Interaction with Groups of Autonomous Characters

Craig Reynolds

Research and Development Group
Sony Computer Entertainment America
919 East Hillsdale Boulevard
Foster City, California 94404

craig_reynolds@playstation.sony.com
http://www.red.com/cwr/
cwr@red.com

Keywords: autonomous characters, behavioral animation, steering behaviors, flocks, real time, spatial data structures.

Abstract

This paper presents a methodology for constructing large groups of autonomous characters which respond in real time to the user's interaction, as well as to each other and their environment. The characters are based on steering controllers under the direction of a simple mental model which mediates between several conflicting behavioral goals. The characters are represented graphically by 3d models with a library of animated motions which are choreographed by the behavioral controllers.

Introduction

The real world is a busy and crowded place, full of people, animals and vehicles. In contrast, many game worlds are relatively static and empty. Usually one character moves under the player's control, a few other characters move autonomously, and maybe a few bits of the environment move in predetermined cycles. This paper describes techniques for populating a game world with large numbers of autonomous characters which behave in a sensible fashion. As required by their role, they can easily coordinate with or react to, each other, the environment, and the user's avatar.

Three key concepts are discussed in this paper. First, *behavioral models:* the control programs which serve as the brains of the autonomous characters. Second *spatial data structures* which are used to accelerate locality queries to allow real time performance. Third, the techniques used to drive animation from behavioral models.

A real time demonstration application, a "mini-game", is presented and used as an example of the concepts presented in this paper.

Related Work

Autonomous characters of some sort exist in most games. Any character not directly controlled by a human player must have some level of autonomy. This includes enemies, allies and sometimes, neutral bit players. The historical trend has been from very simplistic autonomous characters (which may just blindly follow scripts with little or no ability to react to their dynamic environment) towards nimble reactive characters [Reyn99], and finally towards characters which can learn [Gran98], reason [Lair00], and plan [Fung99].

The concepts presented here are an outgrowth of earlier work on simulating bird flocks [Reyn87] and on the underlying concept of steering behaviors [Reyn99]. An extensive bibliography on autonomous characters can be found in [Reyn99].

Directly relevant to the topic of this paper are existing games that contain an element of large groups which interact with, or respond to, the actions of the human user. Many of the Maxis simulation games have this property: *SimCity, SimAnt, SimTower,* and now: *The Sims.* Also games in the *Lemmings* and *Populous* franchises feature large number of autonomous characters which respond to the user. And in a slightly different vein, games like *Myth* and *Warcraft* allow troops to be grouped together and given a goal, which they pursue while avoiding obstacles and each other. Koei's upcoming game *Kessen* looks as if it will feature large groups of horse-mounted soldiers as an element of gameplay.

Behavioral Models

Autonomous characters depend on some sort of *behavioral model* or control program to drive them. In a virtual world, objects might be: static, periodic (running an animation cycle, a flickering flame for example), user controlled (like the human player's avatar) or they can have some degree of autonomy. Autonomous characters are procedurally driven by a control program which implements a behavioral model. This control program *maps* the character's *environment* into character *actions.* The character's environment consists of its virtual world (external environment, perhaps filtered through locality or its sensory perception) and its memory or other simulated mental processes (internal environment). A character's degree of autonomy can vary widely. On one hand an autonomous "car" might simply drive along a street, stopping when it comes to a red light. (Note that for the purposes of this discussion we will lump together a vehicle and its driver/pilot into a single entity.) On the other hand a character might have all the freedom of action afforded to the human player so it can play the role of an enemy or companion.

The characters described in this paper are based on a very simple physical model: a point mass with velocity and a local frame of reference (an orthonormal basis for a coordinate system). The local coordinate system is updated each simulation step to move with the position of the mass and to be aligned with its velocity. The character moves by applying a finite steering force to modify its velocity, and hence its position. Velocity is capped to roughly simulate drag or friction. The local coordinate system is used to attach an animated geo-

metrical model to the point mass. As a result the position, velocity, orientation, and visual appearance of the character is driven by the behavioral model, primarily through control of the steering force. (For a more detailed description of these concepts see [Reyn99].)

Other kinds of characters can express their behaviors with different kinds of action. A character's body could be represented by a physically-based dynamically balanced simulation of walking, providing both realistic animation and behavioral locomotion. While a more mechanical "character," like an autonomous airplane or spaceship, can be realistically portrayed with relatively simple distributed-mass rigid dynamic model. Another approach for articulated (legged) motion is to use a simple (say point-mass-based) locomotion model and an adaptive animation model, like an inverse-kinematics driven walk cycle, to bridge the gap between abstract locomotion and concrete terrain. Or a character's motion can be restricted to a fixed set of pre-animated segments (walk, run, stop, turn left...) which are either selected discretely or blended together.

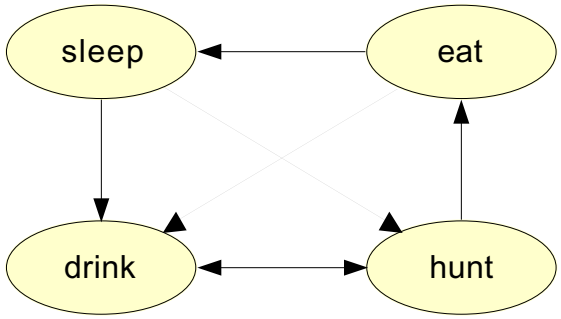

Figure 1: Finite State Machine diagram showing states and transitions between them for a simple predator character.

An autonomous character can have multiple distinct behavioral *states*. For example a fighter jet could be: taking off, landing, patrolling, intercepting, dog-fighting and so on. In each state there is active control, the jet must steer, adjust its thrust and take other actions based on its perception of the local environment, but these will be tempered by it current behavioral states. For example, running low on fuel would elicit difference responses while patrolling than when engaged in a dog-fight. Transitions from one behavioral state to another are triggered by changes in the character's internal/external environment: the detection of an enemy aircraft triggers the transition from patrolling state to intercepting state. In general, transitions from one behavioral state are only allowed to a subset of the other states. The traditional representation of of this concept is a "finite state machine" or FSM, see Figure 1.

Case study: Pigeons in the Park

To make the concepts described in this paper more concrete, a demonstration program called "Pigeons in the Park" was written in which the user/player interacts in real-time with a large group of simple autonomous characters. The demonstration takes place in a simulat-

ed grassy meadow, perhaps in an urban park. A flock of pigeon-like birds are on the ground looking for food. The user drives a toy radio controlled ("RC") car. The moving car worries the birds. If the car approaches slowly the pigeons walk away from its path. If the car gets too close, or moves too fast towards them, the birds panic and take flight. The panic is contagious, so otherwise calm birds will take off if enough of their neighbors do. When the pigeons fly they form a flock according to the *boids* model of coordinated group motion [Reyn87]. When they are are on the ground, they form a loosely organized 2D flock, essentially a herd. As shown in Figure 2, the virtual world of Pigeons in the Park includes the birds, the car, an undulating terrain, an invisible spherical obstacle which surrounds the birds (to keep them from wandering away), and some additional scenery such as a sky and some distant trees.

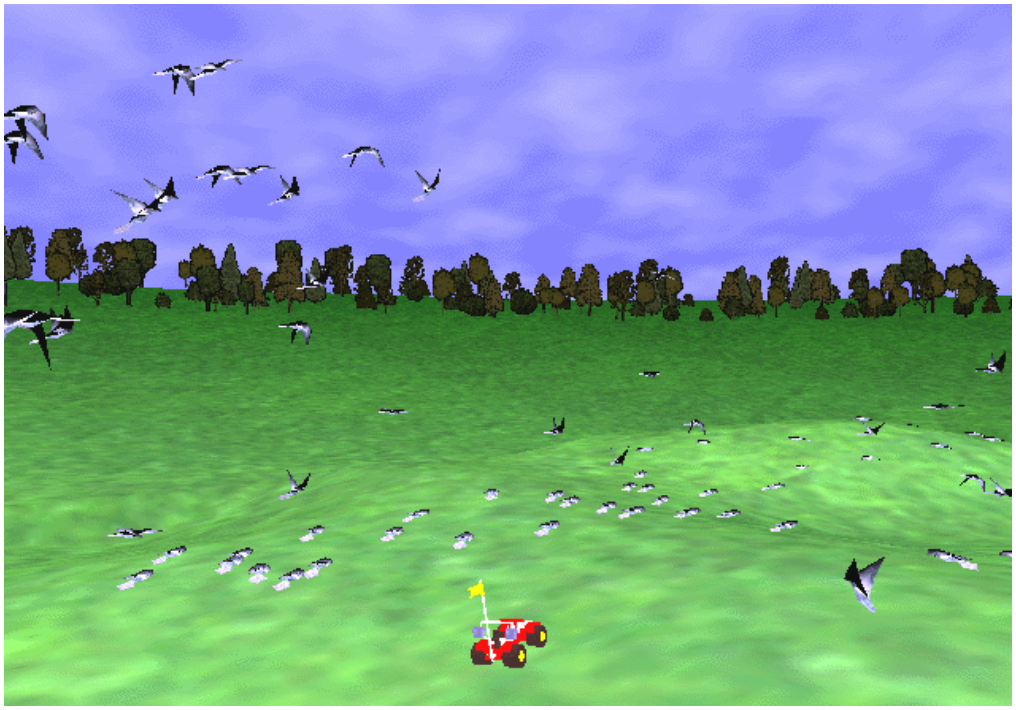

Figure 2: Typical view from *Pigeons in the Park*

In addition to annoying the pigeons by driving the RC car, the user can also push a button which simulates a loud noise, like a hand clap. Either a nearby car or the hand clap will startle the birds and cause them to take flight. In addition, the urge to take off is *contagious* so each bird is sensitive to the percentage of nearby birds who have recently taken off (they are not alarmed by birds simply flying by) and if this percentage exceeds a threshold they will take off too. As a result, if the user scares a single bird in a dense crowd, a large number of birds will take flight in response as a wave of alarm propagates through the crowd.

When startled the first time, a bird will fly a short distance away and then land again so it can return to feeding, as shown in the simple state diagram in Figure 3. But each bird has an *annoyance* value which gets ratcheted up each time time it gets startled, and then slowly de-

cays over time. If a bird is already annoyed when it gets startled it will fly further before it decides to land. As a result the first time a bird is startled it will fly a few feet from its initial position, if it is startled again before it has had time to calm down, it will fly further trying to get away from the bothersome user. Flight duration is roughly controlled by two parameters, one specifying the approximate amount of time the bird spend ascending after takeoff, the other specifying the amount of time before the bird begins to descend toward a landing. Both of these are affected by the annoyance status of the bird. All phases of flight are subject to the effects of obstacle avoidance and flocking, so individual flight paths a rarely a simple arc up and then down, but tend to follow what nearby flockmates are doing.

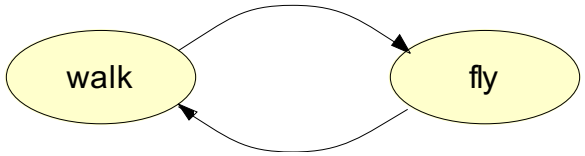

Figure 3: Behavioral state diagram for the pigeons.

For a given behavioral state, a bird's motion is determined by the composition of several *steering behaviors* [Reyn99]. Primarily they are flocking (according to the *boids* rules [Reyn87] which combine *separation, alignment* and *cohesion*) and avoiding obstacles (using the technique called *containment* in [Reyn99]) such as the terrain and the surrounding spherical enclosure. They also react to the user's avatar (the RC car) as described in the next section. While the birds are walking on the ground, their motion is constrained to be tangent to the terrain surface.

Animation for the birds is derived from a small set of pre-animated cycles and a few parameters from the behavioral model. Those behavioral parameters are: the character's position and orientation, the walking/flying flag, and (derived from the orientation) the rate of change of the characters glide slope (the angle between the ground plane and its path of motion). The pre-animated cycles are: *Walking, Flying, Gliding* and *WingFold*. When the birds are on the ground they are normally in the Walking animation, but the WingFold animation forms the transition form Flying to Walking. While in flight, the animation controller selects between Gliding and several variations of Flying based on the flight angle. Climbing uses the Flying animation at a high speed, level flight uses the Flying animation at a slower rate and descending uses the Gliding animation. Transitions between the animated clips are scheduled so that they do not occur until the current cycle gets to the point where it matches into the next selected cycle.

The Pigeons in the Park demo runs on a PlayStation2 at an update rate of 60 fields per second. There are 280 birds in the flock. The entire scene is composed of approximately 55000 triangles. Currently the demo runs on a PlayStation2 Development Tool (DTL-T10000) although it could presumably run on the PlayStation2 consumer unit as well. The code for the demo itself was written primarily in C and runs on the PlayStation2's *Emotion Engine* CPU. The demo uses its own 3d vector arithmetic library and uses the CDS scene graph rendering system [Nagy99] which include portions written in macro and microcode for

the PlayStation2's Vector Units. The actual C language data structures used to represent the pigeons is shown in Figure 4.

```
typedef struct pigeon
{
    boidObject boid;          /* base boid object (see below) */
    int onGround;             /* walking or flying */
    float annoyance;          /* how annoyed is this pigeon? */
    int framesSinceTakeOff;   /* actual flight time so far */
    int framesBeforeDescent;  /* controls approx duration of flight */
    int ascentFrames;         /* controls approx duration of ascent */
} pigeon;

typedef struct boidObject
{
    struct boidObject* next;  /* for linking flock */
    vecObject position;       /* center position */
    vecObject side;           /* side    (x) basis vector */
    vecObject up;             /* up      (y) basis vector */
    vecObject forward;        /* forward (z) basis vector */
    vecObject velocity;       /* current velocity vector */
    vecObject acceleration;   /* current acceleration vector */
    float mass;               /* magnitude of point mass */
    float animationPhase;     /* where in animation loop */
    lqClientObject lqco;      /* locality query proxy */
    int skipCounter;          /* think skip phase */
} boidObject;
```

Figure 4: C structures for Pigeon and its "base class" Boid.

Reacting to the User

Autonomous characters such as those used in "Pigeons in the Park" respond to the user in two different ways: *discrete* and *continuous* reactions. The user can trigger the bird's transition between walking and flying behavioral state either by the "hand clap" signal, or by the velocity and position of the RC car (the user's avatar). Switching states, and so taking flight, is an example of a *discrete* reaction to the user.

In addition, the pigeons want to keep their distance from the user's car, particularly when it is moving fast. When the pigeons sense that the car is getting too close they move in a direction that will take them away from the car and away from its current path. This tendency to blend avoidance motion in with the character's other behavior is an example of a *continuous* reaction to the user.

In both cases these reactions are driven by a dynamically-shaped neighborhood around the car that depends on its position and velocity. (*Velocity* is used here in the sense of unit *heading* times *speed*.) The neighborhood is specified in terms of a radius and a velocity

weighting. Its shape is a half circle behind the car, and in front of the car it is a half of an ellipse, aligned with the car's heading, whose length (eccentricity) is proportional to the car's speed times the given velocity weighting. Figure 5 shows the shape of this neighborhood when the car is stopped, going slow and going fast.

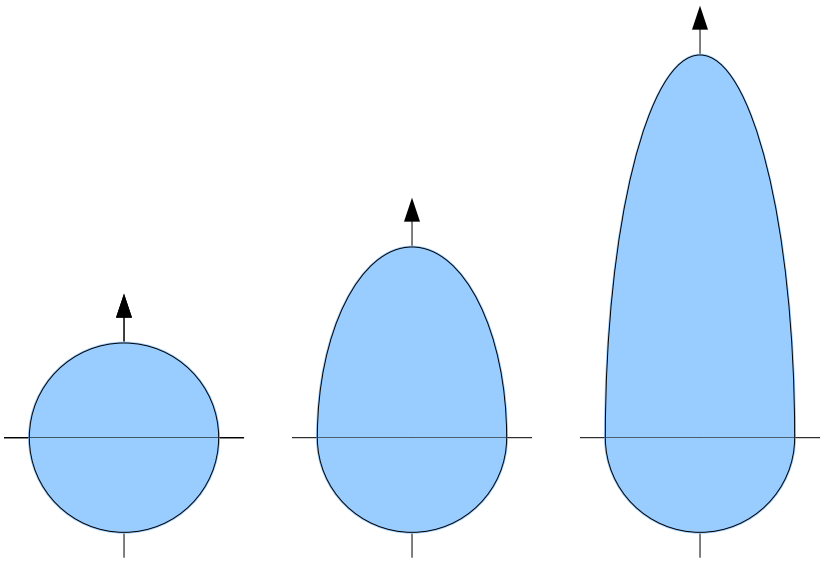

Figure 5: The avatar's neighborhood elongates with speed.

The avatar neighborhood which triggers the pigeon's desire to move away is larger (and less elongated with speed) than the neighborhood which triggers their desire to take off. When pigeons are within the move-away zone, their behavior is to blend in motion which takes them away from potential trouble, while still engaging in their other behaviors (flocking and obstacle avoidance). The avatar avoidance direction is a combination of moving away from its current position and its current path, as indicated in Figure 6.

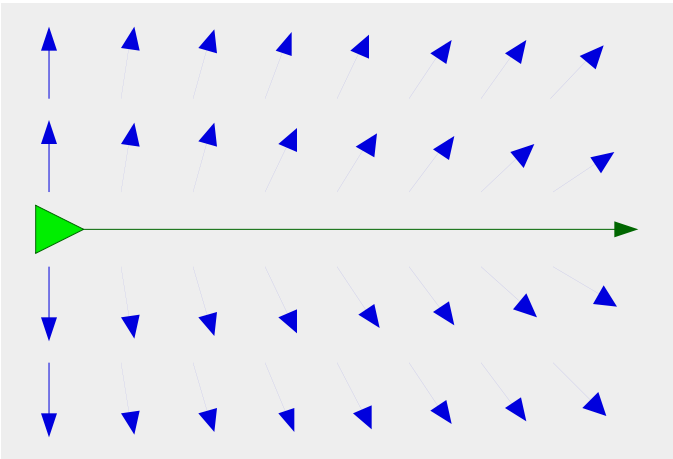

Figure 6: Pigeons near the RC car (in the avatar's neighborhood) want to move away from both the car and its current path.

Realtime Performance

For a multi-agent simulation like "Pigeons in the Park" to run at the desired update rate of 60 fields per second, it is important to keep the overall computational cost low. This section will describe the three main sources of computational load, and how they can each be kept to a minimum.

Rendering

As in any realtime computer animation application, a significant cost is rendering the 3d shaded, textured, scene, in perspective. This cost was minimized by the simple expedient of using someone else's fast code. In this case, realtime rendering was provided by Gabor Nagy's CDS [Nagy99] scene graph rendering utility, which was developed at SCEA R&D specifically for PlayStation2. In the end, rendering turned out to require a fairly small portion (15% to 20%) of the one-sixtieth of a second available for each display cycle. This is due to several factors such as: the power of the PlayStation2 hardware, the efficiency of the CDS software, and the relatively modest demands of the scene in terms of polygon count and simple shading models.

Thinking

A second significant computational cost is the time taken for each of the autonomous bird characters to *think.* As described in the section on Behavioral Models, the "brain" of an autonomous character is a program that must be run to determine what actions to take based on its current environment. While these computations are relatively simple, they must be performed for each of the characters, and so the cost is multiplied by the number of characters. This sort of linear scaling of asymptotic complexity is known as *Order n* and typically written *O(n).* (For an introduction to asymptotic complexity, see [Fren99].)

Because the 60Hz update rate of the simulation and graphics is rather fast compared to the motion of the characters, a simple way to reduce the cost of the behavioral model is to decouple it from the main simulation rate and run the behavioral computation at a lower rate. In the Pigeons in the Park demo it was sufficient to have the pigeons "think" at 10Hz, one sixth of the animation rate. That is, the control program for a given character was run just one out of every six simulation steps (video fields). Between those "thoughtful" frames, the character would continue to take the same action. In the Pigeons in the Park demo this means that the character will apply the same steering force vector for six subsequent simulation steps.

In actuality, the steering changes continuously because the underlying physics model does damping (temporal blending) on acceleration due to steering forces. Also the character's "thinking" about obstacle avoidance is done at a higher rate (approximately 30Hz). Experiments showed that predicting potential collisions too infrequently interfered with smooth and effective obstacle avoidance. This second decoupling allows good obstacle avoidance be-

havior while allowing the main thinking rate to be quite low. (In the natural world, convention wisdom suggests that only modest computational power is required for "bird brains.") Finally note that while each bird does its main thinking at 10Hz, the phase of these cycles is randomized so that approximately one-sixth of the population thinks at each frame.

Locality Queries

The third and potentially most troublesome source of computational effort is the cost of doing locality (proximity) queries between characters. A behavioral simulation of a group (or any other *spatially-explicit multi-agent system*) can be thought of as an *interacting particle system.* These systems all have the property that each of the particles need to interact with all of the other particles, if only to decide which ones they *really* want to interact with. As a result interacting particle system have an asymptotic complexity of $O(n^2)$. So for example, doubling the number of characters in a behavioral simulation quadruples the amount of time needed to do the locality queries. The significant point is that no matter how fast each locality query can be done, as the population increases, the cost of the locality queries will eventually dominate all other computational costs.

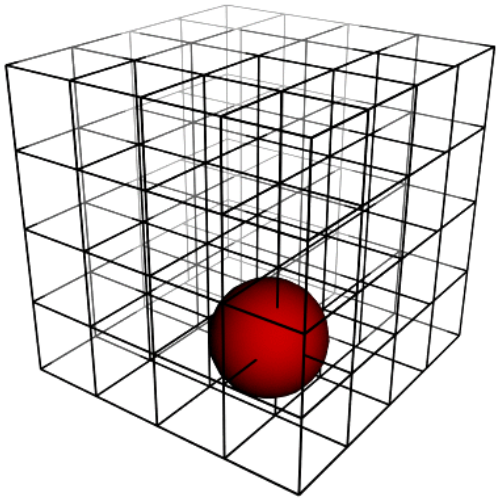

Figure 7: A query sphere inside the bin-lattice spatial data structure.

In a behavioral simulation of a group, all important interactions are between nearby individuals. The $O(n^2)$ part of the algorithm is simply to find the few neighbors of each individual. An effective way to accelerate these *locality queries* is to store the characters in some sort of *spatial data structure* (aka spatial database). Informally, the intent is to keep the characters "pre-sorted" based on their location in space, so that we can quickly find which are in a given neighborhood without having to examine the entire population.

Computational geometers have developed many types of spatial data structures which can be used to accelerate locality queries (see for example [Samet89]). In the Pigeons in the Park demo a simple bin-lattice spatial subdivision technique was used. Binary space partitioning (BSP) trees can be used for this application, as can others. A technique proposed by Popovic [Popo94] may provide the best performance.

In bin-lattice spatial subdivision, a box-shaped region of space is divided into a collection of smaller boxes called "bins". For simplicity the edges of the big box, and the dividing planes which form the faces of the smaller boxes are all aligned with the global axes. At the beginning of the simulation, characters are distributed into bins based on their initial position. Each time they move they check to see if they have crossed into a new bin, and if so, update their bin membership. The big box is selected to surround the area of interest (in this case, the meadow in the pigeon's park). The number of subdivisions along each axis is selected to provide a good tradeoff between more precise localization of characters and less bin switching overhead as characters move. A 10x10x10 lattice of 1000 bins seems to be a good starting point, but actual results depend on many factors.

In this implementation a locality query is performed by specifying a sphere (a location in space and a radius, see Figure 7) and a function. The locality query code identifies all of the bins which at least partially overlap with the sphere (see Figure 8) it examines each of the objects in each of those bins and tests to see if they fall within the query sphere. If so, the supplied query function is applied to the object. The bins themselves consist of a single pointer into a doubly-linked list of proxy objects which contain an XYZ location and a pointer to the real object (in this case, a pigeon character). The doubly-linked list structure allows bin membership to be changed in constant time (rather than proportional to bin population).

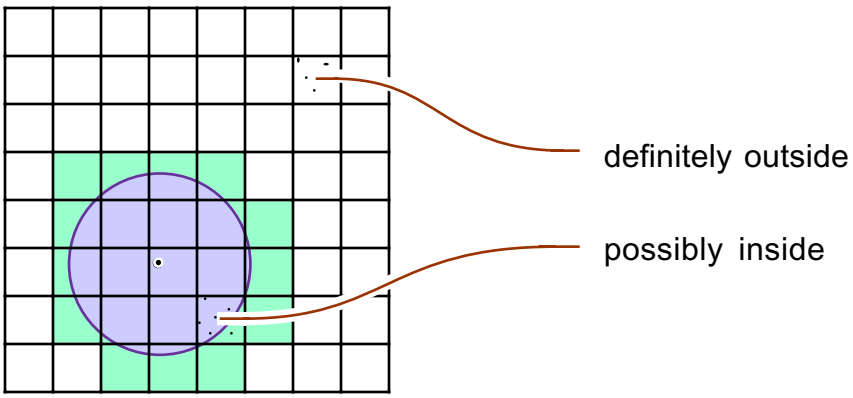

Figure 8: objects (dots) in shaded region *may* be inside sphere, but dots outside the shaded region *cannot* be inside the sphere

Because the characters maintain a minimum separation distance, and so a maximum density, the number of characters within a giving radius is bounded. Call the bound *k.* Therefore using spatial subdivision, an interacting particle system (like the flock model) is accelerated from $O(n^2)$ to $O(nk)$ which, if we consider K to be a constant, is asymptotically

equivalent to O(n). In one test using a flying flock of 1000 simulated birds (versus about 300 in the Pigeons in the Park demo) performing locality queries with the bin-lattice spatial subdivision was about 16 times faster than the naive $O(n^2)$ implementation.

Conclusions

This paper has described a realtime system in which a large group of autonomous characters move themselves around, reacting in a reasonable way to each other, to obstacles in their environment, and the the user's avatar. The implementation is based on a very simple physical model for the characters, a reactive behavioral model composed of multiple behavioral states, and steering behaviors. The characters are portrayed by attaching pre-animated bodies, which are selected based on the output of the behavioral model. Realtime performance is obtained by decoupling the behavioral model from the fast graphical update rate, and by the use of a lattice-bin spatial subdivision technique to accelerate the locality queries needed by the behavioral model of an interacting group of characters.

Acknowledgments

I especially wish to thank my direct collaborators on the "Pigeons in the Park" project. Attila Vass designed and implemented the main framework of the application, creating the virtual world, and the bodies of the characters into which I could plug my behavioral models. Attila also handled all software and media integration issues, and converted some of my vector library routines to run on the PlayStation2 Vector Unit. Guy Burdick created the art resources for this project building 3d models, animation and textures for the characters and environment. Gabor Nagy designed and implemented the CDS modeling and realtime rendering system we used for our demonstration.

Beyond these direct contributors, this work was a product of the supportive environment of the Research and Development group of Sony Computer Entertainment America. I wish to thank all of my colleagues in R&D and particularly our boss, Dominic Mallinson, Director of Technology. The diverse interests of the members of the R&D group, along with the rare freedom we are given to pursue new technologies make this a great place to work. For the opportunity to work in this stimulating environment, we are indebted to the support and foresight of Phil Harrison, Shin'ichi Okamoto, and others in the corporate management of SCEA and SCEI.

Thanks to Glenn Entis (CEO of DreamWorks Interactive) who suggested the basic concept of Pigeons in the Park in a conversation early in 1999, noting how kids seemed to never tire of chasing birds off the ground.

Finally I wish to thank my loving and supportive wife Lisa, and our two wonderful children Eric and Dana, both of whom are enthusiastic natural-born pigeon-chasers.

References

[Fung99] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos (1999) "Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters" in **Computer Graphics**, volume 33 Annual Conference Series (Proceedings of SIGGRAPH 99) pages 29-38. http://www.dgp.toronto.edu/~funge/sig99/fttsig.pdf

[Gran98] Steve Grand and Dave Cliff (1998) "Creatures: Entertainment software agents with artificial life" in **Autonomous Agents and Multi-Agent Systems**, 1(2).

[Lair00] John E. Laird (2000) "It Knows What You're Going To Do: Adding Anticipation to a Quakebot" to appear in the AAAI 2000 Spring Symposium Series: **Artificial Intelligence and Interactive Entertainment**, March 2000.

[Popo94] Zoran Popovic (1994) "The Rapid Simulation of Proximal-Interaction Particle Systems" unpublished manuscript. http://sulfuric.graphics.cs.cmu.edu/~zoran/actin/pop.ps

[Reyn87] Craig W. Reynolds (1987) "Flocks, Herds, and Schools: A Distributed Behavioral Model" in **Computer Graphics** 21(4) (SIGGRAPH 87 Conference Proceedings) pages 25-34. http://www.red.com/cwr/boids.html

[Reyn99] Craig Reynolds (1999) "Steering Behaviors For Autonomous Characters" in Conference Proceedings of the **1999 Game Developers Conference**, pages 763-782. http://www.red.com/cwr/steer/

[Nagy99] CDS Documentation, unpublished working paper (CDS.pdf).

[Samet89] Hanan Samet (1989) **Applications of Spatial Data Structures**, Addison-Wesley.

[Fren99] Michael Frenklach and Kal Sastry (1999), "Chapter 20: Big-O Notation" in course notes to E77N: http://www.me.berkeley.edu/~e77/lecnotes/ch20/ch20.htm

version 1: January 31, 2000
version 2: February 1, 2000