. For the outdoor scene, our method shows less decrease in the number of unnecessary traversals, 22.9% on average, and achieves the 1.2 times performance improvement on average.

VI. CONCLUSION

We have proposed two novel update methods for grid-based occupancy maps based on super rays and culling region. As our main algorithm, we construct a super ray that updates the same set of cells on occupancy maps in a single traversal. Specifically, we have proposed to use a mapping line for efficiently generating super rays in 2D case, and extend it to handle the 3D case. We also have proposed a culling region for reducing the number of unnecessary map updates. We have tested our methods using public datasets to reporting the update speed on grid-based maps, and achieve consistent overall performance across all the tested configurations. This robust performance improvement of our methods is thanks to the drastically reduced number of traversals using super rays as well as culling region. Furthermore, we have compared the proposed methods to recent learning-based maps in the simulation of navigation scenario. Ours, the octree map using our combined update method, shows the best performance for updating the representation and results in the highest representation accuracy compared to other mapping algorithms.



(a) Segment of mapping line (b) Region of mapping plane

Fig. 16. This visualizations represent the notations that we use for proofs of Theorem 1 and Theorem 2. The hatched area in the figure (a) shows a segment of mapping line, $[p_i, p_{i+1})$, closed by two projected points(red). In the figure (b), the hatched area represents a region of mapping plane $R_{(i,j,k)}$, closed by the projected lines from edges of grid points.

Limitations and future work. While our method showed meaningful improvements across different setting, it has certain drawbacks. First, our method showed rather low performance improvement in maps with a high resolution compared to low resolutions. We adopted the re-initialization of a culling region per scan, which is a simple, yet efficient approach for handling a dynamic environment. However, such an approach can have a considerable cost of building the region in high resolution maps. We therefore would like to design a method to efficiently update the culling region on-the-fly for achieving the high performance on the map with high resolution as well as supporting the dynamic environment. Furthermore, we would like to design an optimized update framework that can work well in modern architecture such as multiprocessors.

VII. APPENDIX

Observation 1. *In the 2-dimensional space, if rays have different traversal patterns, these rays are mapped into two different segments, which are divided by a point projected from a grid point of a cell.*

Theorem 1. *All the rays of a super ray generated by mapping line in the 2D space have the same traversal pattern.*

Proof. Our algorithm makes a mapping line by projecting all the grid points within a seed frustum into an arbitrary line. Rays associated with the seed frustum are mapped to a finite region in the line: $[p_0, p_I)$, where p_0 and p_I are the boundary values of the projected seed frustum into the mapping line. Let $P_{proj} = \{p_1, p_2, \ldots, p_{I-1}\}$ be a set of all the projected grid points within the seed frustum; they are within the finite region and assumed to be sorted in the order of distance to p_0 such as $DISTANCE(p_0, p_i) \leq DISTANCE(p_0, p_{i+1})$ for $0 \leq i < I$. The final mapping line computed by our approach consists of a set of segments:

$$ML = \{\, [p_i, p_{i+1}) \,|\, 0 \leq i < I \}$$

(Refer Fig. 16-(a)).

Let us assume that rays mapped into the same segment of the mapping line have the different traversal patterns. This indicates that a new grid point should be projected into the segment according to Observation 1. This contradicts that no

TABLE IV
OVERALL TIME (FPS) INCLUDING TIME SPENT ON GENERATING SUPER RAYS AND CULLING REGION (PROC.) AND TIME SPENT ON UPDATING MAPS (UPDATE). THE NUMBER WITHIN THE PARENTHESIS INDICATES THE NUMBER OF TRAVERSED CELLS FOR UPDATES.

Indoor Dataset															
Resolution	0.1m			0.2m			0.4m			0.8m			1.0m		
Evaluation	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]
OctoMap + Batch	2.7	0	365.1 (3586K)	5.4	0	184.1 (1663K)	9.8	0	102.0 (861K)	15.8	0	63.2 (458K)	17.7	0	56.4 (400K)
OctoMap + Ours	**4.9**	74.3	128.2 (1241K)	**18.0**	14.7	40.9 (382K)	**54.5**	5.5	12.9 (122K)	**138.0**	3.4	3.9 (35K)	**164.3**	3.7	2.4 (22K)
GridMap + 3DDDA	2.9	0	343.2 (2972K)	5.8	0	172.6 (1531K)	10.8	0	92.7 (826K)	18.3	0	54.7 (448K)	21.0	0	47.7 (392K)
GridMap + Ours	**6.9**	63.4	82.2 (632K)	**23.0**	10.8	32.7 (252K)	**69.0**	3.3	11.2 (87K)	**184.8**	2.1	3.4 (25K)	**246.9**	2.0	2.1 (15K)

Outdoor Dataset															
Resolution	0.1m			0.2m			0.4m			0.8m			1.0m		
Evaluation	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]
OctoMap + Batch	0.18	0	5427.7 (55.4M)	0.59	0	1681.8 (17.4M)	1.39	0	719.4 (7.3M)	2.62	0	381.1 (3.6M)	3.18	0	314.6 (2.9M)
OctoMap + Ours	**0.19**	89.1	5289.4 (52.3M)	**0.67**	42.6	1441.3 (14.2M)	**2.06**	21.2	463.6 (4.5M)	**6.11**	14.4	149.3 (1.4M)	**8.61**	13.3	102.8 (1.0M)
GridMap + 3DDDA	0.26	0	3817.4 (25.2M)	0.69	0	1441.3 (12.7M)	1.43	0	698.7 (6.5M)	2.77	0	361.4 (3.4M)	3.35	0	298.8 (2.8M)
GridMap + Ours	**0.27**	37.0	3672.2 (22.2M)	**0.84**	21.5	1175.9 (9.6M)	**2.29**	12.2	423.9 (3.7M)	**6.44**	8.2	147.1 (1.3M)	**9.38**	7.6	99.0 (0.8M)

projected grid point exists within each segment $[p_i, p_{i+1})$ for $0 \le i < I$. Therefore, rays mapped with the same segment of the mapping line have the same traversal pattern. This proves that all the rays in each super ray generated by our algorithm have the same traversal pattern. □

Observation 2. *In the 3-dimensional space, if rays have different traversal patterns, these rays are mapped into three different regions, which are divided by lines projected from edges of a cell.*

Theorem 2. *All the rays of a super ray generated by mapping plane in the 3D space have the same traversal pattern*

Proof. Our algorithm makes a mapping plane by projecting all the edges of grid points within a seed frustum into an arbitrary plane.

Let l_i^x, l_j^y and l_k^z be the lines projected onto the plane from edges aligned to each axis X, Y, and Z respectively. The projected seed frustum on the plane forms a finite region closed by the boundary lines: l_0^x and l_I^x, l_0^y and l_J^y, l_0^z and l_K^z. Let $L_{proj}^x = \{l_1^x, l_2^x, \ldots, l_{I-1}^x\}$, $L_{proj}^y = \{l_1^y, l_2^y, \ldots, l_{J-1}^y\}$, and $L_{proj}^z = \{l_1^z, l_2^z, \ldots, l_{K-1}^z\}$ be sets of projected lines from all the edges of grid points within the seed frustum. The elements of L_{proj}^x and L_{proj}^y are assumed to be sorted in the order of distance to the boundary lines, i.e., $DISTANCE(l_0^x, l_i^x) \le DISTANCE(l_0^x, l_{i+1}^x)$ for $0 \le i < I$. In the case of L_{proj}^z, its elements are assumed to be sorted in the order of angle to the boundary line, such as $ANGLE(l_0^z, l_k^z) \le ANGLE(l_0^z, l_{k+1}^z)$ for $0 \le k < K$. All the projected lines partition the plane and generate the final mapping plane, MP, consisting of a set of regions; each of which is expressed as $R_{(i,j,k)}$:

$$MP = \{ R_{(i,j,k)} \,|\, (0 \le i < I) \wedge (0 \le j < J) \wedge (0 \le k < K) \},$$

where a region closed by the lines is defined as follows:

$$R_{(i,j,k)} = \begin{cases} [l_i^x, l_{i+1}^x), \\ [l_j^y, l_{j+1}^y), \\ [l_k^z, l_{k+1}^z) \end{cases}$$

(Refer Fig. 16-(b)).

We now assume that rays mapped into the same region of the mapping plane have different traversal patterns, for proof by contradiction. This indicates that a line should be projected within the region according to Observation 2. No projected lines, however, exists within the region $R_{(i,j,k)}$ for $0 \le i < I$, $0 \le j < J$, and $0 \le k < K$, contradicting the assumption. Therefore, rays mapped onto the same region of the mapping plane have the same traversal pattern. □

REFERENCES

[1] H. Moravec, "Robot spatial perceptionby stereoscopic vision and 3d evidence grids," *Perception,(September)*, 1996.

[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[3] S. T. OCallaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012.

[4] F. Ramos and L. Ott, "Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[6] Y. Kwon, D. Kim, and S.-e. Yoon, "Super ray based updates for occupancy maps," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4267–4274.

[7] J. Pan, S. Chitta, and D. Manocha, "Probabilistic collision detection between noisy point clouds using robust classification," in *International Symposium on Robotics Research (ISRR)*, 2011.

[8] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3218–3223.

[9] A. Leeper, S. Chan, and K. Salisbur, "Point clouds can be represented as implicit surfaces for constraint-based haptic rendering," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5000–5005.

[10] K. Doherty, J. Wang, and B. Englot, "Bayesian generalized kernel inference for occupancy map prediction," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3118–3124.

[11] V. Guizilini and F. Ramos, "Large-scale 3d scene reconstruction with hilbert maps," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3247–3254.

[12] Y. Roth-Tabak and R. Jain, "Building an environment model using depth information," *Computer*, vol. 22, no. 6, pp. 85–90, 1989.

[13] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[14] J. Wilhelms and A. Van Gelder, "Octrees for faster isosurface generation," *ACM Transactions on Graphics (TOG)*, vol. 11, no. 3, pp. 201–227, 1992.

[15] P. Payeur, P. Hébert, D. Laurendeau, and C. M. Gosselin, "Probabilistic octree modeling of a 3d dynamic environment," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 2. IEEE, 1997, pp. 1289–1296.

[16] 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.

[17] S. Coenen, J. Lunenburg, M. van de Molengraft, and M. Steinbuch, "A representation method based on the probability of collision for safe robot navigation in domestic environments," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4177–4183.

[18] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012, pp. 692–697.

[19] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 778–785.

[20] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of octrees for efficient 3d mapping," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4249–4255.

[21] E. Einhorn, C. Schröter, and H.-M. Gross, "Finding the adequate resolution for grid mapping-cell sizes locally adapting on-the-fly," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1843–1848.

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

[23] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley, "Interactive ray tracing for volume visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 3, pp. 238–250, 1999.

[24] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker, "Ray tracing animated scenes using coherent grid traversal," in *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3. ACM, 2006, pp. 485–493.

[25] 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.

[26] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2. IEEE, 1985, pp. 116–121.

[27] M. Yguel, O. Aycard, and C. Laugier, "Update policy of dense maps: Efficient algorithms and sparse representation," in *Field and Service Robotics*. Springer, 2008, pp. 23–33.