Supporting 3D Route Planning in Indoor Space Based on the LEGO Representation

Wenjie Yuan & Markus Schneider*
Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611, USA
{wyuan, mschneid}@cise.ufl.edu

ABSTRACT

With the prevalence of car navigation systems, indoor navigation systems are increasingly attracting attention in the indoor research area. However, the available models for indoor navigation suffer from the problems that architectural constraints are not considered, route planning is only based on 2D planes, users are represented as points without considering their volumes, and different requirements asked for by different users are ignored. Consequently, the routes provided by existing models may not be suitable for different kinds of users like pedestrians, persons in wheelchairs, and persons driving indoor autos. This paper proposes a cube-based model to compute feasible routes for different users according to their widths, heights, and special requirements (e.g., users in wheelchairs prefer the routes without stairs). In this model, an indoor space is first represented by multiple cubes with different types. Then, according to the heights and types of the cubes, possible passages with the maximum widths and heights are generated by merging cubes into large blocks. Based on these blocks, feasible routes are computed by checking the availability of the connectors between different blocks.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Spatial databases and GIS

General Terms

Design

Keywords

3D indoor navigation, 3D indoor spatial model

*This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574 and NSF-IIS-0915914.

ISA '10, 2 November 2010, San Jose, CA, USA

1. INTRODUCTION

In recent years, car navigation systems have become an established tool for route planning. With the help of car navigation systems, users can easily locate a desired place, obtain a detailed route from location A to location B, or learn the information about the interesting locations nearby. Observing the convenience that car navigation systems bring to human's daily life, researchers have started to design indoor navigation systems to help users find their ways in indoor spaces, especially in large buildings with complex structures.

In the literature, several indoor navigation models have been proposed to support route planning in indoor spaces. Unfortunately, they suffer from at least one of the following problems. First, some of them ignore the architectural constraints (e.g., doors, obstacles, and walls) in an indoor space. In fact, without considering the constraints, the generated routes may not be detailed enough. For instance, if a room has multiple doors, users may not be able to know which door they should take. Second, some of the models are designed for pedestrians only, while some of them are for wheelchairs only. However, few of them are able to provide different routes for different types of users. Third, most of the models represent users by points and paths by lines. However, for some kinds of users, their widths and heights may prevent them from accessing a narrow passage. Thus, a feasible navigation model should take into account the 3D features of both users and routes. Last, most of the models, including 2D models and 3D models, assume that the floors of the buildings are always flat. However, this is not the case. A lot of modern buildings have multiple layers in the same floor, connected by small stairs or slopes. Obviously, routes containing small stairs are not suitable for users in wheelchairs or driving indoor autos.

The goal of this paper is to solve the aforementioned problems from a modeling perspective and propose an approach to providing feasible 3D routes for different users with different widths, heights, and special requirements in an indoor space. In our model, a 3D indoor space is approximated by several *LEGO_cubes*. All the cubes have a basal area of the same size, but each of them has its own height and type according to the object it represents. The maximum accessible spaces in different locations are obtained by merging available cubes into blocks. The generated blocks are also the implicit passages in an indoor space. The connecting part (called a *connector*) of two blocks is the maximum accessible width when users go from one block to the other. Thus, after generating all the blocks and computing their connectors, a

graph can be built to reflect the 3D accessibility information between different places.

The rest of the paper is organized as follows. Section 2 discusses the available approaches to indoor navigation models. Section 3 introduces the LEGO representation we use to represent an indoor space. The approach to constructing 3D paths is proposed in Section 4. In Section 5, a LEGO graph is introduced to support efficient path searching algorithms. Finally, Section 6 draws some conclusions and depicts future work.

2. RELATED WORK

Earlier indoor navigation models are purely symbolic models [3, 17, 5, 15, 16] based on a labeling system without considering the geometry of an indoor space. A route generated by these models is a sequence of object labels. Later models try to generate more detailed routes by considering the geometry of indoor spaces. There are two types of approaches to supporting route planning in indoor spaces: grid-based approaches and path-based approaches. The grid-based models first decompose the space into cells and compute routes by exploring the connectivity of these cells. The path-based approaches construct implicit paths in an indoor space that are directly based on the architectural constraints.

There are two general types of methods for grid-based models. The first type [7] tries to decompose the available space into different shapes of cells, such as triangles and polygons. The union of the generated cells is exactly the available space. The model presented in [7] decomposes the space by using differently shaped cells. It first subdivides the space into several triangles by detecting the bottlenecks inside rooms. These triangles are merged into several convex cells of different sizes. However, this approach is limited to 2D flat planes. This kind of model can precisely represent the space. However, they are inefficient. The other type [2, 22, 1] is to represent the available space by unified shapes (e.g., rectangles). The union of the generated cells may not be exactly the available space, especially for the space on the boundaries. However, since they use simple and unified representative units, they are usually more efficient for route planning. The model proposed in [2] is one of the most popular grid-based models. In this model, the available space is decomposed into cells marked as *obstacle* or *non-obstacle*. Based on this representation, routes can be computed by checking the availability of cell movements to their eight neighbors. This model also supports navigation in the 3D space by filling out the indoor space with the *obstacle* and *non-obstacle* cubes (as shown in Figure 1). The obstacle cubes are further classified into insurmountable and surmountable ones to facilitate the 3D navigation. In [1], a hierarchical model is proposed by merging cells to form topological maps. This model can more efficiently compute the routes when the number of the representative cells is large.

The second type of approaches tries to find the implicit paths directly on the basis of the architectural constraints. Models in [4, 23] use center points to represent cells and build routes based on the reachability of the cells. However, without considering the locations of the exits, the generated routes are very coarse. In addition, the routes generated by these models are circuitous because they always lead users to the center points of the rooms. The *CoINS* model in [12, 13] simplifies the route by eliminating some unnecessary nodes

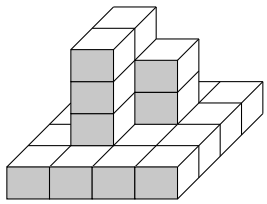


Figure 1: Representing space by using cubes

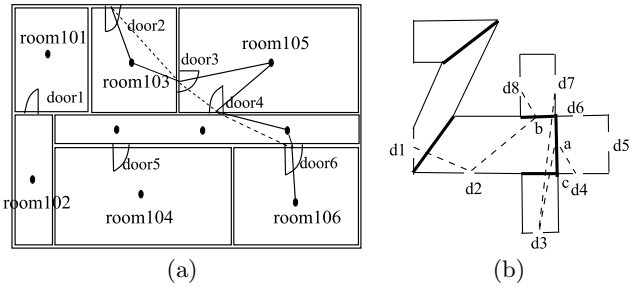


Figure 2: The solid line in (a) indicates Lorenz's model, and (b) is an example of region partitioning in Lorenz's model.

and recalculating the segments between different nodes. The models proposed in [6, 24, 11, 20, 25] take into account more features of an indoor space, such as the locations of the exits and the shapes of the boundaries. [6] proposes a model containing locations and exits, which are decorated by semantic information. This model captures the relationships between the cells and the exits, and organizes the cells and exits in a hierarchical structure according to their reachability. However, this model cannot determine detailed path information like distance and direction. As shown in Figure 2a, the model in [11] employs some representative points to represent rooms, corridors, and exits. The calculation of the path is processed based on the connections between these representative nodes. In [20] the authors extend their model by decomposing cells into several convex regions to provide users better route instructions. A big problem of this model is that it generates circuitous routes. The solid line in Figure 2a is the path from *door2* to *room106* that the model generates. As we can see, this route is not an optimal one. Users will prefer the path specified by the dashed line in Figure 2a. The *iNav* model proposed in [25] eliminates the circuitous routes by constructing the shortest path segments between exits. The shortest paths are constructed by analyzing the location of the exits and the shapes of the cells. Based on these shortest path segments, the iNav model is able to provide the shortest routes between different objects.

The above mentioned models assume that the floor of the cells are 2D planes. Actually, this assumption is insufficient to represent the entire indoor space. Places with the same x and y coordinate values may be located in different floors. Even in the same floor, cells may have different heights. In addition, it is common to have multiple layers connected by small stairs and slopes in one floor. Thus, some models try to propose 3D models for indoor representations and navigation.

In [8, 9], Lee proposes a 3D model to represent the topological relations in indoor spaces. In this model, Poincaré

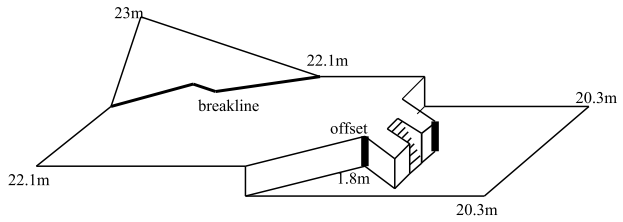


Figure 3: The layered structure of the floor

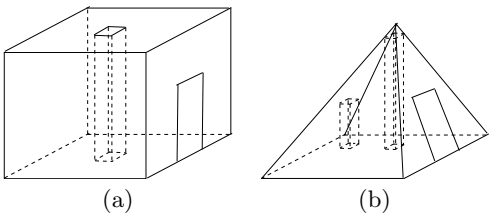


Figure 4: A cube in a cell with regular shape (a), and cubes in a pyramid shaped cell (b)

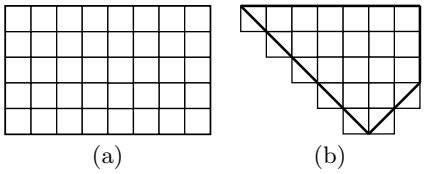


Figure 5: The 2D projection of the approximation in a cell with a rectangular shape (a), and with a triangular shape (b)

duality combined with a hierarchical network structure are used to explore the relations between objects. Although this is a 3D model, the 3D features are only used to distinguish different floors. The floors are considered to be flat, and the construction of the paths in one floor still focuses on 2D data. The same problem arises in the models in [21, 10]. In [14], the authors present a semantic model of interior spaces to facilitate the calculation of evacuation routes. This model takes into account different features of the interiors, such as the types of the passing (e.g. *uni-* or *bi-directional*) and the types of the boundaries (e.g. persistent boundaries like walls and virtual boundaries like openings). By using these features, this model is able to distinguish the accessible parts and non-accessible parts in an indoor space. However, this mode focuses on the surroundings but ignores the structure of the floor plane. The model proposed in [18, 19] is the only model we have found talking about the structure of the floor plane. As shown in Figure 3, this model defines different types of height to store the layered structure of the floor. Thus, small stairs and slopes can be captured during the route planning. However, when the author extends this model to support navigation, the 3D structure of the floor is only used to represent discrete and connected spaces but not to construct different kinds of paths.

3. 3D REPRESENTATION OF THE INDOOR SPACE

An indoor space is composed of different kinds of cells. Each cell unit is restricted by a lot of architectural constraints like floors, walls, and exits. The structure of a cell can be very complicated if the components of the cell have complex structures. For example, a pyramid ceiling may affect the accessibility of certain places inside the cell, and small stairs on the floor are only suitable for pedestrians. In this section, we will discuss our approach to representing the 3D structure of the indoor space.

3.1 LEGO-Based Approximation

Our representation model is inspired by LEGO's toy bricks. By using different shapes of the toy bricks, we are able to build various kinds of cell spaces. Thus, our approach to model the 3D space in cells is to approximate the entire space by cubes. For simplicity, we only use cubes, called *LEGO_cubes*, with the same basal area as our representative units. Each LEGO_cube has its own height and type according to the object it represents. A LEGO_cube is non-dividable, and it can only represent one object or a part of an object. The 2D projection of all the cubes in one cell is a grid with equally sized squares. Figure 4 shows an example of a cube in a cube-shaped cell and two cubes with different heights in a pyramid-shaped cell. As shown in Figure 4b, the height of each cube depends on the distance between the floor and the ceiling in the corresponding area which the cube locates. The size of the square controls the granularity of the representation. The basal area is the minimum unit for representing the 2D structure of the cells. The smaller the square is, the more detailed an indoor space can be represented.

Figure 4a is the simplest example of representing an indoor space by using LEGO_cubes. However, in most cases, the structures in cells are not so simple. Irregular shaped floors and ceilings, obstacles, and complicated shapes of cells will increase the difficulty of the representation. In our model, we classify different objects into three main categories: *planes*, *stairs* and *obstacles*.

The formal definition of the LEGO_cube is given in Definition 1.

Definition 1. A LEGO_cube $LC := (id, type, slope, w, h)$ is a basic 3D unit to represent objects in an indoor space. id is the unique identifier of the LEGO_cube. $type$ records the type of the LEGO_cube. The value of the $type$ can be *obstacle_cube*, *stair_cube* or *plane_cube*. $slope$ is the slope of a plane_cube. If the type of the LEGO_cube is obstacle_cube or stair_cube, the slope value is always 1. w is the width of the basal area of the LEGO_cube. h is the height of the LEGO_cube.

3.1.1 The Approximation of Planes

When a floor and a ceiling are flat, and there is no obstacle between them, it is easy to fill out the 3D available space between them by LEGO_cubes. As shown in Figure 5, the 2D projection of the representation is a grid structure. Figure 5a is an example of a cell with a rectangular shape. If the boundary of the cell is not horizontal or vertical, the boundary is represented by multiple LEGO_cubes connected by the corners of the basal areas of the cubes (as shown in Figure 5b). The resolution of the approximation depends on the size of the basal area. If we reduce the size of the basal area, the space can be represented more precisely.

When a floor or a ceiling is not flat, the available space can be approximated by multiple LEGO_cubes with differ-

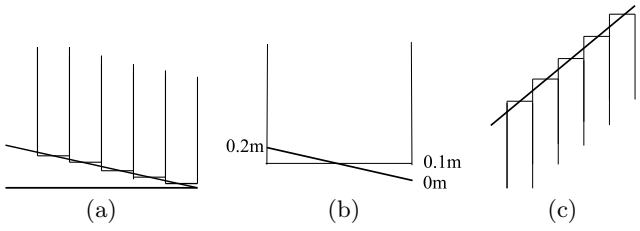


Figure 6: The longitudinal section of the representation of a sloping floor (a) and a sloping ceiling (c). (b) shows how to determine the basal area of a LEGO_cube for a sloping floor.

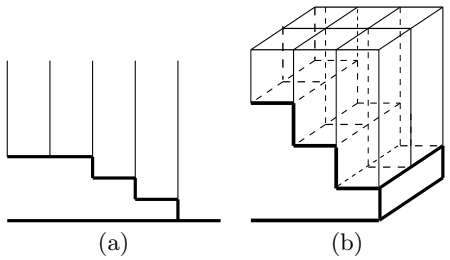


Figure 7: The longitudinal section (a) and the three dimensional figure (b) of the representation for stairs

ent heights. The vertical location of a cell's basal/top area depends on the average height of the corresponding area in the floor/ceiling. Sloping floors or ceilings are represented be a set of gradually ascending or descending LEGO_cubes. Figure 6a shows the longitudinal section of the representation of a sloping floor. Figure 6b shows how to determine the vertical location of the basal area of such a LEGO_cube. Assuming the vertical locations of the end points of the sloping floor are $0m$ and $0.2m$, the vertical location of the basal area of the representative LEGO_cube is $0.1m$, which is the average vertical location of the two end points. The approximation for a sloping ceiling is the same as for a sloping floor. As shown in Figure 6c, the top area of the LEGO_cubes are gradually descending according to the shape of the ceiling.

Sometimes, a plane may be too sloping for users to use. This plane is then actually an obstacle that is unavailable for users. In our model, we use a threshold for the slopes to control the availability of the planes. If the sloping of a plane exceeds the threshold, this plane is considered as an obstacle and is represented by obstacle_cubes.

3.1.2 The Approximation of Stairs

Similar to the approximation of the sloping planes, stairs are represented by a set of ascending or descending LEGO_cubes. The width of the LEGO_cubes cannot exceed the width of the corresponding steps. Thus, each step is represented by one or more LEGO_cubes. The vertical locations of the basal areas are determined by the locations of the corresponding steps. Figure 7 shows an example of the longitudinal section and the 3D diagram of the approximation.

3.1.3 The Approximation of Obstacles

Obstacles refer to the objects whose occupied areas are not available to users. They can be walls, tables, chairs, and other objects. When an obstacle lays on a floor, no matter how high it is, this area is considered as inaccessible for users in wheelchairs. However, it may be available for pedestrians. Usually, different obstacles have different shapes. Their representations can be classified into three types.

The first type refers to obstacles that are too high to be passed over. Typical examples are walls, furniture like tables, and some decorations like potted flowers. Among these obstacles, some of them reach the ceiling and some of them still have free space above them. Since the free space above these obstacles cannot be used as passages, it is actually unavailable to users. Thus, the obstacles of this type are represented by LEGO_cubes whose top areas reach the ceilings. This means the space from the floor to the ceiling in this location is unavailable.

The second type refers to obstacles that pedestrians can pass over. In our model, these obstacles are considered as curbs. They are available for pedestrians but unavailable for users in wheelchairs, for example. These curbs are considered as small stairs and represented by stair_cubes.

The third type refers to obstacles in the air. The spaces below them are available to users. This kind of obstacles will limit the height of the available space below them. The space from the floor to the ceiling in this area will be represented by two different LEGO_cubes. The bottom one represents the available space, and the upper one represents the obstacle. The top area of the bottom cube will be the basal area of the upper cube.

Sometimes, different parts of an obstacle may belong to different types. For obstacles with combined shapes, we first divide them into multiple parts according to the classification of the shapes, and then approximate them by using different strategies. An example is the trees used to decorate the environment in indoor space. The trunks of the trees are the obstacles of the first type, while the extended branches are the obstacles in the air. Users cannot use the space occupied by the trunk, but they can go through the space under the branches if the branches are high enough. Thus, different parts of the tree belong to different types of obstacles.

4. CONSTRUCTING 3D PATHS IN INDOOR SPACE

Our goal is to find a feasible path for the user with particular width, height, and requirements. For example, persons in wheelchairs or driving indoor autos prefer routes without stairs. Figure 8a is the 2D projection of a cell represented by LEGO_cubes. The white, black, and grey cubes represent the available space, obstacles, and stairs respectively. From the figure we can see that the maximum accessible width of the paths between two doors in this cell is restricted by the small gap between the two obstacles. Only the users slimmer than this gap can go through this room from one door to the other. If the user cannot walk through this gap, this room cannot be an available path for the user. An available route we provide to a user should have the following features: (i) be wide and high enough for the user to go through; (ii) avoid all the obstacles; and (iii) avoid stairs if the user requires this.

We achieve our goal of finding the proper routes in three steps. First, LEGO_cubes are merged together to form larger rectangle-shaped blocks. These blocks can reflect the maximum available widths and heights in their locations. Second, we compute the maximum accessible widths between two ad-

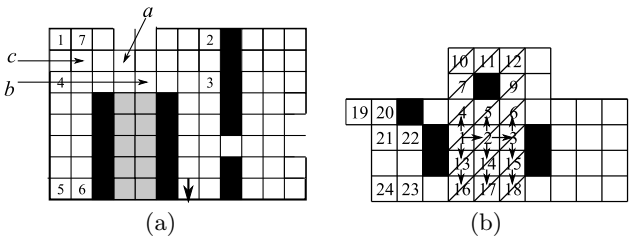


Figure 8: Examples of merging cubes to generate larger blocks

jacent blocks to control the accessibility of the two blocks. Third, a LEGO graph is derived from the cube-based structure to support the existing shortest path searching algorithms.

4.1 Merging Cubes of the Same Type to Form Blocks with Maximum Width

The cube-based representation introduced in Section 3 divides an entire indoor space into multiple LEGO_cubes. Each cube has its unique type indicating whether it is a part of a plane, a staircase, or an obstacle. Usually, an object is represented by several connected LEGO_cubes. Thus, if a LEGO_cube is available to a user (e.g., a part of a plane), its neighbor cubes may also be available to this user. By merging similar LEGO_cubes, we are able to find the maximum, available width for a particular area.

As shown in Figure 8a, by merging nearby cubes, the maximum rectangle-shaped block containing the cube a is the rectangle with the corner cubes 1, 2, 3 and 4. Actually, the width of a block depends on the directions of the movements. If you walk in the vertical direction, the width of this block contains 8 cubes, and if you walk along the horizontal direction, the width of the block becomes 3 cubes. The maximum blocks extended from one LEGO_cube may be different depending on the merging direction. For example, the maximum block containing cube c merged along the horizontal direction is the rectangle $(1, 2, 3, 4)$. However, along the vertical direction, the maximum block is the rectangle $(1, 5, 6, 7)$. Therefore, in our model, there are usually two blocks extended from the same cube.

Before we introduce our merge strategy, we first discuss several conditions involved in the merging process.

First, only the LEGO_cubes with the same type can be merged together. Different types of LEGO_cubes represent different kinds of objects in an indoor space. It is impossible for us to walk from a plane into an obstacle, and for some users, stair cubes are not available to them.

Second, although the heights of LEGO_cubes reflect the heights of the available space inside cells, the accessible height of a cell is actually controlled by the heights of the exits. Assuming the *default height* of one cell is the maximum height among all the exits of this cell, then the LEGO_cubes higher than the default height can be accessed without restriction. Therefore, the LEGO_cubes higher than the default height are able to be merged together. For the LEGO_cubes lower than the default height, only the cubes with the same height can be merged together.

Third, the number of LEGO_cubes used to represent cells will affect the efficiency of the merging process. If there are a lot of LEGO_cubes, the process may take a lot of time. But we observe that the blocks merged from different cubes may be the same. For example, in Figure 8a, the blocks merged from cube a and from cube b coincide to the same block. Thus, in order to avoid duplicates and improve the merging process, we can reduce the number of starting cubes. As shown in Figure 8a, from any randomly selected cube, the maximum width we can obtain always depends on the *boundary cubes*, which are next to other types of cubes. Thus, we only choose the boundary LEGO_cubes to be the starting cubes to perform the merging.

Our merging process on the horizontal direction starts from a boundary cube. The cubes horizontally extended from this starting cube are merged one by one to obtain the maximum width. As shown in Figure 8b, the cubes horizontally next to the cube 1 are merged one by one up to the boundary. This step produces a temporary block with the maximum width obtained from the starting cube (e.g., the rectangle $(1, 2, 3)$). Then, all the cubes directly above the temporary block are checked to see if they have the same type and satisfy the height condition. If so, they will be merged into the temp block. For example, the cubes 4, 5, 6 that are directly above the rectangle (1, 2, 3) can be merged into the rectangle (1, 2, 3). If one of the cubes does not satisfy the condition, the cube extension in this direction will stop. Therefore, the cubes above (4, 5, 6) cannot be merged. The cubes under (1, 2, 3) are merged by using the same strategy. The final block generated from cube 1 is the rectangle (4, 6, 18, 16). If we apply the algorithm along the vertical direction, the rectangle (10, 16) will be the corresponding block extended from cube 1 that contains the maximum vertical width.

The result of the merging process is a set of blocks. There are three relationships between any two blocks. The first relationship is *disjoint*. For example, in Figure 8b, the block (10, 16) vertically extended from cube 1 and the block (12, 18) vertically extended from cube 3 are disjoint. The second relation is *adjacent*. This means that two adjacent blocks share a part of the boundary. For example, the block (19, 20) horizontally extended from cube 19 and the block (21, 22, 23, 24) horizontally extended from cube 22 are adjacent. Two blocks can also *overlap*. In Figure 8b, the block (10, 16) and the block (4, 6, 18, 16) overlap. However, one block cannot be fully inside another block. Because if one block is fully inside another block, it also can be extended to form the outer block by merging nearby cubes. It is also impossible to have two blocks with the relationships similar to Figure 10d since the boundary a of the block B can be extended to meet the boundary b of the block A.

4.2 Improvement of the Merging Strategy

Although this merging strategy can generate blocks with the maximum width in the horizontal and vertical directions in cells, it may produce unnecessary blocks in some situations. For example, the irregular obstacle separating two available spaces in Figure 9 leads to irregular boundary cubes. According to the merging strategy, the block extended from the boundary cubes will be a set of adjacent blocks shown by the bolded lines. However, compared to the slashed area, the small cubes near the boundary usually do not contribute to the available space. Therefore, for simplicity and efficiency, when the boundary between an obstacle and an available space has a zigzag shape, this boundary can be simplified into a straight line before performing the

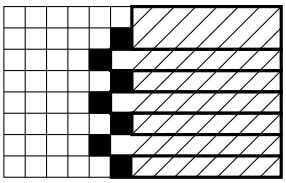


Figure 9: Improve the merging process by simplifying the boundary cubes

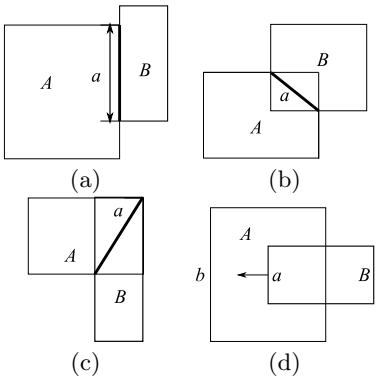


Figure 10: Constructed connectors according to the relationships between blocks

merging.

4.3 Connectors between Blocks

Once we generate the blocks, the routes from the starting place to users' destinations can be considered as block-by-block paths. The connections between two blocks (called *connectors*) control the maximum accessible width and height from one block to the other. There are two types of connectors, one is to connect two blocks with the same type, and the other is to connect two blocks which are in different types (e.g., the connectors between planes and stairs).

4.3.1 Connectors between Two Blocks with the Same Type

According to Section 4.1, there are three relationships between two blocks, namely disjoint, adjacent, and overlap. When the two blocks are adjacent or overlap, users can move from one block to the other if their connector is wide enough.

When two blocks are adjacent, the connector between them is the common boundary. For example, the line a is the connector between block A and B in Figure 10a.

When two blocks overlap, their boundaries will always be crisscross as shown in Figure 10b and c. The connector between two overlapping blocks is the diagonal of the intersecting rectangle. As shown in Figure 10b and c, the line a is the maximum connector between blocks A and B.

4.3.2 Connectors between Available Spaces with Different Types

The connectors between two blocks with the same type can reflect the accessible width between two areas. However, when the task is to find the maximum accessible width between two different types of available areas (e.g., planes and stairs), the blocks generated by applying the merging strategy may produce incorrect maximum connectors, especially when the boundaries are not horizontal or vertical.

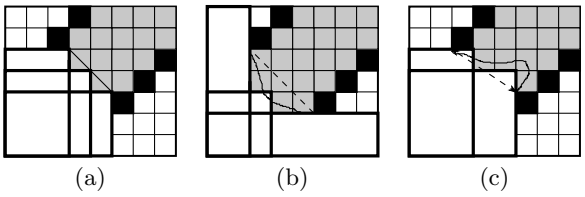


Figure 11: Examples showing that the connectors with the maximum width between a plane and a staircase cannot be found

Figure 11 shows some examples of the boundary between a plane and a staircase. The white part is the available plane, and the grey part represents the area of the staircase. The irregular lines and the black cubes denote the accessible and unaccessible boundaries between the plane and the stairs respectively. The rectangle blocks bounded by bolded lines in the plane are the blocks generated by applying the merging strategy on the boundary cubes. The maximum accessible width between the plane and the stairs for Figure 11a, b, and c are denoted by the dashed lines. From these figures we can see that none of the blocks can capture the maximum accessible width between the plane and the stairs. Therefore, we are unable to obtain the maximum accessible width between these two areas.

The incorrect results are caused by the merging process. The merging process will stop when a cube with another type is met. If the boundary between a plane and a staircase is not horizontal or vertical, this process will not be able to generate a block containing the entire boundary between the two available spaces. In order to capture the maximum accessible width between two different available areas, we have to generate blocks containing as much part of the boundary as possible.

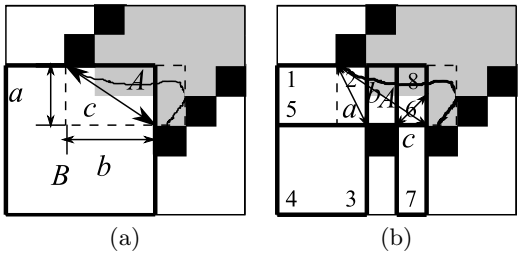
Our approach to solving this problem is to generate blocks by merging plane_cubes and stair_cubes together. This means if a stair_cube (plane_cube) is met when merging plane_cubes (stair_cubes), this stair_cube (plane_cube) is also merged into the block. As shown in Figure 12a, B is the block by merging both the plane_cubes and the stair_cubes. The block A denoted by the dashed lines is the minimum bounding box of the accessible boundary. The connector between A and B is the diagonal c, which reflects the maximum accessible width between the plane and the staircase.

This approach also works when the boundary is affected by obstacles. As shown in Figure 12b, the rectangle (1, 2, 3, 4), (1, 8, 6, 5) and (8, 7) are the blocks produced by applying the new approach. The connectors a, b, and c correctly reflect the maximum accessible widths in different directions in this scenario.

As we mentioned in 4.1, all the cubes in one block either have the same height, or are higher than the default height. For the former case, the height of a block is set to be the height of the cubes in the block, and for the latter case, the default height becomes the height of the block. The height of the connector is the minimum height of the two blocks sharing the connector.

The definition of a connector is given in Definition 2.

Definition 2. A connector $C = (loc_1, loc_2, height)$ is a list of connected cubes representing the maximum accessible



(a) (b)

Figure 12: Computing connectors between a plane and a staircase without the effect of obstacles (a) and with the effect of an obstacle (b)

width and height when moving from one block to the other. loc_1 and loc_2 are the center points of the two end cubes in this connector. *height* is the minimum height of the two blocks sharing this connector.

5. THE LEGO GRAPH

Most of the existing path searching algorithms (e.g., the shortest path search and the A* algorithm) are graph-based algorithms. In this section, we will discuss how to build a graph to support route searching algorithms.

In the previous section, we have introduced our strategy to build blocks with maximum width. Users can walk block by block to reach their targets. The accessible widths and heights are restricted by the connectors between blocks. If we build a graph in which nodes denote blocks and edges represent connectors, this graph is able to reflect the connectivity among the blocks. For example, Figure 13b is such a graph consisting of all the blocks and connectors for the scenario shown in Figure 13a. By using this graph, we are able to find feasible routes for users. However, we cannot guarantee that the obtained routes are the best ones (e.g., the shortest ones) for users.

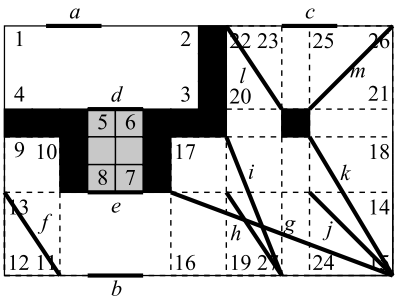
Actually, the process of walking block by block is the same as the process of walking connectors by connectors. Therefore, we build a graph, called *LEGO graph*, in which nodes denote the connectors and edges represent their distances. In order to maintain the types of the blocks in the LEGO graph, we add an attribute *type* to the edges in the LEGO graph. As shown in Figure 10 and Figure 12, our generated blocks are always rectangles, and the connectors are either on the boundary or inside the block. Thus, the path between two connectors is always inside the corresponding block. The length of the edge, denoting the shortest distance between the two connectors, is the length of the straight line between the midpoints of the two connectors. The type of the edge is the type of the corresponding block. The LEGO graph generated from Figure 13a is shown in Figure 13c. Definition 3 provides the formal definition of the LEGO graph.

Definition 3. A LEGO graph $LG = (V, E)$ is a graph which reflects all possible paths with different accessible widths and heights in a given indoor space scenario. V is a set of connectors, and E is a set of implicit paths with the shortest distances and types between two reachable connectors.

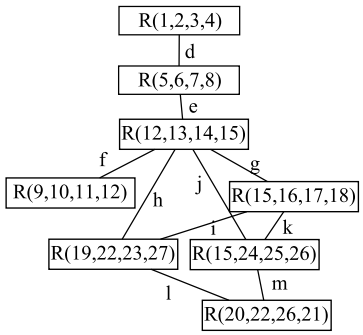
6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a 3D model to support route planning according to users' widths, heights and requirements (e.g., avoid stairs). We have shown how to approximate a 3D indoor space by LEGO_cubes. Based on the cube representation, a LEGO graph containing implicit 3D paths is derived from this structure by merging cubes into blocks and by exploring the connectors between the generated blocks. By using this graph, we are able to provide the best route among all feasible paths.

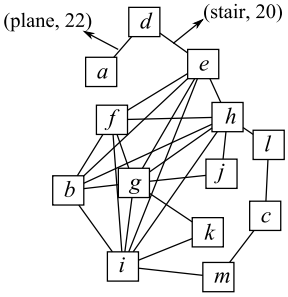
In our model, plane_cubes are merged together without considering their sloping values. Although planes with different sloping values are available for users, it is still too coarse to simply merge them together. For example, routes containing too many small ups and downs are not suitable for users to take. Therefore, how to deal with planes with different sloping values should be further studied.



(a)



(b)



(c)

Figure 13: An example of a floor plane with obstacles and stairs (a), the graph reflecting the connectivity of the blocks (b), and the corresponding LEGO graph (c)

7. REFERENCES

[1] A. Bandera, C. Urdiales, and F. Sandoval. A

Hierarchical Approach to Grid-based and Topological Maps Integration for Autonomous Indoor Navgation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 883–888, 1996.

[2] S. Bandi and D. Thalmann. Space Discretization for Efficient Human Navigation. *Computer Graphics Forum*, 17:195–206, 1998.

[3] B. Brumitt and S. Shafer. Topological World Modeling Using Semantic Spaces. In *In UbiComp 2001 Workshop on Location Modeling for Ubiquitous Computing*, 2001.

[4] P.-Y. Gilliéron and B. Merminod. Personal Navigation System for Indoor Applications. In *11th IAIN World Congress*, pages 21–24, 2003.

[5] P. Hoppenot, G. Pradel, C. Caleanu, N. Perrin, and V. Sommeilly. Towards a Symbolic Representation of an Indoor Environment. In *IEEE CESA, Computing Engineering in Systems Applications*, 2003.

[6] H. Hu and D.-L. Lee. Semantic Location Modeling for Location Navigation in Mobile Environment. In *IEEE International Conference on Mobile Data Management (MDM)*, pages 52–61, 2004.

[7] F. Lamarche and S. Donikian. Crowd of Virtual Humans: A New Approach for Real Time Navigation in Complex and Structured Environments. *Computer Graphics Forum*, 23:509–518, 2004.

[8] J. Lee. A Spatial Access-Oriented Implementation of a 3-D GIS Topological Data Model for Urban Entities. *GeoInformatica*, 8(3):237–264, 2004.

[9] J. Lee. A Combinatorial Data Model for Representing Topological Relations among 3D Geographical Features in Micro-Spatial Environments. *International Journal of Geographic Information Science*, 19(10):1039–1056, 2005.

[10] Y. Li and Z. He. 3D Indoor Navigation: a Framework of Combining BIM with 3D GIS. In *44th ISOCARP Congress*, 2008.

[11] B. Lorenz, H. Ohlbach, and E.-P. Stoffel. A Hybrid Spatial Model for Representing Indoor Environments. volume 4295/2006, pages 102–112. 2006.

[12] F. Lyardet, J. Grimmer, and M. Muhlhauser. CoINS: Context Sensitive Indoor Navigation System. In *8th IEEE International Symposium on Multimedia*, pages 209–218, Dec. 2006.

[13] F. Lyardet, D. W. Szeto, and E. Aitenbichler. Context-Aware Indoor Navigation. In *European Conference on Ambient Intelligence*, pages 290–307, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] N. P. M. Meijers, S. Zlatanova. 3D Geo-Information Indoors: Structuring for Evacuation. In *Next generation 3D City Models*, pages 21–22, 2005.

[15] M. Raubal. Human wayfinding in unfamiliar buildings: a simulation with cognizing agents. In *International Conference on Spatial Cognition: Scientific Research and Applications (ICSC 2000), Cognitive Processing - special issue 2000*, pages 363–388, Rome, Italy, 2000. Pabst Science Publishers.

[16] M. Raubal and M. Worboys. A Formal Model of the Process of Wayfinding in Built Environments. In *Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science, International Conference COSIT '99, Stade, Germany*, volume 1661, pages 381–399. Springer, 1999.

[17] J. Sakamoto, H. Miura, N. Matsuda, H. Taki, N. Abe, , and S. Hori. Indoor Location Determination Using a Topological Model. *Knowledge-Based Intelligent Information and Engineering Systems*, 3684:143–149, 2005.

[18] A. Slingsby. A Layer-Based Data Model as a Basis for Structuring 3D Geometrical Built-Environment Data with Poorly-Specified Heights, in a GIS Context. In *10th AGILE International Conference on Geographic Information Science*. Springer Berlin Heidelberg, 2007.

[19] A. Slingsby and J. Raper. Navigable Space in 3D Models for Pedestrians. In *Advances in 3D Geoinformation Systems*, pages 49–64. Springer Berlin Heidelberg, 2008.

[20] E.-P. Stoffel, B. Lorenz, and H. Ohlbach. Towards a Semantic Spatial Model for Pedestrian Indoor Navigation. *Advances in Conceptual Modeling - Foundations and Applications*, pages 328–337, 2007.

[21] C. N. Thomas Becker and T. H. Kolbe. A Multilayered Space-Event Model for Navigation in Indoor Spaces. In *Advances in 3D Geoinformation Systems*, pages 61–77. Springer Berlin Heidelberg, 2009.

[22] S. Thrun and A. Bücken. Integrating Grid-Based and Topological Maps for Mobile Robot Navigation. In *13th AAAI National Conference on Artificial Intelligence*, pages 944–950, Portland, Oregon, 1996.

[23] R. Urs-Jakob. *Wayfinding in Scene Space: Transfers in Public Transport*. PhD thesis, University of Zürich, 2007.

[24] S. Werner, B. Krieg-Brückner, and T. Herrmann. Modelling Navigational Knowledge by Route Graphs. In *Spatial Cognition II*, pages 295–316. Springer, 2000.

[25] W. Yuan and M. Schneider. iNav: An Indoor Navigation Model Supporting Length-Dependent Optimal Routing. In *13th AGILE Int. Conf. on Geographic Information Science*. Springer, 2010.