Flexible Path Planning Using Corridor Maps

Mark Overmars, Ioannis Karamouzas, and Roland Geraerts

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

Abstract. Path planning is a central problem in virtual environments and games. When computer-controlled characters move around in virtual worlds they have to plan their paths to desired locations. These paths must avoid collisions with the environment and with other moving characters. Also a chosen path must be natural, meaning that it is the kind of path a real human being could take. The algorithms for planning such paths must be able to handle hundreds of characters in real-time and must be flexible.

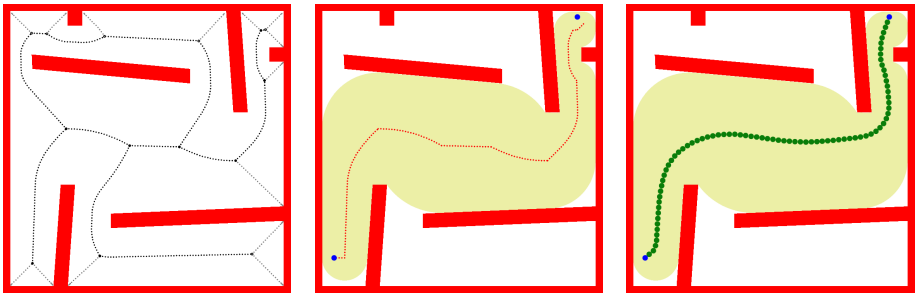
The Corridor Map Method (CMM) was recently introduced as a flexible path planning method in interactive virtual environments and games. The method is fast and flexible and the resulting paths are reasonable. However, the paths tend to take unnatural turns when characters get close to other characters or small obstacles. In this paper we will improve on the CMM by decoupling collision avoidance with the environment and local steering behavior. The result is a method that keeps the advantages of the CMM but has much more natural steering. Also the method allows for more flexibility in the desired routes of the characters.

1 Introduction

Virtual worlds are nowadays commonly used in computer games, simulations, city models, and on-line communities like Second Life. Such worlds are often populated by computer-controlled characters. The characters must move around in the environment and need to plan their paths to desired locations. These paths must avoid collisions with the environment and with other moving characters. Also a chosen path must be natural, meaning that it is the kind of path a real human being could take. The algorithms for planning such paths must be able to handle hundreds of characters in real-time and must be flexible to e.g. avoid local hazards or incorporate animation constraints.

The path planning or motion planning problem had received considerable attention over the past twenty years and many algorithms have been devised to tackle it. (See [1, 2] for an overview.) 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. In virtual worlds the requirements though are completely different. The environment is very complex and even though path planning normally can be performed in the 2-dimensional footprint of the environment, we still need to deal with thousands of polygons. We need to plan the

D. Halperin and K. Mehlhorn (Eds.): ESA 2008, LNCS 5193, pp. 1–12, 2008.



(a) Corridor map (b) Backbone path (c) Resulting path

Fig. 1. The Corridor Map Method in action

motion of hundreds of characters in real-time using only a small percentage of the CPU time. Per character only a (fraction of a) millisecond per second CPU time may be spent on path planning. Also paths need not only be collision-free but they must also be natural. On the positive side, we can represent the character as a disk, and, hence, have to deal with only two degrees of freedom of movement.

In conclusion, virtual world applications require algorithms for path planning that are fast, flexible, and generate natural paths. In practice, currently two approaches are common. The first is to let designers script most of the motion, for example using waypoints, and then using potential field approaches (see e.g. [3]) to avoid obstacles and other characters. Such an approach is only possible when the virtual world is predefined by designers. It is also expensive because of the manual work involved. In addition, the method is not very flexible. The potential field approach has the risk of characters getting stuck in local minima and not reaching their goals. Also, as can be seen from many (recent) games, it leads to rather unnatural paths, in particular when waypoints get blocked.

The second common approach is the put a grid on the world and using searches based on A* to create a path through the empty cells. See for example [4, 5]. This method is guaranteed to find a path if one exists. However, it lacks flexibility because a single fixed path is returned. In addition, the paths tend to be unnatural. Also, even though some optimization algorithms exist, when the grids get large and the motion of many characters must be planned, the approach can become too slow to be applied in real-time [6].

Recently, the Corridor Map Method (CMM) has been proposed as a new path planning method in interactive virtual environments and games [7]. The method is fast and flexible and the quality of resulting paths is reasonable. Globally speaking the CMM works as follows (see Fig. 1 for an example). In a preprocessing phase a roadmap of paths is computed for the static part of the environment. Often the medial axis is used for this. With the roadmap, clearance information is stored, defining collision-free corridors around the roadmap edges. This data structure is called the *corridor map*. When a path planning query must be solved a *backbone path* is extracted from the roadmap together with a collision-free *corridor* around it. We move an *attraction point* along the backbone path which

attracts the character in such a way that no collisions occur with the environment. This leads the character toward the goal. Local motions are controlled by potential fields inside a corridor, providing the desired flexibility.

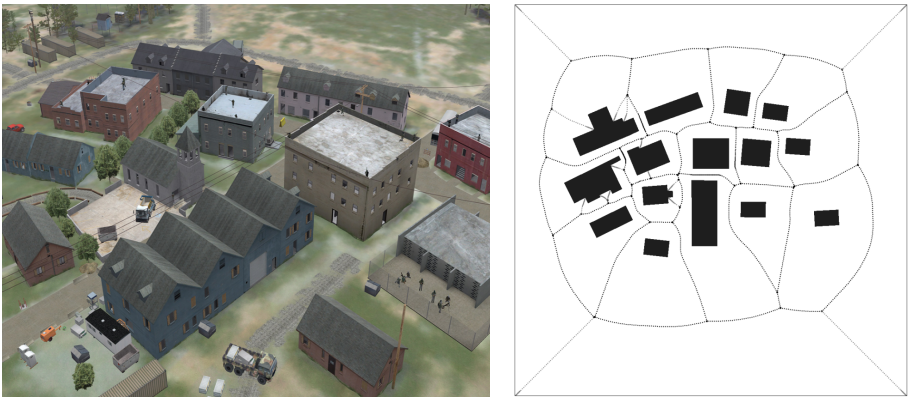
Although the CMM is fast and flexible, the paths tend to take unnatural turns when characters get close to other characters or small obstacles. In this paper we will extend the CMM as follows. We separate the *corridor map* from the so-called *control network*. The corridor map is defined as above. The control network provides a roadmap of paths that can be used to lead the characters to their goals. When a query must be solved a *control path* is extracted from the control network. With the control path we find a corresponding *corridor* in the corridor map. We again move an *attraction point* along the control path but this is only used to lead the character to the goal. Separate forces are used to keep the character inside the corridor. Again we use additional forces to steer the character away from other characters, small obstacles and other hazards. As we will show, separating the collision-avoiding forces in the corridor from the attraction forces along the control path, leads to much more natural paths while hardly increasing the computation time. We initially still use the medial axis for the control network but we will also show how even more flexibility can be obtained by using other control networks and paths.

This paper is organized as follows. In Section 2 we provide definitions of corridors and corridor maps and show how such maps can be computed efficiently. In Section 3 we briefly review the original approach for using corridors for path planning. In Section 4 we present our improved approach in which we use the medial axis as a control network to obtain more natural paths. In section 5 we provide results from experiments that show that the resulting paths are considerably better than those produced by the original CMM. In Section 6 we will indicate how the approach can be extended using other control networks and control paths. Finally, in Section 7 we provide some conclusions and plans for further research.

2 The Corridor Map

The *corridor map* is an efficient data structure representing the (walkable) free space in the environment. It was introduced by Geraerts and Overmars [7] and we will outline the most important aspects here. As the walkable space is normally 2-dimensional we will define the corridor map in the plane. The obstacles are the footprints of the original 3-dimensional obstacles in the environment.

The corridor map is a graph whose edges represent collision-free *corridors*. Such a corridor consists of a *backbone path* and a set of disks centered around this path. More formally, a corridor $\mathcal{B} = (B[t], R[t])$ is defined as a sequence of maximum clearance disks with radii $R[t]$ whose center points lie along its backbone path $B[t]$. The parameter t is an index ranging between 0 and 1, and $B[t]$ denotes the coordinates of the center of the disk corresponding to index t. Together, the backbone paths form the *skeleton* of the corridor map. See Fig. 2 for an example of a virtual city, its footprint, and the skeleton defining the corridor map.



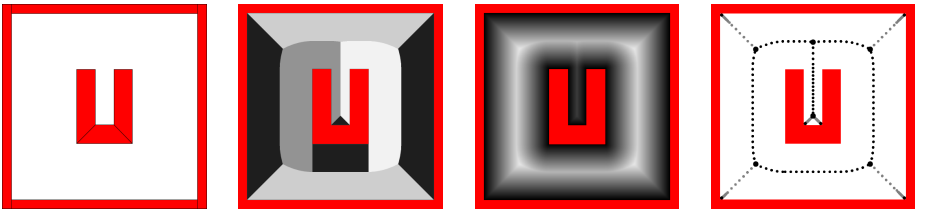
(a) 3D model (b) Footprint and skeleton

Fig. 2. The McKenna MOUT training site at Fort Benning, Georgia, USA

We will use the corridors to provide the flexibility to handle a broad range of path planning issues, such as avoiding other characters and computing natural paths. To approach these issues, we set the following requirements for the corridor map. First, if a path exists in the free space then a corridor must exist in the map that leads the character from its start to goal position. Second, the map includes all cycles that are 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 map have a maximum clearance. Such a corridor provides maximum local flexibility.

These requirements are met by using the Generalized Voronoi Diagram (GVD) as skeleton for the corridor map [8]. A GVD is a decomposition of the free space into regions such that all points p in a region $R(p)$ are closer to a particular obstacle than to any other obstacle in the environment. Such a region is called a *Voronoi region*. The boundaries of the Voronoi regions form the skeleton (i.e. the underlying graph) of the corridor map. We refer the reader to Fig. 2(b) for an example. The boundaries are densely sampled and with each such sampled point, we store the radius of the maximum clearance disk centered at this point. A sequence of these disks forms the corridor.

A GVD can be computed efficiently by exploiting graphics hardware. Like in [9], we compute a 3D distance mesh, consisting of polygons, for each geometric obstacle present in the footprint of the environment. Each of the meshes is rendered on the graphics card in a different color. A parallel projection of the upper envelope of the arrangement of these meshes gives the GVD. The diagram can be retrieved from the graphics card's frame buffer and the clearance values (i.e. distance values) can be found in the Z-buffer. These steps are visualized in Fig. 3. The approach is very fast. For example, the corridor map in Fig. 2(b) was computed in 0.05 seconds on a modern PC with a NVIDIA GeForce 8800 GTX graphics card. Note that the computation of the corridor map happens only once during preprocessing.



(a) Environment (b) Frame buffer (c) Z-buffer (d) Corridor map

Fig. 3. Construction of the Corridor map using graphics hardware

3 The Original Corridor Map Method

In our original description [7], the corridor map is used as follows to answer path planning queries. To plan a path for a character, which is modeled by a disk with radius r, we first compute the shortest backbone path connecting the start to the goal. After connecting the start and goal positions to the roadmap, this backbone path is obtained by applying the A* shortest path algorithm on the skeleton graph. The corresponding corridor is formed by concatenating the corridors of the edges of the backbone path. See Fig. 1(b) for an example of a backbone path.

The backbone path guides the global motions of the character. Its local motions are controlled by continuously applying one or more forces to the character. The basic force steers the character toward the goal and keeps the character inside the corridor. For this purpose, we create an *attraction point* $\alpha(x)$ that runs along the backbone path and attracts the character.

Definition 1 (Attraction point)**.** *Let x be the current position of the character with radius r. The attraction point $\alpha(x)$ is the point $B[t]$ on the backbone path B having the largest time index $t : t \in [0 : 1]$ such that Euclidean distance $(x, B[t]) < R[t] - r$.*

The character is attracted to the attraction point with force $\mathbf{F}_a$. Let d be the Euclidean distance between the character's position x and the attraction point $\alpha(x)$. Then

$$\mathbf{F}_a(x) = f\frac{\alpha(x) - x}{||\alpha(x) - x||}, \text{where } f = \frac{1}{R[t] - r - d} - \frac{1}{R[t] - r}.$$

The scalar f is chosen such that the force will be 0 when the character is positioned on the attraction point. In addition, f will be ∞ when the character touches the boundary of the clearance disk. (However, f will never reach ∞ since we require that the radii of the disks are strictly larger than r.)

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 [7], 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).

4 The Improved Approach

In the previous section, we used a force function $\mathbf{F}_a$ which simultaneously steers the character toward the goal and keeps it inside the corridor. This sometimes results in rather unnatural motions, in particular when characters also have to avoid each other. The cause for this is that due to the choice of the attraction point, the position of the character lies close to the boundary of the clearance disk, and, hence, the force $\mathbf{F}_a$ gets very large.

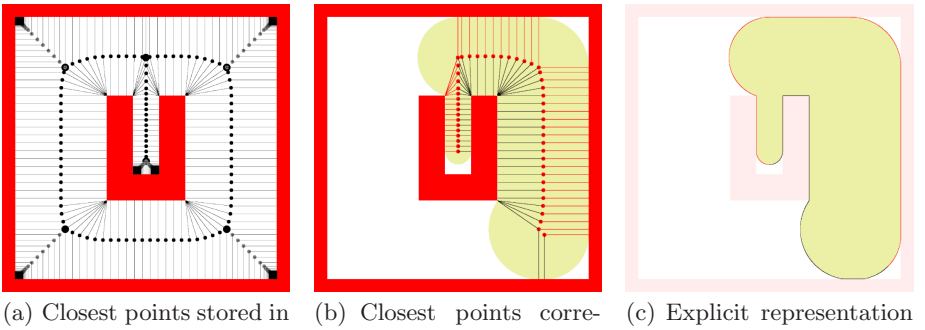
In this section we will show how to avoid this by decoupling $\mathbf{F}_a$ into two forces. The boundary force $\mathbf{F}_b$ will push the character away from the boundary of the corridor. The steering force $\mathbf{F}_s$ will guide the character toward the goal. For the latter we again use an attraction point on a path to the goal. However this path no longer needs to be the same as the backbone path of the corridor. Hence, from now on we refer to this path as the *control path*.

To be able to compute the boundary force we need an explicit representation of the boundary of the corridor.

4.1 Computing an Explicit Corridor Boundary Representation

Up to now we used an *implicit* description of a corridor, i.e. the corridor is retrieved from the map as a sequence of disks. However, such a sequential representation does not allow for easy/efficient computation of a closest point on the boundary which is required for computing the boundary force. Hence, we need an *explicit* description of the corridor's boundary (see Fig. 4(c)).

We can obtain this description by adding information to the corridor map in the preprocessing phase. For each sampled point on the skeleton we compute the set of closest points to the obstacles. By exploiting graphics hardware, we can efficiently compute these closest points. Let B be a sample point on the skeleton. We determine the position of B in the frame buffer. Next we consider the colors



(a) Closest points stored in the corridor map

(b) Closest points corresponding to a corridor

(c) Explicit representation of the corridor's boundary

Fig. 4. Closest points to the obstacles. By concatenating the points with line segments and circular arcs, we obtain an explicit representation of the corridor's boundary.

of the pixels neighboring B. These colors correspond to unique obstacles and the closest points must lie on these obstacles. Computing these points can then be achieved by simple geometric calculations.

Fig. 4(a) shows the corridor map and corresponding closest points of our running example. Each sample point is linked to exactly two closest points, except for the vertices of an edge because they have at least two (and at most four) closest points.

To obtain an explicit description of a corridor's boundary, we need to know for each point B which closest point is on the left side and which one is on the right side with respect to the *local orientation* of the edge at B. This information can easily be obtained by inspecting the location of the pixels in the frame buffer relative to this orientation. An example of a corridor, together with its left and right closest points, is displayed in Fig. 4(b).

From this information we can efficiently compute the closest boundary point $cp(x)$ to any point x in the corridor. First, the sample point B is retrieved whose corresponding left (or right) boundary point is closest to point x. Then the previous and next sample point are extracted along with their corresponding boundary points. In case the three boundary points define a line segment on the outline of the corridor, the closest boundary point $cp(x)$ is computed using simple linear algebra. Otherwise, $cp(x)$ lies on an arc. Let a and b denote the start and the end of the arc, respectively, and $\theta = \arccos(a - B, x - B)$. Then $cp(x) = \mathcal{R}(\theta)\ (a - B)$ where $\mathcal{R}(\theta)$ represents the 2D rotation matrix.

4.2 The Boundary Force

To ensure that the character remains inside the corridor, a repulsive force $\mathbf{F}_b$ from the boundary of the corridor toward the character is applied. Since people prefer to keep a safe distance from walls, streets, buildings, etc. [10, 11], such a force is only exerted if the distance between the character and its corresponding boundary point is below a threshold value. Let d_b be the Euclidean distance between the character's position x and its corresponding closest point $cp(x)$ on the boundary of the corridor. Let r be the radius of the character and let d_{safe} denote the preferred safe distance. Then the force is defined as follows:

$$\mathbf{F}_b = \begin{cases} \mathrm{c}_b \ \dfrac{x - cp(x)}{||x - cp(x)||}, & \text{if } d_b - r < d_{\text{safe}} \\ \\ 0 & \text{otherwise.} \end{cases}$$

The scalar $\mathrm{c}_b = \dfrac{d_{\text{safe}} + r - d_b}{d_b}$ is chosen such that the force will become ∞ when the character and the boundary point touch. By modifying the safe distance d_{safe} a wide variety of behaviors can be achieved. A typical value that is also used in our experiments is to set $d_{\text{safe}} = r$.[1]

[1] Note that the safe distance should be taken into account upon the extraction of a corridor. i.e. $R[t] > r + d_{\text{safe}}$. Otherwise, the character will be continuously pushed from the left to the right side of the corridor and vice versa (d_b will always be less than $r + d_{\text{safe}}$ and hence, a $\mathbf{F}_b$ will be exerted on the character at every time step).

4.3 The Steering Force

The character should also feel the urge to move forward toward its goal position. Thus, at every time step a steering force F_s is needed to guide the character at position x toward an attraction point $\alpha(x)$. The force is defined as

$$\mathbf{F}_s = c_s \frac{\alpha(x) - x}{||\alpha(x) - x||},$$

where c_s specifies the relative strength of the force. This scalar can remain fixed, or it can vary depending on the distance between the character and the attraction point, making the character speed up or slow down. In our experimental setting we used $c_s = 1$.

Having defined the forces that acted upon the character, we calculate its new position by numerically integrating its acceleration and velocity. We use an integration scheme that is quite stable and can deal with stiff differential equations. In our simulations we used Verlet integration with step size $\Delta t = 0.05$ and set the maximum acceleration to $5m/s^2$ in order to keep the error minimal.

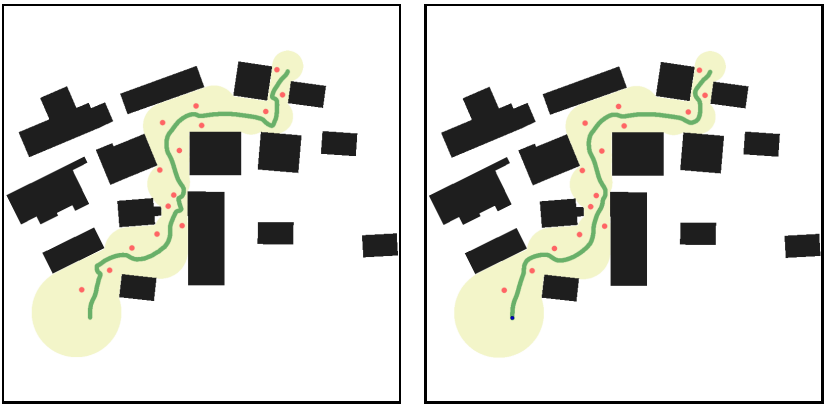
5 Experiments

We have implemented the new method to experimentally validate whether it can generate paths that are smoother than the ones computed by the original CMM. All the simulations were performed on a Pentium IV 2.4 GHz computer with 1GB memory.

The experiments were conducted for the environment depicted in Fig. 2. This is a model of the McKenna MOUT (military operations in urban terrain) training center, hosted at Fort Benning, Georgia, USA. Its corridor map, displayed in Fig. 2(b), was computed in 0.3 seconds (0.05s for the GVD and clearance, and 0.25s for the closest points).

In all of the experiments we used the medial axis as the control network of the new method and defined the attraction points as in the original CMM. Therefore, the two approaches were only differentiated by the forces used to generate the character's motion inside the corridor. In the original CMM the attraction force $\mathbf{F}_a$ makes the character both move forward and stay inside the corridor, whereas in the new approach the two forces ($\mathbf{F}_s + \mathbf{F}_b$) are used to guide the character through the corridor.

To evaluate the quality of the paths when avoiding other characters we populated the environment with a number of static characters (obstacles). We chose static characters because this makes it easier to compare the results. To avoid the static characters, an additional collision response force has to be applied on the character. Thus, both of the methods were enhanced with a simple obstacle avoidance model [7]. At every iteration a repulsive force is exerted from each obstacle $O_i : i \in [1 : n]$ that lies inside the clearance disk corresponding to the attraction point $\alpha(x)$ of the character. This force is monotonically decreasing



(a) Path generated by the original CMM.

(b) Path generated by the revised method.

Fig. 5. Comparing the paths generated by the original and the revised CMM

with the Euclidean distance d_i between the obstacle O_i and the character's position x. Given the radius r of the character and the radius r_i of each obstacle, the total repulsive force $\mathbf{F}_{\text{obs}}$ can be computed as

$$\mathbf{F}_{\text{obs}} = \sum_{i=1}^{n} c_{\text{obs}} \frac{x - O_i}{||x - O_i||}, \text{ where } c_{\text{obs}} = \frac{1}{d_i - r_i - r}.$$

5.1 Results

Fig. 5 shows the paths created by the two methods for an example query ($r = 0.75$, $r_i = 1$). It must be pointed out that some of the artifacts in the resulting paths are due to the simple obstacle avoidance method that was used. A more sophisticated approach would have improved the quality of the paths. However, our goal was to evaluate the two methods regardless of any specific details.

To quantitatively describe the quality of the resulting motions we measured the length as well as the average curvature of the paths. Given any three successive points on a path we approximated the curvature at the middle point as $\kappa = 1/\rho$, where ρ is the radius of the circumscribing circle that passes through each of these three points. By taking the average over all points we were able to detect poor and irregular paths.

Table 1 shows the corresponding statistics for the two paths. As it can be inferred both by the table and Fig. 5(a), the original CMM generates a longer and more erratic path (high-curvature). Due to the way the attraction point is defined (furthest advanced point for which the character is still enclosed by the clearance disk), at every integration step the character lies very close to the boundary of the disk. Thus, an almost infinite attraction force steers the character, rendering impossible to exhibit smooth motions when other obstacles and/or entities are present in the environment. The character has to be very

Table 1. Curvature and path length statistics for the example query, shown in Fig. 5

	Compared Methods	
	Original CMM	**Improved Method**
Path Length	155.12	150.61
Avg. Curvature	0.27	0.13

close to an obstacle to avoid it, and only at the very last moment, it changes its direction (i.e. when the repulsive force from the obstacle becomes very strong). Hence, the resulting motion is far from realistic.

The revised method handles the obstacles more naturally, generating a smoother path (i.e. the path is shorter with less curvature). As it can be observed in Fig. 5(b) the oscillations noted in the original method are reduced. The character is more "relaxed", in the sense that it is not pulled toward the attraction point with an infinite force. Therefore, if an obstacle is encountered the character will start evading soon enough, ensuring a more realistic behavior.

Other queries in the same and other scenes led to similar results. The performance of the two methods was similar (i.e. computing the closest boundary points and decoupling the attraction force into two separate forces influenced the running time marginally). Hence, we can conclude that the revised approach has clear advantages over the original CMM. It is more flexible, it provides better control over the character's motion and consequently leads to more believable paths.

Clearly, the quality of the resulting paths can be further improved by varying the parameters of the revised model and by using a more elaborate approach for collisions avoidance, like Helbing's social force model [12]. For example, more convincing paths can be obtained, as displayed in Fig. 6, as follows. We can increase the safe distance that the character keeps from the boundary of the corridor ($d_{\text{safe}} = 2r$). In addition, we can apply a repulsive force only for obstacles that are perceived within the character's desired direction of motion.

6 Using Alternative Control Paths

Up to now we have used the medial axis as the control network. This works fine in environments in which there are no wide open spaces. However, it will encourage the characters to stay in the middle of the corridors which can be unnatural. So in practice one might want to use alternative control networks and control paths. Such control networks could be indicated manually by a designer to encourage certain character behavior. Also they could be computed automatically based on required behavior. For example we could use the Voronoi-Visibility diagram as introduced in [13] that allows for shortcuts when there is enough clearance. Alternatively we can determine control paths during queries based on perceived danger or interesting places that characters like to visit.

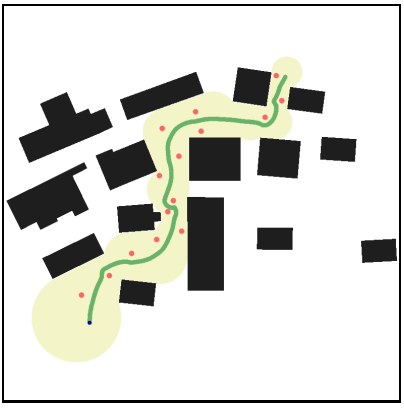


Fig. 6. An alternative path is obtained by increasing the safe distance from the boundary of the corridor

Using alternative control paths is possible but leads to a number of complications. First of all, given such a control path we need to compute the corresponding corridor. The easiest way to achieve this is to retract the control path onto the medial axis [14]. Using the boundary representation described in Section 4 this can be done efficiently.

Secondly, we need a method to choose the location of the attraction point on the control path. The method described above, in which we pick the furthest point along the control path for which the character still lies within the clearance disk, will not be suited anymore when the control path passes close to obstacles. Different options are possible here. We can use an attraction point that moves with constant speed (as long as the character does not lag too far behind). We can also use an attraction point at a particular distance from the character (that can vary over the control path and will determine how closely the control path must be followed). Or we can pick the attraction point based on visibility, although such calculations are relatively expensive. In a future paper we will explore these possibilities further.

7 Conclusions

In this paper we have presented an improved version of the Corridor Map Method. The method can be used to plan in real time natural paths for a large number of characters in complicated environments. It is relatively easy to implement and is flexible enough to incorporate many additional constraints on the resulting paths.

We are currently investigating the effect of using alternative control paths on the behavior of the characters. Also we are studying improved local force models that create even better paths in environments with many moving characters. We also want to incorporate the notions of dangerous and interesting regions and we

want to incorporate small groups of moving characters that stick together. This all should lead to very efficient and high-quality path planning for individuals, groups and whole crowds of computer-controlled characters.

Acknowledgments

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. DeLoura, M.: Game Programming Gems 1. Charles River Media, Inc. (2000)
5. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1994)
6. Geraerts, R., Overmars, M.: Creating high-quality paths for motion planning. International Journal of Robotics Research 26, 845–863 (2007)
7. 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)
8. Geraerts, R., Overmars, M.: Enhancing corridor maps for real-time path planning in virtual environments. In: Computer Animation and Social Agents (2008)
9. Hoff, K., Culver, T., Keyser, J., Lin, M., Manocha, D.: Fast computation of generalized voronoi diagrams using graphics hardware. In: International Conference on Computer Graphics and Interactive Techniques, pp. 277–286 (1999)
10. Stucki, P.: Obstacles in pedestrian simulations. Master's thesis, Swiss Federal Institute of Technology ETH (September 2003)
11. Transportation Research Board, National Research Council Washington, D.C: Highway Capacity Manual (2000)
12. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Physical Review 51, 4282–4287 (1995)
13. Wein, R., Berg, J., Halperin, D.: The Visibility-Voronoi complex and its applications. In: Annual Symposium on Computational Geometry, pp. 63–72 (2005)
14. Ó'Dúnlaing, C., Sharir, M., Yap, C.: Retraction: A new approach to motion planning. In: ACM Symposium on Theory of Computing, pp. 207–220 (1983)