Final Project: Boids

Michael "Tres" Brenan III, John Herrick
CS395T - Physical Simulation

May 19, 2020

Note: There are about 4 pages of actual content; as is expected for a graphics/physical simulation paper, there are multiple large pictures.

1 Project Overview

For our final project, we implemented an expanded version of *Boids*, which is an algorithm for obtaining realistic "flocking" behaviors from many independent agents with limited-range sensors. Beyond implementing the core three rules required to make boids flock (*separation*, *cohesion*, and *alignment*), we also implemented multiple colors of boids, simple obstacle avoidance, predators, and goal-seeking. The project was implemented using the same framework as the other class projects; we discuss the algorithm, extensions, and implementation details in the sections below.

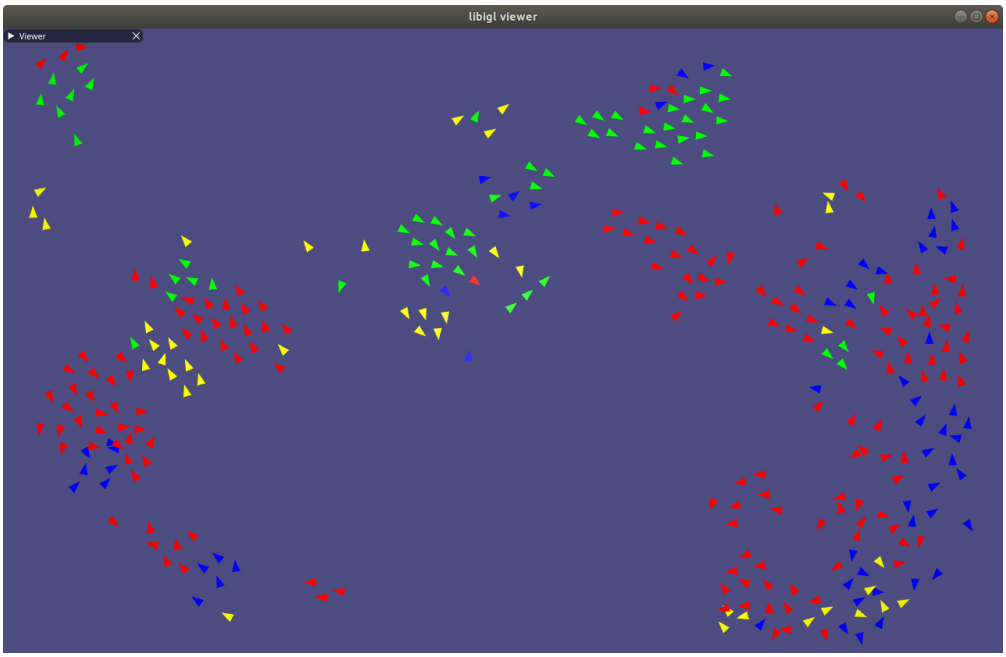
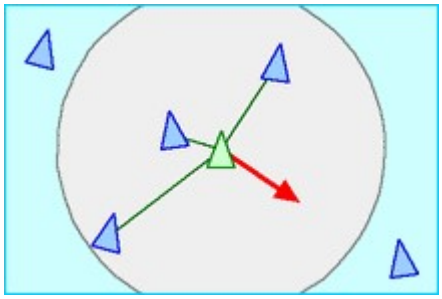


Figure 1: The program running with a few different colors of boids.

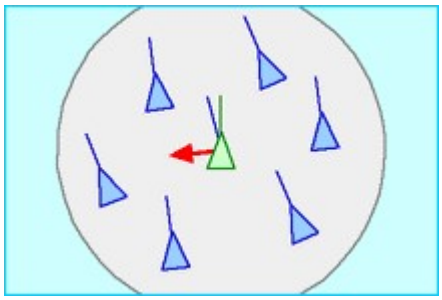
Boids: The Algorithm

The *Boids* algorithm, first presented by Craig W. Reynolds in a 1987 SIGGRAPH paper, is a distributed algorithm which orchestrates many independent actors (called "boids") into cohesive-looking flocks [1]. Each actor in this model is assumed to only have a very local view of the world - it can only observe the state (i.e., position/velocity) of other nearby actors. The actual flocking behavior, then, is created by three simple local rules:

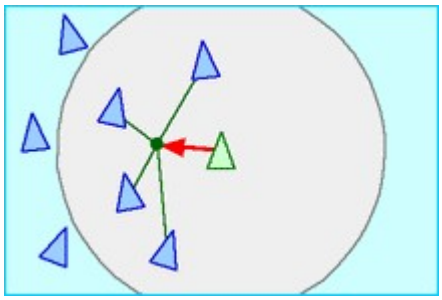
1. **Separation**: Each boid steers to avoid getting to close to nearby flockmates; this spaces out the flock and prevents overlap:



2. **Alignment**: Each boid steers so that it's heading is the same as the heading of nearby flockmates; this aligns the flock to travel in the same general direction.



3. **Cohesion**: Each boid steers towards the average position of nearby flockmates; this keeps boids closer together so that the flock does not slowly fall apart (due to drifting apart naturally). Note this force opposes the separation force.



Each of these rules imposes an independent force on each boid, which are then weighted (to modulate their strength against each other and the current boid velocity), converted to an acceleration, and finally added to the boids current velocity. Determining the proper weights for each rule is more of an art than a science, though the cohesion force should be about as strong as the separation force (so one does not dominate the other). These rules are sufficient to create flocks of boids which move semi-randomly around the grid; they work equally well in 2D as in 3D.

Since the rules all provide independent forces, it is easy to add further boid behaviors by simply defining more rules which provide more independent forces. In our project, we consider three further rules:

4. **Goal-seeking**: All boids are aware of a global goal location, and steer to head directly towards that location. This allows for boids to have more predicable global movement, instead of roaming semi-randomly.

5. **Obstacle avoidance**: Obstacles impose a strong force away from themselves to nearby boids; this prevents boids from passing through obstacles or walls.

6. **Predator avoidance**: Boids feel a strong force to avoid nearby predators (by moving directly away from them).

2 Implementation

The project was implemented using the same framework as previous projects; the implementation is primarily contained in the `lib/boid/` directory of the provided source code, and mainly in the `BoidCore` class. The simulation (which mainly occurs in `simulateOneStep()`) operates by, for each boid:

1. Independently compute each force corresponding to each rule; sum all of the forces together after weighting by parameter-provided constants.

2. Assume boid masses are 1, and obtain the resulting acceleration; add the acceleration directly to the velocity.

3. Limit the velocity to some max velocity, then add it directly to the position.

In effect, this is just explicit euler, though the simulation is stable primarily since there are negative feedback forces (like cohesion) and limits (like max velocities) which prevent it from spiralling out of control. Each force is generally a trivial average/sum over nearby flockmates (or obstacles or predators); to speed up finding nearby flockmates, we use a simple 2D grid spatial data structure, where boids are grouped into grid cells for faster range/distance queries.

Usage

The simulation runs in realtime, and can be interacted with in various ways through the UI:

Figure 2: The large control panel.

Boids come in four colors; boids of a given color will only flock with other boids of the same color, though all boids will try to separate themselves from other boids (to prevent overlaps). Based on the click mode, clicking on the simulation will spawn boids, obstacles, goals, or predators; boids will interact with the mouse depending on the mouse mode (either ignore it, follow it, or avoid it). The obstacle radius allows setting how large a placed obstacle (which is a circle) should be.

The remainder of the controls correspond to the various forces and parameters of the simulation: how fast boids go, how large of a view radius they have (boids only react to flockmates within their view radius), which forces are acting on them, and so on. Predators, whose parameters are at the bottom, are special boids which eat normal boids they come in contact with (and which normal boids actively try to run away from).

Several example images of various boid simulations are included in Figures 3, 4, and 5; to get a decent sense of their behavior, we recommend enabling/disabling core rules to see how flocking behavior changes, as well as inserting obstacles and predators to show how flocks deal with forced dispersals and obstacles.

3 Limitations & Extension Work

We are not aware of any serious bugs at the time of writing this report (though they may certainly still be serious bugs we are *unaware of*!). The biggest limitation of our implementation is the small arena in which the simulation takes place: we do not support arenas larger than the size of the screen, nor do we support the necessary camera controls (zooming/panning) for interacting with such larger arenas. Performance is acceptable up to thousands of boids (i.e., if the arena is almost completely filled with them) thanks to our grid-based spatial data structure, so you should not notice notable performance degredation except in cases where many hundreds of boids are concentrated in a small area.

We spent a decent amount of time tuning the boid simulation parameters to make nice flocking behaviors, but you may find that the boids are somewhat brittle to changes in the strength of various rules - this is unavoidable, and you may need to toy with the other rule strengths to re-obtain flocking behavior.

Our collision avoidance is quite crude - it just forces boids directly away from the obstacle, which may overly correct the boid's trajectory. An interesting alternative we did not have time to explore is to instead just adjust the boid's velocity enough so that it's new trajectory does not collide with the obstacle; this creates smoother boid motion around obstacles and walls.

4 Acknowledgements

Thanks to Dr. Vouga and Xinya for an interesting class and good semester; Dr. Vouga's lectures are some of the best in the CS department.

References

[1] Craig W. Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH '87). Association for Computing Machinery, New York, NY, USA, 25–34. DOI:https://doi.org/10.1145/37401.37406

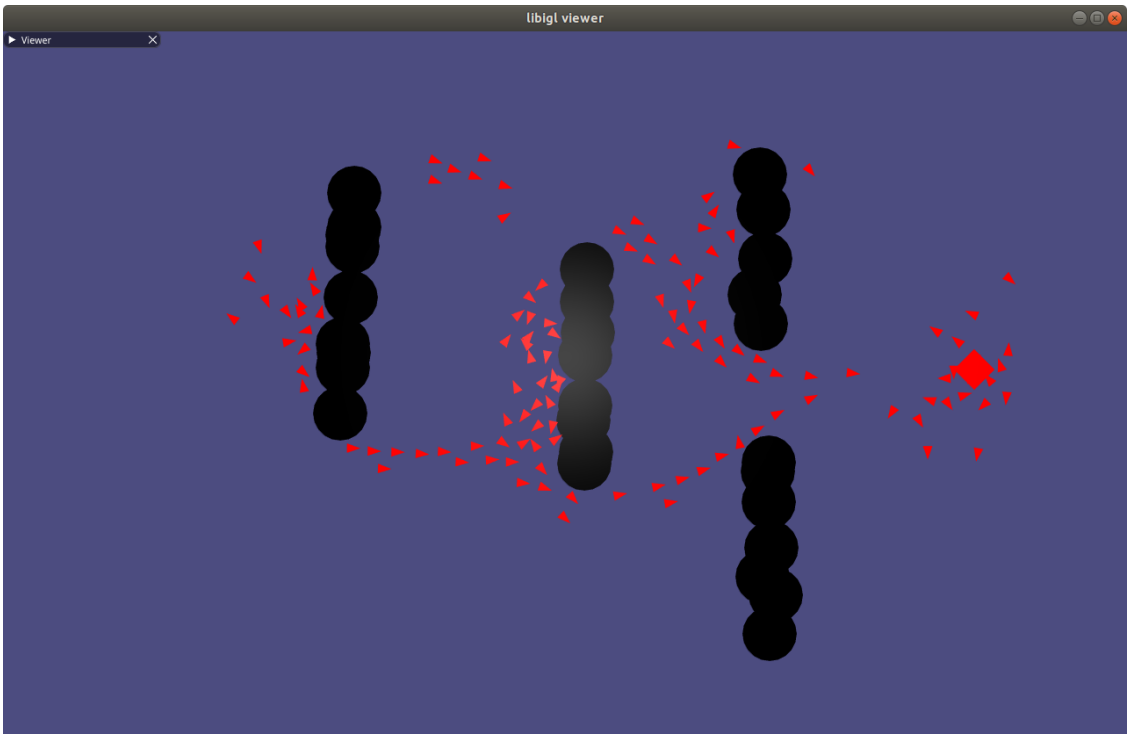


Figure 3: Boids with collision avoidance can move around obstacles towards a goal.

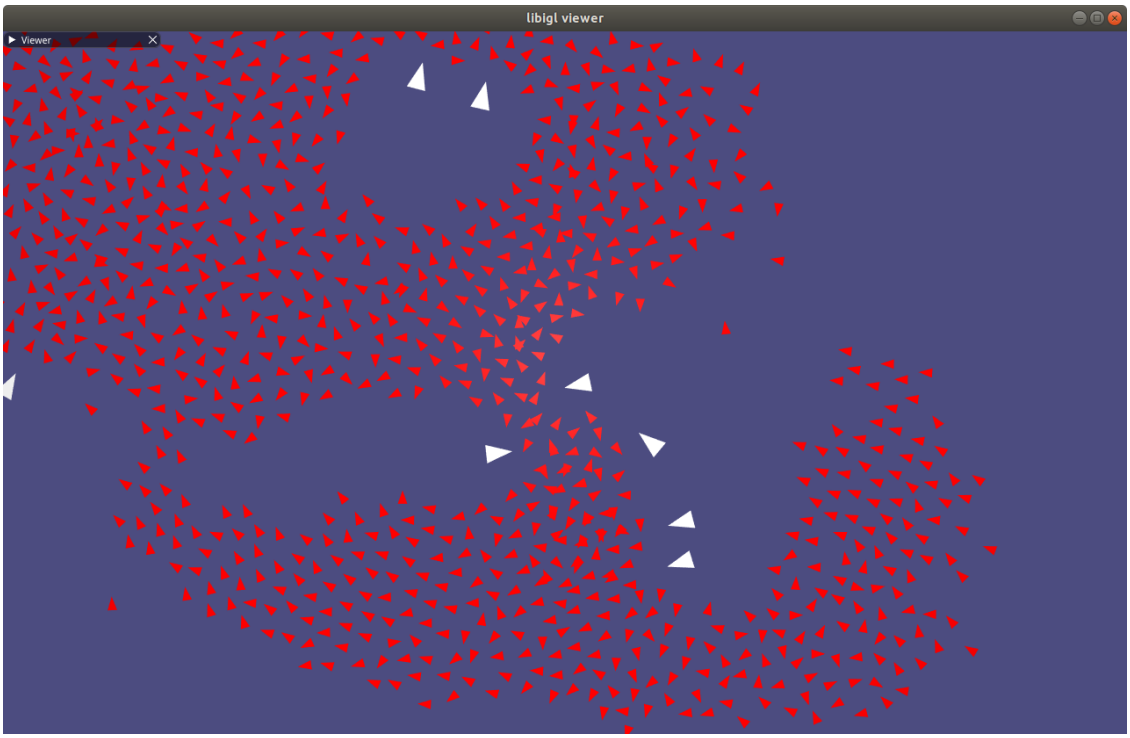


Figure 4: Boid flocks will disperse due to predators.

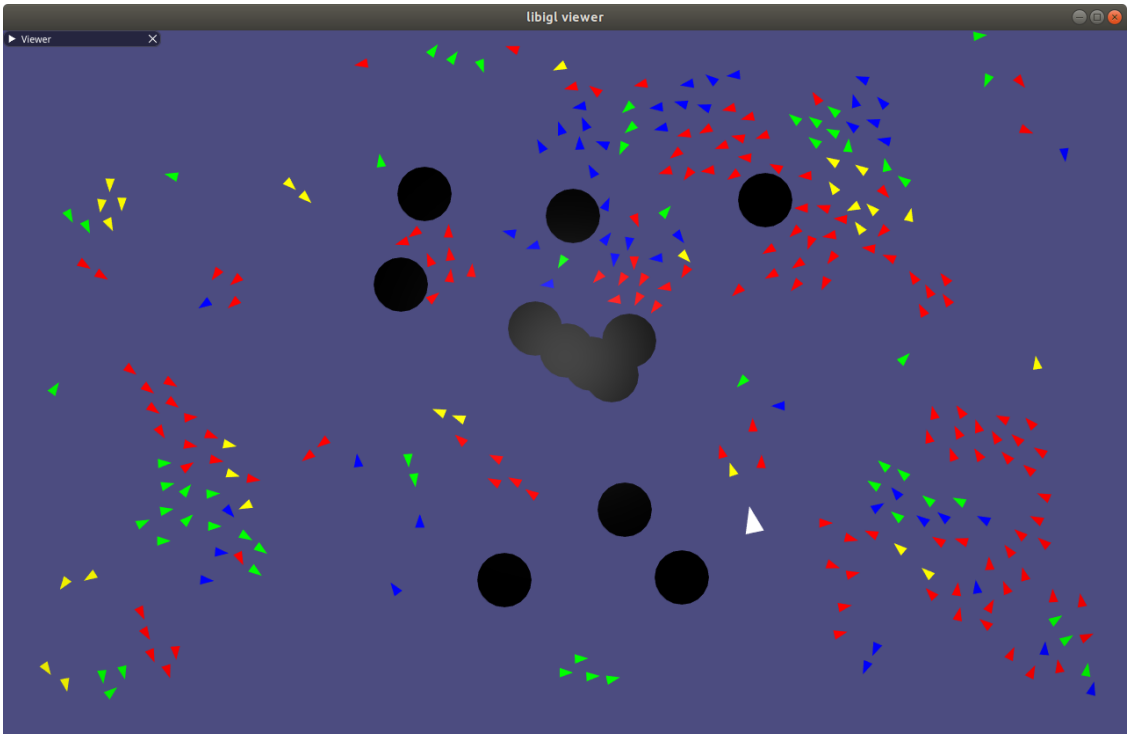


Figure 5: Lots of boids moveing around the screen, avoiding obstacles.