

An Octree- and A Graph-Based Approach to Support Location Aware Navigation Services

Srihari Narasimhan
IPVS
Universität Stuttgart
Stuttgart, Germany

Ralf-Peter Mundani, Hans-Joachim Bungartz
Institut für Informatik
Technische Universität München
Garching bei München, Germany

Abstract - *Over the last decade, there is an increasing trend in the use of more and more mobile navigation devices. In order to make mobile navigation more intelligent, issues such as location and context awareness need to be addressed to offer location-based navigation services. For example, one might think of a scenario of a large commercial center where a customer with some mobile device in hand wishes to go to a drugstore. The system should efficiently identify the current location of the customer, a list of drugstores located nearby, the easiest path to the next drugstore and provide routing and destination suggestions to the customer. In this paper, assuming that the mobile device gives us the co-ordinates of the current location of the customer, we present a graph and an octree-based approach to identify the location of the customer, identify a list of destinations located in the neighborhood and determine the best possible destination and the respective path to the destination.*

Keywords: octree, location awareness, graph algorithms, navigation and routing services

1 Introduction

Today, mobile devices are widely used as navigation systems, not just on the streets (using GPS), but also indoors (using sensors, WLAN, . . .). In this paper, we present an approach to use these positioning devices to provide an intelligent navigation system in a 3D scenario.

At first, it is important to identify the possible paths where customer movement exists and the possible destinations in such a scenario. For this purpose, the architectural model is scanned to obtain a list of all possible paths and these paths are stored in as a graph. The edges of the graph represent the paths obtained from the geometric model. A list of destinations, along with its location in the graph, is also determined. The nodes from the resulting graph are stored in hierarchical octree structure. Now, assuming that the mobile device delivers the current co-ordinates (x, y, z) of its location, the system identifies the closest node to the current position and thereby also identifies the location of the device on the graph. Using octrees, it is also possible to determine the closest destination and the corresponding path to the destination. Furthermore, with the use of pedestrian simulation to simulate the status of the waiting times at various destinations, the system can identify the destination with the least waiting time and an optimal path to the destination.

In the next section, we present the octree model that is used to store the list of nodes in an octree structure. Following this, we show how to determine the position of the customer from the given co-ordinates and the heuristics used is this process. We then present the method to identify a list of

neighboring destinations and the routes to the destinations thereby presenting possible itineraries to the customer. We conclude the paper with presenting the possibilities of integrating the pedestrian simulation together with the octree and graph model, in order to identify the destination and path to it, considering the waiting times at the destination and the congestion along the paths.

2 Octree-Based Modeling

In order to generate an octree structure, a graph containing a list of nodes and destinations of the scenario must first be extracted. The procedure of extracting the graph from the geometric model and the generation of an octree structure is explained as follows.

2.1 Extraction of a Graph from a CAD Model

To extract the graph, the CAD model of the building is parsed through using the Pathscan [1] program to automatically derive a graph of paths and to define the destinations across the building. Pathscan is a flexible tool that reads a CAD model of an architectural building, generates slices of the model in upright direction, identifies all floors where a pedestrian can move and exports the paths on to a graph. The tool also allows the user to define rooms and their properties and to make changes to the final graph, too. Once all adjustments are made, the tool exports the data into an XML file that can be easily parsed by other programs. The resulting data shows the co-ordinates of each node, the edges connecting them, the type of edges (staircase, pathway, private, etc.), the rooms, and the nodes that lie in the rooms. Figure 1 depicts a snapshot of the output generated by Pathscan tool that is used to define the graph and the rooms.

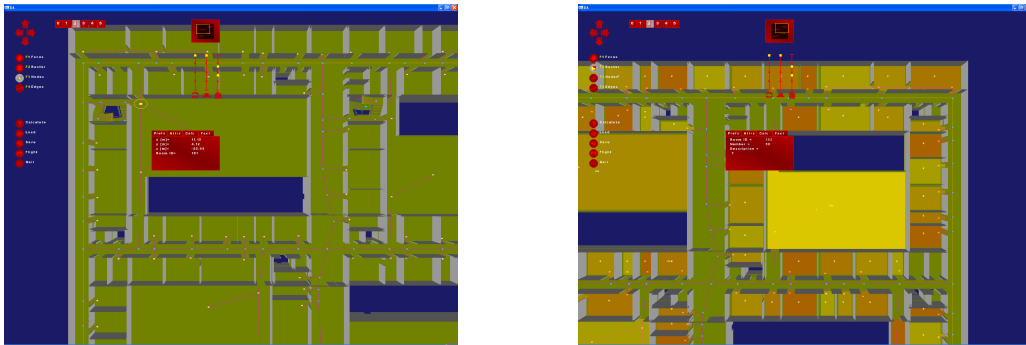


Figure 1: Pathscan tool used to create and modify the graph (left), and to define the rooms and their specifications (right).

2.2 Octree Model to Store the Graph

Octrees are widely used data structures for hierarchically storing data such as a 3D geometry. The octree structure resembles a cube, where the cube is equally divided in all the three dimensions to result in eight smaller cubes or otherwise called as voxels. Each voxel is identified by a unique identifier with the use of the Morton index. The voxels are numbered from '0' to '7' in a specific order¹. The octree

¹The order itself is not relevant, but it has to be consistent among all nodes

is built recursively as long as one voxel contains more than one node. In our scenario, the nodes that are spread in the 3D space are stored in the octree structure. The octree is built recursively until exactly one node is stored in each voxel. The algorithm and its analysis to build such an octree is explained in [2].

The co-ordinates, once normalised on the range of $[-1, 1]$, will belong to one of the 8 octants. The $+$ or the $-$ index of the co-ordinates denote the direction. Each node of the graph is then inserted into the tree and the position of the node in the octree is decided by the direction of the node co-ordinates. Figure 2 demonstrates the building of a quadtree for nodes spread in a 2D space. A similar approach is made to build an octree. Figure 3 shows a snapshot of the resulting octree model for the graph structure used in our scenario.

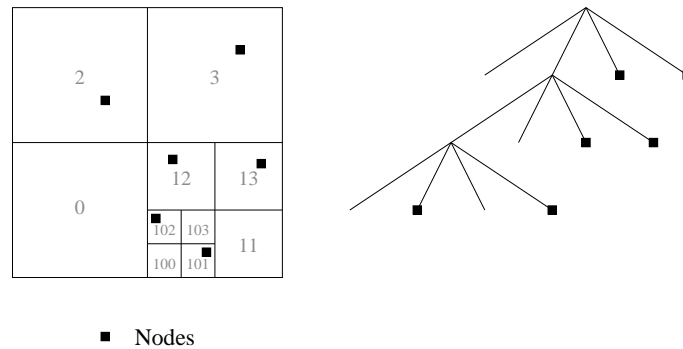


Figure 2: Illustration of a quadtree where each quadrant contains utmost one node.

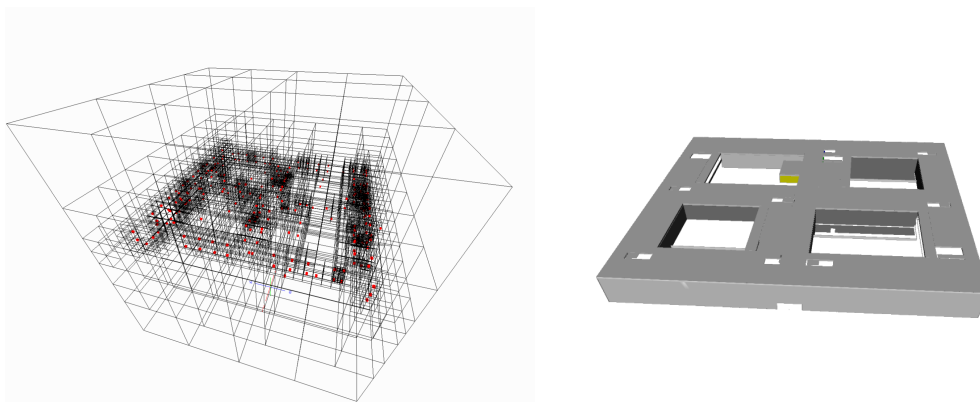


Figure 3: Octree model (left) for the list of nodes of a sample graph extracted from a CAD-Model (right).

As one can see, octrees have the natural property of storing 3D geometries in a hierarchical way, which other tree structures do not possess. Furthermore, an octree structure helps in easy identification of the surrounding neighbors efficiently, which will be discussed in the next section. A similar approach of using octrees for arranging finite element discretization in a hierarchical way is discussed in [3]. Also, the use of octrees to represent geometric data in the field of structural engineering can be seen in [4]

3 Location Identification of the Mobile Device

Now that an octree structure where each voxel contains utmost one node is generated, one can try to locate the voxel or the neighboring node from the location of the positioning device. In simple words, the octant to which the customer belongs to is determined and the node that is already present in the octant is identified.

Now, the co-ordinates from the current location of the positioning device (henceforth denoted as P) are obtained. Just as in the hierarchical approach described in the previous section, the tree is parsed to determine the octant to which the co-ordinates P belong. At one point, the end node that has no children is reached.

Case 1: If P belongs to an octant where a node already exists, the current node is returned. This implies that P lies in the voxel of the node that is returned (Fig.4).

Case 2: If P belongs to an octant where no node exists, the neighboring octants in the same level must be searched. Therefore, the parent of the current node is searched and the children containing a node are returned (Fig.5). The resulting nodes are compared with P to determine the closest node.

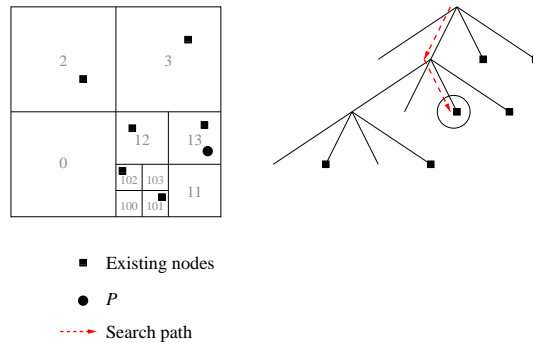


Figure 4: Tree search when P lies in a quadrant where a node exists.

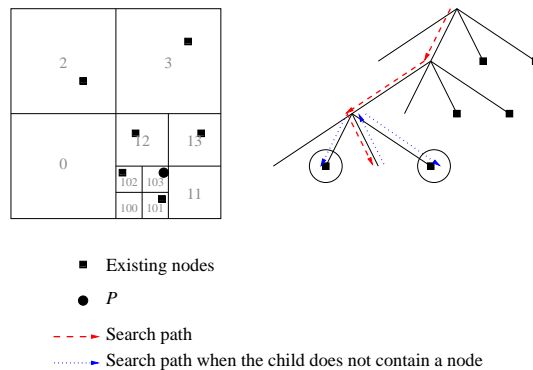


Figure 5: Tree search when P lies in a quadrant where a node does not exist. Therefore, the neighboring nodes of the same level are searched.

However, it must be noted that the nodes stored in the octree structure are not necessarily positioned in the center of the voxel. This means that the position P and the node determined from the above explained steps are not necessarily the closest neighbors. It is therefore necessary to parse the neighboring octants, determine the nodes on those octants and find the node closest to position P . To determine the boundary condition for searching the octree, a sphere is drawn with P as origin and the radius is set to be the distance between P and the closest node obtained so far. Since the octree structure resembles a cube, the resulting sphere is bounded by a cube. A search algorithm is much easier to implement when the search boundary resembles a cube. Now, the octree is parsed through recursively once again and the octants that lie within the search boundary and their respective nodes are determined. The closest neighbor is then determined by comparing the distances between P and the nodes. Figure 6 demonstrates the method to determine the closest neighbor of P .

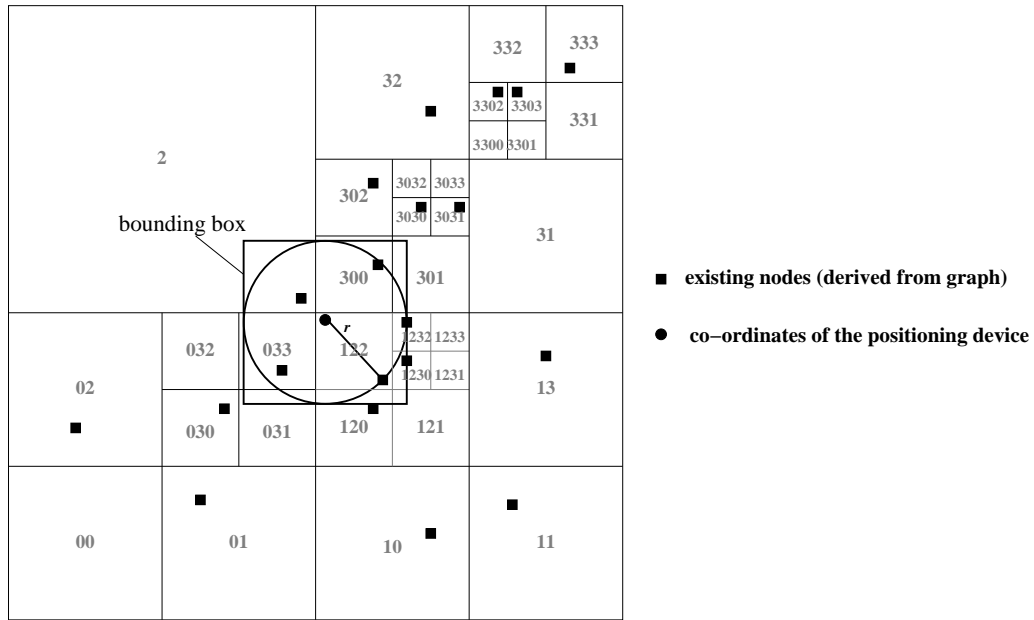


Figure 6: Method to determine the closest neighbor of P in a quadtree example.

In the example (Fig. 6), a quadtree containing certain number of nodes is generated. On searching the position of P , the octant with Morton index (122) is identified. In order to find the nodes that are located closer than the identified node, a circle with P as center and r (distance between P and the identified node) as radius is drawn. An enclosing square is set as the boundary box. Now, on parsing the quadtree to determine the octants and their respective nodes (if a node exist), the octants with Morton indices (033,120,122,1230,1232,2,300) are identified. Octants with Morton indices (031,121,301) are omitted since they do not contain any node. By comparing the distances between P and each of the 7 derived nodes, the closest node is found in the octant with Morton index (2).

The search for the closest neighbor can also be done by comparing the geometric distance of all existing nodes. In this case the search is linear with a complexity of $\mathcal{O}(n)$. Whereas, in case of using hierarchical data structures such as the octrees, the complexity of the node search is $\mathcal{O}(\log(n))$. Since the neighboring nodes are also searched, the search is repeated until the nearest neighbor is found. The complexity therefore is $\mathcal{O}(m * \log(n))$ where m lies in the interval $[1, n]$. In the best case, the

complexity turns out to be $\mathcal{O}(\log(n))$ and in the worst case, the complexity is $\mathcal{O}(n * \log(n))$ which is worse than a linear search. In normal cases, especially for large numbers of n as in our scenario, the complexity tends to be logarithmic. Similar traversing or neighbor searching algorithms for quadtrees and octrees have been proposed in [5, 6].

4 Navigation Using the Octree Model

So far, methods to efficiently locate the positioning device were discussed. Now let us consider a scenario where a customer wishes to go to a drugstore and his current position is P . The graph data extracted from the CAD-Model contains the list of destinations and its properties. From the list, it is possible to extract all nodes associated with a drugstore. One method to find the closest drugstore is to make a linear search through the list. Since the location of P on the graph is known, the distance (here the shortest path on the graph) to all drugstores located on the graph can be measured and the closest drugstore can be detected. The linear search has a complexity of $\mathcal{O}(n)$. However, when the graph covers the entire city and there are thousands of drugstores spread across the city, it would be time consuming to search the entire list to find the closest drugstore. Alternatively, the octree-based model can be again used here to determine the drugstores located in the neighborhood and to identify the closest drugstore. Similar to the previous sections, an octree structure is generated that contains the nodes associated with a drugstore. The node closest to P is then identified. However, it must again be noted that the closest neighbor does not consider the length of the path to the destination. Therefore, the search radius is manipulated such that the neighboring octants are also searched and the closest destination, in terms of the path length, is determined. Since only a small region of the entire graph is searched, the closest neighbor and a list of immediate neighbors are determined in a short time. Once the desired node is identified, the path to the destination is calculated and the customer is navigated along the path. In the best case situation, the search algorithm has a complexity of $\mathcal{O}(\log(n))$. Since additional neighbors are also searched, the algorithm is repeated until the the desired node is identified. In ideal cases, especially for large numbers of n , the complexity tends to be logarithmic, and is much better than a linear search.

In a test scenario of an existing model (Fig. 7), the node of the position P is determined as explained in Section 3. Now, the list of possible destinations according to the customer's preference are identified and an octree structure is generated. A search is then made within the bounded region, and the closest nodes are identified and sorted according to the path length for the customer to make a choice.

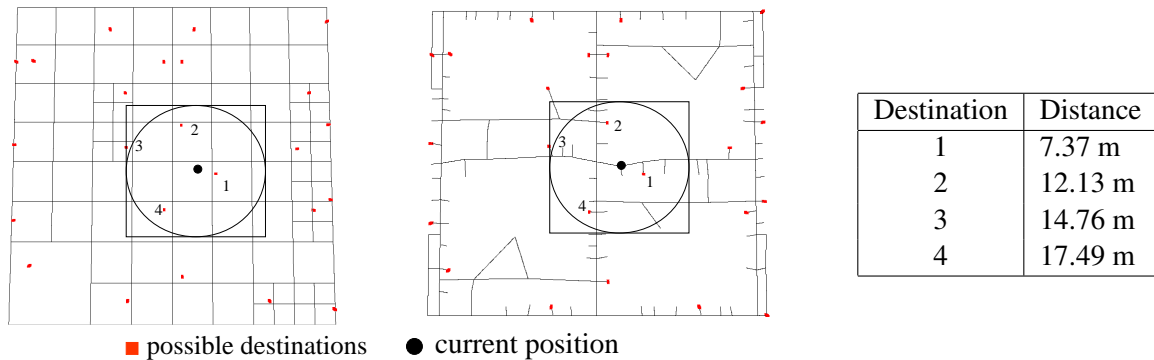


Figure 7: List of possible destinations positioned in the octree (left) and the graph (middle). List of closest destinations sorted according to their distances (right). Apart from that, the

geometric data defines the types of paths in the graph (stairs, escalators, pathway, . . .). Combining these data with the path search algorithms and considering the customer preferences, a customer friendly navigation service can be provided. Furthermore, heuristics on defining the search radius such as, restricting the search radius to the same floor (search in y -axis is avoided), defining the search radius based on customer preferences, etc., have also been worked on. These heuristics are however beyond the scope of this paper and hence not discussed here.

5 Conclusion and Outlook

With the integration of the octree model into the geometric and graph data, it is possible to develop an intelligent pedestrian navigation system. In a very large model covering an entire city, the resulting graph might contain millions of nodes. By storing the data hierarchically in an octree structure, the search is restricted to a very small domain and the efficiency is therefore increased. Work has been done in simulating the pedestrian traffic to identify congestions along the paths and the waiting times at the destinations [7]. By integrating the pedestrian simulation into the navigation system, it is not only possible to determine the closest destination but also to find the destination with the least waiting time and an optimal path to the destination. Suggested itineraries can be made also considering the customer profile (avoid stairs or highly congested paths) such that the customer can decide a plan that suits the customer from the list of suggestions.

References

- [1] Thomas Drexl, *Development of Intelligent Shortest-path Algorithms for Architectural Models Regarding an Info Point System for the FMI in Garching*, Diploma Thesis, Institut für Informatik, Technische Universität München, 2003.
- [2] R.-P. Mundani, *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*, Dissertation, Fakultät für Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, 2006.
- [3] R.-P. Mundani, H.-J. Bungartz, E. Rank, A. Niggel and R. Romberg, *Extending the p -Version of Finite Elements by an Octree-Based Hierarchy*, Proceedings of the 16th International Conference on Domain Decomposition Methods, 2005.
- [4] R.-P. Mundani, H.-J. Bungartz, E. Rank, E. Romberg and A. Niggel, *Efficient Algorithms for Octree-Based Geometric Modeling*, In: Topping, B.H.V. (ed) Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing, Civil-Comp Press, 2003.
- [5] Sarah F. Frisken and Ronald N. Perry, *Simple and Efficient Traversal Methods for Quadtrees and Octrees*, Journal of Graphics Tools, 7(3):1 – 11, 2002.
- [6] P. Bhattacharya, *Efficient Neighbor Finding Algorithms in Quadtree and Octree*, Master Thesis, Indian Institute of Technology, Kanpur, 2001.
- [7] Srihari Narasimhan and Hans-Joachim Bungartz, *Congestion-Aware Optimization of Pedestrian Paths*, In proceedings of the 18th ASIM Symposium in Simulation Technique, Erlangen, Germany, 2005.