Orbs Particle System

Michael Brent Dimapilis

California Polytechnic State University, San Luis Obispo

Orbs Particle System

Particle systems are always usually found in modern video games. They are used to simulate visual effects, which include explosions, moving water, and magical spells. Particle systems are generally popular to implement because individual particles are easier to implement when they are all contained together and follow a set logic. An example of a particle system is the orbs particle system that follows a trailing behavior.

Lume implements a particle system to reflect the behavior of "orbs energy". The algorithm described in this paper works off of Craig Reynolds' boids algorithm with adjustments particular to the game. The behavior of the particles (hereinafter known as "orbs") within this system has gone through many adjustments in order to catch the desired behavior wanted for the game. The process of reaching this desired behavior and the final algorithm used in the game is described in this report.

Using The Boids Algorithm

The particle system implemented in this project initially followed the boids algorithm. The boids algorithm, developed by Craig Reynolds in 1986, is an artificial life program that imitates the flocking behavior of birds. The simulated flock is an elaboration of a particle system, with the simulated birds being the particles [http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/]. Basically, the boids algorithm uses three important heuristics to determine how the particles will be translated in 3D space: particles translate towards the center of mass of neighboring particles, particles keep a certain distance from other particles, and particles try to match the velocity of nearby particles. The initial version of the orbs particle system used the boids algorithm,

adjusted with the tendency to move towards the character's position. **Figure 1.1** illustrates the three different rules that the boids algorithm follows.

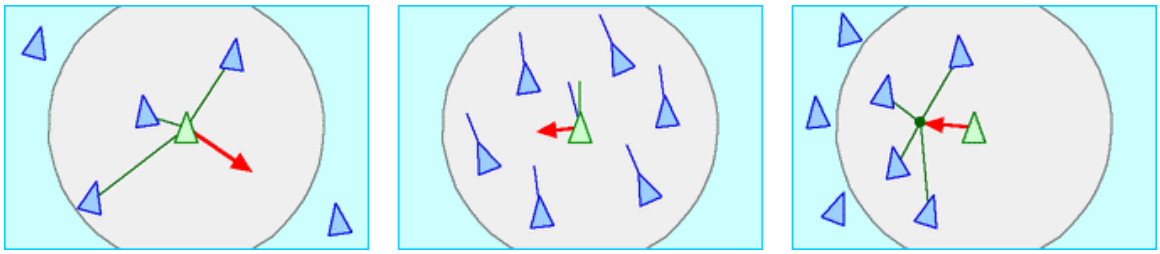

Figure 1.1: Main heuristics of boids algorithm: separation, alignment, and cohesion

Adjustments For The Game

After implementing the orbs particle system to use the boids algorithm, multiple orbs dispersed from a single energy source in different directions with the tendency to translate towards the character. Because all the orbs followed a single target and moved in different directions, at certain times, the orbs would fill most of the screen space, making it difficult for the user to see past a group of orbs. Limiting the number of orbs emitted from an energy source to a low number was not considered a solution to this problem because the game was to illustrate the character absorbing great amount of energy. Allowing a high number of orbs to be released from an energy source while recognizing the user's visibility was essential towards the game. The way to achieve this was for the orbs to translate towards the character in a trail-like manner.

Algorithm

The algorithm described illustrates the behavior of the orbs after they have been activated and made visible.

The Orb Data Structure

The orb takes advantage of a 3D point data structure in order to hold information on the orb's current position and velocity. To create a trail-like behavior, the orb needs only to translate towards the orb ahead of it in the trail as oppose to the character. Therefore, a pointer to another orb lies inside the orb's data structure to represent the orb ahead of itself in the trail. Only for the first orb in the trail does it translate towards the character.

Velocity Calculation

Previously, the velocity of an orb was calculated using the three mentioned heuristics above: collision avoidance, alignment, and cohesion along with velocity limitation and tendency. For the purposes of the game, it currently only follows collision avoidance along with velocity limitation and tendency. The collision avoidance method and the tendency method read in the orb data in order to calculate the right velocity. **Figure 1.2** shows how the velocity is calculated from `pnt3d Source::collisionAvoidance(Orb o)`.

```
pnt3d Source::collisionAvoidance(Orb o) {
   pnt3d c;

   for(int i = 0; i < _orbcount; i++) {
      if(_orbList[i]._active && _orbList[i]._orbId != o._orbId) {
         if(_orbList[i]._pos.distBetween(o._pos) < 0.5) {
            if(_orbList[i]._pos.distBetween(_pos) < 200) {
               c = c - (_orbList[i]._pos - o._pos);
            }
         }
      }
   }

   return c;
}
```

Figure 1.2: Collision avoidance method

The resulting velocities from `pnt3d Source::collisionAvoidance(orb o)` and `pnt3d Source::tendToPlace(Orb o, pnt3d dest, float force)` are then added together and passed to `pnt3d pnt3d Source::limit_velocity(Orb o)` to reduce the resulting velocity if it is larger than a specified value. Finally, the resulting velocity is added to the current position of the orb in order to do the translation.

The interesting part of this algorithm is `pnt3d Source::tendToPlace(Orb o, pnt3d dest, float force)`. Every orb that isn't in the front of the trail holds a pointer to the orb in front of it and is passed to this method. Because the orb in front is translated towards the character, every orb behind it follows naturally, producing a trail-like, fluid behavior. **Figure 1.3** shows how the trail appears in the game.

Figure 1.3: Orb trails in the game Lume

Ending The Trail

When the front orb reaches a certain distance from the character, it is set to inactive and no longer drawn. The orbs behind follow the same procedure, except referencing the orb ahead instead of the character. In order to prevent an orb from ending too soon, the orb is only set to be inactive if the orb ahead is inactive. The energy source determines when the trail has completed when the last orb in the trail has been set to inactive.

In order to allow the orbs to rotate and flow around the character, a time interval was used on the leading orb during the transition to its inactive state. Upon getting within the specified distance to become inactive, the leading orb was not to be set inactive until the time interval expired. Implementing this, orbs flowed around the character rather than simply disappearing.

Results

The algorithm described results in a smooth, fluid, and trail-like movement for the orbs used in the game. **Figure 1.4** lists more images of the orbs particle system evident in Lume.

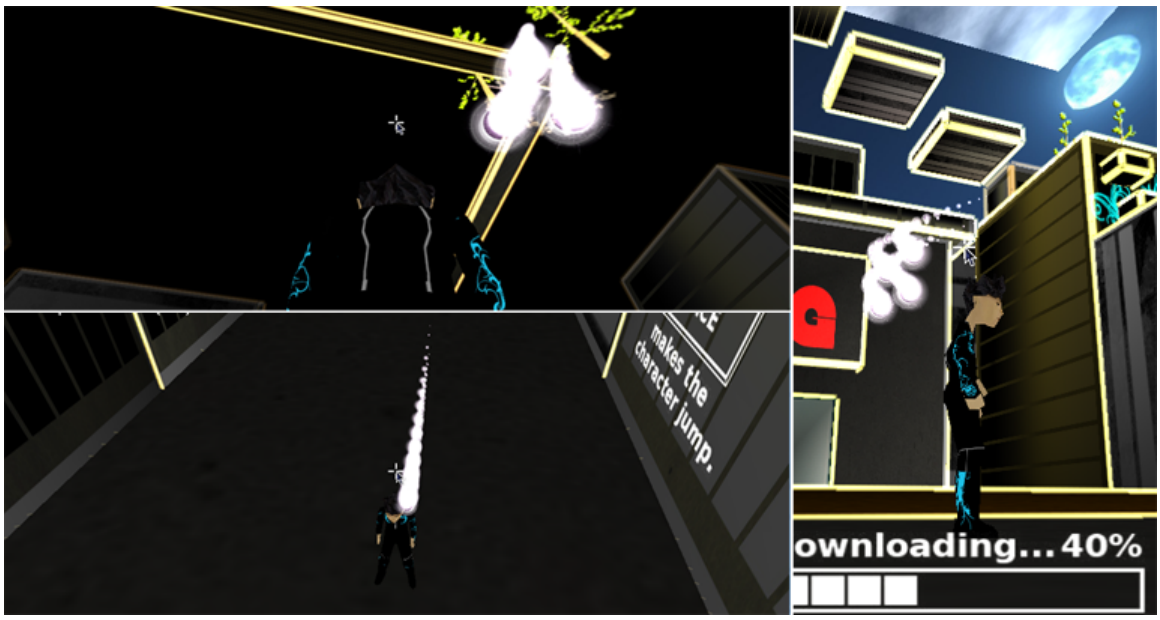

Figure 1.4: Orbs particle system in Lume

References

1. Reynolds, Craig W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/