Using a linear octree to identify empty space in indoor point clouds for 3D pathfinding

Tom Broersen
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
t.broersen-1@student.tudelft.nl

Florian W. Fichtner
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
f.w.fichtner@student.tudelft.nl

Erik J. Heeres
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
e.j.heeres@student.tudelft.nl

Ivo de Liefde
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
i.deliefde@student.tudelft.nl

Olivier B. P. M. Rodenberg
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
o.b.p.m.rodenberg@student.tudelft.nl

Edward Verbree
Delft University of Technology
Julianalaan 134
Delft, the Netherlands
e.verbree@tudelft.nl

Robert Voûte
Delft University of Technology / CGI
Julianalaan 134
Delft, the Netherlands
r.voute@tudelft.nl

Abstract

Indoor pathfinding and routing need fast ways to define connected navigable space, which represents usable paths. This is important because the interior space in buildings changes and often does not follow the architectural design. A workflow is presented that uses a linear octree to segment the space contained in an indoor point cloud, and subsequently derives the structured and connected empty space. Binary locational codes are efficiently generated for every point, which implicitly contain the linear octree structure. The workflow successfully generated the linear octree and derived the empty space. A shortest path algorithm showed the potential of using a linear octree for space segmentation in indoor point clouds, and deriving the structured and connected empty space. The workflow is fast and automated, enriches point clouds with a structure that can be understood by computers and retains all detail of the raw data set.

Keywords: point cloud, linear octree, connected empty space, pathfinding, locational codes, indoor

1 Introduction

Point clouds of interior spaces are useful for many applications, such as automatic generation of indoor models [9] and object recognition [7]. The focus for such applications is mainly on identification of the boundaries of space or the objects inside it, instead of directly on the empty, pointless and usable space itself. Identification of empty space, also called free space [6], can serve a multitude of purposes. It can be used for (3D) pathfinding, estimating building volume and available storage space, and for fitting objects through narrow spaces. Two-dimensional pathfinding purposes require at least the availability of a model with navigable paths, which could be constructed from a 2D floor plan. However, 3D pathfinding purposes require the availability of a 3D explorative representation of empty, navigable space [5, 13]. Furthermore, this empty space needs to be structured; topological relations are needed for pathfinding algorithms to navigate through the identified and connected empty space. One option to do so is by creating an octree data structure, which has been used to efficiently identify and organize empty space before [6].

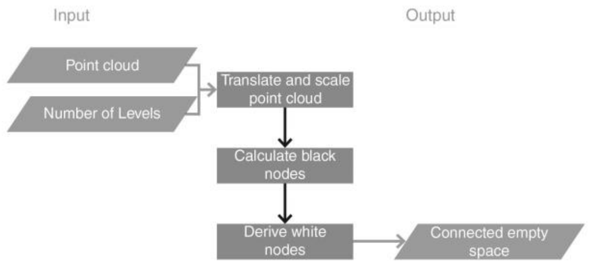
Octrees are also a common approach to segment and structure point clouds [10, 15, 16]. They are used by applications such as Potree [11], which visualises point clouds by making use of the octree structure to select visible nodes from a certain viewpoint. Octrees are based on the recursive and non-uniform subdivision of space. They provide an efficient way of indexing the point cloud and performing neighbour finding operations [8], and result in a hierarchical structure. Octrees can be constructed such that, as long as an octant contains a certain number of points, the octant is again split into eight equally sized octants. This process can be continued either until all octants contain a maximum specified number of points, or until a predefined resolution of space segmentation is reached. From such an octree, the empty space can be efficiently derived.

In this paper a workflow is presented that segments an indoor point cloud by generating a linear octree, and subsequently uses this octree to derive the structured and connected empty space. The performance and scalability of the workflow is analysed, and a 3D pathfinding algorithm is used to present a use case for the identification of structured and connected empty space.

2 Methodology

The three dimensional space contained by the point cloud is segmented and structured using a linear octree based on [2]. Figure 1 shows the method's workflow.

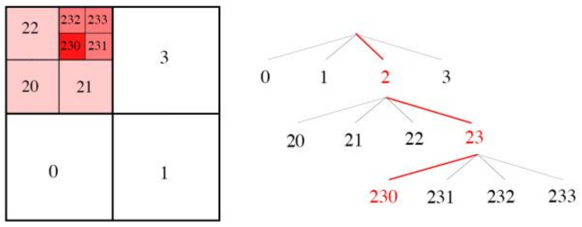
Figure 1: Workflow of presented method.



2.1 Translation and scaling

To allow for efficient octree generation, the native coordinate system of the point cloud needs to be converted to a local Cartesian coordinate system ranging between 0 at the origin, to 2^{nmax} at the axis extremes. The parameter *nmax* gives the desired number of levels in the octree, the octree *depth*, which depends on the required resolution of the space segmentation. First, a translation is applied which sets the point of origin to (0, 0, 0). The offset of the point cloud's bounding box is determined with respect to the desired origin point (0, 0, 0). Every point in the point cloud is then corrected for this offset. Secondly, the point cloud is scaled to set the axis extremes to 2^{nmax}. A ratio is calculated between every dimension of the bounding box and 2^{nmax}. The largest of these ratios is taken as scaling factor, which is applied to every point. To create a cubic bounding box, empty space is added to the end of any axis with an axis extreme below 2^{nmax}.

Figure 2: Two dimensional example of octree enumeration and locational codes. Binary numbers are converted to decimal integers for readability.



2.2 Linear octree generation

Every black leaf node in the octree is assigned a unique key, and the octree structure can be derived from these unique keys. These keys are locational codes [1, 2], which are based on the following principle (Figure 2):

Imagine a cube that is split by three split planes located in the middle of, and perpendicular to, its (x, y, z) axes. This split action results in the formation of eight smaller equally sized octants. It is then possible to encode with three bits in which of these octants a certain point is located. Along every (x, y, z) axis of the cube, this point can be at one of the two sides of the corresponding split plane, which can be indicated by either the digit 0 (false) or 1 (true). It is thus possible to specify for this point in which octant it is located by a combination of three bits; x [0 or 1], y [0 or 1], and z [0 or 1].

This principle is used to generate a locational code for every point in the point cloud:

1. By truncating the point's coordinates to integers, the point is snapped to the minimum vertex of the black leaf node inside which it is located [1]. This is always correct due to the translation and scaling applied to the point cloud.
2. The truncated coordinates of the point are converted into separate binary numbers for the (x, y, z) dimensions. The combined digits on the *i*-th position in these three binary numbers give the location of the point in its containing octant at depth *i* in the octree, which follows from the principle above.
3. The three binary numbers are then iterated through, from left to right, thereby interleaving the digits to form the locational code. The iteration continues until the desired octree depth *nmax* is reached. Any binary number shorter than *nmax*, is simply lengthened by concatenating the digit 0 until the *nmax* length is reached.

By following this procedure for every point in the point cloud, a set of locational codes is obtained. The set of the octree's black leaf nodes can be formed by taking all unique locational codes of the points, thus removing the duplicate locational codes from the set of points. The entire octree structure is implicitly stored in the set of black leaf nodes, and can be reconstructed from it. The method ensures that there are no black leaf nodes in the octree at levels below the maximum octree depth *nmax*, thus the octree always reaches maximum resolution around points.

2.3 Identification of empty space

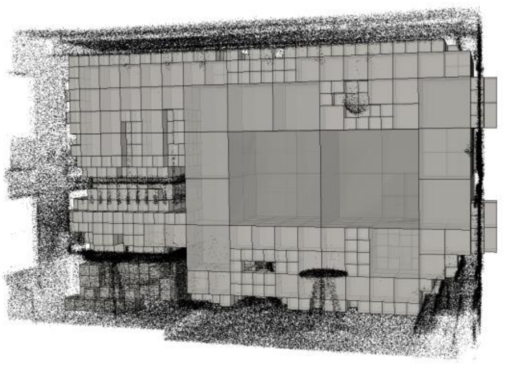
The linear octree is implicitly stored in the set of black leaf nodes, but the white leaf nodes, which contain the empty space, still have to be derived. Unlike black leaf nodes, white leaf nodes can occur in every level of the octree. To identify all white nodes at a certain level *n* in the octree, the following steps are taken:

1. Generate the set of grey nodes at level *n-1*
2. Generate the set of black nodes at level *n*
3. Concatenate to all locational codes of the grey nodes at level *n-1*, one by one the eight 3-bit length binary numbers of the nodes formed by division of this grey node, thus generating the set of all potential white and black nodes at level *n*.
4. Test for the entire set of potential white and black nodes at level *n*, whether these nodes are present in the set of black nodes at level *n*. If they are not

present in this set, then this means the node is a white leaf node, and thus represents empty space.

These steps have to be performed for every level in the octree. Generating the sets of grey and black nodes at a certain level is done by removing [(*nmax-n*) * 3] digits from the locational code of every black leaf node until the required level is reached. Duplicates are removed from the resulting sets. To find all white leaf nodes at level 0, simply compare the set of black nodes at level 0 to the set of eight 3-bit length binary numbers of the nodes formed by division of the root node.

Figure 3: Visualization of structured and connected empty space (grey cubes) in a vertical slice through the point cloud. The points (black) represent the building's walls, floor, ceiling and furniture.



2.4 Determining node geometry and minimum vertex

All octants in this octree are cubic, thus all twelve edges of a node are equal in length. This length is dependent on the level of the node in the octree, and is halved for every deeper level in the tree. Since the minimum edge length of the leaf nodes equals 1, the size of the nodes at level n is equal to 2^{nmax-n}.

To find any node's minimum vertex, or point of origin, the node's locational code is converted back to local coordinates. This is done by iterating over the digits in the locational code from right to left, and accordingly recreating the original separate (x, y, z) binary numbers. At the end of the iteration, these binary numbers can be converted to decimal numbers obtain the (x, y, z) coordinates of the minimum vertex.

3 Implementation

The workflow was implemented using Python and PostgreSQL. The ZEB1 hand-held laser-scanner [3] was used to obtain an indoor point cloud of a building of approximately 13 by 20 m. The implementation was tested on a HP laptop running Windows 10 with Intel(R) Core(™) i7-4700MQ CPU at 2.40 GHz and 8.00 GB RAM.

The A* algorithm is used to compute the shortest path between two points in the octree, which serves as a use case for the structured and connected empty space. Neighbors are identified after [14], thereby utilizing the hierarchical structure of the linear octree. Nodes are regarded neighbors if they share a common face.

The implemented workflow successfully generated the linear octree, and derived the empty space from it. A vertical slice through the point cloud and derived empty space is shown in Figure 3. Due to the octree structure, the empty space is structured into large cubes in the center of the building, and smaller cubes near points. The structured and connected empty space was successfully used by the pathfinding algorithm to compute the shortest route between two points located in the empty space (Figure 4).

4 Performance and Scalability

The performance and scalability of the proposed workflow was evaluated for different octree depths (Table 1), and for different point cloud sizes (Table 2). Different point cloud sizes were obtained by changing point density. To focus only on the proposed workflow, the time spent writing points to PostgreSQL database was not taken into account.

The total workflow runtime required for octrees with maximum depth of 6, 7 and 8 levels is shown for a point cloud of 2.3 million points (Table 1). Increasing the octree depth from 6 to 7 levels, thus essentially doubling the resolution of space segmentation, causes only an 18% increase in total runtime. Increasing the octree depth from 6 to 8 levels, quadrupling the resolution of space segmentation, causes an increase in total runtime of only 36%. This is caused by the fact that the same number of points are processed, while the number of iterations needed to form the locational code of the points increases from 6 to 7 or 8 only. Thus the time spent on calculations per point increases only by a small amount. Similarly, the number of octree levels processed for identification of white nodes, or empty space, increases from 6 to 7 or 8 only. In essence this indicates that the implemented workflow scales well with increasing space segmentation resolution.

Table 1: Workflow runtime for a point cloud of 2.3 million points with different octree depths.

Octree depth in levels	6	7	8
Workflow runtime in seconds	11	13	15
Workflow runtime relative to 6 levels in percent	100	118	136

Table 2: Workflow runtime for a point cloud with octree depth of 8 levels, for different point cloud sizes.

Point cloud size in million points	2.3	4.6	9.2
Workflow runtime in seconds	15	32	54
Workflow runtime relative to 2.3 million points in percent	100	213	360

The total workflow runtime required for differently sized point clouds, is shown for a constant octree depth of 8 levels (Table 2). An increase in the number of points has a pronounced impact on performance. Doubling the point cloud size increased total runtime by 213%, while quadrupling the point cloud size increased total runtime by 360%. These increases in runtime are caused by the fact that there are simply many more points, and thus many more calculations to perform to obtain the locational codes. However, the derivation of white nodes scales well with increasing point cloud size, and runtime for a point cloud of 9.2 million points is still well below one minute. Using a compiled language such as C++ may further speed up the workflow.

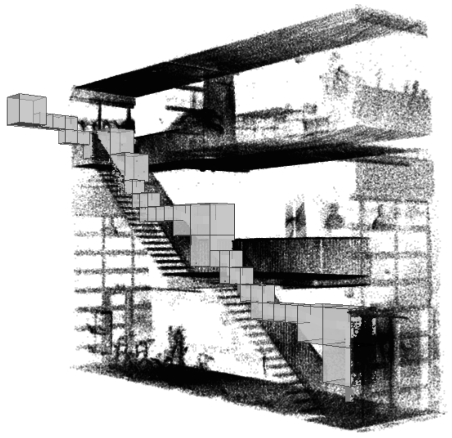
5 Conclusion

A workflow was presented that uses a linear octree to segment the space contained in an indoor point cloud, and subsequently derives the structured and connected empty space. The workflow was implemented using Python and PostgreSQL, and successfully generated the linear octree and derived the empty space.

The performance and scalability of the workflow were analysed for different octree depths, or resolution of space segmentation, and for different point cloud sizes. The workflow was found to scale well, especially for increasing octree depth. The time required for the workflow to generate a linear octree and derive the empty space for a point cloud of 9.2 million points, and octree depth of 8 levels, is only 54 seconds. Thus, the presented methodology performs and scales well, and can especially be applicable if high resolution space segmentation is required. The procedure is fast and automated, enriches point clouds with a structure that can be understood by computers, while keeping all detail of the raw data set.

A 3D pathfinding algorithm was successfully used to compute the shortest path between two points. This use case indicates the potential of using a linear octree for space segmentation in indoor point clouds, and deriving the structured and connected empty space. The high performance of the workflow is important for indoor pathfinding in a changing environment.

Figure 4: Visualization of a shortest route (grey cubes) through the structured and connected empty space in the point cloud.



6 Future research

Future research could aim at connecting the presented method to the "outdoor" coordinate system (world coordinates), thereby connecting indoor and outdoor pathfinding. The workflow could be adapted for the real time identification of empty space with an updateable structure. Furthermore, the derivation of empty space could be beneficial for noise filtering in point clouds through the identification of isolated points in empty space. The locational codes could also be used to improve the database performance [4, 12]. The current implementation has been validated using visual inspection. However, future research could create metrics for validating the results more thoroughly.

Authors of this paper are currently working on identifying the impact of octree depth and connectivity on the performance of pathfinding using an octree structure. Furthermore, the octree data structure is being used to subdivide the 3D empty space for pathfinding across multiple floor levels.

References

[1] Frisken, S. F., and Perry, R. N. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools*, 7(3): 1-11, 2002.

[2] Gargantini, I. Linear octrees for fast processing of three-dimensional objects. *Computer graphics and Image processing*, 20(4): 365-374, 1982.

[3] Geoslam. ZEB1 hand-held laser-scanner. URL: http://geoslam.com/products/#zeb1, 2016.

[4] Laurini, R., Thompson, D. Fundamentals of spatial information systems. Vol. 37. Academic press, 1992.

[5] Nekiber, S. Rich point clouds in virtual globes – A new paradigm in city modeling? *Computers, Environment and Urban Systems,* 34(6): 508–517, 2010.

[6] Payeur, P. A computational technique for free space localization in 3-D multiresolution probabilistic environment models. *IEEE Transactions on Instrumentation and Measurement,* 55(5):1734-1746, 2006.

[7] Rusu, R., Marton, Z., Blodow, N., Holzbach, A., and Beetz, M. Model-based and learned semantic object labelling in 3D point cloud maps of kitchen environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3601–3608, 2009.

[8] Samet, H. Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing,* 46(3):367–386, 1989.

[9] Sanchez, V. and Zakhor, A. Planar 3d modeling of building interiors from point cloud data. In *19th IEEE International Conference on Image Processing (ICIP),* pages 1777–1780, 2012.

[10] Schön, B., Mosa, A. S. M., Laefer, D. F., and Bertolotto, M. Octree-based indexing for 3D point clouds within an oracle spatial DBMS. *Computers & Geosciences*, 51:430–438, 2013.

[11] Schütz, M. Potree. URL: http://potree.org, 2016.

[12] Van Oosterom, P. Spatial Access Methods. In *Geographical Information Systems Principles, Technical Issues, Management Issues, and Applications* (edited by Longley, Goodchild, Maguire en Rhind), Wiley pages 385-400 (vol.1), 1999

[13] Verbree, E. & Oosterom, P.J.M. van. Explorative point clouds maps for immediate use and analysis. *Presentation at European LiDAR Mapping FORUM (ELMF)*, Amsterdam, The Netherlands, December 2014.

[14] Vörös, J. A strategy for repetitive neighbor finding in octree representations. *Image & Vision Computing*, 18(14): 1085–1091, 2000.

[15] Wang, M. and Tseng, Y. Incremental segmentation of lidar point clouds with an octree-structured voxel space. *The Photogrammetric Record*, 26(133): 32–57, 2011.

[16] Zhou, K., Gong, M., Huang, X., and Guo, B. Data-Parallel octrees for surface reconstruction. *IEEE Trans. Visualization & Computer Graphics*, 17(5): 669–681, 2011.