# Super Ray based Updates for Occupancy Maps

Youngsun Kwon[1], Donghyuk Kim[2] and Sung-eui Yoon[3]

***Abstract*— We present a novel approach, Super Ray, for efficiently updating map representations such as grids and octrees with point clouds. In this paper, we define a super ray for points as a representative ray to them with an associated frustum. A super ray is constructed in a way that updating those points has the same set of cells accessed during the map update process. As a result, we can perform the update process with a super ray in a single traversal on the map, resulting in performance improvement without compromising any representation accuracy of the map. For constructing super rays efficiently, we propose mapping lines for handling 2-D and 3-D cases from an observation that edges or grid points branch out the access pattern of updating the map. Our method is general enough to be applied for variety of occupancy map structures based on axis-aligned space subdivisions such as grids and octrees. We test our method into indoor and outdoor benchmarks, and achieve 2.5 times on average (up to 3.5 times) performance improvement over the state-of-the-art update method for OctoMap and grid 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 (e.g., Kinect and Xtion). The point clouds are represented by a large amount of points representing geometric information of environments in high resolution, yet with various levels of sensor noise. In applications such as path planning or SLAM, it is difficult to use such point clouds directly because of the sheer amount of generated data itself as well as the noise.

To address these issues, various occupancy map representations such as grids [1] and octrees [2] have been proposed to represent point clouds, for reducing the memory requirement and considering uncertainty of point clouds. Recent applications use such map representations to achieve higher performance for their goals. For example, path planning algorithms use an occupancy map representing free or occupied states in each cell for efficiently finding a collision free path instead of accessing large and uncertain point clouds.

Unfortunately, constructing such occupancy maps out of point clouds can take a high computation overhead, especially when we use a high resolution for the map to achieve a high representation accuracy, it can take a huge amount of time for traversing and updating the map. On the other hand, when we use a lower resolution for the map, we can achieve a high performance, but comes with a low representation

[1]Youngsun Kwon is at Robotics Program, KAIST, Daejeon, South Korea. [2]Donghyuk Kim, and [3]Sung-eui Yoon are at School of Computing, KAIST, Daejeon, South Korea; youngsun.kwon@kaist.ac.kr, donghyuk.kim@kaist.ac.kr, sungeui@gmail.com

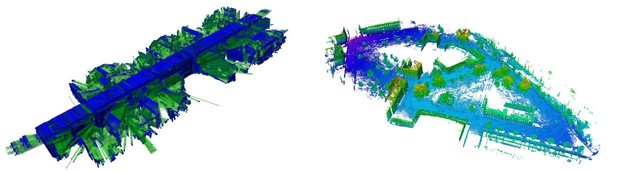

(a) Indoor dataset (b) Outdoor dataset

Fig. 1. These figures visualize map representations for two public datasets [2]. *a*) Blue and green cubes represent occupied and free spaces, respectively. *b*) We use heat colors to represent relative heights for visualizing the dataset.

accuracy, which may result in serious problems for various robotic operations such as motion planning.

**Main contributions** In this paper, we present a novel, efficient map update method based on super rays, while achieving high performance without compromising representation accuracy of map at all. Specifically, we propose to use super rays of points as our main update method for maps. A super ray is a representative ray for set of points, and is constructed in a way that updating the map with those points traverses the same set of cells and can be processed together. To construct such super rays given input points, we propose to use a mapping line for updating 2-D maps (Sec. IV-A), and generalize it to 3-D maps such as grids and octrees (Sec. IV-C).

To demonstrate benefits of our method, we test it with indoor and outdoor scenes (Fig. 1) for two different occupancy map representations: grids and OctoMap. We found that our method is robust enough to show improvement, up to 3.5 times improvement, across a diverse set of configurations over the prior, exact method based on 3DDDA (3-D Digital Differential Analyzer) (Sec. V). These results are achieved mainly thanks to identifying coherent updates on maps and processing them in a single traversal with a super ray.

## II. RELATED WORK

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

### A. Map Representations

Point clouds is one of the most common sensor data that are 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 cheap consumer-level depth sensors become available. Some of recent work use point clouds and apply them directly to applications (e.g., collision detection [3]). Nonetheless, point clouds can have excessive amount of points especially for large-scale scenes, and more severely it can have inherent sensor noise. Due to these issues, many prior approaches [4], [5] convert point clouds to other representations (e.g., triangular meshes) in order to process them in a simple and efficient way.

In robotics, one of popular representations is the grid map [1], [6] approximating point clouds. While this grid map is proposed early, it has certain limitations. Its main drawback is that it requires a tremendous size of memory, when we handle large-scale outdoor environments or require high resolutions for accurate representations.

Tree-based representations such as quad-tree maps in 2-D and octree maps in 3-D have been studied in order to overcome the problems. The octree map divides a 3-D 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 state, 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 [7], [8]. Payeur et al. [9] suggested to augment octrees with probabilistic occupancy states for considering sensor noise. Recently, Wurm et al. [2] adopted unknown states for representing regions occluded by obstacles. Coenen et al. [10] considered the unknown state as the region with a high probability having collision. Many applications such as navigation [11] and point cloud compression [12] have been developed based on this octree map representation.

In this paper, we assume that a robotic application uses grid or octree 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 for Point Clouds

When we have a point from a sensor, it means that we do not have any collisions from the sensor origin to the point. We need to reflect this information on grid or octree based occupancy map representations. 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 point clouds is to decide an adequate resolution of an octree based map representation, instead of updating the full resolution of the octree map. Along these lines, different methods have been proposed for using various resolutions depending on objects [13] or statistics of updated states of each cell [14]. 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 [15], [16]. Recent occupancy map representation, OctoMap [2], uses the 3DDDA based algorithm [15] as an exact method to update the map with point clouds. Wald et al. [17] proposed a method to traverse a grid with coherent rays. This work packetizes rays traversing similar space in the grid to reduce the number of intersection tests used for ray tracing. This work is neither designed for our occupancy maps, nor is applicable to our work. Nonetheless, we are inspired by this approach, and propose super rays to our problem.

Voxel filtering of PCL [18] 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.

## III. Overview

We give backgrounds on occupancy maps and give an overview of our method.

### A. Backgrounds on 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 some objects 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 a map representation accommodating this information acquired from sensors.

Unfortunately, data captured by sensors accommodates 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. The occupancy probability, $P(n|z_{1:t})$, represents the occupancy state of a cell, $n$, 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 [19] as follows:

$$P(n|z_{1:t}) = 1 - \left[1 + \frac{p(n|z_{1:t-1})}{1-p(n|z_{1:t-1})}\frac{p(n|z_{t-1})}{1-p(n|z_t)}\right]^{-1} \quad (1)$$

For the fast update to the map representation, recent approaches use the log-odds notation [20], [2], and the prior equations are transformed into:

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \quad (2)$$

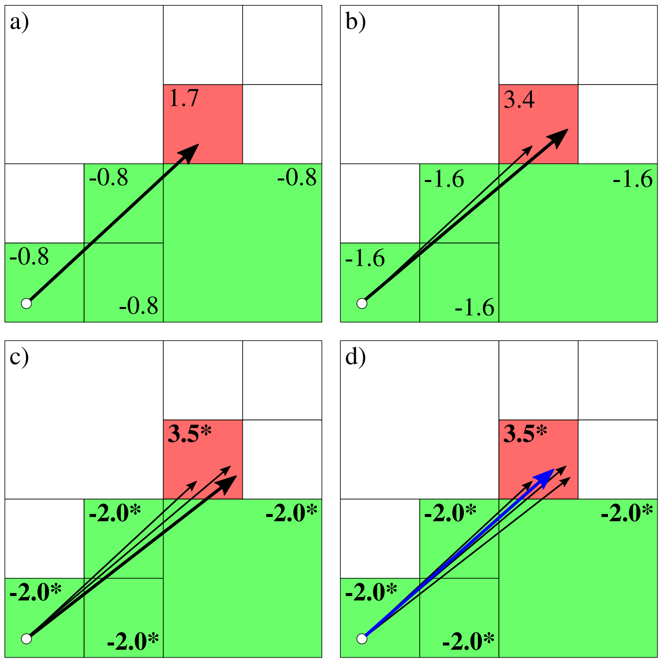


Fig. 2. Each figure represents occupancy probabilities of cells after updating each ray to the octree-based occupancy map representation. In this example, we traverse the same set of cells for four different rays. The bold numbers with * notation in cells indicate that those cells are classified into fully occupied or fully free state. The blue ray in figure *d*) is an example of causing redundant computation, since it does not change any occupancy probabilities of the traversed cells. In this figure, we use $l_{occ} = 1.7$, $l_{free} = -0.8$, $l_{max} = 3.5$, and $l_{min} = -2.0$.

where the inverse sensor model $L(n|z_t)$ is defined as the following:

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

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 occur frequently in dynamic environments. OctoMap [2] solves the problem by using a clamping policy that limits the occupancy probability of a cell based on minimum and maximum state bounds: $l_{min}$ and $l_{max}$. 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 shows an illustration of updating the octree map given point clouds.

### B. *Motivations*

Occupancy maps such as octrees and grids have been widely used for various applications. We, however, found that updating these maps can take a huge amount of computation time. Furthermore, we have identified that the original update method for occupancy maps has redundant computations, because of the discrete nature of grid and octree representations. For example, Fig. 2 shows four different rays traverse the same set of cells in the octree representation, while these rays have different end points. When we update the map with these rays one-by-one, redundant computations are made on traversal and updating through exactly same set of cells, resulting in lower performance.

Additionally, certain rays 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. These problems occur frequently because the original update method does not consider the discrete nature of grid and octree representations.

### C. *Overview of Our Approach*

To overcome these problems, we propose to use super rays and their update method for occupancy maps. We define a super ray of points as a representative ray for rays generated for those points. 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 points associated with the super ray, thus removing redundant computation and achieving higher performance.

Our algorithm consists of three phases. 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 cells traversed for updating the map based on the mapping line (Sec. IV-A). We also explain how to update cells that each super ray passes without compromising the representation accuracy of maps (Sec. IV-D). We explain our concepts based on the 2-D case first and then expand it to the 3-D case (Sec. IV-C).

For the sake of simplicity, we explain our method based on the uniform grid as an occupancy map representation for point clouds. Our method, however, is easily applied to octrees, and results with grids and octrees are reported in the result section.

## IV. Real-Time Updates using Super Rays

In this section, we explain our approach in detail.

### 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 (and its associated super ray) starting from the sensor origin to the cell box containing all the points in the cell $c$, the red box shown in Fig. 3-*a*). Starting from the seed frustum, we partition it into multiple ones, each of which accesses 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 2-D data. For this process, we pick an axis, i.e., $X$, $Y$, or $Z$ axis, for computing such slices,

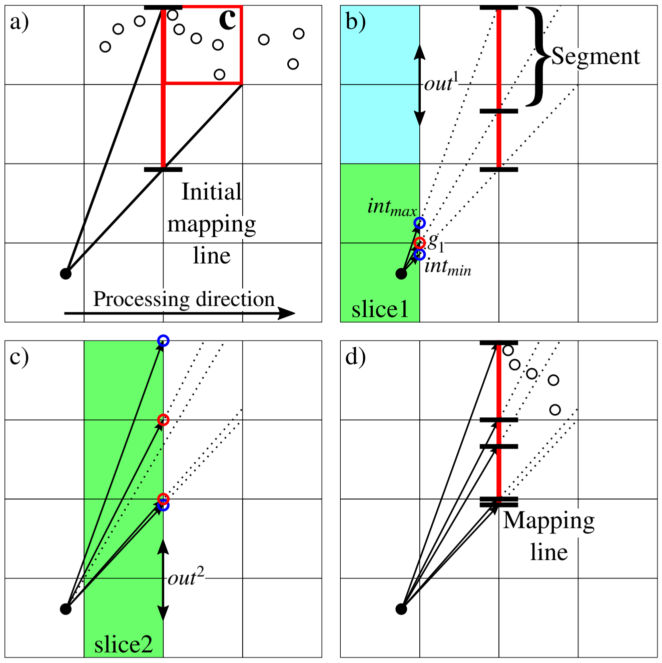


Fig. 3. This figure shows an example of updating a mapping line for a cell $c$. The red grid point $g_1$ in $b$) divides the seed frustum into two sub-frustums, and its projected point generates two segments on the mapping line. In c), two grid points in $\text{out}^2$ in the slice 2 also generates two more segments in the mapping line shown in d).

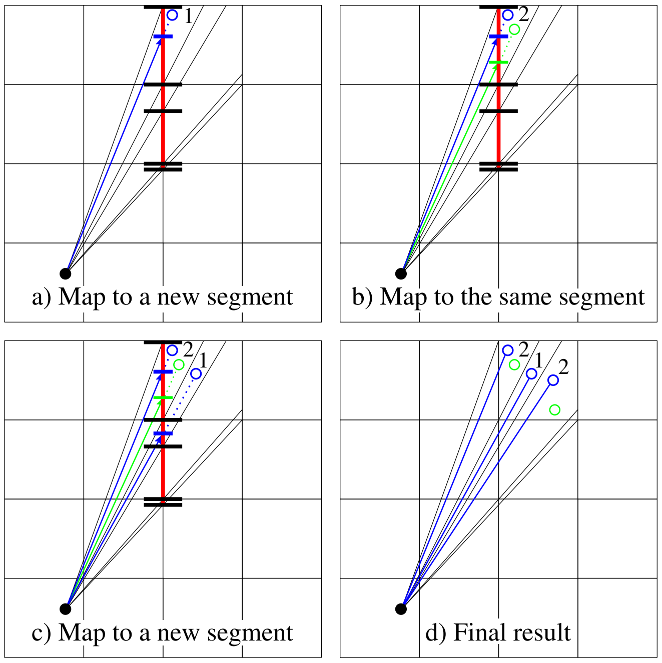


Fig. 4. This figure shows how we compute three different super rays out of five points using the computed mapping line. a) A new point maps to a new segment, and we treat it as a new super ray with a weight of one. b) Another point maps to the prior segment, and we increase its weight to two. c) The new point maps to a new segment and a new super ray is assigned to it. d) shows the final, three super rays with their weights.

and treat it as a processing direction. Fig. 3 shows $X$ as the processing direction and computed slices.

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. 3-$a$) shows an initial mapping line. Each segment of 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 broken into multiple segments corresponding to multiple super rays.

One key observation for updating 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. 3-$b$) 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. 3-$b$)). Based on this observation, the key operations are how we efficiently compute grid points within the frustum.

Let $out^i$ of $i$-th slice to denote the faraway line of the slice along the processing direction. Fig. 3-$b$) shows an example of $\text{out}^1$ for slice 1. We can then compute $int_{min}$ and $int_{max}$ that are two intersection points of the seed frustum for each $i$-th slice like the blue circles in Fig. 3.

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 computing a mapping line works in an iterative manner from slice 1 to slice $N-1$. To compute grid points that differentiate the access 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 $m+1$ sub-frustums, resulting $m+1$ corresponding segments on the mapping line. Each pair of two consecutive elements in the mapping line implicitly defines a segment and its associated frustum (and its super ray). Note that we can easily compute these grid points thanks to the discrete nature of occupancy maps, and compute segments of the mapping line without using expensive sorting methods, resulting in a fast method. The pseudo code of generating a mapping line for a cell is shown in Alg. 1.

### B. Generating Super Rays using the Mapping Line

After we computed the mapping line of each cell at the prior step, we use it for computing how many points are assigned to each computed frustum. To perform this process, we traverse all the input points, project each of them to the mapping line, and count how many points are assigned to each segment of the mapping line that maps to a frustum (Fig. 4).

The points assigned to the same segment in the mapping line have the same access patterns in terms of cells traversed to update the map. Therefore, we treat them to be in a super ray. Especially, we do not store all those points, but store the first point (or any one of them) and the number of assigned points as a weight to the super ray. We use this

**Algorithm 1:** BUILD MAPPING LINE

```
Input: $C_{box}$, a cell box in 2-D, $O$, a sensor origin in 2-D
Output: $M_{line}$, a mapping line
1 $M_{line} \leftarrow InitMappingLine(C_{box})$
2 $S_{slice} \leftarrow InitSlices(O, C_{box})$
3 for $i$ in $1 : length(S_{slice}) - 1$ do
4     $g \leftarrow ComputeGridPoints(S_{slice}[i], C_{box})$
5     for $j$ in $1 : length(g)$ do
6         // project onto mapping line without sorting
          $M_{line}.insert(Projection(g_j))$
7 return $M_{line}$
```

**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
1 $C_{box} \leftarrow ComputeCellBox(P)$
2 $M_{xy} \leftarrow BuildMappingLine(C_{box}(X,Y), O(X,Y))$
3 $M_{yz} \leftarrow BuildMappingLine(C_{box}(Y,Z), O(Y,Z))$
4 $M_{zx} \leftarrow BuildMappingLine(C_{box}(Z,X), O(Z,X))$
5 $S_{ray} \leftarrow GenerateSuperRays(M_{xy}, M_{yz}, M_{zx}, P)$
6 return $S_{ray}$
```

information associated with a super ray to efficiently update our occupancy map (Sec. IV-D).

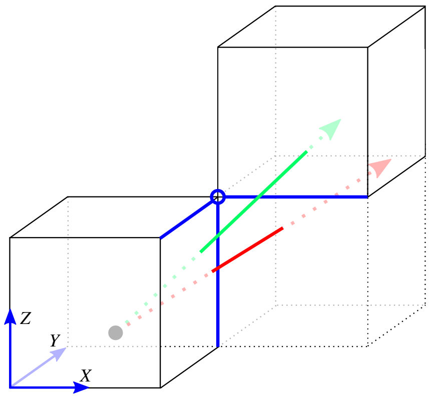
### C. Extension to the 3-D Case

In this section, we explain how we extend our prior 2-D approach into handling 3-D points. Essentially, we handle the 3-D cases by considering them in three different 2-D planes with their mapping lines.

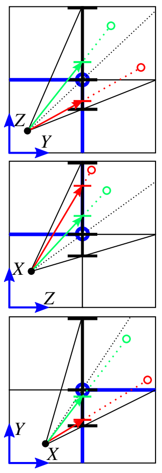
Similar to the 2-D case, we first compute a bounding volume containing point clouds in the map representation. We also construct a seed frustum traversing to the volume, and then partition the frustum into sub-frustums, each of which accesses the same set of cells.

The key observation for the 3-D case is that access patterns of cells differ along edges of cells. Fig. 5-*a*) shows that two rays access different cells since they are partitioned by the blue edge. In this case, the access pattern of two rays differs from the blue grid point projected into the Y-Z plane. Based on this observation, we project points into three different planes, Y-Z, Z-X, and X-Y planes, and check whether those points are partitioned by grid points in those 2-D planes. As a result, we can solve the 3-D problem using our 2-D approach mentioned in Sec. IV-B. We compute each mapping line for each plane.

To generate super rays using three mapping lines, we project input points into each mapping line. When points are assigned to the same segment in all of three mapping lines, those points have the same access patterns. Similar to the 2-D case, we generate a super ray for those points. The pseudo code of generating super rays for a cell in the 3-D case is shown in Alg. 2.



(a) An example of classifying two rays in 3-D



(b) Mapping lines

Fig. 5. This figure shows an example of generating super rays in the 3-D case. The rays access different cells, since they are partitioned by the blue edge. This information can be identified by three different 2-D projections, the Y-Z, Z-X, and X-Y projections shown on the right.

### D. Updating Occupancy Map with Super Rays

To update occupancy maps with super rays, we use existing update methods proposed by the OctoMap [2]. To determine cells needed for the update, we use the 3DDDA based algorithm [15]. Because all the points in a super ray access the same set of cells, we traverse and update those cells 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:

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

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

## V. RESULTS AND DISCUSSIONS

We test our method and others on a machine that has 3.4GHz Intel i7-4770 CPU. For all the experiments, we use two public datasets, indoor and outdoor datasets, used in OctoMap [2]. The indoor dataset consists of 66 scans captured in a corridor, and the outdoor dataset consists of 81 scans captured in a campus (Fig. 1). Scans of the indoor and outdoor datasets have point clouds consisting of 89,446 points and 247,817 points on average, respectively.

**Implementation detail.** Our method of generating super rays has preprocessing cost induced by generating super ray, while it is designed for efficient process. At the worst case, we attempt to generate super rays, but each super ray can have only a single point, demonstrating only the overhead of our method without any benefits. We can estimate such cases depending on two factors: the number of points in a cell and the geometric configuration (e.g., distance and angle) between the sensor origin and the cell. Fortunately, we found

TABLE I

THIS TABLE SHOWS 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 |
| 0.2m | 25064 | 3.6 | 150453 | 1.6 |
| 0.4m | 10668 | 8.3 | 102076 | 2.4 |
| 0.6m | 5106 | 17.5 | 72191 | 3.4 |
| 0.8m | 3072 | 29.1 | 52906 | 4.7 |
| 1.0m | 2073 | 43.1 | 40833 | 6.1 |

that simply checking the number of points in a cell works fine for our method.

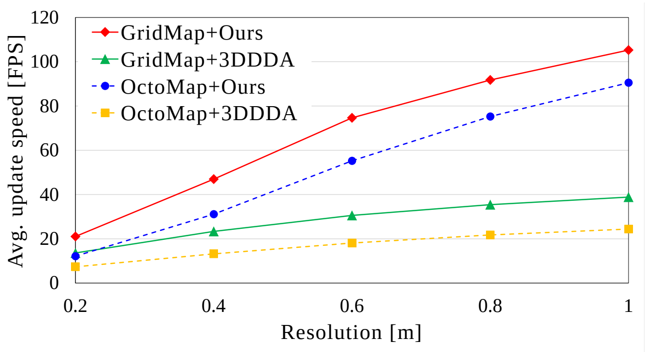
Specifically, we use a threshold value, $k$, on the minimum number of points in a cell for generating super rays. In other words, for a cell with points less than $k$, we process all the points individually by simply creating a super ray per each point in the cell. For the rest of other cells, we apply our method. We tested different $k$ values in a range between 0 and 40, and found that 20 shows the best performance in practice. As a result, we report all the results in this setting.

Table I shows the number of generated super rays as a function of the resolution. In the case of 0.6 m resolution, our method groups on average 17.5 points (up to 43.1 points) and 3.4 points (up to 6.1 points) per super ray in indoor and outdoor datasets, respectively. This high grouping ratio results in the dramatic decrease for the number of cells traversed during the map updates in indoor and outdoor datasets.

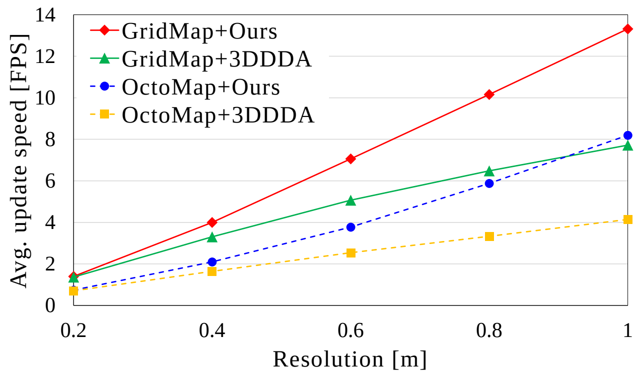
## A. Overall Performance

We compare overall performance of our super rays based method and the prior 3DDDA based method [15]. The overall performance of our method includes both the generation time of super rays and update time of the map with those super rays. In the following experiments, we use the 3DDDA based update method implemented in the OctoMap library [2]. We test these two different methods in OctoMap, the octree based occupancy map, and GridMap, the grid based occupancy map. For the update methods, we use the same $logOdds$ values, $l_{occ} = 0.85$ and $l_{free} = -0.4$, for any kinds of rays (i.e., super and regular rays) to update cells, and the same parameters, $l_{min} = -2$ and $l_{max} = 3.5$, of the clamping policy adopted from OctoMap.

We measure all the computation time of generating super rays and updating cells for both indoor and outdoor datasets with various resolutions, and report the average frame per second (FPS) computed with all the available scans (frames). Fig. 6 shows the average FPS of each tested method with two map representations, OctoMap and GridMap. As can be seen, our method shows higher performance than the 3DDDA in all the tested configurations. Since improvements observed with OctoMap are similar to those with GridMap, we simply mention the average improvement measured from OctoMap



(a) Indoor Dataset



(b) Outdoor Dataset

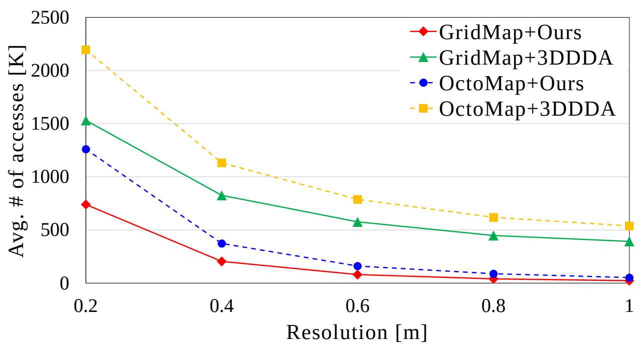
Fig. 6. These figures show average performances, Frame Per Second (FPS), in two datasets according to various map resolutions.

and GridMap in the text for simplicity. Detailed results with generation time and update time are reported in Table II.

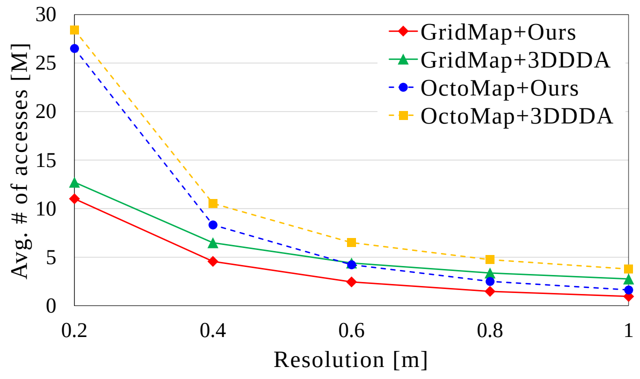
We achieve 2.5 times and 1.6 times faster performances on average compared to the prior 3DDDA method with indoor and outdoor datasets, respectively. In the indoor dataset, our method shows from 1.6 to 3.2 times higher performance over the prior method. In the outdoor dataset, our method shows better, but similar performance to the prior method in a small resolution (e.g., 0.2 m), but shows up to 1.9 times higher performance in the other tested resolutions.

To analyze reasons of achieving such overall performance improvements, we also measure the number of cells traversed and accessed during the update process (Fig. 7). As can be seen in the figure, our method reduces the number of cells traversed across all the tested settings up to 20 times. As a result, it results in significant decrease for the update time of our method.

Table II shows the separate times spent on generating super rays and updating with our method. Overall, our method decreases the update time by a factor of 5.6 times on average, up to 20 times. The time spent on generating super rays varies depending on the resolutions and datasets, while the overall performance of ours is better than the prior one. For example with the indoor dataset consisting of 89 K points, our method spends about 16.6 ms to generate 25.1 K super rays from the 0.2 m resolution. As a result, we generate 1.51 K super rays per ms, where each super ray has 3.6



(a) Indoor Dataset



(b) Outdoor Dataset

Fig. 7. These figures show graphs of average number of accesses to cells in two datasets according to various resolutions of map representation.

points on average. On the other hand, our method spends 8.6 ms to generate 2.1 K super rays from 1 m resolution. This translates that we generate 0.24 K super rays per ms, where each super ray has 43.1 points. This results in overall performance improvement thanks to a high decrease rate in the number of cells traversed.

## B. Discussions

In theory, our method is an exact update method, which provides the same results to those computed by the 3DDDA update method. We also demonstrate numerically how well our method update occupancy probabilities compared to the prior update method. For this purpose, we measure mean squared errors between our occupancy map and the map updated by the prior method. We verify that the numerical errors turn out to be zero across all the tested settings.

In a case that a cell of the map has a few points, updating the map with point clouds can be better than with super rays because of overhead for generating super rays. To overcome this problem, we use the simple threshold $k$ to be 20 for efficiently checking whether it is beneficial to generate super rays in a cell or not. Even without using this threshold, i.e. $k = 0$, we achieve 1.9 times faster performance on average compared to the prior 3DDDA method with two datasets. Our method without using the threshold spends more time on generating super rays, but less time on updating maps in general.

## VI. Conclusion

We have proposed a novel update method for occupancy maps based on super rays. We construct a super ray of points in a way that processing those contained points accesses the same set of cells in occupancy maps. Specifically, we have proposed to use a mapping line for efficiently generating super rays, and extend it to handle the 3-D case. We have applied our method into two different datasets and two different occupancy maps: octree and grid based maps. Our method is robust enough to show consistent overall performance improvement across all the tested configurations. This robust performance improvement is thanks to the fast super ray generation using mapping lines and the drastically reduced number of cells traversed during the map update process.

There are many interesting future research directions. We currently used a simple threshold $k$ not to generate super rays on unpromising cells that do not have many points. We would like to design an optimized technique on this aspect by considering the geometric configuration between the sensor origin and the cell under the update process. We expect that this additional study can result in additional performance improvements, while keeping the overhead of generating super rays low. Furthermore, we would like to extend our method to work well in modern streaming architectures such as GPU for achieving real-time update performance in a rate of 30 ms.

## Acknowledgment

We would like to thank anonymous reviewers for constructive comments and SGLab members, specially Jaepil Heo and Myungbae Son, for useful discussions. This work was supported in part by ICT R&D program of MSIP/IITP[R0126-15-1108] and DAPA/ADD(UD140022PD). Sung-eui Yoon is a corresponding author of the paper.

## References

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

[2] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram 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*, 2010, vol. 2.

[3] Jia Pan, Sachin Chitta, and Dinesh Manocha, "Probabilistic collision detection between noisy point clouds using robust classification", in *International Symposium on Robotics Research (ISRR)*, 2011.

[4] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael 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.

[5] Adam Leeper, Sonny Chan, and Kenneth 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.

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

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

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

TABLE II

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

| Indoor Dataset | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Resolution | 0.2m | | | 0.4m | | | 0.6m | | | 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 + 3DDDA | 7.3 | 0 | 137.6 (2195K) | 13.2 | 0 | 76.3 (1132K) | 18.1 | 0 | 55.6 (788K) | 21.7 | 0 | 46.2 (619K) | 24.4 | 0 | 41.1 (538K) |
| OctoMap + Ours | **12.1** | 16.6 | 67.7 (1260K) | **31.1** | 12.6 | 20.2 (373K) | **55.2** | 10.2 | 8.2 (160K) | **75.2** | 9.2 | 4.3 (88K) | **90.5** | 8.6 | 2.5 (52K) |
| GridMap + 3DDDA | 13.6 | 0 | 74.0 (1531K) | 23.4 | 0 | 43.0 (826K) | 30.6 | 0 | 32.9 (576K) | 35.4 | 0 | 28.3 (448K) | 38.8 | 0 | 25.8 (392K) |
| GridMap + Ours | **21.0** | 16.3 | 32.1 (739K) | **46.9** | 12.3 | 9.3 (205K) | **74.7** | 9.9 | 3.6 (80K) | **91.8** | 9.1 | 1.9 (40K) | **105.2** | 8.4 | 1.2 (23K) |

| Outdoor Dataset | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Resolution | 0.2m | | | 0.4m | | | 0.6m | | | 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 + 3DDDA | 0.7 | 0 | 1516.1 (28.4M) | 1.6 | 0 | 639.5 (10.5M) | 2.5 | 0 | 412.9 (6.5M) | 3.3 | 0 | 314.7 (4.8M) | 4.1 | 0 | 252.7 (3.8M) |
| OctoMap + Ours | **0.7** | 68.3 | 1395.8 (26.5M) | **2.1** | 57.0 | 449.1 (8.3M) | **3.8** | 51.1 | 231.8 (4.2M) | **5.9** | 44.5 | 137.5 (2.5M) | **8.2** | 41.3 | 89.0 (1.6M) |
| GridMap + 3DDDA | 1.4 | 0 | 783.1 (12.7M) | 3.3 | 0 | 321.6 (6.5M) | 5.1 | 0 | 207.7 (4.4M) | 6.5 | 0 | 162.1 (3.4M) | 7.7 | 0 | 136.1 (2.8M) |
| GridMap + Ours | **1.4** | 65.9 | 708.3 (11.0M) | **4.0** | 57.7 | 211.9 (4.6M) | **7.1** | 50.2 | 100.8 (2.5M) | **10.2** | 43.9 | 61.3 (1.5M) | **13.3** | 40.2 | 39.8 (1.0M) |

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

[10] SAM Coenen, JJM Lunenburg, MJG van de Molengraft, and Maarten 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.

[11] Daniel Maier, Armin Hornung, and Maren 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.

[12] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach, "Real-time compression of point cloud streams", in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 778–785.

[13] Kai M Wurm, Daniel Hennes, Dirk Holz, Radu B Rusu, Cyrill Stachniss, Kurt Konolige, and Wolfram 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.

[14] Erik Einhorn, Christof Schröter, and Horst-Michael 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.

[15] John Amanatides, Andrew Woo, et al., "A fast voxel traversal algorithm for ray tracing", in *Eurographics*, 1987, vol. 87, p. 10.

[16] Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, and Peter Shirley, "Interactive ray tracing for volume visualization", *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 3, pp. 238–250, 1999.

[17] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G Parker, "Ray tracing animated scenes using coherent grid traversal", in *ACM Transactions on Graphics (TOG)*. ACM, 2006, vol. 25, pp. 485–493.

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

[19] Alberto Elfes, "Using occupancy grids for mobile robot perception and navigation", *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

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