Efficient Multiresolution Scrolling Grid for MAV Obstacle Avoidance

Eric Dexheimer

CMU-RI-TR-20-26

August 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Michael Kaess (Chair)
George Kantor
Alex Spitzer

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Abstract

Fast, aerial navigation in unknown, cluttered environments requires a suitable map representation for path planning. In this thesis, we propose the use of an efficient, structured multiresolution representation for robot mapping and planning. We focus on expanding the sensor range of dense local grids for memory-constrained platforms. While multiresolution data structures have been proposed previously, we avoid processing redundant information and use the organization of the grid to improve efficiency. By layering 3D circular buffers that double in resolution at each subsequent level, objects near the robot are represented at finer resolutions while coarse spatial information is maintained at greater distances. We also introduce a novel method for efficiently calculating the Euclidean distance transform on the multiresolution grid by leveraging its structure. Lastly, we utilize our proposed framework to demonstrate improved stereo vision-based MAV obstacle avoidance with an optimization-based planner in simulation.

Acknowledgments

First, I would like to thank my advisor, Michael Kaess, for his guidance and support throughout the past two years. I really appreciate his patience with my learning and development. In addition, I am grateful for my colleagues in the Robot Perception Lab. I have learned a lot from everyone in the lab through countless discussions, and they have also made my experience at CMU very enjoyable. I would like to thank the rest of my MSR thesis committee, George Kantor and Alex Spitzer. I really appreciate the time, feedback, and questions they have given me. I am grateful for all of my colleagues in the Air Lab, on the DARPA Subterranean Challenge team, and in the FRC. It has been a pleasure working with all of you. Lastly, I would like to thank my family and friends for providing support during these last two years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As micro-aerial vehicles (MAVs) become more prevalent in the real-world, there is still a need for efficient on-board algorithms. Specifically, navigation in unknown and cluttered environments remains an active area of research. The major components of autonomy, such as localization, mapping, and planning, are all interdependent. Vision-based tracking requires smooth trajectories, low-drift odometry is needed to produce consistent maps, and planning needs an adequate map representation to generate safe and smooth trajectories.

Volumetric representations are frequently used to fuse data temporally into a spatially-organized map. Scrolling local grids maintain a constant memory footprint and can represent distance information densely, which is useful for trajectory optimization. However, the memory consumption of dense grids forces a trade-off between resolution and range for MAVs. Finer resolution is necessary for planning in cluttered environments, and longer-range is needed for fast and smooth flight. Although the limited range of these dense grids is compatible with RGB-D sensors, passive stereo cameras provide longer-range distance measurements.

To address this issue, we propose an efficient multiresolution scrolling grid for robot mapping and planning. Detail can be preserved near the MAV and grid limits can be extended while avoiding storage and computational constraints. A naive multiresolution grid implementation would perform independent map updates at each resolution before fusing the levels together. However, this requires computing redundant information and ensuring consistency across level boundaries. We propose an improved alternative which tracks level boundaries during sensor data integration and transfers data between layers as the grid scrolls.

While occupancy is sufficient for finding collision-free paths, the Euclidean distance transform (EDT) can provide gradient information for optimization-based planners to generate smooth paths. For multiple resolutions, a naive EDT computation would update each level before merging the results. Considering all levels jointly is difficult because of irregular dependencies with multiple levels present. We propose a novel method that handles these dependencies and leverages the grid's structure in order to improve efficiency.

Our main contributions are:

- A structured multiresolution occupancy framework that avoids integrating redundant information and handles data transfer between layers.
- A method for efficiently calculating the multiresolution EDT by exploiting the grid's structure.
- Simulation results demonstrating the memory and performance improvements over a single-resolution grid, as well as improved safety and smoothness of planned trajectories.

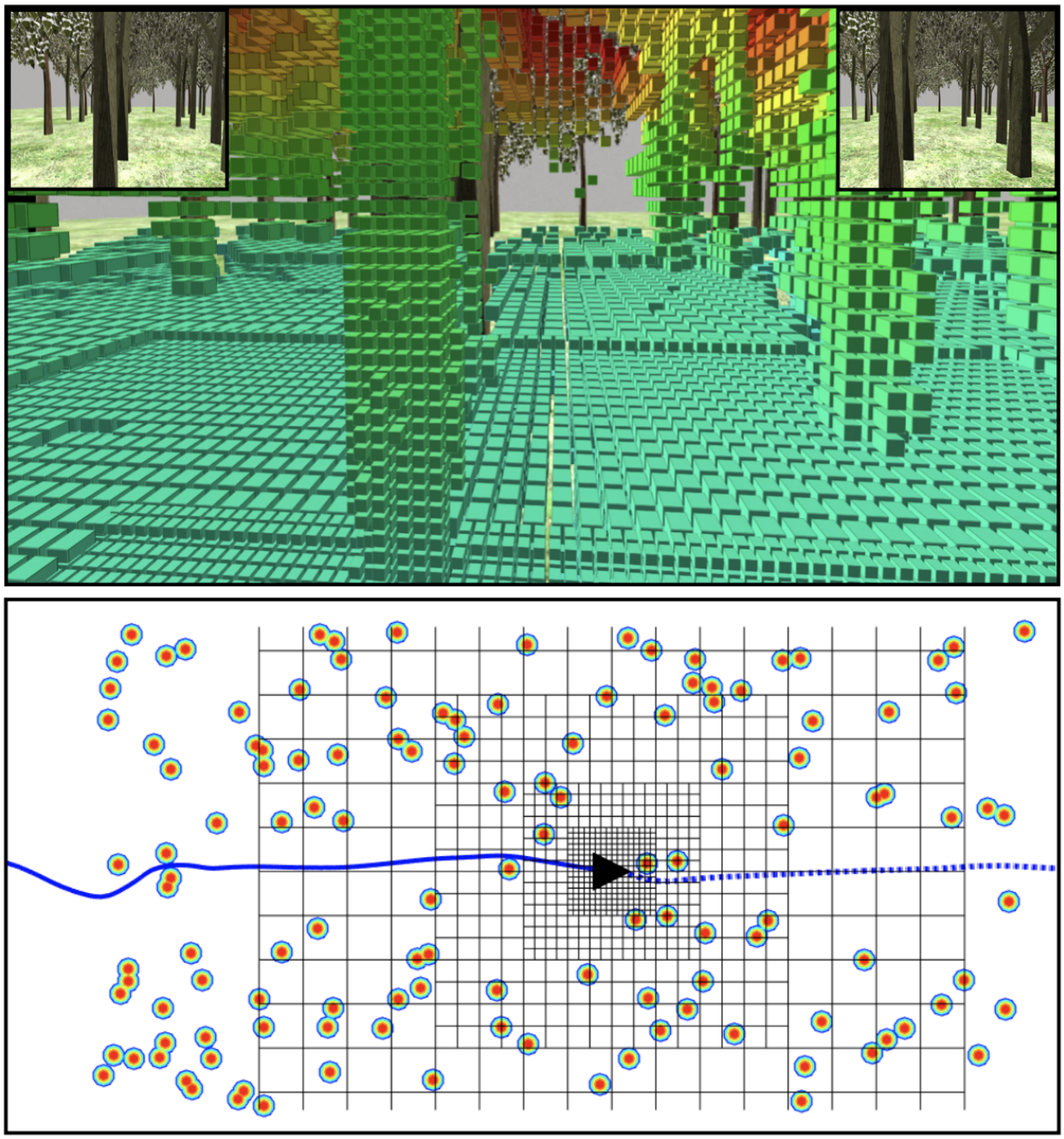

Figure 1.1: *(Top)* 3D multiresolution occupancy grid colored by height in simulated environment along with stereo images. *(Bottom)* Top-down view of multiresolution scrolling grid. The goal of the discrete grid is to approximate the true distance field for path planning.

Chapter 2

Related Work

A robot's map representation depends heavily on the task at hand. For collision avoidance, free-space information is needed. Some representations only maintain obstacles, and assume all other space is free. For example, obstacles can be represented by points in 3D space, such as those obtained from stereo triangulation. NanoMap performs a greedy uncertainty-aware point obstacle search in a history of frames, which is susceptible to stereo outliers [3]. Obstacle avoidance directly in disparity space has also been proposed [4], but the obstacle position directly corresponds to the raw sensor measurement. To reduce the influence of noise, a pose graph for a history of disparity frames, as well as a pseudo-occupancy measure, was introduced in [5]. While polar representations, such as egocylinders [6] and spherical maps [7], can filter out noise with temporal fusion, they are expensive to center around the MAV. Along with single-image methods, they are also subject to occlusion.

In contrast to point-obstacles, occupancy grids store free-space information in a spatially organized structure such that noisy sensor data can be fused temporally [8]. A visual comparison of volumetric data structures with frame-based methods is shown in Fig. 2.1. However, uniform grids consume large amounts of memory for large areas or fine resolutions. Better memory efficiency in global maps can be achieved by multiresolution maps, such as octrees, which prune repetitive spatial information [9]. In [10], a local dynamically-tiled octree is constructed, and cached tiles are loaded upon revisiting an area. Hierarchical voxel hashing reconstructs surfaces via a truncated signed distance field (TSDF) using varying resolutions of sparsely allocated blocks [11]. These global methods require dynamic memory allocation that can grow without bound, which reduces efficiency and cannot support long-term MAV flight. To maintain constant memory usage, a dense scrolling volume can be represented using a 3D circular buffer [12]. While the local grid is suitable for the limited range of RGB-D sensors, it cannot be extended to the range of stereo cameras without sacrificing significant memory or resolution. Finer resolutions allow for precise movement in cluttered environments, while coarser resolutions avoid myopic behavior. A comparison of this resolution trade-off is shown in Fig. 2.2.

Although occupancy or point-based obstacle information is sufficient for determining if query

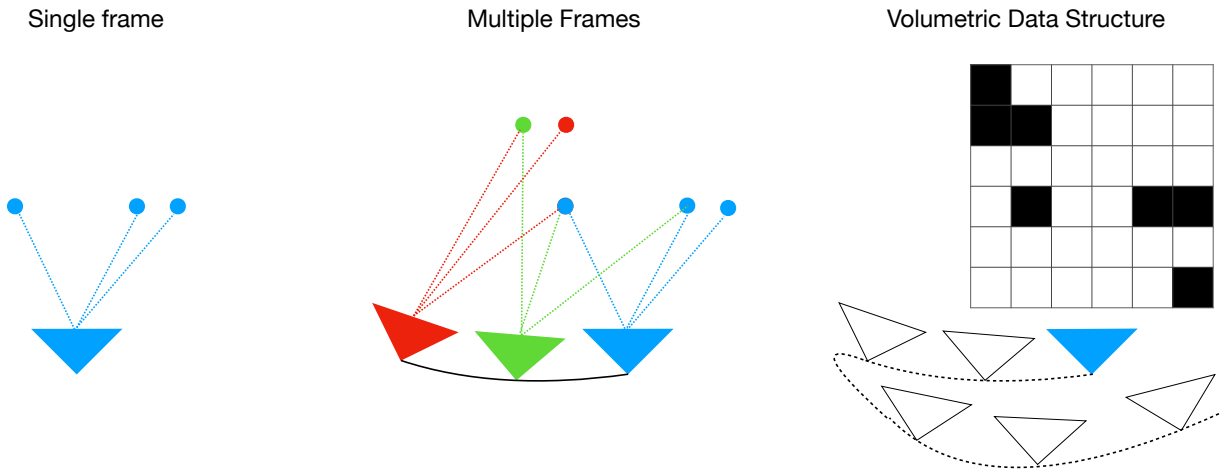

Figure 2.1: Visual comparison of single-frame, multi-frame, and volumetric methods. Volumetric data structures can fuse data temporally into a unified representation.

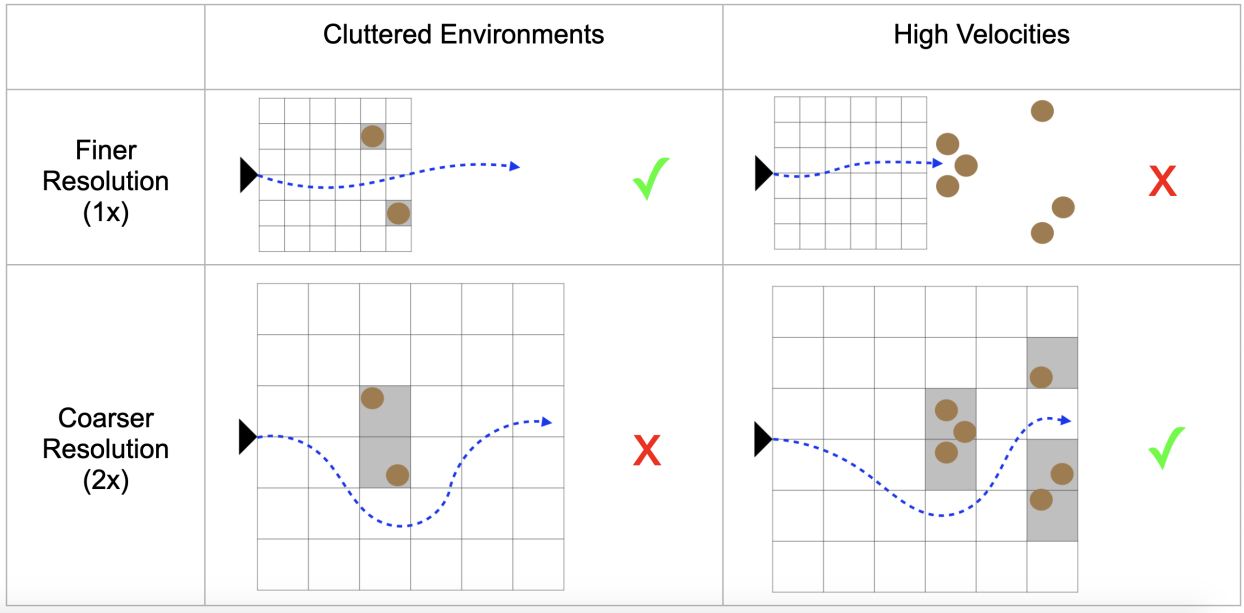

Figure 2.2: High-level comparison of finer and coarser resolutions in cluttered environments and for high velocities. Finer resolutions can better navigate tight spaces while coarser resolutions allow for planning further ahead.

points are in collision, it does not provide a smooth gradient for continuous-time planning. Trajectory optimization is beneficial for local MAV replanning since it avoids discretization of the state space and permits smooth flight, which in turn improves control, state estimation, and mapping [2, 13]. Similar to occupancy, the EDT is defined over a spatial grid, but provides the distance to the nearest obstacle [14]. In [2], a batch EDT is obtained from a scrolling occupancy grid to provide uniform B-spline trajectory optimization with a smooth obstacle cost. Incremental distance transforms have also been proposed to update scrolling grids [15] and larger environments on-board

MAVs [16]. However, these usually assume static environments, require a priority queue, and may touch the same cell multiple times per update. In contrast, local planning for MAVs should be able to update the representation quickly even with large changes in the environment. Since local trajectory optimization requires dense distance information, it is well-suited for a scrolling map representation.

In [17], a static multiresolution TSDF and EDT is used to map specific areas of a room with varying detail. The GPU processing affords the use of extra computation, such that the TSDF and EDT layers are updated independently before merging the results. Similar to [17], we organize all resolutions into a robocentric grid for efficient mapping. However, our method is for CPU processing, so we do not process redundant information. We also handle independently moving and potentially misaligned layers. A layered 3D circular buffer data structure has been proposed for multiresolution surfel-grids [18], which maintains a history of point measurements and surfels for LiDAR scan registration. Unlike [18], we do not handle layers independently. Since there are existing, highly accurate stereo visual-inertial odometry methods, we do not have to keep a history of previous measurements for odometry, and instead focus on constant memory usage. We require efficient occupancy and EDT updates that avoid processing and storing redundant information.

Chapter 3

Structured Multiresolution Occupancy Mapping

We construct a multiresolution occupancy map by layering dense 3D occupancy grids, as shown in Fig 1.1. Starting from an initial resolution for the finest grid at layer $l = 0$, the resolution doubles for each subsequent layer. The grids do not rotate, and independently center themselves around the robot, which triggers data transfer between layers. Since each layer is a dense grid, coarser cells occluded by finer ones exist in memory, but are not actively maintained to avoid redundancy.

3.1 Data Structure

Each layer is a 3D circular buffer stored as a 1D array. A circular buffer is a fixed-size array that circles back on itself when indexed with respect to an offset. In other words, indexing one element past the last element will index the first element. Each level has its own integer offset $\mathbf{o}^l = (o_x^l, o_y^l, o_z^l)^T$ specifying the origin of the grid in the world frame. Circular buffers avoid copying large amounts of data when the map translates, as only the offset and trailing cells leaving the map need to be updated. By keeping the scrolling of layers independent, finer resolutions will scroll more frequently than coarser ones. One alternative would be to scroll only at the coarsest level, which would ensure all grid layers are always aligned. While this would simplify operations such as raycasting and the EDT calculations since there would be no offsets between grids, the robot could reach the edge of the finest grid before the coarsest grid scrolls. This would severely limit the finer resolution mapping volume in the robot's field of view.

Similar to [2], we restrict the grid dimensions to be powers of two so that we can use bitwise operations. However, we allow the z-dimension to differ from the xy-dimensions. To index into layer l of the grid, we first obtain a voxel's integer world coordinate $\mathbf{v}^l = (x^l, y^l, z^l)^T$ by translating the grid coordinates $\mathbf{g}^l$ to the world frame via the offset $\mathbf{o}^l$:

$$\mathbf{v}^l = \mathbf{g}^l + \mathbf{o}^l. \tag{3.1}$$

For a layer l with grid dimensions $\mathbf{d} = (d_x, d_y, d_z)^T$, and world coordinate $\mathbf{v}^l$, we can obtain the 3D circular buffer index along the x-axis $\bar{x}^l$ by using the bitwise AND operator $\&$:

$$\bar{x}^l = x^l \,\&\, (d_x - 1). \tag{3.2}$$

By restricting the dimensions to be powers of two, we can avoid using the modulo operator to get the remainder, which is more expensive to calculate. This equation ensures we do not exceed the grid dimensions by clearing bits that exceed the dimension. To find the full 3D index in the grid, we repeat the above for the y^l and z^l coordinates, and then find the 1D array location:

$$i^l = \bar{x}^l d_y d_z + \bar{y}^l d_z + \bar{z}^l. \tag{3.3}$$

By indexing into the grid using world coordinates, it is also straightforward to index parent and children cells. A parent cell is a coarser-resolution cell containing the current cell, while a parent cell has eight finer-resolution children cells. The world coordinate parent cell index n levels above, x^{l+n}, can be obtained by a right arithmetic bit-shift in each dimension

$$x^{l+n} = x^l \gg n. \tag{3.4}$$

Similarly, the children cells can be found by performing a left bit-shift and iterating over the resulting $2 \times 2 \times 2$ cube. Note that parent or children cells are not guaranteed to be inside the grid, since they are just virtual locations in the world space. We can check whether world x-coordinate x^l resides inside a grid layer l by using the boolean NOT !, bitwise AND $\&$, and bitwise NOT $\sim$ operators:

$$\text{isInsideX}(x^l) = !((x^l - o_x^l) \,\&(\sim (d_x - 1))). \tag{3.5}$$

This equation determines if the world coordinate is within the current grid coordinates given the offset. This can be combined with the y and z dimensions to find whether a voxel lies inside a 2D cross-section or 3D volume.

3.2 Occupancy Updates

Previous structured multiresolution grids, such as [17] and [18], update cells at each level independently before fusing the levels together. To avoid processing redundant information and to lessen the computational load, we always update occupancy at the finest possible level that a ray intersects. Coarser levels further from the robot ensure fewer cells need to be updated, but the raycasting needs to accommodate the change in structure. By centering each grid layer around the robot, we only need to check if the next cell along the ray is within the current level before switching to a coarser level. The structure of the grid prevents having to check the level of every cell, since the ray always marches toward coarser resolutions, which improves performance. Points triangulated outside of the grid are projected into the grid as free rays. At the finest voxel resolution, we perform the Bresenham algorithm [19], which increments the world coordinate along the axis of greatest change at each step. This is efficient since it uses integer operations, and our extension

only requires an additional check on the validity of the level and bit-shifts for indexing. Although cells at coarser levels may be flagged more than once, a fully multiresolution Bresenham was less precise and slower in our experiments. More details on the developed raycasting methods can be found in Appendix A.

For the occupancy probability, we update using a standard log-odds formulation, but also restrict each cell to be updated only once per disparity image. This handles inconsistencies caused by shallow viewing angles of large voxels [9]. This voting-based method is efficient, but a full sensor model could be utilized if desired, especially if other sensors were included.

Prior to integrating sensor data, all levels of the grid are scrolled to be centered around the robot, which will be discussed further in the next section.

3.3 Scrolling

For a single-resolution 3D circular buffer, scrolling only requires that trailing cells and the offset be updated. In our multiresolution grid, we present an efficient method for handling the transfer of data between finer and coarser levels. Since we scroll grid layers independently, they may become misaligned. This needs to be handled explicitly to avoid information gaps along edges.

3.3.1 Coarse-to-Fine

Occupancy grids only represent the probability of an obstacle residing in a cell, so there is no explicit sub-resolution information. Thus, when scrolling from coarse-to-fine, the problem is ill-posed. Rather than copying the probability from the parent to the children cells, we introduce uncertainty while maintaining a conservative estimate. Representing the log-odds of a cell at layer l as L^l, and the maximum log-odds L_{max} as unsigned integers, we set all eight children cells according to

$$L_c^l = \frac{1}{4}L_{max} + \frac{1}{2}L^{l+1}, \quad c = 1 \dots 8. \tag{3.6}$$

This equation ensures that cells retain their occupancy state for a threshold at $L_{max}/2$. However, since the log-odds value is forced closer to the threshold, future measurements will correct the spatial details more quickly. If the coarse cells were naively copied into finer cells, a large number of sensor measurements may be needed to carve out space. While not theoretically correct, the equation essentially introduces uncertainty into the occupancy state without changing it. This still avoids making navigation unsafe because cells are assumed to be occupied until sensor data disagrees with this hypothesis. Once sensor data determines that a finer cell should be free, then it will be carved out and allow for the drone to navigate tighter spaces.

A plot of the an occupancy cell scrolling into a number of finer levels without being modified by sensor data is shown in Fig. 3.1. Note that the line converges to a horizontal line at unknown occupancy, which is the value 128 when using a uint8 datatype for the log-odds, but cells will never transition to a new occupancy state.

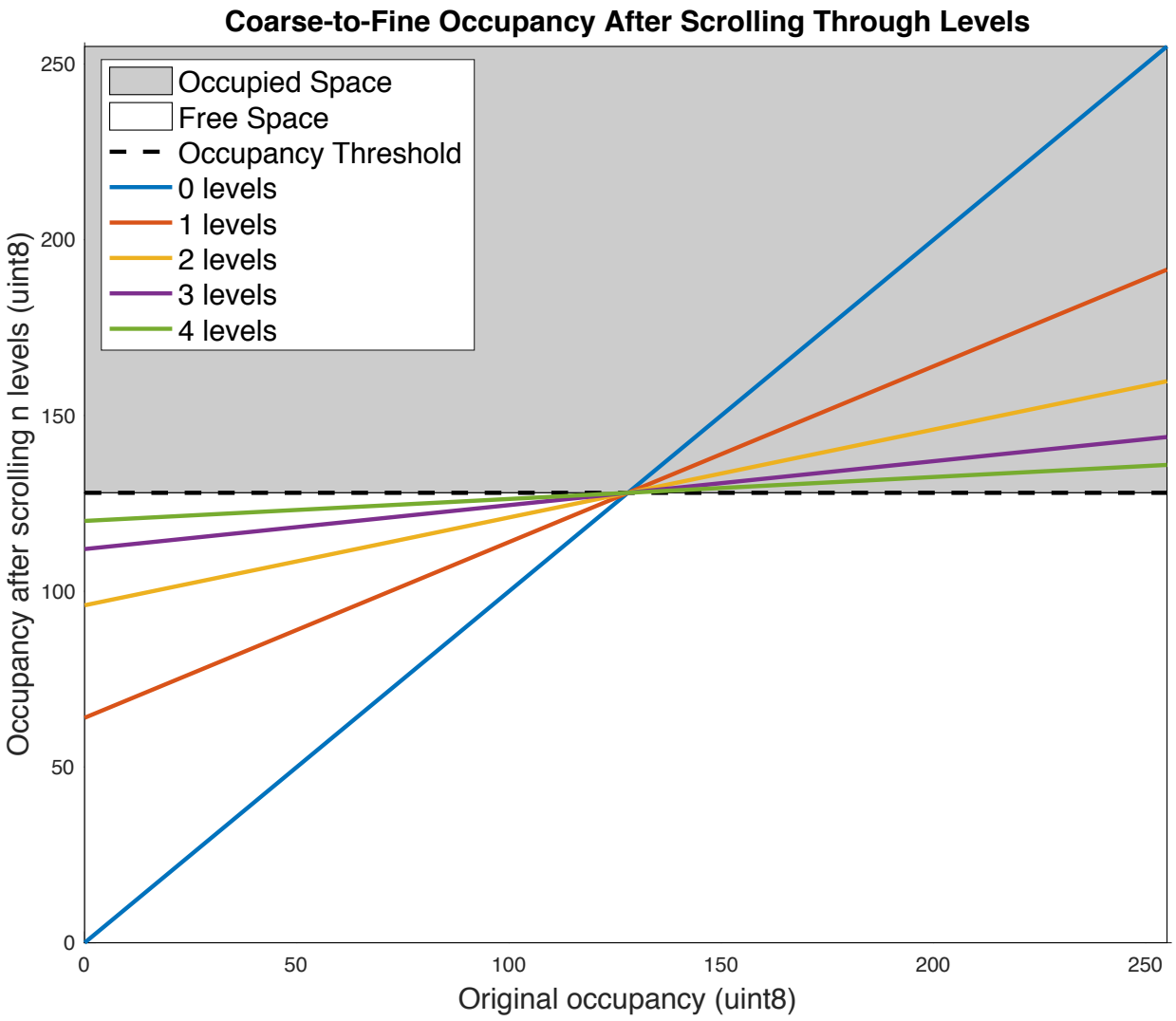

Figure 3.1: Plot of log-odds values after scrolling through a number of coarse-to-fine transitions. In this case, it is assumed that cells are not updated with sensor measurements.

3.3.2 Fine-to-Coarse

When scrolling from fine-to-coarse, we again use a conservative approach by taking the maximum log-odds from the set of children cells:

$$L^l = \max \{L_c^{l-1} : c = 1 \dots 8\} \tag{3.7}$$

where c is the index of the children cell. This is a standard formulation to ensure a cell is counted as occupied if any of the children cells are occupied.

The transfer of data between levels must be handled in a specific order to avoid extra copy operations. The following method is summarized in Algorithm 1. First, the offset of the coarsest level is adjusted by finding the scroll difference needed to center the robot in the grid. The cells from the trailing edge in each dimension are cleared, and now represent the leading edges. In the 3D circular buffer, no new memory is allocated, as the shift in the offset only affects the indexing into the 1D array. Next, the trailing edge of the second coarsest level is transferred to the coarsest level using the fine-to-coarse transfer described above and shown in Fig. 3.2. Since the two levels may not be aligned and partial offsets are possible, all children may not be transferred at the same time. Only the children cells that need to be placed at the leading edge are used. The level offset is updated, and then the coarse-to-fine transfer is performed for the leading edge. All levels including the finest level follow this procedure. Lastly, any coarse cells that become occluded by finer ones are cleared to zero. This ensures that when the grids continue scrolling, and these coarser cells become active again, they start with the smallest possible log-odds value. Furthermore, the max operation from fine-to-coarse scrolling can then be correctly applied even when there is a partial offset, at which point only some of the children cells are factored into the log-odds of the parent cell. To clear the occluded inner cells, the scroll difference from one level lower than that of the level being cleared is required.

Algorithm 1 Multiresolution scrolling

Input: Robot position $\mathbf{p}$,
Number of levels L,
Offsets $\mathbf{o}^l, l = 1 \dots L$,
Resolutions $r^l, l = 1 \dots L$
$\mathbf{s}^L \leftarrow$ GetScrollDiff($\mathbf{p}$, $\mathbf{o}^L$, r^L)
$\mathbf{o}^L \leftarrow \mathbf{o}^L + \mathbf{s}^L$
ClearEdges($\mathbf{s}^L$, L)
for l = $L - 1$ to 1 **do**
  $\mathbf{s}^l \leftarrow$ GetScrollDiff($\mathbf{p}$, $\mathbf{o}^l$, r^l)
  FineToCoarse($\mathbf{s}^l$, l)
  $\mathbf{o}^l \leftarrow \mathbf{o}^l + \mathbf{s}^l$
  CoarseToFine($\mathbf{s}^l$, l)
for l = 2 to L **do**
  ClearInnerCells($\mathbf{s}^{l-1}$, l)

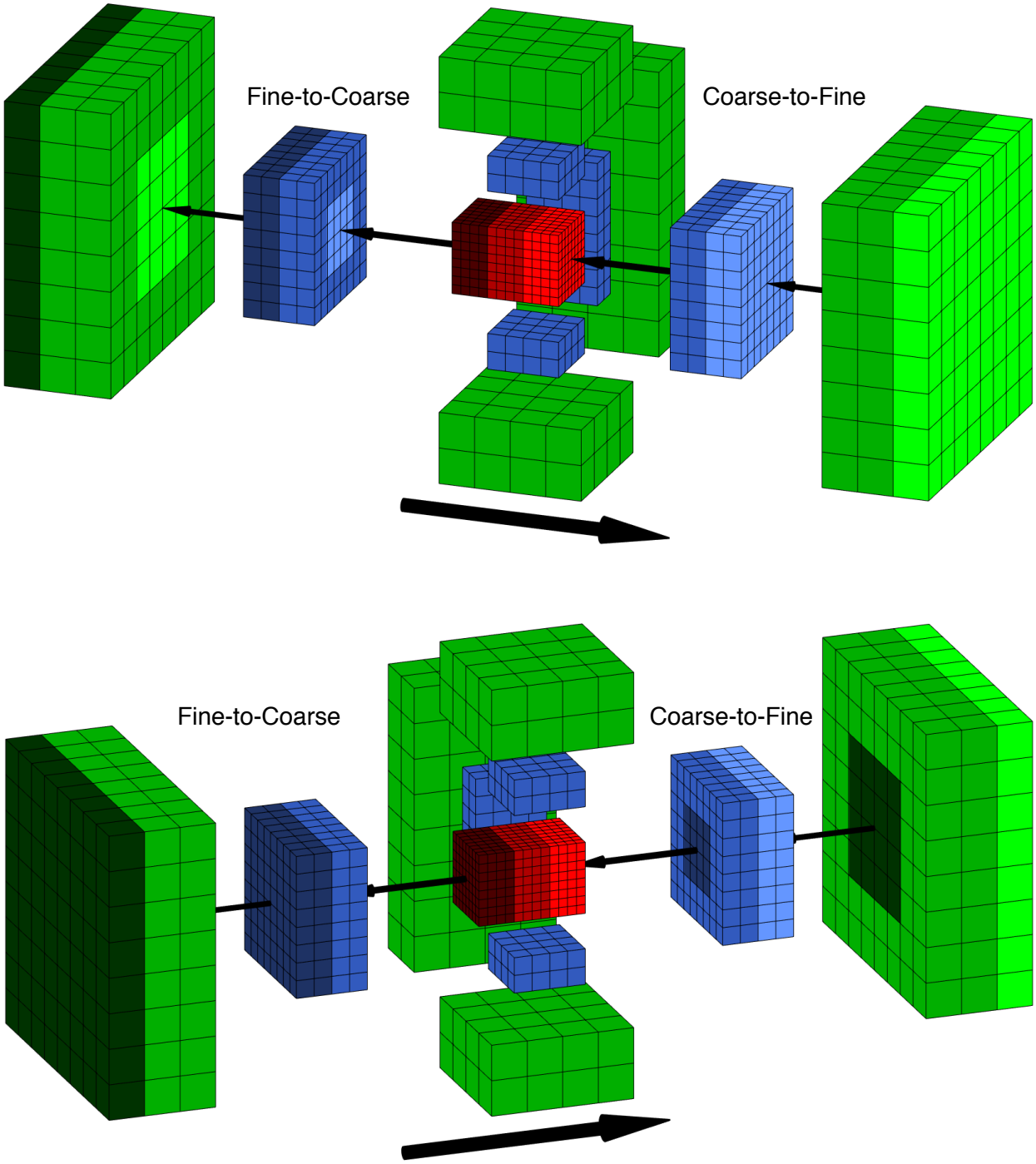

Figure 3.2: Two views of example scrolling between layers with the grid moving to the right. The order of highest resolution to lowest is red, blue, green. Darker colors within a level can be viewed as leaving the volume and being cleared, while lighter colors indicate cells entering the volume and being initialized. While not visible, coarser cells that transfer data to finer layers are also cleared.

Chapter 4

Structured Multiresolution Euclidean Distance Transform

From the occupancy grid, we can now determine the distance to the nearest occupied cell for every cell. The Euclidean distance transform (EDT) is a method for calculating these distances across the grid. By storing this in a volumetric data structure, distances can be interpolated at real locations, and gradients can be accessed for trajectory optimization.

While there are previous methods for calculating the EDT on uniform grids, we propose a novel method for efficiently calculating the EDT on a structured multiresolution grid. We avoid performing the EDT on each grid independently, and instead consider all grids jointly, such that redundant information is not processed. Similar to [20], [14], we decompose the calculation into 1D scans along each axis. However, the multiresolution grid has added complexity, since a 1D scan at the coarsest level has dependencies on multiple finer level scans. We first present the batch multiresolution case, where all operations can be done in place on one multiresolution grid. Since the Euclidean and Manhattan distances are equivalent in the binary 1D case, we first conduct a two-pass L1 scan in the z-axis to find the distance to the nearest obstacle for each cell along the scan direction. Next, L2 scans in the x-axis and then y-axis build upon previous scans by calculating the lower envelope of parabolas, which yields the 3D EDT. We also discuss speed improvements to the batch case at the expense of additional memory usage.

4.1 L1 Distance Scan

For the first scan of a binary image, the L1 and L2 distance are equivalent, so we calculate a two-pass L1 distance in the z-axis. The forward pass calculates the distance for free cells that follow obstacles, while the backward pass corrects overestimates in the true distance. This is useful since the L1 scan is significantly faster than L2 and does not require additional memory to store temporary results. Note that we neglect any of the sub-voxel distances in the x and y directions between coarse and fine cell centers.

The first step is to set all occupied cells to have a distance of zero, and all free cells to have a distance of infinity. For the forward pass, the distance for the next cell is

$$f(p) \leftarrow \min(f(p), f(q) + d(q, p)) \tag{4.1}$$

where p is the current cell position along the scan direction, q is the previous cell position, f is the sampled distance function, and $d(q, p)$ is the distance along the scan direction between the voxel centers of q and p. The same distance equation can be used for the backward pass as well. An example of a 1D L1 scan is shown in Fig. 4.1.

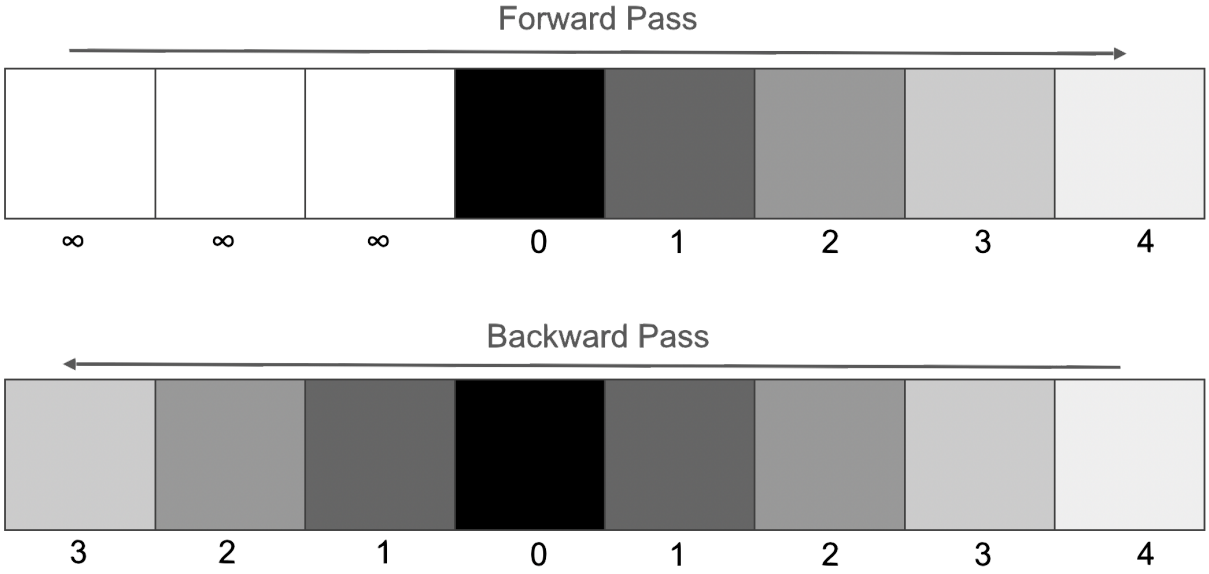

Figure 4.1: L1 scan on uniform grid consisting of forward and backward passes. Free cells are initialized to zero while occupied cells are set to infinity.

The forward scan is organized in the xy-plane into two groups: cells that overlap with finer levels, and cells that do not. For cells that do not overlap, a forward scan passes all the way through. Cells that do overlap are scanned until the next level, and the finer level is initialized. The same strategy is performed through the finest level. At this point, the finer levels initialize coarser levels with the minimum distance of the children. It can be viewed as a series of branching and merging scans, with a 2D example shown in Fig. 4.2. The same strategy is used for the backward pass, and all distances are squared to initialize the L2 scans.

When calculating distances across levels, we account for the partial offset between levels. The partial offset indicates whether the outer edge of a grid level aligns with the level directly above. If the partial offset is odd, the outer edge of the finer level is not aligned with an edge in the coarser level.

During the backward scan, finer cells may be an overestimate of the distance due to the many-to-one dependency between coarse and fine cells in the same scan. This happens when an occupied finer cell propagates distance forward in the scan, which is then "reflected" at the coarsest grid

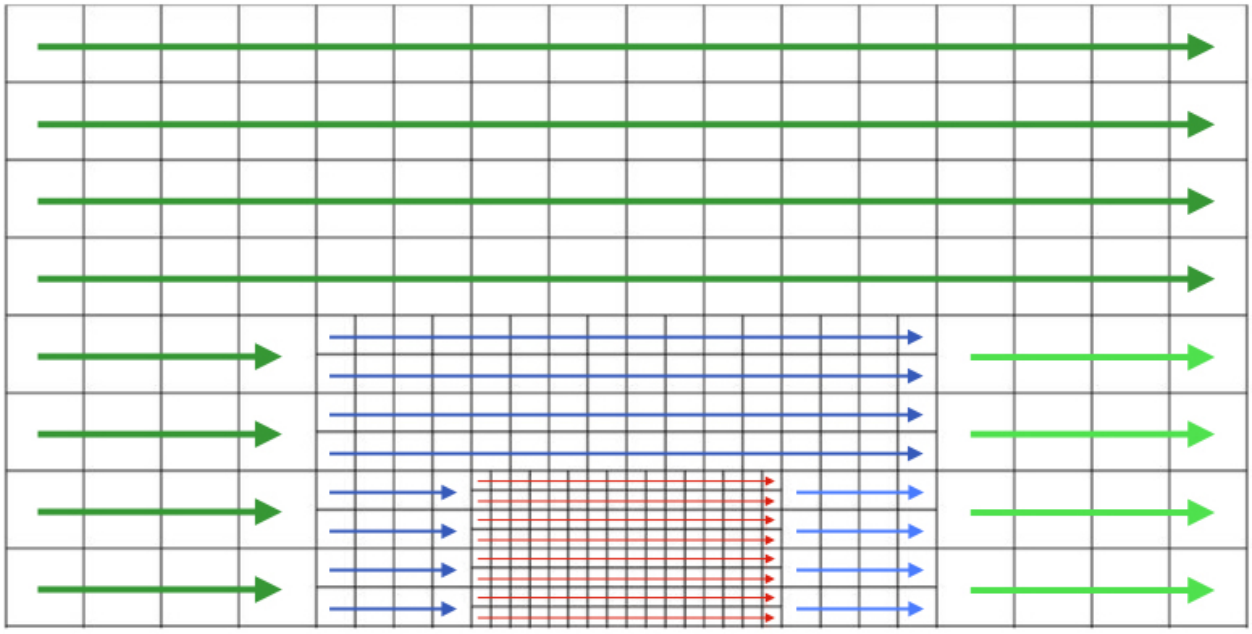

Figure 4.2: Forward pass for L1 scan. Darker colors indicate that the operation is performed first on that layer, while lighter colors are scans that must be initialized from finer levels. A similar backward pass is also needed to finish the scan.

boundary back into adjacent rows. Although this can introduce incorrect distances in the L1 scan, these cells will be corrected in subsequent L2 scans, since they are still overestimates of the true distance. An example of this case is shown in Fig. 4.3.

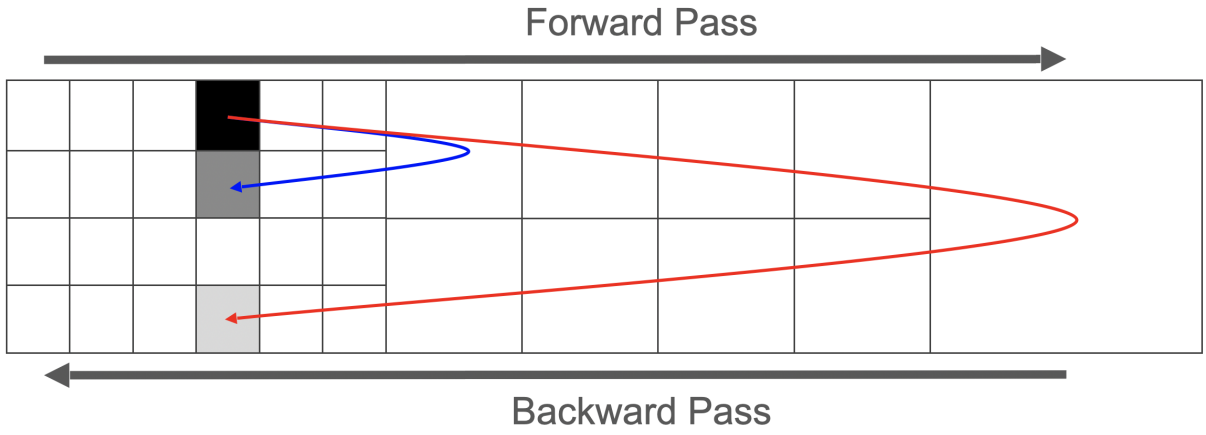

Figure 4.3: Reflection that yields incorrect distance in L1 scan. Since updates are done in place, distance from the occupied cell is propagated into the coarser levels, and then propagated from the coarser cells into finer ones on the backward pass. Two examples of cells that are given incorrect distance values are shaded here. This is an overestimate of the distance, so it is corrected in future scans. The correct version would be to have a distance of infinity in this case, but it will not change the result of subsequent scans.

4.2 L2 Distance Scans

An L2 scan is first conducted in the x-axis, and then the y-axis. Since we ultimately want the nearest squared distance to each cell, we can imagine placing a parabola at each cell, with the vertex being the distance calculated prior to the current scan. Then, we can find the intersections between these parabolas, and fill in the distances of the discrete locations in the grid by looking at the lower envelope of the parabolas. This is the high-level algorithm defined in [14].

More specifically, to get the lower envelope, the intersection s of two parabolas defined by vertices p and q is needed, which is shown in [14] to be

$$s = \frac{(f(p) + p^2) - (f(q) + q^2)}{2(p - q)}, \tag{4.2}$$

where f is the squared distance prior to including the current scan. The intersection point s is used to determine whether the current parabola should be added or the other parabola should be removed from the lower envelope.

In the single resolution case, since there are no levels and the vertices are uniformly spaced, it is straightforward to determine the next vertex and cell index. Thus, the vertex spacing is coupled with the cell index, and can be computed on-the-fly using integer coordinates, with the resolution only needed in post-processing. For the multiresolution grid, the spacing between parabola vertices is not uniform since the resolution varies and there are irregular boundaries when scrolling. To avoid calculating vertex positions, grid indices, and grid levels along the scan direction for all scans, we calculate these values once per EDT update for each direction according to the number of levels a scan must go through. For example, with three levels, the vertices are computed for scans intersecting with one, two, or three levels. Since scrolling will offset the layers with respect to each other, the vertices have to be computed every EDT update. This method saves computation during the lower envelope computation and distance fill-in since the vertices, indices, and levels can be directly accessed from the precomputed vectors.

The L2 scans need the distance values to be stored in a separate array before placing them in the grid, since they would otherwise introduce errors in the parabolic intersections. Thus, to avoid large amounts of temporary memory, the L2 scans are organized by scanning over cells at the coarsest level as shown in a 2D example in Fig. 4.4. However, we must calculate the lower envelope at the finest cell levels independently, and then merge by taking the minimum distance among scans that overlap at coarser levels.

In the 3D case, the scan cross-section is 2D, so we organize the cell dependencies using 2D Morton decoding as shown in Fig. 4.5. With a grid containing three levels, the lower envelopes are independent at the first level. In the second and third levels, a maximum of four and sixteen cells, respectively, must be merged by taking the minimum distance. We also need to monitor cases with partially overlapping cells. A check is done to see if a cell at the finest level used is actually inside the grid, and if not, the corresponding parent cell is used when scanning, as in the blue layers in Fig. 4.5. This ensures full coverage of the multiresolution grid and avoids calculating the same lower envelope more than once when the layers are not aligned.

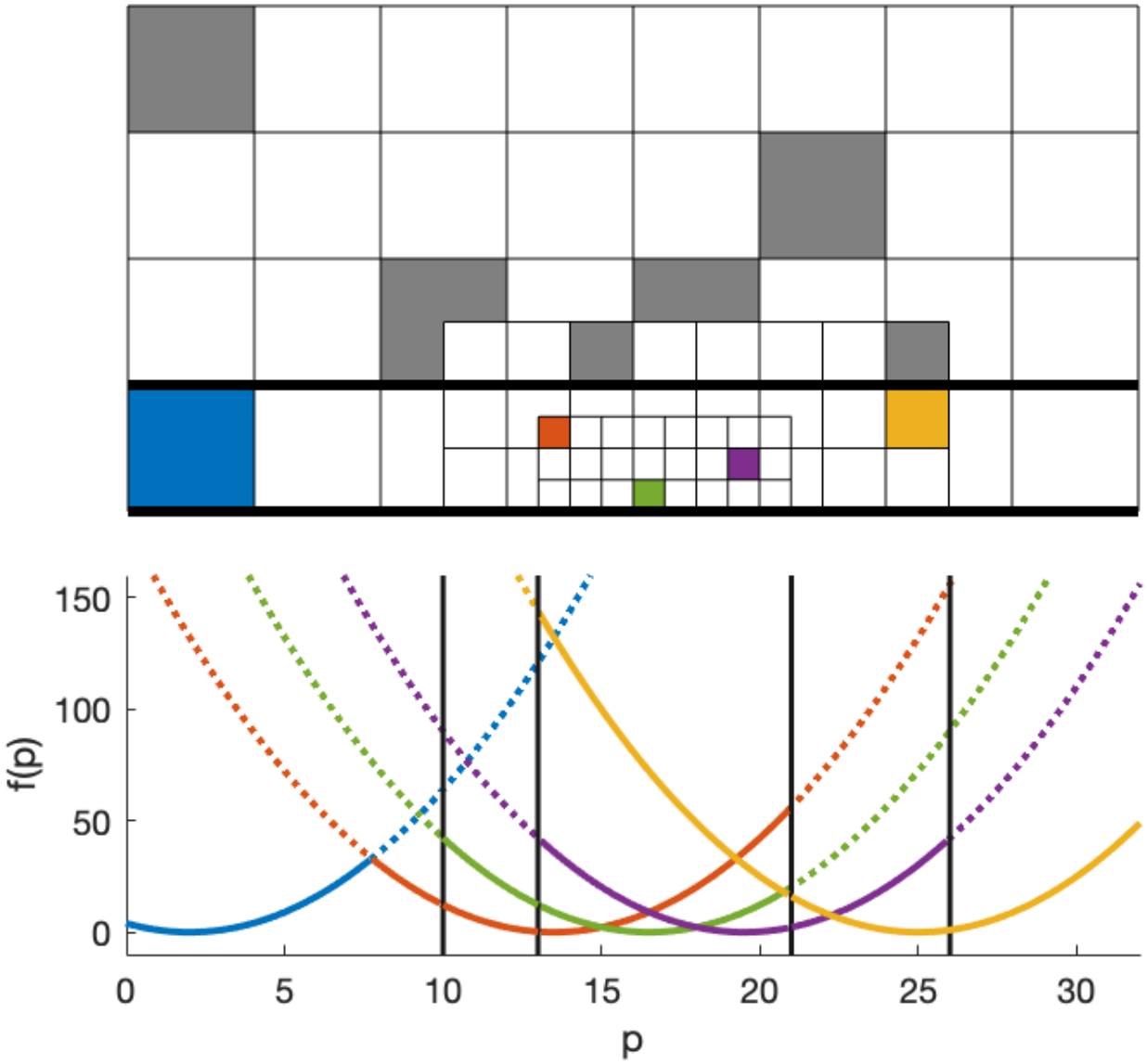

Figure 4.4: Multiresolution lower envelope calculation for bottom row of 2D half-grid. Occupancy is shown above, while the corresponding distance parabolas are below. Lower envelopes are shown as solid lines, while potential parabolas are dotted lines. The vertical lines show the transition from different layers in the scan. In the middle, since there are four valid rows in the finest level, each has its own lower envelope, while there are only two in the second level, and one in the coarsest level. Vertices are shown to be zero at obstacle locations, but will take different values when other scans are involved.

4.3 Beyond Full Batch Solution

We also examine a method for speeding up the distance transform when few cells are updated, without adding significant complexity. For robotic planning, optimization-based planners typically only require a distance up to a maximum value, after which no penalty is incurred in the cost

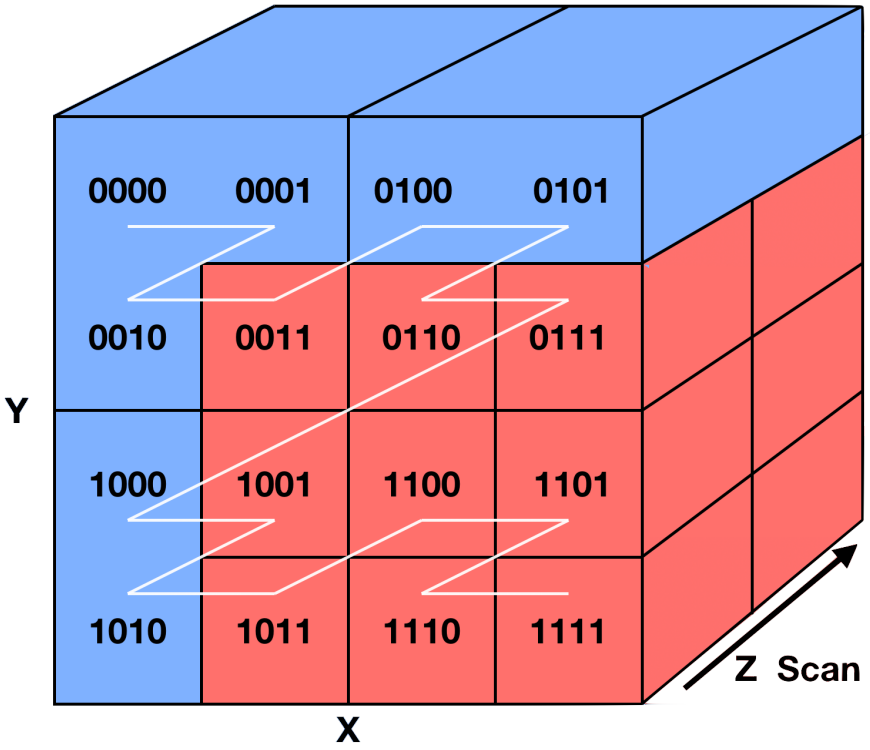

Figure 4.5: Organizing L2 scans at the finest possible level in the 3D case. Morton decoding takes the odd bits as the x-coordinate, and the even bits as the y-coordinate. This yields the iteration pattern shown by the white line, which is used to enforce dependencies among scans for coarser levels. In this case, twelve distinct L2 scans must be completed and merged for the coarsest level shown.

function. Thus, when cells switch from occupied to unoccupied and vice versa, only cells within the truncation distance of these modifications need to be updated. For each of the three axes, a 2D grid at the coarsest cell-level is used to indicate changes. When a cell in a cross-section flips, the indicator grid is dilated by the truncation distance, as shown in Fig. 4.6. This ensures all affected distance cells will be updated. The L1 scan is still computed from scratch at each update because it is not organized at the coarsest level and is relatively fast.

Instead of one 3D grid being updated in-place, a multiresolution distance transform grid is needed for each axis to maintain cells that do not need to be updated. Since our distance transform has no knowledge of the nearest obstacle for each cell, scrolling must be handled conservatively. Scrolling across layers from coarse-to-fine is handled by incorporating the squared distance between the parent cell center and the finer cell centers in the cumulative scan directions. Therefore, the distance along the z-axis is used for the first grid, while the complete distance is used for the third grid. Since updating the distance requires a non-constant square root in the former two cases, the scroll-time increases. However, this increase in scrolling time is a very minor performance hit compared to the speedup achieved in the EDT step since fewer cells need to be updated. For fine-to-coarse scrolling, the maximum distance is used, which means that similar to the occupancy case, coarse cells that are occluded must be cleared as well. Lastly, to ensure that the leading edge of the coars-

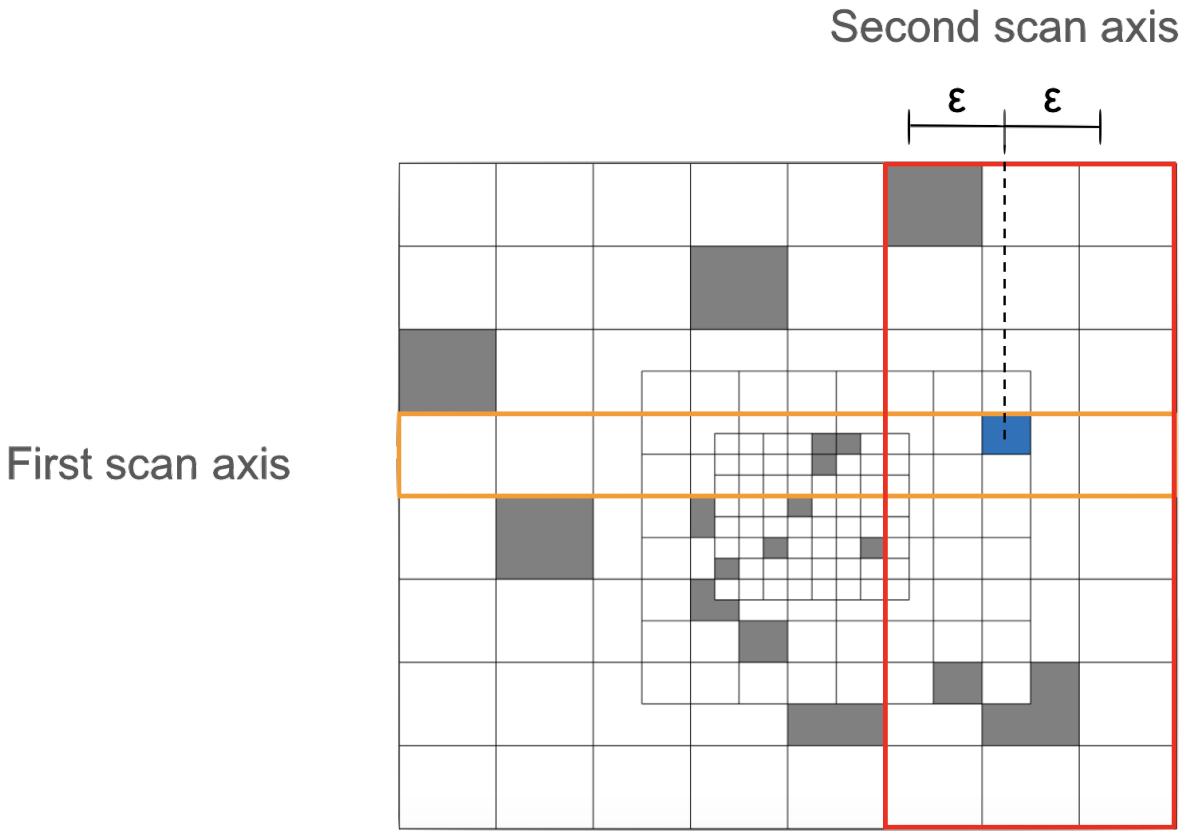

Figure 4.6: Example of updating relevant rows and columns for 2D multiresolution EDT. The blue cell switches from free to occupied, so the row at the coarsest level requires a new scan. In the second scan axis, the truncation distance is added on both sides to determine which columns need to be updated.

est level is correct, the distance of newly cleared cells and neighbors within the truncation distance are updated whenever scrolling occurs.

Chapter 5

Experiments

We demonstrate the memory and performance properties of the proposed representation compared to a baseline method. Simulated results in a realistic environment also demonstrate the use of an expanded grid range with fine resolution near the robot, as the safety and trajectory smoothness are improved.

5.1 Experimental Setup

A Gazebo simulation environment with textured random forest generation from [13] was used for all experiments. Realistic MAV dynamics and sensors are simulated using RotorS [1]. Noisy image data was also generated, which is then used for stereo semi-global block matching (SGBM) [21]. This results in realistic noise profiles of the 3D points used for mapping. An example of the Gazebo environment is shown in Fig. 5.1.

5.2 Memory Considerations

We compare the memory required to map a fixed volume with a single-resolution grid and our multiresolution map. For the uniform grid, we have the number of cells m_1:

$$m_1 = d_x d_y d_z. \tag{5.1}$$

With N layers that double in resolution, the number of cells needed to map the same-sized volume is

$$m_N = N \frac{m_1}{2^{3(N-1)}}. \tag{5.2}$$

Assuming storage of an occupancy grid (uint8), a flag grid (uint8) for updating occupancy, and the three Euclidean distance grids (float), then the uniform 64^3 grid is 3.670 MB. A summary of the memory comparison is shown in Table 5.1.

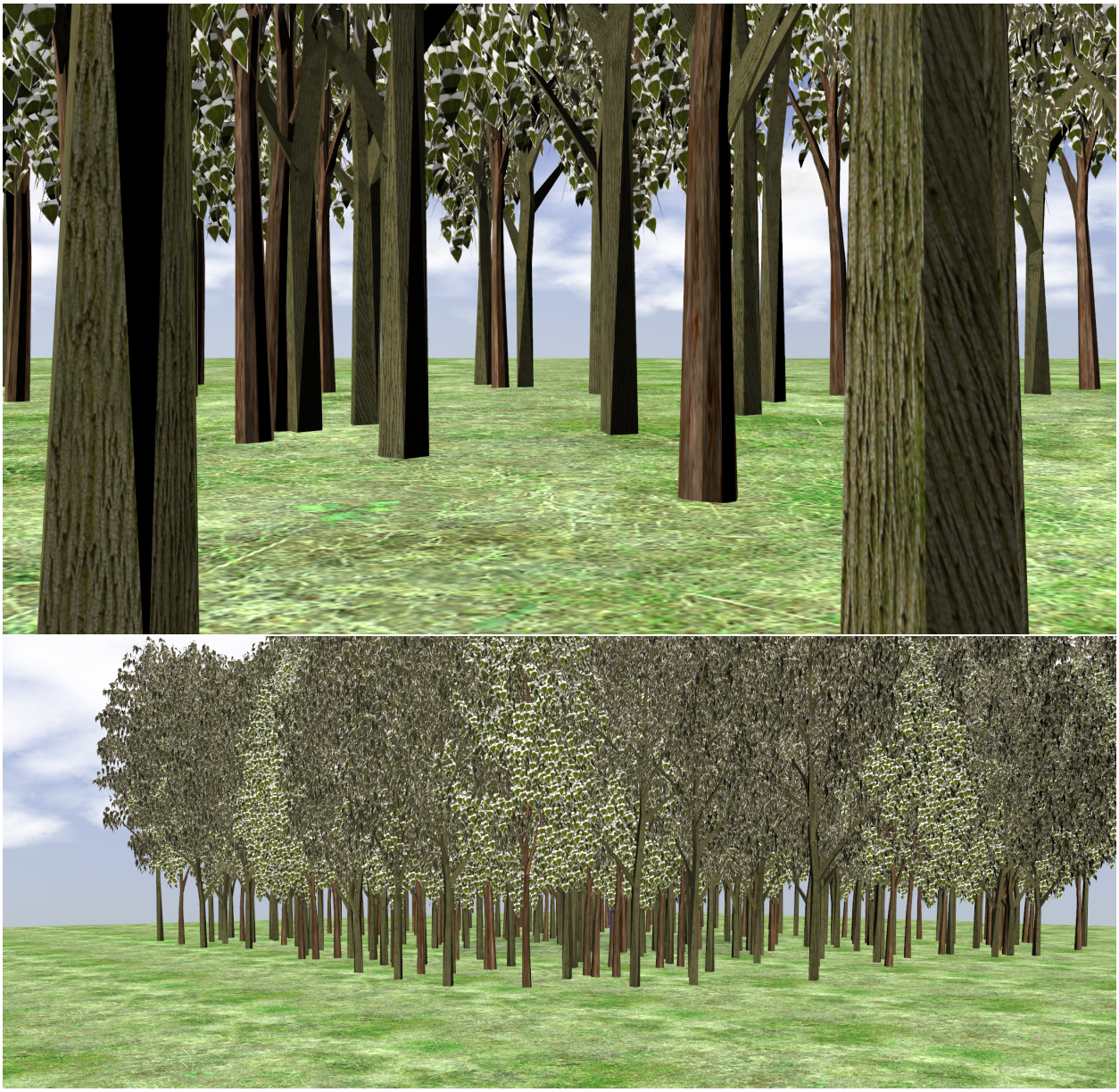

Figure 5.1: Example of realistic, randomly-generated Gazebo forest world used in experiments.

	Uniform 64^3	Uniform 256^3	Ours 64^3 $L = 3$	Ours $64^2 \times 32$ $L = 3$
Memory (MB)	3.670	234.881	11.010	5.505
Factor increase over 64^3 grid	1x	64x	3x	1.5x

Table 5.1: Memory comparison of uniform and multiresolution grids.

Although some coarser cells are occluded by finer resolutions in our representation, the extra memory is useful for preserving the circular buffer structure and efficiency. By reducing the overall memory usage, additional dense information beyond occupancy and distance fields, such as active localization fields [22], could be introduced into a local map without running into on-board memory constraints.

5.3 Timing

To demonstrate the efficiency of the proposed method, we compare the timing of the framework against [2] in Table 5.2. The timings are gathered from simulation runs in RotorS [1] using the uniform B-spline trajectory optimization from [2]. Increasing the dimension of a single-resolution grid quickly becomes infeasible, as shown by the 256^3 grid. Specifically, the distance transform calculation is not suitable for real-time, and the raycasting is significantly slower as well. Although fewer control points could be used with the 64^3 grid since it covers past the mapped area, it is interesting to note the speed-up for larger grids. The greater look-ahead properties of the map allow for better initialization and faster convergence in future steps. Even with the increased sensing horizon, the multiresolution grid provides comparable performance. Note that although the scrolling and raycasting times increase over the single-resolution 64^3 grid, these steps are much less expensive than the EDT update and trajectory optimization. The total times are thus comparable, even though the mapped area is much larger using the multiresolution grid. The additional memory needed over the single 64^3 grid is not significant, and the overall computation is similar as well.

Step	Baseline 64^3	Baseline 256^3	Ours 64^3 $L = 3$	Ours $64^2 \times 32$ $L = 3$
Scrolling	<0.1	1.6	<0.1	<0.1
Occupancy Update	1.0	8.2	3.9	3.4
EDT Update	11.6	631.3	25.8	18.2
Traj. Optimization	33.8	27.8	23.6	20.7
Total	46.4	668.9	53.3	42.3

Table 5.2: Average timing comparison in milliseconds between a "Baseline" uniform grid from [2] and our method. Raycasting is performed on disparity images downsampled to 160×120. Each grid from [2] uses a 0.15m resolution, and we use the same for the finest resolution. Thirteen control points were used for the trajectory optimization. Total average time assumes mapping and planning run at same frequency.

5.4 Planning Simulation

To show that increasing the mapped volume while maintaining a finer resolution near the robot improves the safety and smoothness of the resulting flight, we conduct simulation experiments in multiple environments. Fifty realistic 50m $\times$ 50m forests were randomly generated with a density of 0.1 trees per meter squared using a modified version of the forest generation in [13].

For baselines, we use 64^3 uniform grids from [2] with 0.15m and 0.3m voxel resolutions. For the multiresolution grid, we choose the dimensions for our method to be $64^2 \times 32$ with 3 levels since the timing best matches that of the uniform grid and it is only a 50% increase in memory. The finest resolution of the multiresolution grid is also set to 0.15m. We use the B-spline trajectory optimization framework from [2], and give a straight-line global trajectory through the forest to guide the local replanning. The max velocity is also varied from 1m/s to 3m/s in increments of 0.5m/s, so that for each max velocity and environment, ten trials are conducted. All occupancy, distance transform, and planning parameters are matched to isolate the representation differences. To measure smoothness, we use the integral of squared jerk over the planned path [23]. We denote the cost as

$$J = \int_{t_{\text{start}}}^{t_{\text{end}}} \left\| \mathbf{p}^{(3)}(t) \right\|^2 dt. \tag{5.3}$$

A qualitative example of the uniform and multiresolution grid simulation runs can be seen in Fig. 5.2. The overall success rates, mean smoothness, and standard deviation of smoothness are written in Table 5.3. Since the mean may be skewed by outliers, boxplots can provide a better comparison, as shown in Fig. 5.4.

Altogether, our method has a higher success rate and improved trajectory smoothness over both single-resolution grids. In addition, our method results in more consistently smooth trajectories as seen by the standard deviation and error bars in the box plot. Optimization-based planners are susceptible to local minima, so this explains the overall low success rates, but these results generally agree with that of [2]. In general, using a search-based planner to initialize the trajectory [24] or randomly restarting the optimization initialization as in [13] could mitigate this issue. For our purposes, it is sufficient to use only the optimization-based method since it allows for a more direct comparison of the map representations.

	Baseline 64^3 0.15 m	Baseline 64^3 0.3 m	Ours $64^2 \times 32$ $L = 3$
Success rate	0.538	0.504	**0.636**
Mean J (m^2/s^5)	219.188	196.463	**138.543**
Stdev J (m^2/s^5)	231.421	173.431	**165.410**

Table 5.3: Planning simulation statistics comparison over all 2500 runs. Mean and standard deviation of the integral of squared jerk are counted only for successful runs. The "Baseline" columns refer to the uniform grid implementation from [2].

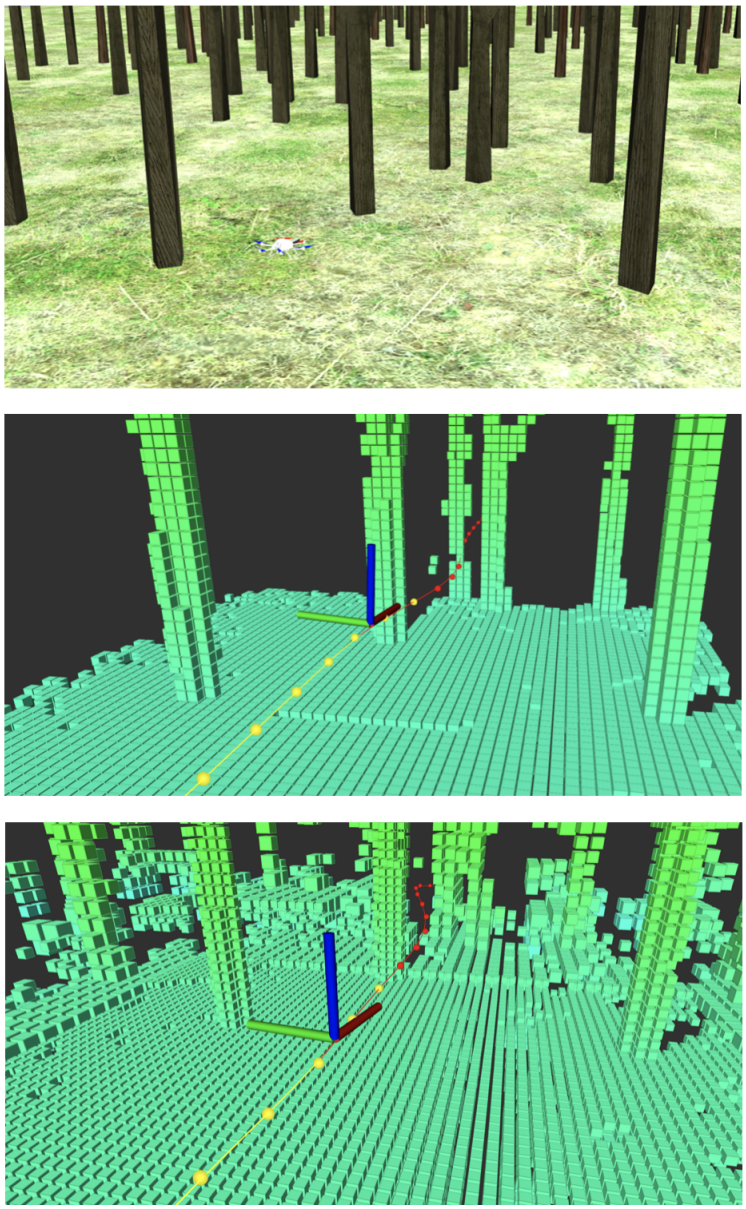

Figure 5.2: Simulation views of Gazebo environment using RotorS MAV [1] and planner from [2]. *(Top)* View of realistic simulation forest. *(Center)* Baseline grid plans around first tree, but new trees appear in collision after scrolling, resulting in less smooth trajectories. *(Bottom)* Our multiresolution grid allows for planning further ahead without sacrificing significant memory or computation.

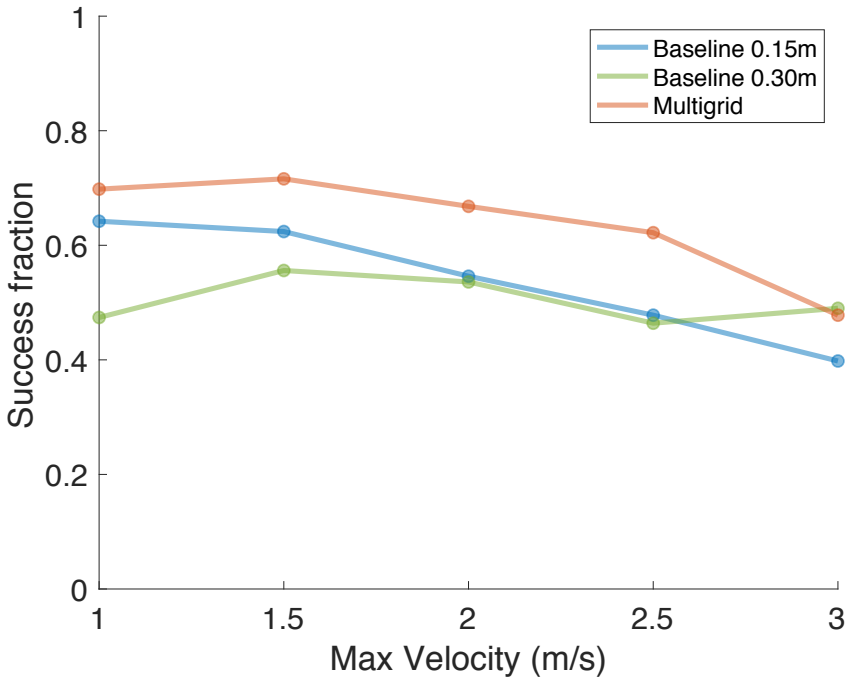

Figure 5.3: Success fraction vs. max velocity for different map representations in environments with 0.1 trees per meter squared. Each max velocity contains results across 50 environments and 10 trials per environment.

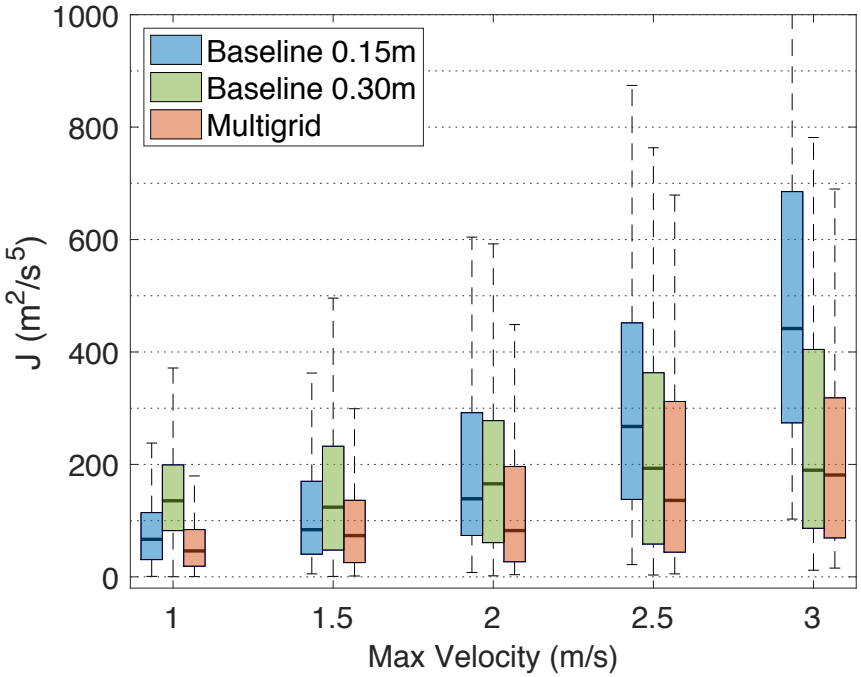

Figure 5.4: Box-plots of integral of squared jerk vs. max velocity for different map representations in environments with 0.1 trees per meter squared. Each max velocity contains results across 50 environments and 10 trials per environment.

The smoother trajectories from our method can be attributed to both the increased range over the 0.15m grid, and the finer resolution around the robot compared to the 0.3m grid, which allows for more precise distance and gradient computation in the trajectory optimization. Note that at lower velocities, the finer resolution baseline outperforms the coarser baseline, but as the velocity increases, the coarser baseline performs better due to the further look-ahead distance. In terms of smoothness, the multiresolution grid outperforms both baselines across all velocities, and the success rate is higher, except at 3m/s, where it is comparable with the coarser baseline.

Qualitative examples of 25 successful runs per method are shown in Fig. 5.5 and Fig. 5.6. In Fig. 5.5, which has a max velocity of 2m/s, the 0.15m uniform grid is myopic, and thus has very sharp turns at multiple points in the trajectory. While the 0.3m uniform grid reduces many of these, there are still points, such as at $y = -3m$ and $y = 11m$ to $y = 15m$ where there are consistently aggressive turns between trees. The multiresolution grid largely reduces these aggressive turns, but does appear to have a wider range of actions starting at $y = 10m$ than the 0.3m uniform grid. Although there are a couple of aggressive turns, the actions are generally less so than the 0.3m uniform grid. In Fig. 5.6, a different environment with a max velocity of 3m/s is shown. The reactive behavior of the 0.15m grid is even more evident, while the 0.3m grid also has more aggressive trajectories than for the 2m/s case. The multiresolution grid has mostly smooth trajectories, although there is some diversity in the trajectory selection at the start.

As the velocity increases, the gap between the multiresolution grid and 0.3m resolution uniform grid starts to decrease. Since the success rate drops to 50%, we reduce the density of the forest to be 0.05 trees per meter squared, and conduct experiments with max velocities of 3, 4, and 5 m/s. The success rate plot is shown in Fig. 5.7, while the smoothness boxplots are shown in Fig. 5.8. The success rate of the multiresolution grid is only slightly lower across all velocities, but the multiresolution grid starts to have a higher smoothness cost at 4 m/s. This can be attributed to the coarser 0.6m cells scrolling into the grid, but due to the high speed of the MAV, there are insufficient observations to carve out free space. The conservative assumption that the occupancy should not change as cells scroll to finer boundaries causes this issue, since the uniform 0.3m grid assumes all points outside the grid are still free space. While this does not result in a large gap in terms of success rate, the behavior highlights the potential coupling between grid size and max velocity. If the multiresolution grid had larger dimensions, then coarser cells would only be initialized much further from the robot, and have more opportunities to be carved out. Increasing the grid dimensions does come at the cost of increased memory and computation, especially for the EDT. Using a TSDF or a volumetric representation that contains sub-voxel information could be used at the expense of more complex and computationally-demanding scrolling logic.

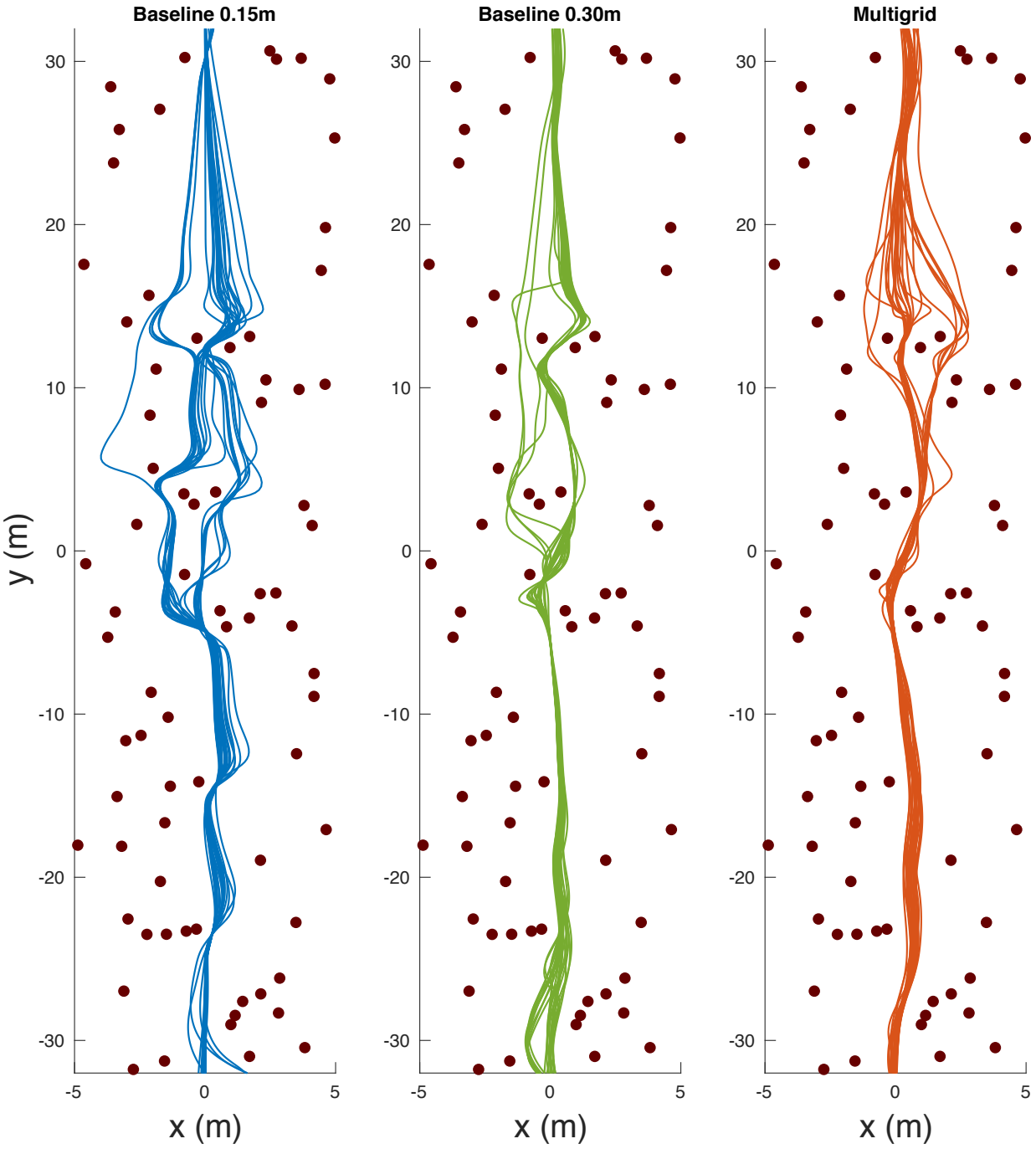

Figure 5.5: Qualitative example with max velocity of 2m/s. Each map representation is shown with 25 successful trials. The aspect ratio of the y-axis to the x-axis is magnified to enhance visualization.

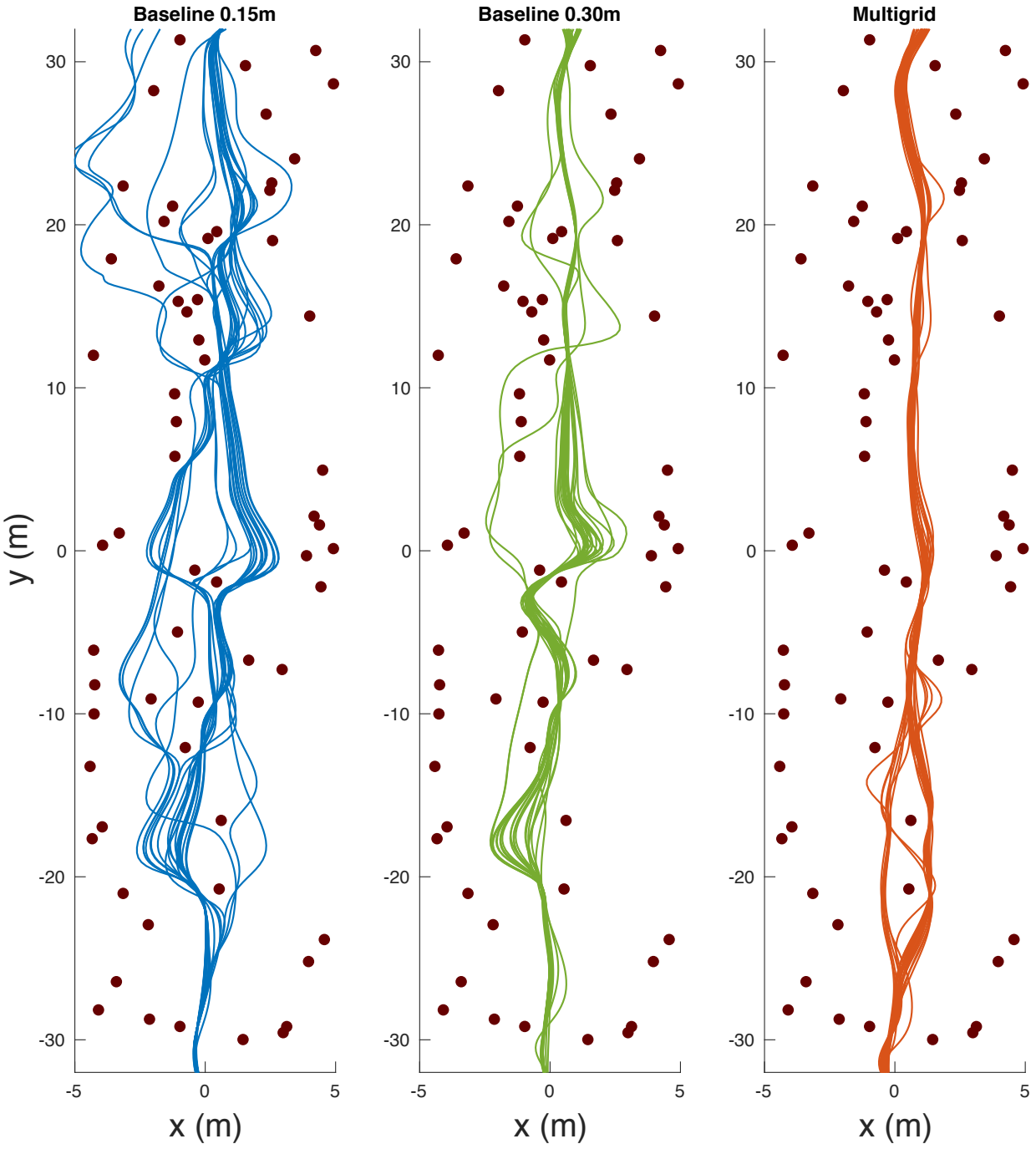

Figure 5.6: Qualitative example with max velocity of 3m/s. Each map representation is shown with 25 successful trials. The aspect ratio of the y-axis to the x-axis is magnified to enhance visualization.

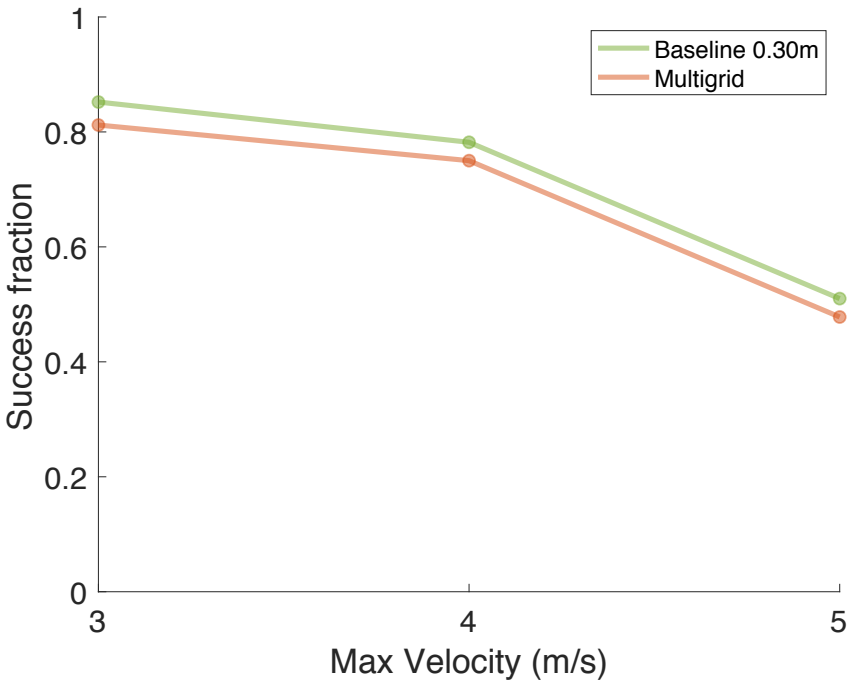

Figure 5.7: Success fraction vs. max velocity for different map representations in environments with 0.05 trees per meter squared. Each max velocity contains results across 50 environments and 10 trials per environment.

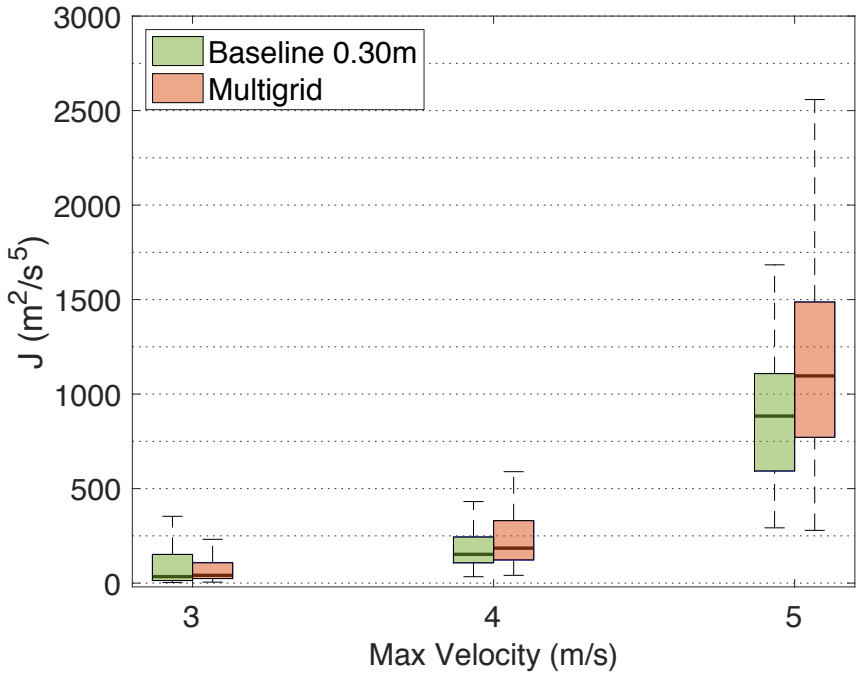

Figure 5.8: Box-plots of integral of squared jerk vs. max velocity for different map representations in environments with 0.05 trees per meter squared. Each max velocity contains results across 50 environments and 10 trials per environment.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have investigated the use of a structured multiresolution scrolling grid for occupancy and distance transform mapping. The multiple layers allow for representing the area near the robot at a finer resolution, while also expanding the sensing range. We presented efficient methods for updating occupancy and Euclidean distance information without processing redundant data. Due to its structure, the multiresolution grid provides both computational and memory benefits. Lastly, integration with an optimization-based planner in simulation demonstrated improved MAV flight in unknown, cluttered environments.

6.2 Future Work

Achieving even higher velocities may require modification to the presented map representation. Coarse voxels may provide over-conservative estimates of free-space, and without sufficient observations, these cells may not be carved out quickly enough. Using a representation that captures sub-voxel information, such as a TSDF or the occupancy formulation in [25], would allow for improved resolution when scrolling between layers. Incremental EDT methods could also be explored to compare against the method presented here. In addition, developing a receding-horizon planner would better suit fast obstacle avoidance, and hardware trials are needed to assess the behavior in different environments. Lastly, incorporating active visual information into the map could be used to supplement the trajectory optimization.

Appendix A

Structured Multiresolution Raycasting

When updating the occupancy values in a grid from sensor measurements, raycasting is often used. A ray from the robot's sensor is followed, updating cells along the way with information. This may take different forms, such as using a sensor-model that defines the probability of occupancy all along the ray, or a voting-method that only updates the cell containing the observed point with an occupancy hypothesis, and all other cells along the way with a free-space hypothesis. In either case, the cells that the ray may influence need to be tracked.

The Bresenham algorithm, is one framework, that first finds an axis-aligned driving-axis, and then marches in voxel-coordinates along this direction [19]. This algorithm is particularly efficient, because all calculations can be completed in integer coefficients. The true sensor position and observed point may be real-valued, so the Bresenham algorithm first rounds the endpoints of the ray into voxel coordinates. This discretization may introduce errors, especially at larger voxel sizes, as the utilized ray no longer corresponds to the true ray. One other popular algorithm, is the voxel traversal algorithm from [26]. In this case, floating-point coordinates are used, and all cells that the ray intersects are included. The downside of this method is the relatively higher computation needed, which becomes more pronounced with an increasing number of sensor points.

For the multiresolution grid, three raycasting methods were implemented: finest-resolution Bresenham, multiresolution Bresenham, and multiresolution voxel traversal. Examples are shown in 2D here, but in practice, the algorithms are easily extended to 3D.

A.1 Finest-Resolution Bresenham

In the finest-resolution Bresenham, the ray is virtually traced in the finest possible voxel size, and the current cell is always indexed up to its non-occluded parent cell. Since the grid is structured and centered around the robot, only one check for the parent level needs to be completed for each driving-axis iteration. The structure avoids having to check through all potential levels, so the ray always marches from the finest level to coarser levels. As mentioned previously, the Bresenham algorithm can introduce errors due to the discretization of the ray's endpoints, but this is minimized

when raycasting only at the finest-resolution, while also leveraging the speed of integer operations in the original algorithm. A visualization of 2D examples is shown in Fig. A.1.

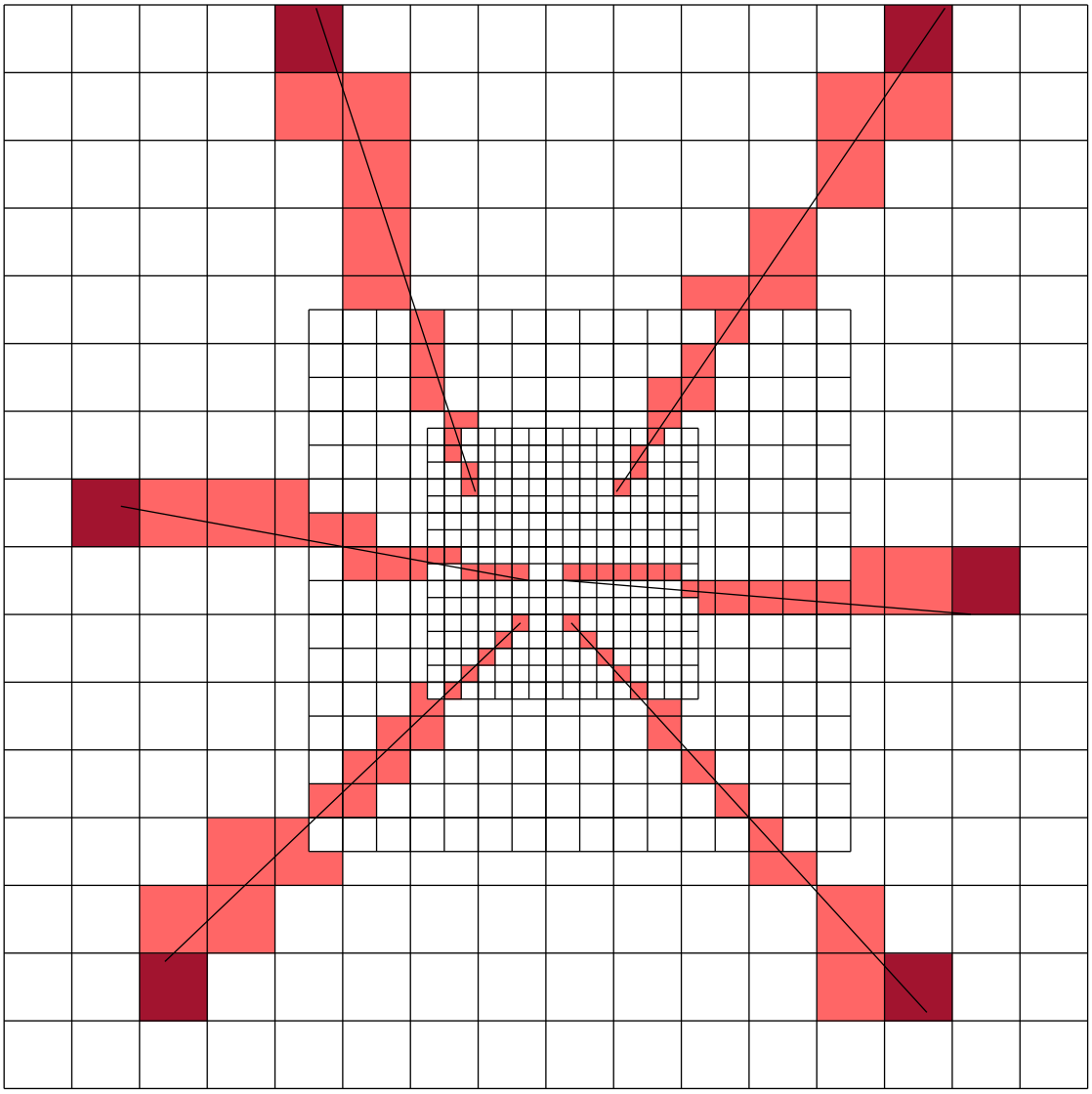

Figure A.1: Bresenham algorithm at finest resolution and indexing into parents of the grid in 2D. Discretization of endpoints to allow for integer operations can create inaccurate raycasting.

A.2 Multiresolution Bresenham

The multiresolution Bresenham algorithm essentially switches the algorithm to use coarser cells every time a level boundary is crossed. Compared to the finest-resolution Bresenham, there are fewer total iterations, since coarser cells are treated as a single cell, while the finest-resolution Bresenham only traverses virtually at the finest resolution. The multiresolution Bresenham can

introduce more significant errors at coarser resolutions, since iterating along the driving-axis may cause intersected cells to be skipped. Examples of the 2D results can be seen in Fig. A.2. For the practical dimensions used in this document, the multiresolution Bresenham was slower than the finest-resolution Bresenham, due to the logic needed to switch over to a coarser resolution loop. In some cases, the multiresolution Bresenham may be faster, if the dimensions of the grids are larger with relatively few levels, as duplicate work in the cell iteration steps may be avoided.

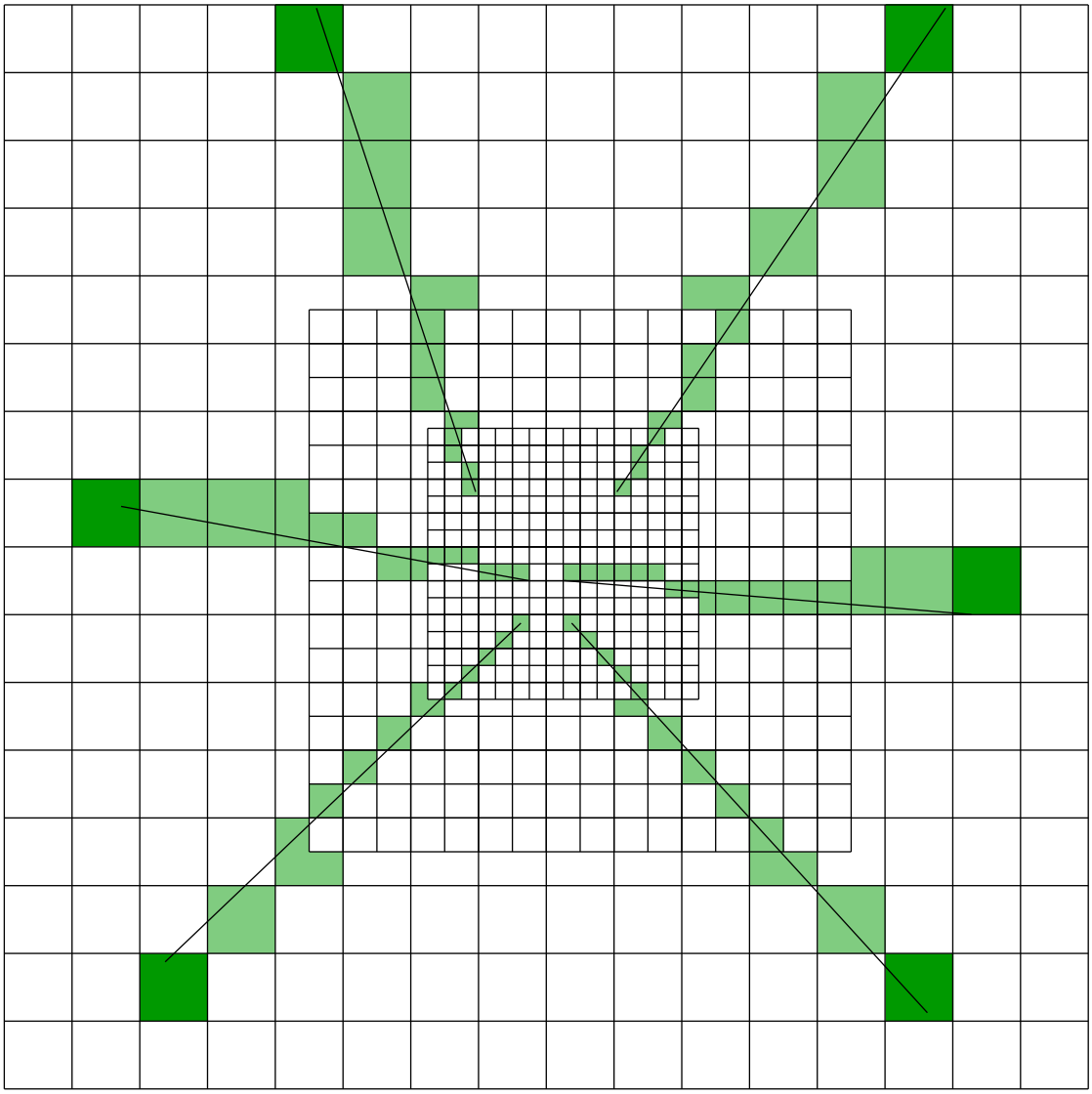

Figure A.2: Multiresolution Bresenham algorithm in 2D. Discretization of endpoints to allow for integer operations and coarser resolutions can both cause inaccurate raycasting.

A.3 Multiresolution Voxel Traversal

The multiresolution voxel traversal algorithm builds off of [26], by checking when the ray leaves a level, and updating intermediate variables accordingly. While it is the most expensive algorithm by around 2x-3x in general, it is also the most accurate, as integer-coordinate rounding and axis-aligned iteration are avoided. A visualization of 2D examples is shown in Fig. A.3.

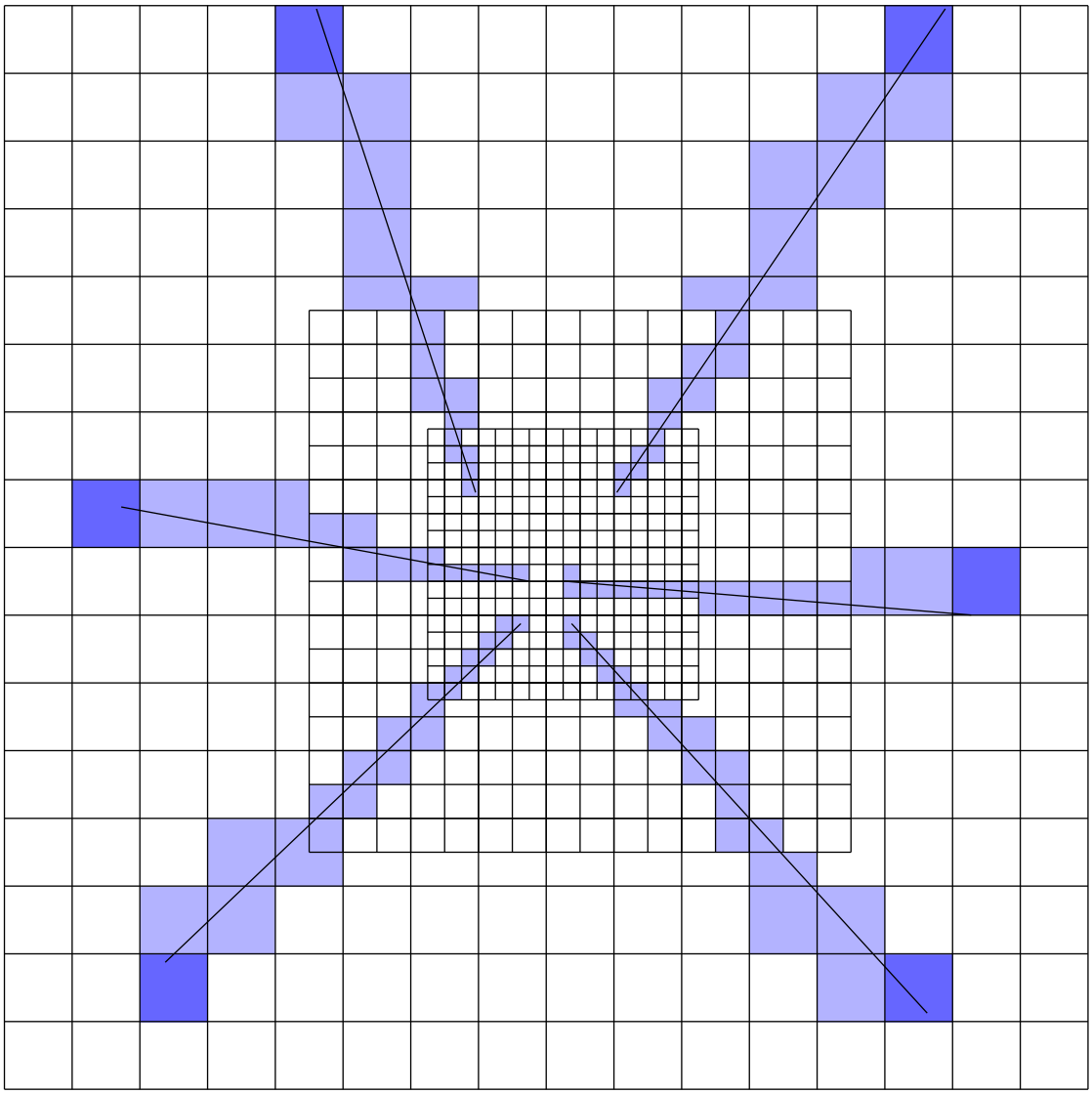

Figure A.3: Multiresolution voxel traversal algorithm in 2D. Note that all cells that the ray touches are flagged.

Bibliography

[1] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, 2016, pp. 595–625. 5.1, 5.3

[2] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 215–222. 2, 3.1, 5.3, 5.4, 5.4

[3] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "NanoMap: Fast, uncertainty-aware proximity queries with lazy search over local 3D data," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 7631–7638. 2

[4] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 3242 – 3249. 2

[5] G. Dubey, S. Arora, and S. Scherer, "DROAN — Disparity-space Representation for Obstacle AvoidaNce," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 1324–1330. 2

[6] C. Cigla, R. Brockers, and L. Matthies, "Image-based visual perception and representation for collision avoidance," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017, pp. 104–112. 2

[7] P. Gohl, D. Honegger, S. Omari, M. Achtelik, M. Pollefeys, and R. Siegwart, "Omnidirectional visual obstacle detection using embedded FPGA," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 3938 – 3943. 2

[8] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 2, 1985, pp. 116–121. 2

[9] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, 2013. 2, 3.2

[10] L. Heng, D. Honegger, G. Hee Lee, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous visual mapping and exploration with a micro aerial vehicle," *J. of Field Robotics*, vol. 31, pp. 654–675, 2014. 2

[11] O. Kähler, V. Prisacariu, J. Valentin, and D. Murray, "Hierarchical voxel block hashing for

efficient integration of depth images," *IEEE Robotics and Automation Letters (RA-L)*, vol. 1, pp. 192–197, 2016. 2

[12] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul. 2012. 2

[13] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 5332–5339. 2, 5.1, 5.4, 5.4

[14] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of Computing*, vol. 8, pp. 415–428, 2012. 2, 4, 4.2

[15] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh, "River mapping from a flying robot: state estimation, river detection, and obstacle mapping," *Autonomous Robots*, vol. 33, pp. 189–214, 2012. 2

[16] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 1366–1373. 2

[17] R. Wagner, U. Frese, and B. Bäuml, "Real-time dense multi-scale workspace modeling on a humanoid robot," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013, pp. 5164–5171. 2, 3.2

[18] D. Droeschel, J. Stückler, and S. Behnke, "Local multi-resolution representation for 6D motion estimation and mapping with a continuously rotating 3D laser scanner," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 5221–5226. 2, 3.2

[19] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, pp. 25–30, 1965. 3.2, A

[20] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and its Applications to Image and Signal Processing*, 2002, pp. 331–340. 4

[21] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 30, no. 2, pp. 328–341, 2007. 5.1

[22] Z. Zhang and D. Scaramuzza, "Beyond point clouds: Fisher information field for active visual localization," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2019, pp. 5986–5992. 5.2

[23] T. Flash and N. Hogan, "The coordination of arm movements: An experimentally confirmed mathematical model," *The Journal of Neuroscience*, vol. 5, pp. 1688–1703, 1985. 5.4

[24] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, pp. 3529–3536, 2019. 5.4

[25] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. Kelly, and S. Leutenegger, "Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping," *IEEE*

Robotics and Automation Letters (RA-L), vol. 3, pp. 1144–1151, 2018. 6.2

[26] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Proc. of the European Computer Graphics Conference and Exhibition (Eurographics)*, 1987, pp. 3–10. A, A.3