University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Terrain Representation for Artificial Human Navigation

Tomáš Vomáčka

Technical Report No. DCSE/TR-YYYY-NN
February, 2010

Distribution: public

Technical Report No. DCSE/TR-YYYY-NN
February 2010

Terrain Representation for Artificial Human Navigation

Tomáš Vomáčka

Abstract

Modern applications in the fields of computer simulation, computer games and others often need to simulate large crowds of people, flocks of birds, schools of fish and other similar groups. Several scientific teams have been researching the behavior of these clusters of entities as well as the environment in which they exist. This technical report brings the state of the art of the most commonly used methods of the space partitioning used together with artificial human simulations.

This work was supported by project No. GAČR 201/09/0097.

Contents

1 Introduction

In modern virtual reality applications as well as in computer games, it is important to be able to simulate human-like behavior for artificial entities. In virtual reality applications, we may for instance need to populate cities with crowds of virtual people to make them seem more realistic. In the game industry, we of course want the environment to look as real as possible (which may as well mean the presence of crowds of people in the environment) but we also want to provide the player with artificial opponents which would be comparable to real human opponents in terms of gameplay and overall behavior. In other applications, virtual crowds may represent a vital part of various kinds of tests – e.g., in traffic simulations.

Although the purpose of the artificial people may vary with each application, the environment used for their navigation does not necessarily need to be different. For the purposes of path-planning and virtual human navigation, the virtual environments used are usually based on various kinds of graph structures and relevant algorithms (see [14, 19] and others) or on particle movement inside potential fields (see [23]).

This report will describe the most commonly employed types of graphs as well as some of the other data structures used for the representation of the virtual terrain that are used for the artificial human navigation in the urban environment as well as in any general terrain. We will address the problem which applications will benefit from which data structure for the crowd simulations and why, describe the type of the path that each of the structures provide, its benefits and drawbacks (can the path be smoothed, is it the shortest path, etc.). And we will provide an overview of the modern trends in general crowd simulation applications and in the game industry.

2 Graph Based Environments

2.1 Regular Space Partitioning

The graph-based structures which use regular space partitioning are represented mostly by various kinds of grids. These grids pose various advantages, perhaps the most obvious of them being that dividing any 2D or 2.5D virtual space (e.g., the streets of a simple virtual city[1]) is extremely straightforward and simple to perform. Furthermore, the result of such a spatial division is a graph, however simple, and thus all the commonly used path-finding algorithms may be employed. On the other hand, the grid technology can only uneasily be used for more complex environments and (in the form of a regular grid) it does not provide us with any tools to effectively describe areas with uneven distribution of details. Although the use of regular spatial partitioning is most often used in 2D or 2.5D environments as said before, there are applications that exploit this type of structure in full 3D (see [10]).

Due to their simplicity, various grid-based path planning methods have been widely used in the computer gaming industry since its beginning. Probably the best known representatives of this terrain representation are such games as *Warcraft*, *Civilization*, *Heroes of Might and Magic* (see [2, 4, 7] respectively) and many others which use a regular square grid; or the first two parts of the *Fallout* (see [5]) series which use a regular hexagonal grid. Other types of grids are not usually used in computer games because these two are the simpliest ones (perhaps apart from dividing the space into triangles) and for historical reasons – inconsiderable part of the first computer games that used such space division originated from pen&paper games such as *Dungeons and Dragons* (see [9]) or *GURPS* (see [6]) which use the same space partitioning. Furthermore, together with the already mentioned triangles, these are the only convex polygons that can be used to tesselate a 2D plane regularly (regular tesselation is a tiling composed of regular polygons symmetrically filling the plane) – see [25]. The examples of use of the hexagonal and square grids are shown in Fig. 1 – in the left part of this figure, we may see a snapshot from the *Fallout* game showing the used hexagonal grid, in the right part, there is a snapshot from *Age of Empires* which use a square grid (the grid itself is not shown but its presence can be easily spotted by the shape of the shadows or the layout of the objects in the figure).

There is one special detail worth noting in the grid problematics – some applications use path planning methods based on a data structure called *path grid*. The problem is that the term is used to mark two completely different things used in a similar context. First, the path grid used in the field of robotics means (simply

[1]Such a virtual city cannot contain bridges over walkable areas, multistory buildings and similar structures.

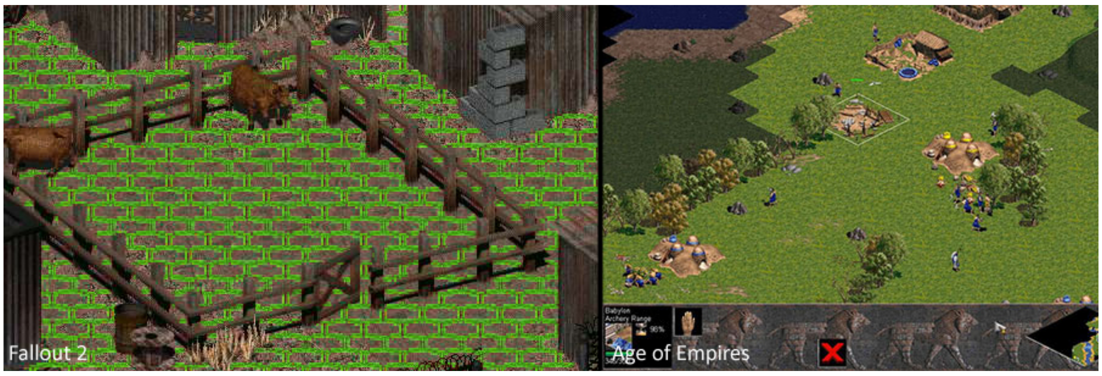


Figure 1: Two examples of applications using a grid terrain representation. Property of [5] and [3] respectively.

said) that the navigation space is divided using a regular square grid and each grid cell containing at least a part of an obstacle is considered to be impassable for any path-planning algorithm. More details may be found for instance in [26]. Second, some game engines (e.g., those used by [1]) use a data structure which is also called path grid for planning the paths of the artificial characters but this structure is in fact a case of waypoint map (see further).

2.2 Waypoint Maps

Waypoint maps (sometimes also called roadmaps) and similar graph structures operate on a very simple principle shown in Fig. 2 – a number of special navigation nodes called waypoints is placed into the environment, representing vertices of a navigation graph. Some of these waypoints are then connected by edges, representing that they are mutualy reachable by the artificial entities. Especially in Fig. 2, we may see a small portion of a cave environment from the *The Elder Scrolls IV: Oblivion* game with the waypoints represented by the blue squares, some of them being connected by the yellow edges. These connected waypoints are then mutualy reachable by the artificial entities inhabiting the cave.

There are numerous ways of constructing a waypoint map – it may be based on a visibility graph (see [13]), some space partitioning structure such as octtree or Delaunay triangulation (see [15]), it may be constructed as a medial axis of the free space in the environment (see [24]) or even entirely constructed by the user (which is one of the most commonly used approaches in the game industry).

If an artificial entity wants to move through the map from its current destination to some other point, it searches the waypoint graph and locates two waypoints – the one that is nearest to its current position (let us mark it A) and the one that is nearest to the destination (let us mark it B). Using some kind of a graph

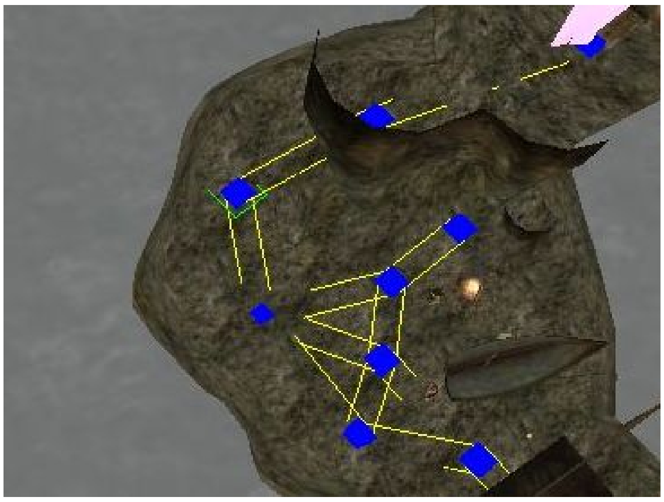


Figure 2: Example of an environment with waypoint graph, [1].

searching algorithm, a path from A to B is then found through the waypoint graph and the entity then moves along this path toward its goal.

Even though this principle may seem to be simple, there are some serious drawbacks which make it hardly usable for crowd simulations. First of them is the fact that for the most of the time the characters move strictly along the edges in the waypoint graph which is extremely unnatural. Even if an artificial character moves through an empty wide-opened space, its path will be "zig-zag" shaped. Furthermore, in its most simple form (as described here) the waypoint map does not allow any trajectory smoothing because it does not contain any information about its neighborhood (on the other hand, the waypoint map itself may be smoothed during its creation – see [24]).

The waypoint graph also allows only one moving character per each edge at a time. If two characters are to be able to move through some space (a stairway, a street, a hallway, etc.) there would have to be multiple edges of the waypoint graph in that area. Otherwise, one of the characters has to wait for the other to leave the edge before entering it. This problem is caused by the same feature of the waypoint graphs as the inability to smooth the paths of the characters – because the edges of the graph contain no additional information about their neighborhood, the characters simply do not know if (and how) they could possibly evade each other if moving along the same edge simultaneously.

If there are two or more different types of artificial entities (which move in a different fashion) we will most probably need a separate waypoint map for each of those entity types because the waypoint map used for one of them will be unusable for the other because it will be too close to the walls of the environment, it will lead through corridors too narrow for some larger vehicles, etc.

Even though the waypoint maps are widely used in the game industry today, for the mentioned reasons they are unsuitable for managing large quantities of artificial humans (i.e., a crowd) and even in the game industry, various producers replace them with more sophisticated data structures that, apart from the information contained in the waypoint maps, also provide the artificial entities with some information about the surrounding environment (see further).

2.3 Corridor Maps and Navigation Meshes

As described before, the commonly used waypoint maps have some unpleasant disadvantages which may make them barely usable for some applications like navigating larger groups of artificial humans or navigation of many different types of artificial entities at once. Most of these disadvantages originate in the fact that the waypoint maps do not describe the surrounding environment. There are two commonly used data structures which remove this drawback and thus do not suffer from the same disadvantages – the corridor maps and the navigation meshes.

Corridor Maps

Let us take a look at the corridor maps first – according to [11, 16] and others, a corridor map represents a system of collision-free corridors for the static obstacles in a given environment. Each corridor consists of a sequence of maximum clearance disks (i.e., maximum disks which contain no static obstacle in the environment) – the centers of these disks form the centerlines of the corridors and their radii provide us with so needed information about the surrounding environment. An example of a corridor map is shown in Fig. 3 (only the centerlines of the corridors are displayed in the left subfigure, the right subfigure also shows one full corridor).

Using the corridor maps for artificial entity navigation is basically very similar to using an ordinary waypoint map. Let us again have an entity which wants to move from point A to point B. First we have to find in which corridors A and B lie and then we use a graph searching algorithm to find a sequence of corridors which connect the first and the last one that the entity will need to traverse (i.e., the ones that contain A and B, respectively). So far the approach is essentialy the same as if we used an ordinary waypoint map, but instead of having the entity move along the centerline of the corridor sequence, we may use the entire space given by the corridors on the path, because we know that they are empty of static obstacles. To do so, we may for instance create an abstract point which attracts the moving artificial entity and move this point along the centerline. Using the attraction point, we create a force which moves the entity

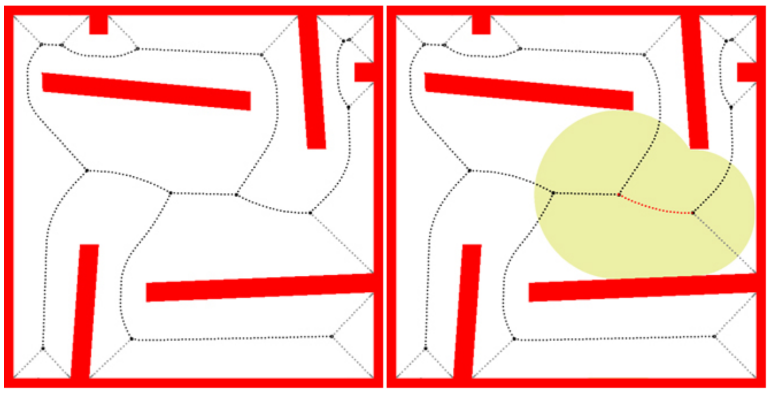


Figure 3: Example of a corridor map and a sample corridor, [18].

towards its destination using possibly the whole space of the corridors en route. Furthermore, this attraction force may be combined with other forces that affect the entity such as grouping forces (which will hold together a group of entities that are supposed to move together, see [14]) or repulsion forces (which will make the entity avoid dynamic obstacles in the corridors such as other pedestrians etc., see [18]) and many other different forces as required by the target application.

Navigation Meshes

The idea behind navigation meshes (also called navmeshes) is basically the same as the idea behind corridor maps – the navigation meshes also represent an extension of waypoint maps but with the idea to keep the environment description as simple as possible (i.e., the aim is on quick construction and low memory consumption). Thus, instead of creating a centerline of the environment and assigning each of its points a radius, we divide the environment into rather small, convex parts – i.e., a mesh. This mesh may be composed of triangles, quadrilaterals (either irregular ones or rectangles), or virtually any convex subparts of the environment – see [20]. Example of a navigation mesh is shown in Fig. 4.

Navigation meshes work essentialy in the same way as corridor maps – they provide us with a data structure that describes the whole environment that the artificial entities can traverse in a form of a graph. Thus any suitable graph-based algorithm for path-planning may be used to find a sequence of navigation mesh primitives that will lead the artificial character from point A to point B. Because the path also contains some part of the surrounding environment, it may be easily smoothed to look more natural. Compared to the corridor maps, the navigation meshes do not provide us with equally quality path because there is no guarantee that the found mesh primitives sequence contains the maximum available surrounding environment and is highly dependent on the type of the

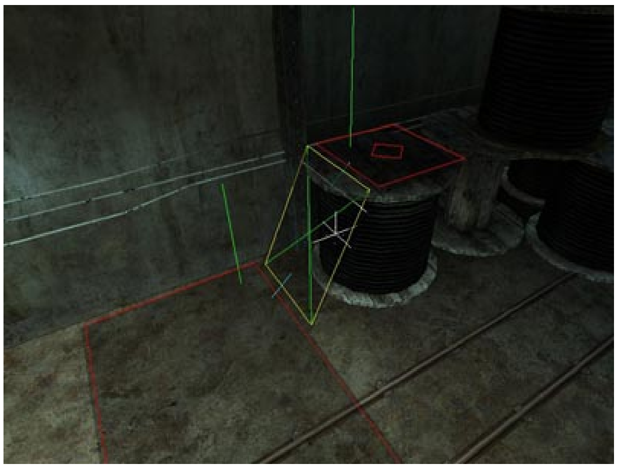


Figure 4: Example of a navigation mesh, [8].

navigation mesh and the construction algorithm. On the other hand, the memory needed for storing a navigation mesh is substantially smaller than the memory needed for corridor maps and if the navigation mesh is well constructed, the found paths will be of good quality.

Similarly as the corridor maps, the navigation mesh itself does not handle any moving obstacles such as other artificial entities. To do so, it has to be combined with some other apparatus such as the attraction and repulsive forces we have already described or for instance kinetic data structures which are described in [12] (in combination with a different terrain representation). In the game industry, the navigation meshes may be found for instance in games such as *Countre Strike: Source* from [8], *Fallout 3* from [1] and others.

2.4 Other Commonly Used Types of Graphs

Of course there are numerous other data structures which may be used for terrain representation in crowd simulations and similar applications that were not discussed here. This section will provide a brief overview of some of them.

Higher Order Voronoi Diagrams

Unlike the ordinary Voronoi diagrams which represent a spatial partitioning of a given space into cells that consist of points nearest to one of the points in the generator set, the k^{th} order Voronoi diagram decompose the given space into cells containing points nearest to $k-$tuples of the generator points (for details on both the ordinary and the k^{th} order Voronoi diagram see [17]).

According to [22], the combination of the 1st and 2nd order Voronoi diagrams may be used to simulate crowd behavior. The relevant subpart of the 2nd order diagram is embedded into each cell of the 1st order diagram, thus creating a graph in which the paths may be found for all of the virtual entities simultaneously (without the need for creating a special Voronoi diagram for each entity which would consider the other entities to be obstacles). Example of the combination of the 1st and the 2nd order Voronoi diagram is shown in Fig. 5 – in the left part of this figure we can see the combination of the two Voronoi diagrams (red edges are those of the 1st order diagram and the black edges are from the 2nd order diagram). The right part of Fig. 5 shows a data structure called Multi-agent Navigation Graph which is used for the navigation itself (see [22]).

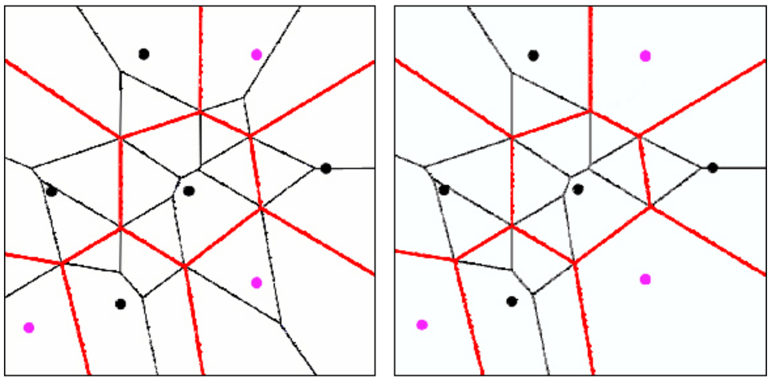


Figure 5: Combination of the 1st and the 2nd order Voronoi diagram, [22].

Visibility Graphs

Although this type of graph is often used as a substep in the process of creating a waypoint map for a given environment, it may also be used for the navigation itself (see [20]). In that case, the visibility graph works exactly as any other waypoint map would. However, some problems may arise from the fact that the edges of a visibility graph usually connect corners in the environment which can make the artificial entities in the environment to move too close to the walls or corners and maybe even collide with them.

Binary Space Partitioning Trees

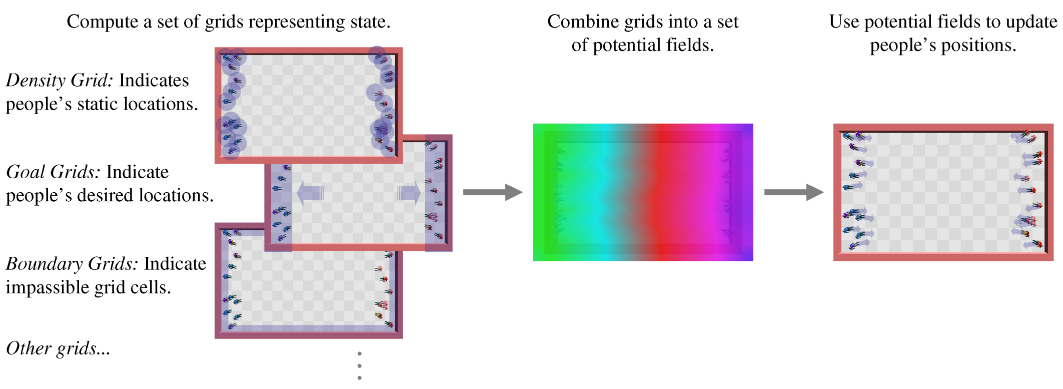
This data structure is most commonly used for rendering acceleration (see [21]), because it allows us to quickly determine which parts of the environment may be seen by the camera. On the other hand, its features allow it to be used for collision detection or even as a base structure for waypoint map or navigation mesh generation. It is, however, not usualy used for navigation purposes.

3 Environments based on Non-Graph Structures

Environments used for artificial crowd simulations and similar problems that are not based on the graph data structures mentioned in the previous sections usually exploit some kind of potential field as the underlying data structure. These methods work in a way that is completely different than the aforementioned graph-based methods (see [23]). The environment is divided into a fine grid and a potential function value is computed for each of the cells of this grid (see Fig. 6). In this figure we can see that the potential function is constructed as a sum of three different functions – the density function is used to prevent the pedestrians to collide with each other by raising the potential values around the location of each of the pedestrians. The goal function is used to determine the targets of the pedestrians and the boundary function prevents the pedestrians from leaving the designated area and colliding with the static obstacles. These functions are the summed together (perhaps with other similar functions not mentioned here) and the pedestrians are then updated according to the neighborhood values of the final potential function.

The potential function thus reflects the probability that an artificial pedestrian will choose this cell as a part of his/her path. Depending on the set of rules used for the pedestrian movement, the function value may be dependent on its position in the environment (the pedestrians are more likely to walk on the pavement than in the middle of the road), the proximity of other pedestrians and other objects (to prevent collisions with both static and dynamic obstacles), the goals of each of the pedestrian and possibly on some other criteria. Even such circumstances as periodically changing environment (such as a pedestrian crossing with traffic lights) or learning from previous experience may be incorporated into the computation. Each pedestrian then finds his/her way through the environment by moving along the path with minimum energy consumption.

The most obvious disadvantages of this kind of approach include the fact that the potential function must be computed periodically at a rather high framerate in order to make the movement seem natural and it must be computed for each pedestrian separately. On the other hand, the global-scope path planning is done together with collision avoidance, group coherence conservation and the preservation of other behavior-based features of the moving crowd as needed and defined by the potential function. Furthermore, the crowd may be divided into several groups of pedestrians that share a common goal and the potential field may then be computed for the entire group. Also, unlike the graph-based methods, this approach is dependent on the size of the environment rather than the number of the people in the crowd.



Compute a set of grids representing state.
Combine grids into a set of potential fields.
Use potential fields to update people's positions.
Density Grid: Indicates people's static locations.
Goal Grids: Indicate people's desired locations.
Boundary Grids: Indicate impassible grid cells.
Other grids...

Figure 6: Example of a potential field computation, [23].

The behavior of the crowd is driven entirely by the potential function and thus it is very difficult to make each pedestrian behave differently. For some applications the diversity of the crowd may easily be ommited for the sake of the simplicity and a performance improvement, but in other applications we might want each artificial pedestrian to move according to slightly different rules and follow a different goal. In such cases, the potential field based methods provide us with only few advantages.

4 Conclusion

The modern crowd simulation applications may be divided into two main branches according to the type of the underlying structure they use for the crowd navigation. The first group uses some kind of a graph to divide the environment into logic parts and uses graph path-finding algorithms to navigate separate agents through the environment. The solutions in this group may be further subdivided according to the type of the graph they use and according to the type of local criteria they use for tasks such as collision avoidance and similar behavior. However, independently on these features, we may say that these approaches provide us with more possibilities to control each agent individually (according to his visibility field, position, overall condition and possibly other features) and are more natural as the crowd is clearly a special group of different human entities.

The other group of approaches is based on gradient methods running on a potential field map. This approach does not need any spatial division of the environment except of the fine grid which discretises the potential function. It is also convenient that the performance of these methods is usually independent on the number of the entities in the environment (it is, however, dependent on the size of the environment and the number of cells in the grid). The potential field based methods provide us with less possibilites to control the crowd as a group of individuals and sometimes suffer from collisions between the pedestrians (which are easily handled by the agent based methods) but they may easily simulate the crowd phenomena such as lane and vortex forming because these problems are well-known from other fluid dynamics applications.

Each of the two approaches provides us with different possibiilites, suffers from different drawbacks and – probably most importantly – has a different focus. The correct use of one or the other of these approaches strongly depends on the desired purpose of the crowd. Agent based methods allow us a more individual control over each human in the crowd which may be important for computer games or other strongly interactive applications, while the potential field based methods are more suitable for applications with the focus on the flow of the entire crowd – e.g., traffic simulations, simulated evacuation scenarios, etc.

References

[1] Bethesda softworks. http://www.bethsoft.com.

[2] Blizzard entertainment. http://us.blizzard.com.

[3] Ensemble studios. http://www.ensemblestudios.com.

[4] Firaxis games. http://www.firaxis.com/.

[5] Interplay entertainment. http://www.interplay.com.

[6] Steven jackson games. http://www.sjgames.com.

[7] Ubisoft entertainment. http://www.ubi.com.

[8] Valve software. http://www.valvesoftware.com/.

[9] Wizards of the coast. http://wizards.com.

[10] Srikanth Bandi and Daniel Thalmann. Space discretization for efficient human navigation. *Computer Graphics Forum*, 17(3):195, 1998.

[11] R. Geraerts and M. H. Overmars. The corridor map method: a general framework for real-time high-quality path planning: Research articles. *Comput. Animat. Virtual Worlds*, 18(2):107–119, 2007.

[12] S. Goldenstein, M. I. Karavelas, D. N. Metaxas, L. J. Guibas, E. Aaron, and A. Goswami. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, 25(6):983–998, 2001.

[13] H. Huang and S. Chung. Dynamic visibility graph for path planning. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2813–2818 vol.3, Sept.-2 Oct. 2004.

[14] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[15] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Comput. Graph. Forum*, 23(3):509–518, 2004.

[16] D. Nieuwenhuisen, A. Kamphuis, and M. H. Overmars. High quality navigation in computer games. *Sci. Comput. Program.*, 67(1):91–104, 2007.

[17] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[18] M. Overmars. Practical algorithms for path planning and crowd simulation. In *Proceedings of the 25th European Workshop on Computational Geometry (EuroCG)*, page 263, Brussels, Belgium, March 2009. Invited speech.

[19] J. Pettré. Autonomous navigation for crowds of virtual humans: part i: interactive design of virtual population using navigation graphs. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–21, New York, NY, USA, 2008. ACM.

[20] S. Rabin. *AI Game Programming Wisdom*. Charles River Media, inc., 2002.

[21] S. Ranta-Eskola. Binary space partitioning trees and polygon removal in real time 3d rendering. Master's thesis, Uppsala University, Box 311, S-751 05 Uppsala, Sweden, January 2001.

[22] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha. Real-time path planning for virtual agents in dynamic environments. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–9, New York, NY, USA, 2008. ACM.

[23] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1160–1168, New York, NY, USA, 2006. ACM.

[24] J. van den Berg and M. Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4253–4258, 29 2007-Nov. 2 2007.

[25] E. W. Weisstein. Tessellation. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Tessellation.html.

[26] H. Zhang, B. Dong, Y. Cen, R. Zheng, S. Hashimoto, and R. Saegusa. Path planning algorithm for mobile robot based on path grids encoding novel mechanism. In *ICNC '07: Proceedings of the Third International Conference on Natural Computation*, pages 351–356, Washington, DC, USA, 2007. IEEE Computer Society.