High Precision 3D Indoor Routing on Reduced Visibility Graphs

Horst Steuer

Abstract Indoor navigation is becoming a most wanted application especially on the background of the wide availability of powerful personal mobile devices and new methods for indoor positioning. Existing approaches do seldom incorporate that people can move freely through e.g. big halls and are not constrained to specific lanes as vehicles are on road networks. Thereby these approaches can only approximate shortest paths and cannot benefit from possible highly accurate indoor positioning methods. In this chapter we show how the concept of visibility graphs can be applied to indoor routing and how it results in highly accurate shortest paths. We demonstrate how any accurate position can be incorporated in the automatically constructed graph. Furthermore we show how the knowledge that different levels of a building are usually sparsely interconnected can be used to speed up the well-known shortest path algorithm A* by introducing a new heuristic. In experiments we show that our approach needs 29 % less run-time than a standard A*-algorithm.

Keywords Indoor routing · Visibility graph · Heuristic · Shortest path

1 Introduction

Navigation in outdoor environments is probably the most used application of geodata in the end-user market, especially since the upcoming of satellite navigation systems. With the ever growing demand of navigation solutions a big

H. Steuer (✉)
Fachgebiet Geoinformationssysteme, Technische Universität München, Arcisstraße 21, 80333 Munich, Germany
e-mail: steuer@tum.de

diversity of products emerged on the market: from hardware to software, expanding automotive navigation to bicycle as well as pedestrian navigation. Accompanying these developments is an ever growing demand for map data: at first only roads needed to be mapped, later bicycle tracks and pedestrian paths were required as well.

With the development of indoor positioning methods and further establishment of mobile devices like smart phones the demand for new indoor navigation methods arises. Indoor navigation can be especially helpful in unclear, public buildings like airports, train stations or malls.

The task of (indoor) navigation for humans can be divided in several essential sub-tasks:

- Data acquisition, the first step, is the process of generating the necessary map and localization data for all of the following tasks. This can be done either by traditional geodetic methods, by building on existing plans or by using volunteered geographical information like Open Street Map (Haklay and Weber 2008).
- Localization is the process of determining the (start-)location by manual or automatic means. The user can give his/her position by addressing it or the navigation system can determine its own position by means of different sensors like GPS or Wi-Fi-positioning (Evennue and Marx 2006). Localization is a necessary step before the subsequent ones, but can also be repeated during the process later.
- Addressing is necessary for enabling the user to specify his or her destination. This can for example be a point in a coordinate system, a specific room number or the exit of a building.
- Routing is the process of computing an optimal path in regard to some optimality criteria. Usually a length optimal path is sought for but there also exist several requests where a safe path for handicapped people is sought (Karimi and Ghafourian 2010).
- Visualizing the optimal path has to be done in the most useful way for the user. This can be done by visualization in a map (e.g. Hagedorn et al. 2009) or as a set of commands which can be presented visually or acoustically (May et al. 2003).

In this chapter we will focus especially on the routing sub-task. Historically, routing is based on graphs since road networks can easily be described as sets of nodes and edges. Another reason why graph based solutions are so popular is that there exist well-known algorithms like Dijkstra's algorithm (Dijkstra 1959) or A* (Hart et al. 1968) which can compute shortest paths on any given graph with edge weights. Since these graph-based shortest path algorithms can guarantee to find shortest paths, the problem of defining an algorithm to find shortest paths in indoor environments is reduced to generating a useful graph.

In pedestrian navigation one has to observe that pedestrians usually are not constrained to straight paths like cars but can move freely in huge places or big halls. A static graph design which represents every possible pedestrian movement would induce the number of nodes and edges to grow to infinity, and is therefore infeasible.

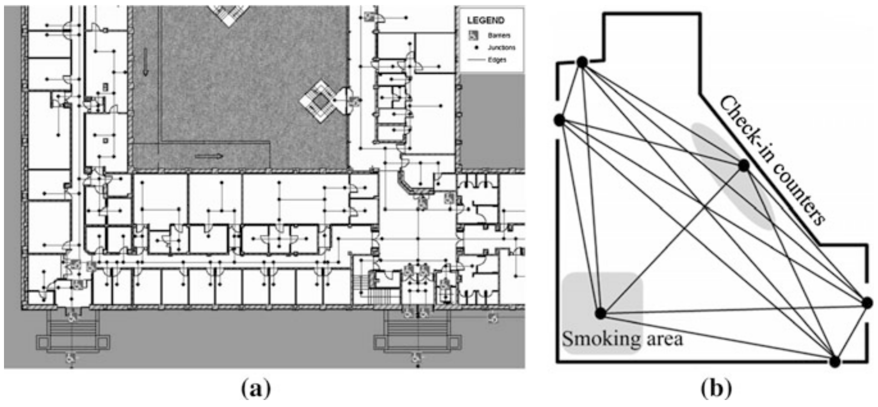

Fig. 1 Examples of graph structures which discretize the available moving navigable space (**a**) an office building where each room is represented by a single node in the graph, corridors and halls are represented by more nodes (Ariza-Villaverde et al. 2010) (**b**) an airport entrance hall with two zones of interest (Goetz and Zipf 2011). Since in both examples only a few points are used as nodes of the graph it is obvious that the given graphs cannot lead to exact shortest paths and can lead to discretization errors when using a self-localization technique

One popular approach to solve this problem is to focus on the topology of rooms and build a graph to represent this topology. In Fig. 1 you can see examples of this approach.

Figure 1a shows a floor of a bureau building where each bureau is represented by one node. Each of these nodes is connected to the corridor by an edge. As can be seen, it is not enough to generate a single node for the corridor but there have to be quite a few nodes in order to compute a useful path. The advantage of this approach is that it results in a relatively small graph. Because of the low number of nodes a shortest path can be computed in very fast run time and the resulting path can be easily described by simple commands like "Move 10 m straight ahead, Turn left, Enter Room 101". Furthermore the addressing sub-task part becomes trivial since for each point of interest an extra node can be added. The disadvantage of this approach is that the space of possible movement is reduced/discretized to these few nodes. Thereby it is not guaranteed to find a geometrically shortest path.

As can be seen in Fig. 1b this becomes even more relevant in less constricted places like an airport entrance hall. Additionally the self-localization poses a problem: The (automatically) determined position has to be projected onto these nodes and the error in initial positioning has to be communicated to the user.

1.1 Contribution

In this chapter we present a different approach to generate a dynamic graph automatically from a floor plan. This approach originates in the field of robotics but has not yet been applied to human indoor navigation to our knowledge. In the next section we describe the structure of the floor plans we need as an input to this approach. In the section "Construction of a Graph" we describe the automatic generation of a so called visibility graph. Using Dijkstra's algorithm and A* with a standard and a custom heuristic we perform several experiments which we present in section "Applying the Dijkstra and A* Algorithms" before we give a conclusion in the last section.

2 Description of Indoor Environments

In the following, we will assume that a single floor of every indoor environment can be modelled or at least be approximated by a polygon with holes. The outer walls of the building, including touching inner walls, make up the outer border of the polygon, while walls which do not touch the outer perimeter and any other obstacle are modelled as holes of this polygon (see Fig. 2). For simplicity reasons we assume that each floor is completely flat, meaning there are no steps or ramps on this floor. Non-flat floors would have to be modelled as a set of more floors in this approach.

Assuming that a human being can approach an obstacle only up to 30 cm measured from his/her barycentre we can add a buffer operation to these polygons. In doing so we also close gaps between obstacles which are too small to walk trough. As a result we obtain a polygon which describes the navigable- or moving space of a person.

In order to connect the different floors of a multi-level building we do not model steps and elevators geometrically but add special nodes to the graph. Especially for the elevators this is a valid approach, since these often have a relatively narrow entrance zone.

3 Construction of a Graph

Given a one-floor polygonal environment we described in the last section we adopt the principle of visibility graphs first introduced in Lozano-Peréz and Wesley (1979). The property of every shortest path in such an environment is "that it is composed of straight lines joining the origin and the destination via a possibly empty sequence of vertices of obstacles" (Lozano-Peréz and Wesley 1979). If we interpret the outside of the polygon as obstacle the visibility graph G(V, E) is defined by a set V of all vertices of all obstacles (inner holes and outer perimeter) and a set of edges E. Each edge e = (v1, v2) of E (v1, v2 element of V) is connected by a straight line which lies completely inside the polygon and does not cross an obstacle. As edge weights we use the euclidean length |(v1, v2)| of the edge.

We can reduce the number of edges in E by removing those which would intersect an obstacle when elongated by an ε (see Fig. 3). This graph G is

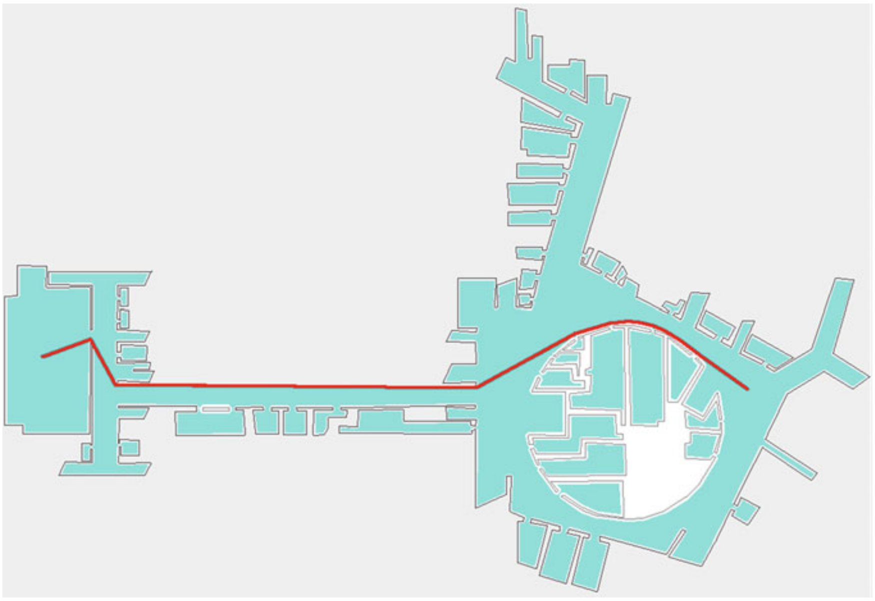

Fig. 2 Floor plan of a mall: the navigable space is described by the *cyan* polygon which we obtained by buffering the input polygon (*grey lines*). Obstacles are modelled as holes in the polygon. The *red line* is an example shortest path computed by the presented algorithm. Note that start and end positions can be at arbitrary positions inside the navigable space

universally useful for the computation of any shortest path in a given environment. Since it does not need to be changed, we call it the static part of the graph. Figure 4a and b show a simple example of such a polygonal environment and the resulting reduced visibility graph (Latombe 1991).

In order to compute a shortest path we add a start vertex s and a goal vertex g to V, as well as edges e = (s, v) and e = (g, v) respectively, where v is an element of V such that e does not leave the polygon or crosses an obstacle. Furthermore we do not need to consider edges as described in Fig. 3. Since these edges have to be established for every shortest path computation anew we call it the dynamic part of the graph. Figure 4c shows an example for these dynamic edges. Using such a graph consisting of a static and a dynamic part we can apply well-known shortest path algorithms as we will describe in detail in the next chapter to obtain a shortest path (see also Fig. 4d). The division of the graph into a static and a dynamic part makes the construction of the graph run-time efficient.

The concept of visibility graphs has several advantages over the space discretizing graphs described in the first section:

- An exact geometrically shortest path can be computed.
- An exact position (e.g. measured by some self-localization system) can be integrated directly into the graph without loss of accuracy due to discretization.

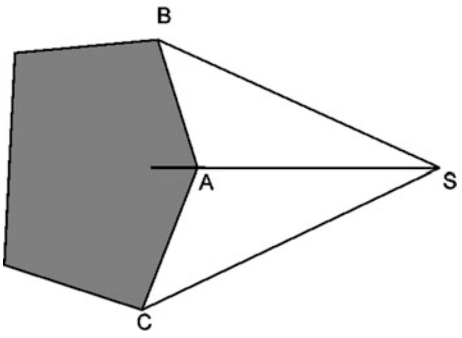

Fig. 3 Reduction of the visibility graph: (S, A) intersects an obstacle when elongated by an ε. Any path starting at S and passing the obstacle has to use either B or C. A path using (S, A) cannot be a shortest path because of the *triangle* inequality: |(S, A)| + |(A, B)| > |(S, B)| and analogously for C. Therefore the edge (S, A) is not needed for the computation of a shortest path and can be removed from the graph

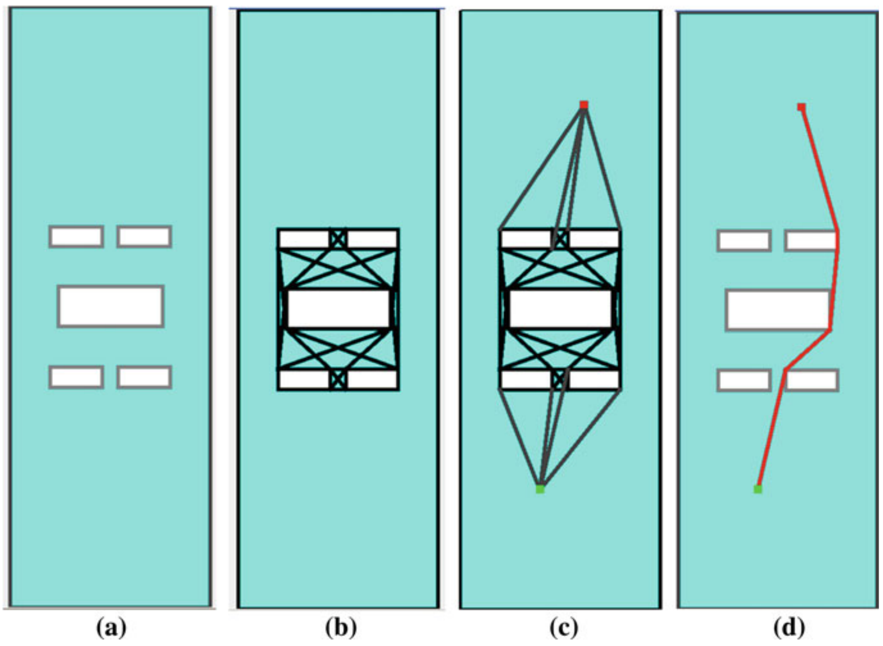

Fig. 4 Computing a shortest path with visibility graph: **a** a room with five obstacles, **b** the resulting reduced visibility graph, **c** adding vertices and edges dynamically to the graph for start and goal positions, **d** the resulting shortest path

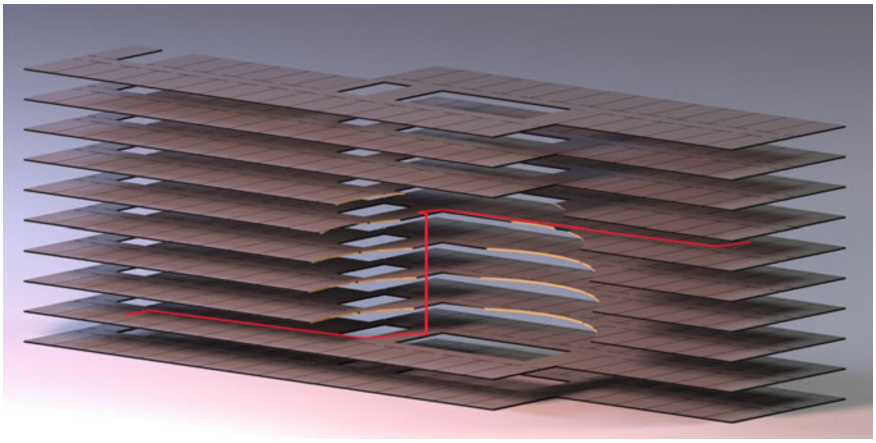

Fig. 5 A 10-story building. The different floors are connected by four *centrally* located elevators and two staircases in the *middle* part and on the *left* side, respectively. The *red line* shows a shortest path using an elevator

4 Shortest Paths with Customized A*

The probably most famous shortest path algorithm is the algorithm given in (Dijkstra 1959). The algorithm works by managing two sets of nodes: the visited and the unvisited nodes. Initially the visited set consists only of the start node while all other nodes are part of the unvisited set. Then one node n is chosen out of the nodes of the unvisited set which is adjacent to a node of the visited set and has a minimum distance d(n) to the start node (adjacent means the node is connected to another node by an edge). This distance is computed by adding the length of the edge which connects the node to one node n_v of the visited set to the distance this node n_v has to the start node.

$$f_{Dijkstra}(n) := d(n) := d(n_{predecessor}) + e(n_{predecessor}, n) \rightarrow min \tag{1}$$

Repeating the last steps until the goal node is added to the visited set yields the shortest path.

In other words the algorithm of Dijkstra expands the set of visited nodes successively in all directions until the goal node is found. In Hart et al. (1968) the A* algorithm was introduced, which can be seen as an extension of Dijkstra's algorithm. They suggest to add a heuristic h(n) which steers the expansion of the set of visited nodes into the direction of the goal node. Hence, A* chooses the node of the unvisited set which is adjacent to a node of the visited set and where the sum of the shortest distance to the start node and the approximated distance to the goal node is minimal.

$$f_{A*}(n) := d(n) + h(n) \rightarrow min \tag{2}$$

These two algorithms introduced by Dijkstra and by Hart et al. can both guarantee to find a shortest path in a weighted graph like the one we constructed in the last section (without negative edge weights). A* additionally needs an admissible heuristic h(n) as input, which estimates the distance of a node n to the goal node. A heuristic is admissible if it does not overestimate the distance to the goal node. In order to be implemented in a run time efficient way the heuristic also has to be monotonic, meaning that for every two adjacent nodes n_1 and n_2 the following in equation is fulfilled:

$$h(n_1) \leq h(n_2) + e(n_1, n_2) \tag{3}$$

The probably most often used heuristic in graphs which represent topologies in a euclidean space is the euclidean distance.

$$\begin{aligned} h_e(n) &:= dist(n, goal) \\ &= \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2 + (n.z - goal.z)^2} \end{aligned} \tag{4}$$

In the visibility graph described above, where edge weights are set to the euclidean distance of the two nodes, in Eq. (3) can be written as

$$dist(n_1, goal) \leq dist(n_2, goal) + dist(n_1, n_2) \tag{5}$$

which is a variant of the triangle inequality and therefore the heuristic h_e is admissible.

Another admissible heuristic is

$$h_0(n) = 0. \tag{6}$$

Using this heuristic h_0 f_{A*} is reduced to $f_{Dijkstra}$ and the A*-algorithm behaves like Dijkstra's algorithm.

In Fig. 6 you can see a behaviour for the three-dimensional case. Here A* still outperforms the algorithm by Dijkstra in regard to the number of expanded nodes but expands many nodes on the floors between the start and goal node. This happens because the heuristic h_e does not differentiate between the three axes of the coordinate system. A* tries to approach the goal node on every floor.

*4.1 Customized A**

We optimize the behaviour of A* by using our knowledge on the graph's structure. We know that there are relatively many nodes and edges on each floor but that these floors are only sparsely interconnected. Therefore on reaching steps or an elevator we would like the algorithm to prioritize a vertical movement over a horizontal one: We achieve this by using a different heuristic:

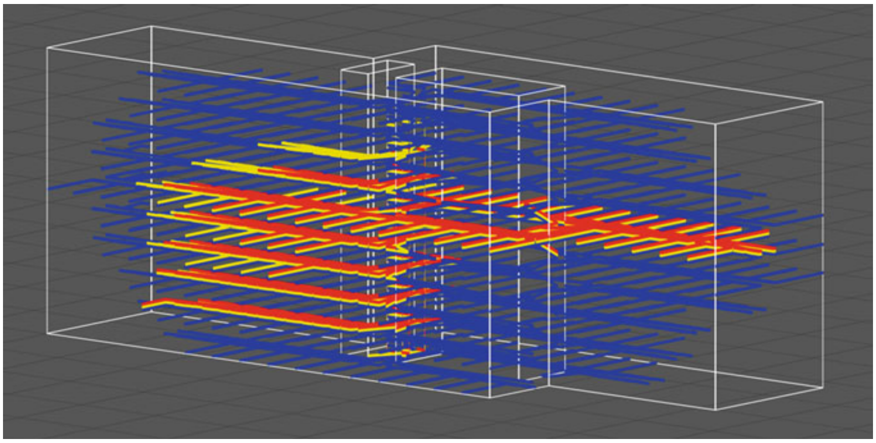

Fig. 6 Three sets of visited nodes: Dijkstra's algorithm (*blue*), A* using the standard heuristic h_e (*yellow*) and A* using the proposed heuristic $h_{3D\_building}$ specialized for routing in 3D buildings. We are visualizing the edges leading to the visited nodes instead of the nodes itself because of clearer visibility

$$h_{3D\_building}(n) := \sqrt{w * ((n.x - goal.x)^2 + (n.y - goal.y)^2) + (n.z - goal.z)^2} \tag{7}$$

with a weight w out of [0,1]. Experiments showed that 0.5 is a valid choice of w in our test case. Using this heuristic the vertical distance gets a greater weight. In Fig. 6 you can see a resulting set of expanded nodes (red). A* using our heuristic $h_{3D\_building}$ outperforms both Dijkstra's algorithm as well as A* using the standard heuristic h_e.

4.2 Experiments

In order to do a more extensive test we chose 100 pairs of points in our 10-story test building (see Fig. 5) at random and computed the shortest path with all three algorithmic options on this set of points. We made sure, that none of these randomly chosen points lies outside of the building so we have no outliers. The test was done on an Athlon 64 X2 system with 2.21 GHz. The graph consists of 3,832 nodes and 11,886 edges. All three algorithmic options find the same path for each of the point pairs.

Table 1 shows that on average A* using our heuristic h_3D_building needs 29 % less run-time than A* using the heuristic h_e and 77 % less run-time than Dijkstra's algorithm.

Table 1 Accumulated computation times and numbers of expanded nodes of 100 shortest paths between randomly chosen points of the 10-story building depicted in Fig. 5

	Dijkstra	A*	A*-3D_building
Time	192 s	63 s	45 s
Expanded nodes	94,950	29,975	25,921

The runtime includes the generation of the dynamic parts of the graph and the shortest path computation itself

5 Conclusion

We showed that using the concept of visibility graphs can improve indoor navigation without requiring excessive amounts of run-time. Based on the assumption that each floor is modelled as a polygon we showed how highly accurate shortest paths can be computed using completely arbitrary positions for start and goal nodes. To achieve this we adopted the concept of reduced visibility graphs which are automatically constructed from polygonal floor plans. Furthermore, we introduced a new heuristic which enables A* to favour vertical over horizontal movement, leading to 29 % less run-time.

Acknowledgments The author would like to thank Bernhard Steuer for the visualisation of the data in Figs. 5 and 6.

References

Ariza-Villaverde AB, de Ravé EG, Jiménez-Hornero FJ, Pavón-Domínguez P, Muñoz-ermejo F (2010) Introducing a geographic information system as computer tool to apply the problem-based learning process in public buildings indoor routing. Comput Appl Eng Educ. doi:10.1002/cae.20442

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271

Evennou F, Marx F (2006) Advanced integration of WIFI and inertial navigation systems for indoor mobile positioning. EURASIP J Appl Signal Process 2006:1–11

Goetz M, Zipf A (2011) Formal definition of an user-adaptive and length-optimal routing graph for complex indoor environments. Geo-spatial Inf Sci 14(2):119–128

Hagedorn B, Trapp M, Glander T et al. (2009) Towards an indoor level-of-detail model for route visualization. Proceeding 10th International Conference on Mobile Data Management: systems services and middleware 692–697

Haklay M, Weber P (2008) Open street map: user-generated street maps. IEEE Pervasive Comput 7(4):12–18

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107

Karimi HA, Ghafourian M (2010) Indoor routing for individuals with special needs and preferences. Trans GIS 14(3):299–329

Latombe JC (1991) Robot motion planning. Kluwer Academic Publishers, Boston

Lozano-Pérez T, Wesley MA (1979) An algorithm for planning collision-free paths among polyhedral obstacles. Commun ACM 22:560–570

May AJ, Ross T, Bayer SH, Tarkiainen MJ (2003) Pedestrian navigation aids: information requirements and design implications. Pers Ubiquit Comput 7(6):331–338