UCSD CSE 167 Assignment 5:

Animation: Boids

In this homework, we will explore some simple computer animation. In particular, we will implement a classical crowd animation algorithm Boids, invented by Craig Reynolds in 1987. The Boids algorithm models the flocking behavior of birds to create collective behaviors. While simple, it presents a breakthrough in Artificial Intelligence in the 80s by showing that modeling simple individual behaviors can form complex systems. It is used in various films for their crowd animation and is also the basis of modern crowd animation algorithms. Furthermore, its algorithm structure is similar to many other particle-based physical simulation algorithms (e.g., Smoothed Particle Hydrodynamics), making it suitable for learning computer animation. The explanation below is mostly based on the article written by Conrad Parker.

The Boids algorithm. The Boids algorithm models birds as a list of particles. Each particle has two attributes: its position, and its velocity. At each time step, we modify the velocity of the particles by applying forces. (Recall that $F = ma$, so $\dot{v} = \frac{F}{m}$. We pretend we are good physicists and set $m = 1$.) The forces are defined by a set of rules: the birds want to avoid hitting each other, but also want to fly together. The pseudo code looks like this:

```
boids = initialize_position_and_velocity()
while True:
  for boid in boids:
    F = ... # compute force based on the neighbor boids
    # update velocity using force
    boid.velocity += F
    # update position using velocity
    boid.position += boid.velocity

  draw_boids()
```

Without going into too much detail, the update scheme above is called the Semi-implicit Euler method, or the sympletic Euler method – notice that we use the updated velocity to update position. This tends to lead to better results compared to updating position first then the velocity. We might talk about it in the class, but CSE 169 and CSE 291 are the right classes to learn more about it.

Now the question is how to compute the force $\mathbf{F}$. We model $\mathbf{F}$ based on the following four principles:

1. **Seperation**: boids steer away from nearby boids.
2. **Cohesion**: boids steer toward the center of nearby boids.
3. **Alignment**: boids match velocity of nearby boids.
4. **Boundary**: boids steer away from the domain boundary.

Below we assume a 2D Boids model, but 3D is similar.

More specifically, the force $\mathbf{F}$ is the sum of the four separate forces:

$$\mathbf{F} = \alpha_s \mathbf{F}_s + \alpha_c \mathbf{F}_c + \alpha_a \mathbf{F}_a + \alpha_b \mathbf{F}_b, \tag{1}$$

where s stands for separation, c stands for cohesion, a stands for alignment, and b stands for boundary. α_s, α_c, α_a, and α_b are the scalar parameters controlling the magnitude of the forces. Below we define each of the forces $\mathbf{F}_s$, $\mathbf{F}_c$, $\mathbf{F}_a$, and $\mathbf{F}_b$.

The separation force $\mathbf{F}_s$ is a force that drives away from each very close boid:

$$\mathbf{F}_s = \frac{1}{N_s} \sum_{i \neq j}^{N} \left[\|\mathbf{p}_j - \mathbf{p}_i\| < d_s \right] (\mathbf{p}_j - \mathbf{p}_i), \tag{2}$$

where N is the total number of boids, j is the index of the current boid which we are applying the force to, $\mathbf{p}_j$ is its position, i is the index of the nearby boid, $\mathbf{p}_i$ is its position, $[x] = x\ ?\ 1 : 0$ is an indicator function, and N_s is the number of nearby boids within the distance threshold d_s, excluding boid j. Notice how $\mathbf{F}_s$ drives the boid j away from all neighbors i.

The cohesion force $\mathbf{F}_c$ is a force that makes boids cluster together:

$$\mathbf{F}_c = \bar{\mathbf{p}} - \mathbf{p}_j, \tag{3}$$

where $\bar{\mathbf{p}}$ is the average position of nearby boids excluding the boid j, defined using the distance threshold d_v and the number of nearby (visible) boids N_v within the distance threshold:

$$\bar{\mathbf{p}} = \frac{1}{N_v} \sum_{i \neq j} \left[\|\mathbf{p}_j - \mathbf{p}_i\| < d_v \right] \mathbf{p}_i. \tag{4}$$

Note that N_v does not include the boid j.

Next, the alginment force $\mathbf{F}_a$ makes boids fly in the same direction:

$$\mathbf{F}_a = \bar{\mathbf{v}} - \mathbf{v}_j, \tag{5}$$

where $\bar{\mathbf{v}}$ is the average velocity of nearby boids excluding the boid j:

$$\bar{\mathbf{v}} = \frac{1}{N_v} \sum_{i \neq j} \left[\|\mathbf{p}_j - \mathbf{p}_i\| < d_v \right] \mathbf{v}_i. \tag{6}$$

Notice that there are two different distance thresholds: d_s and d_v. The separation distance threshold d_s represents the "safety distance" between boids where they would start avoiding each other. The visible distance threshold d_v represents the "perceptual distance" between boids for modeling the neighbor boids each one sees.

The boundary force $\mathbf{F}_b$ makes boids turn around when flying towards the domain boundary:

$$\mathbf{F}_b = \left([\mathbf{p}_j.x < d_{\text{left}}] - [\mathbf{p}_j.x > d_{\text{right}}], [\mathbf{p}_j.y < d_{\text{top}}] - [\mathbf{p}_j.y > d_{\text{bottom}}] \right). \tag{7}$$

We are almost done. As a last step, let's impose a maximum and minimum speed $s_{\max}$ and $s_{\min}$ to the boids after we update the velocity so that they don't fly too fast or too slow:

```
if length(boid.velocity) > 0.0:
    if length(boid.velocity) > s_max:
        vel = vel * (s_max/length(vel))
    if length(boid.velocity) < s_min:
        vel = vel * (s_min/length(vel))
```

As you can see, there are many parameters of this algorithm. This is common for computer animation algorithms. Setting parameters is usually one of the most painful part of implementing computer animation algorithms (open research topic!). Here I provide the parameters I used in my implementation, but feel free to change them to the ones you like more:

$$\begin{aligned}
s_{\max} &= 10.0 \\
s_{\min} &= 5.0 \\
d_s &= 30.0 \\
\alpha_s &= 0.02 \\
d_v &= 150.0 \\
\alpha_a &= 0.03 \\
\alpha_c &= 0.001 \\
d_{\text{left}} &= 200.0 \\
d_{\text{right}} &= \text{resolution.x} - 200.0 \\
d_{\text{top}} &= 160.0 \\
d_{\text{bottom}} &= \text{resolution.y} - 160.0 \\
\alpha_b &= 0.5
\end{aligned} \tag{8}$$

Shadertoy For this homework, instead of balboa, we will implement in a very cool online service called Shadertoy. In Shadertoy, all you have access to are a bunch of fragment shaders that draw to a quad. It is however a surprisingly powerful programming model that can be used to rendering a wide variety of beautiful images. We will use two fragment shaders to implement Boids: the first fragment shader takes a 4-channel texture as input (2D position and 2D velocity) and outputs a 4-channel texture to update the position and velocity. The second fragment shader takes the output from the first fragment shader and draw the boids as circles. We have set this up for you in this link. The texture setting is: for Image, `iChannel0` is set to Buffer A with `nearest` filter and `clamp` warping. For Buffer A, `iChannel0` is set to Buffer A with `nearest` filter and `clamp` warping, and `iChannel1` is set to RGBA Noise Medium in Textures with `linear` filter and `repeat` warping.

To submit, send us a link to your own Shadertoy implementation of Boids (you might need to register). For reference, I have recorded my own implementation in this link.

Bonus: 3D Boids (20 pts). Modify the algorithm above to 3D. You might need to output position and velocity in different location of the texture (say, position in the even location, and velocity in the odd location) since Shadertoy only allows for 4D outputs.

Bonus: User Interaction (15 pts). Add user interaction to the algorithm above. For example, you can add a force to guide where the boids should go to by clicking on a screen position.

Bonus: Predators (15 pts). Make the Boids flee from a randomly moving predator by adding an extra force.