Tree Paths: A New Model for Steering Behaviors

Rafael Araújo Rodrigues[1], Alessandro de Lima Bicho[3], Marcelo Paravisi[1], Cláudio Rosito Jung[2], Léo Pini Magalhães[3], and Soraia Raupp Musse[1]

[1] Graduate Programme in Computer Science - PUCRS
Av. Ipiranga, 6681 - Building 32 - Porto Alegre/RS - Brazil
{rafael.rodrigues,marcelo.paravisi,soraia.musse}@pucrs.br
[2] Graduate Programme in Applied Computing - UNISINOS
Av. Unisinos, 950 - São Leopoldo/RS - Brazil
crjung@unisinos.br
[3] School of Electrical and Computer Engineering - UNICAMP
Av. Albert Einstein, 400 - Campinas/SP - Brazil
{bicho,leopini}@dca.fee.unicamp.br

Abstract. This paper describes a model for generating steering behaviors of groups of characters based on the biologically-motivated space colonization algorithm. This algorithm has been used in the past for generating leaf venation patterns and tree structures, simulating the competition for space between growing veins or branches. Adapted to character animation, this model is responsible for the motion control of characters providing robust and realistic group behaviors by adjusting just a few parameters. The main contributions are related with the robustness, flexibility and simplicity to control groups of characters.

1 Introduction

Behavioral animation has been proposed by Reynolds [1] in 1987, with the main goal to provide easy manners to control groups of individuals. Since that, many models have been proposed in order to provide different ways to steer behaviors of groups, as discussed in Section 2. However, in spite of all existing methods, the current state-of-the-art lacks of flexibility. In this context, we expect to provide different types of behaviors with simple changes in the proposed model: robustness, in order to provide realistic behaviors in low and medium density of people, and simplicity to control (only a small number of parameters are required). In addition, the steering behaviors should serve to control groups of any type of entities (fishes, birds and virtual humans) in virtual spaces.

The application of steering behaviors is very broad, including games, films, simulation tools, among others. Indeed, this technology can be applied in any situation in which mobile entities can be simulated. However, the scope of this paper is focused on groups of individuals and not crowds. The main point is that groups of individuals can only be recognized if density of people is not high, since group structures are not visible in highly dense crowds.

The main advantages of proposed model are robustness, simplicity and flexibility to control groups of individuals in virtual spaces, by populating the environment with markers points. Our model is inspired in a biological algorithm based on competition

Zs. Ruttkay et al. (Eds.): IVA 2009, LNAI 5773, pp. 358–371, 2009.

for space in a coherent growth of veins and branches [2]. We adapted this idea in order to generate motion behavior of groups, which also compete for space in order to move realistically, as described in Section 3. Another important contribution is the connection between two distinct areas such as steering behaviors and the algorithms for generating leaf venation patterns and tree structures.

The remainder of this paper is organized as follows: in next section we discuss some works found in literature, while in Section 3 we describe our model to steer behaviors of groups. Section 4 discusses some obtained results, and draws final considerations.

2 Related Work

Virtual groups have been studied since the early days of behavioral animation. Two seminal papers are related to models based on agents, which have some level of autonomy and individuality. Reynolds [1] simulated flocks of bird-like entities called *boids*, obtaining realistic animation by using only simple local rules. Tu and Terzopoulos [3] created groups of artificial fishes endowed with synthetic vision and perception of the environment, which control their behavior.

Researches are being conducted to the use of path planning algorithms associated to generation of realistic movements of the found path. Lavalle [4] introduced the concept of a Rapidly-exploring Random Tree (RRT) as a randomized data structure for path planning problems. An RRT is iteratively expanded by applying control inputs that drive the system slightly toward randomly-selected points. Choi et al. [5] proposed a model based on a probabilistic path planning and hierarchical displacement mapping to generate a sequence of realistic movements of a human-like biped figure to move from a given start position to a goal with a set of prescribed motion clips. Metoyer and Hodgins [6] proposed a method for generating reactive path following based on the user's examples of the desired behavior. Dapper et al. [7] proposed a path planning model based on a numerical solution for boundary value problems (BVP) and field potential formalism to produce steering behaviors for virtual humans. Rodríguez et al. [8] proposed a heuristic approach to planning in an environment with moving obstacles using dynamic global roadmap and kinodynamic local planning.

More specifically concerning groups motion, Kamphuis and Overmars [9] introduced a two-phase approach, where a path for a single agent (a backbone path) is generated by any motion planner. Next, a corridor is defined around the backbone path and all agents will stay in this corridor. Rodríguez et al. [10] proposed a model using a roadmap providing an abstract representation of global environment information to achieve different complex group behaviors that cannot be modeled with local information alone. Lien and collaborators [11] proposed ways using roadmaps to simulate a type of flocking behavior called shepherding behavior in which outside agents guide or control members of a flock. Data-driven models are quite recent in comparison with other methods, and aim to record motion in a pre-production stage or to use information from real life to calibrate the simulation algorithms. One example is proposed by Musse et al. [12] describing a model for controlling groups motion based on automatic tracking algorithms.

This paper proposes a new model for steering behaviors, where group behaviors can be easily calibrated, while keeping diversity of generated results. Considerations about the methods cited in this section and the proposed model are presented in Section 4.

3 The Proposed Model

This section describes our model to provide steering behaviors for groups of entities. First we discuss some aspects of our model[1], and then we focus on specific behaviors attained for groups control. Next, we present the original model of space colonization algorithm and performed adaptations for describing groups motion.

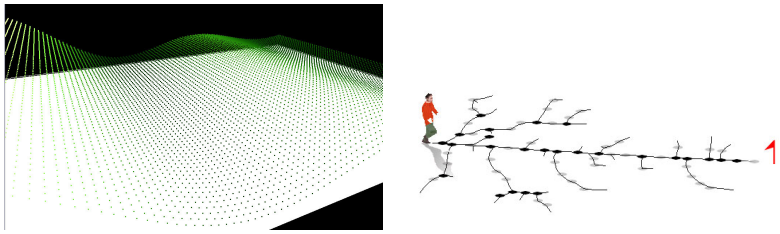
3.1 The Space Colonization Algorithm

The basic model for agents navigation is based on the space colonization algorithm, which has been previously used to develop leaf venation patterns [14] and tree structures [15]. The venation model simulates three processes within an iterative loop: leaf blade growth, the placement of markers in the free space, and the addition of new veins. The markers correspond to sources of the plant hormone auxin, which, according to a biological hypothesis, emerge in the growing leaf regions not penetrated by veins. A set of markers S interacts with the vein pattern, which consists of a set of points V called *vein nodes*. This pattern is extended iteratively toward the markers in the free space. The markers that are approached by the advancing veins are gradually removed, since the space around them is no longer free. As the leaf grows, markers in the free space are added in the space between existing veins and markers. This process continues until the growth stops, and there are no markers left. The interplay between markers in the free space and vein nodes is at the heart of the space colonization algorithm. During each iteration, a vein node is influenced by all the markers closer to it than any other vein node. Thus, veins compete for markers, and thus space, as they grow.

In the present paper, the venation model has been adapted to animate groups. Indeed, the key idea is to represent the space in a explicit way, using a set of markers (dots in the space). The markers define the "walkable space" through a discrete set of points, which are used to compute the paths of agents. These markers should be randomly distributed (according to a uniform probability density function) over the portions of the space that can be effectively occupied by the virtual agents, meaning that obstacles and other regions where agents should not move must not be filled with markers. Fig. 1(a) illustrates an environment populated with markers. The markers allow the organization and facilitates the steering of group behaviors, as discussed in next sections.

The amount of existing markers, as well as their position, have great impact in the generated trajectories. For instance, a higher density of markers uniformly distributed yields a wider range of movements, since there are more possibilities of trajectory for the entities. On the other hand, a lower density of markers tends to generate smaller number of trajectories, but presenting a smaller computational cost. Furthermore, the possibility of restricting use of markers have also impact in generated trajectories for agents and groups. Let is consider that markers are never restricted to only one path.

[1] Details are described in PhD thesis authored by Alessandro Bicho [13].



(a) (b)

Fig. 1. (a) Markers (dots) are discrete representation of walkable space; (b) Agent wants to go reach its goal, illustrated with a red flag. Darker dots describe key positions, lighter dots are related with each simulated step that generates path nodes, and between two path nodes there is the path segment.

In this case, different paths share markers, and it can cross in some region. On the other hand, if markers are temporarily restricted to only one path, more diversity in trajectories can emerge. This characteristic is very important in our model, since we can provide different behaviors by changing few aspects in the algorithm. The following sections describe further details of this model.

3.2 The Algorithm for Characters Motion

Our model based on space colonization algorithm can be used to provide path planning (from initial position to a fixed or mobile goal), as well as to direct motion of virtual agents without specific goals. The main difference between these two cases is that motion planning generates one or more coherent paths (called *tree paths* in this work) to reach a specific position, while the direct motion takes into account desired directions. So, in the last possibility, local minima are allowed, since a global path is never planned. The main application of direct motion is in situations where a specific goal is not possible, like group behavior for alignment, for instance.

Let I denote an agent in the group, having a position $\boldsymbol{p}(t)$ at each iteration t. Depending on the behavior to be applied, agents can have specific goals or not. If agents have an objective, its position at each time[2] t is denoted by $\boldsymbol{g}(t)$. Also, there is a *personal space* for each agent, modeled as a spherical region (with radius R), that represents a "perception field" that limits the range of markers which can be used by each agent.

Let G denote the tree path for an agent, computed by using the venation model proposed by Runions et al. [14] and adapted to group animation. The main difference between [14] and our approach is that, in [14], veins grow everywhere to occupy free space, while the proposed method guides the creation of branches toward the position of goal $\boldsymbol{g}(t)$ of the agent.

A tree path is a set of locations (key positions) organized in a directional graph. Each step created in the path corresponds to a *path node*, while a path segment joins two path nodes. In our model, paths from the current position of a given agent to its

[2] For sakes of clarity, the time index t will be removed from this point on, and used only when necessary.

goal are described through a tree, as illustrated in Fig. 1(b). In this figure, the goal is illustrated through a red flag, darker dots describe key positions where bifurcations occur, and lighter dots are related with each simulated step that generates path nodes. Between two path nodes there is a path segment, and all segments have the same length. Drawing all segments we can see the tree path generated from one agent to a specific goal.

Computing Tree Paths for Agents: The tree path G is computed through a two-stage algorithm within an iterative loop: i) markers processing, and ii) the addition of new path nodes. The markers in [14] correspond to sources of the plant hormone auxin, which emerge in the growing leaf regions not penetrated by veins, according to a biological hypothesis. For path planning, markers describe the "walkable space", in order to define the direction of a path node. During each iteration, a path node is influenced by all the markers closer to it than any other path node.

For a path node n, located at the position $\boldsymbol{n}$, the set of markers (located at positions $\boldsymbol{m}_i$) that are closer to n than any other node is denoted by

$$S(n) = \{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_N\}, \tag{1}$$

where N is the number of markers associated with node n. If $S(n)$ is not empty, a new path node n' will be created and attached to n by an edge representing a path segment. Each path node has an action region[3], that limits the markers that can be evaluated (if they are not closer to any other path node) in order to compute the new direction of growth of the tree path. Fig. 1(b) shows an example of tree path computed from an agent to its goal. To provide a diversity of branch orientations generated for G, increasing the space occupation, the markers closer to each segment are allocated to it, and they can not be used by other path segments from same agent I. On the other hand, the tree path G^* associated to another agent can use the same markers of the tree path G. Consequently, tree paths from different agents can intercept each other. However, this fact could bring a collision situation, which is not desirable. In order to deal with collision-free behaviors, we used the method for minimum distance enforcement among agents, proposed by Treuille et al. [16], and detailed in "Computing the Motion of Agents", in this section.

Mathematically, the algorithm for building the tree path is described as follows. Given a tree node n and a non-empty $S(n)$, a decision must be made whether a child node n' related to n will be created or not. This decision is made in such a way that nodes closer to the goal are more likely to have children, achieving a wider variety of paths in the vicinity of the goal. In fact, that the node n_g that is the closest to the goal will certainly have a child, guaranteeing that the goal will be reached by at least one trajectory. The probability $P(n)$ of any other node n having a child is given by:

$$P(n) = \frac{\|\boldsymbol{n}_g - \boldsymbol{g}\|}{\|\boldsymbol{n} - \boldsymbol{g}\|}, \tag{2}$$

where $\|\cdot\|$ is the L^2 norm of a vector. To decide whether a child will be created for n, a random variable ξ with uniform distribution in the interval $[0, 1]$ is generated, and the

[3] Its size can be calibrated, but normally we use the same radius R defined for the agent personal space.

the child is created if $\xi \leq P(n)$. Clearly, such probability rule prioritizes the creation of child nodes closer to the goal, since $P(n)$ decreases with its distance from the goal g. If a given node n is granted a child n', it will be created at a position $\boldsymbol{n}'$ through:

$$\boldsymbol{n}' = \boldsymbol{n} + \alpha \frac{\boldsymbol{d}(n)}{\|\boldsymbol{d}(n)\|}, \tag{3}$$

where $\boldsymbol{d}(n)/\|\boldsymbol{d}(n)\|$ is a unit vector representing the growth direction of the branch at node n, and α is a constant step that controls the length of the path segments. Vector $\boldsymbol{d}(n)$ is obtained based on the markers in $S(n)$ and their coherence to the goal $\boldsymbol{g}$:

$$\boldsymbol{d}(n) = \sum_{k=1}^{N} w_k (\boldsymbol{m}_k - \boldsymbol{n}), \tag{4}$$

where

$$w_k = \frac{f(\boldsymbol{g} - \boldsymbol{n}, \boldsymbol{m}_k - \boldsymbol{n})}{\sum_{l=1}^{N} f(\boldsymbol{g} - \boldsymbol{n}, \boldsymbol{m}_l - \boldsymbol{n})} \tag{5}$$

are the weights of the markers computed based on a non-negative function f. This function should prioritize both markers that lead to the goal, those that are closer to the current node n. Our choice for f satisfying these conditions is

$$f(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} \dfrac{1 + \cos\theta}{1 + \|\boldsymbol{y}\|} = \dfrac{1}{1 + \|\boldsymbol{y}\|} \left(1 + \dfrac{<\boldsymbol{x}, \boldsymbol{y}>}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}\right), & \text{if } \|\boldsymbol{x}\|\|\boldsymbol{y}\| > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where θ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$, and $< \cdot\,,\, \cdot >$ denotes the inner product. It can be observed that f decreases as the angle between $(\boldsymbol{g} - \boldsymbol{n})$ and $(\boldsymbol{m}_k - \boldsymbol{n})$ increases (so that markers that are aligned with the goal carry a larger weight), and also as $\|\boldsymbol{m}_k - \boldsymbol{n}\|$ increases (so that markers closer to the node carry a larger weight as well). If the number of markers is large, $\boldsymbol{d}(n)$ will point approximately toward the goal (in fact, it can be shown that $\boldsymbol{d}(n)$ points directly toward the goal if the number of markers grow to infinity). However, if the amount of markers is small, $\boldsymbol{d}(n)$ may deviate from the goal, generating a variety of paths. The procedure described so far creates a sequence of nodes connected by path segments, but no bifurcations in the tree. To create bifurcations, one father node n must be connected to at least to two different children nodes n'_1 and n'_2. When the first child node n'_1 is created, it retrieves the markers around it according to a "restriction distance" (so that the number of markers available to n is reduced). Then, $S(n)$ is re-computed with this reduced set of markers, and Equations (2)-(3) are re-applied to obtain a second child node n'_2. This node also retrieves the markers around it, and the process for creating child nodes for n is repeated until there are no markers available to n. It should be noticed that, at each iteration, a new random variable ξ is created and compared to the probability $P(n)$ given in Equation (2) to decide if node n will have a child or not. Hence, some nodes may have more children than others (usually, the number of children increases near the goal). In fact, some nodes may have just one child, and do not present any bifurcation at all. Nodes that present at least two

children are called “key positions”, and there is a unique path between adjacent key positions (see darker dots in Fig. 1(b)).

Fig. 2 shows the generation of branches in the proposed model. The green branch (from green to black path nodes) is created first, and then some markers are restricted (due to restriction distance). The blue path node is then computed with the remaining markers associated to the father node n (black one), and the new node also restrict the markers around it. The same thing happens with the branch from black to red path nodes. Other branches could be still generated from the same father node (since there are remaining markers), but they were not created in this example because of the probability function $P(n)$. One difference from Runions' model to ours is the treatment of restricted markers. In Runions' model, markers around a newly created node are really removed and placed afterwards in data structure. On the other hand, our markers are only not available in the data structure for the agent to which the tree path is being computed, i.e. they are not deleted from data structure and can be used for other tree paths.

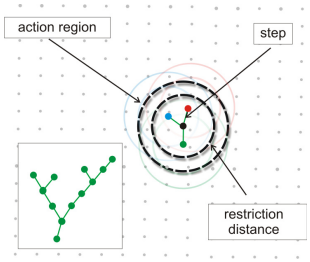


Fig. 2. New branches are generated and they restrict markers within the restriction distance

While a tree path is being computed, the agent is able to walk along the generated paths. Although other path planning algorithms can also be used (such as $A*$ search algorithm), the proposed model presents some advantages. First, for group behaviors (follow, escape and collaborative actions such as surround behavior), one important aspect is the diversity of paths (it is desirable that a group walks to a specific goal by occupying the space with different possible paths). Another interesting aspect is that we are able to recompute our trajectories from the needed position, e.g. when the target of the agent (in follow behavior) crosses the tree path, we can recompute the path from the intersection point. The next section describes how the agents move along the tree, after the path has been computed.

Computing the Motion of Agents: Tree paths provide local goals for agents. However, an important challenge in groups motion should be treated in this model: collision avoidance. As mentioned before, tree paths G and G^* related to different agents can share markers, consequently agents walking in tree paths can collide, passing by closer (or the same) path nodes. To address this problem, we used the method for minimum distance enforcement among agents, proposed by Treuille et al. [16]. Their

method proposes iterations over all pairs of agents within a *threshold distance*, symmetrically pushing them apart, so that the minimum distance is enforced, as describes in Equation (7).

$$\boldsymbol{d}_{\text{enf}}(\boldsymbol{p_i}) = \sum_{\{j | d_{\text{i,j}} < \boldsymbol{t}_{\text{min}}\}} \frac{\boldsymbol{p_i} - \boldsymbol{p_j}}{2}, \tag{7}$$

where $\boldsymbol{t}_{\text{min}}$ is minimum threshold distance allowed and $\boldsymbol{d}_{\text{i,j}}$ is the distance between agents i and j. Consequently, modifying the position of one character impacts all other characters. We implemented a translation for pairs of agents which are closer than the minimum distance. Indeed, this strategy does not present any compromise in the computational time for small number of agents (main focus of this work). Anyway, more details about computational time are discussed later.

There are two manners to provide agents motion. The first one is used when agents have specific goal location; the second, when there are not goals. In the latter case agents are affected by the markers, which can have different weights. Indeed, markers are used as discretized information of the available space, but also they can affect differently the motion of agents, depending on associated weights. Given an agent I at the position $\boldsymbol{p}(t)$ and goal $\boldsymbol{g}(t)$, at each time iteration t and given its computed tree paths G, the next agent position is computed by:

$$\boldsymbol{p}(t+1) = \boldsymbol{p}(t) + \beta \frac{\boldsymbol{d}(\boldsymbol{p}(t))}{\|\boldsymbol{d}(\boldsymbol{p}(t))\|} + \boldsymbol{d}_{\text{enf}}(\boldsymbol{p}(t)), \tag{8}$$

where β is a constant that controls the length of the agent step, $\boldsymbol{d}(\boldsymbol{p}(t))$ is an orientation vector from the agent's current position $\boldsymbol{p}(t)$ to the next path node coming from tree path, indicating its local goal (and attaining the global goal $\boldsymbol{g}(t)$). Also, we compute $\boldsymbol{d}_{\text{enf}}(\boldsymbol{p}(t))$, a result vector for minimum distance enforcement among agents, proposed by Treuille et al. [16] to avoid collisions.

In the goal-based motion of the agent, which includes the tree path computing, three events should be iteratively managed:

1. Agent's decision: when an agent reaches a key position, it should take a decision to which tree branch it should follow. This decision is considered taking into account how close to the goal the tree branch brings the agent.
2. Branch death: There are two reasons to remove the branches in a tree: *i)* When a branch was not chosen by the agent (last item), the branch and its children are removed, and *ii)* When the goal changes position, the branches into a distance from the goal are removed and then recomputed to taking into account the new goal position (defined by *recomputing distance*).
3. Branch reaches agent's goal: tree path stops growing, but the agent keeps walking along the paths until it reaches the goal.

In addition to goal-based motion, there is another manner to compute the motion agents that is useful when goals are not explicit, e.g. in formation and alignment behaviors. In this case, there is no goal vector, but an agent motion direction $\boldsymbol{md}$ which is computed based on a variation of the weights of the markers within the agent's personal space.

Given an agent I at the position $\boldsymbol{p}(t)$ at each iteration t, and given the set of M markers $S(\boldsymbol{p}(t)) = \{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_M\}$ within the personal space of the agent I (all markers are considered), we first find the set $S'(\boldsymbol{p}(t))$ of M orientation vectors from agent I to all the markers in $S(\boldsymbol{p}(t))$, in order to compute the agent direct motion $\boldsymbol{md}$:

$$S'(\boldsymbol{p}(t)) = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_M\}, \quad \text{where} \quad \boldsymbol{v}_k = \boldsymbol{m}_k - \boldsymbol{p}(t). \tag{9}$$

The motion direction $\boldsymbol{md}$ of the agent is computed similarly to Equation (4), where w_k is a weight associated with orientation vector $\boldsymbol{v}_k$, calculated according to the desired behavior (see Sections 3.4 and 3.5 for details):

$$\boldsymbol{md} = \sum_{k=1}^{M} w_k \boldsymbol{v}_k, \tag{10}$$

Therefore, the next agent position is given by:

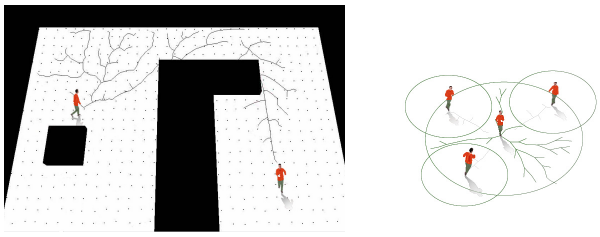
$$\boldsymbol{p}(t+1) = \boldsymbol{p}(t) + \beta \frac{\boldsymbol{md}}{\|\boldsymbol{md}\|} + \boldsymbol{d}_{\text{enf}}(\boldsymbol{p}(t)), \tag{11}$$

where β and $\boldsymbol{d}_{\text{enf}}(\boldsymbol{p}(t))$ are the same parameters used in Equation (8). Next, we present some examples of behaviors that can be obtained using the two strategies for the motion of agents described in this Section: based on goals and based on directions.

3.3 Behaviors: Pursue, Escape and Surround the Goal

The best way to describe the pursuit action takes into account a path between only two individuals (one follower and one target agent). Moreover, as target location can change dynamically, and the path between follower and target should be re-computed iteratively (as described in last section). Fig. 3(a) illustrates such behavior.

In this case, once the target agent crosses the tree path of a follower agent, the tree path is recomputed from the intersection point. In the case of a group of agents following the same target, it is desirable to provide emergent behavior of surround. In this



(a) (b)

Fig. 3. (a) Pursue behavior: one agent tries to reach another one. The illustrated tree path shows the way for follower agent achieve the target; (b) Escape behavior: one agent tries to escape from a circular region, in which three follower agents are included.

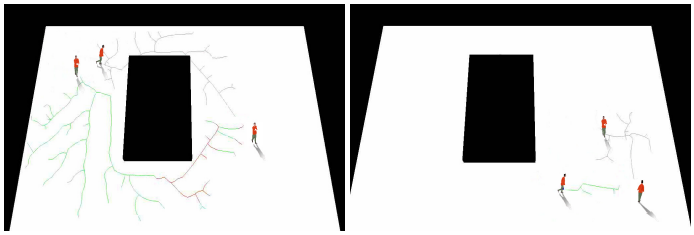


Fig. 4. Surround behavior: two agents try to reach another one. It is possible to observe the diversity of generated tree paths causing the surround behavior.

case, a simple change in the rule to allocate markers can be used to obtain interesting results: the emergent behavior is defined by all markers used in a tree path being allocated to such path and not being shared by other path. Consequently, other agents should fight for space, by trying to find other possibilities to move. Fig. 4 illustrates this behavior.

Finally, another small change in the path planning algorithm can provide escape behavior. In path planning, the path node is accepted in tree path when it brings the agent closer to its goal. In a escape behavior, all agents should try to get away from a pre-defined position $\boldsymbol{c}$. To cope with this condition, a path n node should be accepted (according to the probabilistic function in Equation (2)) when it gets agents far from the $\boldsymbol{c}$. In fact, that can be accomplished by replacing Equation (2) with $P(n) = \frac{\|\boldsymbol{n}-\boldsymbol{c}\|}{\|\boldsymbol{n}_c-\boldsymbol{c}\|}$, where $\boldsymbol{n}_c$ is the node located the farthest away from $\boldsymbol{c}$. Fig. 3(b) illustrates such behavior. In this figure, the escape region is automatically computed based on three follower agents, and consequently the tree path algorithm tries to bring agent outside such region.

3.4 Behavior: Groups Formation

This behavior aims to provide the formation of specific shapes, which is relevant in several entertainment applications. For instance, games and movies, as well as theatrical performances, can use such characteristics to provide group motion. There are at least two different ways to model such behavior. The predefined one considers the generation of specific goals into a shape, and the posterior distribution of the individuals into the group. The drawback of this approach is the low flexibility if a shape changes dynamically, or if more agents want to participate in the performance, since it requires the recomputation of specific goals for each agent. The second approach describes an emergent behavior of agents in order to occupy the space corresponding to the desired shape. We adopted the last approach in our model by using markers in the space.

Initially, a shape region should be defined (as illustrated in Fig. 5, where the shapes are the letters V and H). It can be done by using our markers spray. At the beginning, the environment has markers to allow the agents motion. Then, the user can spray markers to define the shape formation. Consequently, markers are painted over the environment, increasing the density of markers in formation shape. Yet, the markers into the target shape have increased weight to a defined constant (we experimentally set this value to 10). The consequence is that markers into the target shape will have more importance in Equation (10) than markers outside.

(a) (b) (c)

Fig. 5. Agents form explicit shapes of letters V and H

The algorithm is described in two steps. The first one takes into account the tree path algorithm described previously. In this case, an automatic goal into the target shape is attributed for each agent in the simulation, providing motion stimulus for each individual. The automatic goal takes into account the bounding box of the target shape, and it is selected as a random position within it. When the agent gets close to the shape region (it is identified through markers analysis, i.e. if one agent has a target marker into its personal space), the orientation vector computed in tree paths algorithm is not any more taken into account. In other words, Equation (8) is used initially to guide the motion of agents, and then it is replaced by a modified version of Equation (11), using the following weights w_k to obtain the motion direction $\boldsymbol{md}$:

$$w_k = \begin{cases} 10, & \text{if } \boldsymbol{m}_k \text{ is within the target shape} \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

When other agents arrive in formation shape, they fight for space, but they tend to keep inside the formation shape, since the weight of the markers is greater than outside target shape markers. Moreover, if all agents should present same specific orientation in target shape, their distribution is easily regulated after the agent entries in the shape region. The final distribution of agents for the VH shape is given in Fig. 5(c).

3.5 Behavior: Groups Alignment

This behavior, as the one described in last subsection, is useful to provide group performances in entertainment applications. Alignment of people is an interesting feature that can be used in several applications. In our model, we are able to having people moving based on specific goals (using tree path algorithm - Section 3.2) as well as without goals, by changing the weights of markers into agent personal space, and then generating the motion of agents, as in last section.

For our group formation, we are able to create alignment regions with predefined weight masks for markers. One possible mask used to provide horizontal and vertical alignment is illustrated in Fig. 6, on the left. In this case, the markers into the formation mask have their weights increased according to $w_k = \text{dist}_{(I,k)}$, where $\text{dist}_{(I,k)}$ is the Euclidean distance from the current position of agent I to the marker $\boldsymbol{m}_k$. These weights are used to obtain the motion direction $\boldsymbol{md}$ used in Equation (11). If a given marker is within the personal space of more than one agent, these agents compete for the marker. In fact, the same marker may present different weights when viewed by different agents,

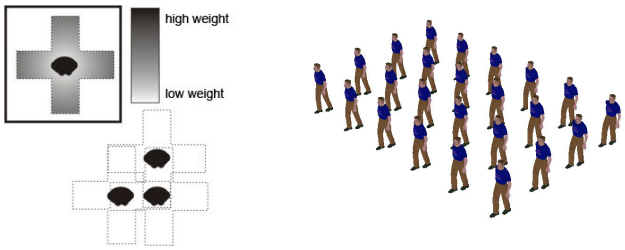


Fig. 6. On the left: weight masks are used in order to define alignement behavior. On the right: an example of generated behavior.

depending on their goals and relative position w.r.t. the marker. To minimize the chance of more than one agent reaching the same marker at the same time, the weight of the marker is increased for the agent that is the closest to it. More specifically, this weight is recomputed as the sum of the weights of that specific marker as viewed by all agents having the marker within their personal spaces, so that the closest agents tends to reach the marker faster than the others.

4 Final Considerations

This paper presented a new algorithm to provide motion of groups of agents. It is based on biologically-inspired technique used in the past for generating leaf venation patterns and tree structures, simulating the competition for space between growing veins or branches. In this paper, we presented some behaviors to simulate groups of agents, such as group path planning (we called tree paths), pursue behavior, surround behavior, escape behavior, groups formation and groups alignment. The key innovation is the simple way in which the paths are created, by "observing" and "occupying" free space, which is represented using a set of marker points, which leads to a simple yet computationally effective implementation of the competition for space. Global tree path is modeled by biasing the influence of the captured marker points according to their agreement with each agent's direction to its goal, which can be assigned to individual agents or groups. In addition, agents motion can be goal-based or, influenced by variation of weights attributed to markers, originating alignement and formation behaviors.

In terms of computational performance, such analysis is very dependent of generated tree paths, which also take into account the simulated environment (obstacles, number of agents, distance to the goal). In Fig. 7 we show the time[4] consumed for 1, 5 and 10 agents, including the number of iterations performed in the simulation. It is important to note that maximum number of iterations observed in our simulations is 49, meaning that at most 49 iterations are required by the agents to reach their goals in the simulated environment. Since our model is focused on low density scenarios, we have tested a

[4] Average of 20 simulated experiments; these results were obtained using monothread implementation without characters' rendering on Intel® Core™ 2 Duo 2.2GHz, 3GB DDR2 at 667MHz and NVIDIA® GeForce® 8400M GS 128MB.

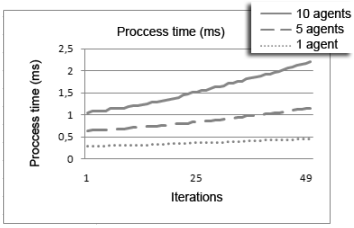


Fig. 7. Computational time (ms) for processing 1, 5 and 10 agents, considering the evolution of tree paths

maximum of 50 agents to better visualize the groups behavior, achieving real-time performance. However, according to the worst possibility (49 iterations to reach the goal) in Fig. 7, it is theoretically possible compute tree paths for thousands of agents.

Comparing with other models, while a RRT explores uniformly the "walkable space", defining positions randomly in this space that will guide the expansion of the search tree, in the proposed model the space is previously discretized using a uniform distribution. Once known the space, the proposed model considers weighted positions so that the growth of branches is directed to the goal. This difference allows optimizing the construction of the tree and the search for the path in this structure. Compared to roadmaps, the proposed model allows the generation of a connected tree using a small amount of edges. In a roadmap, the performance of the connected graph is impaired because to the number of edges necessary to explore the entire space. The proposed model also requires single nearest-neighbor queries, while roadmaps require more-expensive k-nearest neighbor queries. As future work, we intend to provide other group behaviors, focused on individualities, such as agents skills. Also, we are interested on integrating our model with crowds techniques, providing an adaptative framework in which methods can be applied depending on people density.

Acknowledgments

This work was developed in collaboration with HP Brazil R&D and Brazilian research agencies CNPq, FINEP and CAPES.

References

1. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. In: Proceedings of SIGGRAPH 1987, NY, USA, pp. 25–34 (1987)
2. Sachs, T.: Polarity and the induction of organized vascular tissues. Annals of Botany 33(2), 263–275 (1969)
3. Tu, X., Terzopoulos, D.: Artificial fishes: physics, locomotion, perception, behavior. In: Proceedings of SIGGRAPH 1994, NY, USA, pp. 43–50 (1994)
4. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Technical Report TR98-11, Dep. of Computer Science, Iowa State University (1998)

5. 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)
6. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. The Visual Computer 20(10), 635–649 (2004)
7. Dapper, F., Prestes, E., Nedel, L.P.: Generating steering behaviors for virtual humanoids using bvp control. In: Proc. of Computer Graphics International, RJ, Brazil, pp. 105–114 (2007)
8. Rodríguez, S., Lien, J.M., Amato, N.M.: A framework for planning motion in environments with moving obstacles. In: IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems, November 2007, pp. 3309–3314 (2007)
9. Kamphuis, A., Overmars, M.H.: Finding paths for coherent groups using clearance. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 19–28. Eurographics Association, Switzerland (2004)
10. Rodríguez, S., Salazar, R., McMahon, T., Amato, N.M.: Roadmap-based group behaviors: Generation and evaluation. Technical Report TR07-004, Dep. of Computer Science, Texas A&M University (2007)
11. Lien, J.M., Rodríguez, S., Malric, J.P., Amato, N.M.: Shepherding behaviors with multiple shepherds. In: Proceedings of the IEEE Inter. Conf. on Robotics and Automation, pp. 3402–3407 (2005)
12. Musse, S.R., Jung, C.R., Jacques Jr., J.C.S.: Using computer vision to simulate the motion of virtual agents. Computer Animation and Virtual Worlds 18(2), 83–93 (2007)
13. de Lima Bicho, A.: From Plants to Crowd Dynamics: A bio-inspired model (in portuguese, to be published). PhD thesis, State University of Campinas, Campinas, Brazil (July 2009)
14. Runions, A., Fuhrer, M., Lane, B., Federl, P., Rolland-Lagan, A.-G., Prusinkiewicz, P.: Modeling and visualization of leaf venation patterns. ACM Trans. Graph. 24(3), 702–711 (2005)
15. Runions, A., Lane, B., Prusinkiewicz, P.: Modeling trees with a space colonization algorithm. In: Proc. of the Euro. Workshop on Natural Phenomena, Prague, Czech Republic, September 2007, pp. 63–70 (2007)
16. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Trans. Graph 25(3), 1160–1168 (2006)