CMSC 425: Lecture 19
Motion Planning: Multiple Agent Motion

Reading: Today's material is drawn from a variety of sources. 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 pursuit and evasion is taken from a number of random sources including S. M. Lavalle, *Planning Algorithms*, Cambridge University Press, 2006.

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:

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.

Flocking Behavior: Flocking refers to the motion that arises when a group of agents adopt a decentralized motion strategy that is designed to hold the group together (and to perhaps achieve other objectives, such as evading a threat). Such behavior is exemplified by the motion of groups animals, such as birds, fish, and insects (see Fig. 1).

Fig. 1: 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 (see Fig. 2(a)). Whenever another boid gets too close, the force from this field will tend to push them apart.

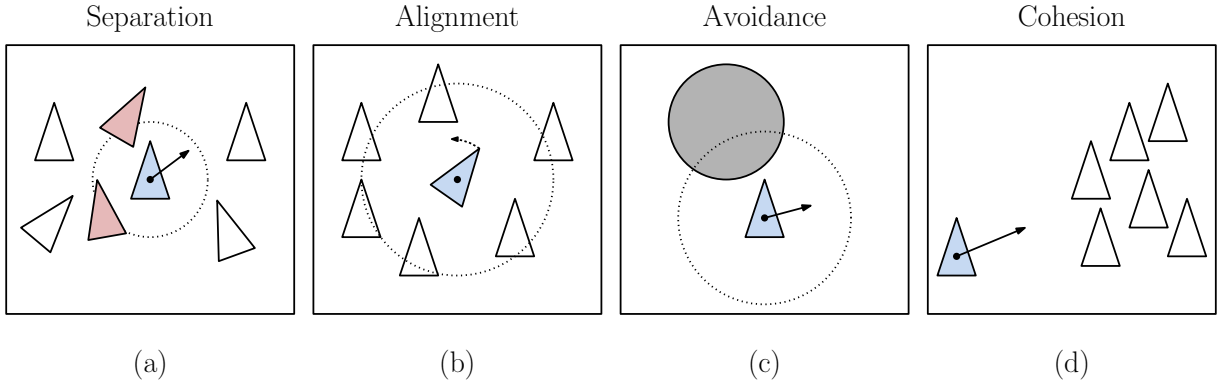

Fig. 2: Forces that control flocking behavior.

Alignment: Each boid's direction of flight is aligned with nearby boids (see Fig. 2(b)). Thus, local clusters of boids will tend to point in the same direction and hence will tend to fly in the same direction. To implement this, we sample the heading vectors for the boids in our immediate neighborhood, and compute their average, which we normalize to unit length. We then apply a torque (turning force) to the current boid that pushes it in the direction of this average.

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 (see Fig. 2(c)). As a boid approaches the object, this repulsive field will tend to cause the boid to deflect its flight path, thus avoiding a collision. The strength of this force grows very rapidly as the boid comes closere to the obstacle. 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 (see Fig. 2(d)). (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.)

Physics-Based Motion: Before getting into the details of how to implement a boids-based motion

system, it is useful to see how we can apply physical forces to induce a motion-planning system. We discussed this briefly in the context of potential-field methods in an earlier lecture. Here, we consider it in the context of a computational process called *numerical physical integration*.

Let us assume we have n boids, indexed from 1 to n. We also assume that time is discretized into small time units. Let Δ denote the time step. (In Unity, this is the time step used for FixedUpdate and might be chosen to be 1/10 of a second.) For example, we start at time $t = t_0 = 0$, and the jth time step is $t_j = j\Delta$. Consider the state of the system at time t. For boid i, let $p_i(t)$ denote this boid's *current position*. This could be represented as the homogeneous coordinates of a point in affine space. Let $\vec{v}_i(t)$ denote this boid's *current velocity*. This could be represented as the homogeneous coordinates of a vector in affine space.

As mentioned above, various forces will be evaluated for each boid at each time step in order to "nudge" the boid into its desired new velocity. Each force is represented as a vector, and the sum of these forces will result in an *aggregate force* vector, denoted $\vec{F}_i(t)$. (For example, this aggregate force will tend to push the boid away from other boids, turn it in the direction that its neighbors are flying, draw it away from obstacles and threats, and pull it back towards the center of the flock.)

Recall from basic physics, the rule $f = ma$ (force is mass times acceleration). In our context, we interpret this as $a = f/m$ (acceleration is force divided by mass). Let m_i denote the mass of the ith boid. (In most systems, they all have the same mass, so this is just a constant "fudge factor" that is applied to all the update computations.) The acceleration of the ith boid at time t is given as a vector $\vec{a}_i(t)$. We can compute this acceleration as

$$\vec{a}_i(t) \leftarrow \frac{\vec{F}_i(t)}{m_i}.$$

(Observe that the right-hand side is a vector quantity, since it is the product of a scalar $1/m_i$ and a vector $\vec{F}_i(t)$.)

The change in velocity over a time step Δ is just the product of Δ and the acceleration. Thus, at each time t we modify the current velocity vector based on the aggregate force vector:

$$\vec{v}_i(t + \Delta) \leftarrow \vec{v}_i(t) + \Delta \cdot \vec{a}_i(t).$$

(Observe that the right-hand side is a vector quantity, since it is the sum of a vector $\vec{v}_i(t)$ and the product of a scalar Δ and a vector $\vec{a}_i(t)$.)

Once we know the new velocity, we can update the boid's position. Its displacement over a time interval Δ is just the product of its new velocity and the elapsed time. That is,

$$p_i(t + \Delta) \leftarrow p_i(t) + \Delta \cdot \vec{v}_i(t + \Delta)$$

The right-hand-side of this equation is the sum of a point ($p_i(t)$) and the product of a scalar (Δ) and a vector ($\vec{v}_i(t + \Delta)$), which by the rules of affine geometry, is a point. Of course, after this movement new forces will come into existence, and the process repeats.

We can simplify this by removing explicit mention of the boid index and the time. Given the current boid state $(p, \vec{v})$, we compute its updated state $(p', \vec{v}')$ as:

$$\vec{a} \leftarrow \frac{\vec{F}}{m}, \qquad \vec{v}' \leftarrow \vec{v} + \Delta \cdot \vec{a} \qquad \text{and} \qquad p' \leftarrow p + \Delta \cdot \vec{v}'.$$

This leads to the simple numerical integrator described in the code block. (We hasten to add that this integrator is *very inaccurate*, and should not be applied for real physical simulation. It will be good enough for our simple boid simulation, however.)

Motion by Simple Physical Integration

```
t = 0
for (i = 1 to n) {
  m[i] = ... mass of ith boid ...
  p[i] = new Point( ... initial position of ith boid ...);
  v[i] = new Vector( ... initial velocity of ith boid ...);
}
repeat forever {
  for (i = 1 to n) {
    F[i] = new Vector( ... aggregate forces acting on boid i ...);
    a[i] = F[i] / m[i];
    v[i] += Delta * a[i];
    p[i] += Delta * v[i];
  }
  t = t + Delta;
}
```

Boid Implementation: Given our simple physical integrator, the only question that remains is how to compute the aggregate force $\vec{F}_i(t)$ acting on boid i at time t. 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 much 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 apply kinematics rigorously in order to determine future motion. 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 $\vec{u}$ pointing in the induced direction and the strength s given as a scalar, we can compute the updated acceleration vector as $\vec{a} = c \cdot s \cdot \vec{u}$, 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 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.

Pursuit Evasion: The next level up from herding behavior, which is characteristic of animals, is the type of purposeful behavior that is characteristic of more human-like intelligence. There are many motion-planning problems where srategic goals are involved, such as a pack of wolves attacking a herd of elk. Both sets of agents (wolves and elk) have their own individual aims (kill or avoid being killed), and as a group they can attempt to coordinate their behaviors in order to achieve these aims. These strategies are generally quite complicated and difficult to describe formally. To make the task simpler, mathematicians have posed extremely simple special cases, called *pursuit-evasion games*, in order to better understand these strategies.

In pursuit-evasion games, the agents are divided into two groups. The pursuers attempt to track down and locate the evaders, while the latter attempt to avoid being caught by a pursuer. These games may be played in a discrete setting (e.g., where agents are located on a chess board and move one square at a time) or continuous setting (e.g., where agents are points on the real plane, and may move along any trajectory they like). There are a number of famous examples of such games whose solutions are quite complex. They come with colorful names, like the *homocidal chauffeur game* (a high-speed car driver with a limited turning radius is trying to run over a slow but highly maneuverable pedestrian) and the *princess and monster game* (a slow-moving monster in a cave is trying to catch a fast moving princess, but both operate in total darkness). Let's look at a couple of well-studied pursuit-evasion problems in greater detail.

Visibility-based Pursuit Evasion We are given a continuous domain (e.g., a simple polygon) in the plane. Two moving points, the pursuer p and the evader e, can move in time but must stay within the domain. Let $p(t)$ and $e(t)$ denote their positions at some time $t \geq 0$. The pursuer has *caught* the evader if at any time $t_1 > 0$, $p(t_1)$ can see $e(t_1)$, meaning that the line segment $\overline{p(t)e(t)}$ lies entirely within the interior of the domain. We assume that the evader knows where the pursuer is at times and can move at arbitrarily high speeds. The question is whether it is possible to plan a path for the pursuer such

that the evader cannot escape being caught eventually. (Note that the notion of being "caught" really just means being "seen.")

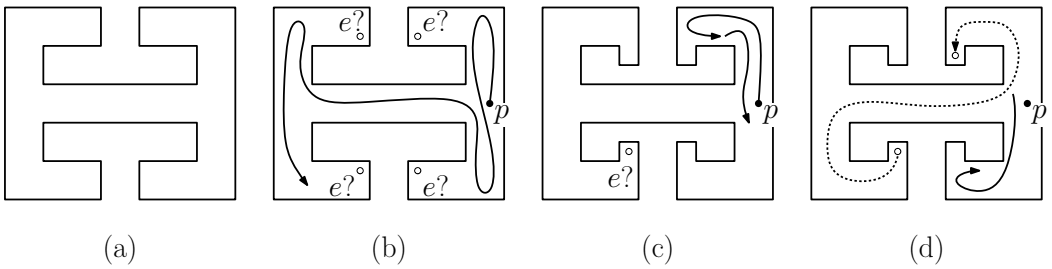

Fig. 3: Visibility-based pursuit evasion. In the case of polygon (a), the pursuer has a winning strategy, but in (c) the evader can always evade the pursuer.

For example, consider the domain shown in Fig. 3. For the domain shown in (a), the pursuer wins the game by following the path shown in (b). No matter where the evader attempts to hide, he will eventually be seen and further he cannot sneak from one hiding place to another without being seen by the pursuer. However if we add four small nobs at the end of each of the four bays, the evader now wins the game (assuming he knows the pursuer's strategy in advance).

Seeing why the evader can elude detection involves an analysis of the various cases of the sequence of bays visited by the pursuer. For example, suppose that the pursuer vists the northeast (NE) bay first, then the southeast (SE), then the northwest (NW), and then comes back to the SE bay, and suppose that the evader starts in the NW bay. The evader could reason as follows. After the pursuer leaves NE and moves down to look into SE, the evader zips from NW to NE. When traverses the central horizontal corridor and starts moving up to the NW bay, the evader is free to move up into the NE bay. Finally, when the pursuer returns to SE, the evader has left it. After a bit of thinking, you should be able to convince yourself that by looking ahead to the pursuer's next move, the evader can always identify a bay in which to hide now and escape from later.

To see whether you understand this, consider the four domains shown in Fig. 4. For which of these does the pursuer have a winning strategy, and for which does the evader?

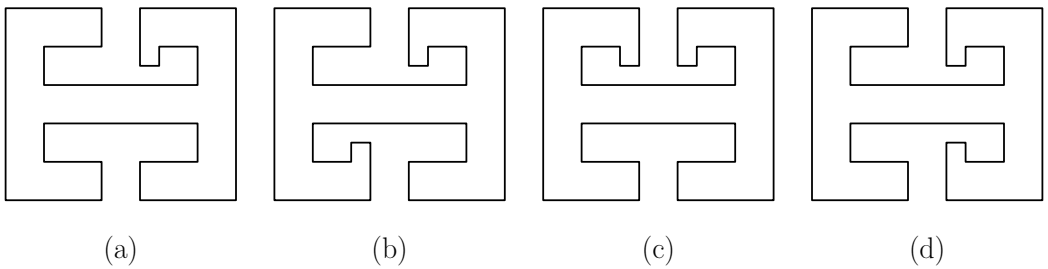

Fig. 4: Visibility-based pursuit evasion. In which of these does the pursuer win? (Hint: The evader wins in only one of them.)

Given a simple polygon, it is possible to determine whether there exists a pursuit path

(that is, a winning solution for the pursuer). However, the algorithm is quite complex. It runs in time $O(n^2)$, where n is the number of vertices in the polygon. The success of the evader depends on its complete knowledge of the pursuer's path. If the pursuer is allowed to use randomization (coin flips) to determine its moves, then it can be shown that the pursuer will eventually get lucky and find the evader, no matter how clever the evader is.

Lion and Man: This game is played in the first quadrant of the real plane ($x \geq 0$ and $y \geq 0$). There is a lion that starts at a point $L = (L_x, L_x)$ and a man at point $M = (M_x, M_y)$. At each step, the man may take one move of length up to 1 unit, and then the lion may take one move of length up to 1 unit, and this repeats (see Fig. 5(a)). Assuming both players play in the best way possible, either the lion will eventually catch the man (when the two points coincide) or the man can evade the lion forever. Based on the starting configuration, which is it?

There are a few cases that are easy to see. First off, if the man is either north ($M_y \geq L_y$) or east ($M_x \geq L_x$) then the man can escape being caught forever. But what if $M_x < L_x$ and $M_y < L_y$? It turns out that the man will eventually be caught. However, it is not easy to see exactly why.

Here is a winning strategy for the tiger for the above case. First, observe that M lies below and to the left of L, then there exists a point C with the following properties: (1) L lies along the line segment $\overline{CM}$, (2) a circle centered at C and passing through L intersects both the positive x- and positive y-axes (Fig. 5(b)).

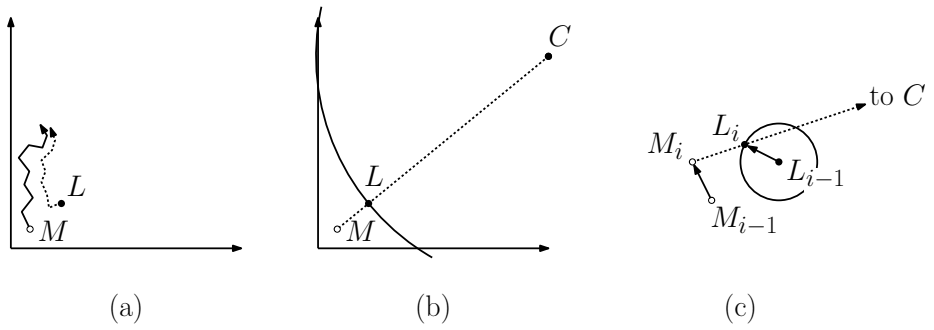

Fig. 5: Lion and Man Problem

Suppose that during the ith step, the man moves from point M_{i-1} through some distance $0 < \delta \leq 1$ to a point M_i. Draw a circle centered at L_{i-1} of radius δ. Observe that (given the relative positions of M_{i-1}, L_{i-1} and C) a disk of radius δ centered at L_{i-1} intersects the line segment $\overline{M_iC}$. Set L_i to the closest point along this line segment to M_i (see Fig. 5(c).) Observe that the invariant is preserved, and L_i is a bit closer to M_i. Eventually, the man will lie within this circle of radius δ, and the lion catches the man.