Path-Planning for RTS Games Based on Potential Fields

Renato Silveira, Leonardo Fischer, José Antônio Salini Ferreira, Edson Prestes, and Luciana Nedel

Universidade Federal do Rio Grande do Sul, Brazil
{rsilveira,lgfischer,jasferreira,prestes,nedel}@inf.ufrgs.br
http://www.inf.ufrgs.br

Abstract. Many games, in particular RTS games, are populated by synthetic humanoid actors that act as autonomous agents. The navigation of these agents is yet a challenge if the problem involves finding a precise route in a virtual world (path-planning), and moving realistically according to its own personality, intentions and mood (motion planning). In this paper we present several complementary approaches recently developed by our group to produce quality paths, and to guide and interact with the navigation of autonomous agents. Our approach is based on a BVP Path Planner that generates potential fields through a differential equation whose gradient descent represents navigation routes. Resulting paths can deal with moving obstacles, are smooth, and free from local minima. In order to evaluate the algorithms, we implemented our path planner in a RTS game engine.

Keywords: Path-planning, navigation, autonomous agent.

1 Introduction

Recent advances in computer graphics algorithms, especially on realistic rendering, allow the use of synthetic actors visually indistinguishable from real actors. These improvements benefit both the movie and game industry which make extensive use of virtual characters that should act as autonomous agents with the ability of playing a role into the environment with life-like and improvisational behavior. Despite a realistic appearance, they should present convincing individual behaviors based on their personality, moods and desires. To behave in such way, agents must act in the virtual world, perceive, react and remember their perceptions about this world, think about the effects of possible actions, and finally, learn from their experience [7].

In this complex and suitable context, navigation plays an important role [12]. To move agents in a synthetic world, a semantic representation of the environment is needed, as well as the definition of the agent initial and goal position. Once these parameters were set, a path-planning algorithm must be used to find a trajectory to be followed.

However, in the real world, if we consider different persons – all of them in the same initial position – looking for achieving the same goal position, each path followed will be unique. Even for the same task, the strategy used for each person to reach his/her goal depends on his/her physical constitution, personality, mood, reasoning, urgency, and so on. From this point of view, a high quality algorithm to move characters across virtual environments should generate expressive, natural, and occasionally unexpected steering behaviors. In contrast, especially in the game industry, the high performance required for real-time graphics applications compels developers to look for most efficient and less expensive methods that produce good and almost natural movements.

In this paper, we present several complementary approaches recently developed by our group [4, 15, 14, 6, 18] to produce high quality paths and to control the navigation of autonomous agents. Our approach is based on the BVP Path Planner [14], that is a method based on the numeric solution of the boundary value problem (BVP) to control the movement of agents, while allowing the individuality of each one.

The topics presented in this article are: (*i*) a robust and elegant algorithm to control the steering behavior of agents in dynamic environments; (*ii*) the production of interesting and complex human-like behaviors while building a navigational route; (*iii*) a strategy to handle the path-planning for group of agents and the group formation-keeping problem, enabling the user to sketch any desirable formation shape; (*iv*) a sketch based global planner to control the agent navigation; (*v*) a strategy to deal with the BVP Path Planner in GPU; (*vi*) a RTS game implementation using our technique.

The remaining of this paper is structured as follows. Section 2 reviews some related works on path-planning techniques that generate quality paths and behaviors. Section 3 explains the concepts of our planner and how we deal with agents and group of agents. Section 4 proposes an alternative to our global path planner, enabling the user interaction. Improvements in the performance are discussed in Section 5. Some results, including a RTS game implementation using our technique, is shown in Section 6. Section 7 presents our conclusions and some proposals for future works.

2 Related Work

The research on path-planning has been extensively explored on the games domain where the programmer should frequently deal with many autonomous characters, ideally presenting convincing behavior. It is very difficult to produce natural behavior by using a strategy focusing on the global control of characters. On the other hand, taking into account the individuality of each character may be a costly task. As a consequence, most of the approaches proposed in computer graphics literature do not take into account the individual behavior of each agent, compromising the planner quality.

Kuffner [8] proposed a technique where the scenario is mapped onto a 2D mesh and the path is computed using a dynamic programming algorithm like Dijkstra.

He argues that his technique is fast enough to be used in dynamic environments. Another example is the work developed by Metoyer and Hodgings [11], that proposes a technique where the user defines the path that should be followed by each agent. During the motion, the path is smoothed and slightly changed to avoid collisions using force fields that act on the agent.

The development of randomized path-finding algorithms – specially the PRM (Probabilistic Roadmaps) [10] – allowed the use of large and more complex configuration spaces to efficiently generate paths. There are several works in this direction [3, 13]. In most of these techniques, the challenge becomes more the generation of realistic movements than finding a valid path. Differently, Burgess and Darken [2] proposed a method based on the *principle of least action* which describes the tendency of elements in nature to seek the minimal effort solution. The method produces human-like movements through realistic paths using properties of fluid dynamics.

Tecchia et al. [16] proposed a platform that aims to accelerate the development of behaviors for agents through local rules that control these behaviors. These rules are governed by four different control levels, each one reflecting a different aspect of the behavior of the agent. Results show that, for a fairly simple behavioral model, the system performance can achieve interactive time. Treuille et al. [17] proposed a crowd simulator driven by dynamic potential fields which integrates global navigation and local collision avoidance. Basically, this technique uses the crowd as a density field and constructs, for each group, a unit cost field which is used to control people displacement. The method produces smooth behavior for a large amount of agents at interactive frame rates. Based on local control, van den Berg [1] proposed a technique that handles the navigation of multiple agents in the presence of dynamic obstacles. He uses a concept, called *velocity obstacles*, to locally control the agents with few oscillation.

As mentioned above, most of the approaches do not take into account the individual behavior of each agent, his internal state or mood. Assuming that realistic paths derive from human personal characteristics and internal state, thus varying from one person to another, we [14] recently proposed a technique that generates individual paths. These paths are smooth and dynamically generated while the agent walks. In the following sections, we will explain the concepts of our technique and the extensions implemented to handle several problems found in RTS games.

3 BVP Path Planner

The BVP Path Planner, originally proposed by Trevisan et al. [18], generates paths using the potential information computed from the numeric solution of

$$\nabla^2 \, p(\mathbf{r}) \;=\; \epsilon \mathbf{v} \cdot \nabla p(\mathbf{r}) \;, \tag{1}$$

with Dirichlet boundary conditions. In Equation 1, $\mathbf{v} \in \Re^2$ and $|\mathbf{v}| = \mathbf{1}$ corresponds to a vector that inserts a perturbation in the potential field; $\epsilon \in \Re$ corresponds to the intensity of the perturbation produced by $\mathbf{v}$; and $p(\mathbf{r})$ is the potential at position $\mathbf{r} \in \Re^2$. Both $\mathbf{v}$ and ϵ must be defined before computing this equation. The gradient descent on these potentials represents navigational routes from any point of the environment to the goal position. Trevisan et al. [18] showed that this equation does not produce local minima and generates smooth paths.

To solve numerically a BVP, we can consider that the solution space is discretized in a regular grid. Each cell (i, j) is associated to a squared region of the environment and stores a potential value $p(i, j)$. Using Dirichlet boundary conditions, the cells associated to obstacles in the real environment store a potential value of 1 (*high potential*) whereas cells containing the goal store a potential value of 0 (*low potential*).

A high potential value prevents the agent from running into obstacles whereas a low value generates an attraction basin that pulls the agent. The relaxation method usually employed to compute the potentials of free space cells is the Gauss-Seidel (GS) method. The GS method updates the potential of a cell c through:

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \tag{2}$$

where $\mathbf{v} = (v_x, v_y)$, and $p_c = p_{(i,j)}$, $p_b = p_{(i,j-1)}$, $p_t = p_{(i,j+1)}$, $p_r = p_{(i+1,j)}$ and $p_l = p_{(i-1,j)}$. The potential at each cell is updated until the convergence sets in.

After the potential computation, the agent moves following the direction of the gradient descent of this potential at its current position (i, j),

$$(\nabla \mathbf{p})_c = \left(\frac{p_l - p_r}{2}, \frac{p_b - p_t}{2} \right).$$

In order to implement the BVP Path Planner, we used global environment maps (one for each goal) and local maps (one for each agent). The global map is the Global Path Planner which ensures a path free of local minima, while the local map is used to control the steering behavior of each agent, also handling dynamic obstacles.

The entire environment is represented by a set of homogeneous meshes $\{\mathcal{M}_k\}$, where each mesh $\mathcal{M}_k$ has $L_x \times L_y$ cells, denoted by $\{\mathcal{C}^k_{i,j}\}$. Each cell $\mathcal{C}^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $\mathcal{P}^k_{i,j}$. The potential associated to each cell is computed by GS iterations (Equation 2) and then used by the agents to reach the goal. In order to delimit the navigation space, we consider that the environment is surrounded by static obstacles.

3.1 Dealing with Individuals

Each agent a_k has one local map m_k that stores the current local information about the environment obtained by its own sensors. This map is centered in the current agent position and represents a small fraction of the global map. The size of the local map influences the agent behavior. A detailed analysis of the influence of the size of the local map on the behavior of the agent can be found in [14].

The local map is divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*); and the border zone (*b-zone*), as shown in Figure 1. Each cell corresponds to a squared region, similar to the global map. For each agent, a goal, a particular vector $\mathbf{v}_k$ that controls its behavior, and a ϵ_k should be stated. If $\mathbf{v}_k$ or ϵ_k is dynamic, then the function that controls it must also be specified.

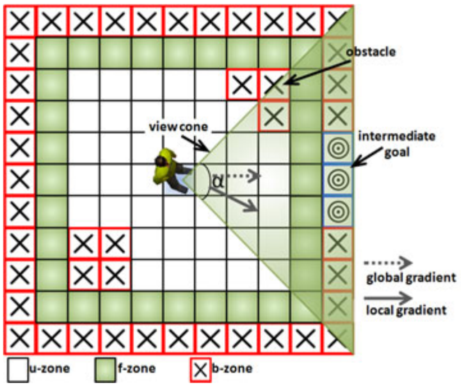

Fig. 1. Agent Local Map. White, green and red cells comprise the *update*, *free* and *border zones*, respectively. Blue, red and light blue cells correspond to the intermediate goal, obstacles and the agent position, respectively.

To navigate into the environment, an agent a_k uses its sensors to perceive the world and to update its local map with information about obstacles and other agents. The agent sensor sets a view cone with aperture α. Figure 1 sketches a particular instance of the agent local map. The *u-zone* cells that are inside the view cone and correspond to obstacles or other agents have their potential value set to 1. Dynamic or static obstacles behind the agent do not interfere in its future motion.

For each agent a_k, we calculate the global descent gradient on the cell of the global map containing its current position. The gradient direction is then used to generate an intermediate goal in the border of the local map, setting the potential values to 0 of a couple of *b-zone* cells, while the other *b-zone* cells are considered as obstacles, with their potential values set to 1. The intermediate goal helps the agent a_k to reach its goal while allowing it to produce a particular motion.

F-zone cells are always considered free of obstacles, even when there are obstacles inside. The absence of this zone may close the connection between the current agent cell and the intermediate goal due to the possible mapping of obstacles in front of the intermediate goal. When this occurs, the agent gets lost because there is no information coming from the intermediate goal to produce a path to reach. *F-zone* cells handle the situation, always allowing the propagation of the information about the goal to the cells associated to the agent position.

After the sensing and mapping steps, the agent k updates the potential value of its map cells using Equation 2 with its pair $\mathbf{v}^k$ and ϵ^k. Hereinafter, it updates its position according to:

$$\varDelta\, \mathbf{d} = \upsilon\ (\cos(\varphi^t), \sin(\varphi^t))\ \varPsi(|\varphi^{t-1} - \zeta^t|) \tag{3}$$

with

$$\varphi^t = \eta\ \varphi^{t-1} + (1-\eta)\ \zeta^t \tag{4}$$

where υ defines the maximum agent speed, $\eta \in [0, 1)$, φ is the agent orientation and ζ is the orientation of the gradient descent computed from the potential field stored on its local map in the central position. Function $\varPsi : \Re \rightarrow \Re$ is

$$\varPsi(x) = \begin{cases} 0 & \text{if } x > \pi/2 \\ \cos(x) & \text{, otherwise} \end{cases}$$

If $|\varphi^{t-1} - \zeta^t|$ is higher than $\frac{\pi}{2}$, then there is a high hitting probability and this function returns the value 0, making the agent stops. Otherwise, the agent speed will change proportionally to the collision risk.

In order to demonstrate that the proposed technique produces realistic behaviors for humanoid agents, we compared the paths generated by our method with real human paths [14] . Associating specific values to the parameters ϵ and $\mathbf{v}$ in the agent local map, the path produced by the BVP Path Planner almost mimics the human path. Figure 2 shows a path generated by dynamically changing ϵ and $\mathbf{v}$. Up to the half of the path, the parameters $\epsilon = 0.1$ and $\mathbf{v}= (0, -1)$ were used. Half the path forward, the parameters were changed to $\epsilon = 0.7$ and $\mathbf{v}= (1, 0)$. These values were empirically obtained. We can visually compare and observe that the calculated path is very close to the real one.

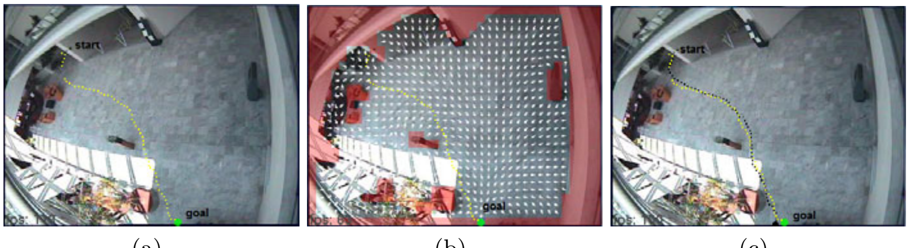

Fig. 2. Natural path generated by the BVP Path Planner: (a) the environment with the real person path (yellow); (b) the environment representation discretized by our planner; (c) the agent path (in black) calculated by our planner after adjusting the parameters

3.2 Dealing with Groups

The organization of agents in groups has two main goals: to allow the control of groups or armies by the user, and to decrease the computational cost through the management of groups instead of handling individuals. In a previous work [15], an approach to integrate group control in the BVP Path Planner was proposed. This approach also support the group formation-keeping while agents move following a given path. Kallmann [9] recently proposed a path planner to efficiently produce quality paths that could also be used for moving groups of agents. Our technique, illustrated in Figure 3, focus on the group cohesion and behavior while enabling the user to interact with the group.

In this approach, each group is associated to a specific formation and a map, called *group map*. The user should provide – or dynamically sketch – any desired shape for the group formation. This formation is then discretized into a set of points that can be seen as attraction points associated to agents – one point attracting each agent towards him. Analogous to an agent local map, the group map is then projected into the environment and its center is aligned with the center of the group of agents in the environment. The center of the formation shape is also aligned with the center of the group map.

Obstacles and goals are mapped to this map in the same way that we have done for the local maps. However, in the group map there is no view cone. Each agent in a group is mapped to its respective position on the group map as an obstacle. In order to obtain the information about the proximity of an agent in relation to the obstacles, we divide the group map into several regions surrounding the obstacles. Each cell in a region has a scalar value that is used to weight the distance between an agent and an obstacle. When the agent is in a cell associated to any of these regions, it will be influenced not only by the force exerted by the formation, but also by the gradient descent computed at its position in the group map. After that, the vector field is extracted and the agent motion is then influenced by two forces: formation force, and the *group map* vector field. As both forces influence in the path definition, they should be properly established, in order to avoid undesired or non realistic behaviors (for more details on combining these forces, refers to [15].

The motion of the agents which are inside the group map is established using these forces while the motion of the entire group is produced by moving the group map along the global map. For this, we consider the group map center point as a single agent and any path planner algorithm can be used to obtain a path free of collisions.

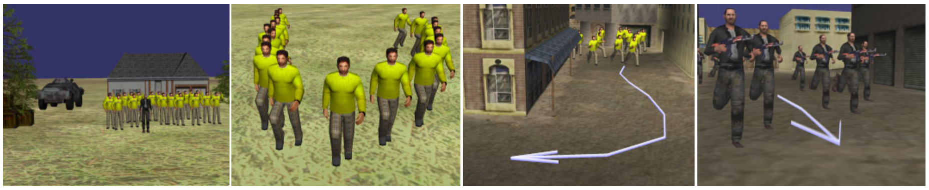

Fig. 3. Group control: agents can keep a formation or move freely; the user can interact and sketch trajectories to be followed by the groups

4 User Interaction with RTS Games

As seen in Section 3, our technique uses a global map to make a global path-planning, and local or group maps to avoid the collision with dynamic elements of a game (e.g. units, enemies, moving obstacles). This fits very well to most RTS games, where the player selects some units and click on a desired location in order to give a “go there” command.

This kind of game-play commonly requires several mouse clicks to give a specific behavior. RTS players commonly want that a group of units overcome an obstacle by following a specific path that conforms to his/her own strategy. Define a strategy in a high level of abstraction is generally hard to be done with only a few mouse clicks.

Based on the ideas of a previous work from our group [5], we suggested an interaction technique where the units are controlled by a sketch based navigation scheme, as an alternative to the global map . The player clicks with the mouse on the screen and draws the desired path for the currently selected units. The common technique of just clicking on the goal location is also supported. This way of sketching the path to be followed is the same one used in paint-like applications, and can be easily adapted to touch screen displays, like Microsoft Surface®and Apple iPad®, for instance.

4.1 Basic Implementation

The technique is divided in two steps: an *input capture* step, where the user draws the path to be followed; and a *path following* step, where the army units run to their goals. In the first step, the path can be drawn by dragging the mouse with a button pressed, or by dragging his/her finger over a touch screen surface. When the user releases the mouse button or removes his/her finger from the screen surface, a list of 2D points is taken and projected against the battlefield, resulting in a list of 3D points.

In the second step, the points generated on the first step are put in a list and used as intermediary goals for the units. Following this list, each point is used as the position goal for all selected units. When the first unit reaches this goal, it is discarded and the next one in the list is used. This continues until the first unit of the group reaches the last goal.

4.2 Splitting the Army

Imagine a situation where the player has walking soldiers and tanks available to attack, and the enemy headquarters is protected by one of these sides by a swamp. Only walking soldiers can walk through it. Then, the player may choose to attack the enemy by one side with the tanks, and the protected side with the walking soldiers.

An extension of the sketching technique may let the player use this strategy by simply drawing one path to the enemy headquarters, where in a certain division point the path divides into two parts: one goes through the swamp and the other avoids it. Then, all units (walking soldiers and tanks) follow the path, and in the division point, the army divides into two groups, one that can trespass the swamp and another that goes through mainland.

In order to allow this maneuver, we suggested a structure where the user sketches are stored in a tree. In this tree, each node represents a division point, start point or end point of the motion, and the links between nodes store one section of the path drawn by the player. As in the Section 4.1, each path section is stored as a list of 3D points.

When the player draws the first path, a node a is created for the 3D point where the path starts, and a node b where the path ends. A link storing all the points between a and b is created, with b being child of a. Then, the user draws a second path, starting nearly the point of division chosen by the player along the link. Considering a tree composed by the previously drawn paths, we search for the link l with the path section that contains the closest point to the start point of the newly drawn path. This link l is broken in two parts, l_1 and l_2, and a new end node p is created between both. Finally, we create a new end node c, a link between p and c, and attach the recently drawn path section to the tree.

5 Improvements and Speedup Using GPU

Solving the Laplace equation is the most expensive part of the BVP Path Planner algorithm. The iterative convergence process of an initial solution to an acceptable solution demands several iterations. So, if we want to improve the convergence speed, we must focus our attention in the relaxation process. We present an approach to improve the convergence on the local maps based on a GPU implementation.

The technique presented here is highly parallelizable, mainly the update of local maps and the computation of Laplace's equation. Each of these steps has several computations to be done, and it can be accomplished in parallel for each agent. We proposed an approach to implement it using nVidia CUDA®and will present it here assuming that the reader knows the CUDA architecture (a detailed explanation can be found in [6]).

Intuitively, each agent detects its observable obstacles in parallel, and each one has its own objective. So, the update of each local map can be also done in parallel. In our algorithm, each agent must seek for obstacles in its own view cone, setting the corresponding cells in the local map as "blocked". Note that, for a given agent, all other agents are seen as obstacles too. We assume that, in the beginning of this step, all the space occupied by the local map is free of obstacles. Then, for each agent, we launch one thread for each obstacle that the agent can potentially see. In each thread, we check if the obstacle is inside the view cone and inside the local map. If it is, then the thread sets each one of the cells that contains part of the obstacle with a tag "blocked".

This scheme of launching one thread for each obstacle of each agent fits very well in the nVidia CUDA architecture, where the processing must be split into blocks of threads. Assigning one block for each agent, each thread of the block

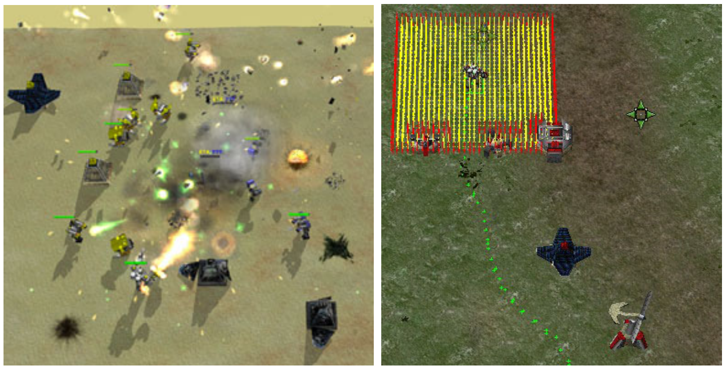
(a) (b)

Fig. 4. Two autonomous army fighting (a). The unit local map and the path produced by the BVP Path Planner, illustrated by green dots (b).

can look for one obstacle. Also, each thread can update the local map without synchronization, because all the threads will, if needed, update one cell from a "free" state to a "blocked" state.

After the update of the local maps with the obstacles position, we need to set the intermediate goal. For this, we simply need one thread per agent, each one updating the cells with a "goal" tag. This must be done sequentially to the previous step, to avoid race conditions and the need to synchronize all threads of each block of threads. When each local map has up-to-date information about what cells are occupied, free, or goals, we must relax it to get a smooth potential field. To do this, we assigned one local map to one block of threads in CUDA. In each block of threads, each thread is responsible for updating a value of potential to a single cell. Each thread stays in loop, with one synchronization point between the cells at the end of each iteration. Each thread updates the potential value of the cell using the Jacoby relaxation method.

Finally, to avoid unnecessary memory copies between the GPU and the main memory, we store each attribute of all agents in one single structure of contiguous memory. With this, some parameters that do not change so frequently (e.g. the local map sizes, and the current goal) can be sent only once to the GPU. When the new agent position must be computed, we fetch from the GPU only the current gradient descent on the agent positions.

With our GPU-based strategy we achieved a speed up to 56 times the previous CPU implementation. For a detailed evaluation of our results, refers to [6].

6 A RTS Game Implementation Using the BVP Path Planner

In order to demonstrate the applicability of the proposed technique, we implemented the BVP Path Planner in the Spring®Engine for RTS games. Spring is an open source and multi-platform game engine available under the GPLv2 license to develop RTS games. We have chosen this engine since it is well known in the RTS community and there are several commercial games made with it and available for use. With our planner, we can populate a RTS game with hundred of agents at interactive frame rates [6]. Figure 4(a) shows a screenshot of the game where two army are fighting using the BVP Path Planner. Figure 4(b) shows one unit and its local map with 33 × 33 cells. Red dots represent blocked cells, while the yellow dots represent free cells. The path followed by the unit is illustrated by green dots. An executable demo using the BVP Path Planner implemented with the Spring Engine can be found at `http://www.inf.ufrgs.br/~lgfischer/mig2010`, as well as a video including examples that demonstrate all the techniques presented in this paper. All animations in the video were generated and captured in real time.

7 Conclusion

This paper presented several complementary approaches recently developed by us to produce natural steering behaviors for virtual characters and to allow interaction with the agent navigation. The core of these techniques is the BVP Path Planner [14], that generates potential fields through a differential equation whose gradient descent represents navigation routes. Resulting paths are smooth, free from local minima, and very suitable to be used in RTS games.

Our technique uses a global and a local path planner. The global path planner ensures a path free of local minima, while the local planner is used to control the steering behavior of each agent, handling dynamic obstacles. We demonstrated that our technique can produce natural paths with interesting and complex human-like behaviors to achieve a navigational task, when compared with real paths produced by humans. Dealing with groups of agents, we shown a strategy to handle the path-planning problem while keeping the agent formations. This strategy enables the user to sketch any desirable formation shape. The user can also sketch a path to be followed by agents replacing the global path planner.

We also described a parallel version of this algorithm using the GPU to solve the Laplace's Equation. Finally, to demonstrate the applicability of the proposed technique we implemented the BVP Path Planner in a RTS game engine and released an executable demo available on the Internet.

We are now developing a hierarchical version of the BVP Path Planner that spends less than 1% of the time needed to generate the potential field using our original planner in several environments. We are also exploring strategies to use this planner in very large environments. Finally, we are also working on the generation of benchmarks for our algorithms.

References

1. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.: Interactive navigation of multiple agents in crowded environments. In: Proc. of the Symposium on Interactive 3D Graphics and Games, pp. 139–147. ACM Press, New York (2008)

2. Burgess, R.G., Darken, C.J.: Realistic human path planning using fluid simulation. In: Proc. of Behavior Representation in Modeling and Simulation, BRIMS (2004)
3. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. 22(2), 182–203 (2003)
4. Dapper, F., Prestes, E., Nedel, L.P.: Generating steering behaviors for virtual humanoids using BVP control. In: Proc. of CGI, vol. 1, pp. 105–114 (2007)
5. Dietrich, C.A., Nedel, L.P., Comba, J.L.D.: A sketch-based interface to real-time strategy games based on a cellular automation. In: Game Programming Gems, vol. 7, pp. 59–67. Charles River Media, Boston (2008)
6. Fischer, L.G., Silveira, R., Nedel, L.: Gpu accelerated path-planning for multi-agents in virtual environments. SB Games, 101–110 (2009)
7. Funge, J.D.: Artificial Intelligence For Computer Games: An Introduction. A. K. Peters, Ltd., Natick (2004)
8. James, J., Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: Magnenat-Thalmann, N., Thalmann, D. (eds.) CAPTECH 1998. LNCS (LNAI), vol. 1537, pp. 171–186. Springer, Heidelberg (1998)
9. Kallmann, M.: Shortest Paths with Arbitrary Clearance from Navigation Meshes. In: Symposium on Computer Animation, SCA (2010)
10. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Trans. on Robotics and Automation 12(4), 566–580 (1996)
11. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. Visual Comput. 20(10), 635–649 (2004)
12. Nieuwenhuisen, D., Kamphuis, A., Overmars, M.H.: High quality navigation in computer games. Sci. Comput. Program. 67(1), 91–104 (2007)
13. Pettre, J., Simeon, T., Laumond, J.: Planning human walk in virtual environments. In: Int. Conf. on Intelligent Robots and System, vol. 3, pp. 3048–3053 (2002)
14. Silveira, R., Dapper, F., Prestes, E., Nedel, L.: Natural steering behaviors for virtual pedestrians. Visual Comput. (2009)
15. Silveira, R., Prestes, E., Nedel, L.P.: Managing coherent groups. Comput. Animat. Virtual Worlds 19(3-4), 295–305 (2008)
16. Tecchia, F., Loscos, C., Conroy, R., Chrysanthou, Y.: Agent behaviour simulator (abs): A platform for urban behaviour development. In: Proc. Game Technology, 2001 (2001)
17. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. In: ACM SIGGRAPH, pp. 1160–1168. ACM Press, New York (2006)
18. Trevisan, M., Idiart, M.A.P., Prestes, E., Engel, P.M.: Exploratory navigation based on dynamic boundary value problems. J. Intell. Robot. Syst. 45(2), 101–114 (2006)