Pierogoids: An Extended Boid Implementation

6.837 Graphics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Claire Cheng
ccheng00@mit.edu

Kye Burchard
kyeb@mit.edu

I. Motivation

The goal is to simulate 3D flocking behavior of groups of bird-like objects. We were inspired by the simplicity and realistic nature of existing boid simulations and wanted to implement our own twist.

Boid simulations utilize relatively simple rules, but complex behaviors arise from the interplay between them. Without implementing any sort of AI or individual control for any boid, they collectively affect one another to create the flocking behavior seen.

Additionally, we decided to inject our own creative twist to the traditional bird model. Instead of birds or triangles, our subjects were 3D pierogies[1], and we added extensions such as user-input sliders, pierogi point-of-view, and other forces that are detailed in Section IV.

Fig. 1. Pierogi 3D Model from Studenckie Koło Astronautyczne

II. Background

Our simulation is based on Craig Reynolds' boids paper[2] in 1987, which aimed to model flocking behavior of birds and produced a simulation that lay somewhere between predictability and randomness — the so-called edge of chaos. The overarching concept is that the individual behavior of each bird matters less so than the holistic group behavior, which makes it easy to model with relatively simple mechanics.

Boid simulations are still widely used, with a variety of extensions and changes. Beyond just video games, they have been used in swarm robotics for controlling UAVs, CGI bat swarms, and even for generating motions for armies on foot or horseback in movies and TV shows.

[1]https://www.facebook.com/SKA.PW/posts/3403330286371177
[2]https://dl.acm.org/doi/10.1145/37402.37406

III. Approach

Starting with the codebase from assignment 3, we built a box in which the simulation is contained in order to keep the boids in camera view forever.

The rules governing flock behavior can be separated into 3 main components:

- Separation — avoid getting too close to each other or colliding
- Alignment — to move in the same general direction, attempt to match the average velocity of the flock
- Cohesion — to stick together, attempt to match the average position of the flock

At each time step of the simulation, we compute a "force" applied by each rule, multiply it by a user-determined coefficient, then integrate to determine the next position and velocity.

A. Separation

Each pierogi in the flock is subjected to a separation "force" in the direction of a vector, which is calculated by averaging over the weighted directions from the pierogi to each of the other pierogies.

Both the desired separation distance and a force coefficient are user adjustable.

B. Alignment

Each pierogi in the flock is also subjected to an alignment "force," which is a vector that points in the direction of the average velocity of the entire flock.

Our alignment force also includes a `fov` or "field of view" parameter. Only boids within the `fov` distance are considered part of the flock, and any outside of

```
steer = vec3(0);
count = 0;
for (boid : boids) {
  d = dist(current_boid, boid);
  dir = cur_pos - boid.pos;
  if (d < desired_separation) {
    steer += normalize(dir)/d;
    count++;
    }
}
return steer/count;
```

Fig. 2. Pseudocode for the separation algorithm.

```
steer = vec3(0);
count = 0;
for (boid : boids) {
  d = dist(current_boid, boid);
  if (d < fov) {
    steer += boid.velocity;
    count++;
    }
}
return steer/count - cur_velocity;
```

Fig. 3. Pseudocode for the alignment algorithm.

that are not included in the force computation. This is a simplified way to account for the complexities of individual vision in flocks.

C. Cohesion

Each pierogi is forced to stay within the flock by a cohesion "force," a vector that points toward the average position of all pierogis.

The strength of the cohesion force is also user-adjustable. Cohesion also takes the `fov` parameter into account.

```
steer = vec3(0);
count = 0;
for (boid : boids) {
  d = dist(current_boid, boid);
  if (d < fov) {
    steer += boid.pos;
    count++;
    }
}
return steer/count - cur_pos;
```

Fig. 4. Pseudocode for the cohesion algorithm.

D. Avoidance

Beyond the simple boid model[3], we also added walls.

In order to prevent collisions with walls, there are a variety of approaches that can be taken[4]. Rather than implementing a computationally expensive AI model of "vision" for each pierogi, we took the "force field" approach and implemented an exponentially-increasing field pushing boids away if they're too close to any of the walls. With a some tuning, this approach resulted in solid, consistent avoidance motion.

Since our environment takes the shape of a box, the force field has a sort of "corner" where pierogies will (rarely) occasionally get stuck in, typically cornered by a predator (see Section VI for a discussion). We explored several solutions to this. For example, using a rounded (spherical) force field instead of one purely proportional to the distance from the walls prevented pierogies from becoming cornered, but created an issue where the flock would very strongly tend to just skim around the sphere in a circle. with no interesting motion.

IV. Extensions

A. Adjustable parameters

A user interface was implemented in order to easily change parameters of the simulation, creating a more useful, interactive simulation.

B. Predator

Birds tend to scatter in a frenzied fashion when faced with a predatory animal, and we extended this to our flock of pierogies. In this case, our predator is a fork, and the fork would closely follow the flock as the pierogies themselves are strongly repelled by the fork and swarm every which way to avoid it.

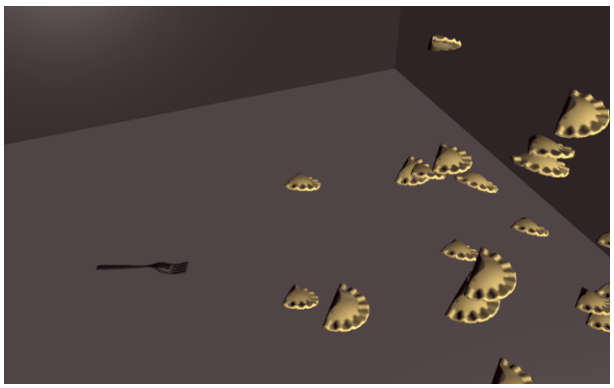

Fig. 5. Pierogies scattering away from fork

[3]Many implementations simply "wrap" around the screen at the borders. While this does provide interesting behavior, we found it to be unrealistic. Our model is closer to what a game or animation might actually use.

[4]https://www.red3d.com/cwr/nobump/nobump.html

This was accomplished by adding a large force to each pierogi in the opposing direction of the vector pointing from itself to the predator. It can be thought of as a stronger version of the Separation force mentioned in Section III.

C. Attractor

Similar to how birds swarm over crumbs of bread on the street, attractors draw the flock towards a particular point. Our attractor was a simple red sphere, and upon activation, the flock would all turn and race towards it.

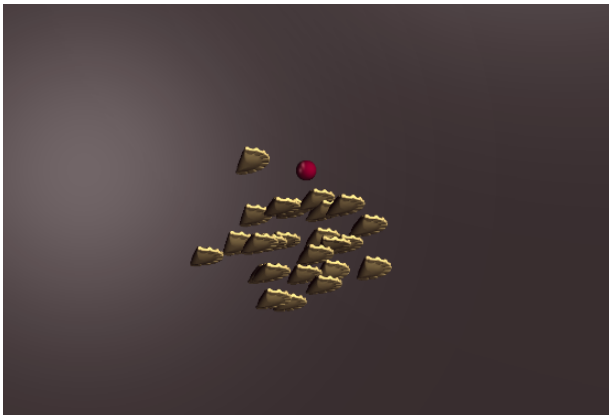

Fig. 6. Pierogies swarming towards attractor sphere

This was accomplished by adding a large force to each pierogi in the direction of the vector pointing from itself to the attractor. It can be thought of as a stronger version of the Cohesion force mentioned in Section III, this time with a stationary subject. The attractor turns "off" once the pierogis reach it, and the attractor can be reactivated in a different position by user input.

D. Point-of-view

To provide a more interesting camera angle, we implemented a second camera mode that puts you in the metaphorical driver's seat as one of the pierogies.

V. Results

Our pierogoid simulation appears reasonably realistic, and the extensions along with the user input add a nice touch of customizability.

A. Applications

Through the controls, application users can achieve a variety of interesting behaviors. Depending on the game genre[5], a simulation like this would be extremely useful for keeping computation overhead low for realistic flock/herd/school behavior. One can imagine placing an attractor node at a character's location temporarily to cause the flock to swarm toward them.

[5]We're imagining a food-based FPS or horror genre, although a simple asset swap could substantially increase its versatility.

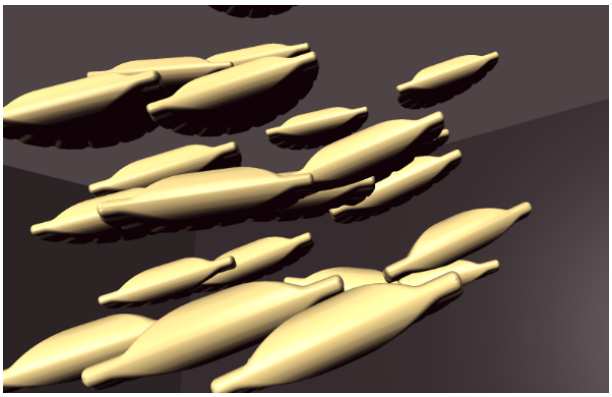

Fig. 7. Visualization from pierogi POV

Fig. 8. General flocking behavior and UI sliders

B. Performance

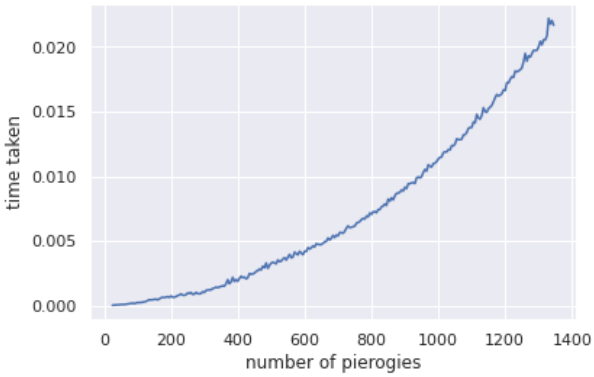

Fig. 9. Performance plot — computation time per frame vs. number of pierogies (N)

Our implementation was focused on creating a visually interesting simulation, and did not attempt to prioritize performance. Much of the computation for the forces on each pierogi at each time step is duplicated across each one.

However, since we're computer engineers, we decided to test performance anyway. For a real-time graphics ap-

plication, simulation computation speed is an important result to evaluate.

To test performance, we initially recorded framerates[6]. Using this method, we were easily able to achieve over 500 pierogies at a reasonable[7] frame rate.

We quickly noticed that the main bottleneck in performance for this system was the actual rendering, while we were more interested in the performance of our algorithm. Measuring just the section of our code used to update the velocities and positions at every time step resulted in much more stable times.

The performance matches what is expected, given the simple algorithms we implemented. Let N be the number of pierogies present. Since each pierogi requires an order-N computation (looping over the rest of the flock for each of the 3 main forces), we expect an $O(N^2)$ runtime per frame. Indeed, fig. 9 shows a quadratic curve.

The boid algorithm, as we implemented it, is inherently $O(N^2)$. The Separation algorithm must be $O(N^2)$, since it computes a difference vector between every possible pair of boids in the flock. Our version also must be $O(N^2)$ in Alignment and Cohesion as well, due to the non-linearity introduced by the `fov` term simulating a "field of view" for each boid.

However, there are still performance gains to be had over our naïve implementation. For example, the same distance and difference vectors are computed several times for each pair of boids since each of the 3 forces are separated into functions — combining into one loop would allow us to compute difference/distance vectors once per pair of boids per frame.

Fig. 10. A flock of over 800 pierogies, taken while conducting performance testing.

[6]Computed using the reciprocal of `delta_time` per frame

[7]We defined "reasonable" as 30 frames per second, but recognize that some people may have higher standards than that.

C. Demonstration of results

A live-recorded video demonstrating most features has been posted on YouTube[8].

VI. Challenges

Due to the "random" nature of the boids we sometimes ran into strange behaviors of the flock. One problem that arose was with cancelling forces. Some of the boids, when cornered by a predator, would stutter in place until one of the contributing forces is readjusted via user input slider, such as the wall force.

Since we focused on modelling flock behavior rather than individualistic behavior, another issue we encountered was that when drawn towards an attractor object, the flock would sometimes begin orbiting the object in the case of overshooting its position, instead of turning around and aiming directly at it, since none of the pierogies were permitted to break away from the flock.

Bounds were also somewhat difficult to implement since our "separation" and "wall forces" were repellent forces, instead of robust 3D collision detection.

VII. Conclusion

There are still several areas of work we would like to explore further:

- Occasionally, the bug mentioned where pierogies become cornered by a predator still occurs. We would like to explore further ways to fix this in a reasonable way, such as avoiding corners altogether.
- Performance is far from optimal. See the Section V-B for a discussion.
- Since we perform vector operations on position and velocity, the motion can occasionally look too-quick — especially when turning tight corners. Exploring approaches that operate on angle and angular velocity/momentum while maintaining a more constant velocity might be able to capture real flocking animal motion better.
- The box our pierogies are trapped in is rather limited. A more open-air environment would be much healthier for them, and could result in more interesting behavior if left to their own devices surrounded by new objects to flock around.

Overall, we were able to implement a successful 3D simulation of flocking behavior that results in believable behavior. The motions of the flock are realistic enough to be used in a quick scene in a movie or game. We enjoyed working on this project a lot, and hope you enjoy the results as well.

[8]https://youtu.be/30657Bs5SEA