# CMSC 425: Lecture 19
# Artificial Intelligence for Games: Multiple Agent Motion
Tuesday, Apr 16, 2013

**Reading:** Today's material is drawn from a variety of sources. Some of the material on particle systems is drawn from lecture notes by Alex Benton from the University of Cambridge and the classic paper "Particle Systems: A Technique for Modeling a Class of Fuzzy Objects," by W. T. Reeves, *ACM Transactions on Graphics*, 2, 1983, 91–108. The material on flocking is based in part on C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, 21, 25–34, 1987. The material on reciprocal velocity obstacles is taken from J. van den Berg, S. Guy, M. C. Lin, D. Manocha, "Reciprocal $n$-body Collision Avoidance," *Proc. Int. Symposium of Robotics Research* (ISRR), 2009.

**Recap:** In the previous lectures, we have discussed techniques for computing the motion of a single agent. Today we will discuss techniques for planning and coordinating the motion of multiple agents. In particular, we will discuss three aspects of this problem:

**Particle systems:** Modeling (unintelligent) motion for a collection of objects

**Flocking behavior:** Where all agents are moving as a unit

**Crowd behavior:** Where a number of agents, each with their own desired destination, are to move without colliding with each other.

**Particle Systems:** A *particle system* is a technique that uses a large number of small graphical artifacts, called *particles*, to create a large-scale, typically "fuzzy," visual effect. Examples of applications of particle systems include many amorphous natural phenomena such as fire, smoke, water, clouds, and explosions. In these examples particles are born and die dynamically, but there are also variants of particle systems where particles are persistent. These are used to produce static phenomena such as galaxies or "stringy" phenomena such as hair, grass, and other plants.

A fundamental principle that underlies particle systems is that our brains tend to cluster an aggregation of similar objects into the impression of a single impression. Thus, the many individual water drops that cascade down a flowing fountain are perceived as a flowing stream of water (see Fig. 1).

Fig. 1: A particle system simulating the flow of a liquid.

Note that particle systems do not really belong in this lecture on artificial intelligence, since their behavior is based solely on physical laws, and not on any model of intelligent behavior. Nonetheless,

we have decided to discuss them here because they do represent a common technique for generating interesting patterns of movement in games and other applications of interactive computer graphics.

Particle systems are almost as old as computer games themselves. The very earliest 2-dimensional games, such as *Spacewar!* and *Asteroids* simulated explosions through the use of a particle system. The power of the technique, along the term "particle system," came about in one of the special effects in the second Star Trek movie, where a particle system was used to simulate a fire spreading across the surface of a planet (the *genesis effect*).

**How Particle Systems Work:** One of the appeals of particle systems is that they are extremely simple to implement, and they are very flexible. Particles are created, live and move according to simple rules, and then die. The process that generates new particles is called an *emitter*. For each frame that is rendered, the following sequence of steps is performed:

**Emission:** New particles are generated and added to the system. There may be multiple sources of emission (and particles themselves can serve as emitters).

**Attributes:** Each new particle is assigned its (initial) individual properties that affect how the particle moves over time and is rendered. These may change over time and include the following:

**Geometric attributes:** initial position and velocity

**Graphics attributes:** shape, size, color, transparency

**Dynamic attributes:** lifetime, influence due to forces such as gravity and friction

**Death:** Any particles that have exceeded their prescribed lifetime are removed from the system.

**Movement:** Particles are moved and transformed according to their dynamic attributes, such as gravity, wind, and the density of nearby particles.

**Rendering:** An image of the surviving particles is rendered. Particles are typically rendered as a small blob.

In order to create a natural look, the process of emitting particles and the assignment of their attributes is handled in the probabilistic manner, where properties such as the location and velocity of particles being determined by a random number generator.

Particle system can be programmed to execute any set of instructions at each step, but usually the dynamic properties of particles are very simple, and do not react in any complex manner to the presence of other particles in the system. Because the approach is procedural, it can incorporate any computational model that describes the appearance or dynamics of the object. For example, the motions and transformations of particles could be tied to the solution of a system of partial differential equations. In this manner, particles can be used to simulate physical fluid phenomena such as water, smoke, and clouds.

**Updating Particles in Time:** There are two common ways to update particles over time.

**Closed-form function:** Every particle is represented by a *parametric function* of time (and coefficients that are given by the particle's initial motion attributes. For example, given the particle's initial position $p_0$, its initial velocity vector $v_0$, and some fixed field force $g$ (representing, for example, the affect of gravity) the position of the particle at time $t$ can be expressed in closed form as:

$$p(t) \;=\; p_0 + v_0 t + \frac{1}{2} g t^2.$$

(If you have ever taken a course in physics, this formula will be familiar to you. If not, don't worry how it was derived. It is a simple consequence of Newton's basic laws of motion.) On the positive side, this approach requires no storage of the evolving state of the particle, just its initial state and the elapsed time. On the negative side, this approach is very limited, and does not allow particles to respond to each other or their environment.

**Discrete integration:** In this type of system, each particle stores in current *physical state*. This state consists of the particle's *current position*, which is given by a point $p$, its *current velocity*, which is given by a vector $v$, and its *current acceleration*, which is given by a vector $a$. (Acceleration should be thought of as the accumulated effect of all the forces acting on the particle. For example, gravity tends to decrease the vertical component of the velocity and air resistance tends to decrease the particle's absolute velocity, without altering its direction.) We can then update the state over a small time interval, $\Delta t$ (for example, the elapsed time between two consecutive frames). By the basic laws of kinematics (which we may discuss later this semester):

$$v' = v + a \cdot \Delta t \qquad \text{and} \qquad p' = p + v \cdot \Delta t.$$

where $p'$ and $v'$ denote the particle's new position and velocity, respectively. (Depending on your implementation, you may also need to update the forces acting on the particle and update acceleration as well.)

**Flocking Behavior:** Next, let us consider the motion of a slightly smarter variety, namely *flocking*. We refer to flocking behavior in the generic sense as any motion arising when a group of agents adopt a decentralized motion strategy designed to hold the group together. Such behavior is exemplified by the motion of groups animals, such as birds, fish, insects, and other types of herding animals (see Fig. 2).

Fig. 2: An example of complex emergent behavior in flocking.

In contrast to full crowd simulation, where each agent may have its own agenda, in flocking it is assumed that the agents are *homogeneous*, that is, they are all applying essentially the same motion update algorithm. The only thing that distinguishes one agent from the next is their position relative to the other agents in the system. It is quite remarkable that the complex formations formed by flocking birds or schooling behavior in fish can arise in a system in which each creature is following

(presumably) a very simple algorithm. The apparently spontaneous generation of complex behavior from the simple actions of a large collection of dynamic entities is called *emergent behavior*. While the techniques that implement flocking behavior do not involve very sophisticated models of intelligence, variants of this method can be applied to simple forms of crowd motion in games, such as a crowd of people milling around in a large area or pedestrians strolling up and down a sidewalk.

**Boids:** One of the earliest models and perhaps the best-known model for flocking behavior was given by C. W. Reynolds from 1986 with his work on "boids." (The term is an intentional misspelling of "bird.") In this system, each agent (or *boid*) determines its motion based on a combination of four simple rules:

**Separation:** Each boid wishes to avoid collisions with other nearby boids. To achieve this, each boid generates a *repulsive potential field* whose radius of influence extends to its immediate neighborhood. Whenever another boid gets too close, the force from this field will tend to push them apart.

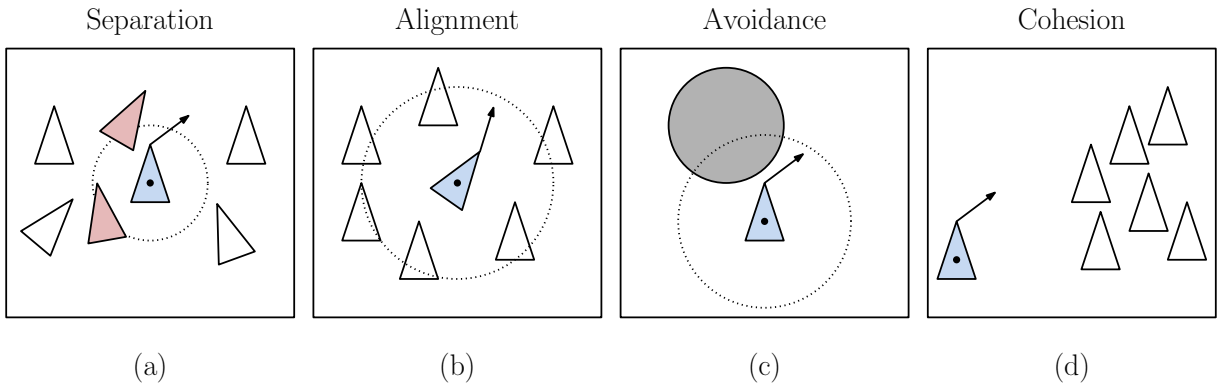


Fig. 3: Forces that control flocking behavior.

**Alignment:** Each boid's direction of flight is aligned with nearby boids. Thus, local clusters of boids will tend to point in the same direction and hence will tend to fly in the same direction.

**Avoidance:** Each boid will avoid colliding with fixed obstacles in the scene. At a simplest level, we might imagine that each fixed obstacle generates a repulsive potential field. As a boid approaches the object, this repulsive field will tend to cause the boid to deflect its flight path, thus avoiding a collision. Avoidance can also be applied to predators, which may attack the flock. (It has been theorized that the darting behavior of fish in a school away from a shark has evolved through natural selection, since the sudden chaotic motion of many fish can confuse the predator.)

**Cohesion:** Effects such as avoidance can cause the flock to break up into smaller subflocks. To simulate the flocks tendency to regroup, there is a force that tends to draw each boid towards the center of mass of the flock. (In accurate simulations of flocking motion, a boid cannot know exactly where the center of mass is. In general the center of attraction will be some point that the boid perceives being the center of the flock.)

**Boid Implementation:** Next let us consider how to implement such a system. We apply the same discrete integration approach as we did used for particle systems. In particular, we assume that each boid is

associated with a state vector $(p, v)$ consisting of its current position $p$ and current velocity $v$. (We assume that the boid is facing in the same direction as it is flying, but, if not, a vector describing the boid's angular orientation can also be added to the state.) We think of the above rules as imposing forces, which together act to set the boid's acceleration. Given this acceleration vector $a$ caused by the boid forces, we apply the same update rules, $v' = v + a \cdot \Delta t$ and $p' = p + v \cdot \Delta t$, to obtain the new position $p'$ and new velocity vector $v'$.

How are these forces computed? First observe that, for the sake of efficiency, you do not want the behavior of each boid to depend on the individual behaviors of the $n - 1$ boids, since this would be way to costly, taking $O(n^2)$ time for each iteration. Instead, the system maintains the boids stored in a *spatial data structure*, such as a grid or quadtree, which allows each boid to efficiently determine the boids that are near to it. Rules such as separation, avoidance, alignment can be based on a small number of nearest neighbor boids. Cohesion requires knowledge of the center of the flock, but this can be computed once at the start of each cycle in $O(n)$ time.

Since these are not really natural forces, but rather heuristics that are used to guiding motion, it is not essential to determine what the laws of kinetics would do. Rather, each rule naturally induces a directional influence. (For example, the force of avoidance is directed away from the anticipated point of impact while the force for alignment is in the average direction that nearby boids are facing.) Also, each rule can be associated with a strength whose magnitude depends, either directly or inversely, on the distance from the point of interest. (For example, avoidance forces are very strong near the obstacle but drop off rapidly. In contrast, the cohesive force tends to increase slowly as the distance from the center of flock increases.) Thus, given a unit vector $u$ pointing in the induced direction and the strength $s$ given as a scalar, we can compute the updated acceleration vector as $a = c \cdot s \cdot u$, where $c$ is a "fudge factor" that can be adjusted by the user that is used to model other unspecified physical quantities such as the boid's mass.

One issue that arises with this or any dynamical system is how to avoid undesirable (meaning unnatural looking) motion, such as collisions, oscillations, and unrealistically high accelerations. Here are two approaches:

**Prioritize and truncate:** Assume that there is a fixed maximum magnitude for the acceleration vector (based on how fast a boid can change its velocity based on what sort of animal is being modeled). Sort the rules in priority order. (For example, predator/obstacle avoidance is typically very high, flock cohesion is low.) The initial acceleration vector is the zero vector (meaning that the boid will simply maintain its current velocity). As each rule is evaluated, compute the associated acceleration vector and add it to the current acceleration vector. If the length of the acceleration vector ever exceeds the maximum allowed acceleration, then stop and return the current vector.

**Weight and clamp:** Assign numeric weights to the various rule-induced accelerations. (Again, avoidance is usually high and cohesion is usually low.) Take the weighted sum of these accelerations. If the length of the resulting acceleration vector exceeds the maximum allowed acceleration, then scale it down

The first method has the virtue that, subject to the constraint on the maximum acceleration, it processes the most urgent rules first. The second has the virtue that every rule has some influence on the final outcome. Of course, since this is just a heuristic approach, the developer typically decides what sort of approach yields the most realistic results.

**Crowd Simulation:** The last, and most interesting, type of motion simulation is that of crowds of agents. Unlike flocking systems, in which it is assumed that the agents behave homogeneously, in a crowd it is assumed that every agent has its own agenda to pursue (see Fig. 4). For example, we might imagine a group of students walking through a crowded campus on their way to their next classes. Since such agents are acting within a social system, however, it is assumed that they will tend to behave in a manner that is consistent with social conventions. (I don't want you to bump into me, and so I will act in a manner to avoid bumping into you as well.)

Fig. 4: Crowd simulation.

Crowd simulation is actually a very broad area of study, ranging from work in game programming and computer graphics, artificial intelligence, social psychology, and urban architecture (e.g., planning evacuation routes). In order to operate in the context of a computer game, such a system needs to be decentralized, where each member of the crowd determines its own action based on the perceived actions of other nearby agents. The problem with applying a simple boid-like flocking behavior is that, whereas flocking rules such as alignment naturally produce systems that avoid collisions between agents, the diverse agendas of agents in crowds naturally brings them directly into collisions with other agents (as in pedestrians traversing a crosswalk from both sides). In order to produce more realistic motion, the agents should anticipate where nearby agents are moving, and then plan their future motion accordingly.

In order to plan such motions, we will next introduce the handy concept of a velocity obstacles and reciprocal velocity obstacles.

**Velocity Obstacles:** Suppose that an agent $a$ is moving in a crowded environment where many other agents are also moving. We assume that $a$ can perceive the motions of nearby agents and generate rough estimates on their velocities. Agent $a$ is also interested in traveling to some destination, and (based on our path finding algorithm) we assume that $a$ has a *preferred velocity*, $v_a^*$, which it would assume if there were no other agents to consider.

For the sake of simplicity, we will model agents as circular disks translating in the plane. Consider two such agents, $a$ and $b$, of radii $r_a$ and $r_b$ and currently positioned at points $p_a$ and $p_b$, respectively (see Fig. 5(a)). If we assume that agent $a$ moves at velocity $v_a$, then at time $t$ it will have moved a distance of $t \cdot v_a$ from its initial position, implying that it will be located at $p_a + t \cdot v_a$. We will refer

to this as $p_a(t)$.

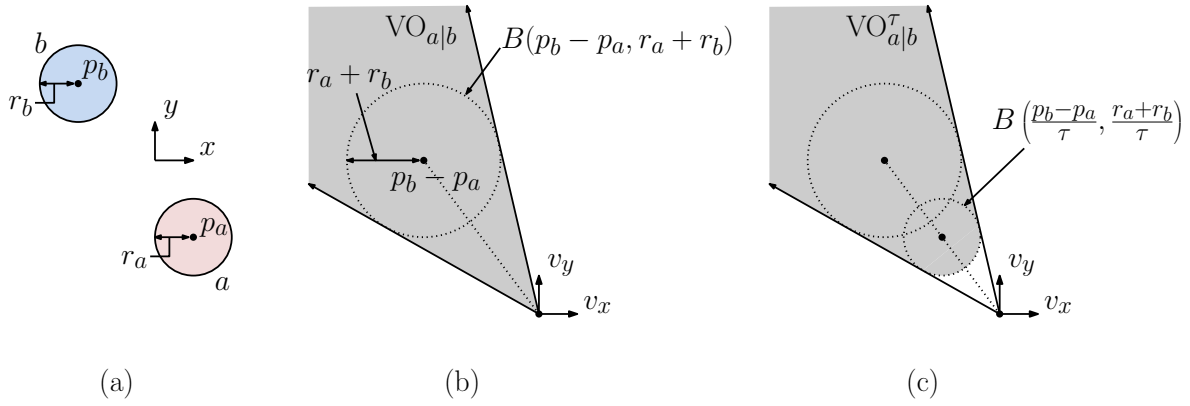


Fig. 5: Velocity obstacle for $a$ induced by $b$ (assuming $b$ is stationary).

For now, let's assume that object $b$ is not moving, that is, $p_b(t) = p_b$ for all $t$. Let's consider the question of how to select a velocity for $a$ that avoids hitting $b$ any time in the future. Two disks intersect if the distance between their centers ever falls below the sum of their radii. More formally, let $\|u\|$ denote the Euclidean length of a vector $u$. Then $\|p_a(t) - p_b(t)\|$ is the distance between $p_a$ and $p_b$ at time $t$ (that is, the length of the vector from $b$ to $a$). Thus, the two agents collide if $\|p_a(t) - p_b(t)\| < r_a + r_b$. By substituting in the definitions of $p_a(t)$ and $p_b(t)$, we find that a velocity $v_a$ will result in a collision some time in the future if there exists a $t > 0$, such that

$$\|(p_a + t \cdot v_a) - p_b\| \ < \ r_a + r_b. \tag{1}$$

We would like to express the velocities $v_a$ that satisfy the above criterion as lying within a certain "forbidden region" of space. To do this, define $B(p, r)$ to be the open Euclidean ball of radius $r$ centered at point $p$. That is

$$B(p, r) \ = \ \{q : \|q - p\| < r\}.$$

We can rewrite Eq. (1) as

$$\|t \cdot v_a - (p_b - p_a)\| \ < \ r_a + r_b,$$

which is equivalent to saying that $t \cdot v_a$'s distance from the point $p_b - p_a$ is less than $r_a + r_b$, or equivalently, the (parametrized) vector $t \cdot v_a$ lies within a ball of radius $r_a + r_b$ centered at the point $p_b - p_a$. Thus, we can rewrite Eq. (1) as

$$t \cdot v_a \in B(p_b - p_a, r_a + r_b),$$

As $t$ varies from 0 to $+\infty$, the vector $t \cdot v_a$ travels along a ray that starts at the origin and travels in the direction of $v_a$. Therefore, the set of forbidden velocities are those that lie within a cone that is centered at the origin and encloses the ball $B(p_b - p_a, r_a + r_b)$.

We define the *velocity obstacle* of $a$ induced by $b$, denoted $\mathrm{VO}_{a|b}$ to be the set of velocities of $a$ that will result in a collision with the stationary object $b$.

$$\mathrm{VO}_{a|b} \ = \ \{v : \exists t \geq 0, \ t \cdot v \in B(p_b - p_a, r_a + r_b)\}.$$

(See Fig. 5(b).)

One problem with the velocity object is that it considers time going all the way to infinity. In a dynamic setting, we cannot predict the motion of objects very far into the future. So, it makes sense to truncate the velocity obstacle so that it only involves a limited time window. Given a time value $\tau > 0$, what the set of velocities that will result in agent $a$ colliding with agent $b$ at any time $t$, where $0 < t \leq \tau$ is the *truncated velocity obstacle*:

$$\text{VO}^{\tau}_{a|b} \;=\; \{v : \exists t \in [0, \tau],\; t \cdot v \in B(p_b - p_a, r_a + r_b)\}.$$

This is a subset of the (unbounded) velocity obstacle that eliminates very small velocities (since collisions farther in the future result when $a$ is moving more slowly). The truncated velocity obstacle is a truncated cone, where the truncation occurs at the boundary of the $(1/\tau)$-scaled ball $B((p_b - p_a)/\tau, (r_a + r_b)/\tau)$ (see Fig. 5(c)). Observe that there is an obvious symmetry here. Moving $a$ with velocity $v$ will result in a collision with (the stationary) $b$ if and only if moving $b$ with velocity $-v$ will result in an intersection with (the stationary) $a$. Therefore, we have

$$\text{VO}^{\tau}_{a|b} \;=\; -\text{VO}^{\tau}_{b|a}.$$

**Collision-Avoiding Velocities:** Next, let us consider how the velocity obstacle changes if $b$ is moving. If we assume that $b$ is moving with velocity $v_b$, then a velocity $v_a$ will generate a collision precisely if the differences in their velocities $v_a - v_b$ generates a collision under the assumption that $b$ is stationary, that is, $v_a - v_b \in \text{VO}^{\tau}_{a|b}$. Equivalently, $v_a$ will generate a collision if $b$ if $v_a$ lies in the offset velocity obstacle $\text{VO}^{\tau}_{a|b} + v_b$ (see Fig. 6(a)).

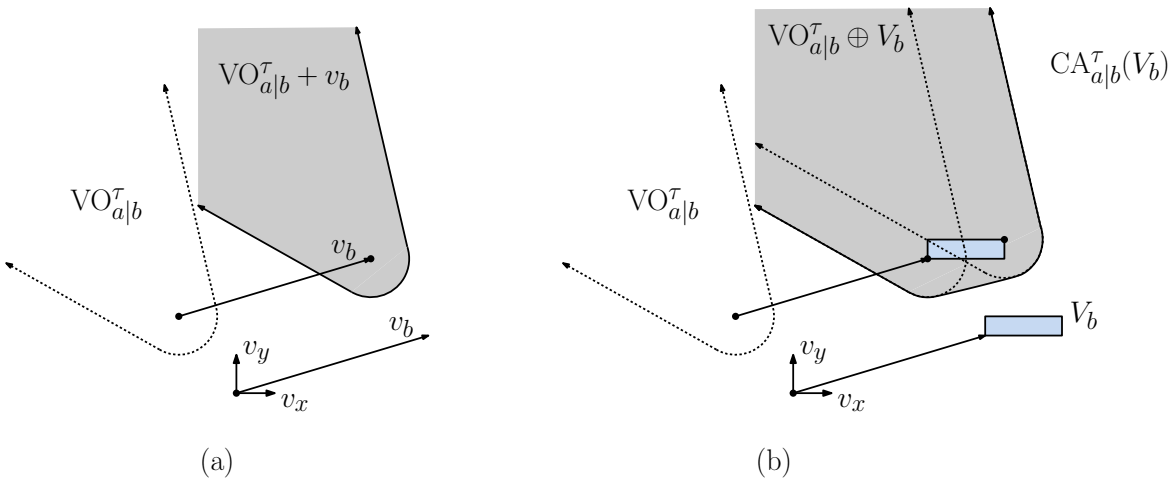


Fig. 6: Velocity obstacles where (a) object $b$ is moving at velocity $v_b$ and (b) object $b$ is moving at any velocity in the set $V_b$.

We can further generalize this. We usually do not know another object's exact velocity, but we can often put bounds on it. Suppose that rather than knowing $b$'s exact velocity, we know that $b$ is moving at some velocity $v_b$ that is selected from a region $V_b$ of possible velocities. (For example, $V_b$ might be square or circular region in space, based on the uncertainty in its motion estimate.)

Let us consider the velocities of $a$ that *might* result in a collision, assuming that $v_b$ is chosen from $V_b$. To define this set, we first define the *Minkowski sum* of two sets of vectors $X$ and $Y$ to be the set consisting of the pairwise sums of vectors from $X$ and $Y$, that is,

$$X \oplus Y \;=\; \{x + y : x \in X \text{ and } y \in Y\}.$$

Then, clearly $a$ might collide with $b$ if $a$'s velocity is chosen from $\text{VO}^\tau_{a|b} \oplus V_b$. Therefore, if we want to avoid collisions with $b$ altogether, then $a$ should select a velocity from outside this region. More formally, we define the set of *collision-avoiding velocities* for $a$ given than $b$ selects a velocity from $V_b$ is

$$\text{CA}^\tau_{a|b}(V_b) \;=\; \{v : v \notin \text{VO}^\tau_{a|b} \oplus V_b\}$$

(see Fig. 6(b).)

Just to recap, if $a$ selects its velocity vector from anywhere outside $\text{VO}^\tau_{a|b} \oplus V_b$ (that is, anywhere inside $\text{CA}^\tau_{a|b}(V_b)$), then no matter what velocity $b$ selects from $V_b$, $a$ is guaranteed not to collide with $b$ within the time interval $[0, \tau]$.

This now provides us with a strategy for selecting the velocities of the agents in our system:

- Compute velocity bounds $V_b$ for all nearby agents
- Compute the intersection of all collision-avoiding velocities for these objects, that is

$$\text{CA}^\tau_a = \bigcap_b \text{CA}^\tau_{a|b}(V_b)$$

  Any velocity chosen from this set is guaranteed to avoid collisions from now until time $\tau$.
- Select the vector from $\text{CA}^\tau_a$ that is closest to $a$'s ideal velocity $v^*_a$.

In practice, we need to take some care in the implementation of this last step. First, there will be upper limits on fast an object can move or change directions. So, we may not be free to select any velocity we like. Subject to whatever practical limitations we have on what the future velocity can be, we select the closest one that lies within $\text{CA}^\tau_a$. If there is no such vector, then we must consider the possibility that we cannot avoid a collision. If so, we can select a vector that overlaps the smallest number of velocity obstacles.

**Issues:** While this might seem to be the end of the story with respect to velocity obstacles, there are still issues that arise. One of these issues was hinted at in the last paragraph. In cases where there are lots of agents and the velocity estimates are poor, the collision-avoiding area may be empty. There are a number of strategies that one might consider for selecting a good alternative.

There is a more significant problem with this approach, however, which arises even if we consider only two agents. The problem is that agents that are moving towards each other can engage in a very unnatural looking oscillating motion. The situation is illustrated in Fig. 7. Two agents are moving to each other. They see that they are on a collision course, and so the divert from their ideal velocities. Let's imagine the best-case scenario, where they have successfully resolved their impending collision (whew!) as a result of this diversion (see Fig. 7(b)). Now, each agent sees that the other agent has diverted from its trajectory and reasons, "Great! The other guy has veered off, and I have won the game of chicken. So, now I can resume on my original velocity" (see Fig. 7(c)). So, both agents

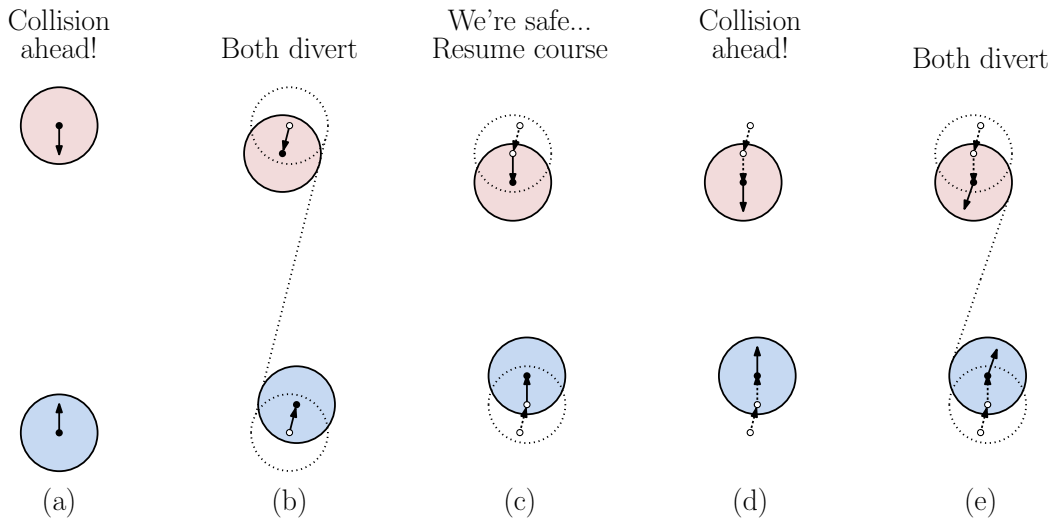


Fig. 7: Oscillation that can result from standard velocity-obstacle motion planning.

return to their original velocity, and they are now right back on a collision course (see Fig. 7(d)) and so again they divert (see Fig. 7(e)). Clearly, this vicious cycle of zig-zagging motion will repeat until they manage to make it around one another.

Although even humans sometimes engage in this sort of brief oscillation when meeting each other head-on, this sort of repeated oscillation is very unnatural, and is due to the fact that both agents are acting without consideration for what the other agent might reasonable do. The question then is how to fix this?

**Reciprocal Velocity Obstacles:** The intuition behind fixing the oscillation issue is to share responsibility. We assume that whenever a collision is possible between two agents, both agents perceive the danger and (since they are running the same algorithm) they both know how to avoid it. Rather than having each agent assume total responsibility for correcting its velocity, we instead ask each agent to take on *half* of the responsibility for avoiding the collision. In other words, each agent diverts its path (that is, alters its velocity) by exactly half the amount needed to avoid the collision. It assumes that the other agent will *reciprocate*, by performing the other half of the diversion. It turns out that this greatly reduces the oscillation problem, since two head-on agents will now divert just enough to avoid each other.

This leads to the concept of *reciprocal velocity obstacles*. Before defining this notion, let us recall the sets $\mathrm{CA}^{\tau}_{a|b}(V_b)$, the collision-avoiding velocities for $a$ assuming that $b$ selects its velocity from $V_b$, and $\mathrm{CA}^{\tau}_{b|a}(V_a)$, the collision-avoiding velocities for $b$ assuming that $a$ selects its velocity from $V_a$. We say that two sets of candidate velocities $V_a$ and $V_b$ are *reciprocally collision avoiding* if

$$V_a \subseteq \mathrm{CA}^{\tau}_{a|b}(V_b) \qquad \text{and} \qquad V_b \subseteq \mathrm{CA}^{\tau}_{b|a}(V_a).$$

This implies a very harmonious set of candidate velocities, since for any choice $v_a \in V_a$ and $v_b \in V_b$, we can be assured that these two agents will not collide.

Note that there is a complimentary relationship between these two candidate sets. As we increase the possible velocities in $V_a$, we reduce the possible set of velocities that $b$ can use to avoid a collision,

and vice versa. Of course, we would like be as generous as we can, by giving each agent as much flexibility as possible. We say that two such candidate velocity sets are *reciprocally maximal* if

$$V_a \;=\; \mathrm{CA}^{\tau}_{a|b}(V_b) \qquad \text{and} \qquad V_b \;=\; \mathrm{CA}^{\tau}_{b|a}(V_a).$$

Note that we face a tradeoff here, since we could make $V_a$ very large, but at the expense of making $V_b$ very small, and vice versa. There are infinitely many reciprocally maximal collision avoiding sets. So what should guide our search for the best combination of candidate sets? Recall that each agent has its preferred velocity, $v_a^*$ and $v_b^*$. It would seem natural to generate these sets in a manner that gives each agent the greatest number of options that are close to its preferred velocity. We seek a pair of candidate velocity sets that are *optimal* in the sense that they provide each agent the greatest number of velocities that are close to the agent's preferred velocity.

There are a number of ways of making this concept more formal. Here is one. Consider two pairs $(V_a, V_b)$ and $(V'_a, V'_b)$ of reciprocally maximal collision avoiding sets. For any radius $r$, $B(v_a^*, r)$ denotes the set of velocities that are within distance $r$ of $a$'s preferred velocity and $B(v_b^*, r)$ denotes the set of velocities that are within distance $r$ of $b$'s preferred velocity. The quantity area$(V_a \cap B(v_a^*, r))$ can be thought of as the "number" (more accurately the measure) of candidate velocities for $a$ that are close (within distance $r$) of its preferred velocity. Ideally, we would like both area$(V_a \cap B(v_a^*, r))$ and area$(V_b \cap B(v_b^*, r))$ to be large, so that both agents have access to a large number of preferred directions. One way to guarantee that two numbers are large is to guarantee that their minimum is large. Also, we would like the pair $(V_a, V_b)$ to be fair to both agents, in the sense that area$(V_a \cap B(v_a^*, r)) =$ area$(V_b \cap B(v_b^*, r))$. This means that they both agents have access to the same "number" of nearby velocities.

Combining the concepts of fairness and maximality, we say that a pair $(V_a, V_b)$ of reciprocally maximal collision avoiding sets is *optimal* if, for all radii $r > 0$, we have

**Fair:** area$(V_a \cap B(v_a^*, r)) =$ area$(V_b \cap B(v_b^*, r))$

**Maximal:** For any other reciprocal collision avoiding set $(V'_a, V'_b)$,

$$\min(\text{area}(V_a \cap B(v_a^*, r)), \text{area}(V_b \cap B(v_b^*, r))) \;\geq\; \min(\text{area}(V'_a \cap B(v_a^*, r)), \text{area}(V'_b \cap B(v_b^*, r))).$$

Now that we have defined this concept, it is only natural to ask whether we have any hope of computing a pair of sets satisfying such lofty requirements. The remarkable answer is yes, and in fact, it is not that hard to do! The solution is described in a paper by J. van den Berg, M. C. Lin, D. Manocha (see the readings at the start of these notes). They define an *optimal reciprocal collision avoiding pair* of candidate velocities, which they denote by $(\mathrm{ORCA}^{\tau}_{a|b}, \mathrm{ORCA}^{\tau}_{b|a})$, to be a pair of velocity sets that satisfy the above optimality properties.

They show how to compute these two sets as follows. First, let us assume that the preferred velocities of the two agents puts them on a collision course. (This is just for the sake of illustration. The construction works even if this is not the case.) That is, $v_a^* - v_b^* \in \mathrm{VO}^{\tau}_{a|b}$. Clearly, we need to divert one agent or both to avoid the collision, and we will like this diversion to be as small as possible. Let $u$ denote the vector on the boundary of $\mathrm{VO}^{\tau}_{a|b}$ that lies closest to $v_a^* - v_b^*$ (see Fig. 8(a)). Since $\mathrm{VO}^{\tau}_{a|b}$ is just a truncated cone, it is not too hard to compute the vector $u$. (There are basically two cases, depending on whether the closest boundary point lies on one of the flat sides or on the circular arc at the base.)

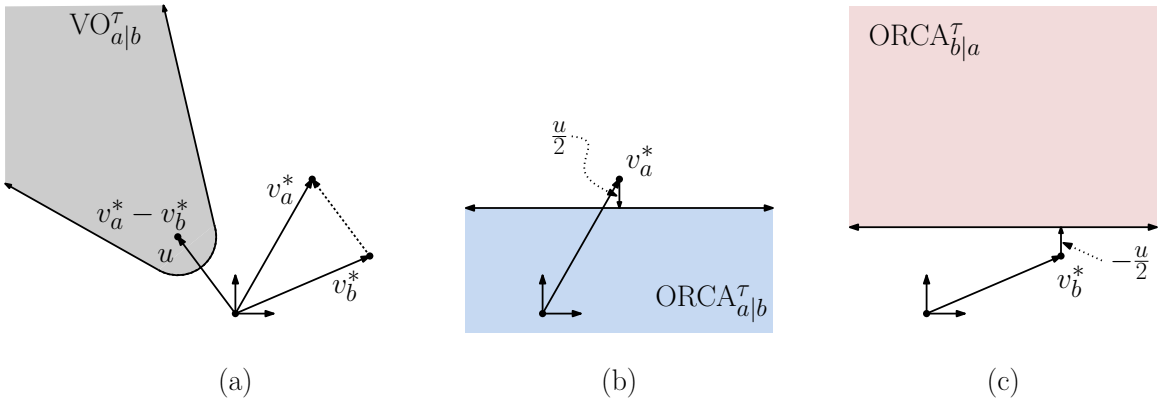


Fig. 8: Computing an optimal reciprocal collision avoiding pair of candidate velocities.

Intuitively, $u$ reflects the amount of relative velocity diversion needed to just barely escape from the collision zone. That is, together $a$'s diversion plus $b$'s diversion (negated) must sum to $u$. We could split the responsibility however we like to. As we had discussed earlier, for the sake reciprocity, we would prefer that each agent diverts by exactly half of the full amount. That is, $a$ will divert by $u/2$ and $b$ will divert by $-u/2$. (To see why this works, suppose that $v'_a = v^*_a + u/2$ and $v'_b = v^*_b - u/2$. The resulting relative velocity is $v'_a - v'_b = v^*_a - v^*_b + u$, which is a collision-free velocity.)

In general, there are a number of choices that $a$ and $b$ could make to avoid a collision. Let $n$ denote a vector of unit length that points in the same direction as $u$. We would like $a$ to change its velocity from $v^*_a$ to a velocity whose orthogonal projection onto the vector $n$ is of length at least $\|u/2\|$. The set of allowable diversions defines a half-space (that is, the set of points lying to one side of a line), and is given by the following formula:

$$\mathrm{ORCA}^{\tau}_{a|b} \;=\; \left\{ v : \left( v - \left( v^*_a + \frac{u}{2} \right) \right) \cdot n \geq 0 \right\},$$

(where the $\cdot$ denotes the dot product of vectors). This formula defines a halfspace that is orthogonal to $u$ and lies at distance $\|u/2\|$ from $v^*_a$ (see Fig. 8(b)). Define $\mathrm{ORCA}^{\tau}_{b|a}$, symmetrically, but using $-\frac{u}{2}$ instead (see Fig. 8(c)). In their paper, van den Berg, Lin, and Manocha claim that the resulting pair of sets $(\mathrm{ORCA}^{\tau}_{a|b}, \mathrm{ORCA}^{\tau}_{b|a})$ define an optimal reciprocally maximal pair of collision avoiding. In other words, if $a$ selects *any* velocity from $\mathrm{ORCA}^{\tau}_{a|b}$ and $b$ selects any velocity from $\mathrm{ORCA}^{\tau}_{b|a}$, and these two sets are both fair and provide the greatest number of velocities that are close to both $a$ and $b$'s ideal velocities.

This suggests a solution to the problem of planning the motion of $n$ bodies. Let $B = \{b_1, \ldots, b_n\}$ denote the set of bodies other than $a$. Compute the ORCA sets for $a$ relative to all the other agents in the system. That is, $\bigcap_{i=1}^{n} \mathrm{ORCA}^{\tau}_{a|b_i}$. Since each of these regions is a halfplane, there intersection defines a convex polygon. Next, find the point $v'_a$ in this convex polygon that minimizes the distance to $v^*_a$. This point defines $a$'s next velocity. By repeating this for every agent in your system, the result is a collection of velocities that are mutually collision free, and are as close as possible to the ideal velocities.

There are two shortcomings with this approach. First, if the agents are very close to one another, it may be that the intersection of the collision-free regions is empty. In this case, we may need to find an alternate strategy for computing $a$'s velocity (or simply accept the possibility of an intersection).

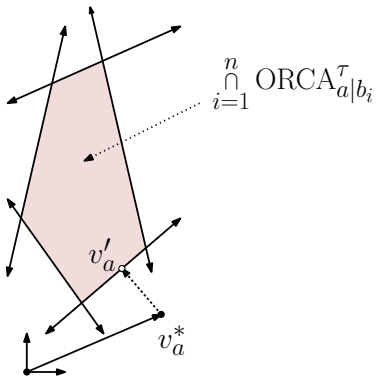


Fig. 9: Computing agent $a$'s velocity.

The other shortcoming is that it requires every agent to know the preferred velocity $v^*_{b_i}$ for each of the other objects in the system. While the simulator may know this, it is not reasonable to assume that every agent knows this information. A reasonable alternative is to form an estimate of the *current velocity*, and use that instead. The theory is that most of the time, objects will tend to move in their preferred direction.