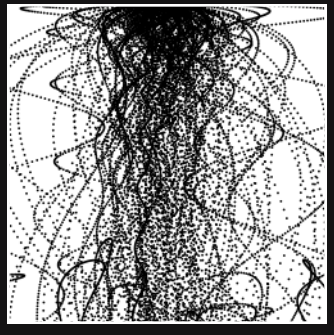
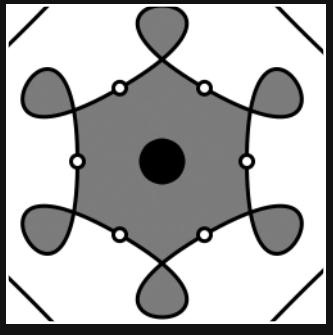
7 CURVES





Curvy stuff is difficult for people to understand. Had I taken a calculus class earlier in my life, this fact would have been blatantly obvious. When drawing directly from a model, edges are indistinct and there are all sorts of subtle foreshortening. In time, you learn how to, in a sense, "not draw" what's not there, which is often the secret to allowing the curve structures to emerge in the drawing. When it comes to coding, curves again prove especially challenging, and often require a more "mathy" approach than when working with lines. Processing does makes it relatively easy to generate simple curves, but being able to precisely control a combination of curves to create a form such as a human figure is very difficult.

That being said, the additional effort and added complexity of dealing with curves is well worth it, as there is something exciting about seeing organic forms and animation emerge; it's what really got me hooked on coding to begin with. In my classes, at the beginning of a semester I'll often show organic code-based animation examples to my students; this always gets them oohing and ahhing. (Of course, they make other sounds when they begin wrestling with the actual curve implementation and math.)

This chapter will explore curves in detail, beginning with how to make the transition between straight lines and curves. After that, I'll discuss creating curves using trig and polynomials, and then I'll explain Processing's curve functions in depth.

Making the transition from lines to curves

I began the last chapter with a discussion about points, and showed how a line is really just a continuous path of points. I eventually showed you how to utilize a more efficient system of terminal vertices to describe the beginning and ending points of lines. Processing's `line()` function takes four arguments (the x and y components of two points) and joins the points with a straight line. In beginning to think about curves, you need to consider how Processing connects the two points with a line, which you can then modify into a curve. As a review, here is a simple script to generate a vertical line utilizing a series of points, as shown in Figure 7-1:

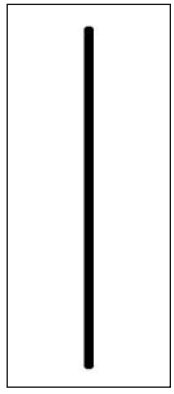


Figure 7-1. Combining points into a line

```
size(200, 200);
background(255);
int margin = height/15;
strokeWeight(5);
for (int i=margin; i<height-margin; i++){
  point(width/2, i);
}
```

Hopefully this looks very familiar. The `for` loop generates a vertical line in the middle of the display window from a series of points, beginning at `margin` and ending at `height` (of the display window) minus `margin`. You'll remember that `point()` is really a call to the `line()` function, using the same two points as arguments; that's why I was able to change the thickness of the points, and thus the line, with `strokeWeight(5)`. As an example, here's the same sketch using `line()`: