Generating sparse navigation graphs for microscopic pedestrian simulation models

Angelika Kneidl[1], André Borrmann[1], Dirk Hartmann[2]
[1] Computational Modeling and Simulation Group, TU München, Germany
[2]Siemens AG, Corporate Technology, Germany
kneidl@tum.de

Abstract. For our contribution we describe the extension of a microscopic model for pedestrian simulation that introduces hybrid navigation strategies based on graphs. To do so, we introduce a navigation graph layer serving as the basis for different routing algorithms to map individual pedestrian behaviours. The automatic generation of a reduced visibility graph to be used as a navigation graph is described in detail. The paper is concluded with a comparison of simulations without a navigation graph and simulations with the graph layer as an extension, where a dynamic routing algorithm is applied.

1 Introduction and Related Work

The simulation of pedestrian crowds has been widely examined using different approaches that focus on different details depending on the objective of the simulation (Schadschneider et al. 2009): To determine minimum evacuation times for buildings or areas, macroscopic models are typically used. These focus on the overall situations of the simulated scenario and calculate mean values. Examples of such models are network flow models (Hamacher & Tjandra 2002), fluid-dynamic models (Henderson 1974) or lattice-gas models (Jiang & Wu 2007). To simulate the individual behavior of pedestrians, microscopic models have been developed. These models consider the movements of each individual and focus on the interaction between individuals. Force models (e.g. Social Force Model by Helbing & Molnár 1995) as well as cellular automata (Burstedde et al. 2001) or agent based models (Ronald, Sterling & Kirley 2007) belong to this category.
One central aspect of microscopic pedestrian simulation is to simulate different movement strategies of individuals. In order to assign individual behavior to pedestrians, agent-based models are common. These assign different navigation strategies to each individual which results in different movement behavior (Reynolds 1999). As these models have to calculate the new position of each pedestrian according to a complex set of rules in every time step, they are very computation intensive. Another possible way to assign individual behavior is to use a navigation graph with different routing algorithms according to the individuals' preference.
A variety of techniques have been developed to create a navigation graph from a given topography. One technique is to divide the space with Generalized Voronoi Diagrams (Aurenhammer 1991) and to use the resulting lines as graph edges and the intersection points of the lines as graph nodes. The resulting graph consists of edges which are equidistant to each obstacle. Another possibility is to derive a visibility graph (Choset 2005). This graph consists of vertices given by sources, destinations and all vertices of all obstacles within a scenario. Two nodes are connected if they are in line-of-sight. To avoid redundant edges a

reduced visibility graph can be constructed by categorizing edges into supporting and separating edges (Choset 2005).

2 Model setup

For our project we have developed a training simulator for security staff of major events. As a test scenario the evacuation of the surroundings of a soccer stadium was chosen. Due to the objective of providing a training environment, an important requirement was that the simulator is able to run in real-time. To achieve the necessary high performance, we chose a cellular automaton model for space discretization in combination with a conservative force model (Klein, Köster & Meister 2010), i.e. a model based on energy potentials that describe the influencing forces on each pedestrian (attracting force of the destination, repellent forces of obstacles as well as the repellent forces of other pedestrians). This combination allows us to update the pedestrians' positions very quickly whilst taking into account the interactions between the pedestrians. However, when using this approach, a key aspect of pedestrian movement is neglected, namely the individual navigation behavior of pedestrians. In order to take into account their knowledge about efficient routes towards their destination as well as individual decisions according to certain criteria (e.g. keep as close as possible to direction to destination, routes with less turns etc) without losing computational speed, we extended the basic model with a navigation graph. Using this graph, different route choice behaviors can be implemented and thus different pedestrian types realized (e.g. pedestrians who are / are not familiar with a particular environment). The navigation graph is based on the concept of visibility graphs. However, our method differs from the method described in (Choset 2005). We first constructed orientation points that will serve as vertices and then connected these vertices according to criteria depending on the location of the sources and destinations of a scenario.

3 Construction of a navigation graph from a geometry

During the simulation, pedestrians are routed from vertex to vertex on the navigation graph, until they reach their destination. Thus, one requirement for the navigation graph is that the graph maps the real routes of pedestrians as closely as possible. The Generalized Voronoi Diagram (GVD) forms a graph with equidistant edges from each obstacle. If we take a GVD as a navigation graph, pedestrians would walk equidistant to all obstacles. This would result in large detours. With visibility graphs, the resulting routes are close to obstacle corners and thus map more realistic routes. Hence, we rely on visibility graphs as navigation graphs.
The derivation of the visibility graph from a given geometry is achieved in multiple steps, which are described in detail below.

3.1 Derivation of orientation points (vertices)

We start with examining the geometry of a given simulation scenario and placing orientation points around each obstacle at each convex corner. Each of these points will correspond to a vertex in the visibility graph. Depending on the obstacles' geometry, an overlap of two neighboring orientation points may occur. Therefore a pruning strategy is applied to merge close orientation points into one resulting point. Another criterion for a valid orientation point is that there is no obstacle on the imaginary sight line between the point and the corresponding obstacle corner. Hence we apply a second check for this criterion and re-locate

these points in such a way that they fulfill this criterion. The exact algorithm is described in (Höcker et al. 2010) and (Kneidl, Borrmann & Hartmann 2010).

3.2 Connecting vertices: A cone-based search method

Once we have created all orientation points, these points are connected according to certain criteria. The first criterion is that the two vertices are connected if they are in a line of sight with each other. This is a basic requirement for our routing algorithms. As we want to construct a directed graph, which is as sparse as possible (to keep the runtime of the navigation algorithms low) redundant edges should be avoided. We define redundant edges as edges that form a loop and are geometrically located very close to each other.
In (Kneidl, Borrmann & Hartmann 2010) we suggest using a spatial index, namely an R*-Tree (Beckmann et al. 1990). Using this data structure, nearest neighbors of a point can be found efficiently in a search space, which exists of one- and two-dimensional objects like polygons and points. In (Kneidl, Borrmann & Hartmann 2010) we have taken the three nearest neighbors to connect them with an edge. Unfortunately, this procedure is quite inflexible as there can be vertices having more neighbors that connect a vertex to important directions. One example is illustrated in Figure 1: On the left picture the complete visibility graph is illustrated; the center picture shows the resulting edges from this method: Here, the source would be connected with the three vertices inside the red circles, but the very left and the very right vertex would not be connected. On the right picture the result of our improved method is shown: The connecting edges lead in every direction.

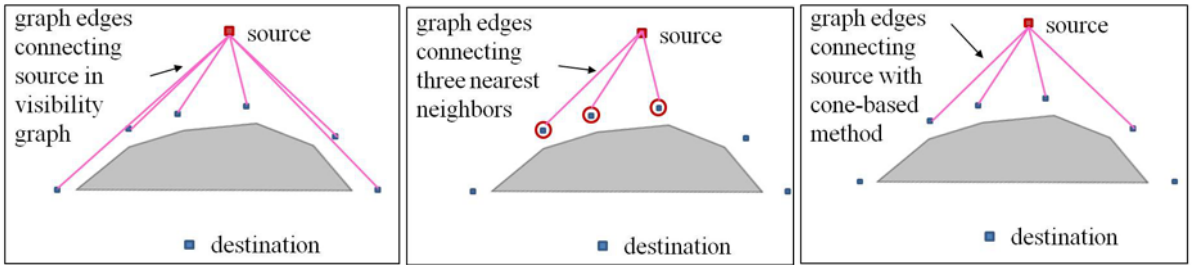

Figure 1: Example for a geometry, where the algorithm presented in (Kneidl, Borrmann & Hartmann 2010) would not find all important neighbour vertices of a *source*: left: a complete visibility graph, middle: connecting three nearest vertices, right: resulting edges for cone-based method

Here, we outline an improvement of the algorithm: a cone-based search for finding the most relevant neighbors to be connected. The basic idea is that the angle between two outgoing edges has to be larger than a certain threshold. If the angle is smaller, we discard the longer edge. The algorithm works as follows: We initialize the graph by inserting all orientation points as vertices. We start with an arbitrary vertex v_i. For this vertex, we define a rectangular search area, such that the inspected vertex v_i as well as all destinations of the given scenario are located inside that area. Furthermore, if any obstacle cuts the imaginary sight lines between v_i and each of the reachable destinations, the search area is extended until it contains these obstacles (Figure 3a). Inside this area we search for all vertices in-sight to v_i and sort them according to their distance to v_i (Figure 3b). An edge is created between the closest vertex and v_i. Starting from v_i, we define a cone-shaped area around the edge with an angle α_{cone}. The search area is updated by removing this cone-shaped area (Figure 3**Fehler! Verweisquelle konnte nicht gefunden werden.**c), i.e. the new search area is given by the old search area minus the cone area. From all remaining vertices inside the new search area we choose the one with the smallest distance to the vertex v. We proceed with this vertex as

described above, i.e. we connect this vertex and remove the corresponding cone-shaped area. The resulting edges of v_i are shown in Figure 3**Fehler! Verweisquelle konnte nicht gefunden werden.**d. The algorithm is repeated for every graph vertex. Figure 2 shows the pseudo code of the algorithm.

Algorithm: Graph construction
Parameters: Set of orientation points; Set of destinations $D \subseteq V$; Obstacles O
Output: Graph G (V, E)
1: Add all orientation points as vertices V to G
2: **For** each vertex v_i in $V \backslash D$ **Do**
3: Define search area A_{search}, such that: $v_i \in A_{\text{search}}, \forall_{d_i \in D} \in A_{\text{search}}$
4: **For** all obstacles o_i in line of sight between v_i and $d_i \in D$ **Do**
5: Expand A_{search}, such that all orientation points of $o_i \in A_{\text{search}}$
6: **End For**
7: Search inside A_{search} for all orientation points in sight $\{n\}$ and sort them according to their distance to v_i.
8: **For** each $n_i \in$ {n} **Do**
9: **If** n_i inside valid area A_{search} **Then**
10: insert edge (v_i, n_i) to G
11: cut cone-shaped section A_{cone_vi} around edge (v_i, n_i): $A_{search} = A_{search} \backslash A_{cone\_vi}$
12: **End If**
13: **End For**
14: **End For**

Figure 2: Pseudo-code for graph generation algorithm

The number of the resulting edges can be varied by changing the value of the angle α_{cone}, which defines the cone-shaped cut areas. The bigger the angle value, the sparser the resulting graph, since larger areas are removed from the search area. As shown in the example above, we chose an angle $\alpha_{cone} = \pi/15$.
The resulting graph provides from each source at least one route to every destination, if any exists. This follows from the construction algorithm: by definition, there is at least one orientation point in line-of-sight in each search area. Thus each vertex is connected to at least one other vertex. This connected vertex refers either to the destination itself or it is connected with a vertex which leads around an obstacle, that is located on the imaginary sight line between this vertex and the destination. Therefore, either we end up with a path from the start vertex to the destination or no connection exists between source and destination. Since the search is directed, the order of the inspected vertices can be chosen arbitrarily. The resulting graph always remains the same.

3.3 Connectivity check

The main goal of the constructed visibility graph is to provide at least one route, which leads from all sources to all assigned destinations. Hence we check if there are any vertices that are not connected to any source or destination. This can occur, because the algorithm inspects every given orientation point (i.e. vertex). These vertices and their corresponding edges can be discarded. To find all these vertices, we search for connected components within in the graph (Gibbons 1999). To find these components, a breadth-first iterator can be used, which starts from each source vertex and checks if at least one destination can be reached. If so, source and destination belong to a connected component and thus all vertices of this

connected component shall be kept. Figure 4 shows an example of a graph, which consists of two connected sets, but only one set contains source and destination, thus the second set can be discarded.

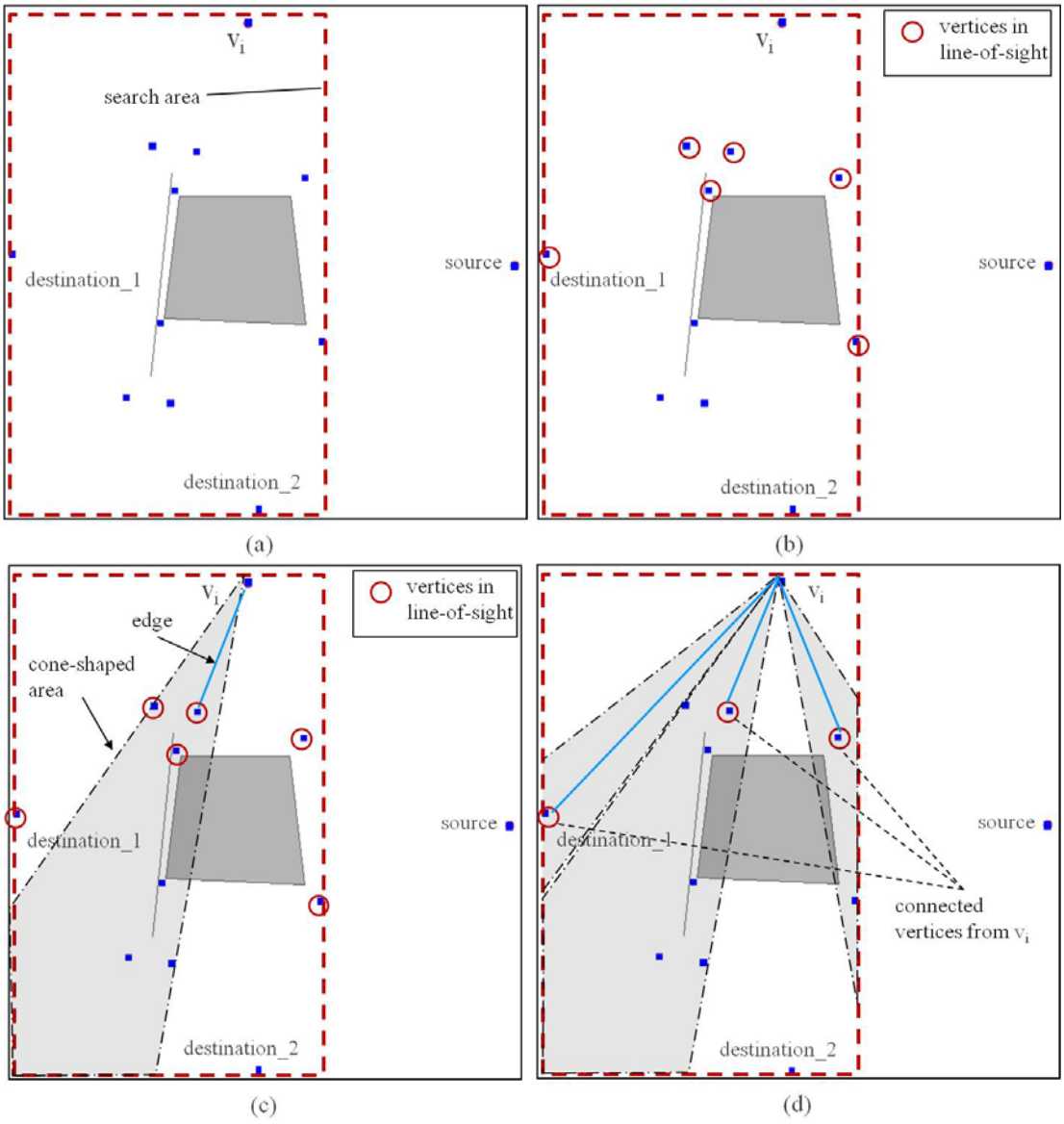

Figure 3: Steps connecting node v_i with neighbours: (a) defined search area (b) all vertices in sight from v_i (c) cone-shaped cut area for excluding vertices in same direction (d) resulting edges from vertex v_i

Note, that by applying this technique the resulting graph is not generic anymore but directly depends on the locations of sources and destinations within the scenario. The advantage of this reduced graph is a better performance in route finding algorithms.

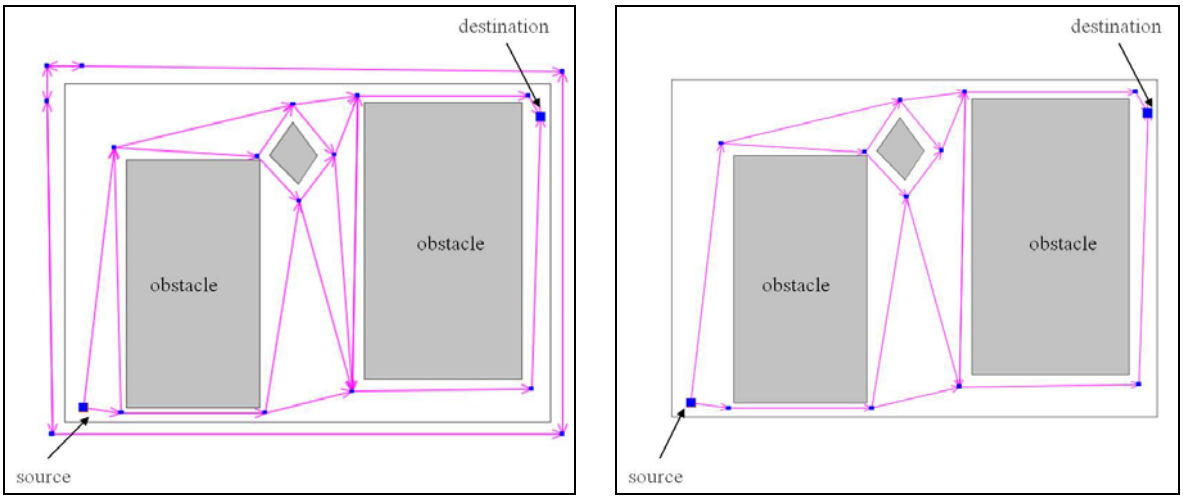

Figure 4: Graph with two connected set (left side) and graph with only one connected set, which contains source and destination (right side)

3.4 Complexity

The complexity to construct this graph is $O(n^2 \log n)$, where n denotes the number of vertices of the graph.
The algorithm which creates and places the orientation points has a complexity of $O(n \log n)$: We examine each corner of every obstacle for the insertion of a possible orientation point. Since the number of corners is at most n (we place one orientation point per corner maximum), this takes n steps. To check whether an intersecting obstacle lies on the sight line between corner and corresponding orientation point, takes worst-case O (log n), since we use a spatial index. Insertion of one vertex into the R*-Tree needs O (log n), which results in O (n log n) for all vertices.
The time required for the edge generation is O (n^2 log n): We have to search for the nearest neighbors inside a defined area (range); this requires O (n log n) (for details, please refer to Beckmann et al. 1990). For each neighbor(R) it is checked if an edge can be inserted (for each neighbor), which requires worst-case (n-1) steps, since all vertices can be located inside the search area.
The connectivity step costs O (n), since the number of sources is significantly smaller than n, and thus can be treated like a constant factor.
In total, the graph construction costs sum up to O (n^2 log n).

4 Comparison of plain simulation and extended simulation using a Navigation Graph

Using this visibility graph as a navigation graph we are able to model individual pedestrian behavior. There are different algorithms like shortest or fastest path algorithms, heuristic algorithms, algorithms based on ant colony approaches that take into account behaviors of other pedestrians. Since definition and validation of individual behavior as well as the description of the algorithms are beyond the scope of this paper, we refer to (Kneidl & Borrmann 2011).
We would rather demonstrate the different results of a simulation without using a navigation graph layer (plain simulation), and a simulation which uses a navigation graph (extended simulation).
In the plain simulation, a pedestrians' position is updated depending on influencing forces, namely the repellent forces of other pedestrians and obstacles as well as the attracting force of

the destination. These forces are superimposed into one potential field. The corresponding value from the potential field is mapped to each cell of the automaton, corresponding to the position of the cell. During simulation, at each time step all neighboring cells are examined for each pedestrian, choosing the cell with the lowest value, if it is not occupied by either an obstacle or another pedestrian. This selection process is conducted until all pedestrians have reached their destination. From this setup it is obvious, that each pedestrian is "short-sighted" and chooses a similar route to walk to his destination. The only varying factor is the number of pedestrians who walk the same way and occupy cells.

In the extended simulation, we choose the Fastest-Path Algorithm, which finds the fastest path for each pedestrian. We calculate this fastest path using the Dijkstra Shortest Path Algorithm (Dijkstra 1959) with dynamic edge weights, taking traveling times instead of distances as edge weights. Hence the assigned edge weights can change over time. We derive this travel time from the density on an edge and the corresponding mean velocity. A detailed description of the algorithm can be found in (Höcker et al. 2010).

The resulting traces of the pedestrians are illustrated in Figure 5. We generate 600 pedestrians walking from the lower left corner to the upper right corner of the room. Without using a graph the resulting way is the same for all pedestrians since there is no possibility of changing the route. The result is shown in the left picture.

In the right picture one can observe the wider spread of routes resulting from the dynamic routing algorithm. In Figure 6, the progress of the extended simulation is illustrated using snapshots of the simulation at different times. In the beginning, the fastest path is identical with the shortest path. After a while, the route becomes too crowded, so much that the route south of the first obstacle becomes faster, as the pedestrians on the shortest route have to slow down according to the density on this route. Even later in the simulation, a third route seems to be the fastest, as the last part of the two former routes are crowded on the last common segment.

This fastest path algorithm does not yet reflect realistic behavior, since it "reacts" too late on crowded areas, but it demonstrates the possibility of routing pedestrians dynamically, depending on occurring events. Improving the navigation strategies with respect to their "reaction times" a realistic behavior is well in reach using the navigation graphs outlined in this contribution.

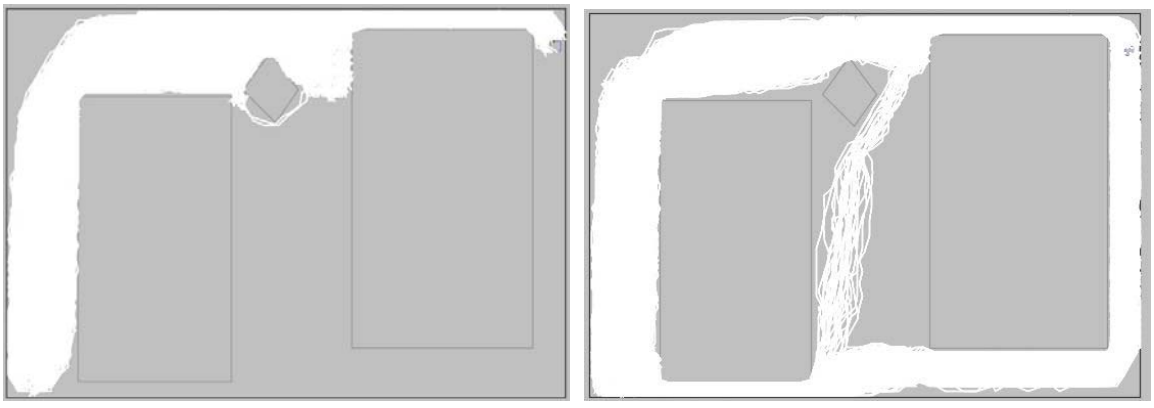

Figure 5: Traces of 600 pedestrians, walking from the lower left corner to the upper right corner. Left without routing, Right with "Fastest Way"-Routing

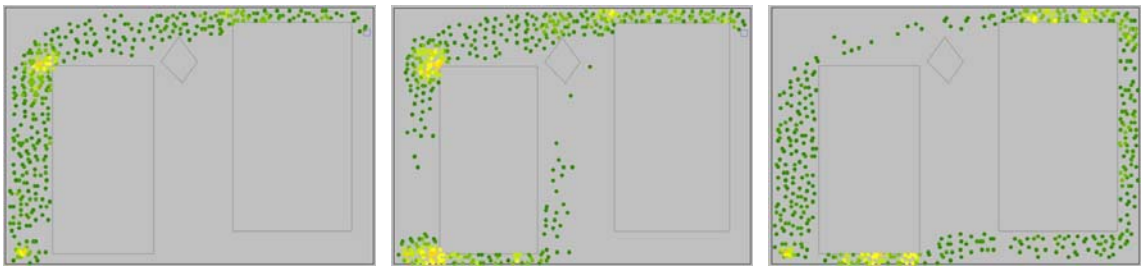

Figure 6: Simulation screenshot after 66 seconds (left), 94 seconds (middle) and 191 seconds (right)

5 Summary

In our contribution, we have described an algorithm that automatically derives a reduced navigation graph from a given geometry. The graph is a subset of the standard visibility graph. It can be used to map individual pedestrian behavior in pedestrian simulations and also to map the large-scale orientation of pedestrians.

The advantage of using such a navigation graph instead of using an agent-based approach is to save computational effort and to efficiently change a scenario during run-time (e.g. closing a door corresponds to deleting an edge).

We have described the construction of a navigation graph from a given geometry, by reducing a visibility graph with a cone-based method search method. The resulting graph is very sparse, since "very close" edges are realized as a single edge in contrast to standard visibility graphs. Furthermore, we discard all vertices, which are not contained in any connected component which holds sources and destinations.

To demonstrate the difference between plain simulation and graph-extended simulation, we showed the results of a simulation example. The applied routing algorithm does not yet reflect real pedestrian behavior, but shows the possibilities of integrating a graph into the microscopic simulation.

To measure the quality of the resulting graph, one next step will be to define a metric and evaluate the navigation graph according to this metric.

Another step will be to integrate different navigation strategies using the reduced navigation graph introduced in this contribution into the pedestrian simulation, so that an individual routing strategy can be assigned to each pedestrian. This will result in a more realistic pedestrian navigation and a heterogeneous distribution on different routes. Thus, a significant increase of the realism of microscopic pedestrian simulations is well reachable in the near future.

References

Aurenhammer, F. (1991). Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. ACM Comput. Surv., vol. 23, pp. 345-405.

Beckmann, N., Kriegel, H., Schneider, R. & Seeger, B. (1990). The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec., vol. 19, no. 2, pp. 322-331.

Burstedde, C., Klauck, K., Schadschneider, A. & Zittartz, J. (2001). Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A: Statistical Mechanics and its Applications, vol. 295, 3-4, pp. 507-525.

Choset, H.M. (2005). Principles of robot motion. Theory, algorithms, and implementation, MIT Press, Cambridge, Mass.

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs, Numer. Math., vol. 1, no. 1, p. 269.

Gibbons, A. (1999). Algorithmic graph theory, Cambridge Univ. Press, Cambridge.

Hamacher, H.W. & Tjandra, S.A. (2002). Mathematical modelling of evacuation problems: A state of the art. Pedestrian and Evacuation Dynamics. International Conference on Pedestrian and Evacuation Dynamics (PED), ed M Schreckenberg, Springer, Berlin, p. 227–266.

Helbing, D. & Molnár, P. (1995). Social Force Model For Pedestrian Dynamics. Physical Review E, vol. 51, no. 5, pp. 4282–4286.

Henderson, L.F. (1974). On the fluid mechanics of human crowd motion. Transportation Research, vol. 8, no. 6, pp. 509–515.

Höcker, M., Berkhahn, V., Kneidl, A., Borrmann, A. & Klein, W. (2010). Graph-based approaches for simulating pedestrian dynamics in building models. 8th European Conference on Product & Process Modelling (ECPPM), University College Cork.

Jiang, R. & Wu, Q. (2007). Pedestrian behaviors in a lattice gas model with large maximum velocity, Physica A: Statistical and Theoretical Physics, vol. 373, pp. 683–693.

Kneidl, A. & Borrmann, A. (2011). How Do Pedestrians find their Way? Results of an experimental study with students compared to simulation results. In: Proc. of EMEVAC - International Scientific and Technical Conference on Emergency Evacuation of People from Buildings, The Main School of Fire Service, Warsaw.

Kneidl, A., Borrmann, A. & Hartmann, D. (2010). Einsatz von graphbasierten Ansätzen in einer mikroskopischen Personenstromsimulation für die Wegewahl der Fußgänger. Forum Bauinformatik 2010, T Krämer, S Richter, F Enge & B Kraft, Shaker, Aachen.

Ronald, N, Sterling, L. & Kirley, M. (2007). An Agent-Based Approach to Modelling Pedestrian Behaviour. Lecture Notes in Computer Science, pp. 145-156.

Schadschneider, A., Klingsch, W., Klüpfel, H., Kretz, T., Rogsch, C. & Seyfried, A. (2009). Evacuation dynamics: Empirical results, modeling and applications. Encyclopedia of Complexity and System Science, pp. 3142–3176.

Reynolds, C. (1999). Steering Behaviors for Autonomous Characters. Game Developers Conference 1999.