A Method for Fast Search of Variable Regions on Dynamic 3D Point Clouds

Eric Wahl[1] and Gerd Hirzinger[1]

Institute of Robotics and Mechatronics,
German Aerospace Center (DLR),
eric.wahl@dlr.de

Abstract. The paper addresses the region search problem in three-dimensional (3D) space. The data used is a dynamically growing point cloud as it is typically gathered with a 3D-sensing device like a laser range-scanner. An encoding of space in combination with a new region search algorithm is introduced. The algorithm allows for fast access to spherical subsets of variable size. An octree based and a balanced binary tree based implementation are discussed. Finally, experiments concerning processing time are shown.

1 Introduction

Many algorithms in 3D-vision use local information of a much larger scene [1, 2]. Separating a subset is a necessary but nontrivial preliminary step, especially when using unorganized point clouds. It strongly depends on the 3D-data representations and therein the realization of spatial neighborhood relations.

A common procedure to organize the data, is applying a spatial subdivision tree. Tree representations differ in their strategies when partitioning space, but they agree on relying on a hierarchical structure. Each data point is mapped to a *unique index* (*key*) that determines the location in the tree. The indexed location is referred to as a *(tree-) node*. Accordingly, an index could be interpreted as the location on a one-dimensional space-filling curve in 3D. The dilemma is that such a curve does not preserve spatial proximity in all directions [3, 4].

On the other hand, tree representations allow for organizing a sequential stream of orderless data dynamically, since an update with incoming data is possible, while access is still limited to complexity $O(log(N))$, where N is the number of nodes.

This work is related to the approach of Sagawa et. al. [5]. They introduced the *technique of the Bounds-Overlap-Threshold* for iterative closest point (ICP) search. The computational costs for registration tasks were reduced by limiting the search to a region within a selected threshold. Their approach accepts imprecision in return for faster computation. Furthermore, the algorithm only provides single points instead of a region. Since it is based on a *k-d tree*, an online balancing seems impractical.

Bodenmüller and Hirzinger use a tree representation capable for online surface-mesh generation [6]. The representation contains linked leaf-nodes, that facilitates access to the direct neighborhood. Adaption to searching a region of variable size would increase memory and processing costs for building the links and is thus infeasible.

Advancing a closest point search, we introduce a novel search algorithm for regions of variable size. It is capable to separate local subsets of much larger point clouds. The approach can be implemented either on an octree or a balanced binary tree and is already embedded in a scene interpretation application [7], that uses the DLR multi-sensory device [8, 9] for data acquisition.

The paper is organized as follows: Section 2 describes the index generation. Next, Section 3 introduces the region search algorithm. Section 4 includes experiments with respect to processing costs. Last, Section 5 closes the paper with a conclusion and prospects to future work.

2 Indexing

In this work, a hierarchical division of the space is applied, where space is a cube of edge size e_0. The first level of depth $d = 0$ consists of eight cubic cells of equal size. Every cell is divided in subcells, themselves. This is continued until a maximum depth $d_{\max}$ is reached. Cells of this level are referred to as *atomic cells*.

The *z-order*, also known as *morton code*, is used to address a cell. The function

$$m : \mathbf{p} \mapsto i \tag{1}$$

maps a point $\mathbf{p}$ to the index i, while

$$c : i \mapsto C \tag{2}$$

maps the index i to the cell C. The length l of an index is one octal digit per depth, where a digit defines one of the eight subcells. Accordingly, the prefix $\hat{i}_d(i)$ determines the d leading digits of the index i. It refers to the index of a higher located parent-cell $C_p = c(\hat{i}_d(i))$, that includes the cell $C = c(i)$.

The implementation of the space is realized as a tree. Thereby, a cell in space corresponds to a node in the tree. A node contains the mean point $\tilde{\mathbf{p}}$ of all points in the corresponding cell. This simultaneously allows for smoothing a point cloud and for restricting the point density to an upper bound.

In the following, two different types of (pointered) tree structures are applied.

2.1 The Octree

The octree uses the same structure as the hierarchical space division introduced above. Higher level cells are symbolized by nodes, while atomic cells refer to *leaf-nodes*. Only leaf-nodes contain information in terms of a 3D-point. Other nodes are exclusively used for traversing the tree. Therefore, an index can be interpreted as a path from the top to the bottom of the octree.

2.2 The Balanced Binary Tree

Adel'son-Vel'skiĭ and Landis [10] first introduced a balanced binary tree known as AVL-tree[1]. In this kind of tree, only atomic cells need to be realized as tree-nodes.

[1] The AVL-tree is used due to perfect balance and comparability to other trees, e.g. a red-black-tree.

The index of a cell still encodes the position of the node, but this position depends on the input order of data and does not exactly coincide with the spatial order. Memory costs for a balanced binary tree are lower than for an octree since binary trees need no additional nodes for higher level cells.

3 The Region Search Algorithm

The aim of every region search algorithm is to collect the points $\mathbf{p}$ that are within a radius r with respect to a region of interest (ROI)

$$\mathcal{F}(r, \mathbf{q}) = \{\mathbf{p} | \|\mathbf{p} - \mathbf{q}\| \leq r\} \tag{3}$$

around the center $\mathbf{q}$.

Step 1: Finding the Center of Search

The algorithm first needs to determine the starting point. For both tree variants the index $i_{\mathbf{q}} = m(\mathbf{q})$ is calculated. A cell has to be found, that is large enough to include the hole ROI, i.e. a cell of depth $d_{\mathcal{F}}$ is chosen if the edge size $e_{d_{\mathcal{F}}} = e_0/2^{d_{\mathcal{F}}}$ fulfills the condition

$$2r \leq e_{d_{\mathcal{F}}}. \tag{4}$$

By chance, the cell includes the complete ROI. Otherwise, if the center $\mathbf{q}$ is closer to the cell border than the size of the radius r, parts of the ROI penetrate neighboring cells. The sections of the ROI that are located in neighboring cells are further referred to as *outliers*. In 3D, each cell has a maximum number of 26 neighbors

$$\mathbf{p}_{[1\ldots6]} = \mathbf{q} + \frac{\mathbf{u}_{[1\ldots6]} - \mathbf{q}}{\|\mathbf{u}_{[1\ldots6]} - \mathbf{q}\|} r, \tag{5}$$

$$\mathbf{p}_{[7\ldots18]} = \mathbf{q} + \frac{\mathbf{v}_{[1\ldots12]} - \mathbf{q}}{\|\mathbf{v}_{[1\ldots12]} - \mathbf{q}\|} r, \tag{6}$$

$$\mathbf{p}_{[19\ldots26]} = \mathbf{q} + \frac{\mathbf{w}_{[1\ldots8]} - \mathbf{q}}{\|\mathbf{w}_{[1\ldots8]} - \mathbf{q}\|} r, \tag{7}$$

where $u_{[1\ldots6]}$ are the projections of $\mathbf{p}$ onto the six cell sides, $v_{[1\ldots12]}$ are the projections of $\mathbf{p}$ onto the 12 cell edges, and $w_{[1\ldots8]}$ are the eight cell corners. Condition 4 reduces the number of outliers to 7 neighbors. That leads to 8 starting points at most[2]. The following steps are individually applied for each starting point.

Step 2: Prefix Descent

All points $\mathbf{p}_s$ and $\mathbf{p}_t$ included in the space of a sub-cell $C = c(i)$ have the common prefix

$$\hat{i}_{d_{\mathcal{F}}}(m(\mathbf{p}_s)) \qquad \forall\{\mathbf{p}_s, \mathbf{p}_t | c(m(\mathbf{p}_s)) = c(m(\mathbf{p}_t))\}. \tag{8}$$

Therein, the algorithm descends the tree until a node with consistent prefix is found. If such a node $\mathbf{n}$ exists, the search continues therefrom, otherwise it terminates.

[2] That is the case, if the ROI is located in a corner of the cell.

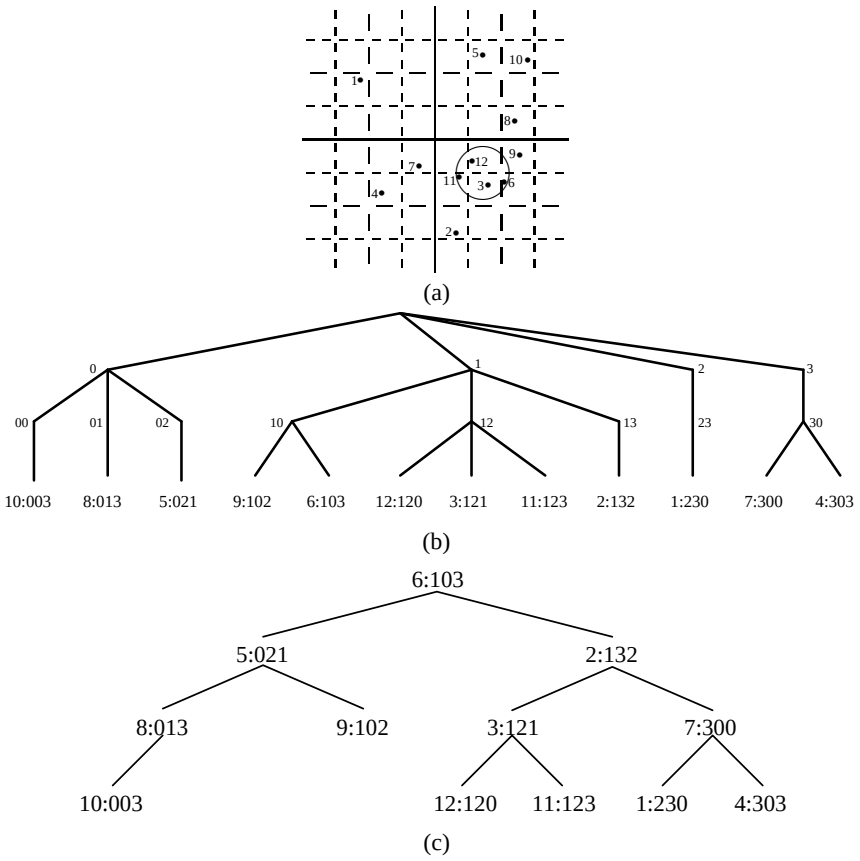

Fig. 1. (a) The example shows a space divided in hierarchical cells. A set of 12 input points $\mathbf{p}_i$ is used ($i \in [1, 12]$). The set $\Omega = \{230, 132, 121, 303, 021, 103, 300, 013, 102, 003, 123, 120\}$ contains the corresponding indices. The notation 1:230 refers to the point $\mathbf{p}_1$ with the index $i = 230$. The circle symbolizes the region of interest. All points inside of this region must be the result of the search. The size and position of the region leads to the prefix $\hat{i}_{d_{\mathcal{F}}=2}(i) = 12$. The cell is able but not presumed to include the focus. Therefore, the neighborhood has to be also examined. Two of the eight possible neighborhood positions point to cells with different prefixes and thus need exploration. (b) The two-dimensional version of an octree is the quadtree used here. The algorithm descends to the starting point in penultimate level, where the index and the prefix match. There, the points $\mathbf{p}_3$, $\mathbf{p}_{11}$ and $\mathbf{p}_{12}$ are received. Next, the prefixes $\hat{i}_{d_{\mathcal{F}}=2} = 10$ for the outliers are calculated and the points $\mathbf{p}_6$ and $\mathbf{p}_9$ are added to the result. (c) In the balanced binary tree the first match of the index with this prefix is searched. That leads to the point $\mathbf{p}_3$. The exploration of the subtree of point $\mathbf{p}_3$ adds the points $\mathbf{p}_{11}$ and $\mathbf{p}_{12}$. Then the outlier search starts for the prefix $\hat{i}_{d_{\mathcal{F}}=2} = 10$. It is successful in the root-node, where the point $\mathbf{p}_6$ is located. Calling the *explore*-operator on the left branch prunes the left side of the child and provides the point $\mathbf{p}_9$ on the right branch. Applying it on the right child prunes the right branch and leads to an already examined subtree, thereupon the search ends.

Step 3: Collecting Points

The collection of points depends on the tree organization. Hence, both trees have to be discussed separately.

Collecting Points in the Octree: Since the octree possesses a spatial organization, the search is straight-forward. All leaf-nodes in the subtree of node $\mathbf{n}$ are checked. If the distance of the mean point $\tilde{\mathbf{p}}_{\mathbf{n}}$ to the center $\mathbf{q}$ of the ROI is smaller than the radius, it is accepted. The complexity in the worst case is

$$O(8\log_8(d_{\mathcal{F}}) + 8^{1+d_{\max}-d_{\mathcal{F}}}\log_8(d_{\max}-d_{\mathcal{F}})). \tag{9}$$

That occurs, when the subtrees of all starting points contain the maximum number of entries and are of the same size.

Collecting Points in the Balanced Binary Tree: The position of a node in a balanced binary tree is less correlated with the spatial position than in the octree. Accordingly, additional operations are required in order to navigate. Comparing the prefix $\hat{i}_{d_{\mathcal{F}}}$ of node $\mathbf{n}$ to the prefix $\hat{i}'_{d_{\mathcal{F}}}$ of a child $\mathbf{n}'$ of $\mathbf{n}$ by the operator

$$\textit{explore}(\hat{i}_{d_{\mathcal{F}}}, \hat{i}'_{d_{\mathcal{F}}}) \begin{cases} \text{explore the right branch of } n', \text{ if} & \hat{i}'_{d_{\mathcal{F}}} < \hat{i}_{d_{\mathcal{F}}} \\ \text{explore both branches of } n', \text{ if} & \hat{i}'_{d_{\mathcal{F}}} = \hat{i}_{d_{\mathcal{F}}} \\ \text{explore the left branch of } n', \text{ if} & \hat{i}'_{d_{\mathcal{F}}} > \hat{i}_{d_{\mathcal{F}}} \end{cases} \tag{10}$$

allows for deciding, which branch needs further examination.

To avoid re-exploration of a subtree , the prefix $\hat{i}_{d_{\mathcal{F}}}$ of $\mathbf{n}$ is stored in an *explored-list* after examination of its complete subtree. The first element of the list is the prefix of the cell that contains the center of the ROI, followed by a maximum of 6 outlier prefixes. The index of each examined node is compared to all list elements. In case of a match, the search on the subtree terminates. Otherwise, the distance of the mean point of a node to the center $\mathbf{q}$ of the ROI is checked. If it passes, the point is collected.

In the worst case, all starting points are located in the top four levels of the tree, while the searched points are based in the leaf cells. Due to two instead of eight childs per node, the maximum tree depth is

$$3d_{\max} = \log_2 8^{d_{\max}}. \tag{11}$$

The limiting complexity is approximated by

$$O(32 + 2^{3d_{\max}-1}\log_2(3d_{\max}-4)). \tag{12}$$

Fig. 1(a) illustrates the algorithm on a two-dimensional example. Fig. 1(b) shows the results on a quadtree (which is the 2D version of an octree), while Fig. 1(c) focuses on the balanced binary tree.

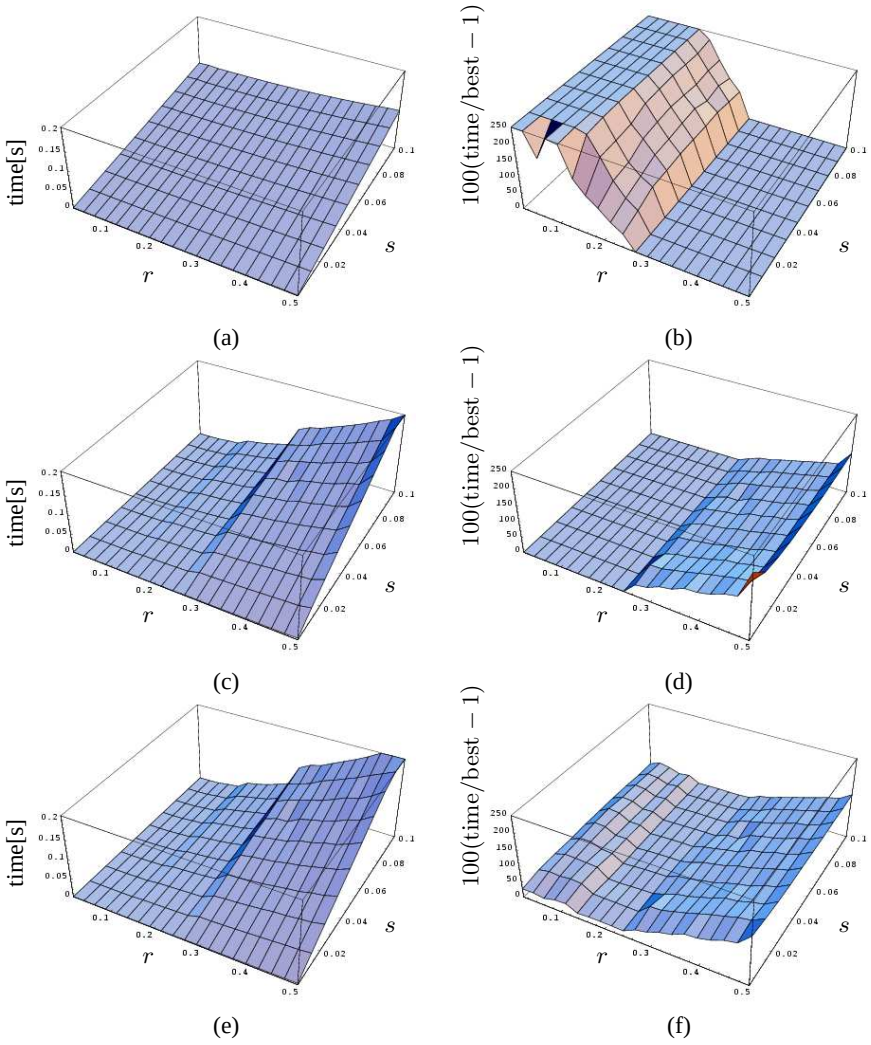

Fig. 2. The plots in the left column show absolute processing times, while the right column addresses the normalized processing times. Normalized times are divided by the measurement of the fastest algorithm regarding a pair of radius and saturation. The plots (a) and (b) are the result of the naive approach, (c) and (d) the result of the octree, while (e) and (f) are the result of the balanced binary tree.

4 Experimental Evaluation

In this section, the processing speed for both the octree and the balanced binary tree is compared to the results of a naive approach. A comparison to ICP approaches is not appropriate since ICP approaches are limited to a search of single closest points.

The naive approach is implemented as a sequential pointered list. The use of an index is not necessary. Since no spatial organization exists, each list node is compared to the center of the ROI. Therefore, the complexity is N, as N is the number of nodes.

The input of every representation is an identical point cloud, i.e. a uniformly distributed set of points within a cube of unitary edge size. Note this is only a theoretical worst case example. Taking realistic data, such as a sampled surface consisting of several objects and background leads to much better but data-dependant results. Nevertheless, the mean processing times achieved with this method embedded in a 3D-scene interpretation approach are depicted in [7].

The term *saturation*

$$s = N/8^{d_{\max}} \tag{13}$$

describes the occupancy of the cube, where the divisor $8^{d_{\max}}$ is the maximum number of possible indices (that equals the maximum number of atomic cells). Accordingly, a saturation of $s = 1$ symbolizes the maximum number of 3D-points which can be encoded with an index of $d_{\max}$ digits. For the tests we used a subdivision space of depth $d_{\max} = 8$. It allows for a maximum number of about $16.8 \cdot 10^6$ entries.

For the experimental evaluation of the processing costs, the radius $r \in [0.01, 0.5]$ of the ROI $\mathcal{F}(r, \mathbf{q})$ was modified in 20 steps and the saturation $s \in [0.001, 0.1]$ of the representations in 10 steps. Each combination of r and s was repeated 500 times. That leads to a total of $20 \times 10 \times 500 = 10^5$ tests. The processing performance at an Intel Xeon 1.7GHz with 1GB RAM are depicted in Fig. 2.

Due to better visualization of the comparison results, we define a normalized notation of processing times. There, each pair (r, s) depicts the processing time of an approach with respect to the processing time of the fastest representation. The result is performed in percent and shifted to the origin, i.e. a value of 0 characterizes an approach as being faster than the others.

As mentioned above, the absolute processing times for the naive approach are a function of the saturation (see Fig. 2(a)), since it defines the length of the list. Furthermore, the processed operation is an expansive floating-point calculation of a distance in contrast to a simple integer comparison of an index.

For both tree representations the absolute processing costs arise either for the parameter r and s. This is caused by an increasing number of nodes which have to be examined in both cases. In the worst case, that is a ROI of similar size to the represented space all nodes have to be processed. Consecutively, additional costs for navigation and calculation of the index occur.

These contemplations are of minor importance, since the algorithm is designed for scene interpretation. There, focused objects are much smaller than the complete scene. A direct comparison of the normalized plots (Fig. 2(b), (d) and (f)) emphasizes the advantages of the search algorithm. It shows the dependency of the method to the radius r only. As desired small radii less than 30% of the edge size of the represented space, meet the expectations. The tree representations achieve significantly faster processing for this parameterization, while the naive approach exceeds the range of the plot.

Due to lower tree-depth and less operations during navigation, the octree outperforms the balanced binary tree with around 10% to 20% faster access. On the other

hand, the octree needs $\sum_{i=0}^{d_{\max}-1} 8^i$ additional nodes for the top of the tree. It has to be considered, therefore, whether processing time or memory costs matter the most.

5 Conclusion

In this paper a novel combination of an encoding of 3D-points and a region search algorithm was introduced. The used input is a data stream as it is typical for sampling a scene with a 3D-sensing device.

It has been shown that the region search algorithm is feasible for an octree representation and a binary balanced tree representation. Both structures were selected to provide the necessary update ability. The search algorithm can be adapted to each tree representation and allows the access to arbitrary sized spherical subsets. The method shows the best performance for small and medium regions of interest. This is of special importance since the algorithm is embedded in a scene interpretation approach that needs to focus differently sized objects with respect to a large environment.

Comparing the octree to the balanced tree representation leads to the conclusion, that the octree is the best choice if memory costs are irrelevant. Keeping in mind that 3D sensing tends to produce very large data sets, the balanced binary tree is a very significant structure even though processing costs are higher.

References

1. Johnson, A.E., Hebert, M.: Using spin images for effi cient object recognition in cluttered 3D scenes. IEEE Transactions on Pattern Analysis and Machine Intelligence **21** (1999) 433–449
2. Yamany, S.M., Farag, A.A.: Surface signatures: An orientation indepentent free-form surface representation scheme for the purpose of objects registration and matching. IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002) 1105–1120
3. Samet, H.: Applications of Spatial Data Structures: computer graphics, image processing, and GIS. Addison-Wesley Publishing Company (1990)
4. Gaede, V., Günther, O.: Multidimensional access methods. ACM Computing Surveys **30** (1998) 170–231
5. Sagawa, R., Masuda, T., Ikeuchi, K.: Effective nearest neighbor search for aligning and merging range images. International Conference on 3-D Digital Imaging and Modelling (2003)
6. Bodenmüller, T., Hirzinger, G.: Online surface reconstruction from unorganized 3D-points for the DLR hand-guided scanner system. 2nd International Symposium on 3D Data Processing, Visualization, and Transmission 3DPVT'04 (2004)
7. Wahl, E., Hirzinger, G.: Local point cloud analysis for rapid scene interpretation. DAGM 2005 (27th Annual meeting of the German Association for Pattern Recognition) (2005)
8. Strobl, K.H., Sepp, W., Wahl, E., Bodenmüller, T., Suppa, M., Seara, J.F., Hirzinger, G.: The DLR multisensory hand-guided device: The laser stripe profi ler. International Conference on Robotis & Automation (ICRA) (2004)
9. Suppa, M., Hirzinger, G.: A novel system approach to multisensory data acquisition. The 8th Conference on Intelligent Autonomous Systems IAS-8 (2004)
10. Adel'son-Vel'skiĭ, G.M., Landis, Y.M.: An algorithm for the organization of information. English translation: Soviet. Math. Dokl. 3 (1962) 1259–1263