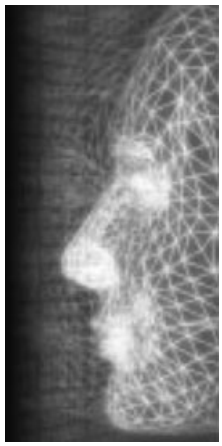
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2007; **18**: 107–119
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/cav.166

The corridor map method: a general framework for real-time high-quality path planning



By Roland Geraerts and Mark H. Overmars*

In many virtual environment applications, paths have to be planned for characters to traverse from a start to a goal position in the virtual world while avoiding obstacles. Contemporary applications require a path planner that is fast (to ensure real-time interaction with the environment) and flexible (to avoid local hazards such as small and dynamic obstacles). In addition, paths need to be smooth and short to ensure natural looking motions.

Current path planning techniques do not obey these criteria simultaneously. For example, A* approaches generate unnatural looking paths, potential field-based methods are too slow, and sampling-based path planning techniques are inflexible. We propose a new technique, the Corridor Map Method (CMM), which satisfies all the criteria. In an off-line construction phase, the CMM creates a system of collision-free corridors for the static obstacles in an environment. In the query phase, paths can be planned inside the corridors for different types of characters while avoiding dynamic obstacles. Experiments show that high-quality paths for single characters or groups of characters can be obtained in real-time.

Received: 10 January 2007; Revised: 31 January 2007; Accepted: 1 February 2007

KEY WORDS: path planning; corridor map method; high-quality paths

Introduction

Path planning is one of the fundamental problems in interactive virtual worlds, such as games. The path planning problem can be defined as finding a path between a start and goal position of a character in an environment with obstacles. In the past 15 years, efficient algorithms have been devised to tackle this problem. They are successfully applied in fields, such as mobile robots, manipulation planning, CAD systems, virtual environments, protein folding, and human robot planning. See the books of Choset *et al.*,[1] Latombe[2], and LaValle[3] for an extensive overview.

Many algorithms require that the complete environment is known beforehand. However, the environment frequently contains dynamic obstacles or other moving characters which can block a computed path. As a result, adaptations of the algorithms are required to avoid the new obstacles in real-time. Even if all the information is available, for example in static virtual environments, methods can have difficulties dealing with the growing sizes of contemporary virtual environments. Often, only the large obstacles are taken into account to save memory and to lower the CPU load. However, also the small obstacles have to be avoided in real-time.

An important question is how long the computation of a path may take to ensure real-time behavior. In a virtual environment, such as a game, very little processor time is scheduled for the path planner. Especially when many paths have to be planned simultaneously, only one (of a few) millisecond per second CPU time per character is allowed. Larger running times will lead to stalls in interactive environments. In conclusion, interactive environments require a path planner that is very fast and flexible.

In the game development community, a common way to plan a path is to use an A* algorithm on a

*Correspondence to: R. Geraerts, Center for Advanced Gaming and Simulation (AGS), Institute of Information and Computing Sciences, Utrecht University, 3508 TA Utrecht, The Netherlands. Email: roland@cs.uu.nl

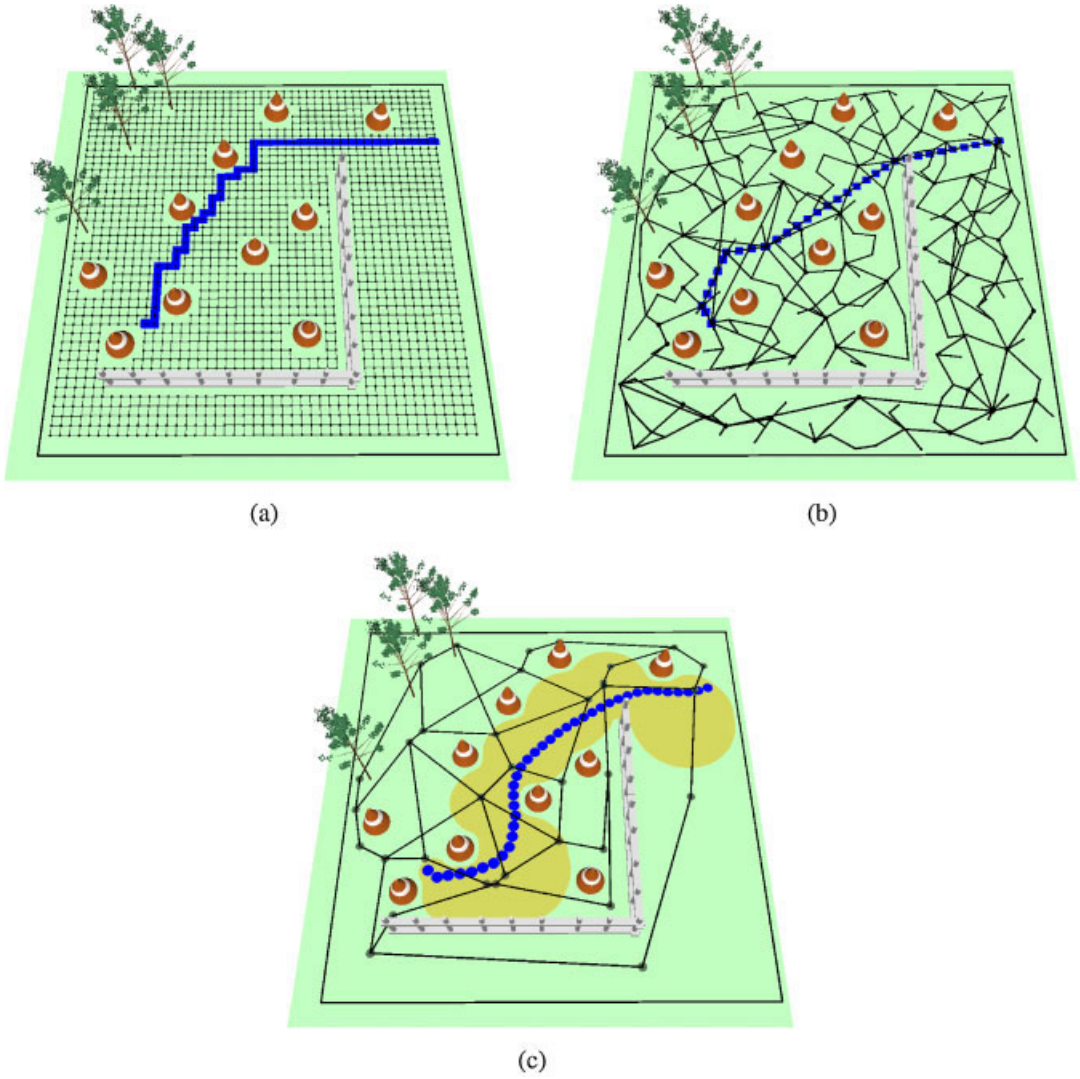


Figure 1. Three methods for path finding. The A algorithm finds the shortest path in the displayed grid, consisting of 1792 nodes and 3321 edges. The PRM-graph is almost six times as small. The CMM-graph is the smallest one containing 44 nodes and 50 edges. (a) A*, (b) PRM, (c) CMM.*

low-resolution grid (see Figure 1a). This search algorithm is popular because it always finds the shortest path in a grid, if one exists. However, as virtual worlds are becoming very large, storing the grid and running the algorithm may consume a huge amount of memory which is not always available, in particular on systems with constrained memory such as mobile systems. In addition, the algorithm may consume too much processor time when many paths have to be planned simultaneously. Paths resulting from A* algorithms tend to have little clearance to obstacles and can be aesthetically unpleasant, so care must be taken to smooth them. In conclusion, A* algorithms have serious drawbacks when dealing with large interactive environments.

In the robotics community, much research has been conducted on the path planning problem. About 20 years ago, flexible planners, such as *Potential Field* methods, were introduced.[4–6] A Potential Field method directs the motion of the character (robot) through an artificial potential field which is defined by a function over the free configuration space C_{free} (that is, the space of all the possible placements for the character in the environment). The character is pulled toward the goal position as it generates a strong attractive force. In contrast, the obstacles generate a repulsive force to keep the character from

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

colliding with them. The path from the start to the goal can be found by following the direction of the steepest descent of the potential toward the goal. While this method has some flexibility to avoid local hazards (such as small obstacles and other moving objects), it is not useful for path planning in interactive virtual environments as it takes too much time to create the path. In addition, the path will not always be found because the character often ends up in a local minimum of the potential.

The Probabilistic Roadmap Method (PRM), developed in the 1990s,[7–9] does not suffer from the local minima problem. This method consists of two phases. In the construction phase, a roadmap is created that captures the connectivity of $\mathcal{C}_{\text{free}}$ with a set of one-dimensional (1D) curves. In the query phase, the start and goal positions are connected to the graph, and the path is obtained by running Dijkstra's shortest path algorithm.

While the PRM has been successfully applied to a broad range of problems, the method generates ugly paths. That is, the paths are only piecewise linear, they have many redundant motions, and they have little clearance to the obstacles, resulting in unnatural looking motions (see Figure 1b). While techniques exist for optimizing the paths,[10–12] they are too slow to be applied in the query phase in real-time applications.

By shifting the optimization process to the off-line construction phase, high-quality paths can be computed in a small amount of time. In Reference [13], we proposed a method that creates high-quality graphs from which relatively short paths and paths with a large amount of clearance can be extracted. While the method may be fast enough for an environment with one character, the method will be still too slow for environments with many characters in the query phase.

A disadvantage of these roadmap-based methods is that they output a fixed path in response to a query. This leads to predictable motions and lacks flexibility when the environment or character changes.

Recently, the concept of path planning inside *corridors* has been introduced.[14,15] By using corridors, the advantages of the techniques described above are combined. That is, global motions are directed by a high-quality roadmap, and local motions are controlled by potential fields inside corridors, providing local flexibility of the path (see Figure 1c). In Reference [4], corridors have been exploited to find paths for coherent groups of characters. Also quantitative measures for the quality of corridors have been devised.[15]

In this paper, we extend and generalize their results by proposing a general framework, called the Corridor Map Method (CMM). We show how to avoid dynamic obstacles and how to create short paths. Then we conduct experiments with two-dimensional (2D) problems and conclude that the framework is capable of creating smooth, short paths for characters avoiding dynamic obstacles in real-time, that is in less than 1 millisecond CPU time per second traversed time of the character.

Corridor Map Method

The CMM creates a system of collision-free corridors for the static obstacles in an environment. Paths can be planned inside the corridors for different types of characters while satisfying additional constraints such as avoiding dynamic obstacles. We assume that the character can be modeled by a ball with radius r.

The CMM consists of an off-line construction phase and a one-line query phase. In the construction phase, a roadmap graph $G = (V, E)$ is built which serves as a skeleton for the corridors (see Figure 2a). Each vertex $v \in V$ corresponds to a collision-free point in a d-dimensional environment (d is typically 2 or 3) and each edge $\epsilon \in E$ corresponds to a local path Π_ϵ. The path Π is defined as follows:

Definition 1 (Path). *A path Π for a point in a d-dimensional environment is a continuous map $\Pi \in [0, 1] \to \mathbb{R}^d$ such that $\forall t \in [0, 1] : \Pi[t] \in \mathcal{C}_{\text{free}}$.*

With each point $\Pi_\epsilon[t]$ on local path Π_ϵ, we associate the radius $R[t]$ of the largest empty ball (in the environment) centered at $\Pi_\epsilon[t]$. This clearance information and the graph are now used to define the *corridor map* (see Figure 2b):

Definition 2 (Corridor map). *The corridor map is a graph $G = (V, E)$ with clearance information. That is, each edge $\epsilon \in E$ encodes a local path Π_ϵ together with the radii R of the corresponding largest empty balls in the environment.*

In the query phase, we have to find a path for a character which connects the start position to the goal position. For now, we assume that these positions are vertices $v', v'' \in V$. By running Dijkstra's shortest path algorithm (while discarding edges for which $\exists t : R[t] < r$), we extract the *backbone path* (if one exists) from G.

Definition 3 (Backbone path). *Let $\epsilon_1 \ldots \epsilon_n$ be the sequence of edges extracted from G that connects v' with v''. The backbone path $B[t]$ is then defined as $\Pi_{\epsilon_1} \oplus \ldots \oplus \Pi_{\epsilon_n}$ where the operator $\oplus$ concatenates the local paths Π_{ϵ_i}.*

The backbone path, together with the clearance information defines a *corridor* (see Figure 2c):

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

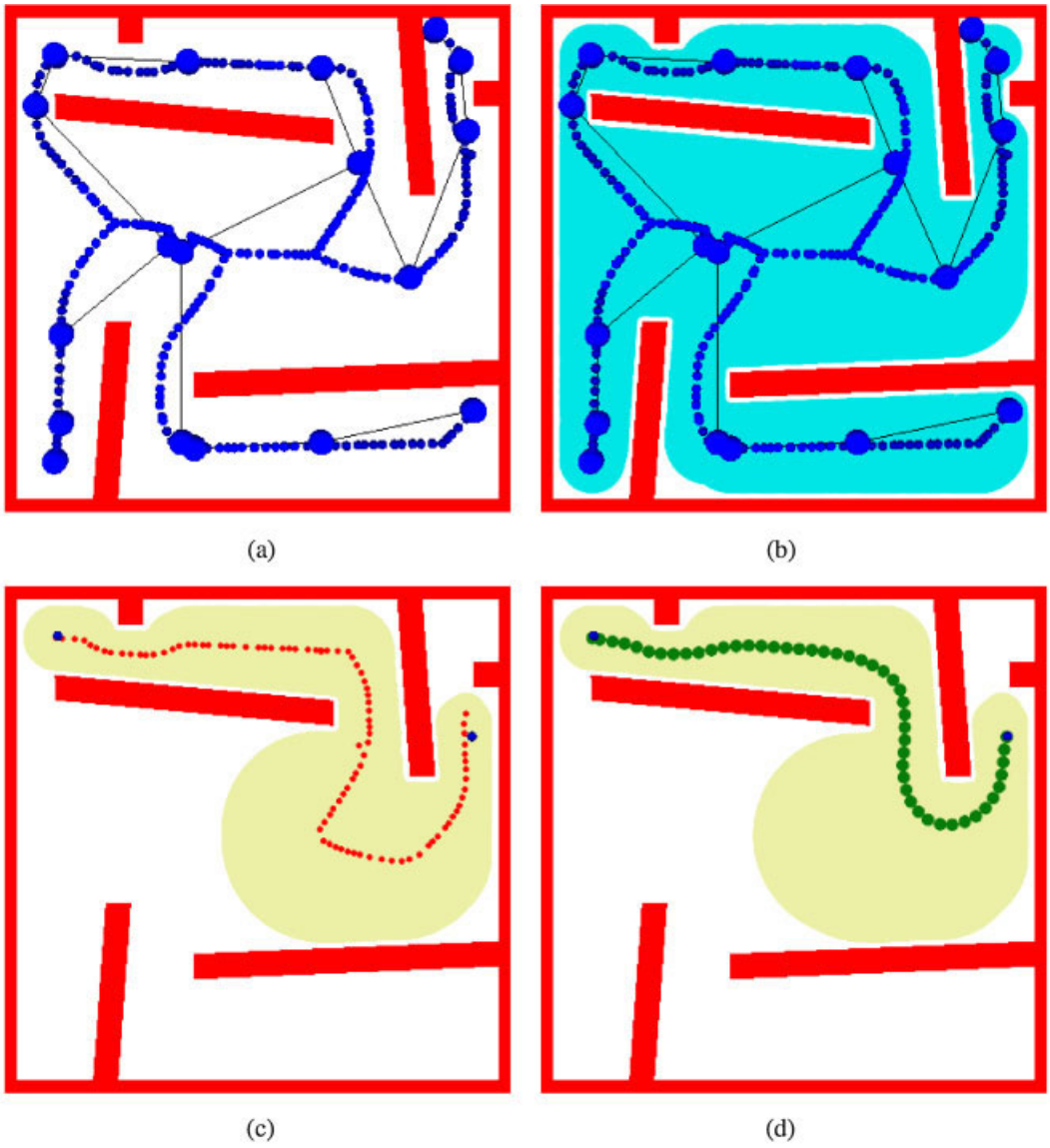


Figure 2. The construction phase (top) and the query phase (bottom) of the Corridor Map Method. (a) Graph, (b) Corridor map, (c) Corridor and backbone path, (d) Final path.

Definition 4 (Corridor). *A corridor* $C = (B[t], R[t])$ *is defined as the union of the set of balls with radii* $R[t]$ *whose center points lie along its backbone path* $B[t]$.

If the start position s or goal position g is not equal to one of the vertices, we have to extend the corridor such that they are included (see Figure 3). Let s' and g' be their closest points on local paths Π_{ϵ_s} and Π_{ϵ_g} whose balls

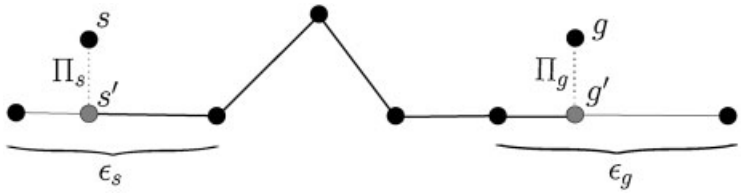


Figure 3. Extending the corridor to include the start and goal positions.

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

include s and g, respectively. Edges ϵ_s and ϵ_g are split such that they include vertices s' and g', respectively. Finally, let Π_s be the straight-line local path between s and s' and Π_g be the straight-line local path between g' to g. Then, the backbone path is defined as path Π_s, concatenated with the shortest path between s' and g', and path Π_g. The radii corresponding to the positions p on Π_s (Π_g) are equal to the clearance corresponding to vertex s' (g') minus the Euclidean distance between s' (g') and p.

Now that we have defined the corridor, which 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 from the start to the goal. The attraction point is defined such that making a step toward this point leads the character toward the goal. In addition, the ball (with radius $R[t]$) corresponding to $\alpha(x)$ encloses the character, ensuring a collision-free motion. If $R[t] \leq r$, then there exists no attraction point, and, hence, no path.

Definition 5 (Attraction point)**.** *Let x be the current position of the character with radius r. The attraction point $\alpha(x)$ for the character at position 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 attraction point attracts the character with force $\mathbf{F}_0$. Let d be the Euclidean distance between the character's position x and the attraction point $\alpha(x)$. Then $\mathbf{F}_0$ is defined as

$$\mathbf{F}_0 = f\frac{\alpha(x) - x}{||\alpha(x) - x||}, \quad \text{where } f = \frac{1}{R[t] - r - d} - \frac{1}{R[t] - r}$$

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 ball. (However, f will never reach ∞ since we require that the radii of the balls are strictly larger than r).

Local hazards (such as small obstacles or other characters) can be avoided by adding repulsive forces to $\mathbf{F}_0$ toward the hazards. Hence, the final force $\mathbf{F}$ is dependent on the problem to be solved. We will show some choices in the following section.

The final path Π is obtained by integration over time while updating the velocity, position, and attraction point of the character. In each iteration, we update the attraction point on the backbone path based on position x of the character. Now we have all the information needed to compute the force $\mathbf{F}$. By integrating $\mathbf{F}$, we compute the new velocity vector for x. In addition, by integrating the velocity vector, we compute the new position for the character. We continue moving the character until the character has reached the goal.

By using this time integration scheme, a *smooth* path is obtained.

Theorem 1 (C^1 continuity of the path)**.** *The CMM generates a path Π that is smooth, that is C^1 continuous.*

Proof. The path Π is obtained by integrating the force $\mathbf{F}$ two times, which adds two degrees of continuity to the path followed by the attraction point. Even though this path can be discontinuous, it can be easily shown that double integration leads to C^1 continuity. To prove that $\mathbf{F}$ can indeed be integrated, we have to show that the denominators in $\mathbf{F}$ are larger than 0: Since the attraction point $\alpha(x)$ is defined as the furthest point on the backbone path, the point lies always 'in front of' the character's position x (except for the goal position), and, hence, the term $||\alpha(x) - x|| > 0$. In addition, the term $R[t] - r - d > 0$ because $R[t] > r$. Since these two terms stay positive, $\mathbf{F}$ can be integrated (two times), resulting in a path Π being C^1 continuous. ■

As an example, consider Figure 2 which shows the stages of the CMM applied to a simple planar environment. For this environment, an input graph was created. Its nodes were sampled on the medial axis to ensure a locally maximum clearance of the nodes. Its edges (black lines) were retracted to the medial axis to provide high-clearance local paths (small discs). The graph, together with the clearance information, forms the corridor map which is displayed in the second picture. The covered area of the map is visualized in a light color. The next picture shows the corridor and its backbone path corresponding to a start and goal position of the query. The final path, displayed in the fourth picture, was obtained by applying the procedure described above.

Specific Choices

An important influence on the quality of the corridor map, and, hence, the quality of the resulting paths, is the quality of the input graph. In Reference [13], we proposed the *enhanced Reachability Roadmap Method*, which is a technique that creates high-quality graphs satisfying the following four properties:

1. The graph is *resolution complete*. This mean that a valid query (which consists of a start and a goal position) can always be connected to the graph. If there exists a

DOI: 10.1002/cav

path between the start and goal, then it can always be found (at a given resolution).
2. The graph is *small*. A small graph ensures low query times and low memory consumption. In addition, when a graph must obey other criteria, a small graph eases manual tuning.
3. The graph contains *useful cycles*. These cycles provide short paths and alternative routes which allow for variation in the (global) routes that characters take.
4. The graph provides *high-clearance local paths*. As the local paths lie on the medial axis, each point on the corridor will have a locally maximum clearance. This will provide the most freedom for the character to move.

In the remainder of this paper, we will use this technique for creating the input graphs.

We have seen that moving inside a corridor (instead of moving along a path) provides enough freedom to obtain a smooth path. Now we will describe how a corridor can be used to avoid dynamic obstacles and to create shorter paths.

Avoiding Dynamic Obstacles

Dynamic obstacles are the obstacles in the environment that are not present (or suppressed) when the corridor map is created. There are two ways to deal with avoiding these obstacles. We can update the force $\mathbf{F}$ (by adding a repulsive force toward the obstacles) or we can change the corridor itself. We assume that the radii of the obstacles are known.

Adding Forces. Our goal is to guide the character around all the dynamic obstacles inside the corridor. To ensure that the character does not collide with the obstacles, repulsive forces are applied. Such a force is only applied if both the character and the obstacle are located in the ball corresponding to the attraction point $\alpha(x)$. For each obstacle $O_i : i \in [1 : n]$, we compute a repulsive force $\mathbf{F}_i$. Let d_i be the Euclidean distance between the center of the character at position x and the center of obstacle i with radius r_i, r be the radius of the character, and $k : k > 0$ be a constant. Then $\mathbf{F}_i$ is defined as

$$\mathbf{F}_i = f\frac{x - O_i}{||x - O_i||}, \quad \text{where } f = \frac{k}{d_i - r_i - r}$$

The scalar f is chosen such that the force will be ∞ when the character and obstacle touch. The larger the distance between them, the lower the force will be. The constant k is used to increase or decrease the influence of the repulsive forces on the character.

The final force $\mathbf{F}$ can now be calculated by adding the attractive force $\mathbf{F}_0$ and repulsive forces $\mathbf{F}_i$, that is,

$$\mathbf{F} = \mathbf{F}_0 + \ldots + \mathbf{F}_n$$

As an example, consider Figure 4(a). It shows our running example, but now five small dynamic obstacles have been added. The final path is obtained after the force function has been extended. The figure shows that the path has only changed locally. While this method is flexible, it is hard to control the 'shape' of the path. In addition, future changes of the path (i.e., shortening the path) are hard since such change has to operate on a path

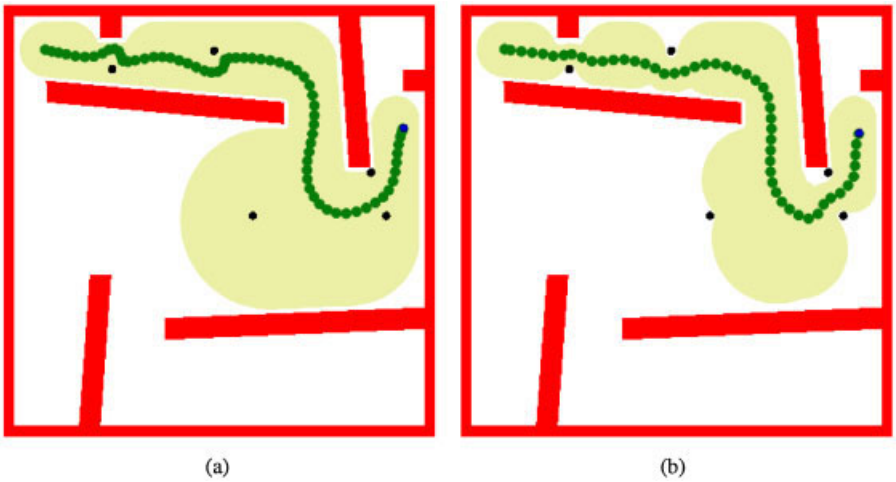


Figure 4. Two techniques for obstacle avoidance. The left picture shows an unchanged corridor that includes five small obstacles. The right picture shows the updated corridor, swaying around the five obstacles. (a) Extending the force function, (b) Updating the corridor.

DOI: 10.1002/cav

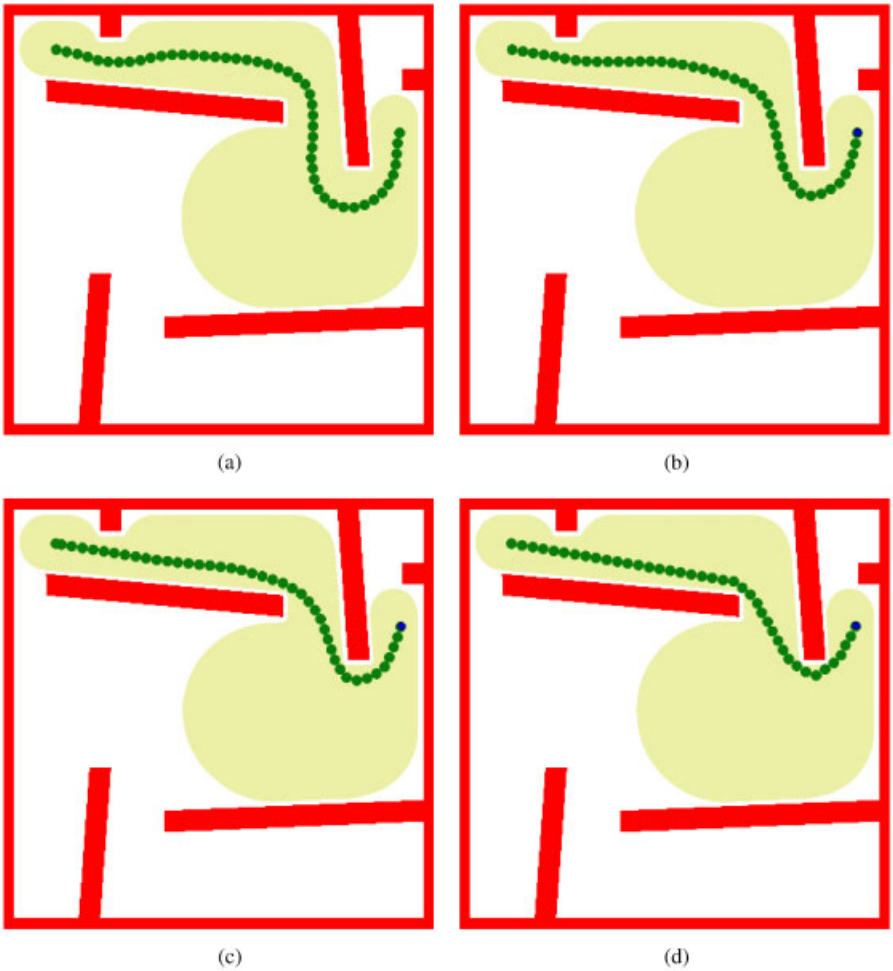


Figure 5. Using the corridor to create shorter paths. Shorter paths are obtained by moving the attraction point $\alpha(x)$ along the backbone path toward the goal. (a) $\alpha(x, 0.00)$, (b) $\alpha(x, 0.05)$, (c) $\alpha(x, 0.10)$, (d) $\alpha(x, 0.25)$.

instead of a corridor. By creating a sub-corridor inside the corridor which excludes the dynamic obstacles, we obtain more freedom.

Creating a Sub-Corridor. Our goal is to create a sub-corridor $C' = (B'[t], R'[t])$ which lies inside the original corridor and is absent from the dynamic obstacles $O_i : i \in [1 : n]$. In the following procedure, we initially set $B'[t]$ and $R'[t]$ to $B[t]$ and $R[t]$, respectively, where $t : t \in [0 : 1]$ is the time index. Then we move the backbone path and update the corresponding radii of the balls, as follows. Let $d_i[t]$ be the Euclidean distance between the center of the ball positioned at $B[t]$ and center of obstacle i, that is, $d_i[t] = ||B[t] - O_i||$. Only if an obstacle O_i is in this ball, that is, $d_i[t] < R[t]$, the sub-corridor is modified. The point $B[t]$ will be moved away in a straight line from obstacle O_i. The distance traveled by this point, dist, equals to

$$\text{dist} = \frac{R[t] - d_i[t]}{2}$$

Hence, the position of ball $B'[t]$ equals to

$$B'[t] = B[t] + \text{dist} * \frac{B[t] - O_i}{d_i[t]}$$

and the radius $R'[t]$ of the ball equals to

$$R'[t] = R[t] - \text{dist}$$

We refer the reader to Figure 4(b) for an example of the method. The resulting smooth path lies in the updated

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

corridor, being absent of the five dynamic obstacles. As a new corridor has been computed, the dynamic obstacles can be discarded when the path is processed even further, for example, when the path is shortened.

Creating Shorter Paths

In this section, our goal is to use the corridor to create shorter paths. In our standard framework, the character at position x is attracted toward the attraction point $\alpha(x)$, corresponding to point $B[t]$ on the backbone path. Shortcuts in the path can be made by creating a second *valid* attraction point $\alpha(x, \Delta t)$, corresponding to point $B[t + \Delta t] : \Delta t \geq 0 \wedge t + \Delta t \leq 1$, to which the character is attracted. We say that an attraction point $\alpha(x, \Delta t)$ is *valid* if the character, moved in a straight-line from its current position x to the attraction point $\alpha(x, \Delta t)$, stays inside the corridor.

Suppose now that we want to create a path using a certain value of Δt. If the attraction point $\alpha(x, \Delta t)$ is not valid, we have to lower Δt. We determine the highest Δt by decreasing Δt with small steps until $\alpha(x, \Delta t)$ is valid.

Given Δt, the resulting force $\mathbf{F}_s$, which has to be added to force $\mathbf{F}$, equals to

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

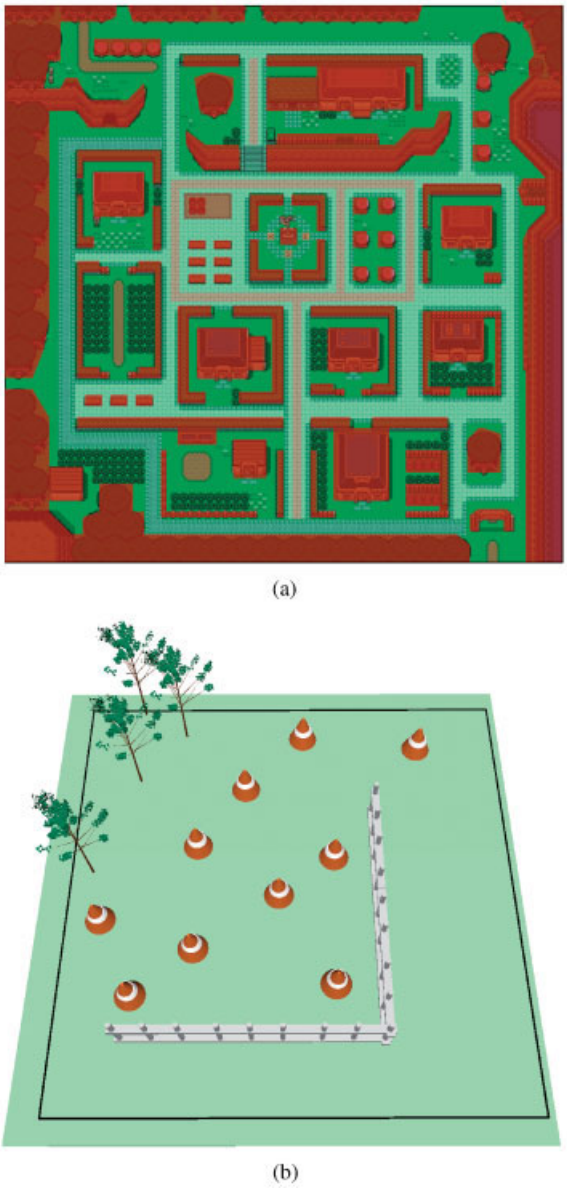
The influence of using different values for Δt on the path can be examined in Figure 5. The first picture shows the path obtained by only attracting the character to its attraction point $\alpha(x)$. The other pictures show the resulting paths for different values of Δt. Indeed, shorter paths are obtained for larger values of Δt at the cost of increased computation time.

Experiments

In this section, we test the CMM on two different environments. We will experimentally check whether the CMM can produce high-quality paths in real-time, that is, in less than 1 millisecond CPU time per second traversed time.

Experimental Setup

We integrated the techniques in a single motion planning system called System for Advanced Motion PLanning Experiments (SAMPLE), which we implemented in Visual C++ under Windows XP. All the experiments were run on a 2.66 GHZ Pentium 4 processor with 1 GB memory. We used Solid for collision checking.[16]



(a)

(b)

Figure 6. The two-test environments. (a) Game, (b) Field.

We conducted experiments with the environments depicted in Figure 6. The graphs, together with their corridor maps are displayed in Figure 7. Since we focused on obtaining low query times and high-quality paths, much time for creating these graphs could be spent off-line by the *enhanced Reachability Roadmap Method* from [13]. This method discretized the environments with 100×100 cells. The character is modeled as a small disc. The environments have the following properties:

DOI: 10.1002/cav

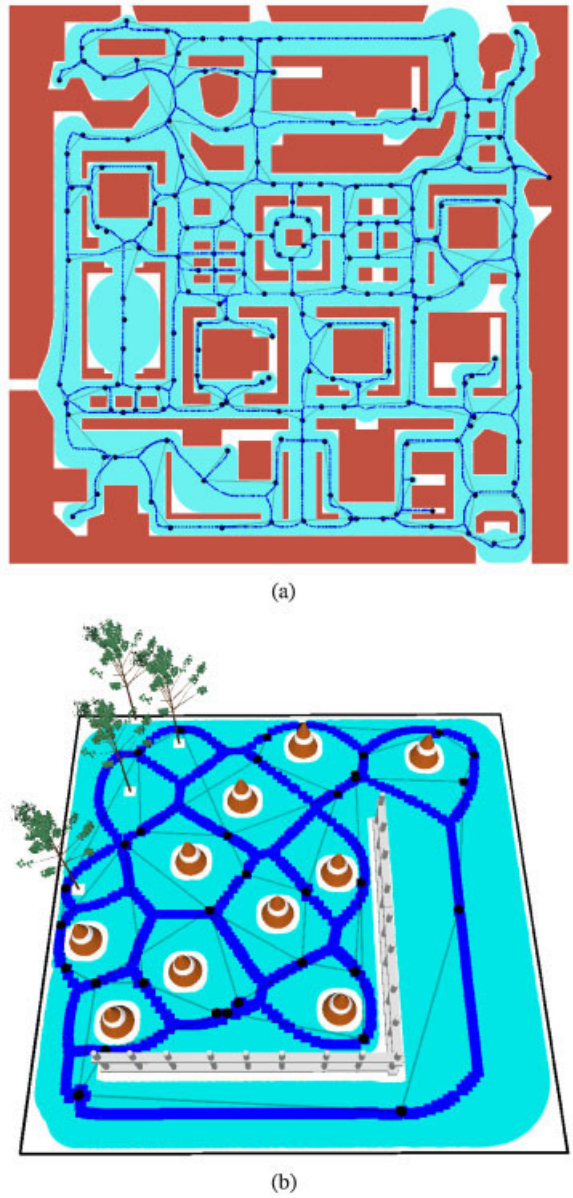


Figure 7. The input graphs and corresponding corridor maps for two-test environments. (a) Game, (b) Field.

Game. This 2D maze-like environment features a level taken from the game *Zelda*™ created by Nintendo. The obstacles are composed of 306 (dark-colored) triangles. The input graph was created in 150 seconds. The graph contains many useful cycles, providing short paths and alternative routes. Since its local paths lie on the medial axis, the corridor map covers a large portion of the free space, providing the character much freedom to move.

Field. This three-dimensional (3D) environment contains 10 cones, 2 fences, and 4 trees. Together, they are composed of 16 000 triangles. Since this environment is much smaller than the Game environment, much less time was needed to create the input graph (i.e., 20 seconds). There are many alternative routes. Again, the corridor map covers a large portion of the free space.

We performed three batches of experiments. In the first batch, we found paths for 100 random queries to get an idea of how long a query takes to compute. In the second batch, we defined one query. We added up to 10 dynamic obstacles close to the backbone path and observed the changes in running times. In the third batch, we studied the effect of choosing different values of Δt on the running time versus path length.

For each experiment, we provide the integral running times (in ms) of the query phase. That is, connecting the query/queries to the roadmap, computing the corridor(s), and extracting the path(s). Often, only the computation for a part of the path is required/desired. Hence, we also provide the CPU time (in millisecond) required for 1 second traversed time of the path. However, statements based on this statistic are rather subjective, that is, increasing the character's speed implies a lower CPU load while decreasing its speed implies a higher CPU load.

Experimental Results

Measuring the Performance of Creating Smooth Paths. For the Game environment, 100 queries were computed in 274 milliseconds. The total traversed time for all the resulting paths was 1051 seconds. Hence, on average, 0.26 millisecond per second traversed time was required which implies a CPU load of 0.026% per character. The average traversed speed for the character was 6.3 cells per second. For the Field environment, the queries were computed in 74 millisecond. The total traversed time was 285 seconds. Hence, on average, 0.26 millisecond per second traversed time was required. Again, this implies a CPU load of 0.026% per character. The average traversed speed for the character was 29 cells per second. These results make clear that the method is very efficient in producing smooth paths.

Dynamic Obstacles. We considered two methods for obstacle avoidance. The first method extended the force function while the second method updated the corridor. Figures 8 and 9 display the corridors and paths obtained while avoiding 10 obstacles. The query times for the Game environment ranged between 11 and

DOI: 10.1002/cav

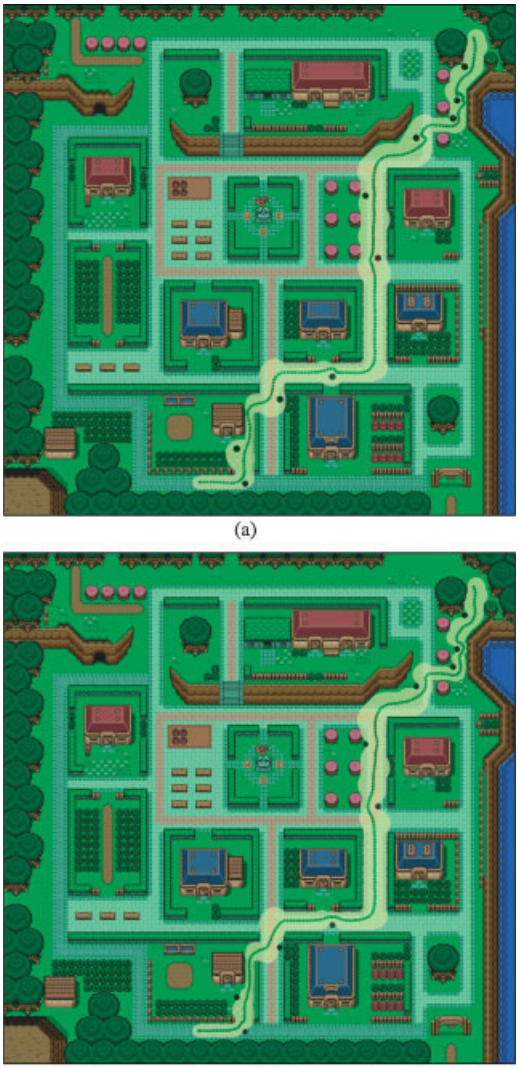


Figure 8. Obstacle avoidance in the Game environment. Ten obstacles are avoided. (a) Extending the force function, (b) Updating the corridor.

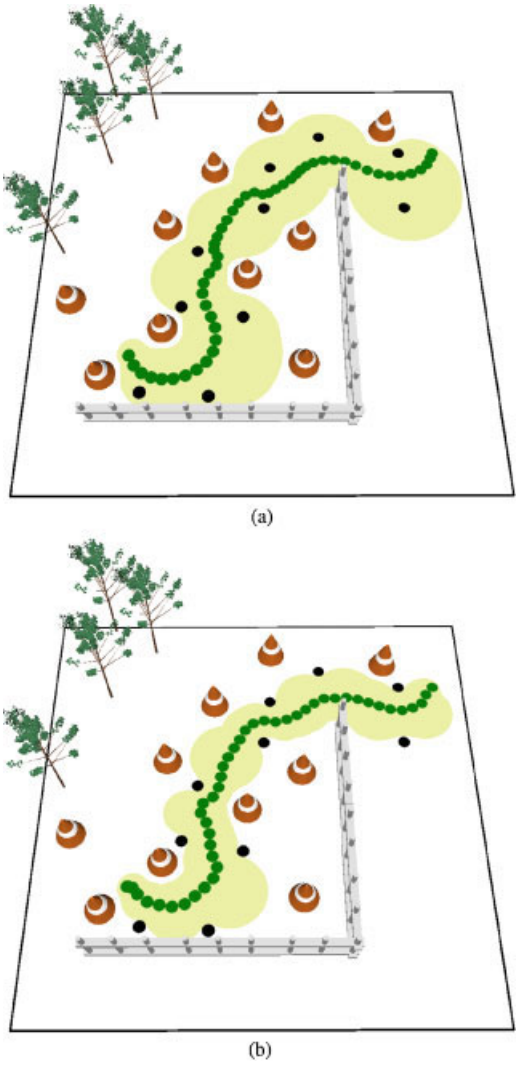


Figure 9. Obstacle avoidance in the Field environment. Ten obstacles are avoided. (a) Extending the force function, (b) Updating the corridor.

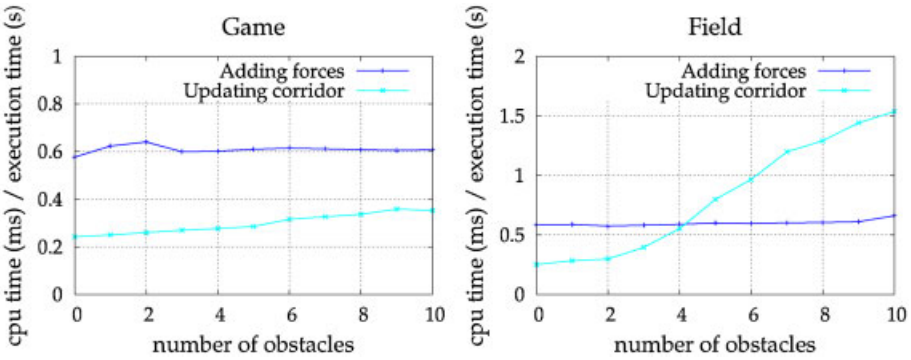


Figure 10. The performance of the two techniques for avoiding obstacles.

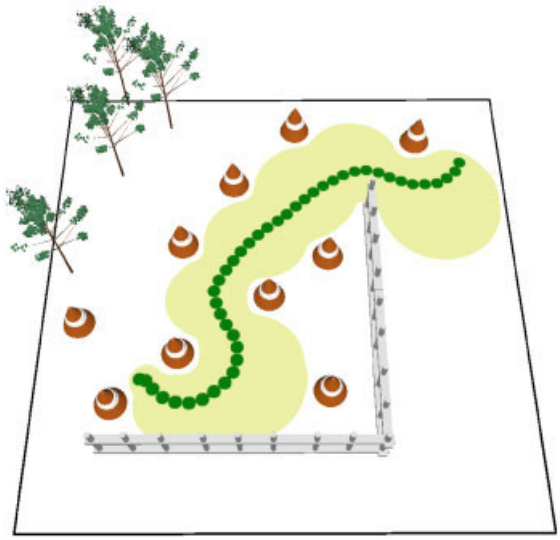
Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

(a)

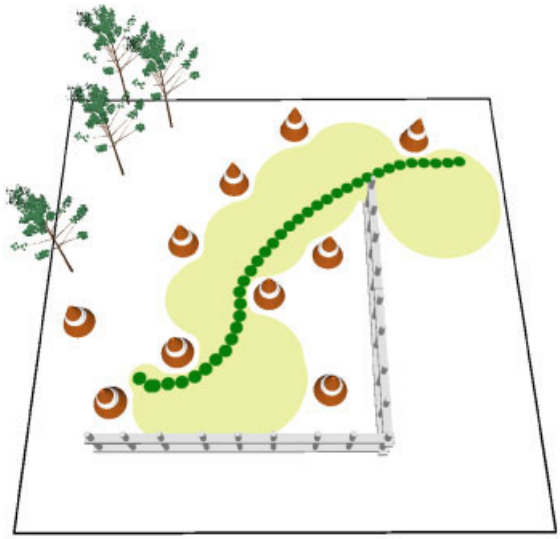
(b)

Figure 11. Using the corridor to create shorter paths in the Game environment. Shorter paths are obtained by moving the attraction point $\alpha(x)$ along the backbone path toward the goal. (a) $\alpha(x, 0.0)$, (b) $\alpha(x, 0.1)$.

12 milliseconds for the first method, and between 5 and 8 milliseconds for the second method. The query times for the Field environment ranged between 25 and 32 milliseconds, and between 10 and 66 milliseconds for the two methods, respectively. Figure 10 shows the results. The figure makes clear that both techniques are efficient.



(a)



(b)

Figure 12. Using the corridor to create shorter paths in the Field environment. Shorter paths are obtained by moving the attraction point $\alpha(x)$ along the backbone path toward the goal. (a) $\alpha(x, 0.0)$, (b) $\alpha(x, 0.1)$.

Short Paths. Shorter paths were created by adding a force toward a new attraction point $\alpha(x, \Delta t)$ placed between $\alpha(x)$ and the goal. Figures 11(a) and 12(a) show the paths which have not been shortened, that is no additional attraction point was used. Figures 11(b) and 12(b) show the paths obtained by using an additional force attracting the character toward $\alpha(x, 0.1)$. We performed preliminary experiments to study the relation between Δt and the path length. The results are shown

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav

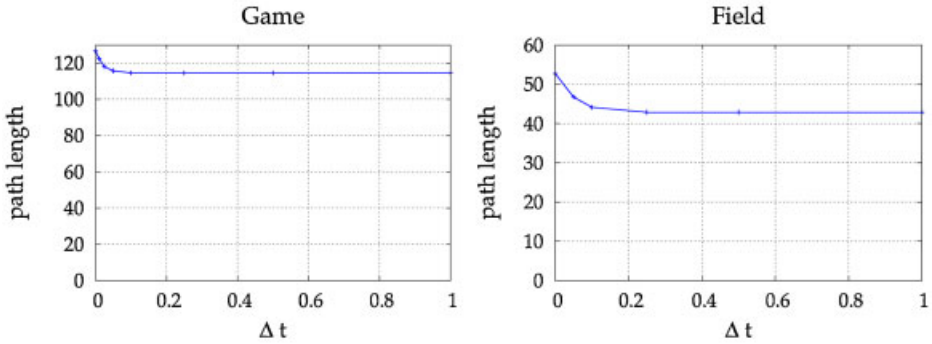


Figure 13. The relation between the value of Δt and the path length.

in Figure 13. We can conclude from these curved paths, that short paths can be obtained even for small values of Δt, that is, $\Delta t = 0.1$. However, the running times were reasonably large due to the relatively expensive computation of the second attraction point. For $\Delta t = 0.1$ the running times were about two times as large as the running times corresponding to not using the new attraction point. For $\Delta t = 1.0$ they were about 12 times as large. In conclusion, the technique is fast enough for a real-time performance for small values of Δt. For larger values, however, the method might not be fast enough. We are currently investigating how to enhance the technique.

Conclusions and Future Work

We presented a new framework, called the CMM, which can be used for path planning in real-time interactive virtual worlds. The CMM directs the global motions by a high-quality roadmap. Local motions are controlled by potential fields inside a corridor, leading to smooth and short paths. In addition, the corridor provides enough flexibility when dynamic obstacles (or other moving characters) have to be avoided. Experiments showed that such motions can be computed in real-time.

The input graphs for the CMM were created by our Reachability Roadmap Method. Also other methods such as [15,17] could be used to create these graphs. To ensure a high quality of the graphs, reasonably high computation times were required in the construction phase. We think that these can be improved dramatically by incorporating learning techniques. Nevertheless, these graphs ensured fast running times in the query phase.

The CMM can also be used for guiding the motions of a group of characters. In addition, tactical information could be incorporated in the corridor map to provide clever routes for the characters. While we focused on 2D problems, the framework is also applicable to higher-dimensional problems. In future work, we will extend the experiments with 3D problems. In addition, we will study how to create alternative paths.

ACKNOWLEDGEMENTS

Part of this research has been funded by the Dutch BSIK/BRICKS Project.

References

1. Choset H, Lynch K, Hutchinson S, *et al. Principles of Robot Motion: Theory, Algorithms, and Implementations* (1st edn). MIT Press: Cambridge, MA, 2005.
2. Latombe JC. *Robot Motion Planning*. Kluwer: Norwell, Mass, 1991.
3. LaValle S. *Planning Algorithms*. http://planning.cs.uiuc.edu, 2006.
4. Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research* 1986; **5**: 90–98.
5. Khosla P, Volpe R. Superquadratic artificial potentials for obstacle avoidance and approach. In *IEEE International Conference on Robotics and Automation*, 1988; pp. 1778–1784.
6. Rimon E, Koditschek D. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation* 1992; **8**: 501–518.
7. Amato N, Wu Y. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation*, 1996; pp. 113–120.
8. Barraquand J, Kavraki L, Latombe JC, Li TY, Motwani R, Raghavan P. A random sampling scheme for path planning. *International Journal of Robotics Research* 1997; **16**: 759–744.
9. Overmars M. A random approach to motion planning. Utrecht University, Technical Report RUU-CS-92-32, 1992.
10. Geraerts R, Overmars M. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, 2004; pp. 2386–2392.

DOI: 10.1002/cav

11. Geraerts R, Overmars M. On improving the clearance for robots in high-dimensional configuration spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005; pp. 4074–4079.
12. Nieuwenhuisen D, Kamphuis A, Mooijekind M, Overmars M. Automatic construction of roadmaps for path planning in games. In *International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004; pp. 285–292.
13. Geraerts R, Overmars M. Creating high-quality roadmaps for motion planning in virtual environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006; pp. 4355–4361.
14. Kamphuis A, Overmars M. Finding paths for coherent groups using clearance. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2004; pp. 19–28.
15. Wein R, Berg J, Halperin D. Planning near-optimal corridors amidst obstacles. In *International Workshop on the Algorithmic Foundations of Robotics*, 2006.
16. Bergen G. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann: San Francisco, CA, 2003.
17. Bouix S, Siddiqi K, Tannenbaum A. Flux driven fly throughs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003; pp. 449–454.

Authors' biographies:

Roland Geraerts received his Ph.D. in computer science in 2006 from Utrecht University in the Netherlands. Currently, he is a researcher/lecturer at the Department of Computer Science at the same university. His interests include motion planning and virtual environments.

Mark Overmars received his Ph.D. in computer science in 1983 from Utrecht University in the Netherlands. Currently, he is a full professor at the Department of Computer Science at the same university. Here, he is scientific director of the Center for Advanced Gaming and Simulation (AGS; www.gameresearch.nl).

His main research interests include motion planning, virtual environments, and game designing. Over the past years, he published over 250 papers in refereed journals and conferences, and he is the author of one of the prime textbooks on computational geometry.

Comp. Anim. Virtual Worlds 2007; **18**: 107–119
DOI: 10.1002/cav