Using the Corridor Map Method for Path Planning for a Large Number of Characters

Roland Geraerts, Arno Kamphuis, Ioannis Karamouzas, and Mark Overmars

Institute of Information and Computing Sciences, Utrecht University
3508 TA Utrecht, the Netherlands
roland@cs.uu.nl

Abstract. A central problem in games is planning high-quality paths for characters avoiding obstacles in the environment. Current games require a path planner that is fast (to ensure real-time interaction) and flexible (to avoid local hazards). In addition, a path needs to be natural, meaning that the path is smooth, short, keeps some clearance to obstacles, avoids other characters, *etcetera*.

Game worlds are normally populated with a large number of characters. In this paper we show how the recently introduced Corridor Map Method can be extended and used to efficiently compute smooth motions for these characters. We will consider crowds in which the characters wander around, characters have goals, and characters behave as a coherent group.

The approach is very fast. Even in environments with 5000 characters it uses only 40% of the processing power of a single core of a CPU. Also the resulting paths are indeed natural.

1 Introduction

One of the main challenges of applications dealing with virtual environments is path planning for characters. These characters have to traverse from a start to a goal position in the virtual world without colliding with obstacles and other characters. In the past twenty years, many algorithms have been devised to tackle the path planning problem [1,2]. These algorithms were mainly developed in the field of robotics, aiming at creating a path for one or a few robots having many degrees of freedom. Usually, much CPU time was available for computing a *nice path*, which often meant a short path having some clearance to the obstacles, because a bad path could be expensive to traverse, and could damage the robot or environment.

While these algorithms were successfully applied in fields such as mobile robots, manipulation planning and human robot planning [1], current virtual environment applications, such as games, pose many new challenges to the algorithms. That is, *natural* paths for many characters traversing in the ever growing environments need to be planned simultaneously and in real-time. Consequently, only a (fraction of a) millisecond per second CPU time may be spent per character for computing the natural path (i.e. a path that is smooth, short, keeps some clearance to obstacles, avoids other characters, *etcetera*).

A. Egges, A. Kamphuis, and M. Overmars (Eds.): MIG 2008, LNCS 5277, pp. 11–22, 2008.
© Springer-Verlag Berlin Heidelberg 2008

In conclusion, current virtual environments require a fast flexible planner which can generate natural paths. A candidate for the flexible planner is a Potential Field method [3], because it can be used to evade characters and to create smooth paths. Due to the method's local behavior, it will not always find a path.

Roadmap based methods, such as Visibility graphs [1], Probabilistic Roadmap Methods [2], and the A* method operating on a graph-like grid [4], can usually ensure that a path can be found if one exists. However, they lack flexibility because they output a fixed path (extracted from a one-dimensional graph). In addition, the paths are unnatural. While some optimization algorithms exist, they are too slow to be applied in real-time [5].

Recently, the Corridor Map Method (CMM) has been proposed, which satisfies the requirements mentioned above [6]. The CMM directs the global motions of a character traversing a corridor. Such a corridor is extracted from the *corridor map* which is a graph with clearance information. Local motions are controlled by potential fields inside a corridor, providing the desired flexibility.

In this paper, we show how the CMM can be used to simultaneously plan the motions of a large number of characters in real-time. To this end, we first, in Section 2, indicate how to create high-quality corridor maps and how to extract a path efficiently. Next, in Section 3, to obtain more natural motions, we introduce variations in the local behavior of the characters. In particular, we will give a simple approach to get natural lane formation. Also, in Section 4, we describe how the characters can avoid each other. Based on this we introduce in Section 5 two ways of modeling large crowds. The first method uses goal oriented behavior by repeatedly planning paths for the individual characters. The second method uses the corridor map itself to create wandering behavior. Finally, in Section 6, we show how we can plan the motions of coherent groups of characters. Experiments show that we can plan the simultaneous motions of thousands of characters in real-time making the CMM favorable over common A* techniques.

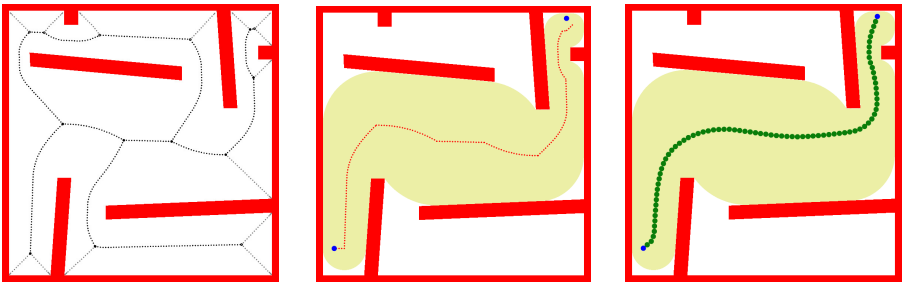
2 The Corridor Map Method

The Corridor Map Method (CMM) consists of an off-line *construction phase* and an on-line *query phase* [6]. These two phases are visualized in Fig. 1.

In the construction phase, a corridor map is created, representing the free space (i.e. the space that is not occupied by the static obstacles) of the environment. The skeleton of the corridors is a graph. We refer the reader to Fig. 1(a) for an example.

A corridor consists of a backbone path and a set of balls centered around this path. A corridor $\mathcal{B} = (B[t], R[t])$ is defined a sequence of maximum clearance balls with radii $R[t]$ whose center points $B[t]$ lie along its backbone path B. The parameter t is an index ranging between 0 and 1, and B is defined as a list of coordinates.

In the query phase, we connect the start and goal position of the character (modeled by a ball with radius r) to the graph and find the shortest backbone path in the graph connecting these positions. From the map, we now extract



(a) Corridor map (b) Corridor and query (c) Path

Fig. 1. The construction phase (a) and the query phase (b,c) of the CMM

a corridor which is formed by concatenating the corridors corresponding to the edges being part of this path. These steps sre visualized in Fig. 1(b).

While the corridor guides the global motions of the character, its local motions are led by an *attraction point*, $\alpha(x)$, moving on the backbone path of the corridor toward the goal. The attraction point is defined such that making a step toward this point leads the character, located at position x, toward the goal.

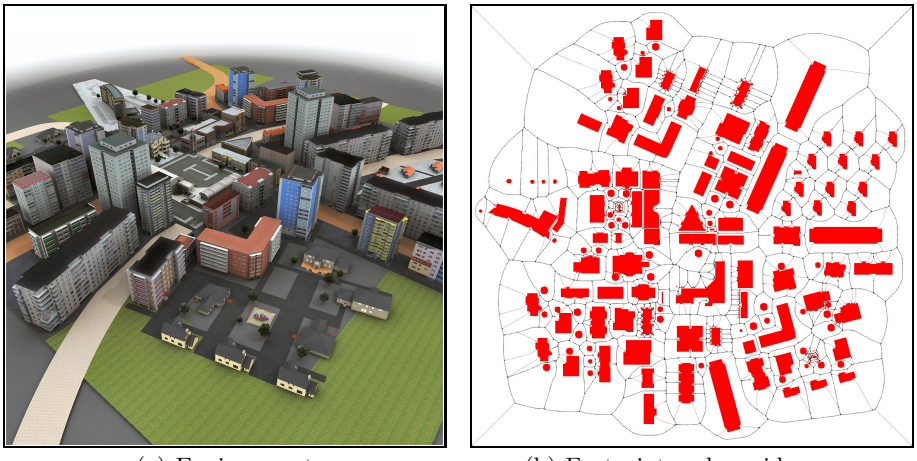
The attraction point attracts the character with force $\mathbf{F}_a$. Let d be the Euclidean distance between the character's position and the attraction point $\alpha(x)$. Then $\mathbf{F}_a(x) = f\frac{\alpha(x)-x}{||\alpha(x)-x||}$, where $f = \frac{1}{R[t]-r-d} - \frac{1}{R[t]-r}$. The scalar f is chosen such that f is 0 when the character is positioned on the attraction point. In addition, f is ∞ when the character touches the ball's boundary.

Additional behavior can be incorporated by adding extra forces to $\mathbf{F}_a$, resulting in a force $\mathbf{F}$. The final path is obtained by iteratively integrating $\mathbf{F}$ over time while updating the velocity, position and attraction point of the character. In [6], it is proved that the resulting path is smooth (i.e. C^1-continuous). An example of such a path is displayed in Fig. 1(c).

2.1 Creating High-Quality Corridor Maps

An important impact on the quality of the paths is the quality of the corridor map. In [7], we described an approach to create high-quality maps. The corridors of the map were extracted from the Generalized Voronoi Diagram (GVD) [8].

Using the GVD as basis for the map has three main advantages. First, if a path exists in the free space then it can always be found (because the GVD is a complete representation of the free space). Second, a GVD includes all cycles present in the environment. These cycles provide short global paths and alternative routes which allow for variation in the characters' routes. Third, corridors extracted from the GVD have a maximum clearance. Such a corridor provides maximum local flexibility.



(a) Environment (b) Footprint and corridor map

Fig. 2. The City environment. While its geometry is three-dimensional, its footprint (used for generating the corridor map) is only two-dimensional.

To keep the map small and to improve the running times in the query phase, we sampled the corridors such that the coverage of the free space was still large while the efficiency in the query phase was high.

As an example, consider Fig. 2 which shows our test environment. The city measures 500x500 meter. Its geometry, displayed in Fig. 2.1, is composed of 220K triangles. Using this number of triangles will lead to a low performance. Instead, we used its footprint (2,122 triangles) to generate the corridor map, see Fig. 2.1. This took 0.64 seconds on a PC with a NVIDIA GeForce 8800 GTX graphics card and an Intel Core2 Quad CPU (2.4 GHz) with 4 GB memory. (While this processor has four CPU's, our application, implemented in Visual C++ and run under Windows XP, used only one core.) The map comprised 1,434 vertices, 1,606 edges and 25,863 samples (i.e. the total number of balls in all corridors).

2.2 Fast Extraction of a Corridor

In [7], we elaborated on how to facilitate efficient extraction of corridors and paths. We showed that the average extraction time for a corridor (excluding the times for finding the shortest backbone path and connecting the query) was 0.87 ms in the City environment. (We took the average of 10,000 random corridors.)

We looked at two different algorithms for finding the shortest corridor enclosing the query. Experiments showed that Dijkstra's algorithm increased the time by 138.2% while A* increased the time by only 11.5%. Then, we looked at two different approaches for connecting a query to the corridor map. Using linear search increased the time by 59.7% while using a search structure based on a *kd*-tree only increased the time by 1.0%. In conclusion, by using A* and a *kd*-tree, the average extraction time of a corridor was low, i.e. 1.19 ms.

2.3 Fast Extraction of a Path

We wanted to know how fast we could compute a path. We extracted 10,000 paths in the City environment and recorded the average running time. Experimental results showed that computing a path took 1.8 ms (including the 1.19 ms for computing a corridor). To view this result in the right perspective, we defined the CPU-load. That is, the CPU-load is the total CPU time / averaged traversed time $*$ 100%. Since the averaged traversed time of a character (walking at 1.2 m/s) was 260 seconds, the CPU-load is 0.00069%. Hence, the CMM can be used for steering many characters simultaneously. This will be discussed further in Section 5 and 6.

3 Path Variation

The original CMM can be easily extended to create high-quality alternative paths that a character can follow within a corridor. This not only presents a more challenging and less predictable opponent for the player, but also enhances the realism of the gaming experience.

To generate slightly different paths every time the same path planning problem has to be solved, a random force (bias) is added to the attractive force $\mathbf{F}_a$. A coherent-noise function like Perlin Noise [9] can be used to control the direction of the bias, ensuring that it will change smoothly at every step of the integration.

In our problem setting, Perlin Noise is implemented as a 2D function. The current attraction point $\alpha(x)$ is given as an input and a direction for the bias is computed which varies pseudo-randomly. Let θ denotes this direction expressed as an angle of rotation with respect to the current attractive vector, i.e. $\alpha(x)-x$. Then, the random force can be defined as:

$$\mathbf{F}_{rand} = c_{rand}\ \mathcal{R}(\theta)\frac{\alpha(x)-x}{||\alpha(x)-x||},$$

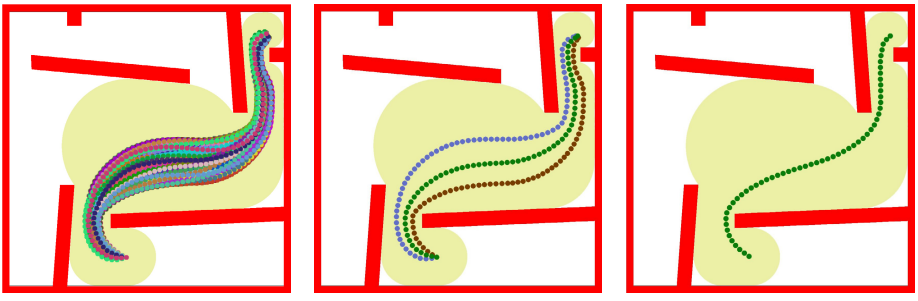
where c_{rand} is a small constant specifying the relative strength of the force, $\mathcal{R}(\theta)$ represents the 2D rotation matrix and the angle $\theta \in [-\pi/2, +\pi/2]$.

The quality of the computed path is affected by the frequency of the noise function. A too high frequency noise leads to the retrieval of unrealistic paths, since the character will change its direction at almost every successive time step. On the contrary, a low frequency results in smooth changes, generating aesthetically pleasant paths like the ones depicted in Fig. 3.

As individuals usually show a preference for certain paths over others, deterministic variations of paths are also necessary to simulate behaviors that have been widely noted in the crowd and pedestrian literature.

For example, lanes can be formed on either side of the corridor by exerting at every time step a force perpendicular to the attractive force, hence $\theta \in \{-\pi/2, +\pi/2\}$. A positive angle steers the character to the left of the backbone path, whereas a negative one to the right. The perpendicular force is defined as:

$$\mathbf{F}_{perp} = c_{perp}\ \mathcal{R}(\theta)\frac{\alpha(x)-x}{||\alpha(x)-x||},$$



(a) 100 random paths (b) Lane formation inside the corridor (c) Following a shorter and more direct path

Fig. 3. Generating high-quality alternative paths using the CMM

where $c_{perp} = k\frac{R[t]-2r}{d}$ and k is a constant used to specify how close to the boundary of the corridor the character moves.

We refer the reader to Fig. 3 for a graphical representation of the described technique. Different left/right paths are computed by varying the value of the constant k.

Another example, resulting in a more natural and visually interesting movement, is to generate paths that either take tight or wide corners. In this approach the direction of the backbone path is used to bias the character's motion [10]. The character looks intelligent, in the sense that it anticipates the direction of the path it has to follow and starts to turn in advance. A convincing path is computed as can be examined in Fig. 3.

3.1 Results

We implemented the presented approaches to test their applicability in real-time applications like computer games. Since the proposed methods can be computed efficiently, which was experimentally confirmed in [10], they influence the running time of the algorithm marginally. Consequently, alternative paths can be computed in real-time.

Apart from the performance, we are also interested in the quality of the resulting paths. Experiments in [10] have indicated that our proposed techniques can successfully generate alternative routes that are aesthetically pleasant to the observer.

The first method uses a noise function to generate slightly different paths in response to a given query. As expected, the computed paths have almost the same clearance and length as the smooth path computed by the original CMM.

The second method forms different lanes on either side of the extracted corridor. Although the computed paths can get close to the boundary of the corridor, they still keep a safe amount of clearance. In addition, no significant difference in the length of the paths is observed.

In the third method the character takes shortcuts through the turns of the backbone path, trying to avoid any unnecessary detour and minimize the time needed to reach its goal. Thus, a shorter and more direct path is generated.

In conclusion, our techniques extend the basic functionality of the CMM by creating in real-time high-quality alternative paths. Combined with existing agent-based models the proposed methods lead to convincing characters that exhibit human-like path planning as discussed below.

4 Obstacle Avoidance

When we are dealing with more than one character, we have to choose an appropriate force such that characters evade each other naturally.

We use a part of Helbing and Molnár's social force model for collisions avoidance because their simulations have shown that it exhibits realistic crowd behavior [11]. The model describes three force functions which represent the acceleration toward the desired velocity of motion, the behavior of characters keeping a certain distance to other characters and borders, and attractive effects among characters. The force $\mathbf{F}_a$ described in Section 2 captures the effect of their acceleration force and the force keeping characters away from borders. As collision avoidance force, $\mathbf{F}_c$, we use the force described in equation (3) of their paper. Unlike [11], we do not model attractive effects among characters.

In the computation of force $\mathbf{F}_c$, we have to find the set of neighbors for each character. Since this operation is carried out many times, we have to revert to an efficient data structure for answering nearest neighbors queries. We maintain a 2D grid storing the locations of all characters (with radius r). Each cell in the grid stores a sorted set of character id's. Preliminary experiments have shown that the cell size c can be chosen fairly small, e.g. $c = 3 + 2r$ meter, to obtain realistic collision avoidance. By including the term $2r$, we only have to check 3 by 3 cells for carrying out a nearest neighbors query. Also, an update corresponding to a changed position of a character is efficient since we use a set.

5 Crowd Simulation

Real-time crowd simulation has gained much attention recently [11, 12, 13, 14]. It requires the modeling of group behavior, pedestrian dynamics, path planning and graphical rendering [13]. We focus on the path planning part.

We model two types of behavior for the characters, i.e. *goal-oriented* and *wandering* behavior. In Section 5.1, we discuss how to model a crowd in which each character has its own (long term) goal. Next, we elaborate on wandering behavior in Section 5.2 which comprises making decisions more locally.

5.1 Goal Oriented Behavior

Goal oriented behavior can be achieved easily by assigning each character a random start and goal position. These positions will fix a corridor. When a character has reached its goal, a new goal will be chosen, *ad infinitum*.

If we only used force $\mathbf{F}_a$ to guide characters toward their goals (and force $\mathbf{F}_c$ to avoid each other), they would have the tendency to clutter up around

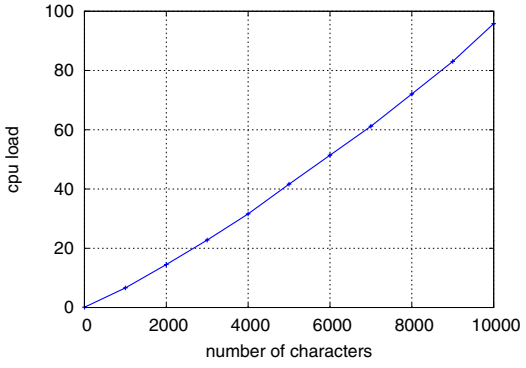


Fig. 4. The relation between the number of characters in the crowd simulation and the CPU-load

their backbone paths. To enforce a nice spread (and path variation) inside the corridors, we add the force $\mathbf{F}_{perp}$ biasing a character to the right with respect to its desired direction.

The simulation consists of some user defined number of iterations. For each character, the task per iteration includes computing the forces, integrating the final force $\mathbf{F} = \mathbf{F}_a + \mathbf{F}_c + \mathbf{F}_{perp}$, adding the new position of the character to its path, and possibly choosing a new goal (and corresponding corridor).

Results. We integrated force $\mathbf{F}$ with Verlet integration with step size $\Delta t = 0.1$ and set the maximum speed to 1.2 m/s [15]. The cell size of the nearest neighbor grid was set to $c = 3 + 2r = 3.5$ meter. In the experiments, we measured the CPU-load for a varying number of characters. When the CPU-load was at most 100%, we considered the performance as real-time.

We simulated crowds in the City Environment from Fig. 2 with a varying number of characters (with a radius of 25 cm, i.e. $r = 0.25$). Fig. 4 shows the performance of our application for crowds with up to 10,000 characters. The figure makes clear that approximately 10,000 characters can be simulated in real-time. In addition, the performance does not degrade significantly when the environment becomes rather crowded. We conclude that the CMM can be used for real-time path planning with many characters.

5.2 Wandering Behavior

Rather than using the corridor map to plan paths for each of the characters in the crowd, there is also an alternative approach in which the characters simply wander around. The idea is as follows. As before, for a character at position x there is an attraction point $\alpha(x)$. This attraction point lies on some edge ϵ of the graph and we assume that we have decided on a direction along which the attraction point will move along ϵ toward a vertex ν. We compute forces as before and move the character accordingly. Next, we update the attraction point by moving it further along ϵ toward ν. If the attraction point reaches ν,

we randomly pick one of the outgoing edges ϵ' of vertex ν, unequal to edge ϵ, and continue moving the attraction point along edge ϵ'. So the attraction point will follow some random walk through the graph and the character follows the attraction point.

There is one complication. When ν is a dead end in the graph, that is, it has only edge ϵ as an outgoing edge, the attraction point must move backwards along the same edge. However, because of the way attraction points are defined (i.e. the furthest point for which x still lies in the clearance disk) the result is that the attraction point jumps far back along the edge, and, hence, the characters will not even get close to vertex ν, leaving part of the dead end in the environment void of characters. To remedy this we make a distinction between short dead ends and long dead ends. We recursively remove short edges that represent dead ends. For long dead ends we create some invisible obstacle at the end vertex ν and create a corridor around it. As a result, characters will move till the end of the dead end, move around the invisible obstacle, and return again along the edge.

The method is very fast because no searches in the graph are required. Preliminary experiments show, in particular when combined with lane formation as described in Section 3, that the approach leads to natural wandering behavior. We can extend the method by giving certain edges preference over other edges and choosing the random continuation edge ϵ' taking these preferences into account. The method can also be combined easily with other characters that do have goal oriented behavior.

6 Coherent Groups

A virtual environment, such as a city, is usually populated with many characters which often operate in groups. Take for example a guided tour through the city. Here, each group member needs to stay in close proximity to other members while moving from one location in the city to another. The CMM enables us to plan such paths very efficiently [16].

For a coherent group of characters, the distances between the characters need to be limited. This can be achieved in two directions, i.e. the *lateral* and *longitudinal direction*. The lateral direction is the direction perpendicular to the direction of movement, i.e. the direction of the backbone path. The longitudinal direction is the direction along the backbone path. We refer to the separation of the characters in the lateral and longitudinal directions as the lateral dispersion and the longitudinal dispersion, respectively. Please note, if the dispersion increases, the coherence decreases.

By using the CMM to create a corridor for the group, the lateral coherence is bounded, namely by the radius $R[t]$ of the clearance balls on every point on the backbone path. Since no character can leave the corridor, the maximum lateral distance between any two characters is limited by the maximum radius of all clearance balls along the backbone path. However, this maximum radius might be too large. Therefore, we introduce a constant g_w that represents the maximum

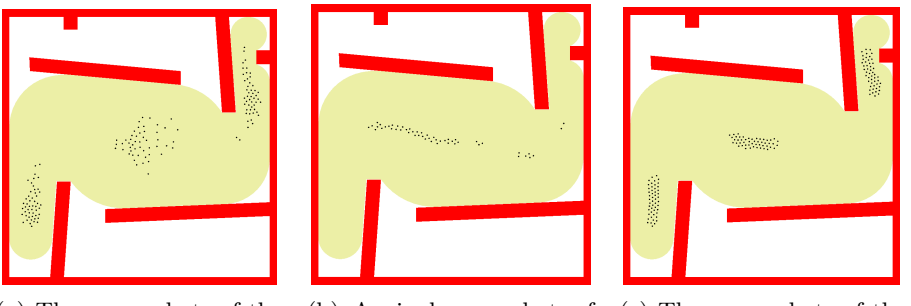
width of the coherence group. Using this constant we adapt the corridor $\mathcal{B}$ to $\mathcal{B}^* = (B[t], \max\{R[t], g_w\})$.

The longitudinal dispersion can be bounded in a different manner. Let us introduce another constant, g_A, called the group area, which represents the area in the corridor that the group is allowed to occupy. The group area is defined as the union of clearance balls from a start point to an end point on the backbone path. The start point is defined as the center of the furthest advanced ball in which the least advanced character is enclosed. The end point is defined as the center of the least advanced ball in which the furthest advanced character is enclosed. Now, we define the minimum attraction point, α_{min}, as the least advanced attraction point of all attraction points, $\alpha(x_i)$, of the characters in the group. By definition, this is the start point of the group area. The maximum allowed attraction point, α_{max}, is defined as the point on the backbone path such that the union of balls from α_{min} to α_{max} is exactly g_A. If an attraction point $\alpha(x)$ for any character is further advanced on the backbone path than the maximum attraction point α_{max}, the maximum attraction point is used as the attraction point of the character. Consequently, characters that are in front of the group will be attracted backward, making them wait for the rest.

By varying the two parameters, g_w and g_A, we can influence the behavior of the group. That is, high values result in a weak coherent group while small values result in a strong coherent group. Please note that the area should not be chosen too small, otherwise the characters will not fit inside this area.

6.1 Results

We generated several paths for a group of 50 characters with varying degrees of coherence. The first series of paths were created with a large value for both g_w and g_A ($g_w = 30$, $g_A = 1000$). A snapshot of the paths is depicted in Fig. 5(a).



(a) Three snapshots of the group at different stages on the paths. The width of the corridor is $g_w = 30$, the group area is $g_A = 1000$

(b) A single snapshot of the group. The width of the corridor is $g_w = 3$, the group area is $g_A = 1000$

(c) Three snapshots of the group at different stages on the paths. The width of the corridor is $g_w = 3$, the group area is $g_A = 60$

Fig. 5. The effect on different lateral and longitudinal dispersions on 50 characters

This results in a weak coherent group, where the characters are scattered over the whole corridor. By decreasing the width of the corridor ($g_w = 3$, $g_A = 1000$) the group becomes more coherent in the lateral direction, see Fig. 5(b). However, the longitudinal coherence is weak. By also decreasing the area of the group ($g_w = 3$, $g_A = 60$) the group becomes more coherent in both directions, see Fig. 5(c).

7 Conclusions and Future Work

In this paper we have described how the recently introduced Corridor Map Method (CMM) can be used to plan the motions of thousands of characters in a virtual world in real-time. We considered crowds of wandering characters without clear goals, characters with individual goals, and groups of characters.

The advantage of using the CMM is that it is fast, it is flexible, and it produces natural paths. Because of the flexibility it is easy to introduce additional constraints on the paths of the characters. For example, we can incorporate all three types of motions simultaneously. Also, we can add additional static or dynamic obstacles that must be avoided. In addition, we can incorporate personal preferences on the motions of the characters, for example to improve the animations.

Currently, we are investigating how we can incorporate influence regions in the environment. Such regions can either be dangerous places the characters prefer to avoid or nice places that they prefer to visit. Also we work on techniques to improve the way the characters avoid each other by extending the Helbing model. And finally, we are experimentally validating the model and approach by observing real people using cameras and motion capture equipment. We expect this to lead to even more natural character motions.

Acknowledgements

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie). In addition, part of this research has been funded by the Dutch BSIK/BRICKS Project.

References

1. Latombe, J.-C.: Robot Motion Planning. Kluwer, Dordrecht (1991)
2. LaValle, S.: Planning Algorithms (2006), `http://planning.cs.uiuc.edu`
3. Rimon, E., Koditschek, D.: Exact robot navigation using artificial potential fields. IEEE Transactions on Robotics and Automation 8, 501–518 (1992)
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 4, 100–107 (1968)

5. Geraerts, R., Overmars, M.: Creating high-quality paths for motion planning. International Journal of Robotics Research 26, 845–863 (2007)
6. Geraerts, R., Overmars, M.: The corridor map method: A general framework for real-time high-quality path planning. Computer Animation and Virtual Worlds 18, 107–119 (2007)
7. Geraerts, R., Overmars, M.: Enhancing corridor maps for real-time path planning in virtual environments. In: Computer Animation and Social Agents (2008)
8. Hoff, K., Culver, T., Keyser, J., Lin, M., Manocha, D.: Interactive motion planning using hardware-accelerated computation of generalized Voronoi diagrams. In: IEEE International Conference on Robotics and Automation, pp. 2931–2937 (2000)
9. Perlin, K.: An image synthesizer. Computer Graphics 19(3), 287–296 (1985); SIGGRAPH 1985 Proceedings
10. Karamouzas, I., Overmars, M.H.: Adding variation to path planning. In: Computer Animation and Social Agents (2008)
11. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Physical Review 51, 4282–4287 (1995)
12. Morini, F., Yersina, B., Maïm, J., Thalmann, D.: Real-time scalable motion planning for crowds. In: International Conference on Cyberworlds, pp. 144–151 (2007)
13. Sud, A., Gayle, R., Andersen, E., Guy, S., Lin, M., Manocha, D.: Real-time navigation of independent agents using adaptive roadmaps. In: ACM symposium on Virtual reality software and technology, pp. 99–106 (2007)
14. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. Transactions on Graphics 25, 1160–1168 (2006)
15. Knoblauch, R., Pietrucha, M., Nitzburg, M.: Field studies of pedestrian walking speed and start-up time. Transportation Research Record, 27–38 (1996)
16. Kamphuis, A., Overmars, M.: Finding paths for coherent groups using clearance. In: Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, pp. 19–28 (2004)