PERGAMON

Computers & Graphics 25 (2001) 983–998

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Scalable nonlinear dynamical systems for agent steering and crowd simulation

Siome Goldenstein[a,1]*, Menelaos Karavelas[b], Dimitris Metaxas[a], Leonidas Guibas[b], Eric Aaron[a,2], Ambarish Goswami[c]

[a] *Computer and Information Science Department, University of Pennsylvania, Pennsylvania, USA*
[b] *Computer Science Department, Stanford University, USA*
[c] *Discreet, USA*

Abstract

We present a new methodology for agent modeling that is scalable and efficient. It is based on the integration of nonlinear dynamical systems and kinetic data structures. The method consists of three layers, which together model 3D agent steering, crowds and flocks among moving and static obstacles. The first layer, the *local layer* employs nonlinear dynamical systems theory to models low-level behaviors. It is fast and efficient, and it does not depend on the total number of agents in the environment. This dynamical systems-based approach also allows us to establish continuous numerical parameters for modifying each agent's behavior. The second layer, a global *environment layer* consists of a specifically designed kinetic data structure to track efficiently the immediate environment of each agent and know which obstacles/agents are near or visible to the given agent. This layer reduces the complexity in the local layer. In the third layer, a *global planning laye*r, the problem of target tracking is generalized in a way that allows navigation in maze-like terrains, avoidance of local minima and cooperation between agents. We implement this layer based on two approaches that are suitable for different applications: One approach is to track the closest single moving or static target; the second is to use a pre-specified vector field, which may be generated automatically (with harmonic functions, for example) or based on user input to achieve the desired output. We also discuss how *hybrid systems* concepts for global planning can capitalize on both our layered approach and the continuous, reactive nature of our agent steering.

We demonstrate the power of the approach through a series of experiments simulating single/multiple agents and crowds moving towards moving/static targets in complex environments.

1. Introduction

Modeling autonomous digital agents and simulating their behavior in virtual environments is becoming increasingly important in computer graphics. In virtual reality applications, for example, each agent interacts with other agents and the environment, so complex real-time interactions are necessary to achieve non-trivial behavioral scenarios. Modern game applications require smart autonomous agents with varying degrees of intelligence to permit multiple levels of game complexity. Agent behaviors must allow for complex interactions, and they must be adaptive in terms of both time and space (continuous changes in the environment). Finally, the modeling approach should scale well with the complexity of the environment geometry, the number and intelligence of the agents, and the various agent–environment interactions.

There have been several promising approaches towards achieving the above goal. Many of them, however, are restrictive in terms of their application domain. They do not scale well with the complexity of the environment. They do not model time explicitly. They do not guarantee that the desired behavior will always be exhibited. This paper presents an alternative:

*Corresponding author.

[1] Supported by Brazil's "Conselho Nacional de Pesquisa e Desenvolvimento" (CNPq).

[2] Partially supported by NSF grant NSF-SBR 8920230.

0097-8493/01/$ - see front matter
PII: S0097-8493(01)00153-4

a scalable, adaptive, and mathematically rigorous approach to modeling complex low-level behaviors in real time.

We employ nonlinear dynamical system theory, kinetic data structures, and harmonic functions in a novel three-layer approach to modeling autonomous agents in a virtual environment. The first layer consists of differential equations based on nonlinear dynamic system theory, modeling the low-level behavior of the autonomous agent in complex environments. In the second layer, the motions of the agents, obstacles, and targets are incorporated into a kinetic data structure, providing an efficient, scalable approach for adapting an agent's motion to its changing local environment. In the third layer, differential equations based on harmonic functions determine a global course of action for an agent, initializing the differential equations from the first layer, guiding the agent, and keeping it from getting stuck in local minima. We also discuss how *hybrid systems* concepts for global planning can capitalize on both our layered approach and the continuous, reactive nature of our agent steering.

In the first layer, we characterize in a mathematically precise way the behavior of our agents in complex dynamic virtual environments. The agents exist in a real-time virtual environment consisting of obstacles, targets, and other agents. Depending on the application, agents reach one or multiple targets while avoiding obstacles; targets and obstacles can be stationary and/or moving. Further, the inclusion of time as a variable in our system makes the formulation efficient, natural and powerful compared to traditional artificial intelligence (AI) approaches.

Our agent modeling is based on the coupling of a set of nonlinear dynamical systems. The first dynamical system is responsible for the control of the agent's angular velocity. It uses carefully designed *attractor* and *repeller* functions for targets and obstacles (respectively). Due to the nonlinearity of these functions, a direct summation can generate undesired attractors that would lead to collisions and other unsuitable behaviors. To remedy this problem, we use a second nonlinear dynamical system that automatically computes the correct weighted contribution of the above functions. A third dynamical system controls the agent's forward velocity.

Each agent is described by its position, geometrical shape, heading angle, forward velocity, and personality. To model low-level personality attributes, (e.g., agility, aggressiveness), we extend the above set of equations through additional parameterization of the governing equations. This results in different low-level agent behaviors, depending on the environment configuration. Agents with different personalities will react differently in the same environment, given the same set of initial conditions. Our approach is general; other low-level behaviors can be easily modeled if needed for a given application.

The nonlinear dynamical systems that model changes in agent attributes are based on local decisions. To avoid local minima, and we use *harmonic function* theory to generate a *nominal* global trajectory based on the static objects in the environment. Harmonic functions are solutions to the *Laplace equation*; they create an artificial potential field for the agent to follow. The underlying mathematical theory [1,2] (used in fluid dynamics, electrostatics, and other engineering fields) guarantees that the minima of the field occur only at the boundary of the agents' configuration space, i.e., a global minimum. The nominal solution is then used to compute the agent's motion at each time instant.

In addition to the "intrinsic" personality attributes of each agent, an agent's behavior is affected by "extrinsic" factors in its immediate environment, such as obstacles, other friendly or hostile agents, etc. In order to make our approach efficient and scalable with the number of agents and the geometric complexity of the environment, we use *kinetic data structures* (KDS) to track the immediate environment of each agent and know which obstacles/agents are near or visible to the given agent. Kinetic data structures [3,4] are a general framework for designing algorithms tracking attributes of a continuously evolving system so as to optimally exploit the continuity or coherence of the motion. A KDS works by caching a certain set of assertions about the environment, certifying the value of the attribute of interest, and then maintaining this assertion cache and the attribute of interest as assertions fail.

We demonstrate the power of our approach through a series of examples that exhibit continuously adaptive low-level behaviors, including flocking. The formulation lends itself naturally to parallelism. It may also be used as a basis for modeling high-level behaviors.

The paper is organized as follows. In Section 2, we list some of the related work in the field, and how it relates to our approach. In Section 3, we describe our basic framework. Sections 4, 5 and 6.1.1 describe our framework in detail. Section 7 explains the demos that are in the video, and Section 8 contains conclusions and directions for future work.

2. Related work

Modeling autonomous agents has been an active area of research in CG since the pioneering work of Reynolds [5]. Since then, methodologies from many fields have been employed to address this problem [6] and have been used in several do-mains including game applications [7–10]. Artificial intelligence (AI) techniques have shown very promising results [11–14] generally require complex inferencing mechanisms and explicit rules, and

they make the explicit modeling of time very difficult. Typically, such methods do not scale well with the number of agents or obstacles, especially when each agent has a different set of governing rules. On the other hand, learning, perception, and dynamics-based techniques [15–21] can easily adapt to dynamically changing environments. Some initial attempts to use nonlinear dynamical systems for low-level behavior modeling have appeared recently in [22,23]. Methods from computational geometry have also been employed [24,25].

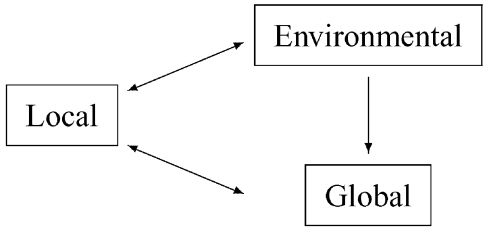
Different systems and approaches have been developed to add distinct behaviors to autonomous agents [13,17,26–28]. The usefulness of a layered approach to modeling low and high-level behaviors has been advocated by several researchers [29–31].

Behavioral robotics deals with similar issues, usually dealing with only two-dimensional environments [32–34].

In this paper, we present a novel integration of nonlinear dynamical systems, kinetic data structures and harmonic functions. This results in a layered, efficient, and scalable approach to modeling low-level behaviors, higher-level planning, and smart environment management.

3. Framework design

We describe a three-layer framework for autonomous agent modeling and motion generation. Each layer is responsible for a different stage of the process, and has to communicate with the others. Different approaches can be taken for each layer, depending on the particular application.



We call the three layers *local*, *environmental*, and *global*.

Local: This layer is responsible for basic reactive actions: obstacle avoidance and moving in the desired direction. This stage needs to be fast, scalable, robust, and parameterized to control the particular way it does these basic tasks. For the system to be scalable, this layer cannot depend on the number of elements in the environment.

Environment: This layer has knowledge about the spatial layout (e.g., agents, obstacles) of the environment. It tracks each agents' immediate environment, which in turn is used to determine the agent's behavior. Such information can then be transmitted to and shared by other agents, so that each agent can build a more complete model of its environment. This layer is also responsible for collision detection, and for notification to the agents involved in it, so that appropriate responses can be generated.

Global Planning: Since the local layer is only concerned with very localized operations, another layer must give the agents some more general knowledge of their environment. The global planning layer gives each agent the general direction it should follow at each point in time. This global layer may be interpreted as a time-varying adaptive flow field, used in the local layer as an educated guess of where each agent should try to go to achieve its goal.

The global planning layer can also alter agents' attributes to generate higher-level behaviors. For instance, an agent can get "tired" as time goes by, moving more slowly or in a more clumsy way. Or, if a critical task must be achieved no matter the cost, an agent might allow itself to get closer to moving obstacles, increasing the risk of being involved in a collision.

In a cooperative scenario, the global planning layer is also responsible for organizing the actions of the individual agents, integrating them toward a common goal. In a pursuit application, for example, it might send some agents directly after the escaping target while sending others in a path to surround the target. Different levels of reasoning, such as graph theory results and computational geometry techniques, can be employed to achieve a chosen strategy.

4. Local system: nonlinear dynamical system

We model the low-level movement of every agent by a nonlinear dynamical system. The movement of an agent at any instant is defined by its heading angle and forward velocity. An agent's behavior is modeled through a parameterization of the nonlinear systems governing the agent's movements.

We first present our model of an agent's heading angle, which is defined through a pair of coupled nonlinear dynamical systems. We then present the modeling of the agent's forward velocity. Our methodology is an adaptation of nonlinear dynamical system theory first used in behavioral robotics [33,34]; intuitively, it amounts to computing an agent's motion so that it always turns away from obstacles and moves in the direction of the targets. The agent's behavior is adaptive and intelligent, since it dynamically computes the varying contributions of the surrounding objects to determine its path towards a target.

4.1. Agent heading angle

The agent's heading angle, ϕ, is computed by a nonlinear dynamical system of the form

$$\dot{\phi} = f(\phi, \mathbf{env}), \tag{1}$$

where f is a nonlinear function and **env** represents the state of the environment, i.e., the position and size of obstacles, targets, and other agents.

The function f is constructed so that each agent avoids obstacles and pursues the desired direction. The first step in the design of f is to model the contribution of obstacles and target direction. We model their contribution so that the direction of an obstacle will act as a *repeller* and the goal direction will act as an *attractor*.

According to the dynamical system in (1), attractors and repellers are located at the zero crossings of f. At these points, an attractor or repeller property will be determined by the derivative of f with respect to ϕ. In both cases, at the zero crossing points $\dot{\phi} = 0$. These points are called *critical points* or *fixed point*s, since ϕ does not change at these points.

The fixed point behaves as an attractor or repeller, depending on the value of the derivative of f with respect to ϕ, i.e., $\partial f/\partial\phi = \partial\dot{\phi}/\partial\phi$. If the derivative is positive, any ϕ slightly off of the critical point will deviate farther away with time. This is a repeller. The inverse situation occurs when the derivative is negative. Values of ϕ slightly away from the critical point will converge towards it. This is an attractor. Shiftings of these functions will place repellers and attractors to the specified positions of obstacles and goal direction, respectively.

Fig. 1 shows several parameters in a situation involving an agent and two obstacles. ψ_i is the angle of obstacle i with respect to the horizontal, $2\Delta\psi_i$ is the angle subtended by obstacle i on the agent and r_i is the distance between obstacle i and the agent. We will use these parameters to define a repeller and an attractor function that correspond to the targets' and obstacles' contribution to f, f_{tar} and f_{obs}, respectively.

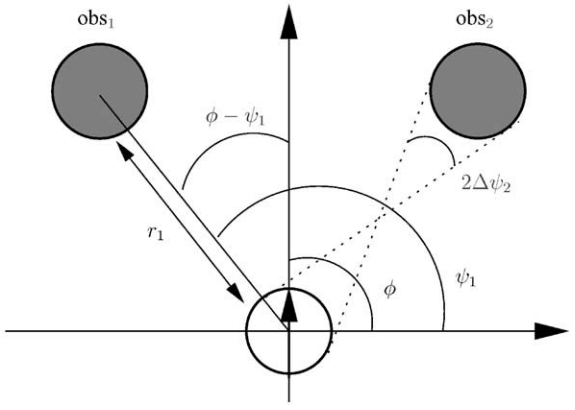


Fig. 1. Two obstacles in front of an agent.

The design of new attractor/repeller functions can be a complex task, but once it is done it results in a very fast computer implementation—it only amounts to function evaluations. Fig. 2 shows the plotting of two typical examples of these functions. All the necessary equations are in Appendix A. More details and design choices can be found in [23].

The definitions of the attractor f_{tar} (A.1) and repeller f_{obs} (A.2) functions are used to define the environment function f that takes into account both obstacles and goal direction. A simplistic approach would be to sum the obstacle and target contributions to obtain

$$\dot{\phi}(\phi, \mathbf{env}) = f = f_{\text{tar}} + f_{\text{obs}}. \tag{2}$$

Unfortunately, this approach does not always produce the desired result. Undesirable behaviors may result due to the nonlinearity of the summed functions [22,35]. To overcome this problem we use an adaptive weighted sum of the target and obstacle contributions (see (3)) instead of a direct sum.

4.1.1. Non-spherical obstacles

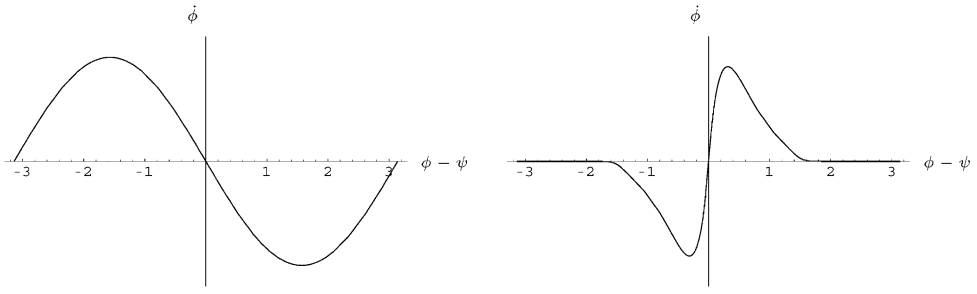
Certain environments need non-spherical objects to be well described. (Examples include walls in rooms or mazes, the ocean's bottom, or the sides of a building.) A repeller function for such objects can be simulated by placing enough small non-interleaving spheres sampling the desired area, but this can increase the complexity of an environment (number of total agents to be taken care of) by orders of magnitude.

There are two cases that need to be distinguished when taking into account a non-spherical obstacles (such as a two-dimensional segment of line). The first is when the object is far enough from the agent. It can then be treated as a normal repeller, where the midpoint of the segment is used to calculate ψ and $\Delta\psi$. The distance between the agent and the object is also calculated using standard geometric line–point properties.

If the agent is close enough, the line segment can be thought as extending itself through infinity. It is unreasonable to expect that the movement of the agent will be influenced by the midpoint of the line segment alone. Instead, ψ and $\Delta\psi$ are calculated using the closest point to the agent in the line segment. This is equivalent to using the angle between the direction of the agent and the normal of the line segment to calculate the repeller component. A reflecting effect is thus obtained.

4.1.2. Adaptive weighting of environment contributions

The repeller functions, f_{obs} are designed so that they will work as expected when there are multiple obstacles, but their direct sum with a target's function f_{tar} will not always produce the expected result. A simple example to



(a) (b)

Fig. 2. In (a), attractor function for a target in $\psi = 0$. In (b) a repeller with $\psi = 0$, $\Delta\psi = \pi/10$, $r_i = 3$, $d_0 = 6$ and $\delta = 0.8$.

demonstrate this occurs when two obstacles are too close together to allow the agent to move between them, and a target is hidden behind them. We would expect that the agent will go around the obstacles. Instead, the resulting direct sum of these functions creates an undesired attractor just in front of the agent, leading eventually to a collision.

One way to avoid this problem is to introduce a mechanism for selecting the relative weights for target and obstacle contributions in the sum. This could be accomplished with a rule-based/AI-based approach. However, the design of such rules is not obvious, and they do not scale well with the number of targets and obstacles.

Instead, we use a nonlinear dynamical system approach to introduce the real-time computation of the necessary adaptive weight functions. These functions are used to determine a *weighted sum* of the target and obstacle functions. The weight values change during the course of the simulation depending on the environment.

Based on our application domain, we use a two-dimensional weight space (obstacles and goal), but more classes can be used (allowing different types of obstacles to be treated differently, for example) depending on the application and the system's stability conditions. (Similar approaches are used in *Hopfield Neural Net-works* [36].) Therefore, the computation of the heading angle is based on a nonlinear system of the form

$$\dot{\phi} = f(\phi, \mathbf{env}) = |w_{\text{tar}}| f_{\text{tar}} + |w_{\text{obs}}| f_{\text{obs}} + n, \tag{3}$$

where w_{tar} and w_{obs} are the weights for the targets and obstacles, respectively, and n is a noise term used to escape from the unstable fixed points created by obstacles. These weights are the *fixed points* of the following nonlinear constraint competition dynamical system:

$$\begin{cases} \dot{w}_{\text{tar}} = \alpha_1 w_{\text{tar}}(1 - w_{\text{tar}}^2) - \gamma_{12} w_{\text{tar}} w_{\text{obs}}^2 + n, \\ \dot{w}_{\text{obs}} = \alpha_2 w_{\text{obs}}(1 - w_{\text{obs}}^2) - \gamma_{21} w_{\text{obs}} w_{\text{tar}}^2 + n, \end{cases} \tag{4}$$

where α_1, α_2, γ_{12} and γ_{21} are parameter functions to be designed. Based on our approach, at every iteration during the computation of $\dot{\phi}$ using (3), we compute w_{tar} and w_{obs} based on the computation of the fixed points ($\dot{w}_{\text{tar}} = \dot{w}_{\text{obs}} = 0$, which results in two nonlinear equations with unknowns w_{tar} and w_{obs}) of the system defined by (4).

A system like (4) chosen arbitrarily might not converge to a fixed point. Therefore, a careful stability analysis is required to determine its convergence properties. The stability analysis is done by first identifying the positions of the fixed points, i.e., $\dot{\mathbf{w}} = (\dot{w}_{\text{tar}}, \dot{w}_{\text{obs}})^{\text{T}} = 0$, and then for each fixed point checking the conditions such that all the eigenvalues of the *Jacobian matrix* $(\partial \dot{w}_i / \partial w_j)$ at that point are less than zero. This is a classical nonlinear dynamical system stability analysis [37]. Based on this procedure, we have computed that the above system (4) has four critical points: (0, 0), $(\pm 1, 0)$, $(0, \pm 1)$ and $(\pm A_1, \pm A_2)$, where

$$A_1 = \pm\sqrt{\frac{\alpha_2(\alpha_1 - \gamma_{12})}{\alpha_1\alpha_2 - \gamma_{12}\gamma_{21}}}, \quad A_2 = \pm\sqrt{\frac{\alpha_1(\alpha_2 - \gamma_{21})}{\alpha_1\alpha_2 - \gamma_{12}\gamma_{21}}}. \tag{5}$$

In Appendix A, Table 1 lists the stability conditions for each of these fixed points in terms of the coefficients α_i and γ_{ij}, where these carefully designed coefficients are also found.

4.2. Agent's forward velocity

Many different approaches can be used for modeling forward velocity. One possible approach is to assign a constant value to the forward velocity. This approach has drawbacks in a real-time environment: If an obstacle is suddenly in front of the agent, there might not be enough time for the agent to change direction; and a collision may occur. A better approach is to have the agent move faster when there are no objects around and slower in a crowded area. The agent should also retreat when it is too close to an obstacle. An equation for the

Table 1
Stability conditions

w_{tar}	w_{obs}	Stability	
0	0	Unstable	$\alpha_1, \alpha_2 > 0$
0	± 1	Stable	$\gamma_{12} > \alpha_1$ $\alpha_2 > 0$
± 1	0	Stable	$\gamma_{21} > \alpha_2$ $\alpha_1 > 0$
$\pm A_1$	$\pm A_2$	Stable	$\alpha_1, \alpha_2 > 0$ $\alpha_1 > \gamma_{12}$ $\alpha_2 > \gamma_{21}$ $\gamma_{12}\gamma_{21} < 0$ or $\gamma_{12}, \gamma_{21} > 0$

forward velocity that satisfies the above design criteria is as follows:

$$v = \frac{r_{\min} - d_1}{t2c}, \tag{6}$$

where $r_{\min}$ is the distance to the closest obstacle, d_1 the safety distance and $t2c$ is the time to contact. This method basically applies a *constant time to contact* approach (see [38] for similar approaches). If the closest obstacle is far then the forward velocity is large. If the closest obstacle is at a distance smaller than d_1, then the resulting forward velocity will be negative, meaning that the agent will retreat. Note that only obstacles in front of the agent should be considered for this calculation. We have used the above method in all our examples.

4.3. Extension to three dimensions

In three dimensions, angular velocities are no longer scalars, but this does not change the general approach. Basically, each contribution now is a vectorial angular velocity that can be summed.

The contribution of an attractor or repeller is obtained through the normal 2D method in the plane that contains both the current direction the agent is facing and the direction to the given object. This scalar contribution is then converted in a fully 3D angular velocity multiplying it by the normal of this plane ($p - pos \times v_{dir}$). We see this situation in Fig. 3.

Treating the non-spherical case is similar. First, we find the plane where the component is active; then, the contribution calculation is a local two-dimensional problem on this plane.

4.4. Modeling low-level personality attributes

The above dynamical system models every agent's movement in the same way. This can be a limitation when modeling more complex environments and behaviors. We add three scalar parameters (in the interval [0, 1]) that significantly change the way agents react.

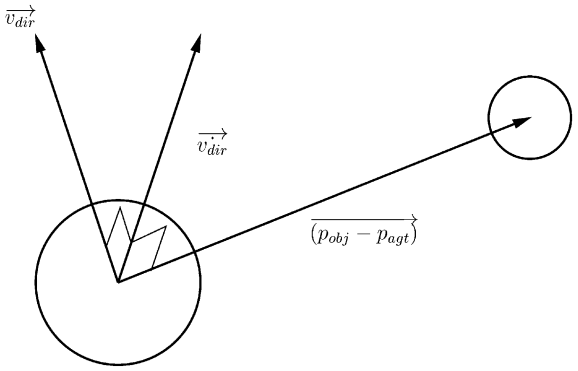


Fig. 3. Obtaining the three-dimensional angular velocity.

This approach gives each agent a certain degree of personality uniqueness. It is then possible to create a simulation with a large number of agents, each one with its own individual characteristics. Each parameter of a group of agents can then be defined as a normal distribution with given average and standard deviation. The group will act with some common characteristics but each agent will still have a particular way of doing things. This approach gives the simulation designer a great deal of flexibility.

Because these parameters only change the way numerical functions are evaluated, there is no further impact on the complexity of the system.

4.4.1. Angular acceleration—moment of inertia

Intuitively, this trait tries to model the agent's agility, i.e., how fast it can change direction. In our system, this means imposing some kind of controllable (depending on the trait's value) limitation on the change in the heading angle.

We use this parameter to model how fast $\dot{\phi}$ can change. The above designed system (3) gives us a new value for $\dot{\phi}$ at each time instant, without any consideration of the previous history of the system. We need some kind of memory that will take into account the previous values of $\dot{\phi}$, combine it with the new proposed value, and give the final value $\dot{\phi}$.

A possible realization of the above idea is to use a low-pass filter for the value of $\dot{\phi}$. Electrical engineers study and develop methods for designing different kinds of filters [39]. For our application, however, we need a very efficient solution that allows a limited degree of control over the filtering. A larger amount of dexterity allows faster agent angle changes (larger band) while a smaller one limits the angle change (smaller band). We therefore choose to use an infinite impulse response (IIR) filter, whose discrete implementation is shown in Fig. 4. The block $\boxed{z^{-1}}$ represents a fixed time delay, τ (the sampling rate, or in this case the integration step). The

equation describing this filter is

$$\dot{\phi}(n\tau) = (1-k)*\dot{\phi}((n-1)\tau) + k*\tilde{\dot{\phi}}(n\tau), \quad (7)$$

where $\tilde{\dot{\phi}}(t)$ is the output of (3) at time t, and $\dot{\phi}(t-\tau)$ is its previous evaluated value. The parameter k defines how fast the response of the system is to sudden variations; k is obtained as an affine transformation from the raw parameter. This way, we can avoid the pathological case where $k=0$ and the value of ϕ does not change. Notice that this model is extremely efficient, since it requires only a single addition and two multiplications.

4.4.2. Forward acceleration—mass

We use the second parameter to model how fast the agent can change its forward velocity. This is a kind of imposition on the forward acceleration and the problem is very similar to the one analyzed in the last section for dexterity.

We model this contribution using the same kind of low-pass filter as for the dexterity trait, but now the forward velocity is modified instead of $\dot{\phi}$.

4.4.3. Aggressiveness—a subjective quantity

Intuitively, this parameter changes how the agent is willing to take risks, how close it is willing to get to the obstacles, and how fast it should move when close to them.

In the original system (3), the constant d_0 (Eq. (A.6) in Appendix A) controls the distance where obstacles start to be taken into account. Larger values of d_0 imply that farther obstacles are also taken in account. Therefore, to model the willingness of the agent to get close to obstacles we use a mapping from courage to d_0, where large values of courage lead to small values of d_0 and small values of courage lead to large d_0. The mapping we use is

$$d_0(\mathrm{co}) = \mathrm{e}^{k(1-\mathrm{co})} - 1.0, \quad (8)$$

where $k = 2\log(d_{0base}+1)$. This function is illustrated in Fig. 5(a). For small values of the parameter, it will result in large values for d_0. When the parameter is equal to zero, it will give d_{0base} and when it is one, $d_0 = 0$.

In our nonlinear dynamic system approach, the forward velocity also depends on the constant d_1 (6). The agent will even retreat when the distance to the closest obstacle is smaller that d_1. It is important then to reduce d_1 when this aggressiveness parameter increases. We model

$$d_1 = d_{0vel} - v_0, \quad (9)$$

where

$$d_{0vel} = \begin{cases} d_{0base} & \text{if co} < 0.5, \\ 2d_{0base}(1-\mathrm{co}) & \text{if co} \leqslant 0.5. \end{cases} \quad (10)$$

Fig. 5(b) shows the plot of this function, setting the constant d_{0base} to 1.

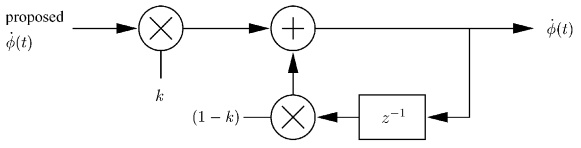


Fig. 4. Two obstacles in front of the agent.

5. Environment control: KDS

We now describe our data structures for keeping track of the near environment of each agent. We model the near environment of an agent x by a disk C_x centered at the agent. We may wish to know what obstacles are near the agent, and what other agents, friendly or hostile, are close to x and visible to it, etc. This information can be used for calculating environmental contributions to the dynamics of x, for collision avoidance and detection, and so on. The challenge here is how to keep this information up-to-date as the agents move and the environment changes. Though, in principle, this information can be recomputed from scratch at each time step, this is wasteful and would not allow our system to

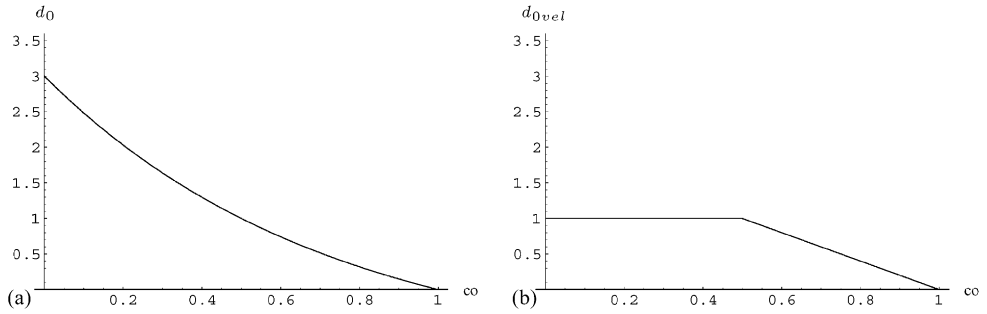


Fig. 5. Mapping from courage to d_0 and d_{0vel}. In these figures $d_{0base} = 1$.

scale well to situations where large numbers of agents are interacting.

We have developed a new kinetic data structure (KDS) tailored to this task. The KDS exploits continuity of motion and temporal coherence by focusing only on those relevant relationships in the environment that can be the ones to change next. Effectively, the KDS maintains a mathematical proof that the objects in C_x, the near environment of agent x, are the ones the KDS knows about. This proof is supported by a number of simple atomic relationships (such as distance comparisons); the set of objects forming the near environment of x cannot change unless one of those support relationships fails. The KDS tracks these atomic relationships, called the *certificates* of the KDS and, when one of them fails, it updates both the description of the near environment of x and the associated proof incrementally. A well-designed KDS for this problem attempts to come as close as possible to the goal of processing such certificate failures only when the near environment of x actually changes.

To make these general ideas concrete, consider the simple case where we model the agents as n moving points in the plane and there are no obstacles. As x moves, and C_x moves with it, we wish to track the set of other points that are inside C_x. We could do so by testing the distance of all other points from x at each time step; if we were to do this for a fraction of all the points, we have to deal with a $\Theta(n^2)$ update algorithm at each time step. Instead, our KDS solution proceeds as follows. We compute the well-known Delaunay triangulation of the n points; this structure is appropriate for our purposes, because it contains a lot of proximity information about the points. Furthermore, it is known how to maintain the Delaunay triangulation as the points move continuously, by simply doing certain edge-flip operations [40]. We have discovered the surprising fact that the only points that can enter of leave C_x are the endpoints of Delaunay edges crossed (exactly once) by the boundary of C_x. Thus, if we maintain the set of Delaunay edges crossed by C_x, we need to focus on a much smaller set of points when it comes to tracking points that may enter or exit C_x. This is depicted in Fig. 6, where the endpoints of the solid edges are the only ones that can make a transition next.

The KDS certificates, in this case, include those certifying the Delaunay triangulation (these allow us to know when edge flips need to be done) and the ones asserting that a certain set of edges is crossed by C_x. Updating the Delaunay triangulation, the set of points in C_x, and the set of crossed edges at certificate failure times is straightforward. The details are given in Appendix A.3. The appendix also discusses the extension of these ideas to handle obstacles, visibility conditions, and generalization to 3D. As compared with the naive solution to tracking the contents of C_x, the KDS solution has many advantages.

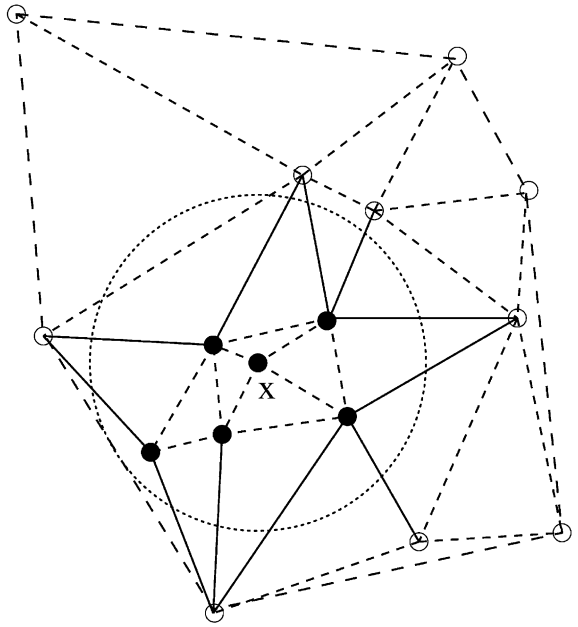


Fig. 6. Tracking the agents that may enter or exit C_x.

- Conservative estimates for the failure times of the KDS certificates can be computed using motion estimates provided by the nonlinear dynamical system; this means that most KDS certificates need not be verified at each time step but only when necessary.
- The number of other agents that can potentially enter or exit C_x has been reduced to the endpoints of Delaunay edges crossed by C_x. Though exactly these are how many depends on the radius of C_x, it is reasonable to assume that agents have a minimum size or separation and that the radius of C_x is small compared to the size of the entire environment. These conditions imply that this set of interest has size which is a small constant, irrespective of the overall number of agents. Thus, in general, our update cost will be $O(1)$ per agent, or $O(n)$ overall.

6. Global planning layer

Using local knowledge of the surroundings to determine how to reach the goal is not always enough. The agent will likely fail to reach the goal under trap situations like the one suggested in Fig. 7. In this example, using just the direct direction to the target, the agent enters the dead-end region until it decides that it has to turn off the target contribution. By then it turns around and leaves the U-shaped obstacle, but shortly

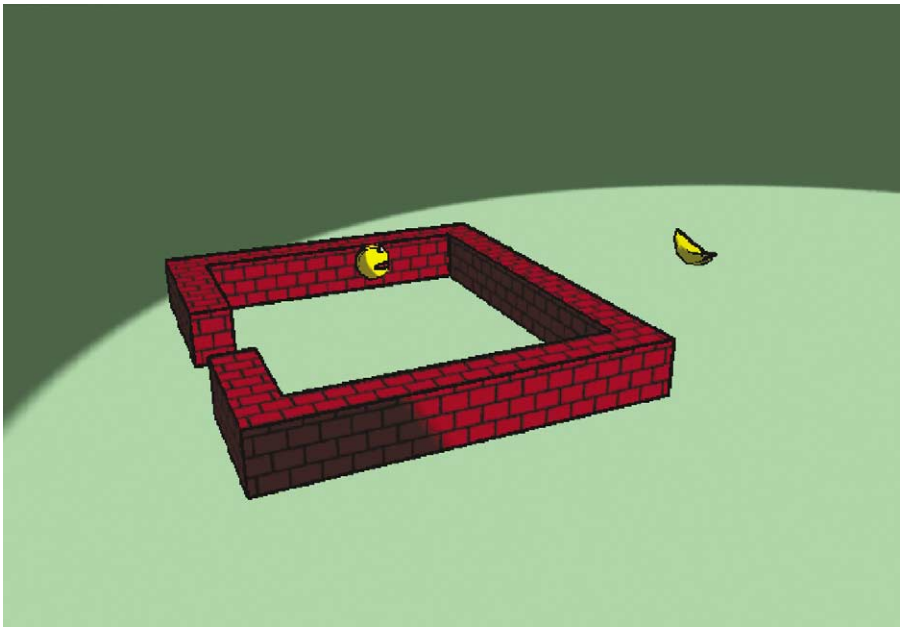


Fig. 7. A trap situation where the agent could possibly fail to reach the goal. The agent is represented by the pacman-like figure, while the target is represented by a banana. It is a good example when a more sophisticated approach for the global planning layer is necessary.

after, it again takes the target into consideration, reestablishing the cycle. Every now and then, because of the noise inserted into the equations, the agent might escape this dead-end cycle—but there is no guarantee of how or when it would happen.

At each moment in time, the global planning layer tells each agent what direction it should pursue to achieve its goal. As a simple example, it might assert that the direction pointing directly to the desired target is the correct course. Although this is not always the best suggestion, in many applications it is an adequate, simple solution (see examples "chicken-2D", "chicken-3D", "flock1", etc.).

When several agents are to cooperate on a task, coordinating their efforts is the key to success. The global planning layer is the intelligence behind such multiple-agent scenarios. For example, it can use graph theory and computational geometry techniques to figure out ways to surround a target. It can simulate changes in behavior of different agents, depending on their sub-tasks. It can even simulate an agent getting tired and becoming less responsive as time passes. (The local layer could not perform this kind of multi-agent coordination. An agent's only local-layer goal is to reach a point in space while avoiding obstacles; the agent is only aware of its own target.)

An advantage of having an independent layer for doing such global calculations is that, as long as it communicates the same way with the other layers, it may be straightforwardly replaced with a more sophisticated approach as needed.

6.1. Precomputed velocity fields

We now explain a more sophisticated way to determine what the local layer should use as the goal direction: a pre-computed velocity field.

The ability to use arbitrary velocity fields gives the designer of the environment/simulation a great flexibility to achieve effects that might not be normal or expected, but that are desired or necessary for a given application. An initial velocity field can also be obtained through automatic procedures, like the use of *harmonic functions* (i.e. also known as *potential functions*), to obtain a good initial global guess of how to avoid local traps.

The potential field construction is a computationally expensive task, and to do it in every integration step would rule out any real/interactive time application. What we do is pre-compute the potential field for only static obstacles known in advance. This is not a limitation, since main landmarks of the environment are usually already known. In a real-terrain simulation, it is absolutely reasonable to expect that the position of rivers, crevices and buildings are known in advance, in games and VR applications the "map" is already known and many times there are even pre-computed BSP-trees.

Moving obstacles, as well as the static ones, are taken into account by the steering and collision avoidance

capabilities of the two other layers, and the direction of the velocity field is then fed into the local layer whenever the target is not in clear direct sight. This technique avoids local minima situations like the one in the Fig. 7 since there the goal direction in the "U-shaped" room would be towards the exit and not in the local direct line to the target.

Potential functions have already been used in robotic path planning applications [41]. Due to certain inherent properties such as the guaranteed absence of local minima, harmonic potential functions are particularly suitable when robust trajectory generation is important [42]. Harmonic potentials are also used in several physical domains, such as hydrodynamics and electrostatics, to analyze complex fluid flow situations and to compute electric potentials. Modeling the agent motion as a flow allows the use of the well-developed underlying mathematical theory and the associated powerful tools.

6.1.1. Harmonic functions

Harmonic functions are used, among other things, to model the behavior of ideal (incompressible and non-viscous) fluids. They are solutions to the *Laplace equation*:

$$\nabla^2\lambda = \frac{\partial^2\lambda}{\partial x^2} + \frac{\partial^2\lambda}{\partial y^2} = 0, \tag{11}$$

where, $\nabla^2 = \nabla \cdot \nabla$ is the *Laplacian* operator.

In the absence of viscosity, a fluid flow is irrotational; it cannot possess eddies and vortices. Since vorticity is represented by the *curl* of the velocity vector $\boldsymbol{v}$, an irrotational flow satisfies *curl* $\boldsymbol{v} = \nabla \times \boldsymbol{v} = 0$. Equivalently, one can express v as the gradient of a potential function, $\boldsymbol{v} = -\nabla\lambda$, where λ is a scalar potential. For an incompressible fluid, according to the continuity equation, the *divergence* of the velocity field must be zero [43,44], $\nabla \cdot \boldsymbol{v} = 0$.

The path adopted by a fluid particle in a steady flow is called a *streamline*. A stream-line is the integral curve of (11) and may be represented by the gradient of the harmonic potential. Streamlines may not intersect each other but external perturbations, or even integration errors, will cause a particle to jump from one streamline to another.

For our purposes, we only use the direction (and ignore the magnitude) of the local gradient vector as the nominal goal direction of our agent. Our boundary conditions in Laplace's equation are the target potential (the potential of the region covered by the target) and the obstacles' potential, which include the configuration space boundary. Typically, the target potential is set to a low value (the global minimum) and the obstacle potential is set to a high value. Since fluid tends to flow towards a decreasing potential, all the streamlines generated in this setup will eventually converge to the goal point.

The implementation details for the potential field can be found in Appendix A.

6.2. Hybrid systems

Another approach to global planning involves *hybrid dynamical systems* (*hybrid systems*, for short), i.e., systems that combine continuous and discrete dynamics. Intuitively, global planning may be described in a state-transition framework: Each state may represent a mode of continuous behavior, corresponding to a particular "strategy" for low-level reactive navigation (e.g., "Go in the direction of the target," or "Ignore the target for now, just escape this apparent local minimum"); high-level planning decisions may be represented by discrete, instantaneous transitions between states. This hybrid discrete/continuous mix neatly matches both well-understood hybrid systems theory and natural abstractions of autonomous agent behavior.

The formal theory of hybrid systems [45–47] has been employed in diverse application domains, but animation is not typically considered in hybrid systems literature. Conversely, animation systems commonly combine discrete and continuous dynamics, but rarely is it clear how such systems relate to the theoretical foundations of hybrid systems. Nonetheless, theoretical models of hybrid systems allow us to rigorously specify a state-transition approach to global planning without sacrificing any aspect of our technique for local, continuous behavior. In the paper [48], our agent steering method was the basis for crowd simulation and agent navigation animations that were formally specified and generated using the general purpose hybrid system tool, CHARON [49]. That paper also presents a simple demonstration that, in some obstacle/target configurations, the ability to switch between continuous behaviors in a hybrid system may result in more effective global planning than if only one continuous behavior were possible. (These results do not add to the foundations of our approach to agent navigation, so a full discussion of them is outside the scope of this paper. See [48] for more details.) Because our method for agent navigation is designed to accommodate a layered approach to planning, it is an excellent fit in a hybrid systems framework.

Another benefit emerges from a hybrid systems-oriented approach to global planning: a framework for reasoning about animation systems. Animators may want to verify properties of animations that could not easily be seen by watching a single animation, e.g., properties of agent speed and precise distance that may be too subtle for the human eye, or properties of a large class of animation systems. Such a task would be extremely difficult without the assistance of an expressive logical framework. Indeed, merely finding a formal language to express interesting properties of animation systems may be non-trivial. By working within the

theoretical framework of hybrid systems, however, animators may employ logics for hybrid systems to formally specify complex properties of animation systems. In addition, there are practical *model checkers*—tools that can mechanically verify some properties of simple hybrid systems—that can be applied to simple animation systems. These model checkers have significant limitations: Many properties are theoretically undecidable, and as a practical matter, even decidable properties may only be feasibly checked in simple cases. Nonetheless, Aaron et al. [48] report a successful application of model checking to verify collision-avoidance in a simple race-like game. Further, as we discuss in Section 8, there are approaches to reasoning about complex systems that allow us to circumvent some undecidability barriers.

7. Experiments

We demonstrate the power of our approach with a series of two- and three-dimensional experiments. We demonstrate how changes in the intrinsic parameters of an agent can generate different behaviors, in situations with few agents in crowd simulations. All our experiments run in the range of 1–700 frames per second on a Pentium II 400 MHz Linux workstation.

In the two-dimensional flocking example, we used a random set of intrinsic attributes for each of the agents. We show how some of the chosen attributes are not sufficient for the agents to reach the target. For example, there are two agents that sometimes get stuck behind the obstacles. Their attributes allow them to get really close to the target but do not give them enough flexibility to steer away in time. The forward velocity component of the system is responsible to avoid actual collisions.

We use harmonic functions on the two maze-structured examples, and in the theater example in order to provide a global direction that guarantees the agents do not get stuck in dead-ends.

In the three-dimensional examples, we illustrate how the intrinsic parameters of the agents affect the general flocking property of a group.

8. Conclusions and future work

In this paper, we have presented a three-layered system for agent steering and customizable behavior in real-time environments. The system uses nonlinear dynamical systems, harmonic functions, and a specially designed kinetic data structure to navigate around static and/or moving obstacles to reach static and/or moving targets.

We demonstrated our principled approach in several examples, including flocking simulation (which is itself a challenging area of research). We also discussed a variety of low-level, continuously adaptive behaviors generated using personality parameters. Systems based on our approach can make it easier for users to achieve efficiently complex low-level agent behaviors and easily build new personality traits. Our approach can serve as a basis for modeling higher-level behaviors, since it provides a continuum of complex low-level behaviors. The embedded controlling parameters allow time-changing agent behaviors. It is possible to simulate fatigue in plane pilots, and adrenaline rush, and also possible to simulate a highly inertial vehicle, etc., all within the realm of the low-level layer.

The kinetic data structure (KDS) perfectly suits the environment layer, allowing an implementation to simulate an agent's perception of its local environment. A KDS can maintain a substantial amount of information and deal with continuous time collision detection (in precise points between integration steps), all with no significant impact on computational complexity.

By applying hybrid systems theory in the global planning layer, we may be able to formally specify and reason about properties of animations using expressive modal logics for hybrid systems. Despite the significant undecidability barriers [50] mentioned in Section 6.2, we need not simply abandon hope of verifying complex systems: There are *approximation* techniques for verification that might be applicable to animation systems. Many relevant properties of animation systems (such as those formally specified in [48]) are *reachability* properties, fundamentally about whether some proposition holds in the states reachable by a system. It is often impossible to effectively reason about the exact set of reachable states of a complex system, but we may be able to verify properties on an approximation of that set. That is, if S is the actual set of states reachable by a system, and we cannot decide property P on S, we might instead be able to overestimate S by a computationally simpler set $S' \supset S$ on which P is decidable. Then, if we prove that P holds on all states in S', we know P also holds on all states in S. This kind of reasoning by approximation is an active area of research in the hybrid systems community [51–53], but it has not yet been explored in the context of animation systems.

Our ability to reason about virtual words is essential to our ability to create intelligent virtual agents and design animation systems. Therefore, integrating concepts from hybrid systems, AI, and computational geometry into the global planning layer could yield significant advances; for example, a global, centralized structure could be created to facilitate distributed, cooperative planning among intelligent agents. For several reasons, a hybrid systems-oriented approach seems particularly well-suited for such purposes. It would naturally exploit a critical feature of our layered approach to planning, distinguishing low-level (i.e.,

continuous-level) tasks such as obstacle avoidance from abstract, higher-level (i.e., discrete-level) tasks. It could directly express properties of time, escaping a typical AI-based restriction. Further, the highly expressive framework of hybrid systems allows formal reasoning while also permitting us to retain the character of our reactive agent-steering behavior. No matter what techniques are used, however, success is likely to arise from the natural degree of abstraction that our approach provides between high-level and low-level planning.

Appendix A. Formulas and equations

In this appendix, we provide some of the details omitted from the main text of this paper to provide a smoother overview. These details, however, are still necessary for reproducing our work.

A.1. Nonlinear dynamical system equations

A.1.1. Heading attractor and repeller

The model we use for the attractor function f_{tar} is defined as a sinusoidal shifted by ψ, whose argument is the agent's heading angle, i.e., the relative angle between the agent's direction and the object direction:

$$f_{\text{tar}} = -a\sin(\phi - \psi). \tag{A.1}$$

It is worth noting that this function generates an attractor at $\phi - \psi = 0$ (when the target is just in front of the agent) and a repeller at $\phi - \psi = \pi$ (when the target is right behind the agent). Here a is always taken as 1.

To define a repeller function, a simple sinusoidal function is not enough. Two obstacles which are close to each other should sum up to form a larger repeller, but if they are far enough apart, an agent should be able to pass through them. In the design of a repeller, some other aspects of the environment geometry should also be taken into account, such as the distance from the agent. In addition, we would like to parametrically control certain properties of the repellers such as their spatial extent. Therefore, as a model for a repeller, we use the multiplication of three different functions, R_i, D_i, W_i, each modeling a different property, i.e.,

$$f_{\text{obs}_i} = R_i W_i D_i. \tag{A.2}$$

Function R_i models a generic repelling property of an obstacle i, and is defined as

$$R_i = \frac{(\phi - \psi_i)}{\Delta\psi_i} e^{(1-(\phi-\psi_i)/\Delta\psi_i)}, \tag{A.3}$$

where $\phi - \psi_i$ is the angle of obstacle i with respect to the agent's direction and $2\Delta\psi_i$ is the subtended angle of the obstacle by the agent.

The second function, W_i, is responsible for limiting the angular range of the repeller's influence in the environment and is defined as

$$W_i = \tfrac{1}{2}\big[\tanh(h_1(\cos(\phi - \psi_i) - \cos(2\Delta\psi_i + \sigma))) + 1\big], \tag{A.4}$$

which models a window-shaped function. h_1 is responsible for the inclination of the window's sides and is defined as

$$h_1 = 4/(\cos(2\Delta\psi) - \cos(2\Delta\psi + \delta))x, \tag{A.5}$$

where δ is a safety margin constant (in all our applications, we kept $\delta = 0.8$).

The third function, D_i, models the influence of the ith obstacle in the environment based on the distance of the obstacle from the agent. It is modeled as

$$D_i = e^{-r_i/d_0}, \tag{A.6}$$

where r_i is the relative distance between the obstacle and the agent, and d_0 controls the strength of this influence as the distance changes (exponential decay).

Fig. 2(a) is a plot of f_{tar} for a target located just in front of the agent ($\phi - \psi = 0$). Fig. 2(b) is a plot of f_{obs} for an obstacle located just in front of the target, where $\phi - \psi = 0$, $\Delta\psi = \pi/10$, $r_i = 3$, $d_0 = 6$ and $\delta = 0.8$.

A.1.2. Adaptive weighting of contributions

Table 1 lists the stability conditions for each fixed point [23].

From the above table, we can conclude the following. The first fixed point case, $w_{\text{tar}} = w_{\text{obs}} = 0$, results in turning off contributions from both targets and obstacles. Clearly, we would like that this fixed point is always unstable. Therefore, we should always have $\alpha_1 > 0$ and $\alpha_2 > 0$.

The second fixed point, $w_{\text{tar}} = 0$ and $w_{\text{obs}} = \pm 1$, should be stable when the agent must stop considering the target and concentrate only on obstacle avoidance. This happens when $\gamma_{12} > \alpha_1$ and $\alpha_2 > 0$.

The third fixed point, $w_{\text{tar}} = \pm 1$ and $w_{\text{obs}} = 0$, should be stable when there are no obstacles around. In this case, the agent only focuses on the target. This point is stable when $\gamma_{21} > \alpha_2$ and $\alpha_1 > 0$.

The fourth fixed point (w_{tar} and w_{obs} have values different than either zero or one) amounts to what is called an averaging condition. It will be stable when $\alpha_1 > \gamma_{12}$, $\alpha_2 > \gamma_{21}$ and either $\gamma_{12}\gamma_{21} < 0$ or $\gamma_{12}, \gamma_{21} > 0$. The exact value of the fixed point will depend on the actual values of the four coefficients computed from (5).

The above mathematical facts lead to the following design of the parameter functions α_i, γ_{ij}. First we need to design a fixed point detector for each of the target f_{tar} (A.1) and obstacle f_{obs} (A.2) functions. We define a fixed

point detector for f_{tar} as follows:

$$P_{\text{tar}} = \text{sgn}\left(\frac{\partial f_{\text{tar}}}{\partial \phi}\right) e^{c_1 |f_{\text{tar}}|}. \quad (A.7)$$

The first multiplication term (signum function) will distinguish the attractor fixed point from the repeller fixed point of f_{tar} (cases when the target is just in front or is just behind the agent) by detecting the sign of the slope ($\partial f_{\text{tar}}/\partial \phi$). The exponential is 1 for fixed points (i.e., $f_{\text{tar}} = 0$) and is close to 0 elsewhere (these values are controlled by the constant c_1, which here is set to 2).

In case of an obstacle we can not use the exact same expression, since f_{obs} has a limited range of influence (due to the window function W_i (A.4)). There are several intervals in f_{obs} that have zero value but do not indicate the location of the obstacle (see Fig. 2(b)). The solution to this uses the same design as in (A.4).

$$P_{\text{obs}} = \text{sgn}\left(\frac{\partial f_{\text{obs}}}{\partial \phi}\right) e^{c_1 |f_{\text{obs}}|} \left(\sum_i W_i\right). \quad (A.8)$$

Now, using P_{tar} and P_{obs}, we can design γ_{12} as follows:

$$\gamma_{12} = \frac{e^{-c_2 P_{\text{tar}} P_{\text{obs}}}}{e^{c_2}}. \quad (A.9)$$

When $\gamma_{12} > \alpha_1$ (see Table 1) the deactivation of the target contribution ($w = (0, \pm 1)$) is a stable solution. This definition results in a γ_{12} that is slightly larger than zero in most regions, but will have a narrow peak (c_2 controls how narrow, here is set to 2) around the region where targets and obstacles are aligned.

The relationship between γ_{21} and α_2 will choose between the stable points $w = (\pm 1, 0)$ and $w = (\pm A_1, \pm A_2)$. The only instant where we should turn off the obstacles' contribution is when there are no obstacles close to the agent. The condition for this to happen is $\gamma_{21} > \alpha_2$. Therefore, we use a small value of 0.05 for γ_{21}. α_2 should be related to the distance of the obstacles to the target. The closer the obstacles are from the target, the larger α_2 should become. To achieve this effect, we use the functions D_i defined in (A.6) and we define α_2 as

$$\alpha_2 = \tanh\left(\sum_i D_i\right). \quad (A.10)$$

The hyperbolic tangent will smoothly limit the range of values of α_2 between 0 and 1.

We still have to select α_1. Remember that the relationship between γ_{12} and α_1 will deactivate the target when $\gamma_{12} > \alpha_1$. To improve the collision avoidance, if there are close obstacles the target should be disregarded. To achieve this, we decrease α_1 as α_2 increases and we use the following equation:

$$\alpha_1 = 0.4(1 - \alpha_2). \quad (A.11)$$

A.2. Potential field implementation details

This section presents the computational scheme for solving (11) and is adapted from [2,54,55]. In a numerical scheme, the configuration space is discretized and is replaced by a grid.

Applying Taylor's theorem to $\lambda(t)$ between $x - h \leqslant t \leqslant x + h$, where typically $h = 1/N$, we can write

$$\lambda''(x) = \frac{\lambda(x+h) - 2\lambda(x) + \lambda(x-h)}{h^2}. \quad (A.12)$$

The above equation may be used to obtain the numerical counterpart of (11),

$$\frac{\lambda(x_{j+1}, y_k) - 2\lambda(x_j, y_k) + \lambda(x_{j-1}, y_k)}{h^2} + \frac{\lambda(x_j, y_{k+1}) - 2\lambda(x_j, y_k) + \lambda(x_j, y_{k-1})}{h^2} \approx 0, \quad (A.13)$$

ignoring terms involving h^2 and fourth-order derivatives of λ. From (A.13) we can compute $\lambda_h(x_j, y_k)$ at node (j, k),

$$\lambda_h(x_j, y_k) = \tfrac{1}{4}\big[\lambda_h(x_{j+1}, y_k) + \lambda_h(x_j, y_{k+1}) + \lambda_h(x_{j-1}, y_k) + \lambda_h(x_j, y_{k-1})\big]. \quad (A.14)$$

In other words, the harmonic potential of a node is the average of the potentials of its four immediate neighbors.

We first set the target potential to 0 and the obstacle and boundary potential to 1. To initiate the iterations, the empty configuration space nodes are arbitrarily set to a value of 0.5. In the Gauss–Seidel [55] approach, the mth iteration step for node points $k = 1, 2, \ldots, N-1$ and $j = 1, 2, \ldots, N-1$ in an $N \times N$ mesh is expressed as

$$\lambda_h^{(m)}(x_j, y_k) = \tfrac{1}{4}\Big[\lambda_h^{(m-1)}(x_{j+1}, y_k) + \lambda_h^{(m-1)}(x_j, y_{k+1}) + \lambda_h^{(m)}(x_{j-1}, y_k) + \lambda_h^{(m)}(x_j, y_{k-1})\Big], \quad (A.15)$$

whereas for the boundary point nodes,

$$\lambda_h^{(m)}(x_j, y_k) = 0 \text{ or } 1 \quad \text{for all } m \geqslant 0. \quad (A.16)$$

We terminate the iterations when the difference between the nodal values at the current and previous time steps is below a chosen threshold (in all our experiments we set it to $10e^{-15}$). The Gauss–Seidel method may be significantly accelerated by using the *successive over relaxation* or the SOR method [55].

A.3. Kinetic near neighbors

In this Section, we provide details about the kinetic data structure for maintaining the near environment of each agent. Geometrically, the agents are modeled as moving points, obstacles (fixed or moving) as line segments, and the near environment as a ball of a certain radius around an agent. We accomplish these

goal by maintaining variants of the Delaunay triangulation of the moving agents.

A.3.1. Nearest neighbors in 2D

Let $G(V,E)$ be a planar straight-line graph, where $|V| = n$, and let S be a subset of V of size k. With each point p in S, we associate a circle C_p centered at the p that is of radius r_p. Let T be a triangulation of G. We call a point q in V *approachable* from a point p in S if q is inside C_p and there exists a path from p to q in T that lies entirely in C_p. We also say that an edge e of T *properly intersects* C_p if one endpoint of e lies outside of C_p and the other endpoint of e is approachable from p.

We assume now that the points in V are moving points in the plane and that the radii of the points in S may be time-dependent. We consider two cases: the *unconstrained* case, when there are no obstacles present, and the *constrained* case, with obstacles. In the unconstrained case, we maintain for each point p in S the set of points of V that are inside C_p. In the constrained case, we maintain the set of points that are inside C_p which are not blocked from p by a constrained edge that interests C_p, but not properly (these correspond to the agents near to and visible from the given agent). It turns out that, if T is the constrained Delaunay triangulation (CDT) [56], the set that we want to maintain for a point p in S is the set of points that are approachable from p. In particular, in the unconstrained case, every point inside C_p is approachable from p, whereas in the constrained case the only points inside C_p that are not approachable from p are those that are blocked by a constrained edge that crosses C_p, but not properly.

The key observation that enables us to maintain the above mentioned sets is captured in the following lemma:

Lemma 1. *Let T be the CDT of G and let C_p be a circle centered at $p \in S$. If point $q \in V$ enters/exits the circle C_p at some time t_0 and is visible from at least one point inside C_p, then at t_0 there exists an edge of T between q and a point inside C_p.*

Proof. At time t_0, q is on the boundary of C_p. Let $\{C_r\}$ be the family of circles with center r that pass through q, where r is a point on the segment pq. Consider the circle $C_{r'}$ such that r' is at maximal distance from q, visible from q and the circle $C_{r'}$ contains no points of V. Note that because the set of points of V that are visible from q at t_0 is non-empty by assumption, such a circle $C_{r'}$ always exists. Clearly, the edge qr' is a constrained Delaunay edge. □

This observation immediately provides a way to kinetize the maintenance of neighbors of every point p in S by maintaining the CDT of S. In addition, for every p in S keep the set of approachable points A_p and the set of properly crossing edges E_p. If the combinatorial structure of T does not change, then the endpoints of these edges are the only points that could possibly enter or exit C_p. In other words the maintenance of the neighbors of the points in V consists of two parts: one is the maintenance of the triangulation itself, and the second is the maintenance of the triangulation edges crossed by the circles C_p.

The CDT can be maintained using standard edge-flip operations, just like the regular Delaunay triangulation [40,57]. Whenever an edge flip happens, however, we also need to update the sets E_p, $p \in S$: when an edge in E_p, for some p, disappears we need to delete that edge from the edge set E_p, whereas when an edge that intersects the circle C_p, for some p, appears we need to add it to E_p.

The situation is somewhat more complicated if a point enters or exits the circle. In case a point q enters the circle C_p, we have to look at q's neighbors. For those neighbors that are outside, we only need to add the corresponding edges to E_p. For those that are inside and approachable, we need to remove the corresponding edges from the edge set E_p. Finally, for the neighbors that are inside but not approachable (this can only occur in the constrained case) we need to add them to the point set A_p and perform the same tests for their neighbors recursively. When a point q exits C_p the situation is entirely symmetric: for all the neighbors that are outside, we delete the corresponding edges from E_p. For the neighbors that are inside and remain approachable after the point exits, we need to add the corresponding edges to the set E_p. Finally as far as the remaining neighbors are concerned, we have to delete them from the set A_p of approachable neighbors, as well as delete any edges in E_p that adjacent to them and recursively do the same for their neighbors.

A.3.2. Nearest neighbors in 3D

In three dimensions, we have implemented only where there are no obstacles present. For every point $p \in S$, we now associate a sphere C_p of radius r_p. The definitions of *approachability* and *proper intersection* are the same as in the two-dimensional case.

Our goal is to maintain for each point $p \in S$ the set of points that are inside C_p. As in two dimensions, if T is the Delaunay triangulation, this set is the same as the set of A_p of approachable points, and Lemma 1 is still true.

The three-dimensional Delaunay triangulation is maintained by simply doing some face-edge of edge-face flips [58]. Hence, what we need to do in the 3D case in order to update our nearest neighbor structure is essentially the same as in the two-dimensional unconstrained case. The only difference now is that when we

have a flip an edge either appears or disappears, whereas in the 2D case, whenever a flip occurs, exactly one edge appears and exactly one disappears.

References

[1] Fox RW, McDonald AT. Introduction to fluid mechanics. New York: Wiley, 1989.

[2] Press WH, Teukolsky SA, Vetterling TV, Flannery BP. Numerical recipes in C. Cambridge, UK: Cambridge University Press, 1992.

[3] Basch J, Guibas L, Hershberger J. Data structures for mobile data. Journal of Algorithms 1999;31:1–28.

[4] Guibas LJ. Kinetic data structures—a state of the art report. In Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics (WAFR), 1998. p. 191–209.

[5] Reynolds C. Flocks, herds, and schools: a distributed behavioral model. In Proceedings of SIGGRAPH '87, vol. 21. 1987. p. 25–34.

[6] Cavazza M, Earnshaw R, Magnenat-Thalmann N, Thalmann D. Survey: motion control for virtual humans. IEEE Computer Graphics and Applications, Sep/Oct 1998; 18(5):24–31.

[7] Stout B. Smart move: Path-finding. Game Developer, Oct 1996.

[8] Vinckle S. Real-time pathfinding for multiple objects. Game Developer, June 1997.

[9] Pottinger D. Coordinated unit movement. Game Developer, Jan 1999.

[10] Reinolds C. Steering behaviors for autonomous characters. In: Proceedings of Game Developers Conference, 1999.

[11] Haumann D, Parent R. The behavioral test-bed: obtaining complex behavior from simple rules. The Visual Computer 1988;4(6):332–47.

[12] Lethebridge T, Ware C. A simple heuristically-based method for expressive stimulus–response animation. Computers and Graphics 1989;13(3):297–303.

[13] Beardon C, Ye V. Using behavioral rules in animation. In: Computer graphics: developments in virtual environments. New York: Academic Press, 1995. p. 217–34.

[14] Funge J, Tu X, Terzopoulos D. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. Proceedings of SIGGRAPH 99, 1999. p. 29–38.

[15] Ridsdale G. Connectionist modelling of skill dynamics. Journal of Visualization and Computer Animation 1990;1(2):66–72.

[16] Renault O, Thalmann N, Thalmann D. A vision-based approach to behavioural animation. The Journal of Visualization and Computer Animation 1990;1(1):18–21.

[17] Tu X, Terzopoulos D. Artificial fishes: physics, locomotion, perception, behavior. In: Proceedings of SIGGRAPH '94, 1994. p. 43–50

[18] Noser H, Renault O, Thalmann D, Thalmann N. Navigation for digital actors based on synthetic vision, memory and learning. Computer and Graphics, 1995.

[19] Grzeszczuk R, Terzopoulos D. Automated learning of muscle-actuated locomotion through control abstraction. In Proceedings of SIGGRAPH '95, 1995. p. 63–70.

[20] Hodgins JK, Pollard NS. Adapting simulated behaviors for new characters. In: SIGGRAPH '97 Conference Proceedings, Annual Conference Series, 1997. p. 153–62.

[21] Brogan D, Metoyer R, Hodgins J. Dynamically simulated characters in virtual environments. IEEE Computer Graphics and Applications. Sep/Oct 1998;18(5):59–69.

[22] Goldenstein S, Large E, Metaxas D. Dynamic autonomous agents: game applications. In: Proceedings of Computer Animation 98, June 1998.

[23] Goldenstein S, Large E, Metaxas D. Non-linear dynamical system approach to behavior modeling. The Visual Computer 1999;15:349–69.

[24] Wilfong G. Motion planning in the presence of movable obstacles. In: Proceedings of the 4th ACM Symposium on Computational Geometry, 1988. p. 179–288.

[25] Guibas LJ, Sharir M, Sifrony S. On the general motion planning problem with two degrees of freedom. In: Proceedings of the 4th ACM Symposium on Computational Geometry. New York: ACM Press, 1988. p. 289–98.

[26] Costa M, Feijo B. Agents with emotions in behavioral animation. Computers and Graphics May 1996;20(3): 377–84.

[27] Blumberg B. Go with the flow: synthetic vision for autonomous animated creatures. In: Poster on AIII Conference on Autonomous Agents, 1997.

[28] Musse S, Thalmann D. Hierarchical model for real time simulation of virtual human crowds. IEEE Transactions on Visualization and Computer Graphics, 2001;7(2):152–64.

[29] Kurlander D, Ling DT. Planning-based control of interface animation. In: Proceedings of the CHI95 Conference. New York: ACM Press, 1995. p. 472–9.

[30] Magnenat-Thalman N, Thalmann D. Digital actors for interactive television. Proceedings of the IEEE (Special Issue in Digital Television, Part 2) July 1995; 83(7):1022–31.

[31] Perlin K, Goldberg A. IMPROV: a system for scripting interactive actors in virtual worlds. In: SIGGRAPH '96 Conference Proceedings, ACM SIGGRAPH, 1996. p. 205–16

[32] Arkin R. Integrating behavioral, perceptual, and world knowledge in reactive navigation. Journal of Robotics and Autonomous Systems 1990;6:105–22.

[33] Steinhage A, Schöner G. The dynamic approach to autonomous robot navigation. In: Proceedings IEEE International Symposium on Industrial Electronics, 1997.

[34] Large E, Christensen H, Bajcsy R. Scaling the dynamic approach to path planning and control: competition among behavioral constraints. International Journal of Robotics Research, 1999;18(1):37–58.

[35] Large E, Christensen H, Bajcsy R. Scaling the dynamic approach to path planning and control: competition among behavioral constraints. International Journal of Robotics Research, 1998.

[36] Haykin SS. Neural networks: a comprehensive foundation. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[37] Perko L. Differential equations and dynamical systems. Number ISBN-0387974431 in Texts in Applied Mathematics. Berlin: Springer, February 1991.

[38] Schöner G, Dose M, Engels C. Dynamics of behaviour: theory and applications for autonomous robot archi-

tectures. Robotics and Autonomous Systems 1996;16(2–4): 213–46.
[39] Oppenheim A, Willsky A, Young I. Signal and systems. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[40] Guibas LJ, Mitchell JSB, Roos T. Voronoi diagrams of moving points in the plane. In: Schmidt G, Berghammer R, editors. Proceedings of the 17th International Workshop Graph-Theoretical Concepts in Computer Science. Lecture Notes in Computer Science, vol. 570. Berlin: Springer, 1991. p. 113–25.
[41] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. In: IEEE International Conference on Robotics and Automation. March 1985. p. 500–5.
[42] Connolly CI. Application of harmonic functions to robotics. In: International Symposium on Intelligent Control, 1992.
[43] Foster, N, Metaxas D. Realistic animation of liquids. In: Proceedings GI '96, 1996. p. 204–12.
[44] Foster N, Metaxas D. Modeling the motion of hot, turbulent gas. In: Proceedings of SIGGRAPH '97, 1997.
[45] Proceedings of the IEEE, vol. 88, July 2000.
[46] Alur R, Henzinger TA, Sontag ED, editors. Hybrid Systems III, Lecture Notes in Computer Science, vol. 1066. Berlin: Springer, 1996.
[47] Lynch N, Krogh BH, editors. Hybrid systems: computation and control, Lecture Notes in Computer Science, vol. 1790. Berlin: Springer, 2000.
[48] Aaron E, Metaxas D, Ivančić F, Sokolsky O. A framework for reasoning about animation systems. In: Proceedings of the 3rd International Workshop on Intelligent Virtual Agents, 2001.
[49] Alur R, Grosu R, Hur Y, Kumar V, Lee I. Modular specification of hybrid systems in CHARON. In: Lynch N, Krogh BH, editors. Hybrid systems: computation and control, Lecture Notes in Computer Science, vol. 1790. Berlin: Springer, 2000.
[50] Alur R, Henzinger T, Lafferriere G, Pappas G. Discrete abstractions of hybrid systems. Proceedings of the IEEE July 2000;88:971–984.
[51] Dang T, Maler O. Reachability analysis via face lifting. In: Henzinger T, Sastry S, editors. Hybrid systems: computation and control, Lecture Notes in Computer Science, vol. 1386. Berlin: Springer, 1998. p. 96–109.
[52] Henzinger TA, Ho P-H. A note on abstract-interpretation strategies for hybrid automata. In: Antsaklis P, Nerode A, Kohn W, Sastry S, editors. Hybrid Systems II, Lecture Notes in Computer Science, vol. 999. Berlin: Springer, 1995. p. 252–64.
[53] Pappas GJ, Sastry S. Towards continuous abstractions of dynamical and control systems. In: Antsaklis P, Kohn W, Nerode A, Sastry S, editors. Hybrid Systems IV, Lecture Notes in Computer Science, vol. 1273. Berlin: Springer, 1997. p. 329–41.
[54] Sato K. Deadlock-free motion planning using the laplace potential field. Advanced Robotics 1993;7(5):449–61.
[55] Atkinson KE. An introduction to numerical analysis. New York: Wiley, 1978.
[56] Chew LP. Constrained Delaunay triangulations. Algorithmica, 1989;4:97–108.
[57] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational geometry: algorithms and applications. Berlin: Springer, 1997.
[58] Facello MA. Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. Computer Aided Geometric Design June 1995;12(4): 349–70.