Realistic Crowd Simulation with Density-Based Path Planning

Wouter G. van Toll
Utrecht University
Department of Information and Computing Sciences
W.G.vanToll@uu.nl

Atlas F. Cook IV
Utrecht University
Department of Information and Computing Sciences
A.F.CookIV@uu.nl

Roland Geraerts
Utrecht University
Department of Information and Computing Sciences
R.J.Geraerts@uu.nl

Abstract

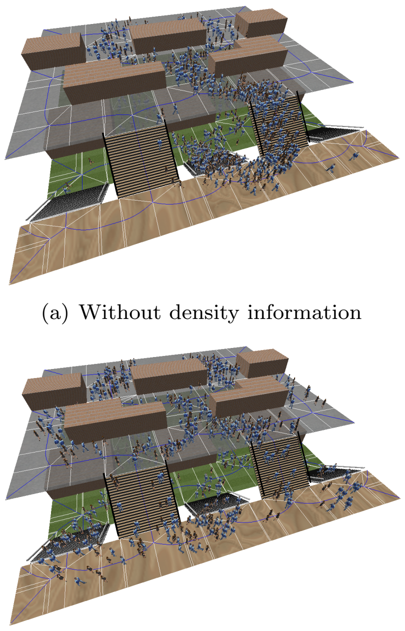
Virtual characters in games and simulations often need to plan visually convincing paths through a crowded environment. This paper describes how crowd density information can be used to guide a large number of characters through a crowded environment. Crowd density information helps characters avoid 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 2D or multi-layered 3D 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, given by the fraction of the area that is being occupied by characters. 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.

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 *shortest* paths can be used to guide characters through an environment, traffic jams can occur when many characters traverse the same route.

This paper has been submitted for presentation at ICT.OPEN 2012. This is the national Dutch ICT conference and serves, for this paper, the role of training the presentation skills rather than to push scientific limits. ICT.OPEN does not claim copyright. For this reason, ICT.OPEN 2012 encourages authors to submit mostly papers that have been sent to or have recently been presented at international conferences.


(a) Without density information

(b) With density information

Figure 1: A multi-layered 3D environment. Characters move from the bottom floor to the top floor. (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.

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 1 shows the effect of our method. When all characters simply look for the shortest route, they cause congestions which cannot be easily solved with local collision avoidance. Our density-based planning algorithm lets characters prefer areas that are less crowded, thus spreading them among the available routes. This behaviour is *emergent*: it is caused by the individual choices of the characters.

Our method works for any polygonal 2D or multi-layered 3D environment. The latter is a set of connected two-dimensional layers, e.g. a collection of floors and staircases [23]. Figure 1 shows an example.

The real-world concept of *crowd density* is often expressed in characters per square meter [25]. Field studies have shown that as the crowd density of a region increases, characters will move more slowly through that region [2]. Our technique will model this behavior by reducing the expected walking speed of characters based on the crowd density.

We maintain crowd density information in a *navigation mesh* that partitions the environment into walkable regions. Our mesh is based on the *medial axis*: the set of all points in the walkable space that have more than one closest point on the boundary of obstacles [21].

During the simulation, each region in the mesh stores a local crowd density value that is updated as characters enter or exit the region. We refer to the set of regions and their densities as the *density map*. These densities are mapped onto the medial axis to form a weighted graph, and we use A* search [6] on this graph 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 can respond to congestions as they appear or disappear. We present a version of the A* algorithm that lets characters replan their paths *partially*, which is more efficient than replanning the entire paths. Intuitively, density values of areas that are *far away* are not immediately important, because they may have changed drastically by the time a character is closer to these areas and wants to replan again.

We improve upon related work [7] by presenting a more effective density representation, a generalized planning algorithm, and an efficient replanning method.

1.2 Related Work on Crowd Simulation

Many techniques exist to steer characters through virtual environments. Graph-based techniques such as probabilistic roadmaps [12], rapidly-exploring random trees [15], and waypoint graphs [22] represent the environment using a set of *one-dimensional* edges. By contrast, a *navigation mesh* partitions the environment into *two-dimensional* walkable regions [4], [8], [16], [20], [26]. A character can first plan a global route through these regions, and then locally control its movement within each region [5]. This flexibility also allows collision avoidance between characters.

Most navigation meshes can only be used for two-dimensional problems. By contrast, the Navigation Graphs technique of Pettré et al. [20] uses a sampling-based approach to capture the topology of 3D *multi-layered* environments. Our own method [23] computes the *medial axis* of a multi-layered environment, leading to a compact, exact, and flexible navigation mesh.

A navigation mesh typically returns a global route through the environment. Local collision-avoidance routines should then ensure that characters avoid each other along the way [1], [11]. One drawback to this two-level approach is that a character can get stuck when the global route is congested 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* combine the global and local planning phases. A potential field is a grid representation of the walkable space in which each cell stores the optimal walking direction towards a fixed goal [17], [18], [24]. While these grids can model large crowds, they are expensive to store and update. To ensure real-time performance, the crowd is often assumed to consist of homogeneous groups. Within a group, characters share a potential field and cannot have individual goals. Hence, potential fields alone do not permit individual planning properties for each character.

Yersin et al. [27] use a navigation graph for global planning, and grid-based methods for local avoidance in only the high-interest regions of the environment. This hybrid method is very scalable, but the grids are again based on homogeneous groups.

Unlike these field-based approaches, we separate the global and local planning phases, such that each character can use its individual properties to plan its *own* path based on the current crowd density information. We focus on *global* planning on a navigation mesh; local collision-avoidance techniques can still be added as an extra level [1], [11]. Our density-based planning algorithm can prevent the congestion problems of previous two-level techniques.

1.3 Related Work on Crowd Density

In a practical study, Weidmann [25] shows that a person's movements are influenced by environmental factors (e.g. the incline of a surface) and personal factors (e.g. age). He also shows that the expected walking speed of a person decreases as the *crowd density* around that person increases. This observed relation has influenced several simulation models [3]. Our density-based planning algorithm is based on the same principle.

Karamouzas et al. [9] use a grid for density-based crowd simulation. They mark a grid cell as 'dense' when a character enters it, and this density value decreases gradually over time. Path planning on this grid leads to natural variety among characters. However, the grid is an approximation of the environment's geometry, and path planning is usually more expensive on a grid than on a sparse graph such as our navigation mesh [23]. Their technique is also not based on the real-world density concept validated in many studies [3], [25].

Pettré et al. [19] propose subdividing a crowd into separate flows according to density. Their *Navigation Flow* queries can dispatch entities that move between *shared* locations. In our method, characters can have individual start and goal locations.

Our density-based planning algorithm is a generalization of the *Fastest-Path Algorithm* by Höcker et al. [7]. Kneidl and Borrmann [13] have shown that the Fastest-Path Algorithm can lead to behavior that matches real crowds. However, Höcker et al. [7] use a collection of squares to approximate the local density information. These squares can overlap and can cause some parts of the walkable space to be represented more than once, which implicitly makes them more important. Furthermore, some parts of the walkable space may not be represented at all. By contrast, our method partitions any 2D or multi-layered 3D polygonal environment into a set of non-overlapping polygonal regions. Another improvement is that we allow replanning during the simulation as the density information changes over time.

2 Density-Based Navigation

In this section, we describe our new algorithm for density-based crowd simulation. Section 2.1 introduces our *density map* data structure. Section 2.2 describes the algorithm that we use to plan global density-avoiding routes. Section 2.3 introduces an efficient replanning approach that permits characters to periodically update their global routes as the densities change.

2.1 Density Map

We present the *density map* as a representation of the local crowd densities in an environment. Our density map subdivides the walkable space into polygonal, non-overlapping regions, each of which keeps track of the current local density value. Unlike grid-based decompositions, our method partitions the entire walkable space in a compact and exact manner.

The basis of our algorithm is the *Explicit Corridor Map* (ECM) navigation mesh [4], [23]. The ECM can efficiently answer global path planning queries for characters of all sizes, and it provides flexibility for local behavior. It is based on the *medial axis*, a compact structure that is well-defined for all locally planar environments, including those with non-convex obstacles and multiple layers [23].

The ECM annotates the medial axis with closest-obstacle information for a linear number of positions. By connecting these positions to their closest obstacle points, we partition the walkable space into a set of non-overlapping regions $\mathcal{E}$ =$\{R_1, ..., R_m\}$. Each region R_i contains exactly *one* edge e_i of the medial axis. Figure 2 shows an example.

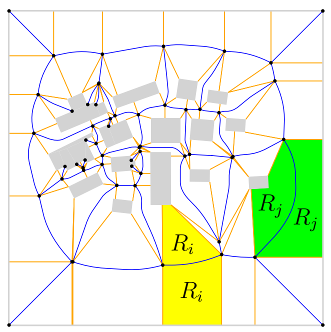


Figure 2: The ECM of an environment, with obstacles shown in gray. The medial axis is shown in blue. Orange line segments connect the vertices of the medial axis to their nearest obstacles. This partitions the environment into a set of walkable regions. We have highlighted two regions as an example. Each region contains one medial axis edge.

We define the *density value* $\rho_i \in [0, 1]$ of each $R_i \in \mathcal{E}$ as the area of all characters currently inside R_i divided by the total area of R_i. Unlike the more common definition in 'persons per square meter', our definition permits each character to have a distinct size.

We can now refer to the set $\mathcal{E}$ =$\{R_1, ..., R_m\}$ as the *density map*, because it maps any point in the environment onto its containing region R_i and onto the associated density value ρ_i. This density value will be used to weight the corresponding medial axis edge e_j.

Each time the characters move, the density values for all of the walkable regions need to be updated. We keep track of each character's current walkable region. When a character leaves a region R_i, we subtract the area of that character from the occupied area of R_i. When a character enters a new region R_j, we add the area of that character to the occupied area of R_j.

2.2 Planning Algorithm

Our planning algorithm is based on *time*: it looks for a path that can be quickly traversed, assuming that higher local densities lead to a longer traversal time.

Let $v(\rho)$ be the expected speed of a character at density $\rho \in [0, 1]$. According to field studies [25], $v(\rho)$ should return the maximum speed of the character when $\rho = 0$, and it should decrease to zero as ρ increases. Various functions have been used in practice [3]. We let $v(\rho)$ decrease linearly, down to $v(1) = 0$.

Next, let $||e_i||$ denote the arc length of an edge e_i. We define the cost $c(e_i)$ for traversing an edge e_i as follows:

$$c(e_i) = t_{\text{min}}(e_i) + w \cdot (t_{\text{real}}(e_i) - t_{\text{min}}(e_i)),$$

where $t_{\text{min}}(e_i) = ||e_i||/v(0)$ is the time required to traverse e_i at maximum speed, $t_{\text{real}}(e_i) = ||e_i||/v(\rho_i)$ is the expected traversal time due to the density of e_i, and w is a non-negative weight.

With the weight w, we can naturally interpolate 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. [7]. Note that each character can have its own value of w.

By running an A* search [6] on the medial axis with these edge costs, the optimal path for any character can be quickly determined. For the A* heuristic function, we use the straight-line time to the goal, assuming that the density is zero. This function never overestimates the actual time to the goal, i.e. it is admissible.

2.3 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. Since it may be infeasible for characters to recompute their entire paths in large simulations, we describe an efficient and optimal technique for *partial* replanning.

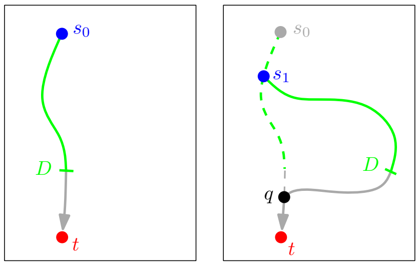
The key idea is to speed up the replanning step by only permitting the character to 'see' nearby density information. Given a character at a position s, the *replanning distance* $d_r(s,t)$ from this character to any point t on the medial axis is determined as follows. Let n be the closest point on the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

Let D be a threshold value. All points p on the medial axis for which $d_r(s,p) \leq D$ are said to be *visible* to the character. All other points on the medial axis are *invisible*. During the A* search on the medial axis, a character sees the density for only the visible parts. For the other parts, we assume that $\rho = 0$. Note that all of the density information will be considered if $D = \infty$.

An advantage of this approach is that we can often make *replanning* more efficient without losing optimality. Figure 3 shows a character that plans a path from a point s_0 to t. The path is visible up to the replanning distance D, and the remainder is invisible. Later, the character replans the path to t, from its current position s_1. Because the character has progressed along its path, the set of visible points has changed.

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. During replanning, the A* algorithm may reach such a mutually invisible point, as in Figure 3b. Let $\pi(a,b)$ denote a path between two points a and b. The following theorem holds:

Theorem 1 *As soon as A* reaches the first mutually invisible point q, we can halt the search and simply copy $\pi(q,t)$ into the new replanned path. The resulting path $\pi(s_1,t)$ is optimal.*



(a) Initial path (b) Replanned path

Figure 3: (a) A character initially computes a path from a point s_0 to a target point t. The visible and invisible parts of the path are shown in green and gray, respectively. (b) When the character replans its path from a position s_1, the new path eventually meets the original path at a mutually invisible point q. The sub-path from q to t does not change.

Proof. First, note that $\pi(s_1,q)$ is optimal, because A* returns optimal paths when the heuristic is admissible.

In the old situation, $\pi(q,t)$ was found without using densities. Hence, it is the shortest sub-path, computed with only static information. In the new situation, q is still invisible, so the character can still only use static information to find a new path $\pi'(q,t)$. Thus, $\pi'(q,t) = \pi(q,t)$, and we do not need to recompute it. □

Section 3.4 will show that the threshold D provides a tradeoff between speed and accuracy. An alternative approach to replanning is the D* Lite algorithm [14]. However, D* Lite uses too much memory overhead per character to be applicable for large crowds.

3 Experiments

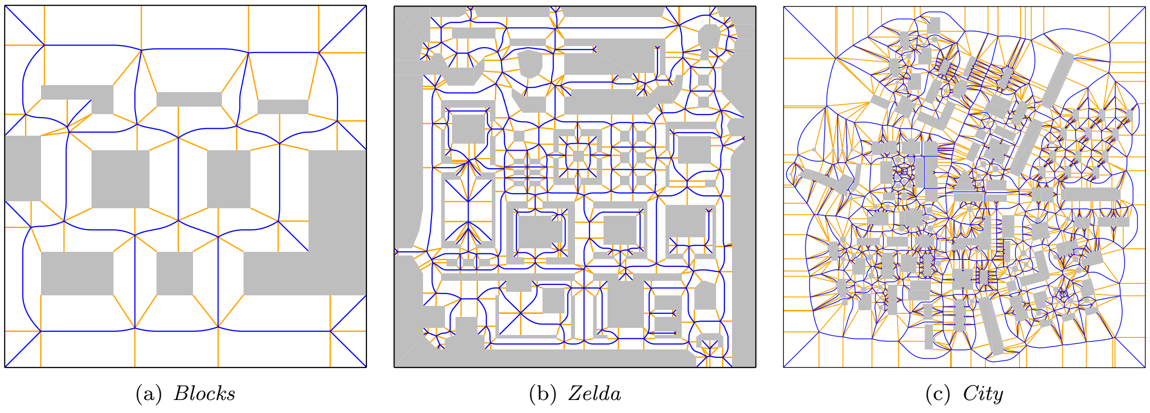
We have implemented our density-based crowd simulator in C++ with Microsoft Visual Studio 2008. All of the experiments were performed on a PC with an 2.5GHz Intel Xeon E5420 processor, an NVIDIA Quadro FX 1700 graphics card, and 4 GB of RAM. The machine uses Windows XP (64-bit, Service Pack 2). All experiments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

3.1 Environments and Settings

Our experiments were performed in one multi-layered 3D environment and in three 2D environments.

We refer to the environment in Figure 1 as the *Layers* environment. It consists of three floors connected by five staircases. It also contains a number of non-convex obstacles that let characters choose between numerous routes when they traverse the environment.

Figure 4 illustrates our 2D environments. *Blocks* is a small environment with a number of potential routes that characters may choose. It was also used by Karamouzas et al. [9]. *Zelda* is a medium-sized village from a popular video game. *City* is a large virtual



(a) *Blocks* (b) *Zelda* (c) *City*

Figure 4: The three 2D environments used in our experiments. Obstacles are shown in gray, the medial axis is shown in blue, and boundaries of the density regions are shown in orange.

city with many polygonal obstacles and routes. More details can be found in Table 1.

Environment			Navigation Mesh	
Name	Width	Vertices	Vertices	Build Time
Layers	100m	332	248	38ms
Blocks	100m	52	132	25ms
Zelda	100m	560	1,243	49ms
City	500m	2,638	6,273	403ms

Table 1: Details of the four environments. The ECM was constructed with a GPU-based algorithm [4]. For *Layers*, *Blocks*, and *Zelda*, we used a resolution of 1,000x1,000 pixels. For *City*, the resolution was 4,000x4,000 pixels. This is the reason for the longer construction time.

Although our method supports individual character properties, we follow the suggestions of Weidmann [25] and model each character as a disk with a radius of 0.24m (i.e. an area of approximately 0.18m^2), and with a maximum speed of 1.4m/s. In Figure 1, we gave each character a random side preference within its homotopic route. In our experiments, all characters follow the medial axis. Characters are pushed towards their goal by attractive forces [10], at 10 frames per second.

No collision avoidance was used. Instead, we used the density-speed function $v(\rho)$ to decrease a character's maximum walking speed based on the density in its current region. This gives a good approximation of the slowdowns caused by high densities.

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 2 shows the time to perform each frame of the simulation for the entire crowd.

For all environments, the density measurements required at most 2ms per simulation step. Thus, updating the densities only marginally affects the running time, and crowds can still be steered in real-time.

Environment	Characters	Step Time: Density?	
		No	Yes
Layers	10,000	23ms	24ms
Blocks	10,000	15ms	16ms
Zelda	10,000	16ms	17ms
City	20,000	32ms	34ms

Table 2: Running times for simulation steps without and with density measurements.

3.3 Crowd Variety

Our second experiment used the *Blocks* environment to test the effect of planning paths with different values of the density weight w. We also compared simulations *without* replanning to simulations *with* replanning. In every simulation step, we added two characters at random positions at the bottom of the scene and with random goal positions at the top of the scene. We ran the simulation for 5,000 steps (500 seconds).

Without using density information (i.e., with $w = 0$), all of the characters moved along a shortest path in the graph. Hence, almost all of the paths ran through the environment's middle section. This led to traffic jams that slowed down the crowd.

With density information ($w = 1$) but without any replanning, characters in the *Blocks* environment initially took a shortest path. This led to traffic jams in the central sections. Newly created characters detected this congestion and chose other paths. These characters crowded the other routes while the central section was gradually emptied. This periodic behavior repeated several times during the simulation. On average, the walking speed of the characters was much higher while the traversed paths were not much longer.

We also ran the experiment with $w = 1$ while letting characters replan their paths every 100 steps. This spread the characters among the available routes, and the crowd did not leave any periodic gaps. On average, the speed of the characters was high while the traversed paths were still quite short. This setting led to the most natural-looking crowd flow.

With $w = 5$ and replanning, characters switched between routes more frequently and took large detours to avoid congested regions. While the average walking speed of the characters increased, their paths became much longer due to increased indecisiveness.

Table 3 shows the average path length and walking speed for each setting.

Experiment	Avg Path Length	Avg Speed
$w = 0$ (No Density)	$106.87m$	$0.67m/s$
$w = 1$	$115.90m$	$0.91m/s$
$w = 1$, Replanning	$114.85m$	$0.96m/s$
$w = 5$, Replanning	$132.96m$	$1.01m/s$

Table 3: The average path lengths and walking speed of the characters for different settings in *Blocks*.

Figure 1 shows the difference between $w = 0$ and $w = 1$ (with replanning) in the *Layers* environment. Without density awareness, nearly all characters follow the same route, causing a traffic jam. By contrast, when density information *is* considered, characters will naturally spread out among all of the available routes.

3.4 Replanning Efficiency

Our third experiment investigated how the replanning time can be reduced by changing the density viewing threshold D of the characters. We simultaneously added 5,000 characters to the *City* environment with random start and goal positions in such a way that the Euclidean distance between each character's start and goal was at least 100 meters. We set $w = 5$ for all experiments so that convincing detours would be explored. 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 5 illustrates the time to update an existing path based on the updated density information.

With $D = \infty$, each character could use all of the updated density values. Routes with 20 vertices took between $0.2ms$ and $1ms$ to compute, per character. Routes with 40 vertices took between $0.5ms$ and $2.5ms$. Routes with 60 vertices took between $1.5ms$ and $3ms$. On average, a character needed $2ms$ to replan a path.

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

With $D = 0m$, the characters could not use any density information, and replanning reduced to simply copying the remaining part of the old path. The average time to perform each replanning operation was $0.3ms$.

Notice that replanning was more expensive for larger values of D, because more vertices had to be explored before an old sub-path could be copied. Likewise, smaller values of D made replanning more efficient, in exchange for a loss of some density information.

Table 2 shows that 20,000 characters can be steered through *City* in $34ms$ per frame, leaving $66ms$ for other tasks such as replanning. If $D = \infty$, replanning takes $2ms$ on average, so at most 33 characters can replan their paths in one frame. Consequently, all 20,000 characters can replan their paths every $60.6s$. If $D = 350m$, this interval becomes $30.3s$. Hence, our method enables real-time periodic replanning for large simulations.

3.5 Multi-Threaded Speedup

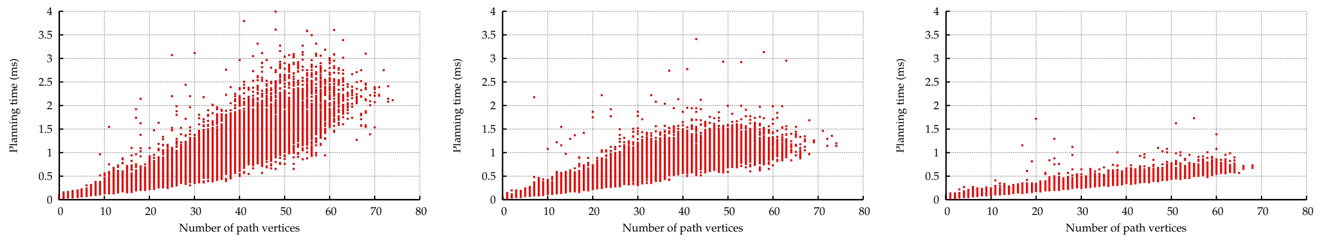
Finally, we have built the simulator using OpenMP technology so that it can steer multiple characters in parallel. Using only one CPU core, our method can steer 50,000 characters through the *Blocks* environment in $90ms$ per frame. With 4 CPU cores, we can reduce this time to $30ms$ per frame. These results show that our method is prepared for future hardware improvements.

4 Conclusion

Although it is common to steer virtual characters along short paths to their destinations, high crowd densities can lead to unnatural traffic jams while longer routes are underutilized. To enhance realism, we use crowd density information to guide a large number of characters along various routes. We do this by building a navigation mesh and weighting the desirability of routes based on the crowd density along the path. Our technique can guide tens of thousands of characters through a 2D or multi-layered environment in real-time.

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 based on 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. This *flow* information might play an important role in e.g. crowd evacuation scenarios. It would also be interesting to replan paths based on density-change events rather than replanning paths periodically.



(a) $D = \infty$ (b) $D = 350m$ (c) $D = 0m$

Figure 5: Replanning times for 5,000 characters in the *City* environment with (a) $D = \infty$, (b) $D = 350m$, and (c) $D = 0m$. 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.

Acknowledgments

This research has been supported by the GATE project (`http://gate.gameresearch.nl`), INCONTROL Simulation Solutions, and the Netherlands Organization for Scientific Research (NWO).

References

[1] J.P. van den Berg, M. Lin, and D. Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[2] W. Daamen. *Modelling passenger flows in public transport facilities*. PhD thesis, Delft University of Technology, 2004. Thesis number: T2004/6, TRAIL series.

[3] W. Daamen and S.P. Hoogendoorn. Level difference impacts in passenger route choice modelling. In *Proceedings of the 8th TRAIL conference: A world of transport, infrastructure and logistics*, pages 103–127, 2004.

[4] R. Geraerts. Planning short paths with clearance using Explicit Corridors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1997–2004, 2010.

[5] R. Geraerts and M.H. Overmars. Enhancing corridor maps for real-time path planning in virtual environments. *Computer Animation and Social Agents*, pages 64–71, 2008.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] M. Höcker, V. Berkhahn, A. Kneidl, A. Borrmann, and W. Klein. Graph-based approaches for simulating pedestrian dynamics in building models. In *eWork and eBusiness in Architecture, Engineering and Construction*, pages 389–394, 2010.

[8] M. Kallmann. Path planning in triangulations. In *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 49–54, 2005.

[9] I. Karamouzas, J. Bakker, and M.H. Overmars. Density constraints for crowd simulation. In *Proceedings of the ICE Games Innovations Conference*, pages 160–168, 2009.

[10] I. Karamouzas, R. Geraerts, and M.H. Overmars. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 113–120, 2009.

[11] I. Karamouzas, P. Heil, P. van Beek, and M.H. Overmars. A predictive collision avoidance model for pedestrian simulation. In *Proceedings of the 2nd International Workshop on Motion in Games*, pages 41–52, 2009.

[12] L.E. Kavraki, P. Švestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[13] A. Kneidl and A. Borrmann. 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.

[14] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3), 2005.

[15] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[16] M. Mononen. Recast Navigation. *Google Project: `http://code.google.com/p/recastnavigation`*, 2011.

[17] R. Narain, A. Golas, S. Curtis, and M.C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics*, 28:1–8, 2009.

[18] S. Patil, J.P. van den Berg, S. Curtis, M.C. Lin, and D. Manocha. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, 17:244–254, 2010.

[19] J. Pettré, H. Grillon, and D. Thalmann. Crowds of moving objects: Navigation planning and simulation. *IEEE International Conference on Robotics and Automation*, pages 3062 – 3067, 2007.

[20] J. Pettré, J.-P. Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *Proceedings of the First International Workshop on Crowd Simulation*, 2005.

[21] F. Preparata. The medial axis of a simple polygon. In *Mathematical Foundations of Computer Science*, volume 53, pages 443–450. Springer, 1977.

[22] S. Rabin. AI game programming wisdom 2. *Charles River Media Inc., Hingham*, 2004.

[23] W.G. van Toll, A.F. Cook IV, and R. Geraerts. Navigation meshes for realistic multi-layered environments. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 3526–3532, 2011.

[24] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Transactions on Graphics*, 25:1160–1168, 2006.

[25] U. Weidmann. *Transporttechnik der Fussgänger - Transporttechnische Eigenschaften des Fussgängerverkehrs*. Literature Research 90, ETH Zürich, Institut für Verkehrsplanung, Transporttechnik, Strassen- und Eisenbahnbau, 1993. In German.

[26] R. Wein, J.P. van den Berg, and D. Halperin. The Visibility-Voronoi Complex and its applications. *Computational Geometry: Theory and Applications*, 36(1):66–78, 2007.

[27] B. Yersin, J. Maïm, F. Morini, and D. Thalmann. Real-time crowd motion planning: Scalable avoidance and group behavior. *The Visual Computer*, 24:859–870, 2008.