Steering Behaviors For Autonomous Characters

Craig W. Reynolds

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: Animation Techniques, Virtual/Interactive Environments, Games, Simulation, behavioral animation, autonomous agent, situated, embodied, reactive, vehicle, steering, path planning, path following, pursuit, evasion, obstacle avoidance, collision avoidance, flocking, group behavior, navigation, artificial life, improvisation.

Abstract

This paper presents solutions for one requirement of autonomous characters in animation and games: the ability to navigate around their world in a life-like and improvisational manner. These "steering behaviors" are largely independent of the particulars of the character's means of locomotion. Combinations of steering behaviors can be used to achieve higher level goals (For example: get from here to there while avoiding obstacles, follow this corridor, join that group of characters...) This paper divides motion behavior into three levels. It will focus on the middle level of steering behaviors, briefly describe the lower level of locomotion, and touch lightly on the higher level of goal setting and strategy.

Introduction

Autonomous characters are a type of *autonomous agent* intended for use in computer animation and interactive media such as games and virtual reality. These agents represent a character in a story or game and have some ability to improvise their actions. This stands in contrast both to a character in an animated film, whose actions are scripted in advance, and to an "avatar" in a game or virtual reality, whose actions are directed in real time by a human player or participant. In games, autonomous characters are sometimes called *non-player characters*.

An autonomous character must combine aspects of an autonomous robot with some skills of a human actor in improvisational theater. These characters are usually not real robots, and are certainly not human actors, but share some properties of each.

The term "autonomous agent" is used in many contexts, so the following is an attempt to locate the terminology of this paper in relation to other fields of study. An autonomous agent can exist in isolation, or it can be *situated* in a world shared by other entities. A "data mining" agent is an example of the former, and a controller for a power grid is an example of the latter. A situated agent can be reactive (instinctive, driven by stimulus) or it can be deliberative ("intellectual" in the classic AI sense). An autonomous agent can deal exclusively with abstract information ("softbot", "knowbot", or "information agent") or it can be *embodied* in a physical manifestation (a typical industrial robot or an autonomous vehicle). Combinations of *situated, reactive,* and *embodied* define several distinct classes of autonomous agents.

The category of situated, embodied agents usually suggests autonomous robots: mechanical devices that exist in the real world. Sometimes robots are studied via computational simulation. But that practice is viewed with suspicion by purists in the robotics field because the simulation may diverge from reality in unpredictable ways. There is another class of situated, embodied agent based on a computational model. This paper will use the term *virtual* (as in virtual reality) to denote these agents which, rather than being simulations of a mechanical device in the real world, are instead **real** agents in a virtual world. (Analogous to a *physically-based model* in computer animation.) Hence the autonomous characters of this paper's title are: situated, embodied, reactive, virtual agents.

The term *behavior* has many meanings. It can mean the complex action of a human or other animal based on volition or instinct. It can mean the largely predictable actions of a simple mechanical system, or the complex action of a chaotic system. In virtual reality and multimedia applications, it is sometimes used as a synonym for "animation." In this paper the term behavior is used to refer to the improvisational and life-like actions of an autonomous character.

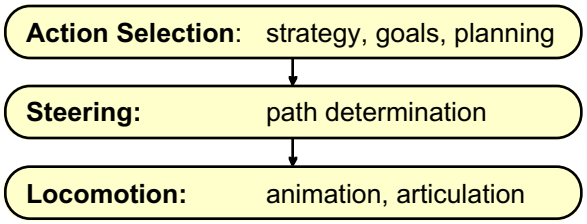

Figure 1: A hierarchy of motion behaviors

The behavior of an autonomous character can be better understood by dividing it into several layers. These layers are intended only for clarity and specificity in the discussion that will follow. Figure 1 shows a division of motion behavior for autonomous characters into a hierarchy of three layers: *action selection*, *steering*, and *locomotion*. Certainly other dissections are possible. A similar three layer hierarchy is described by Blumberg and Galyean [Blumberg 95], they call the layers: *motivation*, *task*, and *motor*. Note that while the behavioral hierarchy presented here is intended to be widely applicable to motion behaviors, it is not well suited for other types of autonomous actions, for example the conversational behaviors of a "chatterbot" require a significantly different structure.

Consider, for example, some cowboys tending a herd of cattle out on the range. A cow wanders away from the herd. The trail boss tells a cowboy to fetch the stray. The cowboy says "giddy-up" to his horse and guides it to the cow, possibly avoiding obstacles along the way. In this example, the trail boss represents *action selection:* noticing that the state of the world has changed (a cow left the herd) and setting a goal (retrieve the stray). The *steering* level is represented by the cowboy, who decomposes the goal into a series of simple subgoals (approach the cow, avoid obstacles, retrieve the cow). A subgoal corresponds to a steering behavior for the cowboy-and-horse team. Using various control signals (vocal commands, spurs, reins) the cowboy steers his horse towards the target. In general terms, these signals express concepts like: go faster, go slower, turn right, turn left, and so on. The horse implements the *locomotion* level. Taking the cowboy's control signals as input, the horse moves in the indicated direction. This motion is the result of a complex interaction of the horse's visual perception, its sense of balance, and its muscles applying torques to the joints of its skeleton. From an engineering point of view, legged locomotion is a very hard problem [Raibert 91], [Hodgins 95], but neither the cowboy nor the horse give it a second thought.

This paper will focus on steering, the middle layer of the behavioral hierarchy. It will briefly describe a simple model of the locomotion layer, but only in enough detail to provide a concrete foundation for the discussion of various steering behaviors. There will be some brief discussion of action selection, but primarily in the context of combining and blending basic steering behaviors.

Path-finding is a topic related to, but separate from, the subject of this paper. Path-finding algorithms such as A* and Dijkstra's operate on networks (often representing grids) and essentially solve mazes. Such a solution could serve as a specification to the steering techniques described in this paper. An analogy might be to compare the written driving instructions for getting from one place to another with the act of driving the car along that route. For an excellent over view of path-finding see [Reese 99].

In order to understand the thrust of this work, it should be noted that the steering behaviors discussed here relate to “fast” motion: running versus crawling. This is an informal notion, but is meant to suggest that the typical velocity of a character is large relative to its maximum turning acceleration. As a result, the steering behaviors must anticipate the future, and take into account eventual consequences of current actions.

Related Work

Steering behaviors for autonomous characters draw on a long history of related research in other fields. Autonomous machines, servomechanisms, and control theory have their roots in the 1940s as described in Norbert Wiener’s 1948 book *Cybernetics, or Control and communication in the Animal and the Machine* [Wiener 48]. The term *cybernetics* came from a Greek word meaning *steersman*. During the late 40s neurophysiologist Grey Walter constructed autonomous robotic *turtles* [Walter 50] which embodied several of the steering behaviors described here and were among the first machines to exhibit emergent life-like behavior.

In the early 1980s Valentino Braitenberg extrapolated Walter’s prototypes into thought experiments about a series of fanciful “vehicles” with progressively more complex behaviors [Braitenberg 84]. David Zeltzer began applying techniques and models from artificial intelligence to animation applications [Zeltzer 83]. And in 1987, I created an animated behavioral model of bird flocks using techniques closely related to those presented in this paper [Reynolds 87].

The list below of related research is divided into three general categories: robotics, artificial intelligence, and artificial life, although in some cases the distinction is somewhat arbitrary. Generally these works are oriented towards animation to some extent: they are located in the overlap between animation (or games, VR, and multimedia) and these three other fields.

Work related to robotics. Rodney Brooks popularized the then-radical notion of building reactive controllers for robotic systems [Brooks 85]. While originally inspired by ethological (animal behavior) research, the work of Ron Arkin [Arkin 87, 89, 92] has centered on application of steering behaviors to mobile robots. Arkin’s research has paralleled much of the work presented in this paper, but his *schema* (perception›action mappings) are expressed in terms of potential field models as opposed to the procedural approach described here. In some cases this is a distinction without a difference, but in other cases (such as obstacle avoidance) it leads to significantly different agent behavior. Marc Raibert and Jessica Hodgins

both began in legged robotics research and now both work in animation applications of physically realistic legged systems. In both cases, their work has touched on steering and path planning aspects of these systems [Raibert 91, 91b], [Hodgins 95]. Work by Zapata *et al*. on steering controllers for fast mobile robots focused on strategies which had to deal with momentum and other aspects of fast mechanical motion [Zapata 1992]. Maja Mataric has worked extensively in collective robotics [Mataric 93] and a central theme of this work is steering.

Work related to artificial intelligence. Ken Kahn created an early system that generated animation of character motion from story descriptions [Kahn 79]. David Zeltzer [Zeltzer 83, 90] pioneered AI-based animation, popularizing the idea of abstract "task level" specification of motion. Gary Ridsdale [Ridsdale 87] created characters capable of improvising complex motion, getting from A to B while avoiding static obstacles and other actors. Steve Strassmann's Desktop Theater work [Strassmann 1991] extended these notions to include handling of props and emotional portrayal. Mônica Costa's agent-based behavioral animation work [Costa 90] allows a character to navigate around a house while reactively avoiding obstacles. Research on improvisational, dramatic characters, which touches on steering behavior, is ongoing at Project Oz (and now Zoesis) by Joseph Bates *et al.* [Bates 92] and at The Virtual Theater Project by Barbara Hayes-Roth *et al.* [Hayes-Roth 96].

Work related to artificial life (and other fields). The 1987 *boids* model of flocks, herds, schools and related group motion [Reynolds 87], decomposed this complex group behavior to three simple steering behaviors at the individual level. The following year related steering behaviors for obstacle avoidance [Reynolds 88] were presented. At the 1987 Artificial Life Workshop Mitchel Resnick presented work on autonomous vehicles implemented in LEGO LOGO [Resnick 89] and Michael Travers demonstrated his AGAR Animal Construction Kit [Travers 89]. (See also more recent work by these authors [Resnick 93] and [Travers 94].) Steering behaviors were a key element in The Virtual Fishtank, a multiuser VR installation at The Computer Museum created by teams from MIT's Media Lab and NearLife [Resnick 98]. Armin Bruderlin procedurally generated goal directed animation of human walking [Bruderlin 1989]. Randall Beer's dissertation on an artificial cockroach [Beer 90] is noteworthy for the depth and complexity of its neuroethological model. Central to this model are neural implementation of several tropisms (such as chemotaxis and thigmotaxis) which are direct analogs of steering behaviors described below. In [Wilhelms 90] Jane Wilhelms and Robert Skinner investigate architectures for vehicle-like characters. Thalmann *et al.* created behavioral animation characters who navigated down corridors and around obstacles using vision as simulated with 3D rendering [Thalmann 90]. Michiel van de Panne created controllers for tasks like parallel parking of an automobile using state-space search [van de Panne 90]. G. Keith Still has modeled large human crowds using a model of the steering behavior of each individual [Still 94]. Using a modified genetic algorithm, Karl Sims simultaneously evolved brains and bodies for artificial creatures for various styles of locomotion and for goal seeking [Sims 94]. In work first reported at SAB94 and updated at SAB96 [Cliff 96], Cliff and Miller coevolved pursuit and evasion behaviors for predator and prey agents. Xiaoyuan Tu *et al.* developed an elaborate and strikingly realistic model of the biomechanics, locomotion, perception, and behavior of fish in [Tu 94, 96] which included physically based locomotion, steering behaviors, and an ethologically based system for action selection. In [Blumberg 94] Bruce Blumberg described a detailed mechanism for complex

action selection and with Tinsley Galyean in [Blumberg 95] discussed the design for a VR character capable of both autonomous improvisation and response to external direction. One application of these characters was in the ALIVE system [Maes 95] by Patties Maes *et al.* The Improv system by Ken Perlin and Athomas Goldberg [Perlin 96] also covers the gamut from locomotion to action selection, but uses a unique approach based on behavioral scripting and Perlin's 1985 procedural synthesis of textures [Perlin 85] applied to motion. James Cremer and colleagues have created autonomous drivers to serve as "extras" creating ambient traffic in interactive automobile driving simulators [Cremer 96]. Robin Green (of Bullfrog/EA) has developed a mature system for autonomous characters used in Dungeon Keeper 2 which was inspired in part by an early draft of this paper. Dave Pottinger has provides a detailed discussion of steering and coordination for groups of characters in games [Pottinger 1999].

Locomotion

Locomotion is the bottom of the three level behavioral hierarchy described above. The locomotion layer represents a character's *embodiment*. It converts control signals from the *steering* layer into motion of the character's "body." This motion is subject to constraints imposed by the body's physically-based model, such as the interaction of momentum and strength (limitation of forces that can be applied by the body).

As described above, a cowboy's horse can be considered as an example of the locomotion layer. The rider's steering decisions are conveyed via simple control signals to the horse who converts them into motion. The point of making the abstract distinction between steering and locomotion is to anticipate "plugging in" a new locomotion module. Imagine lifting the rider off of the horse and placing him on a cross-country motorcycle. The goal selection and steering behavior remain the same. All that has changed is the mechanism for mapping the control signals (go faster, turn right, ...) into motion. Originally it involved legged locomotion (balance, bones, muscles) and now it involves wheeled locomotion (engine, wheels, brakes). The role of the rider is unchanged.

This suggests that with an appropriate convention for communicating control signals, steering behaviors can be completely independent of the specific locomotion scheme. Although in practice it is necessary to compensate for the "agility" and different "handing characteristics" of individual locomotion systems. This can be done by adjusting tuning parameters for a given locomotion scheme (which is the approach taken in the steering behaviors described below) or by using an adaptive, self-calibrating technique (the way a human driver quickly adapts to the characteristics of an unfamiliar automobile). In the first case a steering behavior might determine via its *a priori* tuning that the character's speed in a given situation should be 23 mph, in the second case it might say "slow down a bit" until the same result was obtained.

The locomotion of an autonomous character can be based on, or independent from, its animated portrayal. A character could be represented by a physically-based dynamically balanced simulation of walking, providing both realistic animation and behavioral locomotion. Or a character may have a very simple locomotion model (like described in the next section) to which a static (say a spaceship) or pre-animated (like a human figure performing a walk cycle) portrayal is attached. A hybrid approach is to use a simple locomotion model and an adaptive animation model, like an inverse-kinematics driven walk cycle, to bridge the gap between abstract locomotion and concrete terrain. Finally, locomotion can be restricted to the motion

inherent in a fixed set of pre-animated segments (walk, run, stop, turn left...) which are either selected discretely or blended together.

A Simple Vehicle Model

The approach taken in this paper is to consider steering behaviors as essentially independent from the underlying locomotion scheme. A simple locomotion model will be presented in order to make the discussion of steering behaviors more concrete. This locomotion model will be based on a simple idealized *vehicle*. The choice of the term "vehicle" is inspired to some degree by [Braitenberg 84]. It is intended to encompass a wide range of conveyances, from wheeled devices to horses, from aircraft to submarines, and (while certainly stretching the terminology) to include locomotion by a character's own legs. The vehicle model described here is so simplistic and generic that it is an equally good (or equally bad) approximation to all of those.

This vehicle model is based on a point mass approximation. On the one hand that allows a very simple and computationally cheap physically-based model (for example, a point mass has velocity (linear momentum) but no moment of inertia (rotational momentum)). On the other hand, it cannot be a very compelling physical model because point masses do not exist in the real world. Any physical object with mass must have a non-zero radius and hence a moment of inertia. This use of an oversimplified non-physical vehicle model is merely for convenience and intended to be "without loss of generality" — it should always be possible to substitute a more plausible, more realistic physically based vehicle model.

A point mass is defined by a *position* property and a *mass* property. In addition, the simple vehicle model includes a *velocity* property. The velocity is modified by applying forces. Because this is a vehicle, these forces are generally self-applied, and hence limited. For example, a typical force which adjusts a vehicle's velocity is *thrust*, generated by the vehicle's own power plant, and hence limited in magnitude by the capacity of the power plant. For the simple vehicle model, this notion is summarized by a single "maximum force" parameter (*max_force*). Most vehicles are characterized by a top speed. Typically this limitation is due to the interaction between acceleration due to their finite thrust and the deceleration due to viscous drag, friction, or (in legged systems) the momentum of reciprocating parts. As an alternative to realistic simulation of all these limiting forces, the simple vehicle model includes a "maximum speed" parameter (*max_speed*). This speed limit is enforced by a kinematic truncation of the vehicle's velocity vector. Finally, the simple vehicle model includes an *orientation*, which taken together with the vehicle's position form a velocity-aligned local coordinate space to which a geometric model of the vehicle can be attached. (The terms *localize* and *globalize* will be used in this paper to connote transforming vectors into and out of this local space.)

Simple Vehicle Model:

mass	scalar
position	vector
velocity	vector
max_force	scalar
max_speed	scalar
orientation	*N* basis vectors

For a 3D vehicle model, the *position* and *velocity* vector values have three components and the *orientation* value is a set of three vectors (or a 3x3 matrix, or a quaternion). For a 2D vehicle, the vectors each have two components, and the *orientation* value is two 2D basis vectors or can be represented as a single scalar heading angle.

The physics of the simple vehicle model is based on forward Euler integration. At each simulation step, behaviorally determined steering forces (as limited by *max_force*) are applied to the vehicle's point mass. This produces an acceleration equal to the steering force divided by the vehicle's mass. That acceleration is added to the old velocity to produce a new velocity, which is then truncated by *max_speed*. Finally, the velocity is added to the old position:

```
steering_force = truncate (steering_direction, max_force)
acceleration = steering_force / mass
velocity = truncate (velocity + acceleration, max_speed)
position = position + velocity
```

The simple vehicle model maintains its velocity-aligned local space by *incremental adjustment* from the previous time step. The local coordinate system is defined in terms of four vectors: a position vector specifying the local origin, and three direction vectors serving as the basis vectors of the space. The basis vectors indicate the direction and length of coordinate units in each of three mutually perpendicular directions relative to the vehicle. These axes will be referred to here as *forward*, *up*, and *side*. (These correspond, of course, to *X*, *Y* and *Z* axes of R^3. But some people think *up* is obviously *Y* while some think it is obviously *Z*. The descriptive terms will be used in place of the Cartesian names for clarity.)

In order to remain aligned with velocity at each time step, the basis vectors must be rotated into a new direction. (If velocity is zero the old orientation is retained.) Instead of using explicit rotations, the local space is reconstructed using a combination of substitution, approximation, and reorthogonalization. We start with the new velocity and an approximation to the new *up* direction. For example, the old *up* direction can be used as an approximation to the new *up*. We use the vector cross product operation to construct the new basis vectors:

```
new_forward = normalize (velocity)
approximate_up = normalize (approximate_up)      // if needed
new_side = cross (new_forward, approximate_up)
new_up = cross (new_forward, new_side)
```

The basic idea is that the approximate *up* is nearly perpendicular to the new *forward* direction, because frame-to-frame changes in orientation are typically small. The new *side* direction will be perpendicular to new *forward*, from the definition of cross product. The new *up* is the cross product of the perpendicular *forward* and *side* and so is perpendicular to each.

The concept of "velocity alignment" does not uniquely specify an orientation. The degree of freedom corresponding to rotation around the *forward* axis (also known as *roll*) remains unconstrained. Constructing the new local space relative to the previous one (by, for example, using the old *up* direction as the initial approximation to the new one) will ensure that the roll orientation at least remains consistent. Defining the "correct" roll value requires further heuristics, based on the intended use of the vehicle model.

For a "flying" vehicle (like aircraft, spaceship, and submarines) it is useful to define roll in terms of *banking*. The basic idea of banking is to align the "floor" of the vehicle (-*up* axis) with the apparent gravity due to centrifugal force during a turn. Conversely we want the *up* direction to

align with the centripetal force that produced the maneuver. In the presence of gravity, the down direction should align with the sum of turning acceleration and gravitational acceleration. We also want to add in the current orientation in order to damp out abrupt changes in roll. So to implement banking in the simple vehicle model, the approximate *up* direction is a weighted sum of: steering acceleration, gravitational acceleration, and the old *up*.

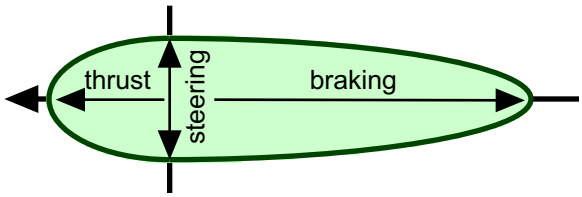

Figure 2: asymmetrical steering forces

For a "surface hugging" (wheeled, sliding, or legged) vehicle, we want to both constrain the vehicle's position to the surface and to align the vehicle's *up* axis to the surface normal. In addition the velocity should be constrained to be purely tangential to the surface. These requirements can be easily met if the surface manifold is represented in such a way that an arbitrary point in space (corresponding to the old vehicle position) can be mapped to: (1) the nearest point on the surface, and (2) the surface normal at that point. The velocity can be made tangent by subtracting off the portion normal to the surface. The vehicle's position is set to the point on the surface, and the surface normal becomes its *up* axis.

In this simple vehicle model, the control signal passed from the steering behaviors to the locomotion behavior consists of exactly one vector quantity: a desired steering force. More realistic vehicle models would have very different sets of control signals. For example an automobile has a steering wheel, accelerator and brake each of which can be represented as scalar quantities. It is possible to map a generalized steering force vector into these scalar signals: the *side* component of the steering vector can be interpreted as the steering signal, the *forward* component of the steering vector can be mapped into the accelerator signal if positive, or into the brake signal if negative. These mappings can be asymmetrical, for example a typical automobile can decelerate due to braking much faster than it can accelerate due to engine thrust, as shown in Figure 2.

Because of its assumption of velocity alignment, this simple vehicle model cannot simulate effects such as skids, spins or slides. Furthermore this model allows the vehicle to turn when its speed is zero. Most real vehicles cannot do this (they are "non-holonomic") and in any case it allows undesirably large changes in orientation during a single time step. This problem can be solved by placing an additional constraint on change of orientation, or by limiting the lateral steering component at low speeds, or by simulating moment of inertia.

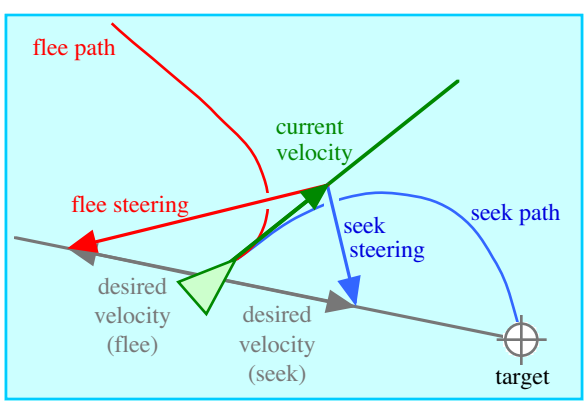

Figure 3: **seek** and **flee**

Steering Behaviors

This discussion of specific steering behaviors assumes that locomotion is implemented by the simple vehicle model described above, and is parameterized by a single steering force vector. Therefore the steering behaviors are described in

terms of the geometric calculation of a vector representing a desired steering force. Note that generally the magnitude of these steering vectors is irrelevant, since they will typically be clipped to *max_force* by the vehicle model. Note also that many of the calls to *length* and *normalize* functions in these formulations can be replaced by fast routines that use an approximation to length as in [Ohashi 94]. The terms "we" or "our" will sometimes be used to indicate the first person perspective of the character being steered by a given behavior. Animated diagrams illustrating these behaviors can be found on the web at **http://www.red.com/cwr/steer/**

Seek (or pursuit of a static target) acts to steer the character towards a specified position in global space. This behavior adjusts the character so that its velocity is radially aligned towards the target. Note that this is different from an attractive force (such as gravity) which would produce an orbital path *around* the target point. The "desired velocity" is a vector in the direction from the character to the target. The length of "desired velocity" could be *max_speed*, or it could be the character's current speed, depending on the particular application. The steering vector is the difference between this desired velocity and the character's current velocity, see Figure 3.

```
desired_velocity = normalize (position -
target) * max_speed
steering = desired_velocity - velocity
```

If a character continues to **seek**, it will eventually pass through the target, and then turn back to approach again. This produces motion a bit like a moth buzzing around a light bulb. Contrast this with the description of **arrival** below.

Flee is simply the inverse of **seek** and acts to steer the character so that its velocity is radially aligned away from the target. The desired velocity points in the opposite direction.

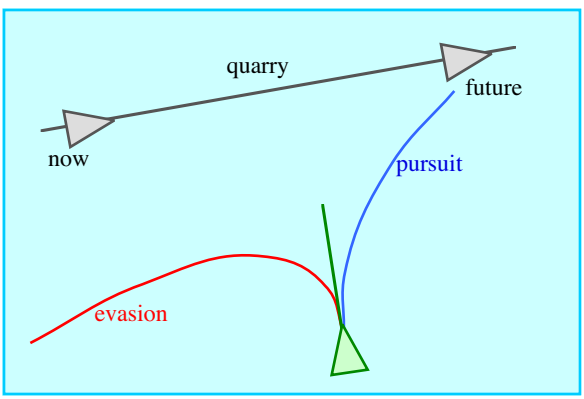

Figure 4: **pursuit** and **evasion**

Pursuit is similar to **seek** except that the quarry (target) is another moving character. Effective pursuit requires a prediction of the target's future position. The approach taken here is to use a simple predictor and to reevaluate it each simulation step. For example, a linear velocity-based predictor corresponds to the assumption that the quarry will not turn during the prediction interval. While this assumption is often incorrect, the resulting prediction will only be in use for about 1/30 of a second. The position of a character T units of time in the future (assuming it does not maneuver) can be obtained by scaling its velocity by T and adding that offset to its current position. Steering for **pursuit** is then simply the result of applying the **seek** steering behavior to the predicted target location. See Figure 4.

The key to this implementation of **pursuit** is the method used to estimate the prediction interval T. Ideally, T would be the time until interception, but that value is unknowable because the quarry can make arbitrary and unpredictable maneuvers. T could be assumed to be a constant, which while naive, would produce better pursuit than simple **seek** (which corresponds to T=0). However for reasonable performance T should be larger when the pursuer is far from the quarry, and small when they are nearby. A simple estimator of

moderate quality is T=Dc where D is the distance between pursuer and quarry, and c is a turning parameter. A more sophisticated estimator can be obtained by taking into account the relative headings of pursuer and quarry, and whether the pursuer is generally ahead of, behind, or to the side of, the quarry. These two metrics can be expressed in terms of simple dot products (between unit *forward* vectors, and between the quarry's *forward* and the offset to the pursuer's position). Note that care must be taken to reduce T (e.g to zero) when the pursuer finds itself aligned with, and in front of, its quarry.

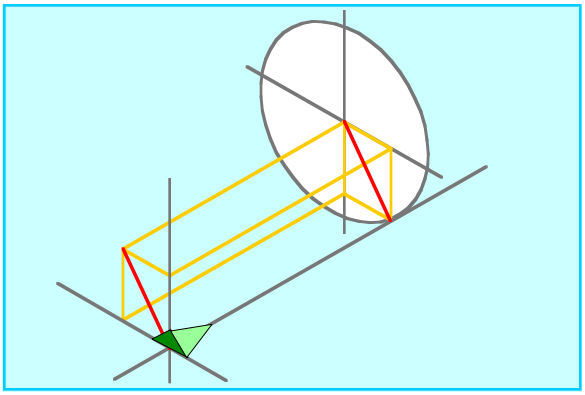

Figure 5: **offset pursuit**

Another approach to both **seek** and **pursuit** is based on the fact that when our character is on a collision course with a target, it will appear at a constant heading in our character's local space. Conversely our character can steer toward interception by contriving to keep the target at a constant heading.

Evasion is analogous to **pursuit**, except that **flee** is used to steer away from the predicted future position of the target character. Optimal techniques for pursuit and evasion exist in the field of control theory [Isaacs 65]. The versions given here are intended to be lightweight and are nonoptimal. In natural systems, evasion is often "intentionally" nonoptimal in order to be unpredictable, allowing it to foil predictive pursuit strategies, see [Cliff 96].

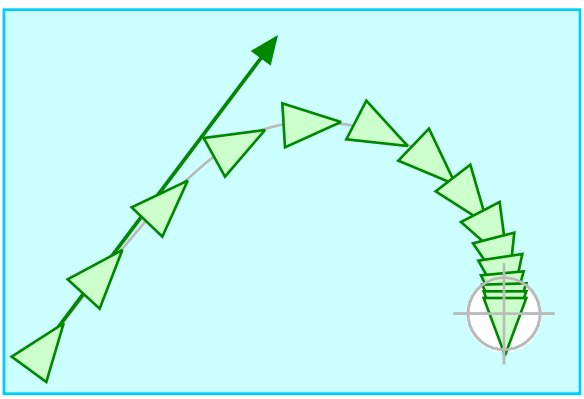

Figure 6: **arrival**

Offset pursuit refers to steering a path which passes near, but not directly into a moving target. Examples would be a spacecraft doing a "fly-by" or an aircraft doing a "strafing run": flying near enough to be within sensor or weapon range without colliding with the target. The basic idea is to dynamically compute a target point which is offset by a given radius R from the predicted future position of the quarry, and to then use **seek** behavior to approach that offset point, see Figure 5. To construct the offset point: localize the predicted target location (into our character's local coordinate space) project the local target onto the character's *side-up* plane, normalize that lateral offset, scale it by -R, add it to the local target point, and globalize that value.

Arrival behavior is identical to **seek** while the character is far from its target. But instead of moving through the target at full speed, this behavior causes the character to slow down as it approaches the target, eventually slowing to a stop coincident with the target, as shown in Figure 6. The distance at which slowing begins is a parameter of the behavior. This implementation is similar to **seek**: a desired velocity is determined pointing from the character towards the target. Outside the stopping radius this desired velocity is clipped to *max_speed*, inside the stopping radius, desired velocity is ramped down (e.g. linearly) to zero.

```
target_offset = target - position
distance = length (target_offset)
ramped_speed = max_speed * (distance / slowing_distance)
clipped_speed = minimum (ramped_speed, max_speed)
desired_velocity = (clipped_speed / distance) * target_offset
steering = desired_velocity - velocity
```

Real world examples of this behavior include a baseball player running to, and then stopping at a base; or an automobile driving towards an intersection and coming to a stop at a traffic light.

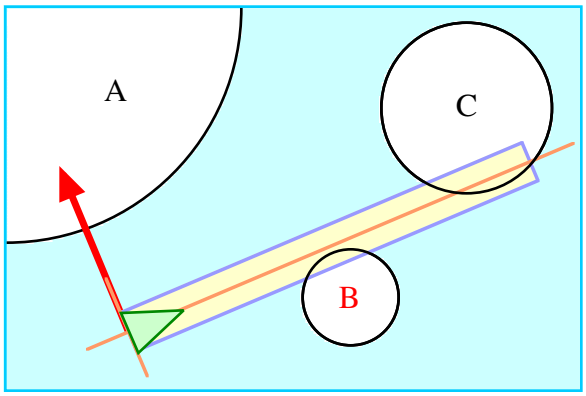

Figure 7: **obstacle avoidance**

Obstacle avoidance behavior gives a character the ability to maneuver in a cluttered environment by dodging around obstacles. There is an important distinction between **obstacle avoidance** and **flee** behavior. **Flee** will always cause a character to steer away from a given location, whereas **obstacle avoidance** takes action only when a nearby obstacle lies directly in front of the character. For example, if a car was driving parallel to a wall, **obstacle avoidance** would take no corrective steering action, but **flee** would attempt to turn away from the wall, eventually driving perpendicular to it.

The implementation of **obstacle avoidance** behavior described here will make a simplifying assumption that both the character and obstacle can be reasonably approximated as spheres, although the basic concept can be easily extend to more precise shape models. Keep in mind that this relates to obstacle avoidance not necessarily to *collision detection*. Imagine an airplane trying to avoid a mountain. Neither are spherical in shape, but it would suffice that the plane's bounding sphere avoids the mountain's bounding sphere. A decomposable hierarchy of bounding spheres can be used for efficient representation of shapes for collision detection [Hubbard 96], and presumably for obstacle avoidance too. An unrelated obstacle avoidance technique is described in [Egbert 96].

The geometrical construction of **obstacle avoidance** behavior bares some similarity to the **offset pursuit** behavior described above. It is convenient to consider the geometrical situation from the character's local coordinate system. The goal of the behavior is to keep an imaginary cylinder of free space in front of the character. The cylinder lies along the character's *forward* axis, has a diameter equal to the character's bounding sphere, and extends from the character's center for a distance based on the character's speed and agility. An obstacle further than this distance away is not an immediate threat. The **obstacle avoidance** behavior considers each obstacle in turn (perhaps using a spatial portioning scheme to cull out distance obstacles) and determines if they intersect with the cylinder. By localizing the center of each spherical obstacle, the test for non-intersection with the cylinder is very fast. The local obstacle center is projected onto the *side-up* plane (by setting its *forward* coordinate to zero) if the 2D distance from that point to the local origin is greater than the sum of the radii of the obstacle and the character, then there is no potential collision. Similarly obstacles which are fully behind the character, or fully ahead of the cylinder, can be quickly rejected. For any

remaining obstacles a line-sphere intersection calculation is performed. The obstacle which intersects the *forward* axis nearest the character is selected as the "most threatening." Steering to avoid this obstacle is computed by negating the (lateral) *side-up* projection of the obstacle's center. In Figure 7 obstacle A does not intersect the cylinder, obstacles B and C do, B is selected for avoidance, and corrective steering is to the character's left. The value returned from obstacle avoidance is either (a) the steering value to avoid the most threatening obstacle, or (b) if no collision is imminent, a special value (a *null* value, or the zero vector) to indicate that no corrective steering is required at this moment.

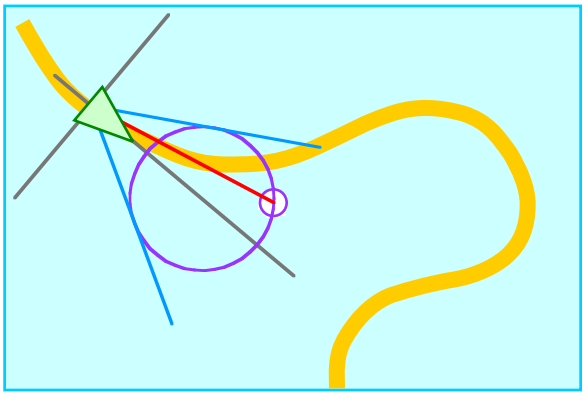

Figure 8: **wander**

A final note regarding interaction of obstacle avoidance and goal seeking. Generally we only care about obstacles which are between us and our goal. The mountain beyond the airport is ignored by the airplane, but the mountain between the plane and the airport is *very* important.

Wander is a type of random steering. One easy implementation would be to generate a random steering force each frame, but this produces rather uninteresting motion. It is "twitchy" and produces no sustained turns. A more interesting approach is to retain steering direction state and make small random displacements to it each frame. Thus at one frame the character may be turning up and to the right, and on the next frame will still be turning in almost the same direction. The steering force takes a "random walk" from one direction to another. This idea can be implemented several ways, but one that has produced good results is to constrain the steering force to the surface of a sphere located slightly ahead of the character. To produce the steering force for the next frame: a random displacement is added to the previous value, and the sum is constrained again to the sphere's surface. The sphere's radius (the large circle in Figure 8) determines the maximum wandering "strength" and the magnitude of the random displacement (the small circle in Figure 8) determines the wander "rate." Another way to implement **wander** would be to use coherent *Perlin noise* [Perlin 85] to generate the steering direction.

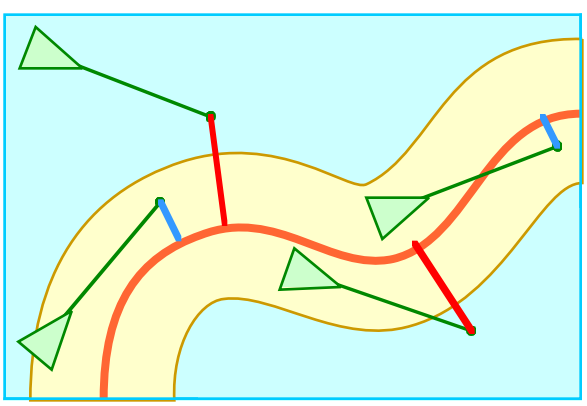

Figure 9: **path following**

Related to wander is **explore** (where the goal is to exhaustively cover a region of space) and **forage** (combining wandering with resource seeking). See [Beer 90] and [Tu 96] for more details.

Path following behavior enables a character to steer along a predetermined path, such as a roadway, corridor or tunnel. This is distinct from constraining a vehicle rigidly to a path like a

train rolling along a track. Rather **path following** behavior is intended to produce motion such as people moving down a corridor: the individual paths remain near, and often parallel to, the centerline of the corridor, but are free to deviate from it. In the implementation described here, a path will be idealized as a *spine* and a *radius*. The spine might be represented as a spline curve or a "poly-line" (a series of connected line segments). The path is then a "tube" or "generalized cylinder:" a circle of the specified radius, swept along the specified spine. The goal of the **path following** steering behavior is to move a character along the path while staying within the specified radius of the spine. If the character is initially far away from the path, it must first approach, then follow the path.

To compute steering for **path following**, a velocity-based prediction is made of the character's future position, as discussed above in regard to **obstacle avoidance** behavior. The predicted future position is projected onto the nearest point on the path spine. See Figure 9. If this projection distance (from the predicted position to the nearest on-path point) is less than the path radius, then the character is deemed to be correctly following the path and no corrective steering is required. Otherwise the character is veering away from the path, or is too far away from the path. To steer back towards the path, the **seek** behavior is used to steer towards the on-path projection of the predicted future position. Like in **obstacle avoidance**, a null or zero value is returned is returned if no corrective steering is required. A path can be followed without regard to direction, or in a specified direction (from A to B or from B to A) by adjusting the target point along the path in the desired direction.

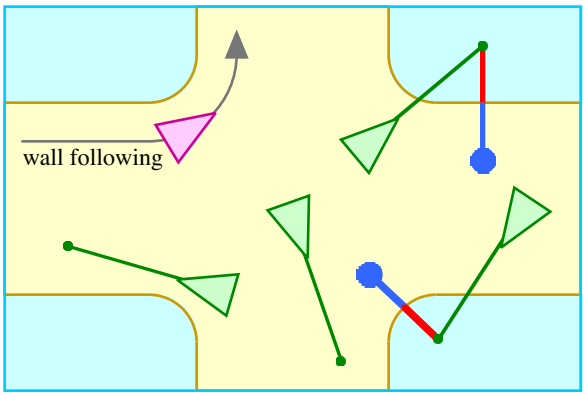

Figure 10: **wall following**, **containment**

Variations on **path following** include **wall following** and **containment** as shown in Figure 10. **Wall following** means to approach a "wall" (or other surface or path) and then to maintain a certain offset from it [Beer 90]. For a discussion of offset goals, see **offset pursuit** above.

Containment refers to motion which is restricted to remain within a certain region. **Path following** is a type of **containment** where the allowable region is a cylinder around the path's spine. Examples of **containment** include: fish swimming in an aquarium and hockey players skating within an ice rink. To implement: first predict our character's future position, if it is inside the allowed region no corrective steering is necessary. Otherwise we steer towards the allowed region. This can be accomplished by using **seek** with an inside point (for example, we can project the future position to the obstacle surface, and then extend this offset to obtain a

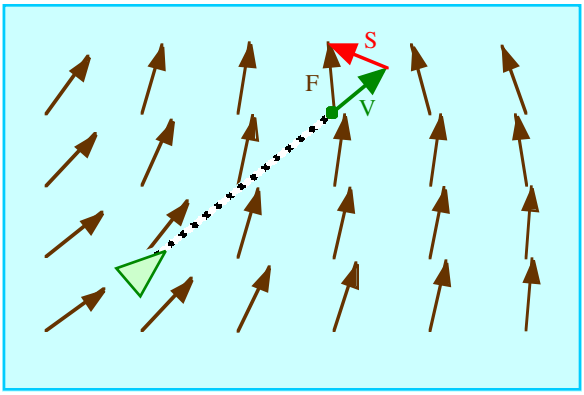

Figure 11: **flow following**

target point) or we can determine the intersection of our path with the boundary, find the surface normal at that point, and then use the component of the surface normal which is perpendicular to our forward direction as the corrective lateral steering.

Flow field following steering behavior provides a useful tool for directing the motion of characters based on their position within an environment. It is particularly valuable in some production teams because it allows motion specification to be made without use of programming and so can used by the art staff directly. In the case of game production this person might be a "level designer" and in animation production they might be a "scene planner" or "layout artist."

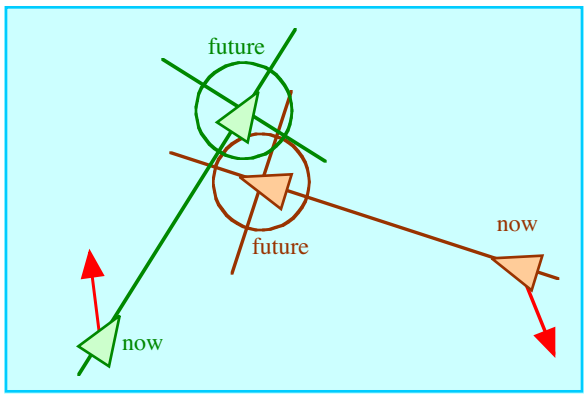

Figure 12:
unaligned collision avoidance

In **flow field following** behavior the character steers to align its motion with the local tangent of a *flow field* (also known as a *force field* or a *vector field*). The flow field defines a mapping from a location in space to a flow vector: imagine for example a floor with arrows painted on it. Such a map, typically representing the floor plan of an environment, can be easily created by an artist with a special purpose "paint" program which allows them to draw the desired traffic flow with a paint brush. The implementation of **flow field following** is very simple. The future position of a character is estimated and the flow field is sampled at that location. This flow direction (vector F in Figure 11) is the "desired velocity" and the steering direction (vector S) is simply the difference between the current velocity (vector V) and the desired velocity.

Unaligned collision avoidance behavior tries to keep characters which are moving in arbitrary directions from running into each other. Consider your own experience of walking across a plaza or lobby full of other walking people: avoiding collisions involves predicting potential collisions and altering your direction and speed to prevent them. If all nearby characters are *aligned*, a less complicated strategy can be used, see **separation** below.

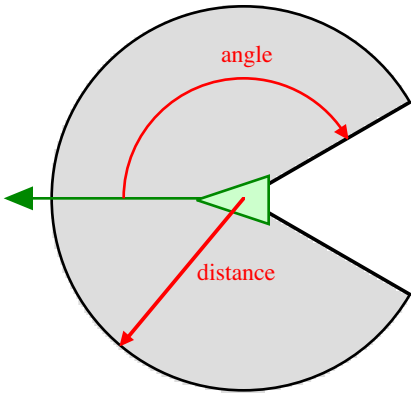

Figure 13: neighborhood

To implement this as a steering behavior, our character considers each of the other characters and determines (based on current velocities) when and where the two will make their *nearest approach*. A potential for collision exists if the nearest approach is in the future, and if the distance between the characters at nearest approach is small enough (indicated by circles in Figure 12). The nearest of these potential collisions, if any, is determined. The character then steers to avoid the site of the predicted collision. It will steer laterally to turn away from the

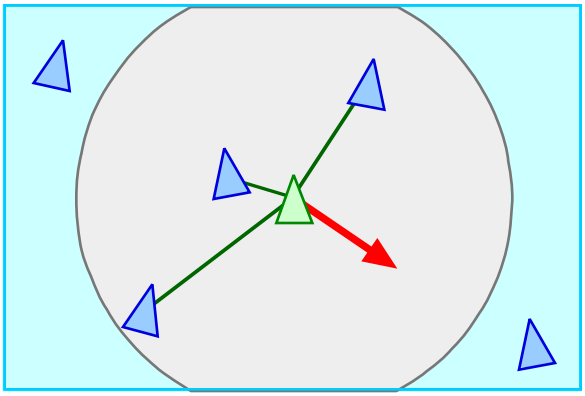

Figure 14: **separation**

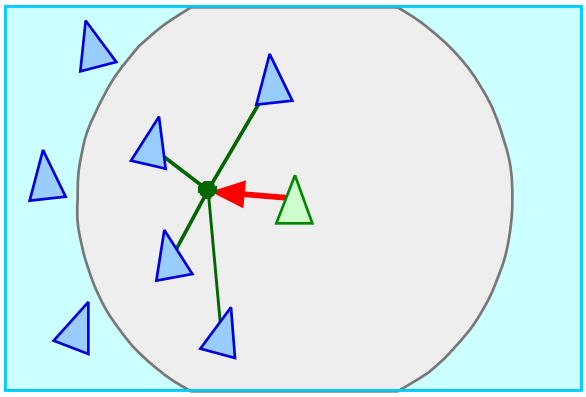

Figure 15: **cohesion**

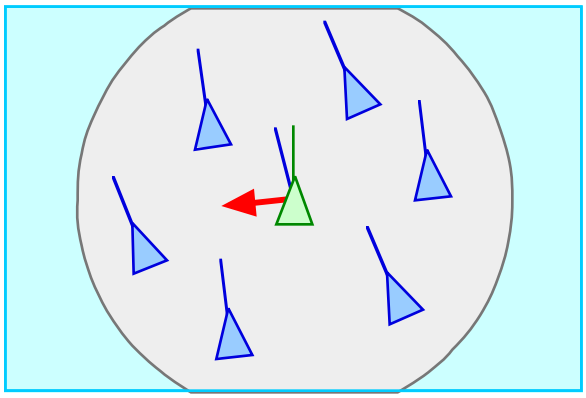

Figure 16: **alignment**

potential collision. It will also accelerate forward or decelerate backwards to get to the indicate site before or after the predicted collision. In Figure 12 the character approaching from the right decides to slow down and turn to the left, while the other character will speed up and turn to the left.

The next three steering behaviors: **separation**, **cohesion**, and **alignment**, relate to groups of characters. In each case, the steering behavior determines how a character reacts to other characters in its local *neighborhood*. Characters outside of the local neighborhood are ignored. As shown in Figure 13, the neighborhood is specified by a *distance* which defines when two characters are "nearby", and an *angle* which defines the character's perceptual "field of view."

Separation steering behavior gives a character the ability to maintain a certain separation distance from others nearby. This can be used to prevent characters from crowding together. To compute steering for **separation**, first a search is made to find other characters within the specified neighborhood. This might be an exhaustive search of all characters in the simulated world, or might use some sort of spatial partitioning or caching scheme to limit the search to local characters. For each nearby character, a repulsive force is computed by subtracting the positions of our character and the nearby character, normalizing, and then applying a 1/r weighting. (That is, the position offset vector is scaled by $1/r^2$.) Note that 1/r is just a setting that has worked well, not a fundamental value. These repulsive forces for each nearby character are summed together to produce the overall steering force. See Figure 14.

Cohesion steering behavior gives an character the ability to cohere with (approach and form a group with) other nearby characters. See Figure 15. Steering for **cohesion** can be computed by finding all characters in the local neighborhood (as described above for **separation**), computing the "average position" (or "center of gravity") of the nearby characters. The steering force can applied in the direction of that "average position" (subtracting our character position from the average position, as in the original boids model), or it can be used as the target for **seek** steering behavior.

Alignment steering behavior gives an character the ability to align itself with (that is, head in the same direction and/or speed as) other nearby characters, as shown in Figure 16. Steering for **alignment** can be computed by finding all characters in the local neighborhood (as described above for **separation**), averaging together the velocity (or alternately, the unit *forward* vector) of the nearby characters. This average is the "desired velocity," and so the steering vector is the difference between the average and our character's current velocity (or alternately, its unit *forward* vector). This steering will tend to turn our character so it is aligned with its neighbors.

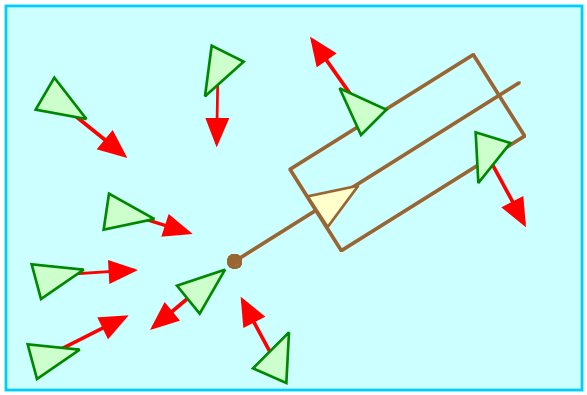

Figure 17: **leader following**

Flocking behavior: in addition to other applications, the **separation**, **cohesion** and **alignment** behaviors can be combined to produce the *boids* model of flocks, herds and schools [Reynolds 87] (see also [Tu 94], [Tu 96] and [Hodgins 94]). In some applications it is sufficient to simply sum up the three steering force vectors to produce a single combined steering for flocking (see Combining Behaviors below). However for better control it is helpful to first normalize the three steering components, and then to scale them by three weighting factors before summing them. As a result, boid flocking behavior is specified by nine numerical parameters: a weight (for combining), a distance and an angle (to define the neighborhood, see Figure 13) for each of the three component behaviors.

Leader following behavior causes one or more character to follow another moving character designated as the *leader*. Generally the followers want to stay near the leader, without crowding the leader, and taking care to stay out of the leader's way (in case they happen to find them selves in front of the leader). In addition, if there is more than one follower, they want to avoid bumping each other. The implementation of **leader following** relies on **arrival** behavior (see above) a desire to move towards a point, slowing as it draws near. The arrival target is a point offset slightly behind the leader. (The offset distance might optionally increases with speed.) If a follower finds itself in a rectangular region in front of the leader, it will steer laterally away from the leader's path before resuming **arrival** behavior. In addition the followers use **separation** behavior to prevent crowding each other. See Figure 17.

Finally, here are quick sketches of some other steering behaviors that fit into the same general category as those described in more detail above. **Interpose** steering behavior attempts to put its character between two other moving characters, for example a soccer player trying to block a pass between two members of the opposing team. The general approach is similar to **pursuit** described above: predict the future position of the two other characters, determine a target point by interpolating between the future positions, and use **seek** to steer toward the target point. Related to **leader following** and **pursuit**, we could **shadow** our quarry by approaching and then using **alignment** to match their speed and heading. The **arrival** behavior described above can be considered a constraint on position and speed. This can be extended to simultaneously constrain orientation (to produce **docking**), or a non-zero velocity, and/or to meet these constraints at a given time. **Hide** behavior involves identifying a target

location which is on the opposite side of an obstacle from the opponent, and steering toward it using **seek**.

Combining Behaviors

The individual steering behaviors described above serve as building blocks for more complex patterns of behavior. They are components of a larger structure, like notes of a melody or words of a story. In order to make interesting and life-like behaviors we need to select among, and blend between, these individual components. Unless an autonomous character exists in an very simple world, it would seldom make sense for the character to continually execute a single steering behavior.

Combining behaviors can happen in two ways. A character may sequentially switch between behavioral modes as circumstances change in its world. For example, imagine caribou grazing in a meadow when suddenly they sense wolves approaching. This event triggers a discrete behavioral switch. All thoughts of grazing are forgotten as the caribou herd turns to flee from the predators. There is no tendency to mix these behaviors: a caribou will not slow down while running from a wolf in order to grab another bite of food. These discrete changes of behavioral state take place at the *action selection* level, the top of the three level behavioral hierarchy discussed in the Introduction. There is a extensive discussion of action selection in [Tu 94], [Tu 96] and in [Blumberg 94].

On the other hand, some kinds of behaviors are commonly blended together, effectively acting in parallel. For example, as the caribou flee through the forest, they blend **evasion** and **obstacle avoidance** together to allow them to escape from the wolves while dodging trees. A caribou cannot afford to ignore either component behavior, it must always be moving in a direction that *both* takes it away from the wolf *and* avoids collisions with trees. This behavioral blending occurs at the middle *steering* level of the behavioral hierarchy.

Blending of steering behaviors can be accomplished in several ways. The most straightforward is simply to compute each of the component steering behaviors and sum them together, possibly with a weighting factor for each of them. (Note that steering vectors are especially easy to blend, other kinds of behaviors, producing other kinds of values (e.g. conversational behaviors) could be much harder to combine.) This simple linear combination often works well, but has at least two shortcomings: it is not the most computationally efficient approach, and despite adjusting the weights, component behaviors may cancel each other out at inopportune times.

The computation load can be decreased by observing that a character's momentum serves to apply a low-pass filter to changes in steering force. So rather than compute several steering components each simulation step and average them together, we could instead select one steering component to compute and apply each frame, and depend on momentum (and perhaps some explicit damping of acceleration) to blend them together.

The problem of components canceling each other out can be addressed by assigning a priority to components. (For example: first priority is **obstacle avoidance**, second is **evasion** ...) The steering controller first checks to see if **obstacle avoidance** returns a non-zero value (indicating a potential collision), if so it uses that. Otherwise, it moves on to the second priority behavior, and so on.

A hybrid of these techniques that the author has found useful is “prioritized dithering”: with a certain probability the first priority behavior is evaluated, and if it returns a non-zero (non-null) value that will be used. Otherwise (if the behavior returns zero, or it was skipped over due to the random selection) the second priority behavior is considered, and so on.

In [Reynolds 87] a blending scheme called “prioritized acceleration allocation” was used with the boids flocking model. The basic idea was that by adjusting their magnitude, higher priority behaviors could decide whether or not to leave any steering force for use by lower priority behaviors. In the course of several reimplementations of boids over the years, a simple linear combination of the component behaviors has proved sufficient. When combining flocking with other behaviors such as obstacle avoidance, both simple summing and prioritized dither have been used successfully.

Conclusions

This paper defined “autonomous character” in terms of autonomous agents and improvisational action. It presented a decomposition of the task of constructing motion behaviors for autonomous characters into a three level hierarchy of: action selection, steering, and locomotion. It has defined a minimal implementation of the locomotion level in terms of a “simple vehicle model.” It has then presented a collection of simple, common steering behaviors. (Including: **seek**, **flee**, **pursuit**, **evasion**, **offset pursuit**, **arrival**, **obstacle avoidance**, **wander**, **path following**, **wall following**, **containment**, **flow field following, unaligned collision avoidance**, **separation**, **cohesion**, **alignment**, **flocking**, and **leader following**.) Finally it has described some techniques for blending these simple steering behaviors together.

Acknowledgments

The techniques described in this paper have been developed over the last twelve years, for many different projects, at several companies. I wish to acknowledge the helpful cooperation of all the companies and coworkers involved. Specifically I wish to thank the following people for managerial support and technical collaboration. At Sony Computer Entertainment America: Phil Harrison, John Phua, Attila Vass, Gabor Nagy, Sky Chang, and Tom Harper. At DreamWorks Feature Animation: Dylan Kohler, Bart Gawboy, Matt Arrott, Lance Williams, Saty Raghavachary, and Mike Ullner. At SGI’s Silicon Studio: Bob Brown, Leo Blume, Roy Hashimoto, and especially my behavioral animation colleague Xiaoyuan Tu. At Electronic Arts: Luc Barthelet, Steve Crane, Kelly Pope, Steve Sims, and Frank Giraffe. At Symbolics Graphics Division: Tom McMahon, Andy Kopra, Larry Malone, and Michael Wahrman. Finally, loving thanks to my wife Lisa and our children Eric and Dana. Before they become fully autonomous, I hope I steer the kids in the right direction. Certainly they are already real *characters*!

References

Arkin, Ronald (1987) “Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior”, Proceedings of IEEE Conference on Robotics and Automation, pages 264-271.

Arkin, Ronald (1989) “Motor Schema-Based Mobile Robot Navigation”, International Journal of Robotics Research, 8(4) pages 92-112.

Arkin, Ronald (1992) “Behavior-based Robot Navigation in Extended Domains”, Journal of Adaptive Behavior, 1(2) pages 201-225.

Bates, Joseph; Loyall, Bryan; Reilly, Scott (1992) “An Architecture for Action, Emotion, and Social Behavior”, Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World, S.Martino al Camino, Italy. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-92-144.ps

Beer, R. D. (1990). Intelligence as Adaptive Behavior: Experiments in Computational Neuroethology. New York: Academic Press. See also: http://yuggoth.ces.cwru.edu/johng/robopaper/robopaper.html

Blumberg, Bruce and Galyean, Tinsley (1995) Multi-Level Direction of Autonomous Creature for Real-Time Virtual Environments, Proceedings of SIGGRAPH 95, in Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 47-54. http://bruce.www.media.mit.edu/people/bruce/Siggraph95.final.ps

Blumberg, Bruce (1994) “Action-Selection in Hamsterdam: Lessons from Ethology” in the Proceedings of the Third International Conference on the Simulation of Adaptive Behavior (SAB94), D. Cliff, P. Husbands, J-A Meyer, and S. Wilson, Editors, MIT Press, Cambridge, Massachusetts, pages 108-117

Braitenberg, Valentino (1984) Vehicles: Experiments in Synthetic Psychology, The MIT Press, Cambridge, MA.

Brooks, Rodney A. (1985) "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation 2(1), March 1986, pp. 14--23; also MIT AI Memo 864, September 1985. See: http://www.ai.mit.edu/people/brooks/papers/AIM-864.pdf

Bruderlin, Armin and Calvert, Tom (1989) "Goal-Directed, Dynamic Animation of Human Walking", ACM SIGGRAPH'89, Proceedings, vol. 23, pp 233-242.

Cliff, Dave and Miller, Geoffrey (1996) “Co-Evolution of Pursuit and Evasion II: Simulation Methods and Results”, From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96), Maes, Mataric, Meyer, Pollack, and Wilson editors, ISBN 0-262-63178-4, MIT Press. http://www.cogs.susx.ac.uk/users/davec/pe.html

Costa, Mônica; Feijó, Bruno; and Schwabe, Daniel (1990) Reactive Agents in Behavioral Animation, *Anais do SIBGRAPI 95*, Lotufo and Mascarenhas editors, São Carlos, SP, pages 159-165. http://www.icad.puc-rio.br/~monica/pubs.htm

Cremer, James; Kearney, J.; Willemsen, P. (1996) “A Directable Vehicle Behavior Model for Virtual Driving Environments”, Proceedings of 1996 Conference on AI, Simulation, and Planning in High Autonomy Systems, La Jolla, CA. http://www.cs.uiowa.edu/~cremer/papers/aisp-final.ps

Egbert, Parris and Winkler, Scott (1996) “Collision-Free Object Movement Using Vector Fields”, IEEE Computer Graphics and Applications, 16(4), pages 18-24. http://www.computer.org/cga/cg1996/g4toc.htm

Johnson, Michael (1994) WavesWorld: A Testbed for Three Dimensional Semi-Autonomous Animated Characters, PhD Thesis, MIT Media Lab. http://wave.www.media.mit.edu/people/wave/PhDThesis/outline.html

Hayes-Roth, Barbara. and van Gent, R. (1996) “Improvisational Puppets, Actors, and Avatars”, Proceedings of the 1996 Computer Game Developers’ Conference. Stanford Knowledge Systems Laboratory Report KSL-96-09. file://www-ksl.stanford.edu/pub/KSL_Reports/KSL-96-09.ps

Hodgins, Jessica and Brogan, David (1994) “Robot Herds: Group Behaviors for Systems with Significant Dynamics”, Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV), Brooks and Maes editors, MIT Press, Cambridge, MA, pages 319-324. http://www.cc.gatech.edu/gvu/animation/papers/alife.ps.gz

Hodgins, Jessica; Wooten, W; Brogan, D; and O’Brien, J (1995) “Animating Human Athletics”, Proceedings of SIGGRAPH 95, in Computer Graphics Proceedings, Annual Conference Series, Robert Cook editor, ACM SIGGRAPH, pages 71-78. http://www.cc.gatech.edu/gvu/animation/papers/sig95.ps.gz

Hubbard, Philip (1996) “Approximating Polyhedra with Spheres for Time-Critical Collision Detection”, ACM Transactions on Graphics, 15(03), pages 179-210. http://www.acm.org/pubs/tog/hubbard96/

Isaacs, Rufus (1965) Differential Games: A Mathematical Theory with Application to Warfare and Pursuit, Control and Optimization, John Wiley and Sons, New York.

Kahn, Kenneth (1979) “Creation of Computer Animation from Story Descriptions”, Technical Report 540, MIT AI Lab, Cambridge, Mass.

Maes, Pattie; Darrell, T.; Blumberg, B. (1995) “The Alive System: Full Body Interaction with Autonomous Agents”, Computer Animation ’95 Conference, IEEE Press, pages 11-18.

Mataric, Maja (1993) “Designing and Understanding Adaptive Group Behavior”, Adaptive Behavior 4(1), pages 51-80. http://www-robotics.usc.edu/~maja/ http://www-robotics.usc.edu/~maja/publications/abj95.ps.gz

Ohashi, Yoshikazu (1994) “Fast Linear Approximations of Euclidean Distance in Higher Dimensions”, in Graphics Gems IV, Paul Heckbert editor, Academic Press. See ftp://princeton.edu/pub/Graphics/GraphicsGems/GemsIV/

Perlin, Ken (1985) “An Image Synthesizer”, in SIGGRAPH '85 Proceedings, Computer Graphics 19(3), pages 287-296. See http://www.mrl.nyu.edu/perlin/doc/oscar.html

Perlin, Ken (1995) “Real Time Responsive Animation With Personality”, IEEE Transactions on Visualization

and Computer Graphics, 1(1) pages 5-15, ISSN 1077-2626.

Perlin, Ken and Goldberg, Athomas (1996) “Improv: A System for Scripting Interactive Actors in Virtual Worlds”, Proceedings of SIGGRAPH 96, in Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 205-216. http://www.mrl.nyu.edu/improv/ http://www.mrl.nyu.edu/improv/sig-paper96/

Pottinger, Dave (1999) “Coordinated Unit Movement” and “Implementing Coordinated Movement”, Game Developer Magazine, January, 1999. See:
http://www.gamasutra.com/features/game_design/19990122/movement_01.htm
http://www.gamasutra.com/features/game_design/19990129/implementing_01.htm

Raibert, Marc and Hodgins, Jessica (1991) “Animation of Dynamic Legged Locomotion”, Computer Graphics 25(4), Proceeding of SIGGRAPH 91, Thomas Sederberg editor, ISSN 0097-8930, pages 349-358.

Raibert, M., Hodgins, J., Ringrose, R., Playter, R., Borvansky, L., Campbell, L., Evans, D., Crane, C., Lamb, M. (1991) “On The Run”, SIGGRAPH `91 Electronic Theater, SIGGRAPH Video Review, Issue 72, http://www.ai.mit.edu/projects/leglab/simulations/otr/otr.html

Reese, Bjørn and Bryan, Stout (1999) “Finding a Pathfinder” to appear: AAAI 99 Spring Symposium on Artificial Intelligence and Computer Games. See also: http://www.red.com/breese/navigation.html

Resnick, Mitchel (1989) “LEGO, Logo, and Life”, Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (ALife ’87), SFI Studies in the Sciences of Complexity, Volume 6, Christopher Langton editor, Addison-Wesley, Redwood City, CA, USA, pages 397-406.

Resnick, Mitchel (1993) Behavior Construction Kits, CACM, 36(7), http://el.www.media.mit.edu:80/groups/el/Papers/mres/BCK/BCK.html

Resnick, Mitchel *et al.* (1998) “The Virtual Fishtank” http://el.www.media.mit.edu/groups/el/projects/fishtank/

Reynolds, C. W. (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

Reynolds, C. W. (1988) Not Bumping Into Things, in the notes for the SIGGRAPH 88 course Developments in Physically-Based Modeling, pages G1-G13, published by ACM SIGGRAPH.
http://www.red.com/cwr/nobump/nobump.html

Ridsdale, Gary (1987) “The Director’s Apprentice: Animating Figures in a Constrained Environment”, Ph.D. thesis, School of Computing Science, Simon Fraser University.

Sims, Karl (1994) “Evolving Virtual Creatures”, Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, Andrew Glassner editor, ACM SIGGRAPH, ISBN 0-89791-667-0, pages 15-22. ftp://think.com/users/karl/Welcome.html ftp://think.com/users/karl/siggraph94.ps

Still, G Keith (1994) “Simulating Egress using Virtual Reality - a Perspective View of Simulation and Design”, IMAS Fire Safety on Ships.

Strassmann, Steve (1991) Desktop Theater: Automatic Generation of Expressive Animation, PhD thesis, MIT Media Lab http://www.method.com/straz/straz-phd.pdf

Thalmann, Daniel; Renault, Olivier and Magnenat-Thalmann Nadia (1990) “A Vision-Based Approach to Behavioral Animation”, Journal of Visualization and Computer Animation, John Wiley & Sons, 1(1) pages 18-21.

Travers, Michael (1989) “Animal Construction Kits”, Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (ALife ’87), SFI Studies in the Sciences of Complexity, Volume 6, Christopher Langton editor, Addison-Wesley, Redwood City, CA, USA, pages 421-442.
http://lcs.www.media.mit.edu/people/mt/agar/agar.html

Travers, Michael (1994) “LiveWorld: A Construction Kit for Animate Systems”, Proceedings of ACM CHI’94 Conference on Human Factors in Computing Systems, pages 37-38.
http://mt.www.media.mit.edu/people/mt/papers/chi94/chi94.html

Tu, Xiaoyuan and Terzopoulos, Demetri (1994) “Artificial Fishes: Physics, Locomotion, Perception, Behavior”, Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, Andrew Glassner editor, ACM SIGGRAPH, ISBN 0-89791-667-0, pages 43-50.
http://www.dgp.toronto.edu/people/tu/papers/sig94.ps

Tu, Xiaoyuan (1996) Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior, PhD dissertation, Department of Computer Science, University of Toronto.
http://www.dgp.toronto.edu/people/tu/thesis/thesis.html

van de Panne, M., Fiume, E., and Vranesic, Z. G., (1990) "Reusable Motion Synthesis Using State-Space Controllers", Proceedings of SIGGRAPH `90, In Computer Graphics Proceedings, ACM SIGGRAPH, pages 225-234. See http://www.dgp.toronto.edu/people/van/papers.html

Walter, W. Grey (1950) “An Imitation of Life”, Scientific American, 182(5), pages 42-45. (see also http://gate.uwe.ac.uk:8002/IAS/gwonline.html and http://www.newscientist.com/ns/980725/robotman.html)

Grey-Walter, W. (1953) The Living Brain. Gerald Duckworth and Co., Ltd.

Wiener, Norbert (1948) Cybernetics, or control and communication in the animal and the machine.

Cambridge, Massachusetts: The Technology Press; New York: John Wiley & Sons, Inc.

Wilhelms, Jane and Skinner, Robert (1990) A “Notion” for Interactive Behavioral Animation Control, IEEE Computer Graphics and Applications, 10(3), pages 14-22.

Zapata, R., Lepinay, P., Novales, C., and Deplanques, P. (1992) “Reactive Behaviors of Fast Mobile Robots in Unstructured Environments: Sensor-Based Control and Neural Networks” in From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92), Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 108-115

Zeltzer, David (1983) Knowledge-Based Animation, Proceedings SIGGRAPH/SIGART Workshop on Motion, pages 187-192.

Zeltzer, David (1990) “Task Level Graphical Simulation: Abstraction, Representation and Control,” in Making Them Move: Mechanics, Control and Animation of Articulated Figures, N. Badler, B. Barsky and D. Zeltzer, editors., Morgan Kaufmann Publishers.