COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2012; **23**:59–69
Published online 9 February 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1424

SPECIAL ISSUE PAPER

Real-time density-based crowd simulation

Wouter G. van Toll, Atlas F. Cook IV and Roland Geraerts*

Department of Information and Computing Sciences, Utrecht University, 3584 CC Utrecht, The Netherlands

ABSTRACT

Virtual characters in games and simulations often need to plan visually convincing paths through a crowded environment. This paper describes how crowd density information can be used to guide a large number of characters through a crowded environment. Crowd density information helps characters avoid congested routes that could lead to traffic jams. It also encourages characters to use a wide variety of routes to reach their destination. Our technique measures the desirability of a route by combining distance information with crowd density information. We start by building a navigation mesh for the walkable regions in a polygonal two-dimensional (2-D) or multilayered three-dimensional (3-D) environment. The skeleton of this navigation mesh is the medial axis. Each walkable region in the navigation mesh maintains an up-to-date density value. This density value is equal to the area occupied by all the characters inside a given region divided by the total area of this region. These density values are mapped onto the medial axis to form a weighted graph. An A* search on this graph yields a backbone path for each character, and forces are used to guide the characters through the weighted environment. The characters periodically replan their routes as the density values are updated. Our experiments show that we can compute congestion-avoiding paths for tens of thousands of characters in real-time. Copyright © 2012 John Wiley & Sons, Ltd.

KEYWORDS

crowd simulation; crowd density; navigation mesh; path planning

***Correspondence**

Roland Geraerts, Department of Information and Computing Sciences, Utrecht University, 3584 CC Utrecht, The Netherlands.
E-mail: R.J.Geraerts@uu.nl

1. INTRODUCTION

Virtual characters often need to plan visually convincing paths through a crowded environment. Such paths should be easy to compute and should permit characters to avoid static obstacles as well as other moving characters. Although the *shortest* paths can be used to guide characters through an environment, traffic jams can occur when many characters traverse the same route. Please refer to Figures 1 and 2 for examples.

1.1. Goal and Contributions

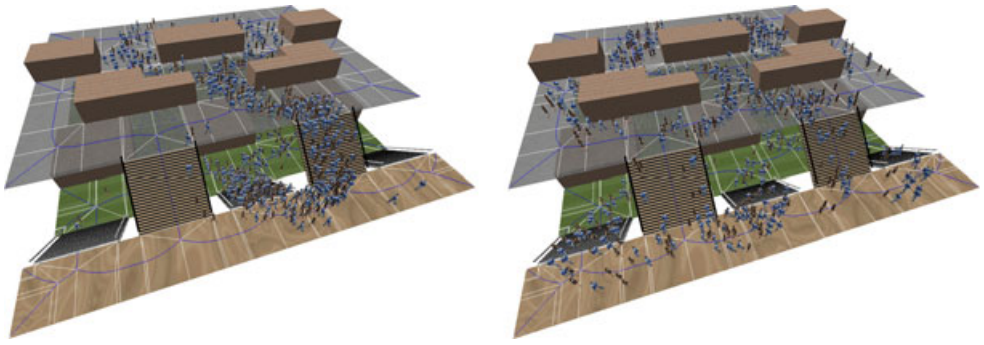
The goal of this paper is to use continuously updated density information to guide tens of thousands of characters through a crowded environment in real-time. The desirability of a route is measured by combining distance information with crowd density information.

Figure 2 shows the intuition behind our approach: density-aware path planning can guide characters around congested areas, thus, avoiding difficult local collision-avoidance problems. When applied to an entire crowd, as in Figure 1, the characters will spread among multiple routes on the basis of their individual choices.

Our algorithms work for any polygonal two-dimensional (2-D) or multilayered three-dimensional (3-D) environment. A *multilayered 3-D environment* is represented by a set of 2-D *layers* and a set of *connections*. Each layer is a collection of 2-D polygons that all lie in a single plane, and each connection provides a means of moving between layers [1]. Examples of multilayered 3-D environments are airports and multistory buildings. Please refer to Figure 1 for an example of a multilayered 3-D environment.

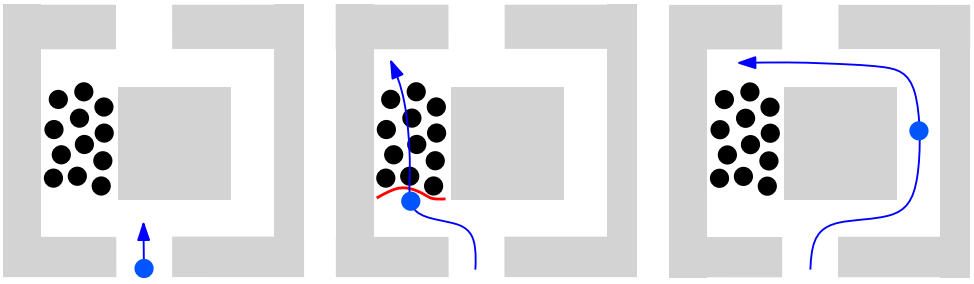
The real-world concept of *crowd density* is often expressed in characters per square meter [2]. Studies have shown that when crowd density is low, characters can move quickly through the environment. As density increases, real-world characters will move more slowly through the environment [3]. Our technique will model this behavior by reducing the maximum speed of characters on the basis of the crowd density information.

We maintain crowd density information in a *navigation mesh* that partitions the environment into a collection of walkable regions. Each walkable region maintains a *density value* that represents the crowd density in that walkable



(a) Without Density Information (b) With Density Information

Figure 1. A multilayered three-dimensional environment is shown, which has three layers and five staircase connections. (a) Without density information, most of the characters follow the same short path. This leads to a traffic jam. (b) When density information is considered, the characters will naturally spread out among the available routes.



(a) Scenario (b) Short Path (c) Density-Aware Path

Figure 2. (a) Scenario: A character (shown as a blue disk) wants to move to the top-left corner of an environment with light gray obstacles. The left half of the scene is occupied by many other characters (shown as black disks). (b) Short path: The shortest route, for the blue character, runs through the congested region. Local forces cannot steer the blue character around the black disks. (c) Density-aware path: If we use *density* information when determining a global path, uncongested routes become more attractive.

region. This density value is equal to the total area occupied by the characters inside the walkable region divided by the total area of the walkable region. We update the density values each time the characters move. These density values are mapped onto the medial axis to form a weighted graph, and an A* search [4] on this graph is used to guide each character through the crowded environment. Because the medial axis is sparse, it can be searched more efficiently than a grid.

Finally, periodic *replanning* ensures that characters avoid crowded routes whenever possible. We present a version of the A* algorithm that allows characters to replan their paths *partially*, which is more efficient than replanning the entire paths. The intuition behind this approach is that the density values of areas that are *far away* are not immediately important because they may have changed drastically by the time a character is close to these areas and wants to replan again.

1.2. Related Work on Crowd Simulation

Many techniques exist to help characters move realistically through virtual environments. Graph-based techniques such as probabilistic roadmaps [5], rapidly exploring random trees [6], and waypoint graphs [7] represent the environment by using a set of *one-dimensional* edges. By contrast, a *navigation mesh* partitions the environment into walkable regions that are *two-dimensional* [8–12]. These walkable regions permit characters to control their movements inside each 2-D region [13]. This flexibility also makes it much easier for characters to avoid other moving characters.

Many navigation meshes can only be used for 2-D problems. By contrast, the navigation graphs technique of Pettré *et al.* [11] uses a clever sampling-based approach to capture the topology of any 3-D *multilayered* environment. It can also handle level-of-detail optimizations for high-performance applications. Our own method [1] computes the medial axis of a multilayered environment, leading to a compact and *exact* navigation mesh.

A navigation mesh is typically used to compute a global route through the environment. Local collision-avoidance routines are then applied to avoid other moving characters that are encountered along the route [14,15]. One drawback to this two-level approach is that a character can get stuck when the global route is very congested

Comp. Anim. Virtual Worlds 2012; **23**:59–69 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

by other characters. Our new approach can prevent such congestions by guiding the global planning phase with local density information.

Techniques based on *potential fields* address the same problem by *combining* the global and local path planning phases. A potential field is a grid representation of the walkable space in which each cell stores the optimal walking direction toward a fixed goal. These optimal directions can be updated by using continuum-based techniques [16–18]. Whereas potential fields can successfully model large crowds in real-time, they use a grid to approximate the environment. Such grids are expensive to store and update. For us to ensure real-time performance, the crowd is often assumed to consist of homogeneous groups. Within a group, characters share a single potential field and cannot have individual goals. Hence, potential fields alone may not be suitable for simulations that require individuality among the characters.

Yersin *et al.* [19] present a hybrid approach that uses a navigation graph for global planning and grid-based methods for local avoidance. They define potential fields only for those parts of a navigation graph that lie in a high-interest region. This combined method is very scalable, but the potential fields are again based on homogeneous groups. By contrast, our approach permits each character to plan its own distinct global path on the basis of current crowd density information.

Hence, unlike these field-based approaches, we separate the global and local planning phases to support individuality among characters. This paper focuses on *global* planning on a navigation mesh; local collision-avoidance techniques can still be added as an extra level [14,15]. Our density-based planning algorithm can prevent the congestion problems of the previous two-level techniques.

1.3. Related Work on Crowd Density

Crowd density has been previously studied by other researchers. Weidmann [2] shows that a character's movements are influenced by environmental factors (e.g., weather conditions or the incline of a surface) and personal factors (e.g., age or gender). The study also reveals that the expected walking speed of a character decreases as the crowd density near the character increases. These real-world observations have influenced several recent simulation models [20]. Our model will reduce the maximum speed of a character as the crowd density near that character increases.

Karamouzas *et al.* [21] present a grid-based method for density-based crowd simulation. They mark each cell in a grid as 'dense' when a character enters it, and this density value decreases gradually over time. It is shown that path planning on this grid leads to a natural variety among characters. However, this grid is an approximation of the environment's geometry, and path planning is usually more expensive on a grid than in methods such as ours that describe the walkable space in an exact and compact fashion. Their technique is also not based on the real-world density concept which has been validated in many studies [2,20].

Pettré *et al.* [22] propose subdividing a crowd into separate flows through their navigation graphs according to density. They construct *navigation flow* queries to successfully dispatch many entities that move between shared locations. In our method, characters can have individual start and goal locations. We also investigate periodic replanning by entities as the density information changes over time.

Our density-based planning algorithm is a generalization of the *fastest path algorithm* in Höcker *et al.* [23]. Kneidl and Borrmann [24] have shown that the fastest path algorithm can lead to behavior that matches real crowds. However, Höcker *et al.* [23] use a collection of squares to approximate the local density information. A drawback of this method is that these squares can overlap and cause some parts of the walkable space to be represented more than once. This leads to a bias where some parts of the walkable space are implicitly considered to be more important than other parts. Furthermore, some parts of the walkable space may not be represented at all. By contrast, our method partitions any 2-D or multilayered 3-D polygonal environment into a set of nonoverlapping polygonal regions. Another improvement is that we address the issue of (partial) replanning during the simulation.

The rest of this paper is organized as follows. Section 2 describes the main algorithm that takes the density values into account. Section 3 discusses experiments that steer tens of thousands of characters simultaneously through a crowded environment in real-time.

2. DENSITY-BASED NAVIGATION

A navigation mesh partitions the environment into a set of two-dimensional walkable regions. These two-dimensional regions give characters the flexibility to move anywhere inside a walkable region while avoiding other moving characters.

Our navigation mesh is based on the *medial axis* of a polygonal environment and the *Explicit Corridor Map* (ECM) data structure [1,8]. The medial axis is the set of all points in an environment that have more than one distinct closest point on the boundary of the environment [25]. The medial axis is a compact structure that is well defined for all environments, including 'difficult' environments that are highly detailed and have very narrow corridors [1].

The ECM is a structure that uses the medial axis to partition the environment into a set of nonoverlapping regions such that each region contains exactly *one* edge of the medial axis. For us to convert the medial axis into an ECM, a linear number of extra line segments are added to connect the vertices of the medial axis to the closest obstacles. As illustrated in Figure 3, these extra line

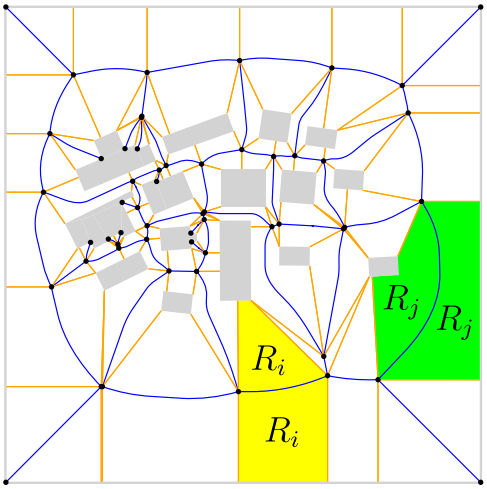


Figure 3. Our navigation mesh is an augmented medial axis. The medial axis of the environment is shown in blue. Orange line segments connect the vertices of the medial axis to the nearest obstacles. The arrangement of the orange edges partitions the environment into a set of walkable regions $\mathcal{E} = \{R_1, \ldots, R_m\}$. For example, we have highlighted two walkable regions R_i and R_j. Notice that each of these walkable regions contains exactly one edge of the medial axis.

segments partition the environment into a set of walkable regions $\mathcal{E} = \{R_1, \ldots, R_m\}$. Notice that each walkable region contains exactly one edge of the medial axis.

An advantage of the ECM data structure, with respect to approaches that triangulate the environment, is that the ECM efficiently represents all homotopic routes through the environment. This means that an A* search [4] will only examine one vertex at each possible point where the homotopy class of the route could change. By contrast, methods that triangulate the environment typically have more vertices, and this increases the running time of an A* search.

Let $R_i \in \mathcal{E}$ be a simple polygon that represents a walkable region. We say that the *density value* ρ_i of the walkable region R_i is the area of all characters currently inside R_i divided by the total area of R_i. Although our method permits each character to have a distinct size, we follow the suggestion of Weidmann [2] and model each character as a disk with a radius of 0.24 m.

We refer to the set $\mathcal{E} = \{R_1, \ldots, R_m\}$ of all walkable regions in a polygonal environment as a *density map* because any point in the environment can be mapped onto its containing region R_i and onto the associated density value ρ_i. When compared to grid-based decompositions, this paradigm has the advantage that it partitions the entire walkable space of the environment in a compact and exact manner.

The density value of each walkable region will be used to weigh the medial axis edge that touches this region. The resulting medial axis serves as a weighted graph $G = (V, E)$. The graph G has a set V of vertices and a set E of edges. G can be quickly searched to determine a global route for each character.

Each time the characters move, the density values for all of the walkable regions need to be updated. We do this by keeping track of each character's current walkable region. When a character leaves the current walkable region R_i, we subtract the area of that character from an area sum for R_i. Similarly, when a character enters a new walkable region R_j, we add the area of that character to an area sum for R_j.

Section 2.1 describes the medial axis that we use to plan the global density-avoiding routes. Section 2.2 introduces an efficient partial replanning approach that permits characters to periodically update their global routes as the density information changes. Section 2.3 summarizes our approach.

2.1. Planning Algorithm

Given the density values for all of the walkable regions, we can easily compute the density value for each of the medial axis edges. The density value for any medial axis edge $e \in E$ is simply the density value of the walkable region that contains e. It will be useful to let $\|e\|$ denote the arc length of e and to let $\rho(e)$ be the density value of e. The cost to traverse any single graph edge e equals

$$t_{\min}(e) + w \cdot t_{\text{delay}}(e)$$

where $t_{\min}(e)$ is the time required to traverse the edge e at maximum speed, $t_{\text{delay}}(e)$ is the expected extra traversal time caused by the density of e, and w is a nonnegative weight.

The weight w provides a natural means of interpolating between the shortest path and the least dense path in the graph. If $w = 0$, characters will look for the shortest path. As w increases, characters will have an increasing desire to avoid dense regions. If $w = 1$, characters will look for the fastest path as in Höcker *et al.* [23]. Note that it is possible to choose a unique value of w for each character.

The monotonic function $t_{\text{delay}}(e)$ should return zero when $\rho(e) = 0$, and it should return ∞ when $\rho(e) = 1$. A variety of functions are known to successfully model this concept [20]. We choose the following function:

$$t_{\text{delay}}(e) = \frac{\|e\| \cdot \rho(e)}{v_{\max}(1 - \rho(e))}$$

where $v_{\max}$ is the maximum speed of a character.

The above equations can be used to assign a weight to each edge of the medial axis to produce a weighted graph. By running the A* search algorithm [4] on this graph, the best path for any character can be quickly determined. The straight-line time to reach the goal is used as the A* heuristic function. This heuristic function estimates the time to reach the goal position and ignores the density information (by assuming that $w = 0$).

Comp. Anim. Virtual Worlds 2012; **23**:59–69 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

2.2. Partial Replanning

For a moving crowd, the density values of the walkable regions can change rapidly. This means that characters should regularly replan their global routes. Because it may be infeasible for characters to recompute their entire paths in large simulations, we describe an efficient and optimal technique for *partial* replanning.

The key idea is to speed up the replanning step by only permitting the character to 'see' nearby density information. Given a character at a position s, the *replanning distance* from this character to any point t on the medial axis is determined as follows. Let p be the closest point on the medial axis to s. We say that the replanning distance from s to t equals the Euclidean distance $\|s-p\|$ plus the weighted length of an optimal path through the medial axis from p to t.

Let D be some threshold value. All points on the medial axis that have a replanning distance of, at most, D are said to be *visible* to a character. Similarly, all points on the medial axis that have a replanning distance larger than D are said to be *invisible*.

During the A* search of the medial axis, a character sees the true density values for only the visible points on the medial axis. All invisible points on the medial axis are assumed to have zero density.

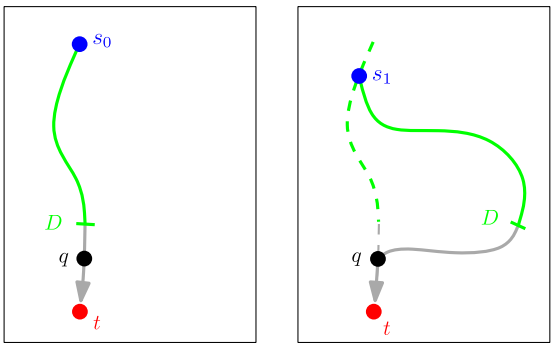
An advantage of this approach is that we can often halt the A* search early without losing optimality. As illustrated in Figure 4, consider an *invisible* point q on the medial axis that has a replanning distance larger than D. Once the A* search reaches the invisible point q, all future points that can be reached from q must also be invisible because they must have a replanning distance larger than D.

We call a point on the medial axis *mutually invisible* if and only if that point is invisible in both the original path and the replanned path. Let q be the first mutually invisible point that is encountered during replanning.

Theorem 1. *As soon as the A* search reaches a mutually invisible point q, we can halt the search and simply copy all points from q to the target point t on the original path onto the new replanned path. The resulting replanned path is optimal.*

Proof. Both the original path and the replanned path must contain the exact same subpath from q to t. This follows because this subpath is based on the same distance and density information in both cases. This means that the A* algorithm can stop searching as soon as it reaches q. □

An alternative to our optimal replanning algorithm is the D* Lite algorithm [26]. However, the D* Lite algorithm uses a state-space representation for each character which is too much overhead for a crowd. The traditional D* Lite algorithm also takes *all* updated density values into account. By contrast, our approach uses a threshold value D to provide a tradeoff between speed and accuracy.



(a) Initial Path (b) Replanned Path

Figure 4. (a) Initial path: A character initially computes a path from a point s_0 to a target point t. The portion of the path with the replanning distance, at most D, to the character is shown in green, and the remainder of the path is shown in gray. (b) Replanned path: When replanning this character's path at some future position s_1, the character plans a new route to t, and this new route eventually meets the original path at a mutually invisible point q. The subpath from q to t is always shared by both paths.

Our experiments in Section 3.4 will show that smaller values of D result in faster replanning operations because they take less density information into account. By contrast, larger values of D lead to slower replanning operations, but the replanned paths are more accurate because they take more density information into account. Note that choosing a value of $D = \infty$ will cause all of the density information in the environment to be considered during the optimal replanning step.

2.3. Approach

Our algorithm can be summarized as follows. Given a polygonal 2-D or multilayered 3-D environment, we compute the medial axis of this environment. Extra edges are added to connect the medial axis to the nearest obstacles as in [8]. The resulting augmented medial axis defines a navigation mesh that partitions the walkable environment into a set of walkable regions. Given a set of moving characters, density values are computed for each walkable region, and these density values are used to assign weights to the edges of the medial axis. The resulting medial axis is a weighted graph. Given the current position and the goal position of a character, a global path can be computed by determining the nearest points on the medial axis to the current position and goal position. An A* search is then used to compute a path through the weighted graph of the medial axis. At each step of the simulation, characters are pushed toward their goal by attractive forces [27]. The density values are updated each time a character enters a new walkable region. Periodic replanning operations permit characters to update their global routes in real-time as the density information changes.

3. EXPERIMENTS

We have implemented our density-based crowd simulator in C++ with Microsoft Visual Studio 2008. All of the experiments were performed on a personal computer with a 2.5 GHz Intel Xeon E5420 processor, an NVIDIA Quadro FX 1700 graphics card, and 4 GB of RAM. The machine uses Windows XP (64-bit, Service Pack 2).

Section 3.1 describes the environments and settings that were used in the experiments. Section 3.2 shows that density values can be efficiently maintained. Section 3.3 describes several experiments that influence crowd movements. In Section 3.4, we test the performance of our replanning algorithm by varying the amount of density information that is visible to each character. Section 3.5 describes how often replanning operations can occur without losing real-time behavior. Section 3.6 describes a performance boost that is achieved by using multiple central processing unit (CPU) cores. All other experiments use only a single CPU core.

Although our method permits each character to have a distinct size, we follow the suggestion of Weidmann [2] and model each character as a disk with a radius of 0.24 m and an area of approximately $0.18\,m^2$. Characters have a walking speed of 1.4 m/s through walkable regions that have a density value of 0. As the density value of a walkable region increases from 0 to 1, the walking speed of a character decreases linearly to 0 m/s. The simulation step time was set to 10 frames per second as in [27].

3.1. Environments and Settings

Our experiments were performed in one multilayered 3-D environment and in three 2-D environments. We refer to the multilayered 3-D environment in Figures 1 and 5 as the *Layers* environment.

The *Layers* environment contains three layers ('floors') that are connected by five staircases. The environment also contains a number of nonconvex obstacles that permit characters to choose from numerous routes when they traverse the environment. As shown in Figure 1(a), nearly all of the characters follow the same short path when density information is not considered. This leads to a traffic jam. By contrast, Figure 1(b) shows that characters will naturally spread out among all of the available routes when density information is considered. Each character in the simulation has also been assigned an offset between −1.0 and 1.0. This offset changes where a character prefers to walk along a homotopic route. An offset of 0 ensures that a character prefers to walk along the edges of the medial axis. An offset of −0.5 makes a character prefer to walk halfway between the medial axis and the nearest obstacle to the left of the current medial axis edge. An offset of 1.0 causes a character to walk as close as possible to the nearest obstacle to the right of the current medial axis edge.

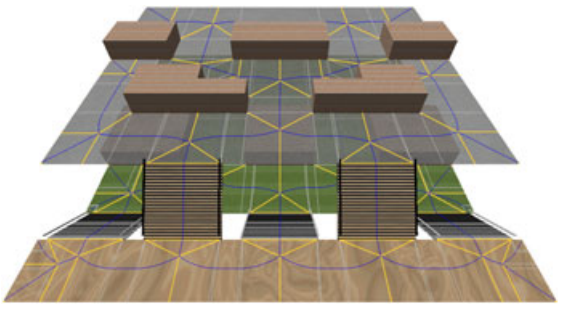


Figure 5. The multilayered 3-D environment that is used in our experiments. There are three layers and five staircase connections between layers. Obstacles are shown as raised blocks, the medial axis is displayed in blue, and the boundaries of the walkable regions are shown in orange.

Figure 6 illustrates our three 2-D environments. The *Blocks* environment is a small example that has a number of potential routes which characters may choose. This environment has also appeared in [21]. The *Zelda* environment is a medium-sized example of a village that appeared in the computer game *The Legend of Zelda: A Link To the Past*. The *City* environment is a large virtual city that contains many polygonal obstacles and routes. More details for these scenes can be found in Table I. The medial axes for these environments were all built with the Graphics Processing Unit (GPU) Explicit Corridor Map ECM algorithm of Geraerts [8].

3.2. Overhead of Density Values

Our first experiment measured the overhead to maintain the density values for the walkable regions and the medial axis. We inserted a large number of characters with random start and goal positions into each of our environments and simulated their movement. Characters were removed from the environment as soon as they reached their goal position. Table II shows the time needed to perform each frame of the simulation for the entire crowd.

In the *Layers* environment, simulating 10 000 simultaneously moving characters took 23 millisecond per step on average when density updates were switched off. With the densities enabled, the average step time for 10 000 simultaneously moving characters was 24 millisecond. Consequently, the total CPU load with densities enabled was 24% during the simulation. In the *Blocks* environment, the step time for 10 000 characters was 15 millisecond without density updates, and 16 millisecond with density updates. In the *Zelda* environment, the step time for 10 000 characters was 16 millisecond without density updates *versus* 17 millisecond with density updates. The step time for 20 000 characters in the *City* environment was 32 millisecond without densities, and 34 millisecond with densities. Thus, updating the crowd's density information only marginally affects the running time, and large crowds can still be steered in real-time.

Comp. Anim. Virtual Worlds 2012; **23**:59–69 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

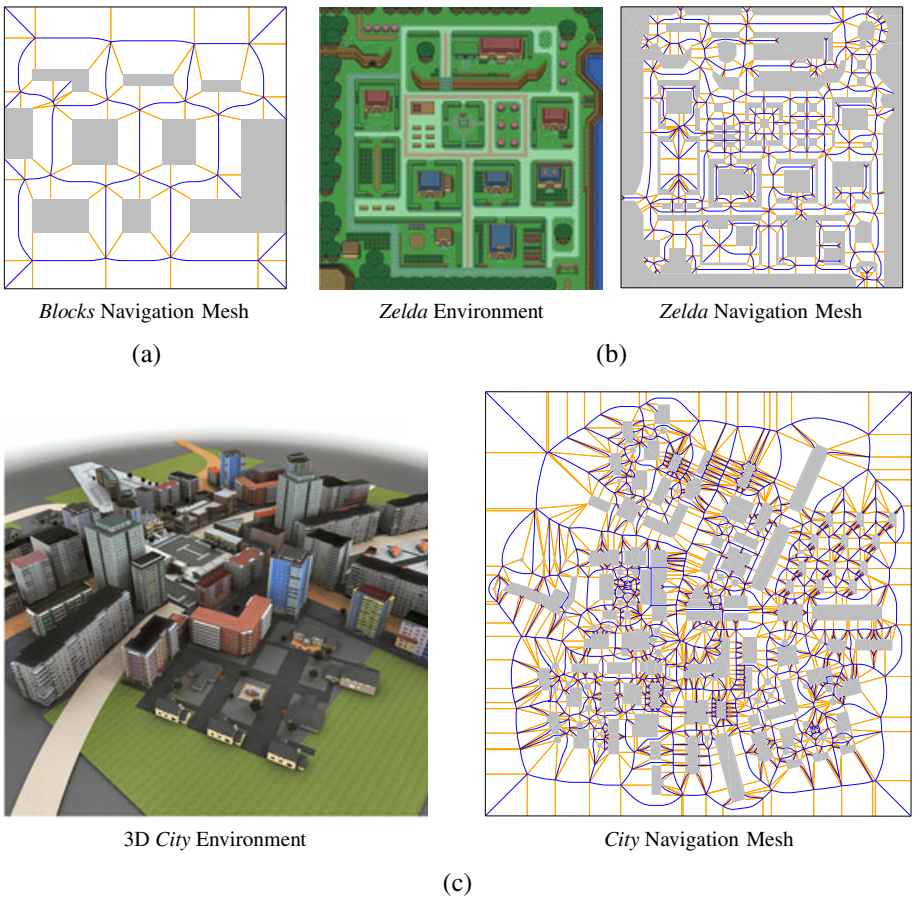


Figure 6. Three 2-D environments were used in our experiments. Obstacles are shown in gray, the medial axis is shown in blue, and the extra edges that connect the medial axis to the nearest obstacles are shown in orange. (a) The *Blocks* environment contains many alternative routes that can be used by characters to get from the bottom of the environment to the top of the environment. (b) The *Zelda* environment represents a village in a video game. (c) The *City* environment represents a large city.

Table I. The navigation meshes for the *Layers*, *Blocks*, *Zelda*, and *City* environments were constructed with the GPU. The navigation meshes of *Layers*, *Blocks*, and *Zelda* were computed using a resolution of 1000 × 1000 pixels. For *City*, we used a resolution of 4000 × 4000 pixels. This is the reason for the longer construction time.

Environment			Navigation Mesh	
Name	Size (meters)	Vertices	Vertices	Construction time (milliseconds)
Layers	100 × 100	332	248	38
Blocks	100 × 100	52	132	25
Zelda	100 × 100	560	1243	49
City	500 × 500	2638	6273	403

3.3. Crowd Variety

Our second experiment used the *Blocks* environment to test the effect of planning paths first without density information, then with density information but without any replanning, and finally with both density information and periodic replanning. In every step of the simulation, we added two characters at random positions at the bottom of the *Blocks* scene and with random goal positions at the top of the scene. We ran the crowd simulation for 5000 steps (i.e., 500 seconds). Without considering any density information (i.e., setting $w = 0$), all of the characters moved along the shortest path in the graph. Hence, almost all of the paths ran through the environment's middle section.

Table II. The overhead of maintaining the density values is small.

Environment	Characters	Step time without densities (milliseconds)	Step time with densities (milliseconds)
Layers	10 000	23	24
Blocks	10 000	15	16
Zelda	10 000	16	17
City	20 000	32	34

Table III. The average speed and path lengths of characters.

Experiment	Average path length (meters)	Average speed (meters per second)
No density values ($w = 0$)	106.87	0.67
Density values ($w = 1$)	115.90	0.91
Density values and replanning ($w = 1$)	114.85	0.96
Density values and replanning ($w = 5$)	132.96	1.01

This led to traffic jams that slowed down the crowd. As illustrated in Table III, characters traversed 106.87 m, on average, to reach their goal positions, but they had a low average speed of 0.67 m/s. Figure 7(a) shows a snapshot of the simulation.

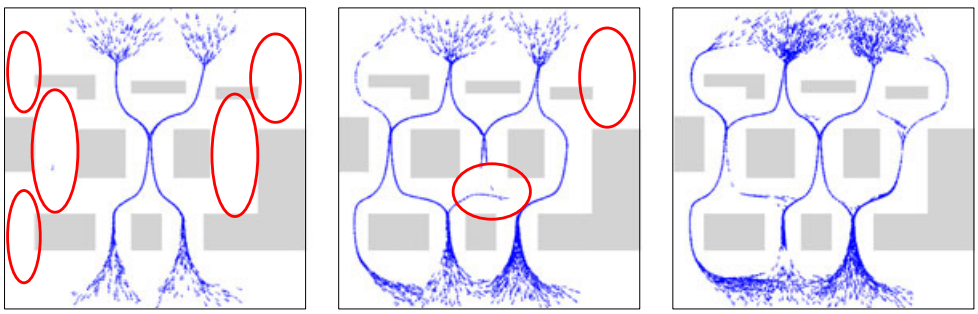
With density information (i.e., $w = 1$) but without any replanning, characters in the *Blocks* environment initially took the shortest path. This led to traffic jams in the central sections. Newly created characters detected this congestion and chose other paths. These paths crowded the other routes, whereas the central section was gradually emptied. This periodic behavior was repeated several times during the simulation. A snapshot is shown in Figure 7(b). The average walking speed of each character was much higher (0.91 m/s), whereas the traversed paths were not much longer (115.90 m on average).

We also ran the experiment with density information (i.e., $w = 1$) while letting characters replan their paths every 100 steps using the current density information. This spread the characters among the available routes. As shown in Figure 7(c), the crowd looked visually convincing because it did not leave any gaps in the available routes. The average speed of the characters was high (0.96 m/s), whereas the distance traversed by each character was still quite short (114.85 m).

Choosing a weight value of $w = 5$ along with replanning every 100 steps caused characters to switch between routes frequently and to take large detours to avoid congested regions. The average walking speed of the characters increased to 1.01 m/s, and their paths became longer (132.96 m) because of increased indecisiveness.

3.4. Replanning Efficiency

Our third experiment investigates how the path-planning time can be reduced by changing the maximum distance D at which updated density information is visible to the characters. We simultaneously added 5000 characters to the *City* environment with random start and goal positions



(a) Without Density Information (b) With Density Information (c) With Density + Replanning

Figure 7. Screenshots of a crowd simulation with a variety of settings. Obstacles are shown in gray, and characters are shown in blue. Routes with little utilization are highlighted in red. (a) When density information is not considered, there are many routes that are not very much utilized. (b) If density information is taken into account, but characters never replan their paths, then some routes are still underutilized. (c) When density information is taken into account and replanning is performed, we obtain an emergent crowd flow that efficiently spreads the characters among the available routes.

DOI: 10.1002/cav

in such a way that the Euclidean distance between each characters' start and goal positions was at least 100 m. We set $w = 5$ for all experiments so that convincing detours would be explored and added all characters to the environment at the same time. We let each character replan its path every 10 seconds, and we ran the simulation for over 40 seconds to ensure that each character replanned up to four times. Figure 8 illustrates the time to update an existing path on the basis of updated density information.

With $D = \infty$, each character can use all of the updated density values. Optimal routes containing 20 vertices took between 0.2 and 1 millisecond to compute, whereas routes with 40 vertices required between 0.5 and 2.5 milliseconds. Routes with 60 vertices took between 1.5 and 3 milliseconds to update. The average time to replan each path was 2 milliseconds.

With $D = 350$ m, each character can see all density updates within 350 m of the current position. The average time to perform each replanning operation was reduced to 1 millisecond.

With $D = 0.1$ m, each character, essentially, cannot see any updated density values. The average time to perform each replanning operation was 0.3 millisecond.

Notice that larger values of D take longer to compute because they take more density information into account. Likewise, smaller values of D take less time to compute because they take less density information into account.

We performed a similar experiment in the *Zelda* environment. With $D = \infty$, replanning operations took between 0.5 and 0.8 millisecond to calculate. With $D = 0.1$ m, replanning operations took only 0.3 millisecond. This means that replanning operations can be performed in real-time.

3.5. Real-time Replanning

If we set each character to replan its path periodically, then we can compute the highest replanning frequency that preserves real-time performance. Let t_{step} be the number of milliseconds required for each step of the simulation without replanning. Given that there are 10 simulation steps per second, we have $t_{\text{rem}} = 100 - t_{\text{step}}$ milliseconds left in each frame for other tasks. Let us assume that t_{rem} can be spent entirely on replanning so that we can ignore rendering, local collision avoidance, and other tasks.[†]

Recall that in Table II, without any replanning, 20 000 characters can be steered through the City environment in $t_{\text{step}} = 32ms$ per frame. Hence, $t_{\text{rem}} = 66$ milliseconds. If $D = \infty$, then the average replanning time is 2 milliseconds per character. This means that 33 characters can replan their paths each frame with a 100% CPU load. Consequently, all 20 000 characters can replan their paths every 60.6 seconds without losing real-time performance.

[†]In reality, these extra steps are needed to obtain a complete simulation. However, for a simple theoretical analysis, it is useful to ignore these factors.

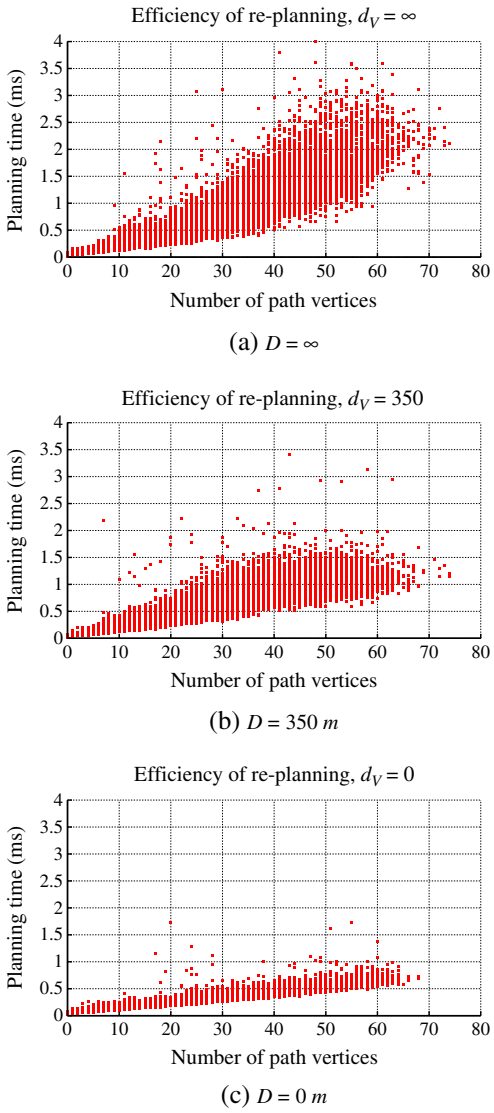


Figure 8. Replanning times for 5000 characters in the *City* environment with (a) $D = \infty$, (b) $D = 350$, and (c) $D = 0.1$ m. Characters replan their path every 10 seconds. Each point in this figure corresponds to one replanning action. The horizontal axis shows the number of vertices in the newly computed path. The vertical axis shows the running time of our path planner in milliseconds.

By similar reasoning, all 20 000 characters can replan their paths every 30.3 seconds when $D = 350$ m.

3.6. Multithreaded Speedup

We have built the simulator by using OpenMP technology, so that it can plan paths for multiple characters at the same time. Using only one CPU core, our method can simultaneously steer 50 000 characters through the *Blocks* environment in 90 milliseconds per frame. However, with four CPU cores, we can simultaneously steer 50 000 characters

through the *Blocks* environment in 30 milliseconds per frame. This means that using four processors can make the simulation 3 times as fast. These runtimes suggest that in the very near future our method should be able to steer hundreds of thousands of characters at interactive rates.

4. CONCLUSION

Although it is common to steer characters along short paths to their destinations, high congestion can lead to traffic jams that seem unnatural when many routes are underutilized in an environment. To improve the realism of our simulations, we use crowd density information to guide a large number of characters along a wide variety of routes. We do this by building a navigation mesh and weighing the desirability of routes on the basis of the crowd density along the path. Our technique can guide tens of thousands of characters through a polygonal 2-D or multilayered 3-D environment in real-time. The attached movie highlights the effectiveness of these techniques in both 2-D and 2.5-D environments (see Supporting Information).

One limitation of our technique is that an extremely small walkable region could have a very high density value if a large character suddenly enters that region. This large density value could adversely affect the desirability of an entire route. If this behavior is a problem, it might be interesting to scale the density values on the basis of the length of the affected medial axis edge.

As future work, we are interested in looking beyond density information to the speed and direction of a crowd. We expect that this *flow* information will play an important role in determining a character's route in scenarios such as crowd evacuation [28]. It would also be interesting to replan paths on the basis of specific density-change events rather than simply replanning paths periodically. Finally, we would like to distinguish between different terrain types and to account for height and slope information.

ACKNOWLEDGEMENTS

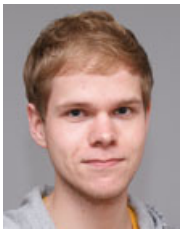
This research has been supported by the GATE project (http://gate.gameresearch.nl), INCONTROL Simulation Solutions, and the Netherlands Organization for Scientific Research (NWO). The authors are part of the Institute of Information and Computing Sciences, Utrecht University, 3584 CC Utrecht, the Netherlands.

REFERENCES

1. van Toll WG, Cook IV AF, Geraerts R. Navigation meshes for realistic multilayered environments, In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2011; 3526–3532.
2. Weidmann U. *Transporttechnik der Fussgänger - Transporttechnische Eigenschaften des Fussgängerverkehrs*, Literature Research 90, ETH Zürich, Institut für Verkehrsplanung, Transporttechnik, Strassen- und Eisenbahnbau, 1993. In German.
3. Daamen W. Modelling passenger flows in public transport facilities, *PhD thesis*, Delft University of Technology, 2004. Thesis number: T2004/6, TRAIL series.
4. Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 1968; **4**(2): 100–107.
5. Kavraki LE, Švestka P, Latombe JC, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 1996; **12**: 566–580.
6. Kuffner JJ, LaValle SM. RRT-connect: an efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation* 2000: 995–1001.
7. Rabin S. *AI Game Programming Wisdom 2*. Charles River Media Inc., Hingham, 2004.
8. Geraerts R. Planning short paths with clearance using explicit corridors, In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010; 1997–2004.
9. Kallmann M. Path planning in triangulations, In *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005; 49–54.
10. Mononen M. Recast navigation, 2011. Google Project: http://code.google.com/p/recastnavigation.
11. Pettré J, Laumond J-P, Thalmann D. A navigation graph for real-time crowd animation on multilayered and uneven terrain, In *Proceedings of the First International Workshop on Crowd Simulation*, 2005.
12. Wein R, van den Berg JP, Halperin D. The Visibility–Voronoi complex and its applications. *Computational Geometry: Theory and Applications* 2007; **36**(1): 66–78.
13. Geraerts R, Overmars MH. Enhancing corridor maps for real-time path planning in virtual environments. *Computer Animation and Social Agents* 2008: 64–71.
14. van den Berg JP, Lin M, Manocha D. Reciprocal velocity obstacles for realtime multi-agent navigation, In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008; 1928–1935.
15. Karamouzas I, Bakker J, Overmars MH. Density constraints for crowd simulation, In *Proceedings of the ICE Games Innovations Conference*, 2009; 160–168.
16. Narain R, Golas A, Curtis S, Lin MC. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics* 2009; **28**: 1–8.
17. Patil S, van den Berg JP, Curtis S, Lin MC, Manocha D. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics* 2010; **17**: 244–254.

18. Treuille A, Cooper S, Popović Z. Continuum crowds. *ACM transactions on graphics* 2006; **25**: 1160–1168.
19. Yersin B, Maïm J, Morini F, Thalmann D. Real-time crowd motion planning: scalable avoidance and group behavior. *The Visual Computer* 2008; **24**: 859–870.
20. Daamen W, Hoogendoorn SP. Level difference impacts in passenger route choice modelling, In *Proceedings of the 8th TRAIL conference: A world of transport, infrastructure and logistics*, 2004; 103–127.
21. Karamouzas I, Geraerts R, Overmars MH. Indicative routes for path planning and crowd simulation, In *Proceedings of the 4th International Conference on Foundations of Digital Games*, 2009; 113–120.
22. Pettré J, Grillon H, Thalmann D. Crowds of moving objects: navigation planning and simulation. *IEEE International Conference on Robotics and Automation* 2007: 3062–3067.
23. Höcker M, Berkhahn V, Kneidl A, Borrmann A, Klein W. Graph-based approaches for simulating pedestrian dynamics in building models, In *eWork and eBusiness in Architecture, Engineering and Construction*, 2010; 389–394.
24. Kneidl A, Borrmann A. How do pedestrians find their way? Results of an experimental study with students compared to simulation results, In *Emergency Evacuation of people from Buildings*, 2011.
25. Preparata F. The medial axis of a simple polygon. In *Mathematical Foundations of Computer Science*, Vol. 53. Springer, Berlin / Heidelberg, 1977; 443–450.
26. Koenig S, Likhachev M. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics* 2005; **21**(3).
27. Karamouzas I, Heil P, van Beek P, Overmars MH. A predictive collision avoidance model for pedestrian simulation, In *Proceedings of the 2nd International Workshop on Motion in Games*, 2009; 41–52.
28. Zheng X, Zhong T, Liu M. Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment* 2009; **44**(3): 437–445.

AUTHORS' BIOGRAPHIES



Wouter G. van Toll received his BSc in Computer Science in 2009 and his MSc in Game and Media Technology in 2011, both from Utrecht University in the Netherlands. Wouter is interested in path planning research, visual arts, game design, and game development. In his final year as a master's student, he extended the Explicit Corridor Map data structure to handle dynamic obstacles, multilayered environments, and density-based crowds. He implemented the last two concepts during an internship at INCONTROL Simulation Solutions.

Atlas F. Cook IV is a postdoctoral researcher at the Games and Virtual Worlds group in the Department of Information and Computing Sciences at Utrecht University in the Netherlands. He received his PhD in Computational Geometry in 2009 from the University of Texas at San Antonio. He loves path planning research, gaming, dancing, and smiling.

Roland Geraerts is an assistant professor at the Games and Virtual Worlds group in the Department of Information and Computing Sciences at Utrecht University in the Netherlands. There, he obtained his PhD on sampling-based motion planning techniques. In addition, he studied quality aspects of paths and roadmaps. His current research focuses on path planning and crowd simulation in games and virtual environments. Furthermore, he teaches several courses related to games and crowd simulation. Roland has organized the *Creative Game Challenge* and is one of the cofounders of the annual *Motion in Games* conference.