Efficiently Using Bitmap Floorplans for Indoor Navigation on Mobile Phones

Martin Werner
Mobile and Distributed Systems Group
Ludwig-Maximilians-Universität München, Germany
Email: martin.werner@ifi.lmu.de

***Abstract*—Pedestrian navigation applications and especially indoor navigation applications need to describe a way in a form that is easy to understand, remember and follow. While in the case of outdoor vehicle navigation the distance in meter together with a turn hint is a good description such a description is merely useless for the indoor area. Pedestrians do not have a good understanding of distances and turns are not always well-defined in the indoor area. Landmarks seem to bring the solution to this problem. With this paper we define an algorithm to identify relevant landmarks and a general algorithm enhancing the visualisation quality of an indoor way using bitmap floorplans. This opens the door for automatically generated waypoint graphs with bad visualisation properties to be used for a multimedia indoor navigation application. Furthermore the algorithms are implemented on a smartphone and results on the final performance are given. The main contribution of this paper is the reformulation of some indoor navigation tasks as image processing tasks.**

***Keywords*-Navigation; Image Processing;**

I. INTRODUCTION

In the last decade, many new multimedia services have been designed. These include numerous applications from the field of location based services. Furthermore the ongoing trend towards mobile computing and the immense development in the field of cellular phones leads to more and more context-information that can actually be used. While it is relatively easy to provide location based services for the outside area it is relatively difficult to do the same for the indoor area. For the indoor area there are many problems, that are solved outdoors. The first problem is the availability of digital map data. While for most outdoor location based services the map functionality offered by major Internet companies (e.g., Google Maps) is enough, there is no comparable service for the indoor area. There are several reasons behind: The complexity of indoor maps would be much higher than outdoors (different floors, differences in the treatment of free space). Moreover the contents of indoor maps is often protected by intellectual property rights of architects. Finally, there is no cheap positioning technology available, which makes indoor location based services attractive.

Nevertheless indoor navigation is a very promising technology. People want to have technological support for indoor orientation, which is comparable to the outdoor situation. Moreover there are several buildings that are either very complex (industrial buildings) or not known to the majority of guests (foreign airports, exhibitions) or not known exactly enough for safety services (firefighter, police, etc.). The most difficult task for indoor navigation is of course the positioning of the users. But in the last decade many promising technologies are under development, which will solve this problem completely in the near future. These include classical signal-strength methods based on Wireless LAN [1], [2] or more specialised approaches based on UWB [3] as well as advanced statistical treatments of this measurement data [4], [5].

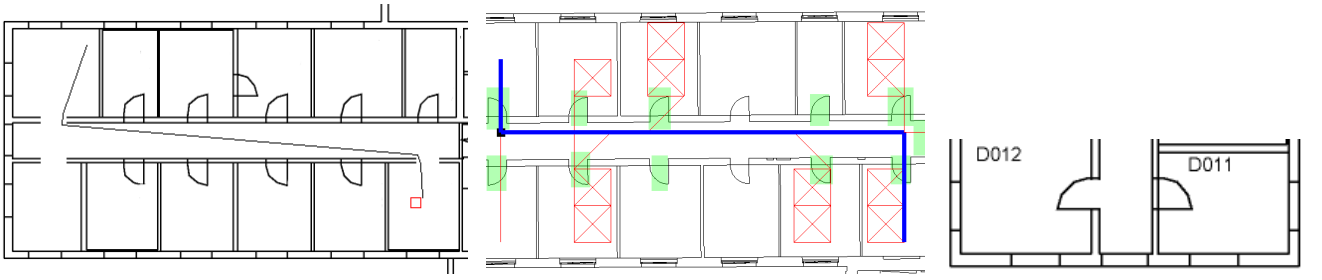
Cellular phones are becoming more and more powerful in terms of calculational power as well as in terms of sensing capabilities. A complete integration of all data that a modern cellular phone can sense will lead to indoor positioning with acceptable accuracy in the future.

Often indoor navigation has been implemented in a relatively small area due to the focus on positioning technology. In such small settings it is not really important to have efficient algorithms for several tasks. However large-scale indoor navigation is an upcoming topic [6].

With this paper we want to show how the reformulation of some subproblems of indoor navigation into image processing tasks is possible and that modern smartphones are able to run this type of algorithms in acceptable time. Note that the results of these algorithms are useful not only for indoor navigation but also for other context-aware indoor location based services and ubiquitous computing trends.

We are focussing on two very complex tasks an indoor navigation system has to accomplish. The first task is the transformation of a shortest way into an augmented form, which is easy to understand, remember and follow. Essentially we are talking about how to transfer a line-strip into a sequence of textual instructions and about how to make several decision points in the navigation solution easy to remember. In this paper we present a parallelisable image processing algorithm, which finds all landmarks that can be used to augment a given way. A good treatment of how landmarks can be used to aid indoor navigation is [7]. We are not talking about how to actually use this landmark information subset for textual instruction generation, because this would go beyond the scope of this paper and is really indoor navigation specific, while the other algorithms have a more general scope.

 ISBN: 978-1-61208-140-3



(a) A shortest way inside a corner graph

(b) A shortest way inside an accessibility graph

(c) Examples of different doors. The left door is not legal with respect to our properties while the right door is legal

Figure 1. Examples of waypoint graphs and problems

The second task, which we accomplish by a sequence of image processing tasks, is to enhance the visualisability of a given way. Navigation in general is often based on a waypoint graph [8], [9]. A waypoint graph consists of a set of waypoints and an edge in between those waypoints, which are directly reachable from each other (usually on a straight line of walking). As the impact of the graph size on the performance of shortest path algorithms is high, people have studied algorithms generating small graphs. However the reduction of the number of waypoints in a graph leads to fewer ways inside the graph. Hence the shortest way inside the graph is not a way, that a human being would choose. One of the smallest and most efficient graphs is for example the corner graph. The corner graph is a graph, which contains all corners of the building (map) as vertices and an edge between two corners if and only if the direct line inbetween these two points is inside the building and free (walkable) space. The resulting ways tend to scrape along walls as can be seen in Figure 1(a). Another well-behaved form of a navigation graph consists of a subset of a grid of waypoints in navigation space defined by some properties (such as a minimal distance to a wall and being inside the building) and edges between waypoints if there is free space inbetween. A shortest way inside such a graph can contain some flaws as can be seen in Figure 1(b). In this example the relatively large grid size leads to the effect, that the chosen way is too near to the left wall. Note, that a finer grid does not help because a finer grid finally leads to a situation comparable to the corner graph. This type of problem is exactly what we want to remedy with the second algorithm in this paper.

When it comes to indoor navigation it is essential to fix a model of the surroundings. It is important to define exactly in terms of available data, what is meant when we are talking for instance about a room or a door. In the following chapter we give possible definitions for this in terms of images. Of course there are other views (especially concerning GIS databases), which can be more flexible, but our choice is made on the background that good GIS data does seldom exist for the indoor area while a basic floorplan (possibly scanned from a building blueprint) is almost always available. Furthermore cellular phones have more difficulties with handling vector GIS-data and the associated queries than with handling and manipulating bitmaps. Moreover the standardisation of bitmap file formats allows for flexible and sustainable treatment of indoor navigation data.

In the following section, we fix an environmental model in terms of bitmap floorplans, in Section III we explain the problem of finding relevant landmarks and describe an algorithm solving this problem. In Section IV we describe an algorithm to enhance the visualizability of ways, which also clarifies turning points useful for textual description of the way. We then explain an implementation of this algorithm for mobile phones and give experimental results on the performance. In Section VI we finalize this paper with a conclusion.

II. The Indoor Environmental Model and its Association with Bitmaps

For this paper we want to describe now exactly our environmental model and how it is setup with different bitmaps. Starting with a building (or site) in some reference system, we define the bitmap projection by first projecting the building information into an orthogonal coordinate system (if it is not given in an orthogonal coordinate system) and map the bounding box of the building to a bitmap by a choice of pixel size. In our experiments, pixel sizes of 0.15m x 0.15m seem to work fairly well.

For this bitmap floorplan we start with some definitions, which do not exactly resemble the definitions of common agreement.

Definition 1: An **area** is a subset of available pixels.

Note that an area need not be connected or otherwise have properties, which the term area describes in other contexts.

Definition 2: The **floodfill-closure** of a point (x, y) is the area inside the bitmap that would be filled by a usual 4-neighbour-floodfill operation at (x, y) with a colour that is not used elsewhere in the bitmap. The 4-neighbour-floodfill operation is defined to fill a pixel if it is of the same

 ISBN: 978-1-61208-140-3

colour as the start pixel and in this case continues with the neighbouring pixels above, left, right and below.

Definition 3: An **area** is called floodfill-connected if and only if it is the floodfill-closure of one (and hence any) of its points.

Definition 4: A **room** is a floodfill-connected area.

Definition 5: The **outside-space** is the floodfill-closure of the pixel coordinate (0,0).

Definition 6: A room R_1 is **inside** another room R_2 if and only if R_1 is in the convex closure of R_2.

Of course this definition does not recover the usual term of a room is inside another room. But as it is unlikely that a room that is contained in another room in our sense is not reachable easyily this discrepancy to the real world is not severe.

With these definition in place we formulate some properties that a floorplan should have for the application of our algorithms below:

- *Closed-Building-Property*: The building is closed by black lines and completely surrounded by white space. This essentially means, that the outside space resembles the usual definition of what is outside a building.
- *Doors-Are-Rooms-Property*: A door inside the building is drawn such that it itself is a room in the sense of Definition 4. An example of a legal door and an illegal door is given in Figure 1(c).
- *Walkable-Space-Is-Known*: There is an oracle telling us whether a straight line between two points lies in walkable space (which is defined to be the space where a human being can walk).

For the rest of the paper, we assume that we are given a bitmap - simply called floorplan in the sequel - where these properties hold.

III. Landmark Search

Pedestrian navigation results need a different presentation form as compared to vehicle navigation results. While for a vehicle navigation system based on GPS the typical errors of the positioning system do not have much influence on the identification of a turn and the orientation of the vehicle is known by the orientation of the street, this is not true for the indoor area. We usually have positioning systems with low accuracy and different possible turns within this accuracy. Typically the positioning errors do not allow the distinction of two doors, which are directly next to each other. In this situation, we want to augment a navigation solution with semantical information such that it is easy to remember and follow. For this the concept of a landmark is often used. Landmarks are objects in the surroundings having a local uniqueness and being eye-catching. All classical signs are landmarks in this sense. But even shops and plants can serve as landmarks. With landmark information the problems of coarse positioning can be reduced. If landmarks are drawn into a map in form of a pictogram (e.g., the logos of the shops) the relation between the proposed way and those shops can be easily remembered and used for difficult way decisions. This role of a landmark is best explained by the following textual instruction, which can be generated if good landmark information is available: ”Before the post office turn left. Then you will see a red sculpture in 300m distance.” As you can see from this sentence, landmark information can be used to make explicit the position of a turn in relation to semantical information (as opposed to geometric information). Landmark information can also be used to enhance the confidence in having a good orientation as you can see from the second sentence.

The difficult task is now to reduce the set of landmarks (which is usually very big) to the set of landmarks that are visible from the way to facilitate more complex election algorithms for the actual integration of landmark information into visual and textual representations of the way.

The following algorithm is a good symbiosis of a search technology with a geometric enhancement technology. The results of this algorithms is an image containing a set of visible landmarks identified by a colour convention. It is possible to very quickly extract this information as a list of visible landmarks or to directly integrate the graphical result into the visualisation pipeline. One could for example highlight the visible space, draw a pictogram over visible landmark positions and so on.

A. Landmark Search By Image Processing

With the following algorithm we solve the problem of finding relevant landmarks out of a list of landmarks visible from within a way. Common algorithms to solve this problem are more or less searching for landmarks by checking whether a given landmark is visible. As there is no good geometric ordering of landmarks (i.e., reducing the search space by a distance limit will miss good landmarks in long rooms) it is not easy to do such a search efficiently. This type of search problem also shows up in other problems of ubiquitous and context-aware computing.

A landmark in the sense of the following algorithm consists of a pixel coordinate and a connected information (identification in a database, etc.). The input of the algorithm consists of

- A set of landmarks
- A floorplan
- A way (given as a list of points forming a line-strip)

The configurable parameters influencing this algorithms are

- The maximal viewing distance
- The length threshold used during tessellation of the way

1) Step 1: Prepare Landmark Map: The very first step is to overlay our floorplan with drawn landmark locations. Therefore we use a colour palette mapping a colour (that is not used in the floorplan) to the identification data. This mapping is symbolised in the following pseudo-codes by the function `landmark_to_colour(landmark l)`.

```
void draw_landmarks()
copy (floorplan, landmark_map)
for each landmark l int landmark_set{
putpixel(landmark_map,
         position,
         landmark_to_colour(l))
}
```

This step is of course general and the result can be cached for subsequent applications of this algorithm.

2) Step 2: Tessellation of the Way: As it is computationally very expensive to calculate the set of pixels, which are visible from a line-strip, we approximate this set of pixels by the set of pixels visible from the points of a tessellation of the line-strip. To obtain this tesselation, we keep inserting middlepoints between two subsequent points until the distance between all points is shorter than the *tessellation length*.

```
global tessellation_length
void tessellate(linestrip l)
for each segment s of l{
if (s.length() > tessellation_length)
{
    s.split()
    return tessellate(l)
}
}
```

3) Step 3: Calculate the mask bitmap: In this step we calculate a mask, which resembles the set of pixels visible from the way. This is done by preparing the mask bitmap to be of the same size as the floorplan and filled with black. We then use a radial floodfill operation starting at each point of the tessellated way and copying every examined pixel into the mask bitmap.

```
void calculate_mask()
for p in tessellated_way{
   radial_flood_fill(p);
}
```

The radial floodfill algorithm is an algorithm, which fills out the area surrounding a point as long as there is a direct line between each point and the starting point. Note that the resulting area need not be convex.

4) Step 4: Multiply the landmark map with the mask: In this step, we stamp the visible area out of the landmark map. For each black pixel in the mask bitmap, we black out the same pixel in the landmark map.

```
void mask_out()
for each (x,y) in landmark_map{
if (mask_map(x,y) != black)
   result(x,y) = landmark_map(x,y)
}
```

This results in a bitmap containing exactly the visible landmarks together with their geometric location. It is now up to the rest of the visualisation pipeline how to work further. One could quickly scan the image and get a list of visible landmarks or one could just overlay all landmark pixels with a pictogram assigned to the landmark.

Though this algorithm seems to be very complex, it has some beneficial properties. First of all it is a formulation of the landmark problem in terms of basic image processing. Secondly its result is near to a complete visualisation of the landmarks and thirdly it is highly parallelisable and designed to be run on dedicated graphics hardware. Examples for an application of this algorithm are given in Figure 2.

IV. Post-Processing Ways

Waypoint graphs are either large or do not contain nice-looking ways. And even if a waypoint graph contains all nice-looking ways, these ways will not be the shortest. So for an indoor navigation system it is difficult to find ways, which are short and at the same time have a specific quality with respect to visualisation. As the efficiency of search algorithms is tightly coupled with the number of vertices and edges inside the graph, it is common that people try to have small waypoint graphs. Shortest ways in such graphs tend to scrape along walls or seem otherwise unnatural.

With our algorithm, we want to post-process such ways to obtain a relatively nice visualisation with acceptable computational overhead.

A. An Algorithm for Post-Processing Ways for Better Visualisation

Therefore we propose the following algorithm, which essentially is a series of image processing operations. The input of the algorithm consists of

- A way (given as a list of points forming a linestrip)
- A collision map

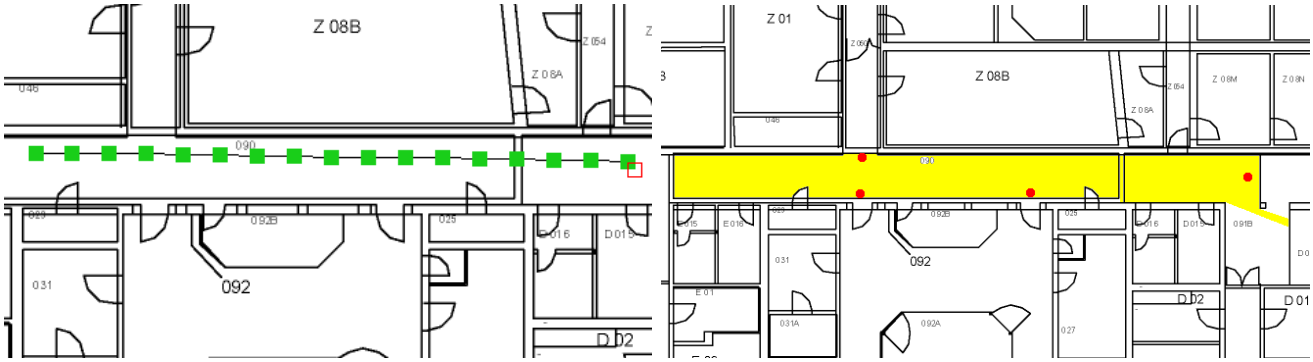
The configurable parameters influencing this algorithms are

- The maximal length a point may move during the algorithm
- The length threshold used during tessellation of the way
- The valuation choosing the best movement in a set of possible movements

1) Step 1: Tessellation of the Way: For this algorithm we need a tessellation of the way just as for the previous algorithm. The reader is referred to Section III-A2 for details.

2) Step 2: Move the tessellation points: For each point in the tessellation of the way determine a set of points, where we could move this point. Therefore we grow a circle until this circle collides and if this circle collides, we move the middlepoints away from the collision points until we get stuck. Hence we find out the position where - in a limited

 ISBN: 978-1-61208-140-3



(a) The tessellation of a way

(b) The mask and landmarks (large dots) found

Figure 2. The results of the Landmark Search Algorithm

neighbourhood of each tessellation point - the biggest circle fits into free space and move this tessellation point to this position.

The following pseudo code illustrates this step. We found out in experiments, that a maximal movement distance of four times the *tessellation length* makes sense for relatively fine tessellations.

```
global tessellation_length
for each point p {
  res = grow_a_circle(p,
          tessellation_length * 4)
}
```

The method `grow_a_circle` is just scanning possible positions and calculating the maximal circle in free space centered around these positions. Each position is then assigned a valuation composed out of the radius of the circle and the distance of the movement. In the examples throughout this paper, we just used a valuation, which prefers the biggest circle in the allowed space and inbetween all those circles with maximal radius the one, which is nearest to the original point.

```
void grow_a_circle(point p, double d)
last_result = p;
for  q in box (p-(d,d), p+(d,d)) {
r = maximal_radius(q);
if (is_better_than_last_result(q,r)){
last_result = p;
}
}
```

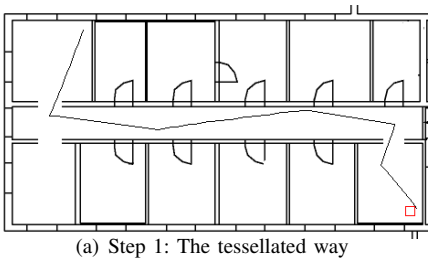
The results of this algorithm are given in Figure 3. As you can see, this algorithm leads to a fairly good way. We intentionally left a problem in the room, where the journey ends. The algorithm is trying to keep away from the black box (it might be a desk). The intention is to stress, that we need a really correct map of walkable space and to emphasise, that small disturbances can have severe influence on this algorithm. What is not obvious but has been tested with a multitude of other ways is the fact, that the algorithm has the beneficial side effect of having a relatively clear turn (e.g., a turn of merely exactly 90 degree in the figure above) exactly where a turn should be indicated by a text generation engine. Furthermore due to using a rotation-invariant definition of good way, the orientation of the rooms inside the bitmap is not important.

V. An Implementation of PPW for Mobile Phones

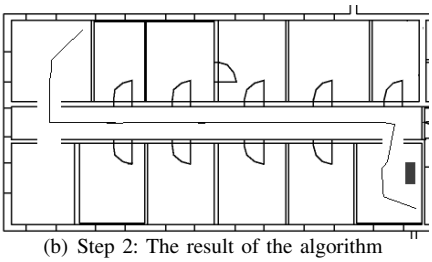
The algorithms presented in this paper are relatively complex. If we apply these algorithms to long ways, the uniform tessellation algorithm leads to many points in the tessellation and for each of those points another complex operation is needed. To enhance clarity, we decided to explain the algorithms in the most simple form given above. Of course the growing circle algorithm can gain a real performance boost from not growing the radius one pixel at a time. Starting with an exponential growth of the radius and correcting the first collision by a nested interval algorithm in comparison to the last non-colliding circle will gain much speed. Moreover for plans where the magnitude of rooms is constructed from parallel lines and the number of randomly placed obstructions is small, the circle can be replaced by a square without any harm. Using integral images in this case makes the question, whether a square collides computable in constant time (not depending on the size of the square) gaining even more speed. If we can afford the memory and the map does not change too often, we can even compute the maximal radius for each pixel and store it as a color component value inside another bitmap. In this case (ignoring the time of constructing this map) we can omit the process of growing a circle and concentrate on the movement of the center point.

We implemented the PPW algorithms for modern smartphones running Android OS 2.2 and above. The Post-Processing Ways algorithm (Section IV) is running fast enough. The system is rendering a map into a screen buffer

 ISBN: 978-1-61208-140-3



(a) Step 1: The tessellated way



(b) Step 2: The result of the algorithm

Number of Points	Running Time
5	0.759s
9	0.985s
17	1.633s
33	2.955s
65	5.623s

(c) Performance on HTC Desire

Figure 3. The Post-Processing Way algorithm and its performance running on HTC Desire (complete screen buffer)

(a Java bitmap of the exact pixel size of the screen), which is then passed to our implementation of this algorithm via Java natives. In Java natives, we are performing the image processing as described on a 16-bit-per-pixel bitmap (using essentially 5 bits per color). Therefore we implemented a fast and stable bitmap manipulation library. The tesselation is being performed in Java. Experimental performance results for the Post-Processing Ways algorithm are given in Figure 3(c).

As you can see, for moderate numbers of tesselation points the running time of the algorithm is quite acceptable. The algorithm has to be run only once for each navigation result. As the effective screen resolution of a full-screen application (not drawing over the status bar) on the device is 480x725, the number of tesselation points to consider will not exceed 20 points. As mobile devices are able to run this type of algorithms natively, we are able to provide full navigation functionality with navigation graphs, which have relatively bad visualisation properties such as corner graphs.

VI. Outlook

The feasibility of this algorithm has been shown with corner graphs as well as with grid-based accessibility graphs. The corner graph example from the introduction has been used in the description of the algorithm in Section IV.

As all algorithms in this paper perform a specific independent task (e.g., radial floodfill or growing circles) on many elements (the points of the tessellation), they are perfectly suited for implementation on modern many-core architectures such as NVIDIA CUDA [10] or the upcoming platform independent OpenCL standard [11]. In these many-core environments hundreds of cores are ready to perform a specific task on a stream. The only precondition for using such architectures is that the parallelised operation is completely independent from each other in the sense that the ordering of execution is arbitrary. This type of execution does of course only apply to server-hosted navigation systems unless these many-core architectures are available on smartphones.

This work allows us to follow a new philosophy for indoor navigation graph generation. As the visualisation quality of the graph is less important if our algorithms are applicable, we can drop this requirement and generate and readily use minimal graphs such as the corner graph for navigation.

References

[1] Y. Chen and H. Kobayashi, "Signal strength based indoor geolocation," in *Proceedings of IEEE International Conference on Communications (ICC '02)*, vol. 1, 2002, pp. 436 –439.

[2] F. Evennou and F. Marx, "Advanced integration of wifi and inertial navigation systems for indoor mobile positioning," *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 164–164, uary.

[3] C. Falsi, D. Dardari, L. Mucchi, and M. Z. Win, "Time of arrival estimation for uwb localizers in realistic environments," *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 152–152, uary.

[4] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174 –188, feb. 2002.

[5] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation and tracking," in *IEEE Transactions on Signal Processing*, vol. 50, no. 2, 2002, pp. 425–437.

[6] P. Ruppel, F. Gschwandtner, C. K. Schindhelm, and C. Linnhoff-Popien, "Indoor navigation on distributed stationary display systems," *Computer Software and Applications Conference, Annual International*, vol. 1, pp. 37–44, 2009.

[7] A. Butz, J. Baus, A. Krüger, and M. Lohse, "A hybrid indoor navigation system," in *IUI*, 2001, pp. 25–32.

[8] P. Tozour, "Search space representations," *AI Game Programming Wisdom 2*, vol. 2, pp. 85–102, 2004.

[9] K. Yu, "Finding a natural-looking path by using generalized visibility graphs," in *PRICAI 2006: Trends in Artificial Intelligence*, ser. Lecture Notes in Computer Science, Q. Yang and G. Webb, Eds. Springer Berlin / Heidelberg, 2006, vol. 4099, pp. 170–179.

[10] NVIDIA, *NVIDIA CUDA Programming Guide 2.0*. NVIDIA, 2008.

[11] K. O. W. Group, "The opencl specification, version 1.0.29," Web site: http://khronos.org/registry/cl/specs/opencl-1.0.29.pdf [Last accessed: 13 October 2010], 2008.

 ISBN: 978-1-61208-140-3