Indicative Routes for Path Planning and Crowd Simulation

Ioannis Karamouzas Roland Geraerts Mark Overmars

Department of Information and Computing Sciences, Utrecht University
Padualaan 14, De Uithof, 3584CH Utrecht, The Netherlands
{ioannis, roland, markov}@cs.uu.nl

ABSTRACT

An important challenge in virtual environment applications is to steer virtual characters through complex and dynamic worlds. The characters should be able to plan their paths and move toward their desired locations, avoiding at the same time collisions with the environment and with other moving entities. In this paper we propose a general method for realistic path planning, the Indicative Route Method (IRM). In the IRM, a so-called indicative route determines a global route for the character, whereas a corridor around this route is used to handle a broad range of other path planning issues, such as avoiding characters and computing smooth paths. As we will show, our method can be used for real-time navigation of many moving characters in complicated environments. It is fast, flexible and generates believable paths.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Kinematics and dynamics*; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*games*

1. INTRODUCTION

The path planning problem has been extensively studied over the past years and many sophisticated algorithms have been devised to tackle it. These algorithms were mainly developed in the field of robotics, where the focus of the research was to generate collision-free and short paths for mostly static environments with a few moving robots (see [14, 15] for an extensive overview).

However, in interactive applications, such as computer games, the set of requirements is rather different. Complex and dynamic virtual environments are populated by a large number of computer-controlled characters. The characters must plan their paths and move toward their desired locations avoiding at the same time collisions with the environment and with other moving entities. The algorithms for computing these paths should be able to handle hundreds of characters in real-time using only a small percentage of the CPU time. They should also generate natural paths, i.e. paths that would be taken in real-life or paths that at least look convincing and believable to the player. In this paper, we present a new approach to plan such paths.

ICFDG 2009, April 26-30, 2009, Orlando, FL, USA

1.1 Related Work

The game development community mainly uses grid-based methods, navigation meshes, or waypoints graphs for the global navigation of the characters. Collisions with obstacles and other moving entities are then typically resolved by applying a reactive steering approach.

Grid-based methods divide the environment into a grid of cells that can be searched using A*-based algorithms [3, 25]. This approach guarantees that a path, if one exists, will be found. However, it lacks flexibility, since the same path is retrieved in response to a fixed query. In addition, grid-based methods can become computationally expensive, especially for large and complicated environments with many moving characters. Also, paths created by A* algorithms tend to be unnatural, as can be observed in many (recent) games.

Navigation meshes [3] partition the terrain into convex polygons that represent the walkable area of the environment. The static part of the game world is taken into account upon the construction of the mesh, and thus, the character only has to consider collisions with dynamic obstacles. Similar to a grid, a navigation mesh consists of linked cells that share a common edge. As a result, a free path can be retrieved by applying an A* search algorithm, which leads to the same drawbacks discussed above.

Another commonly used approach is to let the designers explicitly describe the motion of the characters using, for example, waypoint graphs. However, this is a labor intensive process and cannot adapt well to changes in the environment, limiting its applicability. Furthermore, the method is not very flexible, leading to repetitive behavior that can easily destroy the suspension of disbelief of the player.

Compared to the previous approaches, reactive methods can handle large dynamic environments providing enough flexibility for the characters to avoid local hazards, such as small obstacles and other moving entities. In general, reactive steering is based on variants of potential field methods [24]. However, the local nature of these methods gives no guarantees on the resulting paths. The characters are running the risk of getting stuck in local minima and not being able to reach their goals. This easily leads to deadlock situations that can only be resolved by rather unnatural motions, such as characters moving through walls or (re)appearing at different locations.

Path planning has recently also received a lot of attention in the computer graphics and animation community. Many models have been proposed to simulate individuals, groups and crowds of characters, including agents-based methods [22, 23], rule-based techniques [16, 17], (social) forces and particle systems [2, 19, 7]. These approaches work well in open environments and generate reasonable natural movements. The main drawback is that the characters base their decisions on local information and hence, they can easily get stuck in cluttered environments.

To address this, local models have been combined with global navigation techniques [1, 28, 13, 21, 26]. However, these methods do not scale well to dynamic environments with many characters. An alternative approach based on continuum dynamics has been proposed by Treuille *et al.* [30]. Although related to our research, their method simulates homogeneous groups of characters moving toward a common goal. In contrast, we focus on independent characters that have distinct characteristics and goals.

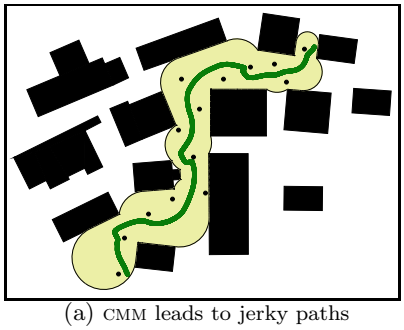
Crowd simulation has also been extensively studied in civil and traffic engineering and numerous models have been generated that exhibit emergent behaviors. The most popular is the work of Helbing [9]. Helbing simulated the behavior of pedestrians and traffic using physical forces. These are closely related to the potential field methods mentioned above and suffer from the same local-minima problems, because only local information is taken into account.

More recently, we have proposed the Corridor Map Method (CMM) as a new path planning method in interactive virtual worlds and games [5]. The intuition behind the CMM is straightforward. In a preprocessing phase the medial axis of the virtual environment is approximated (by exploiting graphics hardware) and a high-quality roadmap is constructed. This roadmap, enhanced with clearance information, defines the *corridor map*. When a character has to plan its path to a specified goal position a so-called *backbone path* is extracted from the corridor map together with a collision-free *corridor* around it. Then, a potential field approach is used to guide the local motion of the character inside the corridor, providing the desired flexibility. In particular, an *attraction point* moves along the backbone path and attracts the character in such a way that no collisions occur with the environment. The CMM has been successfully used to steer in real-time thousands of characters that navigate through complex virtual worlds, as well as to plan the motions of coherent groups of characters [4].

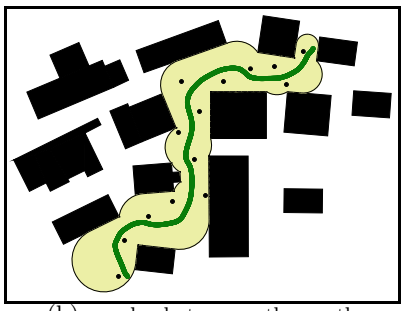
Although the method is fast and flexible, it limits the global behavior of the characters since the medial axis is used for global navigation. This encourages the characters to stay in the middle of the environment leading to unrealistic behavior. Furthermore, the generated paths tend to look unnatural when the characters have to avoid collisions with small static obstacles and with each other (see Figure 1(a)). The reason is that, due to the way the attraction point is defined, an almost infinite force steers the character during the simulation. Thus, the character has to be very close to an obstacle to avoid it and only at the very last moment adapts its motion, while in real-life people react much earlier.

1.2 Contributions

In this paper, we propose a general method for planning natural paths, called the Indicative Route Method (IRM), which is applicable to a wide range of interactive applications in computer games and virtual environments.



(a) CMM leads to jerky paths



(b) IRM leads to smoother paths

Figure 1: Obstacle avoidance inside corridors.

Our framework is inspired by the CMM, but provides more flexibility to handle a broad range of path planning issues, such as avoiding other characters and computing believable paths. We introduce the notion of *indicative routes* to indicate the global routes of the characters. This allows more freedom in the global planning, as the characters no longer have to follow the medial axis of the environment.

Given the indicative route of a character, we again form a *corridor* around that path and move an *attraction point* to steer the character toward its goal. We use, though, a different potential function to direct the character inside the corridor, providing better control over the local motions. Briefly, separate forces are applied to move the character forward and keep it inside the corridor. Additional forces are exerted to steer the character away from other characters, small obstacles and local hazards (see Figure 1(b) for an example). Furthermore, we take into account random variations in the character's behavior to increase the realism of the simulation.

As we will show, our method is relatively easy to implement and can be used for real-time navigation of many moving characters in complex and dynamic environments. It is fast, flexible and generates believable paths.

2. THE INDICATIVE ROUTE METHOD

In our problem setting, we are given a virtual environment, either 3D or 2D, in which a character has to move from a start to a specified goal position without colliding with the environment and other characters or dynamic obstacles. For simplicity, we assume that the character moves on a plane or a terrain and is modeled as a disc (cylinder in 3D) with radius r. At a fixed time t, the character is at position $\mathbf{x}(t)$, defined by the center of the disc, and moves with

velocity $\mathbf{u}(t)$ (hereafter, for notational convenience, we will not explicitly indicate the time dependence). In addition, the actual motion of the character is limited by a maximum speed $u^{\max}$.

We propose a three-phase method to tackle the path planning problem. The first phase of our approach consists of computing the so-called *indicative route* of the character. The indicative route $\Pi_{\mathcal{IR}}$ is a path that runs from the start ($\mathbf{s}$) to the goal position ($\mathbf{g}$) of the character and provides an indication/rough estimation of the character's preferred route. The character should be able to traverse this path, which means that the clearance at every point on the path must be at least the radius r of the character. More formally, the indicative route is a continuous map $\Pi_{\mathcal{IR}} \in [0,1] \rightarrow \mathbb{R}^2$, such that $\Pi_{\mathcal{IR}}[0] = s$, $\Pi_{\mathcal{IR}}[1] = g$ and $\forall s \in [0,1] : \Pi_{\mathcal{IR}}[s] \in \mathcal{C}_{\text{walk}}$. $\mathcal{C}_{\text{walk}}$ denotes the walkable (free) space in the environment consisting of all placements where the character does not intersect with the environment.

An indicative route could be indicated manually by a designer using, for example, waypoints. Alternatively, one could calculate the medial axis of the environment and extract a path with enough clearance. Another option is to divide the environment into a coarse grid and search for an appropriate path using A*-like approaches.

Note that the paths produced by these methods are in general not natural. Natural paths typically follow smooth curves, keep a preferred amount of clearance from obstacles, avoid unnecessary detours, allow for variations and can be easily adapted, if additional constraints (like avoiding other characters) impose this.

Therefore, the character should not traverse exactly the indicative route, but rather use it as a guide to plan its final motion. The second phase will allow for this by creating a *corridor* around that route, where the character can move without colliding with the environment. The corridor keeps the character's path in the same homotopic[1] class as the indicative route, providing at the same time enough flexibility for the character's local movements. Section 3 enunciates this phase.

In the third phase of our approach the path Π_f of the character is extracted from the corridor using a potential field approach. In particular, we move an attraction point along the indicative route and apply (social) forces to guide the character's motion through the corridor. This leads to a high-quality path as discussed in Section 4.

The specific choice of the indicative route is application and character dependent and falls outside the scope of this paper. Hence, for the rest of the paper we assume that the indicative route of the character is given and we will mainly focus on the other two phases of our proposed method. However, in Section 5 we will give an example of how indicative routes can be obtained for crowd simulation.

3. COMPUTING CORRIDORS

In this section we extend the indicative route to form a corridor in which the character can move freely. We first outline the basic aspects of the *corridor map* structure, introduced in [5] as an efficient data structure that represents the walkable (free) space of the environment and provides a network of collision-free corridors. Then, we explain how the corresponding corridor of an indicative route can be efficiently computed given the corridor map of the environment.

3.1 The Corridor Map

The corridor map is an enhanced graph $G = (V, E)$ whose edges represent collision-free corridors. These corridors are extracted from the Generalized Voronoi Diagram (GVD) or medial axis of the environment.

The GVD decomposes the free space of the environment into regions, such that all points in a region 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 (*Voronoi edges*) maintain a maximum clearance from the obstacles and define the so-called *backbone paths* of the corridor map. Together, the backbone paths form the skeleton (i.e. the underlying graph) of the map. See Figure 2 for an example of a virtual city, its footprint and the skeleton defining the corridor map.

The skeleton of the map is densely sampled and for each sampled point, the radius of the largest empty disc centered at this point is stored. A corridor $\mathcal{C} = (B[s], R[s])$ can then be defined as a sequence of maximum clearance discs with radii $R[s]$ whose center points B [s] lie along the backbone path B. The parameter s is an index ranging between 0 and 1. We refer the reader to Figure 2(c) for an example of a corridor.

Each sampled point on the skeleton is also enhanced with its set of closest obstacle points. This allows us to obtain an explicit representation of the corridor's boundary. Such an explicit description is needed to accurately compute the boundary force that is exerted on a character (see Section 4.1).

The corridor map can be efficiently constructed by exploiting graphics hardware as proposed in [6]. Note that the map is created only once during the preprocessing phase. Hence, it does not impose any limitations on the performance of the path planner.

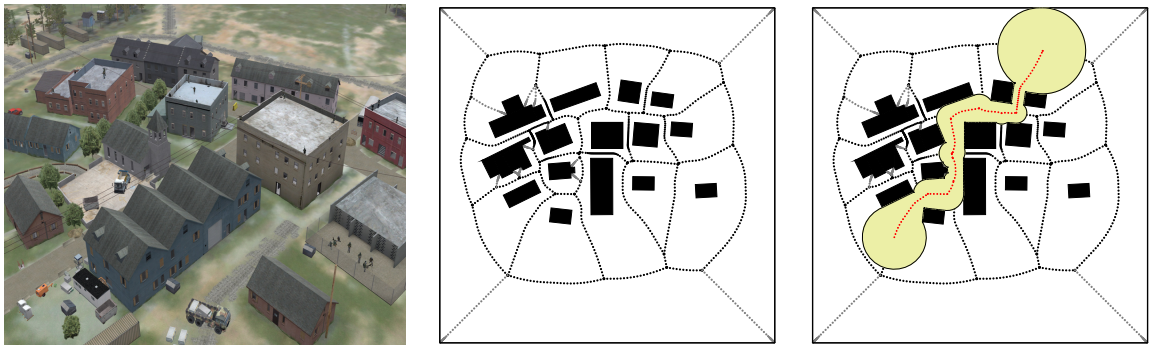
3.2 Retraction and Corridor Construction

The corridor map provides a system of collision-free corridors whose backbone paths lie on the medial axis. Thus, to create a corridor around the indicative route, we retract that route onto the medial axis and then retrieve the corresponding corridor from the corridor map structure.

Up to now we referred to the character's indicative route as a continuous path. However, such a continuous representation does not allow for an efficient/easy retraction of the route onto the medial axis. Hence, a discrete representation will be used. This can be obtained by dividing the route into a series of n adjacent points $\pi_0 \dots \pi_{n-1}$, where the distance $d(\pi_i, \pi_{i-1})$ is at most a predetermined step size.

To retract a point p of a discrete path onto the medial axis, we need to find its closest obstacle point cp_o and then move p toward the $\overrightarrow{cp_op}$ direction until the closest obstacle point changes [18]. From the corridor map structure we already know the points that lie on the medial axis, as well as their corresponding closest obstacle points. Thus, the problem delegates to retrieve the edge in the corridor map and the point p' on this edge so that the distance between p and the line connecting p' and its closest obstacle point becomes minimal. This can be efficiently implemented using

[1]Two paths Π_0 and Π_1, with endpoints fixed, are said to be homotopic only if one path can be continuously transformed into the other without intersecting any obstacles.



(a) 3D Environment[10] (b) Skeleton of the corridor map (c) Backbone path and its corridor

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

a kd-tree data structure.

Consequently, a simple approach to retract the indicative route of the character would be to query the kd-tree for every point of the route. However, this is a rather time-consuming process. A more efficient solution is the following. Suppose that the corridor map edge e and the retracted point corresponding to the start position of the character (i.e. the π_0) have been retrieved from the kd-tree. Let us also assume that the indicative route moves along that edge toward a vertex ν in the graph. Then, for each point $\pi_i : i \in [1, n-1]$ we only have to search for its retracted point along the current edge. This can be easily achieved using simple geometric calculations. If the retracted point reaches the end of the edge at vertex ν, we query the outgoing edges of ν and retrieve the new Voronoi edge from the corridor map. The procedure continues until all the points of the indicative route are retracted onto the medial axis.

The resulting backbone path is then used to define the corridor of the character's indicative route. As an example, consider Figure 3(b). It depicts the backbone path and its corresponding corridor for the indicative route shown in Figure 3(a). The method is very fast. Retracting this route took 0.21 *ms* on a Pentium IV 2.4 GHz.

4. LOCAL NAVIGATION IN IRM

Once the corridor has been created, the character plans its final path using a force field approach. Several forces are defined that act on the character and influence its movement. First, the character must stay inside the corridor and hence, a repulsive force pushes the character away from the boundary of the corridor. Second, a steering force advances the character toward its goal. Third, a noise force allows for random variations in the character's path. Finally, a repulsive force is exerted on the character to avoid collisions with static and dynamic obstacles, as well as other characters.

4.1 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 an individual prefers to keep a safe distance from walls, streets, buildings, *etcetera* [27, 29], such a force is only exerted if the Euclidean distance d_b between the character at position $\mathbf{x}$ and its corresponding closest point on the boundary of the corridor $\mathbf{b}(\mathbf{x})$ is below a threshold value. Let r be the radius of the character and let d_s denote the preferred safe distance. Then, the force is defined as follows:

$$\mathbf{F}_b = \begin{cases} \mathrm{c}_b \dfrac{\mathbf{x} - \mathbf{b}(\mathbf{x})}{\|\mathbf{x} - \mathbf{b}(\mathbf{x})\|}, & \text{if } d_b - r < d_s \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The scalar $\mathrm{c}_b = \dfrac{d_s + r - d_b}{(d_b - r)^\kappa}$ is chosen such that the force will become infinite when the character and the boundary point touch, whereas the constant κ indicates the steepness of the repulsive potential. By modifying the safe distance d_s a wide variety of behaviors can be achieved. For example, a small safe distance allows the character to get close to obstacles, whereas a large one makes the character stay in the middle of the corridor.

However, to guarantee that in any step of the simulation the character will not collide with the environment, a lower bound on the safe distance is set based on the character's maximum speed $u^{\max}$. In addition, to avoid oscillations, an upper bound is also determined by taking into account the minimum clearance $R^{\min}$ of the corridor. Thus, the set of admissible safe distances $\mathcal{D}$ for the character is defined as:

$$\mathcal{D} = \left\{ d_s \,|u^{\max}\Delta t < d_s < R^{\min} - r \right\}, \quad (2)$$

where Δt denotes the time step size of the simulation.

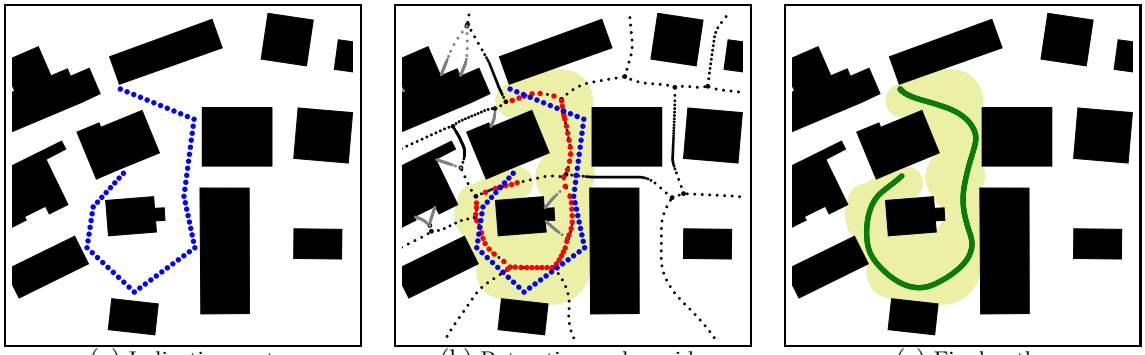
4.2 The Steering Force

While the boundary force keeps the character inside the corridor, an additional force is needed to guide the character forward toward its goal position. For this purpose, we move an *attraction point* along the indicative route of the character. Based on this attraction point, a steering force $\mathbf{F}_s$ is generated that steers the character positioned at $\mathbf{x}$ toward the attraction point $\mathbf{p}_\alpha(\mathbf{x})$. Let c_s be a constant specifying the relative strength of the force. Then, $\mathbf{F}_s$ is given by:

$$\mathbf{F}_s(\mathbf{x}) = c_s \frac{\mathbf{p}_\alpha(\mathbf{x}) - \mathbf{x}}{\|\mathbf{p}_\alpha(\mathbf{x}) - \mathbf{x}\|} \quad (3)$$

4.3 Choosing an Attraction Point

To advance the character along its indicative route, the chosen point of attraction is of paramount importance. In every cycle of the simulation the attraction point should



(a) Indicative route (b) Retraction and corridor (c) Final path

Figure 3: The Indicative Route Method.

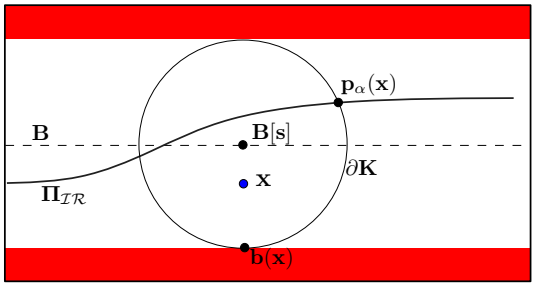


Figure 4: The attraction point $\mathbf{p}_\alpha(\mathbf{x})$ for the character positioned at $\mathbf{x}$.

steer the character toward its goal, ensuring at the same time that it does not get stuck in local minima.

Let $\mathbf{x}$ be the current position of the character, $B[s]$ denote its corresponding (retracted) point on the backbone path and $K(B[s], R[s])$ be the largest clearance disk with radius $R[s]$ centered at $B[s]$, where $s : s \in [0, 1]$.

Then, the attraction point $\mathbf{p}_\alpha(\mathbf{x})$ can be defined as the furthest point on the character's indicative route $\Pi_{\mathcal{IR}}$ that still intersects the boundary ∂K of the clearance disc K (see Figure 4):

$$\mathbf{p}_\alpha(\mathbf{x}) = \Pi_{\mathcal{IR}} \left(\underset{s' \in (s,1]}{\arg\max} \left\{ \Pi_{\mathcal{IR}}[s'] \cap \partial K \neq \emptyset \right\} \right) \quad (4)$$

Such an intersection point always exists, since there is a one-to-one mapping between the character's indicative route and the backbone path. In general, it can be easily shown that if the number of intersection points is odd, the character's goal lies inside the clearance disc and thus, it is used to attract the character. If an even number of points intersects the clearance disc, selecting the one with the largest index drives the character forward toward its goal.

This intersection point lies always ahead of the character, i.e. $\|\mathbf{p}_\alpha(\mathbf{x}) - \mathbf{x}\| > 0$, and hence, the magnitude of the steering force $\mathbf{F}_s$ is always larger than zero (except when the goal position is reached). In addition, the intersection point can never be located on the closest boundary point $\mathbf{b}(\mathbf{x})$ corresponding to the character's current position, since we require the clearance at every point along the indicative route to be at least the radius r of the character. Therefore, the boundary force $\mathbf{F}_b$ does not cancel out the steering force $\mathbf{F}_s$ and consequently $\mathbf{F}_s + \mathbf{F}_b \neq 0$. The latter guarantees that, in the absence of any local hazards, a path to the goal will always be found.

4.4 Path Variation

To take into account random variations in the paths that the characters follow and generate a (slightly) different path every time the same path planning problem has to be solved, a noise force $\mathbf{F}_n$ is also introduced in our model. Like in [11], we use a Perlin noise [20] function to control the direction of the force, ensuring that it changes smoothly during the simulation. In our problem setting, Perlin noise is implemented as a function over time. The current time step of the simulation is given as an input and a direction for the force is returned that varies pseudo-randomly. Let θ denote this direction, expressed as an angle of rotation around the current steering vector. Then, the noise force is defined as:

$$\mathbf{F}_n(\mathbf{x}) = c_n \, \mathcal{R}(\theta) \, \frac{\mathbf{p}_\alpha(\mathbf{x}) - \mathbf{x}}{\|\mathbf{p}_\alpha(\mathbf{x}) - \mathbf{x}\|}, \quad (5)$$

where $\mathcal{R}(\theta)$ represents the 2D rotation matrix and $\mathbf{p}_\alpha(\mathbf{x})$ is the attraction point of the character positioned at $\mathbf{x}$.

The constant c_n specifies the relative strength of the force. It must be small with respect to the magnitude of the steering force, whereas $\theta \in [-\pi/2, +\pi/2]$. As a result, the character retains its steering direction while at the same time performs small random displacements. However, care should be taken not to vary the direction of the force (i.e. the noise function) too frequently. A low frequency leads to smooth movements, whereas a relatively high frequency generates erratic and unusable behavior.

4.5 Avoiding Obstacles

Although the corridor provides a collision-free area around the control path, the character may still have to avoid a number of obstacles while moving inside the corridor. Such obstacles can either be small static obstacles that were not taken into account when the corridor map was created or dynamic obstacles, such as other characters and moving entities. Thus, an additional collision response force has to be applied.

Let $\mathrm{O} \in \mathbb{R}^2$ denote a set of n obstacles that the character needs to avoid, while planning its motion inside the corridor. Every obstacle $O_i : i \in [1 : n]$ exerts a repulsive force, which

in its simplest form is monotonically decreasing with the Euclidean distance d_i between the obstacle and the character at position $\mathbf{x}$. Such a force is only applied if the character lies inside the influence region of the obstacle defined by d_o, that is, if $d_i - r_i - r \leq d_o$, where r is the radius of the character and r_i the radius of the obstacle. Let k_r be a parameter used to scale the effect of the repulsive force on the character. Then, the total obstacle avoidance force $\mathbf{F}_o$ that acts on the character can be described by:

$$\mathbf{F}_o(\mathbf{x}) = \sum_{i=1}^{n} c_o \frac{\mathbf{x} - O_i}{\|\mathbf{x} - O_i|}, \text{ where } c_o = \frac{k_r}{d_i - r_i - r} \qquad (6)$$

Figure 1(b) shows an example path created by our method, using the medial axis as the indicative route of the character. The repulsive potential described above has been applied to avoid a number of static obstacles inside the extracted corridor. Clearly, the resulting path is more natural than the path produced by the CMM (Figure 1(a)).

Although we do not distinguish between static and dynamic obstacles, the problem becomes more challenging when many characters populate the virtual environment. Therefore, more elaborate avoidance approaches can be used to e.g. simulate crowds, like the models proposed in [8, 9].

4.6 Time Integration and Final Path

Given the applied forces, the final force $\mathbf{F}$ exerting on the character at position $\mathbf{x}$ is defined as:

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}_s(\mathbf{x}) + \mathbf{F}_b(\mathbf{x}) + \mathbf{F}_n(\mathbf{x}) + \mathbf{F}_o(\mathbf{x}) \qquad (7)$$

The force $\mathbf{F}$ results in an acceleration term as described by Newton's second law of motion, i.e. $\mathbf{F} = m\mathbf{a}$, where m is the mass of the character and $\mathbf{a}$ its acceleration. Assuming $m = 1$, the position of the character at time t is computed by integrating the equation of motion, that is

$$\frac{d^2(\mathbf{x}(t))}{dt^2} = \mathbf{F}(\mathbf{x}(t)) \qquad (8)$$

Any numerical integration scheme, such as Euler's method, can be used to solve this differential equation. However, since we are dealing with forces that have widely different scales and can vary quickly, a stable integration scheme like Verlet integration [31] is recommended. To retain stability and avoid stiffness a relatively small time step size Δt should also be preferred.

The final path Π_f of the character can then be obtained by iteratively integrating $\mathbf{F}$ over time, while updating the character's position, velocity, and attraction point. As an example consider Figure 3(c). It displays a *smooth* (i.e. C^1-continuous) path that was generated by applying the above mentioned integration scheme.

5. CROWD SIMULATION IN IRM

Up until now, we have assumed that the indicative route of a character is provided by the level designer. However, if we want to steer crowds of characters, this process would be inefficient and time consuming. Thus, to facilitate the creation and extraction of indicative routes we introduce the notion of *indicative networks*.

An indicative network $\mathcal{IN} = (V, E)$ is a roadmap of paths that can be used to lead the characters toward their goals. Each vertex $\nu \in V$ of the network corresponds to a collision-free point in a 2D or 3D workspace, whereas each edge $e \in E$ corresponds to an indicative route Π_e that a character can follow. Such a network is constructed in a pre-processing phase.

In practice, any roadmap graph that captures the connectivity of the free space is suited for guiding the motions of the characters. For example, we can use the medial axis itself to encourage the characters to stay in the middle of the environment. We can also compute an indicative network based on the Voronoi-Visibility diagram [32], allowing characters to take shortcuts when there is enough clearance.

Alternatively, an indicative network can be determined manually by the level designer, to encourage certain behavior, e.g. characters that prefer to stay on the sidewalks of a road. The network should be flexible enough so that the designer can easily manipulate it by adding, changing, or removing indicative routes.

Given an indicative network, created in a preprocessing phase, the indicative route of the character is extracted from the network as follows. We first connect the start position s of the character to the network. This can be done by adding an edge between the start position and the closest visible point that lies on one of the local routes Π_e of the network. We proceed in the same way for the goal position g of the character. Then, we run an A* shortest path algorithm on the network graph to retrieve the character's indicative route. Given this route, we can now use the IRM method to compute the path of the character.

Since we are dealing with many moving characters, we have to choose a more realistic collision avoidance approach than the one described in Section 4.5, so that the characters can evade each other naturally. Therefore, we use part of the social force model proposed by Helbing and Molnar in [9] and known to exhibit emergent crowd behavior.

In their model, every character has its own personal space, which is defined as an ellipse directed toward the character's motion. As soon as a character steps into the personal space of another character, a repulsive force is acted from the latter to the former, ensuring that they will avoid each other in a human-like manner.

For computing the repulsive forces, we have to know/find the neighbors of each character. Since this operation is performed many times, an efficient data structure for answering nearest neighbors queries has to be implemented. For example, a spatial hash data structure or a 2D grid map that stores and updates at every simulation step the locations of the characters can be employed [26, 6].

6. EXPERIMENTS

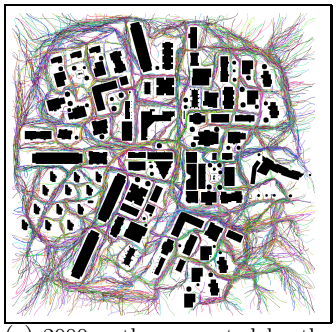
We have implemented our proposed IRM to experimentally test its effectiveness and applicability in real-time applications. All the experiments were performed on a PC running Windows XP, with a 2.4 GHz Intel Core2 Duo CPU and 2 GB memory.

The experiments were conducted for the environment depicted in Figure 5(a). This is a model of a virtual city. The city measures 500x500 meter. Its geometry is composed of 220K triangles, while its 2d footprint, displayed in Figure 5(b), consists of 2122 triangles. An indicative network was used for the extraction of the indicative routes.

In our simulations we used Verlet integration with a small step size $\Delta t = 0.1$ to keep the errors minimal. The maximum speed of a character was set to 1.4 m/s [12] ($u^{\max} = 1.4$), its radius to 0.25 cm ($r = 0.25$) and its safe distance from

(a) 3D Model

(b) Footprint



(c) 2000 paths generated by the IRM

Figure 5: The City environment.

the boundaries of the corridors to 2.0 m ($d_s = 2$).

6.1 Results and Discussion

To test the performance of our method, we selected a varying number of characters and placed them randomly in the city environment. Each character had to advance toward a random goal position. When it had reached its destination, a new goal was assigned. See Figure 5(c) for an example.

We performed experiments for up to 5,000 characters and set the maximum number of simulation steps to one thousand. We measured the CPU-load as the amount of CPU time that is required to generate one second of characters' movements, that is CPU-load $= \frac{\text{total CPU time}}{\text{avg traversed time}} * 100\%$. Bearing in mind that in virtual environment applications, such as computer games, only a small fraction of processor time is scheduled to the path planner, we considered the performance as real-time when the CPU-load was at most 30%.

During our benchmark, the scene and character rendering was disabled, as our goal was to analyse the path planning cost of the IRM. Note that due to our current implementation only one CPU core was used for path computation. The results are shown in Figure 6.

As an example, consider the processor usage that was needed for 1,000 characters. On average, it took 7.25 ms to compute a character's path. The average time to traverse all the resulting paths was 118 s. Hence, 61.5 ms per second traversed time were required to simultaneously steer the characters, which implies a cpu-load of 6,15%.

As the figure indicates, approximately 3,500 characters can be simulated in real-time. In addition, the CPU-load scales almost linearly with the number of characters. Even for 5,000 characters our method can efficiently plan paths at interactive rates requiring less than 55% of the processor power. We conclude that the IRM can be used for real-time path planning of a large number of characters.

Besides the performance, we are also interested in the quality of the paths generated by the IRM. As can be seen from Figure 1, the quality of the resulting paths is much better than using the CMM method.

We refer the reader to *http://people.cs.uu.nl/ioannis/irm* for the results we obtained in simulating crowds. As we used part of the Helbing's social force model for the local interaction of the characters, we also observed the phenomenon of lane formation that is widely noted in pedestrian literature. However, an artifact of this model is that the characters

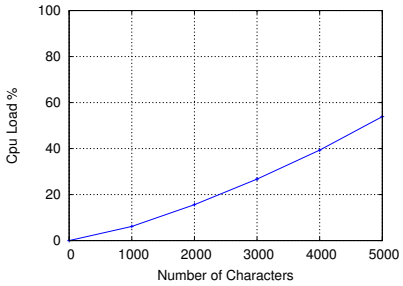


Figure 6: The performance of the IRM on the crowd-simulation scenario.

tend to adapt their motions rather late in order to evade a potential collision. Such behavior is visible in, basically, all crowd simulation solutions available today. In the IRM, though, it is easy to incorporate other local methods, which we are currently investigating.

In addition, using corridors, our framework can be easily extended to capture a wide variety of crowd characteristics. For example, we can include in our simulations coherent groups of characters or characters that wander through a virtual environment without any specific goal [4].

7. CONCLUSION AND FUTURE WORK

In this paper, we have presented a new method for high-quality path planning, the indicative route method (IRM). In the IRM, an indicative route directs the global motion of the character, whereas a potential field approach is used to guide the character through a corridor around this route.

We have experimentally shown that our method can be used to plan in real-time high quality paths for a large number of characters in complex and dynamic environments. It is relatively easy to implement and is flexible enough to incorporate many additional constraints.

Inspired by our current research, we believe that the IRM has a lot of potential and can form the basis for further research. We currently work on techniques to automatically create networks of indicative routes. To facilitate this, we plan to exploit existing video recordings and motion capture data to understand how people solve a typical path planning problem in real life.

We also want to incorporate the notion of influence regions into our path planner, so that the characters can adapt their indicative routes based on local information. Such regions can either be dangerous places the characters prefer to avoid or appealing locations they would like to visit.

8. 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).

9. REFERENCES

[1] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Better group behaviors in complex environments using global roadmaps. In *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages 362–370, Cambridge, MA, USA, 2003. MIT Press.

[2] O. Cordeiro, A. Braun, C. Silveira, S. Musse, and G. Cavalheiro. Concurrency on social forces simulation model. In *Proc. First International Workshop on Crowd Simulation (V-Crowds'05)*, 2005.

[3] M. DeLoura. *Game Programming Gems 1*. Charles River Media, Inc., 2000.

[4] R. Geraerts, A. Kamphuis, I. Karamouzas, and M. H. Overmars. Using the corridor map method for path planning for a large number of characters. In *Motion in Games, First International Workshop, MIG 2008, Utrecht, The Netherlands. Revised Papers*, pages 11–22. Springer-Verlag, 2008.

[5] R. Geraerts and M. Overmars. The corridor map method: A general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds*, 18:107–119, 2007.

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

[7] L. Heigeas, A. Luciani, J. Thollot, and N. Castagné. A physically-based particle model of emergent crowd behaviors. In *Graphicon*, 2003.

[8] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[9] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical Review E*, 51:4282–4286, 1995.

[10] M. Inc. Metavr real-time pc-based 3d visual simulation. http://www.metavr.com (Accessed February 18, 2009).

[11] I. Karamouzas and M. H. Overmars. Adding variation to path planning. *Computer Animation and Virtual Worlds*, 19(3-4):283–293, 2008.

[12] R. Knoblauch, M. Pietrucha, and M. Nitzburg. Field studies of pedestrian walking speed and start-up time. *Transportation Research Record*, 1538:27–38, 1996.

[13] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23:509–518, 2004.

[14] J.-C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. Also available at http://planning.cs.uiuc.edu/.

[16] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. *Theory and Practice of Computer Graphics*, 2003.

[17] S. R. Musse and D. Thalmann. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[18] C. Ó'Dúnlaing, M. Sharir, and C. Yap. Retraction: A new approach to motion planning. In *ACM Symposium on Theory of Computing*, pages 207–220, 1983.

[19] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108, 2007.

[20] K. Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, July 1985. Proc. of SIGGRAPH '85.

[21] J. Pettré, P. de Heras Ciechomski, J. Maïm, B. Yersin, J.-P. Laumond, and D. Thalmann. Real-time navigating crowds: scalable simulation and rendering. *Computer Animation and Virtual Worlds*, 17(3-4):445–455, 2006.

[22] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987. Proceedings of SIGGRAPH '87.

[23] C. W. Reynolds. Steering behaviors for autonomous characters. In *Proc. of Game Developers Conference*, pages 763–782, San Jose, California, 1999.

[24] E. Rimon and D. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8:501–518, 1992.

[25] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[26] W. Shao and D. Terzopoulos. Autonomous pedestrians. *Graphical Models*, 69(5-6):246–274, 2007.

[27] P. Stucki. Obstacles in pedestrian simulations. Master's thesis, Swiss Federal Institute of Technology ETH, 2003.

[28] M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528, 2004.

[29] Transportation Research Board, National Research Council, Washington, D.C. *Highway Capacity Manual*, 2000.

[30] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[31] L. Verlet. Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98+, July 1967.

[32] R. Wein, J. Berg, and D. Halperin. The Visibility-Voronoi complex and its applications. In *Annual Symposium on Computational Geometry*, pages 63–72, 2005.