Intro to particle systems

In 1982, William T. Reeves, a researcher at Lucasfilm Ltd., was working on the film *Star Trek II: The Wrath of Khan*. Much of the movie revolves around the Genesis Device, a torpedo that when shot at a barren, lifeless planet has the ability to reorganize matter and create a habitable world for colonization. During the sequence, a wall of fire ripples over the planet while it is being “terraformed":

The term ***particle system***, an incredibly common and useful technique in computer graphics, was coined in the creation of this particular effect.

“A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.” —William Reeves, "Particle Systems—A Technique for Modelling a Class of Fuzzy Objects," ACM Transactions on Graphics 2:2 (April 1983), 92.

Since the early 1980s, particle systems have been used in countless video games, animations, digital art pieces, and installations to model various irregular types of natural phenomena, such as fire, smoke, waterfalls, fog, grass, bubbles, and so on.

This section will be dedicated to looking at implementation strategies for programming a particle system. How do we organize our code? Where do we store information related to individual particles versus information related to the system as a whole? The examples we’ll look at will focus on managing the data associated with a particle system. They’ll use simple shapes for the particles and apply only the most basic behaviour’s (such as gravity). However, by using this framework and building in more

interesting ways to render the particles and compute behaviour’s, you can achieve a variety of effects.

As the quote described above, a particle system is a collection of simple objects. We've already dealt with programming collections of objects before - like arrays of Movers that simulate bouncing balls. But for particle systems, our collections are more complex. The collections will range in size: sometimes there will be zero particles, sometimes ten, sometimes ten thousand. The collections themselves will have behaviour and properties, not just the particles they're made up of. Our goal is to be able to write a program that looks like this:

```
var ps = new ParticleSystem();


draw = function() {
    background(255, 255, 255);
    ps.run();
};
```

No single particle is referenced in that code, yet the result will be full of particles flying all over the screen. We'll create programs with multiple object types, and objects that keep track of other collections of objects, which will help us create powerful particle systems but also prepare us to write more powerful programs generally.

In figuring out how to program particle systems, we'll use two advanced object-oriented programming techniques: ***inheritance*** and ***polymorphism***. With the examples we’ve seen up until now, we’ve always had an array of a single type of object, like "movers" or “oscillators.” With inheritance (and polymorphism), we’ll learn a convenient way to store a single array that contains objects of different types. This way, a particle system need not only be a system of a single type of particle.

We'll look at the most typical uses of particle systems in this section, but the fact that the particles in this chapter look or behave a certain way should not limit your imagination. Just because these particle systems tend to look sparkly, fly forward, and fall with gravity doesn’t mean that those are the characteristics yours should have. The focus here is really just how to keep track of a system of many elements. What those elements do and how those elements look is up to you.