Chapter 25
Navigating Through Virtual Worlds:
From Single Characters to Large Crowds

Norman Jaklin
Utrecht University, Netherlands

Roland Geraerts
Utrecht University, Netherlands

Proofing copy. Copyright IGI Global. May not be posted or distributed.

ABSTRACT

With the rise and success of digital games over the past few decades, path planning algorithms have become an important aspect in modern game development for all types of genres. Indirectly-controlled playable characters as well as non-player characters have to find their way through the game's environment to reach their goal destinations. Modern gaming hardware and new algorithms enable the simulation of large crowds with thousands of individual characters. Still, the task of generating feasible and believable paths in a time- and storage-efficient way is a big challenge in this emerging and exciting research field. In this chapter, the authors describe classical algorithms and data structures, as well as recent approaches that enable the simulation of new and immersive features related to path planning and crowd simulation in modern games. The authors discuss the pros and cons of such algorithms, give an overview of current research questions and show why graph-based methods will soon be replaced by novel approaches that work on a surface-based representation of the environment.

INTRODUCTION

Over the past decade, educational games have received increasing attention. By now, a large body of research has been carried out to determine how the principles behind engaging digital games can be utilized to improve learning. In general, this is an open research question with new methods still being developed, e.g. (Jeuring, van Rooij, & Pronost, 2014). It has been shown for particular cases that engaging virtual environments can yield a higher motivation and better learning results for students (Murray, Bogost, Mateas, & Nitsche, 2006; Ketelhut, Dede, Clarke, Nelson, & Bowman, 2007; Ketelhut, 2007; Gee, 2007; Schmitz, Specht, & Klemke, 2012).

DOI: 10.4018/978-1-4666-9629-7.ch025

Copyright © 2016, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

A key requirement to achieve a high level of engagement and thus better learning results is the game being *immersive*. Immersion can be achieved in many ways, among which are the technological aspects of a game. Modern computer hardware and smarter algorithms have radically changed and shaped the overall appearance of digital games in general and educational games in particular. Technological aspects of games such as graphics, modeling or physics simulation have received much attention, and this led to a wide range of novel techniques to generate visually convincing and believable pictures, character models, and animations.

By contrast, the *paths* traversed by virtual characters[1] are often *not* visually convincing. Many educational games do not let their virtual characters move around autonomously. This is due to the fact that moving characters might not be necessary to achieve the learning goals. Furthermore, the computation of realistic and visually convincing paths is difficult and might even ruin a user's immersion when done in a cheap way. However, many educational games try to simulate a realistic virtual 3D environment to better match recent advances in entertainment games. An example of such a game is *The River City Project* (2002 - 2007). The game has been successfully used by teachers and students in the U.S., and results indicate that using virtual environments in education "might act as a catalyst for change in student's self-efficacy and learning processes" (Ketelhut, 2007). *The River City Project* simulates a virtual city from the late 19th century with buildings and different terrain, and the residents of the city are displayed as virtual 3D non-player characters. These, however, seem stationary and do not autonomously move around the city in a realistic way. We believe that state-of-the-art path planning and crowd simulation methods can improve such games with respect to the users' immersion and overall learning experiences.

Many games that *do* support moving characters rely on predetermined or scripted paths. In such games, a designer can manually create believable paths that are of high quality. While this does not ruin a user's immersion, it also limits the flexibility and design possibilities of a game. By contrast, games that are more flexible and allow unpredictable interactions among characters rely on *algorithms* to handle path planning.

In this chapter, we will discuss such algorithms. We focus on path planning for single virtual characters and large virtual crowds. The latter have become increasingly important in simulation software for mass events and evacuation training, e.g. (SportEvac, 2014), which could be used in school education to increase the students' awareness of potential dangers during such events.

The overall goal in this research field is to compute visually convincing and natural paths for a large number of characters in real-time to improve the user's immersion. We will discuss how this field and the methods used have changed over the past few decades, and give an overview of state-of-the-art techniques and open research problems.

We will start with discussing different ways to represent traversable space in a virtual environment. A classical approach is to use a grid or waypoint-graph representation. Path planning is then done using a graph-based search algorithm. We will explain why a graph representation is not adequate for many challenging path planning and crowd simulation problems.

A more recent approach to represent traversable space is to use a *navigation mesh*. We will discuss the advantages of navigation meshes over graph representations. In a believable and immersive game world, autonomous virtual characters need to react adequately to dynamic changes in the environment. Such dynamic changes might allow traversing paths that were previously blocked, or they might block paths that were previously traversable. Examples are parking cars on a sidewalk or an automatic door that opens and closes. Many games avoid solving these issues by using work-around techniques. Characters may be permitted

Proofing copy. Copyright IGI Global. May not be posted or distributed.

to temporarily walk through obstacles, or the discs used for collision avoidance are very small. These workarounds may destroy a player's experience in an otherwise immersive world. A navigation mesh needs to be able to efficiently represent virtual environments that can dynamically change over time, and we will discuss cutting-edge research on this topic.

Some recent path planning methods that use a navigation mesh are based on a multi-level planning hierarchy. They compute a first rough path and then use advanced path following techniques that modify the input path to generate a smooth curve with certain desired properties. In this context, we will discuss the *Indicative Route Method*, which uses an augmented navigation mesh called *Explicit Corridor Map*. We also discuss its successor, the *Modified Indicative Routes and Navigation* method, which allows path planning in heterogeneous virtual environments that feature different region types. We will also give an overview of different collision-avoidance techniques and discuss their advantages and drawbacks.

Another topic we will discuss is social group behavior. While many of the aforementioned path planning methods solve issues related to single characters, they lack simulating social aspects for groups. In real life, people tend to form social groups and split up again after a while. Members of such groups try to stay in particular social formations whenever possible. Other people display different walking behavior when encountering a social group. For instance, social territories occupied by groups are usually avoided by other people. An immersive virtual world needs to reflect this.

Handling large virtual crowds requires different approaches. Virtual crowds that consist of hundreds of individual characters can be rendered with modern gaming hardware. In an educational game that features large crowds, however, only a small portion of the overall CPU power can be assigned to handle crowd simulation. Thus, coordinating and handling the navigation of crowds requires advanced strategies on a theoretical and practical level.

We now start by discussing different ways to represent traversable space in a virtual environment. Such representations form a basis on top of which path planning and crowd simulation can be performed.

REPRESENTING TRAVERSABLE SPACE

How should traversable space be represented in a game? The choice for a certain representation is strongly connected to the complexity and required efficiency of the application at hand. In this section, we discuss several representations of traversable space and their pros and cons.

Historically, digital games have been developed for entertainment purposes, while educational games are a comparably new phenomenon. Thus, we will consider examples from popular entertainment games when discussing traversable space. The same observations and conclusions can be applied to recent and future educational games.

Depending on the type of game, more or less effort needs to be spent on this matter. As an extreme example in which representing traversable space is a trivial task, consider the classic 2D arcade game *Space Invaders* (Taito Corporation, 1978). Here, the enemies move in a predetermined and scripted way, which does not involve any path planning. The player's traversable space is only a 1D straight-line segment at the bottom of the screen for which no complex representation is necessary. Note that there is no automated path planning for the directly-controlled player character either, but the player's traversable space is still important for collision avoidance with the left and right screen limits. A game that is more complex with respect to path planning is *Pac-Man* (Namco, 1980). The enemies follow different strategies to catch the player. Here, the 2D game world consists of rectilinear tiles that are either

Proofing copy. Copyright IGI Global. May not be posted or distributed.

traversable or blocked. Thus, a simple tile-based grid approach is a sufficient way to represent all traversable space. By contrast, many modern games tend to simulate an open world (e.g. *Grand Theft Auto V* (Rockstar Games, 2013)), or they provide a sandbox environment, which allows players to generate their own content (e.g. *Minecraft* (Mojang, 2009)). Both types of games feature highly dynamic multi-layered 3D environments with different terrain types. Characters in future variants of such game genres should not only plan collision-free paths, but also autonomously detect and use areas where climbing or jumping over gaps is possible to access difficult-to-reach areas in the game world. Such features require a representation of traversable space that is far from trivial and still open to future research.

The topology of the game world also influences the choice of how to represent traversable space. While the topology of the world in *Space Invaders* is a 2D plane, there are games with more complex topologies. In *Pac-Man*, characters are allowed to exit to the left and right screen edges to appear on the opposite side. This behavior follows the topology of a cylinder. An example of a game with the world topology of a torus is *Asteroids* (Atari Inc, 1979). The player's ship as well as asteroids and flying saucers can exit the screen to all four edges and appear on the opposite side. 3D Games that allow a character to fly, such as *Descent* (Parallax Software, 1994), feature a 3D space topology. In *Super Mario Galaxy*, the player can fully circumnavigate small planets and, hence, the world topology equals a sphere (Nintendo EAD Tokyo, 2007). In *Super Paper Mario* (Nintendo SPD, 2007), the player has to interactively switch between 2D and 3D perspectives to solve puzzles. In *Portal* (Valve Corporation, 2007), the player can create traversable connections between arbitrary points in particular areas of the game world. The game *Monument Valley* (Ustwo, 2014) features dynamic Escher-like worlds that are physically impossible. Other exotic topological spaces such as the *Möbius Strip* (Möbius & Listing (independently), 1858) or the *Klein Bottle* (Klein, 1882) require different path planning strategies, but may also enable novel gameplay elements.[2]

Note that we only discuss games that require two-dimensional traversable space representations. Even when the game world is technically a (multi-layered) 3D environment, a 2D representation is sufficient as long as characters need to traverse 2D surfaces only. For actual 3D path planning (e.g. for autonomous flying characters), the problems we discuss in this chapter are more complex and require even more advanced algorithms beyond the scope of this chapter.

For more information on world topologies and the concept of traversable space in digital games, we refer to the discussion *Theorizing Navigable Space in Video Games* (Wolf, 2009/2010). Its main focus is on space the *player* can traverse from an abstract level. For path planning purposes, however, we are interested in representing space for autonomous characters or player characters that are controlled in an indirect way (e.g. by assigning a goal position via a mouse click or a finger press on a touchpad). Therefore, we will now list several methods to represent traversable space that have been used in past and present digital games, and discuss their advantages and drawbacks.

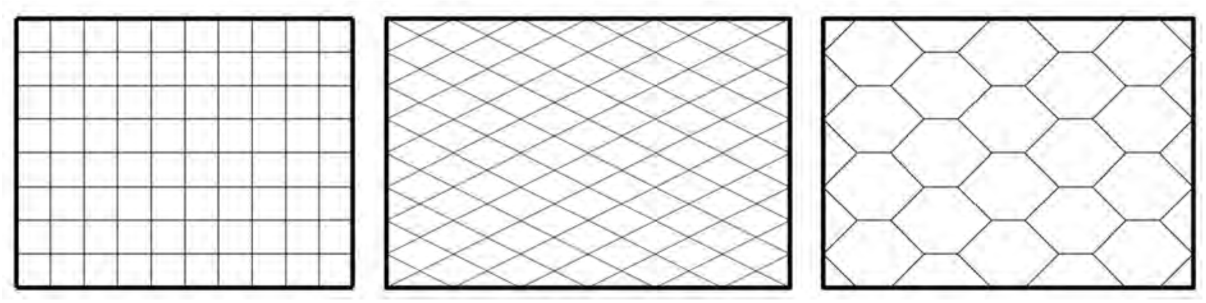
Grids

Grids can be intuitively described as regular subdivisions of the plane into cells of a particular shape and with a particular grid resolution. From an abstract point of view, grids can be seen as a special case of *lattice structures*. This means that by adding so-called *generator* vectors to a given cell point in the grid, any neighboring cell point can be obtained. For a 2D grid, two generators representing the *up* and *right* directions are sufficient to obtain all grid cell points (LaValle, 2006).

Grids are intuitive and easy to implement, which makes them a popular representation of traversable space. The most common types are rectilinear or square grids. However, other types

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 1. Rectilinear, isometric and hexagonal grids



have been widely used in games, too. Hexagonal grids are common, as well as grids with isometric diamond-shaped tiles; see Figure 1. From a topological point of view, isometric grids and rectilinear grids are equal. They are commonly used in 2D games to simulate an isometric view on pseudo-three-dimensional game worlds in which correct clipping is easily achieved by rendering objects on the grid from top to bottom along the screen. They are also used in many traditional and modern board games, and have also been widely used in *pen and paper role-playing games* to simulate combat scenarios.

When traversable space is represented using a grid, the actual path finding is usually performed on the *dual graph* of the grid cells. This is because a wide range of graph-search algorithm exists that can compute shortest paths efficiently (see Section "A* and its variants"). Furthermore, game objects are usually placed in the center of the grid cell. In the dual graph, each grid cell is represented as a vertex and adjacent cells are connected by an edge. While a rectilinear, square or isometric grid keeps its structure when considering its dual graph (except for an offset translation), hexagonal grids become triangular and vice versa, see Figure 2. Thus, using the center points of hexagonal cells as possible character positions is technically the same as performing path-finding on a triangular graph. In the case of square grids, typical variants are 4-neighbor and 8-neighbor square grids, depending on whether diagonal movement from one cell to another is allowed or not.

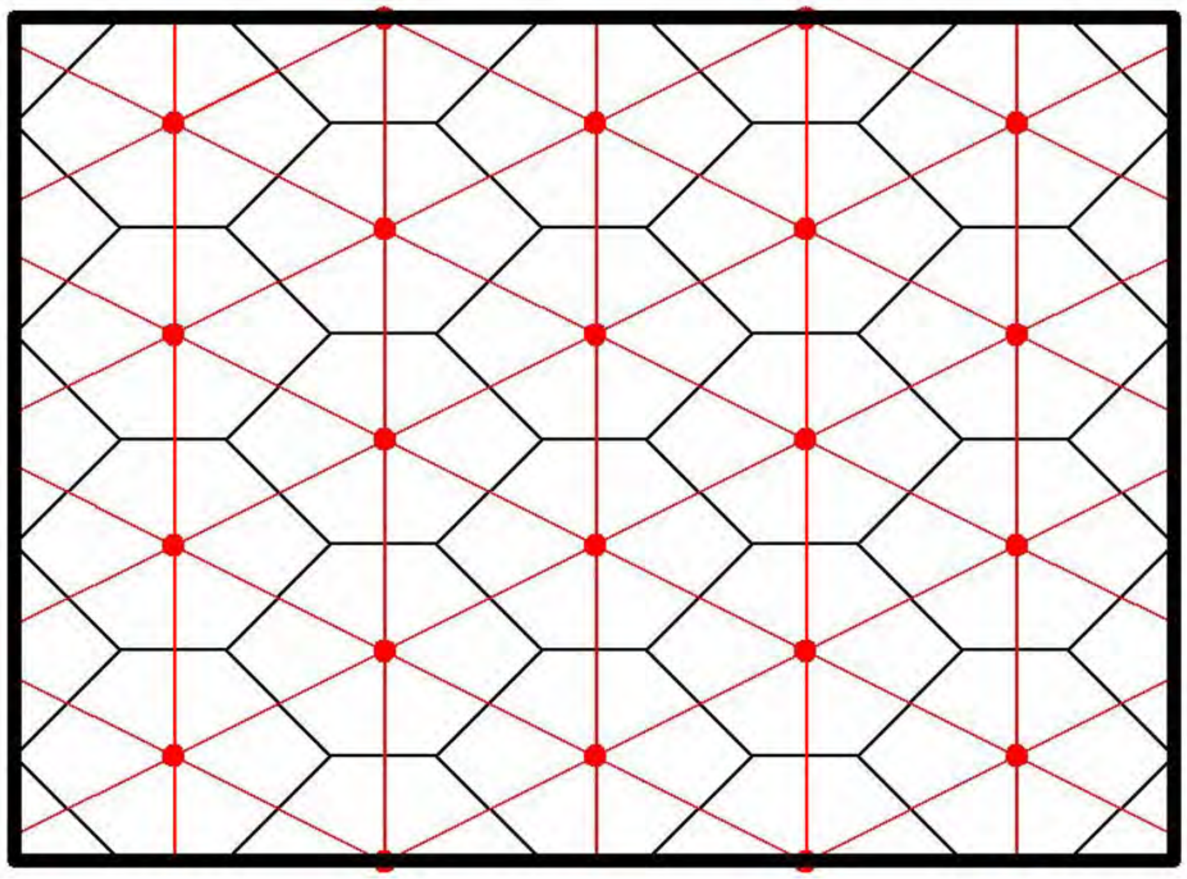
Path planning can also be done on the edges or vertices of the grid itself. *The Settlers II*, for instance, uses a hexagonal grid (Blue Byte Software (today: Ubisoft Blue Byte), 1996). The player can build roads along the edges and place flags along the vertices of the grid. Objects are then distributed and moved along the roads. The edges and vertices of the grid are also used in adaptations of abstract board games such as *Go* or in many puzzle games.

While grids are easy to implement, a major problem is that grids may not cover all of the traversable space visually shown to the player. Some corners of the game world and important passages between two obstacles might not be traversable due to a too coarse grid resolution; see Figure 3.

The question of how much a grid path deviates from a shortest path on the exact geometry has been mathematically answered via path-length analysis proofs. Nash (2012) presented a unified proof structure for upper bounds on the length of square, triangular, hexagonal and cubic 3D grid paths. The author showed that a grid path on a triangular, 4-neighbor square, 8-neighbor square, or hexagonal grid can be at most 2, 1.41, 1.08, or 1.15 times as long as an optimal (shortest) path, respectively. In addition, Jaklin, Tibboel, and Geraerts (2014) presented a path-cost analysis proof for 8-neighbor square grid paths in environments with multiple region types (e.g. different terrain

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 2. Hexagonal grids and triangular grids are dual graphs of each other. Each point is a vertex in the triangular graph (red), and it corresponds to a hexagonal cell in its dual graph representation.



types), where the cost for traversing a particular region is given as a weight .They showed that a grid path can be at most 5.08 times as long as an optimal path in this situation, when all regions are aligned with the grid.

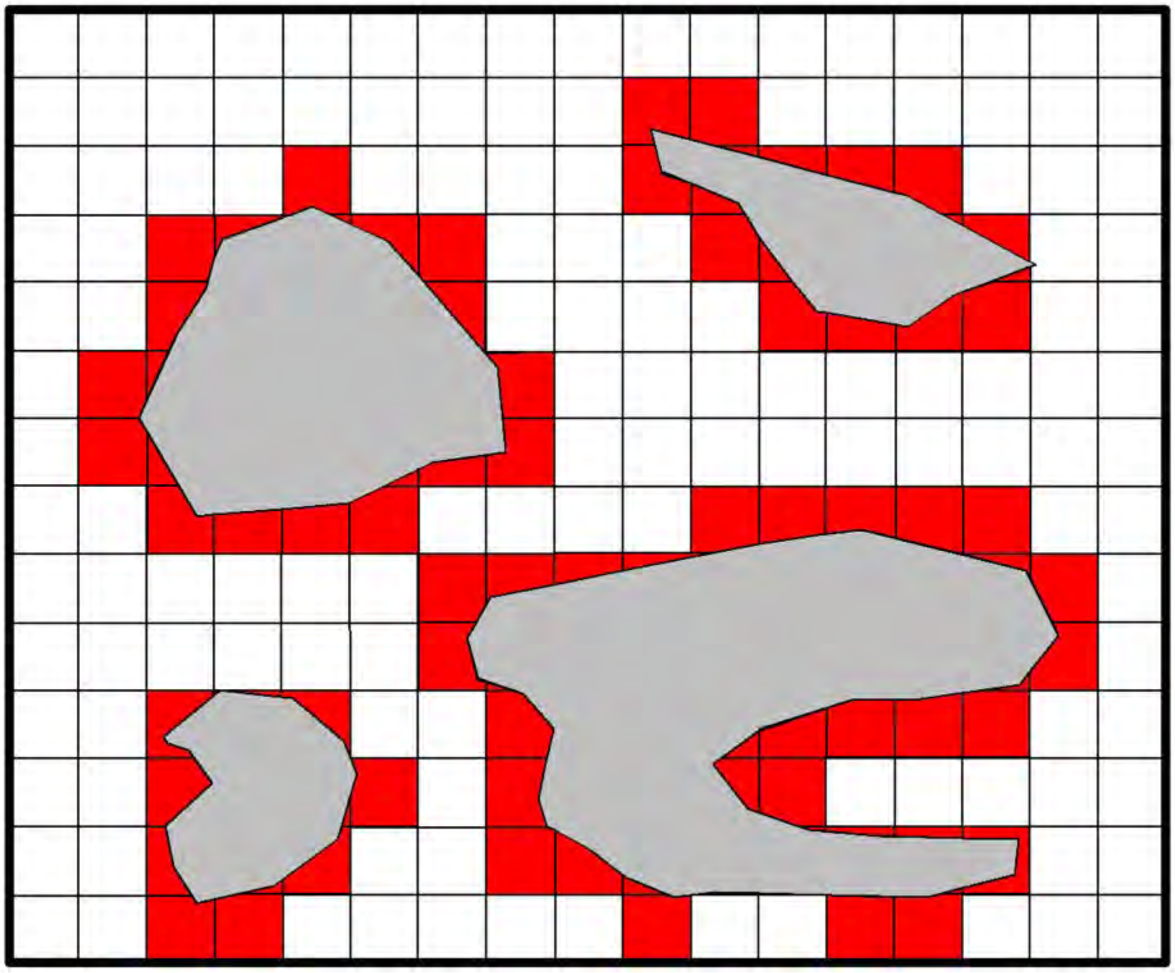
Grids or their dual graphs are also inflexible because characters' motions are hard to coordinate when two or more characters follow the same edge bi-directionally. In addition, they impose only an infinitely small subset of all possible trajectories. Resulting motions may not be visually convincing because the underlying graph edges are not smooth and do not cover energy-optimal paths. Smoothing the paths can be expensive and often requires a global approach. This is undesired because digital games require real-time responses. Furthermore, the dual graph of a grid requires computationally expensive updates when the obstacles are inserted, deleted, or moved. Note that these issues also occur with other graph-based approaches such as waypoint graphs, which we discuss now.

Waypoint Graphs or Road Maps

Waypoint graphs and road maps are two interchangeable terms. The nodes (or waypoints) resemble locations in the game world in which a character can be located. An edge between two nodes in the graph resembles a straight-line path which characters can traverse without hitting obstacles; see Figure 4.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 3. Squared grid overlaid on a polygonal environment: Red (dark) squares count as obstacles because they are partially covered by polygons. Some passages are not traversable due to the too coarse grid resolution.

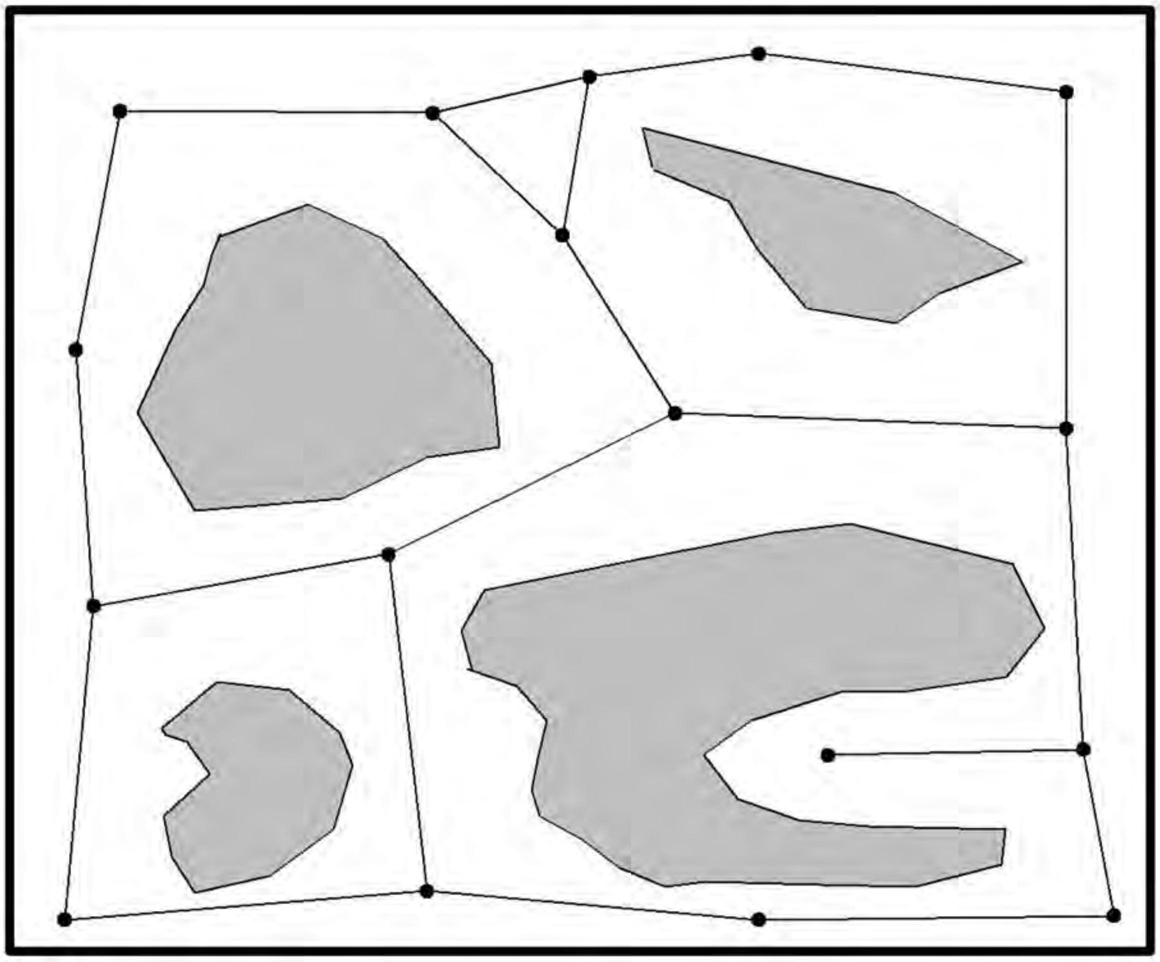


Waypoint graphs can be automatically generated based on the *visibility graph* of the environment or based on some other decomposition of the traversable space such as the *medial axis* of the environment (see Section "Navigation Meshes" and (de Berg, van Kreveld, Overmars, & Schwarzkopf, 2000). Other examples from the robotics community are *Probabilistic Road Maps* (Kavraki, Svestka, Latombe, & Overmars, 1996) and *Rapidly-Exploring Random Trees* (LaValle, 1998). The latter two are more general because they can be used to solve higher-dimensional problems, e.g. finding a path for a robot manipulator arm with six joints that has to grasp an object and move it through a cluttered 3D environment.

The nodes of a *visibility graph* are the vertices of the polygonal obstacles in the environment. An edge between two nodes is added in the visibility graph, if the straight-line segment between them does not intersect any obstacles, or, in other words, if the two nodes *see* each other; see Figure 5 (left). Note that the complexity of the visibility graph

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 4. A waypoint graph for an environment with four obstacles

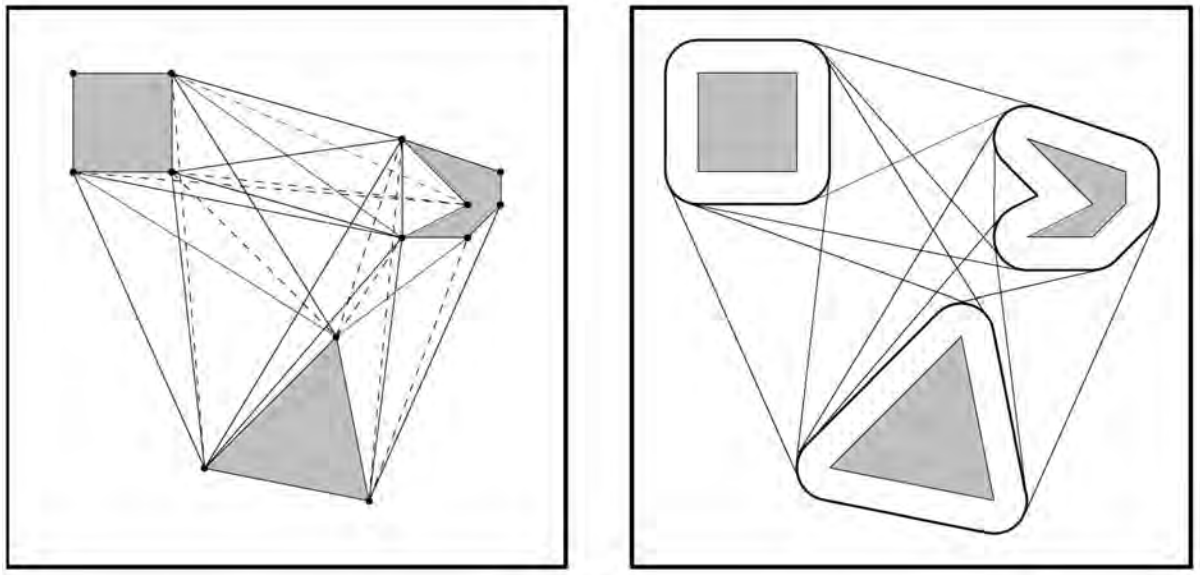


Proofing copy. Copyright IGI Global. May not be posted or distributed.

can be quadratic in the number n of polygon vertices. The graph can be computed in $O(n^2)$ worst-case optimal time (Welzl, 1985; Ghosh, 2007).

Laumond (1987) introduced the concept of the *generalized visibility graph*, which has been used for path planning by several authors (Yu, 2006; Masehian & Amin-Naseri, 2004; Wein, van den Berg, & Halperin, 2007). For a character that is represented as a disc with a maximum size $r \geq 0$, the obstacles in the generalized visibility graph are first inflated by r using Minkowski's operations. There are two types of nodes in the resulting graph. Vertices of the first type are concave corners of the inflated obstacles. Vertices of the second type are *fictitious vertices* that correspond to convex arcs of the inflated obstacles. In general, a shortest path in a polygonal scene consists of segments that are tangent to the borders of the obstacles. Therefore, it can be shown that all vertices of type one need not be connected by additional edges because they are never part of a shortest path. For the set of fictitious vertices, a generalized notion of visibility is used: Two fictitious vertices *see* each other, if and only if there is at least one pair of points on the corresponding

Figure 5. Left: A visibility graph for a scene with three obstacles. Dashed edges are pruned in the generalized visibility graph because their corresponding vertices in the generalized graph do not see each other. Right: A generalized visibility graph for the same scene with inflated obstacles and corresponding visibility edges.



two arc segments such that their straight-line connection is tangent to both arc segments. See Figure 5 (right) for an example. The generalized visibility graph is not only smaller in the number of edges, but it also makes a character automatically keep clearance from obstacles. Since the search for tangents can be performed in $O(n^2)$ time, the overall time complexity to compute the generalized visibility graph is the same as for the visibility graph.

Another common way to build a waypoint graph is to manually determine waypoints and edges for a given environment in the level design phase of game production. Some games such as real-time strategy games (e.g. *Starcraft II* (Blizzard Entertainment, 2010)) even allow players to set waypoints themselves and use this as a tactical gameplay element.

As mentioned before, waypoint graphs are generally smaller in the number of nodes and edges compared to the dual graph of a grid. This decreases the time to perform a path-finding algorithm on that structure (see Section "A* and its Variants").

However, all graph-based techniques suffer from a range of problems that are inherent to these techniques. A graph does not provide information about the actual geometry of the scene, and it does not allow characters to deviate from their paths induced by the graph edges. This becomes an even bigger problem for a large crowd and collision-avoidance among the crowd members. The resulting paths may neither be natural nor visually convincing.

Given all these issues, we believe that the end of graph-based methods for advanced path planning has come. Recent techniques based on a surface-representation of the virtual environment - such as navigation meshes - are a promising next step into the future of digital games that feature large crowds or require other advanced navigational tasks.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Navigation Meshes

A *navigation mesh* is a set of two-dimensional simple polygons representing the traversable space in a game world. With this general definition, the aforementioned grids can also be seen as special types of navigation meshes. However, navigation meshes can be built with polygons representing the exact geometry of the environment, or with a sufficient approximation of it. Figure 6 shows an example of a navigation mesh.

Games that come with level editors and support path planning with navigation meshes either use an automatic navigation mesh generation algorithm or let the user define traversable space for the level geometry within the editor. For example, in *Counter-Strike: Source*, automatic navigation mesh generation for user-generated maps is done by a flood-filling algorithm (Valve Corporation, 2004). The user defines spawn-points for players on traversable parts of the game map. From those points, the algorithm computes all traversable space that can be reached from the initial positions. Additionally, the user can explicitly mark traversable areas in case the algorithm fails to detect some parts due to steep stairs or ramps.

An open source library to automatically construct navigation meshes out of 3D level geometry is the project *recastnavigation* (Mononen, 2009). *Recas*t generates a voxel mold from the 3D level geometry and automatically detects and prunes non-traversable areas. The resulting walkable space is then divided into simple overlaid 2D regions with one single non-overlapping contour. By tracing the boundaries of the regions, the algorithm produces a set of traversable convex polygons as a final output. The resulting navigation mesh can be then used for path planning. For example, it can be fed into *Detour*, which is a path-finding toolkit accompanying *Recast*.

Kallmann (2010a) introduced a navigation mesh called a *Local Clearance Triangulation* (LCT). It can be used to answer path queries for characters of different size. Locally shortest paths can be computed in optimal time. If global optimality is required, an extended search is used to gradually improve the path. Furthermore, Kallmann discussed several algorithms and operations that are based on generic triangulation-based navigation meshes and on the LCT in particular (Kallmann, 2010b). Among these are methods for automatic agent placement, tracking moving obstacles, and ray-obstacle intersection queries. Kallmann's navigation mesh yields an exact representation of the environment and can handle dynamic updates efficiently. However, (multi-layered) 3D environments are not discussed.

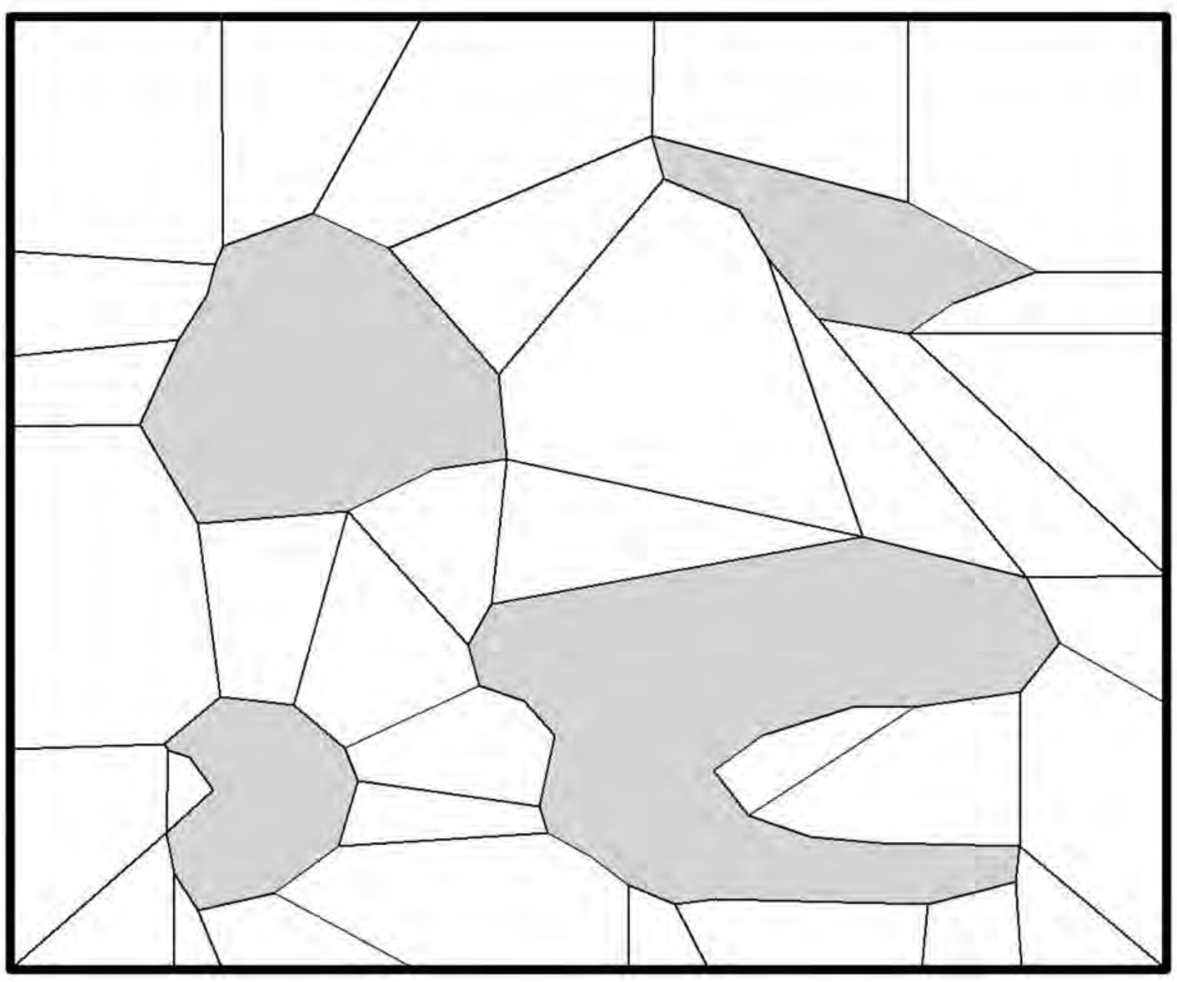
Pettré et al. (2005) introduced a navigation mesh based on discs and cylindrical areas. It supports multi-layered 3D environments and characters of variable size. However, it does not support an exact representation of the navigable space and efficient dynamic updates of the environment.

A navigation mesh that combines the advantages of the aforementioned approaches is the *Explicit Corridor Map* (ECM) (Geraerts, 2010). It is based on the *medial axis* of the environment, which is the set of all points that are equidistant from at least two distinct closest obstacle points; see Figure 7.

The medial axis can be seen as a special type of waypoint graph in which all edges run through the center of the free space between pairs of obstacle polygons. For each vertex of the medial axis graph, there are either at least three obstacle polygons that have the same distance from that vertex, or the vertex is placed in a non-convex corner of an obstacle. An edge between two vertices of the medial axis consists of a sequences of straight-line segments and parabolic arcs, depending on the type of corresponding obstacles to its left and right, see Figure 7 (left). With each element in this sequence, its left and right closest obstacle points are stored. This partitions a 2D environment into a set of walkable areas in $O(n \log n)$ time and $O(n)$ space, where n is the total number of obstacle vertices. Each area cor-

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 6. A navigation mesh for the environment shown in Figure 3 and Figure 4

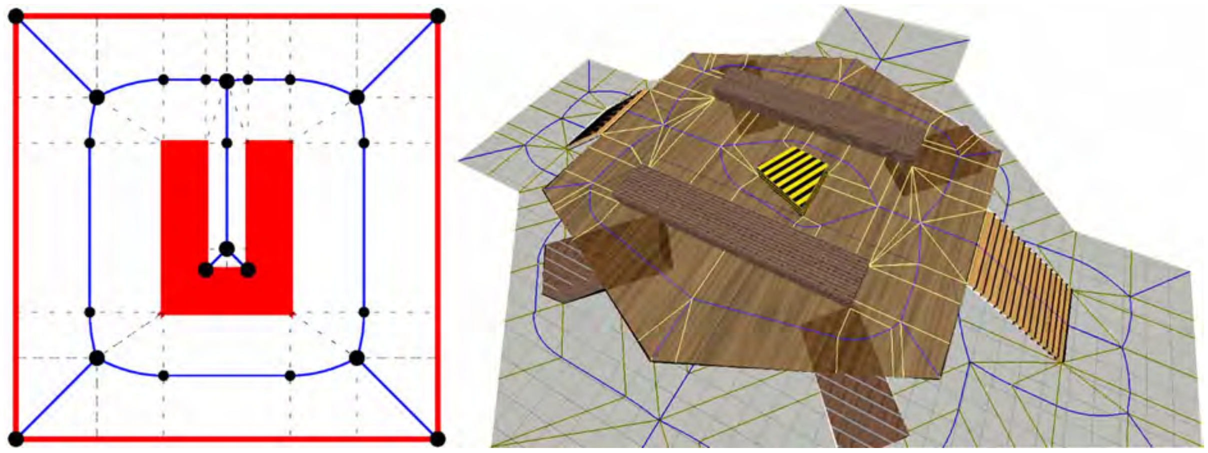


responds to one particular obstacle polygon, as all points in that area are closer to that obstacle than to all other obstacles.

In the field of Computational Geometry, a similar structure is known as the *Generalized Voronoi Diagram* (GVD), and it has been widely studied during the last decades. It can be efficiently computed using graphics hardware (Hoff, Keyser, Lin, Manocha, & Culver, 1999): First, for each two-dimensional site (i.e. the obstacle polygons, lines or points) a three-dimensional *distance mesh* is computed and drawn by the graphics hardware, each mesh in a different color. By projecting the distance meshes back onto the 2D plane and tracing the boundary lines of the different regions in the color buffer, a feasible approximation of the GVD can be obtained. This approach requires the obstacle polygons to be convex, so concave polygons are first subdivided into a convex ones. This yields edges and vertices that are not part of the GVD for the original scene. For path planning purposes, these additional edges are desired if they run into convex corners. Otherwise characters cannot access such corners. All other additional edges, however, are redundant. The medial axis

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 7. Left: Obstacles (shown in red; the center U-shape and the four bounding line segments), medial axis (solid blue), ECM vertices (large discs) and closest points, subdividing the free space into ECM cells (dotted gray lines). Right: a multi-layered 3D environment and its medial axis (van Toll, Cook IV, & Geraerts, 2012).



is basically the same as the GVD, but with the redundant edges and vertices pruned.

The main drawback of the ECM is that it is difficult to implement in a robust way. There are software libraries available for computing GVDs (Held, 2011; The Boost C++ Library, 2015), but they often struggle with imprecision in the data due to numerical rounding errors, especially when a high level of detail is needed. Consequently, the input data (i.e. obstacles) need to be preprocessed.

Still, the ECM has many advantages. All traversable space is represented with respect to the correct topology of the environment. This resolves the issues that are inherent to all approximated representations that we discussed before. Furthermore, the ECM is space-efficient and supports time-efficient extraction of global paths with any desired amount of clearance from obstacles. The ECM works both for 2D and multi-layered 3D environments; see Figure 7. In addition, the ECM can be dynamically updated in real-time by only applying *local* changes to the mesh whenever an obstacle is inserted or deleted (van Toll, Cook IV, & Geraerts, 2012).

For more algorithms on automatic navigation mesh generation and related path planning topics, we refer the interested reader to the *AI Game Programming Wisdom* book series (Rabin, 2008). This concludes the first part on representing traversable space. We have discussed a wide range of representations, as well as their advantages and drawbacks. Now we will show how the corresponding structures can be queried to compute paths.

PATH PLANNING FOR SINGLE CHARACTERS

In this section, we will elaborate on both classic and recent path planning strategies. We start with the popular A* search algorithm on a graph. We then present the *Indicative Route Method* (IRM), and its successor, the *Modified Indicative Route and Navigation* (MIRAN) method. Afterwards, we give a general overview of local collision avoidance strategies.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

A* and Its Variants

The A* algorithm (Hart, Nilsson, & Raphael, 1968) is one of the best known and probably the most used path-finding algorithm in the domain of digital games. This is because it can efficiently compute shortest paths in a graph and combines the advantages of *Dijkstra's algorithm* (1959) with a greedy *Best-First-Search* strategy. With only small and easy to implement variations, a wide range of variants (e.g. shorter computation time to find good paths rather than optimal ones) can be obtained.[3] This makes the A* algorithm a flexible and powerful tool for many path-finding applications (Trovato, 1996).

The main idea of A* is to combine actual traversal costs from a start node with heuristic values that estimate the distance to a target node. Given an edge-weighted graph with two designated nodes s and t, a function assigns to each node n in the graph the costs of the currently known shortest path from s to n. Furthermore, a heuristic function h assigns to each node in the graph the estimated costs of a path from n to t. A* then starts to search for a path from s to t by computing $f = g + h$ for each node under consideration and expanding a node with minimum f-value in each step. The g-value of a node n is updated whenever a shorter path than the current one from s to n is detected. A* manages two lists of nodes, the *open list* and the *closed list*. While the open list stores all nodes that are currently under consideration, the closed list stores all nodes that have already been expanded and do not need to be visited again.

It can be proven, if the heuristic function h does not overestimate the actual costs for each node, that A* will always find an optimal (i.e. shortest) path. On the contrary, if h overestimates the costs for some or all nodes, the computation time of the search may be decreased, but the path may not be optimal.

If h is an exact estimation of the actual costs, then A* has the nice property to find a shortest path in optimal time. In this case only the nodes that are contained in a shortest path as well as their neighbors are expanded; see Figure 8 (bottom). Even if all paths in the graph are shortest, A* will expand only one of them depending on the sorting strategy of the open list. A theoretical approach to exploit this property could be to compute all costs of optimal paths for all pairs of nodes in the graph as a preprocessing step (without storing the paths themselves to save space). These costs could then be used as an exact heuristic to make A* compute optimal paths in optimal time. However, this is only practical for small graphs and is usually not a feasible solution in gaming applications.

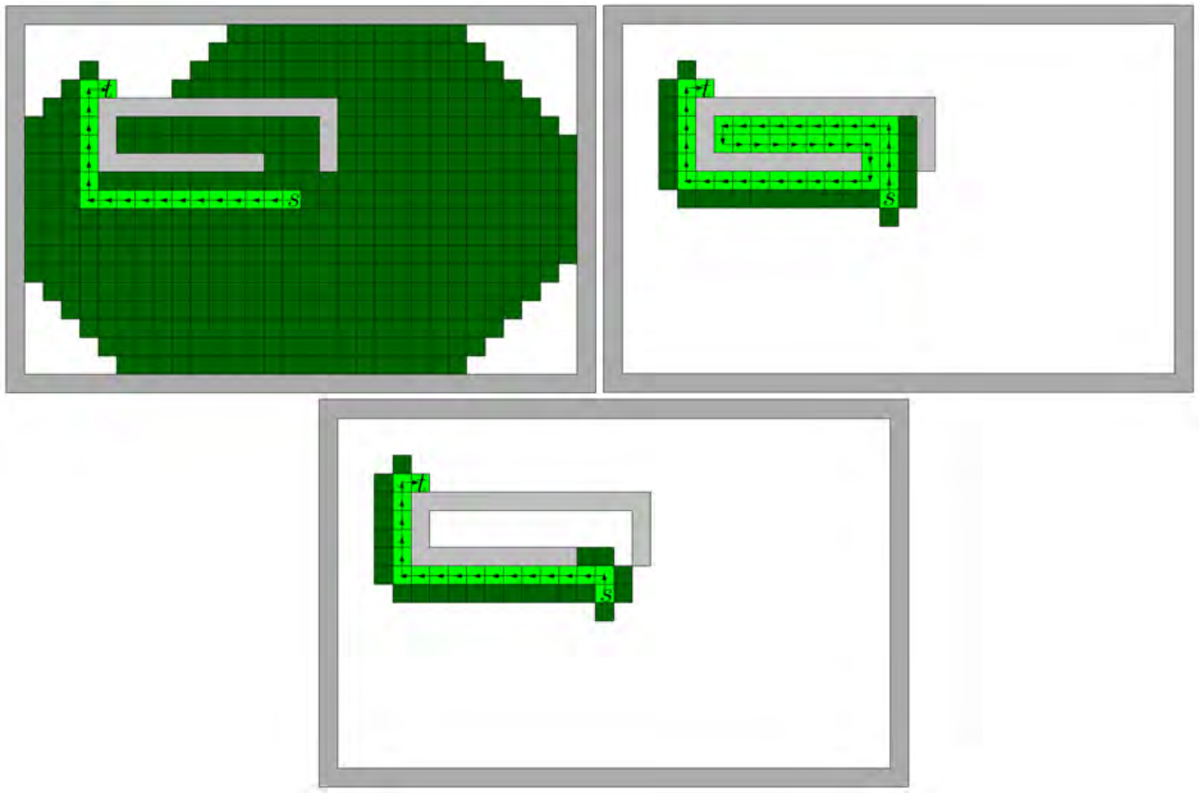
If we do not consider the heuristic function h and let $f = g$, we get *Dijkstra's algorithm* (1959). If we only consider h and ignore all the g-values in each node (i.e. $f = h$), we get a greedy *Best-First-Search* strategy. Therefore, A* can be seen as generalization of both strategies. Figure 8 illustrates the three different strategies on a grid with uniform costs. It shows that Dijkstra's algorithm finds an optimal path, but expands a high number of nodes. The greedy strategy expands only a few nodes, but the resulting path is far from optimal. A* combines both strategies, finding an optimal path while expanding only a few nodes. Note that in the example, we assume to have an exact heuristic h, which reduces the number of expanded nodes to a minimum.

In general, finding a feasible heuristic can be difficult because the quality of the heuristic depends on the environment. The Euclidean distance is a popular choice. However, it is not well-suited for maze-like environments in which the length of a path between two points may differ a lot from the Euclidean distance between them.

There are many more variants and modifications of A*. As the size of the open list strongly influences the computation time for the final path,

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 8. Comparison of different search strategies on a grid with uniform costs. Obstacles are shown in grey, free space is shown in white. Expanded cells are shown in dark green. Cells that are part of the final path are shown in bright green (with arrows pointing towards the next node on the final path). Top Left: Dijkstra's algorithm expands many nodes, but finds an optimal path. Top Right: Best-First-Search expands only a few nodes, but finds a non-optimal path. Bottom: A combines both advantages (an exact heuristic is chosen to illustrate the strength of A*).*



many variants aim at keeping the number of nodes in the open list small. For example, the size of the open list can be limited by a constant number of nodes, and nodes with the highest f-values can be dropped whenever the open list becomes bigger than that number (*beam search* (Lowerre & Reddy, 1980)).

Instead of generating one large set of opened and closed nodes, performance can also be improved by searching from s to t and from t to s simultaneously and stop when the two paths meet in the same node. However, the result is not guaranteed to be optimal.

Another concept is to inflate the h-values by some given weight $w \geq 1$ and to compute the f-values as $f = g + wh$. With this *weighted* A* variant (Pohl, 1970), additional emphasis is put on the heuristic. In this way, the expansion of nodes that appear to be closer to the goal are preferred, thus yielding a trade-off between computation time and quality of the path. This idea is further extended to *anytime variants* of A*, which start with a high weight for the heuristic values to compute a first rough solution quickly. This first solution is then improved over time by adjusting the weights. One of those anytime variants is ARA* (Likhachev, Gordon, & Thrun, 2004), in

Proofing copy. Copyright IGI Global. May not be posted or distributed.

which the weight w is based on a linear trajectory and two additional user-controlled parameters. The user has to set the initial value of w together with a fixed step size Δw by which the weight is decreased between the computation of solutions. Another anytime variant of A* is called ANA* (*Anytime Non-parametric A**) (van den Berg, Shah, Huang, & Goldberg, 2011). It uses a novel criterion for deciding which node to expand next in each step. Instead of expanding the node with lowest weighted f-value, it expands the node that maximizes the term $e = (G - g) / h$, with G being the costs of the currently best solution (which is set to infinity in the first iteration). The term e can be intuitively understood as the ratio between the "budget" that is left to improve the current-best solution and the estimated costs between the node and the goal. Maximizing e corresponds to picking the largest weight w such that $f \leq G$, and to expanding the node that is most promising to improve the current-best solution. ANA* overcomes the user's problem in ARA* of finding appropriate parameters, and has comparable and in some cases better performance than ARA*.

Despite the fact that A* and its variants are widely used for path planning in digital games, it is not the final answer. Many open research questions cannot be answered by using A* on a graph. Recent approaches use the aforementioned navigation meshes and the possibility to store additional information. Solving advanced path planning and crowd simulation problems on 2D-surface representations of the environment rather than on 1D representations such as graphs require new strategies, which we will discuss in the following sections.

The Indicative Route Method

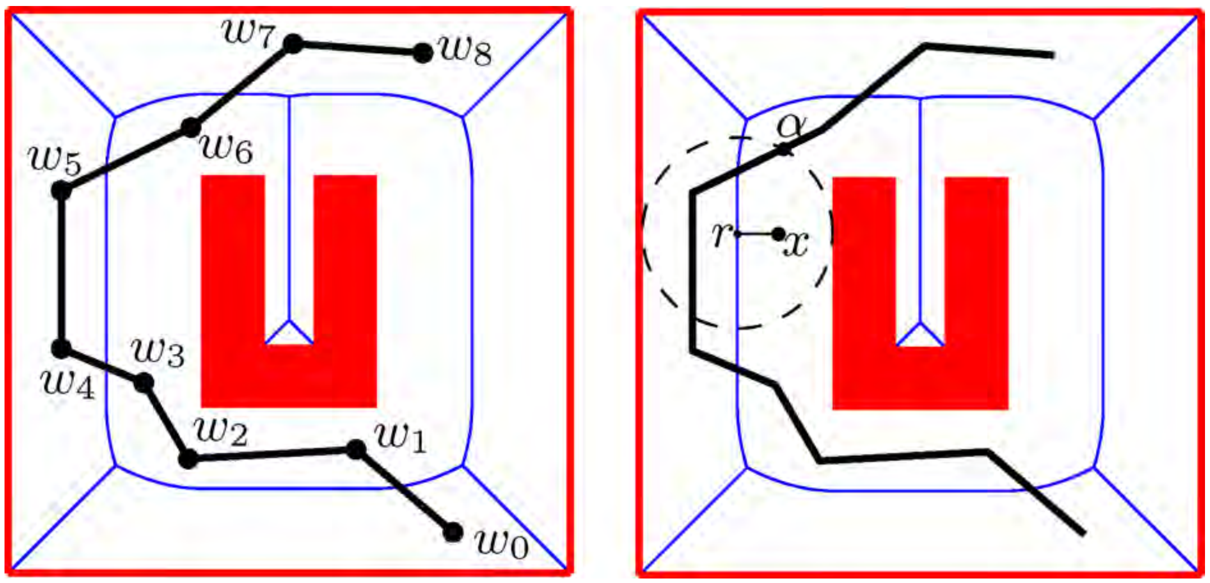
The *Indicative Route Method* (IRM) (Karamouzas, Geraerts, & Overmars, 2009) is a path planning strategy based on the *Explicit Corridor Map* (ECM); see Section "Navigation Meshes". The IRM combines the general concept of global planning using the ECM together with local force-based strategies. Given a character with a start position and a designated goal position, the method works as follows.

First, an indicative route is created. An indicative route is a route from the start to the goal position, and it functions as a guiding line and a rough estimation of the character's preferred route through the environment. This route can be either manually designed or automatically computed by a global path planning approach (e.g. by an A* search on a grid; see Sections "Grids" and "A* and its Variants"). We assume that the indicative route is given as a straight-line strip between a set of waypoints; see Figure 9 (left).

Second, the actual path is computed by letting the character move towards an attraction point in a number of discrete time steps. In each step, the character's current position x is *retracted* onto the medial axis. This retraction works as follows: We trace the line between x and its closest obstacle point in opposite direction until it intersects the medial axis. This intersection point is the retraction point of x, and it is denoted by r; see Figure 9 (right). Now the radius of the maximum clearance disc around r is given as the distance between r and its closest obstacle point. In other words, the maximum clearance disc is the disc centered at r that is as large as possible before it intersects an obstacle. If the goal position is contained inside the disc, the character will be attracted to it. Otherwise, the attraction point $\boldsymbol{\alpha}$ is defined as the last point on the indicative route that intersects the boundary of the clearance disc; see Figure 9 (right). The attraction point $\boldsymbol{\alpha}$, the character's current position x and its preferred speed induce a preferred velocity for the character in each step of the simulation. This velocity is then integrated using Euler's integration (Butcher, 2008) or a similar integration scheme to produce smooth and natural paths; see Figure 10 for an example in a 3D city environment.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 9. Left: an indicative route with waypoints $w_0, \ldots, w_8$. *Right: a character with center point* x, *its retraction point* r *on the medial axis, and attraction point as the intersection of the indicative route and the maximum clearance disc (dashed) around* r.



For a modest number of characters, the IRM runs at framerates suitable for real-time applications. However, the IRM has its drawbacks. For large crowds of tens of thousands of individual characters, the method is computationally inefficient. Furthermore, the retraction strategy requires indicative routes to have no self-intersections and backtracks, thus limiting the flexibility and power of the method for specific scenarios. Furthermore, as the definition of the attraction point depends on the available amount of free space around the character's current (retracted) position, the method causes undesired behavior in large areas of free space, but also in confined spaces. When there is a large area of free space around the character, the maximum clearance disc has a large radius. This results in attraction points being far ahead of the character's current position. The resulting path smoothes the indicative route to a great extent and skips parts that might be important. By contrast, in confined spaces such as a narrow but slightly winding and long passage, the maximum clearance is small. The IRM makes the character frequently change its orientation although a straight movement in the direction of the corridor is expected.

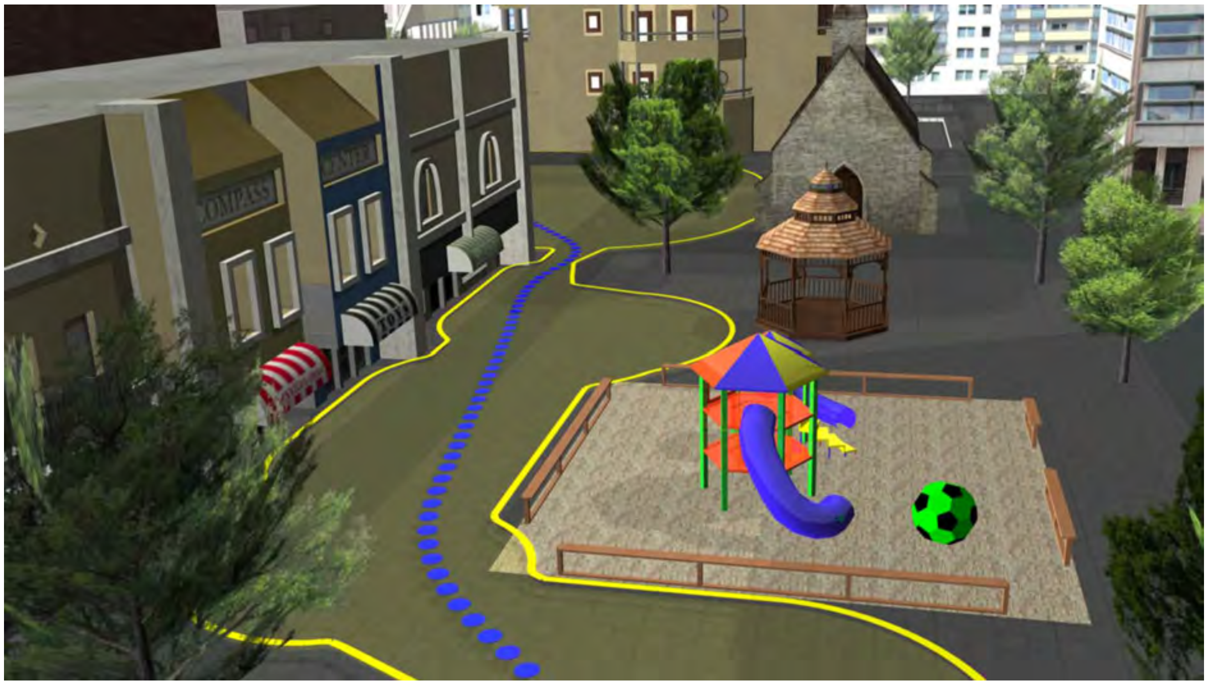
To address these limitations, the *Modified Indicative Routes and Navigation* method was developed. This method can be seen as a successor of the IRM, and we will describe it now in more detail.

The Modified Indicative Routes and Navigation Method

The *Modified Indicative Routes and Navigation* (MIRAN) method adopts from the IRM the concept of using an indicative route and computing an attraction point in each step of the simulation (Jaklin, Cook IV, & Geraerts, 2013). It computes several *candidate* attraction points in each step of the simulation. The best candidate is then picked according to a weight-function that takes region preferences into account.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 10. A smooth path computed by the Indicative Route Method using the Explicit Corridor Map (Geraerts, 2010)



MIRAN improves on the IRM in several aspects. Its main contribution is the simulation of heterogeneous environments for different character profiles. Each character profile has its own specific preferences for certain region types (roads, bike-lanes, sidewalks, lawns, etc.). MIRAN can be used for path following without requiring a copy of the navigable space for each character profile. Furthermore, the method allows the user to control how closely an indicative route should be followed and how much freedom a character has to deviate from or skip parts of it. Thus, the method is independent of local clearance information and overcomes the aforementioned problems of the IRM.

While MIRAN makes the above contribution, it has its limitations. MIRAN is defined for a character represented as a point in the environment, not a disc or similar 2D shape. Thus, different character sizes and arbitrary clearance from obstacles are not supported by this method. Furthermore, MIRAN does not handle collision avoidance with region-preferences. Characters do independently take regions into account while travelling, but they ignore this information as soon as collisions among them need to be avoided. Solving these issues is subject of current research.

The quality of paths generated with the MIRAN method also depends on the quality of the indicative routes that are used. These can be computed by using A* on a grid (Jaklin, Cook IV, & Geraerts, 2013), but recent approaches rather consider the exact geometry of the environment (Jaklin, Tibboel, & Geraerts, 2014) (see Figure 11).

Local Collision Avoidance

In this section, we give an overview of different strategies to handle local collision avoidance between characters. The algorithms can generally be divided into three categories. First, *reactive steering* methods let the character adapt its path

Proofing copy. Copyright IGI Global. May not be posted or distributed.

to dynamic and static obstacles. Second, *collision prediction* methods assume that a character moves with a constant velocity. Future motions are predicted by linearly extrapolating the character's current movement. In this way, collisions can be detected and avoided in advance. Third, *example-based* methods use a database of example behaviors from captured crowd data. A character selects an example behavior that closely matches its current state with respect to a given metric. We now discuss related work for each of the three collision avoidance categories.

Reactive steering methods originated from the robotics community. Force-based approaches are usually a variant of a potential field as described in Section "Potential Field and Flow-Based Methods". In the field of animation and crowd simulation, reactive steering was introduced by the seminal work of Reynolds on flocks, herds and schools (Reynolds C. W., 1987). In this model, three behavioral rules are applied to each flock member: avoiding collision with nearby flock members, matching velocity with the nearby flock members, and staying close to nearby flock members. Collision avoidance is achieved by repulsive forces that are exerted on nearby flock members. The shorter the distance between two flock members, the stronger the repulsive force. Velocity matching can be seen as predictive collision avoidance because flock members that successfully match their velocities are unlikely to collide. The three behavioral rules are ordered by priority and accumulated to produce the flock members motion. The order can be dynamically adapted during the simulation to temporarily split the flock for avoiding collisions.

Another reactive and force-based approach popular in the civil and traffic engineering community is the social force model introduced by Helbing and Molnár (1995). The behavior of pedestrian crowds is simulated by solving Newton's equations of motion, using both physical and socio-psychological forces. Repulsive and tangential forces are used to model interactions between individuals with respect to their relative distance, and to avoid collisions with obstacles. Heigas, Luciani, Thollot and Castagné (2003) propose an approach in which characters and obstacles are represented as particles. Interactions are represented by physically-based forces exerted between each pair of particles using a mass-spring-damper model. The distance between interacting particles influences the stiffness and viscosity of the model.

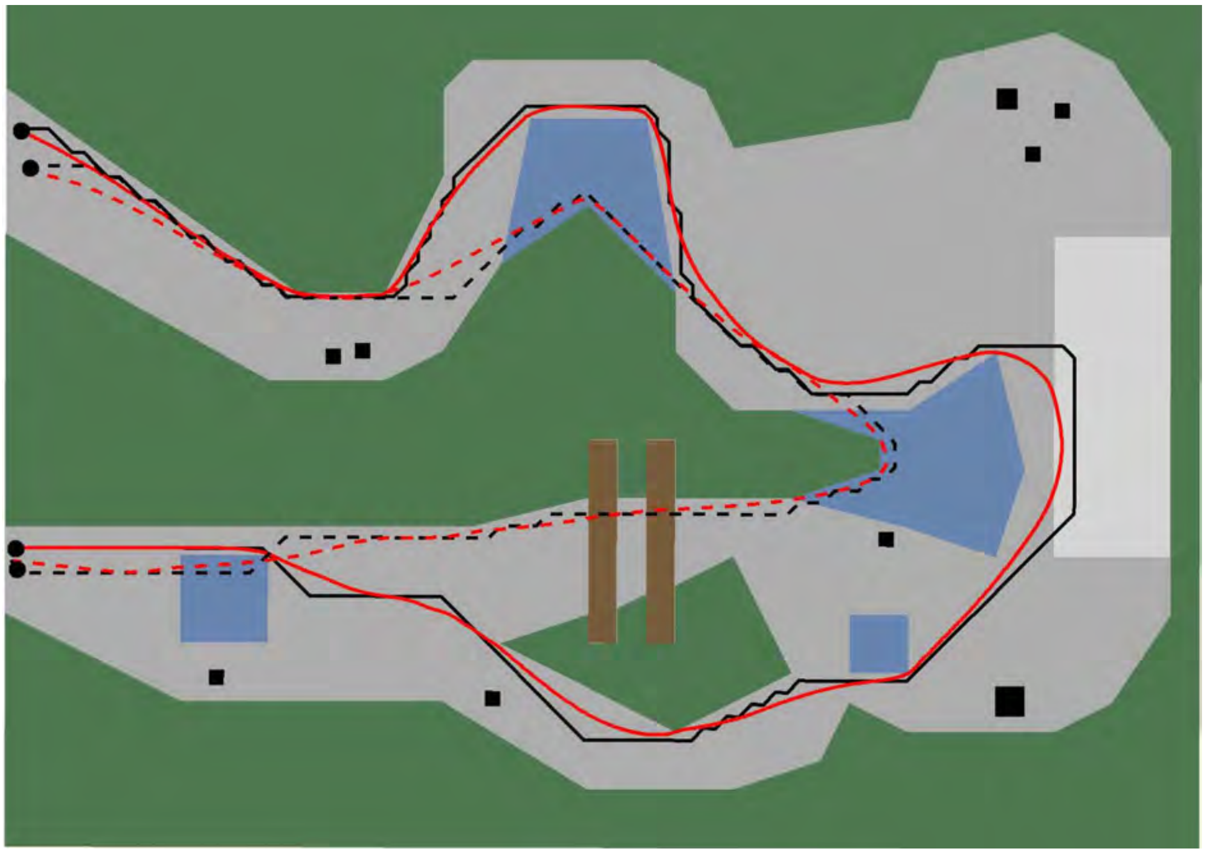
Reactive steering methods need not necessarily be force-based. Other reactive approaches comprise rule-based systems on a 2D grid (Loscos, Marchal, & Meyer, 2003) or by using cellular automata (Wolfram, 1994). Behavioral models such as the work of Terzopoulos, Tu, and Grzeszczuk (1994) to simulate artificial fish also belong to this category.

Reactive steering methods have the drawback that characters react to avoid collisions when other characters or obstacles get sufficiently close. This may result in unnatural trajectories because humans tend to resolve collisions in advance by the principle of least effort (Zipf, 1949). Furthermore, the resulting motions are often subject to oscillations.

Predictive methods address these issues. Reynolds (1999) presents a collection of steering behaviors, one of them being the *unaligned collision avoidance behavior*. Collisions are predicted by extrapolating current motions and resolving them by adjusting orientation and movement speed accordingly. Pettré, Ondřej, Olivier, Cretual, and Donikian (2009) describe a model based on experimental study. In this model, pedestrians take different roles during interactions (e.g. passing first or giving way). Adaptations of speed and orientation to avoid collisions depend on the specific roles of the characters. Among the predictive methods are also the more recent *vision-based* approaches. Those are based on the assumption that paths are most natural if decisions are based on the characters' visual stimuli. Ondřey, Pettré, Olivier, and Donikian (2010) combine

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 11. The MIRAN method in a forest environment. An adult character (solid non-smooth indicative route and solid smooth path,) avoids puddles and dense wood, but is attracted to a region with a panoramic view (light grey). A child (dashed non-smooth indicative route and dashed smooth path) crosses puddles and climbs fallen tree logs (Jaklin, Cook IV, & Geraerts, 2013).



visual stimuli with motor response laws. Fiorini and Shiller (1998) introduced the concept of a *Velocity Obstacle* (VO). For each character, the VO represents the set of all velocities that would lead to a collision with another character at some future time step. By taking the union of VOs for all characters under consideration, collisions can be avoided by selecting a velocity outside of the combined VO. Van den Berg et al. (2008) extended this concept to the *Reciprocal Velocity Obstacle* (RVO) that guarantees oscillation-free motions. A variety of models have improved on the VO and RVO concepts; see for example (Snape, Berg, Guy, & Manocha, 2009; Guy, et al., 2009; van den Berg, Guy, Lin, & Manocha, 2009). Vision-based approaches tend to be computationally expensive. This becomes particularly relevant when simulating crowds with high densities. In those scenarios, flow-based methods may be preferred (Treuille, Cooper, & Popovic, 2006). To further overcome the problems of short-range avoidance strategies, *long-range* collision avoidance has recently been taken into consideration (Golas, Narain, Curtis, & Lin, 2014; Anvari, Bell, Angeloudis, & Ochieng, 2014).

Among the example-based techniques are the works of Ju et al. (2010), Lee, Choi, Hong, and Lee (2007) and Lerner et al. (2007). Using a database of captured crowd behaviors, each character selects an example behavior that is close to its current

Proofing copy. Copyright IGI Global. May not be posted or distributed.

state with respect to a given metric. Example-based models may simulate realistic motions and capture complex avoidance maneuvers based on actions taken by real members of a crowd. However, corresponding databases are limited to the number of different behaviors. Situations may occur in which all matching examples lead to collisions. In addition, the scenario must always be similar to the real-life scenario from which the examples were extracted. Lastly, example-based solutions are rather used for offline-simulations and are not applicable to real-time applications such as digital games due to a high computational complexity.

As we have described in this section, there is a great variety of strategies that all have its advantages and drawbacks. Factors that influence the choice for a specific method are: What number of characters does a game need to handle simultaneously? Does it have large crowds or few individual characters? What is the desired degree of realism? How many collisions are expected in each time frame, and how important are low computation times for resolving those collisions? How is traversable space represented, and what collision avoidance strategies are best suited for this representation? Answers to such questions need to be answered in the design phase of a game.

CROWD SIMULATION STRATEGIES

The strategies we discussed in the previous sections mainly focus on computing feasible path trajectories for individual characters. With modern hardware, they can be used to simulate a crowd by applying the corresponding techniques to all crowd members simultaneously and adding local collision-avoidance techniques. A different approach is to handle a crowd as one large entity by simulating the motion of each member as a per-particle energy minimization, thus handling a crowd similar to a fluid or gas. We now give an overview of such models.

Potential Fields and Flow-Based Methods

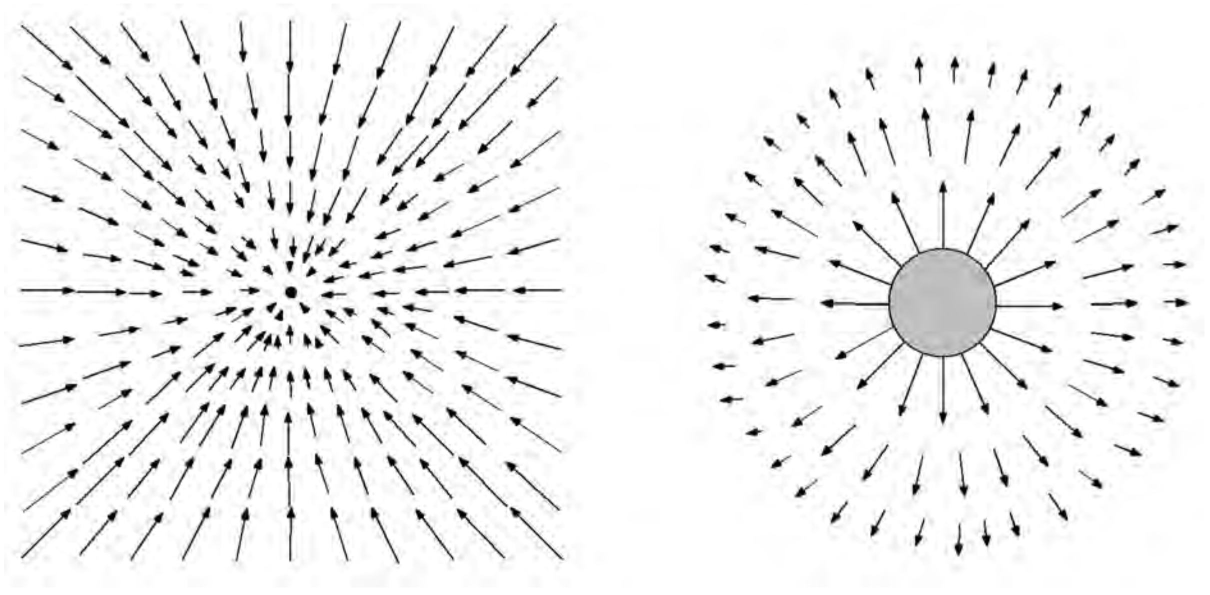
Using *Potential Fields* for path planning is an approach that has emerged from the field of robotics (Latombe, 1991). The general task in behavior-based robotics is to have a robot plan and execute its actions in three steps. First, it receives input through its sensors. Second, it evaluates the input to compute a desired behavior (i.e. *seek the goal point* or *avoid a wall*). Third, it maps this plan to a corresponding sequence of motor actions. For path planning in digital games, we are mainly interested in the second step of this approach. Potential Fields are widely used in this context because they are an intuitive and well-studied mathematical concept.

Moving characters in a potential field can be compared to a leaf floating on a stream of water, or a marble rolling through an environment of steep ramps or hills. The surrounding local topology or geometry of the environment directly influences the path of the moving object.

From an abstract point of view, a potential field can be described as a mapping from an input vector to an output vector. In robotics, the input vector usually describes the robot's current orientation and speed, and the same can be applied to moving characters in a digital game or simulation. The output vector describes a motion from the current position to the next. If the environment is represented by a finite and discrete approximation of all traversable space (e.g. by a grid, see Section "Grids"), the potential field can be visually shown as a *vector field*. It shows the union of all output vectors for every possible location of the character in the environment; see Figure 12. It might be necessary to compute the whole set of output vectors for large crowds that have a shared goal. However, when using potential fields for a small number of characters, the whole set of output vectors is hardly ever computed and only the ones are calculated that are actually necessary for the characters' motion.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Figure 12. Left: an attractive potential field with a goal point. Right: a repulsive potential field with an obstacle.



For path planning purposes, a given goal point in the environment acts as an attractor for the character. The corresponding potential field applies forces on the character such that it is directed towards the goal; see Figure 12 (left). On the contrary, obstacles apply a repulsive force on the characters to avoid collisions; see Figure 12 (right).

Just like a rolling marble can get stuck in a hole or valley, characters can easily get stuck in local minima of the potential field. This is one of the main drawbacks when using potential fields for path planning. Potential fields that are free of local minima do exist, but they are expensive to compute (Connolly & Grupen, 1992), (Rimon & Koditschek, 1992), (Sundar & Shiller, 1994). Workarounds are usually considered to handle this problem (e.g. predict the next few steps and change a character's motion if the current motion will lead to a local minimum).

Other flow-based methods have also been used for crowd simulation. Hughes (2003) modeled human pedestrians as a continuous density field, using partial differential functions to describe crowd dynamics. Treuille, Cooper, and Popovic (2006) presented a real-time crowd model based on continuum dynamics. Their model exhibits emergent phenomena observed in real crowds such as lane formation or vortex formation when several groups cross each other. Furthermore, it can be used in real-time applications. The authors showed that they can efficiently steer two opposing armies or let a crowd in a city react in panic to a user-controlled flying-saucer. Kerr and Spears (2006) presented a simulation model based on gas-kinetics for mobile robots. Pimenta, Michael, Mesquita, Pereira, and Kumar (2008) used a model based on Smoothed Particle Hydrodynamics to simulate swarms of mobile robots.

The main advantage of the abovementioned models is that they can efficiently simulate large numbers of characters, as long as there are only a few goal-states involved. In such cases, they usually outperform agent-based models in terms of computation times. However, the computational complexity of such models becomes too high for real-time applications when a large number of different crowds with different goal states needs to

Proofing copy. Copyright IGI Global. May not be posted or distributed.

be simulated. Furthermore, flow-based models are not well-suited for low- to medium crowd densities, in which the individuality of each crowd member has a larger impact on the overall crowd behavior than in high-density scenarios. Lastly, none of the above models focuses on social interactions among crowd members. We now give an overview of models that simulate groups, formations and social interactions.

Groups and Formations

Simulating social interactions among the members of a crowd is a challenging step towards more immersive educational games. Pedestrians in a city environment such as *The River City Project* (2002 - 2007) should not walk individually, but form small social groups, walk together and automatically adapt to dynamic changes like spotting a friend in a crowd.

In this section, we give a brief overview of research on simulating small and large social groups of virtual characters. The basis for many of these approaches is the behavioral model by Reynolds to simulate flocks, herds, and schools (Reynolds C. W., 1987), which the author later extended by adding steering behaviors (Reynolds, 1999) and interactions within groups (Reynolds, 2000). Since this model mainly aims at simulating the behavior of groups of animals, other researchers have focused on autonomous *human* characters.

Musse and Thalmann (1997) presented a rule-based model for crowd behavior of multiple pedestrian groups. Characters in this model exhibit flocking behavior and are able to switch between groups based on sociological factors. The authors do not elaborate on how group coherence is being addressed, and the proposed collision-avoidance model yields undesired behavior compared to state-of-the-art methods described in Section "Local Collision Avoidance".

Kamphuis and Overmars (2004) proposed a path planning model that maintains group coherence and avoids spatial separation between members of the same group. The model focuses on large groups such as armies, and is thus well-suited for military games and applications rather than simulations of small groups in which social factors determine the overall behavior.

Kimmel, Dobson and Bekris (2012) also addressed the problem of maintaining group coherence. Their work is based on the *Velocity Obstacle* approach by Fiorini and Shiller (1998), and it extends this method by adding team behavior. The limitations of this model are that the characters all move at the same speed and that only one single formation is being kept.

Qiu and Hu (2010) presented a model to simulate dynamic groups based on utility theory and social comparison theory. Their model allows characters to dynamically leave and join groups based on spatial and social factors.

Park, Quek, and Cao (2012) presented a model that is based on *common ground theory.* This model aims at simulating the social interactions between group members. Each group has a leader, and group coherence is measured by the distance from a group member to its leader projected in the leader's direction of motion.

Karamouzas and Overmars (2012) presented a method of how to simulate local behavior of small pedestrian groups consisting of two or three individuals in city environments. Their approach is based on empirical studies regarding the spatial organization of such groups (Moussaïd, Perozo, Garnier, Helbing, & Theraulaz, 2010), (Peters & Ennis, 2009). From those empirical studies, they derived three formations that small groups tend to adapt to, namely *line-abreast, v-like* and *river-like* formations. The *line-abreast* formation functions as the default one when the group members communicate with each other while moving towards their goal. It is best suited for keeping the social interaction, provided there is enough space to do so. If the environment becomes more crowded or obstacles prevent the *line-abreast* formation, the *v-like* formation is taken. It still provides space for communication between the group members

Proofing copy. Copyright IGI Global. May not be posted or distributed.

while adapting to the environmental situation. The *river-like* formation is taken when crowd density is too high to keep the *v-like* formation, or when the group has to move through a small hallway or an equivalent environmental setup. For groups of two pedestrians, the *v-like* formation is replaced by a more compact line-abreast formation that reduces the distance between the two members.

The model by Karamouzas and Overmars (2012) has been tested and visualized in different scenarios. Group interactions have been tested in a narrow corridor, a shopping mall and a busy crosswalk. The virtual shopping mall scenario was compared to original video footage of a real shopping mall. Different metrics have been defined to measure the quality of the results in a quantitative evaluation of the model.

Similar to collision-avoidance methods, it depends on the application at hand which crowd simulation model is best. A wide range of approaches and paradigms exist, and most of the recent models work well in specific scenarios. Some of them are beyond the scope of a typical gaming application, as they try to handle complex factors among social groups from fields such as sociology or psychology. With games becoming more and more realistic, though, these kind of factors might one day be a crucial and integral component of immersive gaming experiences, too.

FUTURE WORK

The advanced algorithms we have discussed in this chapter could be used in future educational games to enhance a player's immersion and thus the overall learning experience. Autonomous virtual characters could change their appearance from static, stiff and unnatural to lively, dynamic and more believable by using the methods we covered in this chapter.

For instance, in a game such as *The River City Project* (2002 - 2007), a navigation mesh such as the *Explicit Corridor Map* (Geraerts, 2010) could enable the residents to autonomously move around the city with respect to the exact geometry of the walkable space. Using a method such as MIRAN (Jaklin, Cook IV, & Geraerts, 2013) makes them stay on preferred terrain and avoid less attractive regions without treating such regions as hard obstacles, thus allowing more flexible behavior and creating smooth and visually convincing paths. Social-group methods such as (Karamouzas & Overmars, 2012) could be used to simulate the residents' social behavior. The characters could walk individually, but temporarily form small social groups to have conversations or walk together in socially-friendly formations. Novel crowd simulation models enable the simulation of large crowds at interactive rates for both low- and high-density situations (van Goethem, Jaklin, Cook IV, & Geraerts, 2015). These could be used to populate a virtual environment like *River City* with even more characters that coordinate their walking behavior and react to dynamic changes in the environment.

Furthermore, virtual crowds are essential in training software for evacuation scenarios and crowd management such as *SportEvac* (2014). Such games could not only be used by managers of mass events and safety personnel, but also in P-12 education to increase the students' awareness of potential dangers during such events.

The aforementioned methods could not only improve a player's immersion to a great extent, but they could also enable advanced gameplay and novel game-design elements in future educational games. For example, virtual characters that display social behavior could be used in a game that aims at training communication skills. A player's learning experience could be enhanced when the virtual characters not only react to the player's negative actions by simply stating that they are displeased, but actually start avoiding the player and tell their fellow characters about the incident, who in turn also display a negative attitude towards the player.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Future educational games could also simulate social-group behavior on a psychological level. The question of how individual characters meet, automatically form groups and depart has been addressed, but still leaves room for improvement. Studies from fields such as psychology and sociology should be used as a basis for more in-depth methods to simulate human behavior.

CONCLUSION

As discussed in the previous sections, educational games often lack realism when characters need to navigate through a virtual environment. Characters display stiff movements and lack flexibility in the paths they choose. For instance, paths are often scripted and manually created by a designer. The resulting repetitive behavior reveals the underlying technological aspects of a game, which in turn destroys a player's immersion and the overall learning experience.

Many different approaches have been taken to tackle path planning and crowd simulation issues in virtual environments. While the A* method on a grid is a flexible and general approach that has been widely and successfully used in the past, it is not the final answer to problems related to path planning and crowd simulation. We have discussed that surface-based representations of traversable space via navigation meshes gives developers the opportunity to create more advanced algorithms.

For all problems we have discussed in this chapter, a general and theoretically substantiated basis is missing. Therefore, novel approaches need to be taken, and new algorithms need to be developed to enhance future educational games with respect to their path planning and corresponding game design possibilities. We believe that the end of graph-based methods is near, and that the future lies in combining data structures and novel algorithms that work on a surface-based representation of the virtual environment.

In this chapter, we have mainly focused on the navigation aspects of path planning and crowd simulation for educational games. Other important aspects are behavioral patterns, visualization and animation. These are addressed in the book by Thalmann and Musse (2013).

Overall it can be said that the field of path planning and crowd simulation is continually growing and improving. Promising new ideas might soon evolve into algorithms providing many immersive features, thus enhancing a player's gaming experience and improving learning results in educational games for training and education purposes.

ACKNOWLEDGMENT

This work was funded by the COMMIT/ project, http://www.commit-nl.nl.

REFERENCES

Anvari, B., Bell, M. G. H., Angeloudis, P., & Ochieng, W. Y. (2014). Long-range Collision Avoidance for Shared Space Simulation based on Social Forces. *Transportation Research Procedia*, *2*, 318–326. doi:10.1016/j.trpro.2014.09.023

Atari Inc. (1979). USA: Asteroids.

Butcher, J. (2008). *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons. doi:10.1002/9780470753767

Connolly, C., & Grupen, R. (1992). Harmonic Control. *Proceedings of the 1992 International Symposium on Intelligent Control*, (pp. 503-506). doi:10.1109/ISIC.1992.225142

de Berg, M., van Kreveld, M., Overmars, M., & Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications* (2nd ed.). Springer-Verlag. doi:10.1007/978-3-662-04245-8

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Dijkstra, E. (1959). A Note on two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*(1), 269–271. doi:10.1007/BF01386390

Entertainment, B. (2010). *Starcraft II*. USA.

Fiorini, P., & Shillert, Z. (1998). Motion Planning in Dynamic Environments using Velocity Obstacles. *The International Journal of Robotics Research*, *17*(7), 760–772. doi:10.1177/027836499801700706

Gee, J. (2007). *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan.

Geraerts, R. (2010). Planning Short Paths with Clearance using Explicit Corridors. Proceedings of the *IEEE International Conference on Robotics and Automation* (pp. 1997-2004). doi:10.1109/ROBOT.2010.5509263

Geraerts, R., & Overmars, M. (2007). The Corridor Map Method: A general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds*, *18*(2), 107–119. doi:10.1002/cav.166

Ghosh, S. K. (2007). *Visibility Algorithms in the Plane*. Cambridge University Press. doi:10.1017/CBO9780511543340

Golas, A., Narain, R., Curtis, S., & Lin, M. C. (2014). Hybrid Long-Range Collision Avoidance for Crowd Simulation. *IEEE Transactions on Visualization and Computer Graphics*, *20*(7), 1022–1034. doi:10.1109/TVCG.2013.235 PMID:24080711

Grand Theft Auto V. (2013). Rockstar Games. USA.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M. C., Manocha, D., & Dubey, P. (2009). Clearpath: Highly parallel collision avoidance for mulit-agent simulation. Proceedings of the *ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation (SCA '09)* (pp. 177-187). doi:10.1145/1599470.1599494

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. doi:10.1109/TSSC.1968.300136

Heigeas, L., Luciani, A., Thollot, J., & Castagné, N. (2003). *A Physically-Based Particle Model of Emergent Crowd Behaviors*. Graphicon.

Helbing, D., & Molnár, P. (1995). Social force model for pedestrian dynamics. *Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, *51*(5), 4282–4286. doi:10.1103/PhysRevE.51.4282 PMID:9963139

Held, M. (2011). VRONI and ArcVRONI: Software for and Applications of Voronoi Diagrams in Science and Engineering. *Proceedings of the 8th International Symposium on Voronoi Diagrams in Science and Engineering* (pp. 3-12). doi:10.1109/ISVD.2011.9

Hoff, K., Keyser, J., Lin, M., Manocha, D., & Culver, T. (1999). Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques SIGGRAPH '99* (pp. 277-286). doi:10.1145/311535.311567

Hughes, R. (2003). The Flow of Human Crowds. *Annual Review of Fluid Mechanics*, *35*(1), 169–182. doi:10.1146/annurev.fluid.35.101101.161136

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Jaklin, N., Cook, I.V.A. IV, & Geraerts, R. (2013). Real-time Path Planning in Heterogeneous Environments. [CAVW]. *Computer Animation and Virtual Worlds*, *24*(3), 285–295. doi:10.1002/cav.1511

Jaklin, N., Tibboel, M., & Geraerts, R. (2014). Computing High-Quality Paths in Weighted Regions. *Proceedings of the 7th International ACM SIGGRAPH Conference on Motion in Games (MIG 2014)* (pp. 77-86).

Jeuring, J., van Rooij, R., & Pronost, N. (2014). The 5/10 Method: A Method for Designing Educational Games. In Games and Learning Alliance (pp. 364-369). Springer International Publishing.

Ju, E., Choi, M. G., Park, M., Lee, J., Lee, K. H., & Takahashi, S. (2010). Morphable crowds. *ACM Transactions on Graphics*, *29*(6), 1. doi:10.1145/1882261.1866162

Kallmann, M. (2010a). Navigation queries from triangular meshes. *Proceedings of the Third international conference on Motion in Games* (pp. 230-241). Springer-Verlag.

Kallmann, M. (2010b). Shortest paths with arbitrary clearance from navigation meshes. *Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA).*

Kamphuis, & Overmars. (2004). Finding paths for coherent groups using clearance. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '04)* (pp. 19-28).

Karamouzas, I., Geraerts, R., & Overmars, M. (2009). Indicative routes for path planning and crowd simulation. *Proceedings of the 4th international conference on foundations of digital games FDG '09* (pp. 113-120). doi:10.1145/1536513.1536540

Karamouzas, I., & Overmars, M. (2012). Simulating and evaluating the local behavior of small pedestrian groups. *IEEE Transactions on Visualization and Computer Graphics*, *18*(3), 394–406. doi:10.1109/TVCG.2011.133 PMID:22241282

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580. doi:10.1109/70.508439

Kerr, W., & Spears, D. (2006). Robotic Simulation of Gases for a Surveillance Task. *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1160-1168).

Ketelhut, D. (2007). The Impact of Student Self-efficacy on Scientific Inquiry Skills: An Exploratory Investigation in River City, a Multi-user Virtual Environment. *Journal of Science Education and Technology*, *16*(1), 99–111. doi:10.1007/s10956-006-9038-y

Ketelhut, D., Dede, C., Clarke, J., Nelson, B., & Bowman, C. (2007). Studying situated learning in a multi-user virtual environment. In *Assessment of problem solving using simulations* (pp. 37–58). Lawrence Erlbaum Associates.

Kimmel, D., & Bekris (2012). Maintaining team coherence under the velocity obstacle framework. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '12)* (Vol. 1, pp. 247-256).

Klein, F. (1882). *Eric W. Weisstein - MathWorld– A Wolfram Web Resource.* Retrieved from http://mathworld.wolfram.com/KleinBottle.html

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers. doi:10.1007/978-1-4615-4022-9

Laumond, J. (1987). Obstacle growing in a non-polygonal world. *Information Processing Letters*, *25*(1), 41–50. doi:10.1016/0020-0190(87)90091-3

Proofing copy. Copyright IGI Global. May not be posted or distributed.

LaVælle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning [Technical report].

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press. doi:10.1017/CBO9780511546877

Lee, K. H., Choi, M. G., Hong, Q., & Lee, J. (2007). Group behavior from video: a data-driven approach to crowd simulation. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '07)* (pp. 109-118).

Lerner, A., Chrysanthou, Y., & Lischinski, D. (2007). Crowds by Example. *Computer Graphics Forum, 26*(3), 655-664.

Likhachev, M., Gordon, G., & Thrun, S. (2004). ARA*: Anytime A* with Provable Bounds on Sub-Optimality. Advances in Neural Information Processing Systems 16 *Proceedings of the 2003 Conference (NIPS-03)*. MIT Press.

Loscos, C., Marchal, D., & Meyer, A. (2003). Intuitive crowd behaviour in dense urban environments using local laws. *Proceedings of the theory and practice of computer graphics*. IEEE Computer Society. doi:10.1109/TPCG.2003.1206939

Lowerre, B., & Reddy, D. (1980). The harpy speech understanding system. *Trends of Speech Recognition*.

Masehian, E., & Amin-Naseri, M. (2004, June). A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Field Robotics*, *21*(6), 275–300.

Minecraft. (2009). Mojang.

Möbius, F., & Listing (independently), J. B. (1858). *Eric W. Weisstein - MathWorld–A Wolfram Web Resource*. Retrieved from http://mathworld.wolfram.com/MoebiusStrip.html

Mononen, M. (2009). *recastnavigation*. Retrieved from https://github.com/memononen/recastnavigation

Monument Valley. (2014). Ustwo.

Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., & Theraulaz, G. (2010). The walking behavior of pedestrian social groups and its impact on crowd dynamics. *PLoS ONE*, *5*(4), e10047. doi:10.1371/journal.pone.0010047 PMID:20383280

Murray, B., Bogost, I., Mateas, M., & Nitsche, M. (2006). Game design education: Integrating computation and culture. *Computer*, *39*(6), 43–51. doi:10.1109/MC.2006.195

Musse, & Thalmann. (1997, September 2–3). A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation 97, Proceedings of the Eurographics Workshop in Budapest, Hungary* (pp. 36-51). Vienna: Springer.

Nash. (2012). *Any-Angle Path Planning* [PhD thesis]. University of Southern California.

Super Mario Galaxy. (2007 Nintendo, E.A.D. *Tokyo*, Japan.

Super Paper Mario. (2007). Nintendo SPD.

Ondřey, J., Pettré, J., Olivier, A.-H., & Donikian, S. (2010). A Synthetic-Vision Based Steering Approach for Crowd Simulation. *ACM Transactions on Graphics*, *29*(4).

Pac-Man. (1980). Namco. Japan.

Descent. (1994). Parallax Software.

Park, Q., & Cao. (2012). Modeling social groups in crowds using common ground theory. *Proceedings of the Winter Simulation Conference (WSC '12)* (pp. 1-12). doi:10.1109/WSC.2012.6465119

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Pelechano, N., Allbeck, J. M., & Badler, N. I. (2007). Controlling individual agents in high-density crowd simulation. *Proceedings of the 2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '07)* (pp. 99-108).

Peters, C., & Ennis, C. (2009). Modeling groups of plausivle virtual pedestrians. *IEEE Computer Graphics and Applications*, *29*(4), 54–63. doi:10.1109/MCG.2009.69 PMID:19798863

Pettré, J., Laumond, J.-P., & Thalmann, D. (2005). A navigation graph for real-time crowd animation and mulitlayered uneven terrain. *Proccedings of the first international workshop on crowd simulation (V-CROWDS'05).*

Pettré, J., Ondřej, J., Olivier, A.-H., Cretual, A., & Donikian, S. (2009). Experiment-based modeling, simulation and validation of interactions between virtual walkers. *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09)* (pp. 189-198). doi:10.1145/1599470.1599495

Pimenta, L., Michael, N., Mesquita, R., Pereira, G., & Kumar, V. (2008). Control of Swarms Based on Hydrodynamic Models. *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)* (pp. 1948-1953). doi:10.1109/ROBOT.2008.4543492

Pohl, I. (1970). Heuristic Search viewed as Path Finding in a Graph. *Artificial Intelligence*, *1*(3), 193–204. doi:10.1016/0004-3702(70)90007-X

Qiu, & Hu. (2010). Modeling group structures in pedestrian crowd simulation. *Simulation Modelling Practice and Theory, 18*(2), 190-205.

Rabin, S. (2008). *AI Game Programming Wisdom 4*. Charles River Media.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*(4), 25–34. doi:10.1145/37402.37406

Reynolds (1999). Steering Behaviors for Autonomous Characters. *Proceedings of the Game Developers Conference 1999* (pp. 763-782).

Reynolds. (2000). Interaction with groups of autonomous characters. *Game Developers Conference 2000* (pp. 449-460).

Rimon, E., & Koditschek, D. (1992). Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, *8*(5), 501–518. doi:10.1109/70.163777

Schmitz, B., Specht, M., & Klemke, R. (2012). An Analysis of the Educational Potential of Augmented Reality Games for Learning. *Proceedings of the 11th World Conference on Mobile and Contextual Learning 2012* (pp. 140-147).

INCONTROL Simulation Solutions. (2014). SportEvac.

Snape, J., Berg, J. v., Guy, S. J., & Manocha, D. (2009). Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 5917-5922).

Space Invaders. (1978Taito Corporation. Japan.

Sundar, S., & Shiller, Z. (1994). Optimal obstacle avoidance based on the hamilton-jacobi-bellmann equation. *ICRA*, *94*, 2424–2429.

Terzopoulos, D., Tu, X., & Grzeszczuk, R. (1994). Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, *1*(4), 327–351. doi:10.1162/artl.1994.1.4.327

Thalmann, & Musse (2013). *Crowd Simulation.* Springer.

The Boost C++ Library. (2015). Retrieved from http://www.boost.org/

The River City Project. (2002 - 2007). Retrieved from http://muve.gse.harvard.edu/rivercityproject/index.html

Proofing copy. Copyright IGI Global. May not be posted or distributed.

The Settlers II (Original German title: Die Siedler II: Veni, Vidi, Vici). (1996). Blue Byte Software. Germany.

Treuille, A., Cooper, S., & Popovic, Z. (2006). Continuum Crowds. *ACM Transactions on Graphics*, *25*(3), 1160–1168. doi:10.1145/1141911.1142008

Trovato, K. (1996). *A* Planning in Discrete Configuration Spaces of Autonomous Systems* [PhD Thesis]. University of Amsterdam.

Counter Strike: Source. (2004). Valve Corporation. USA.

Portal. (2007). *Valve Corporation*. USA.

van den Berg, J., Guy, S. J., Lin, M. C., & Manocha, D. (2009). Reciprocal n-body collision avoidance. Proceedings of the *International Symposium on Robotics Research*.

van den Berg, J., Lin, M. C., & Machona, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. Proceedings of the *IEEE International Conference on Robotics and Automation* (pp. 1982-1935). doi:10.1109/ROBOT.2008.4543489

van den Berg, J., Shah, R., Huang, A., & Goldberg, K. (2011). ANA*: Anytime Non-Parametric A*. *Proceedings of the AAAI conference on artificial intelligence*.

van Goethem, A., Jaklin, N., Cook, I. V. A., & Geraerts, R. (2015). *On Streams and Incentives: A Synthesis of Individual and Collective Crowd Motion* [Technical Report UU-CS-2015-005]. Utrecht University.

van Toll, W., Cook, I. V. A. IV, & Geraerts, R. (2012). A navigation mesh for dynamic environments. *Computer Animation and Virtual Worlds*, *23*(6), 536–546. doi:10.1002/cav.1468

Wein, R., van den Berg, J., & Halperin, D. (2007). The visibility-voronoi complex and its applications. *Computational Geometry*, *36*(1), 66–87. doi:10.1016/j.comgeo.2005.11.007

Welzl, E. (1985). Constructing the Visibility Graph for n Line Segments in O(n^2) time. *Information Processing Letters*, *20*(4), 167–171. doi:10.1016/0020-0190(85)90044-4

Wolf, M. (2009/2010). *Theorizing Navigable Space in Video Games*. Retrieved from http://opus.kobv.de/ubp/volltexte/2011/4980/

Wolfram, S. (1994). *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley.

Yu, K. (2006). Finding a natural-looking path by using generalized visibility graphs. *Proceedings of the 9th Pacific Rim international conference on Artificial Intelligence* (pp. 170-179). doi:10.1007/978-3-540-36668-3_20

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press.

KEY TERMS AND DEFINITIONS

Character: Any moving entity in a digital game, either autonomous or controlled by the player.

Crowd: A large number of static or moving characters in a digital game, either with a shared goal or with individual goals.

Indicative Route: A rough path from a start to a goal position, indicating the approximate direction a character should follow. Used for guiding a character, but not used as a final path.

Local Collision Avoidance: The task to detect and avoid a collision among several moving characters on a local level, thus not taking global knowledge of the entire environment into account.

Navigation Mesh: A data structure to represent the traversable space in a digital game, consisting of 2D polygons.

Proofing copy. Copyright IGI Global. May not be posted or distributed.

Path Planning: The task to compute and follow a path from a given start to a given goal position in the game world.

Traversable Space: The space that can be traversed by characters in a digital game.

ENDNOTES

1 Throughout this chapter, we will refer to any moving entity as a *character*. Characters do not need to be humanoid, but can also be vehicles or any type of automated moving entities, depending on the specific gaming application.

2 Visit http://www.geometrygames.org/TorusGames/ for a collection of games such as *Chess* or *Tic Tac Toe* that run on a torus or a *Klein Bottle* topology (Klein, 1882). Accessed January 22, 2015.

3 Visit http://theory.stanford.edu/~amitp/GameProgramming/ for additional information on path planning with A*. Accessed January 22, 2015.

Proofing copy. Copyright IGI Global. May not be posted or distributed.