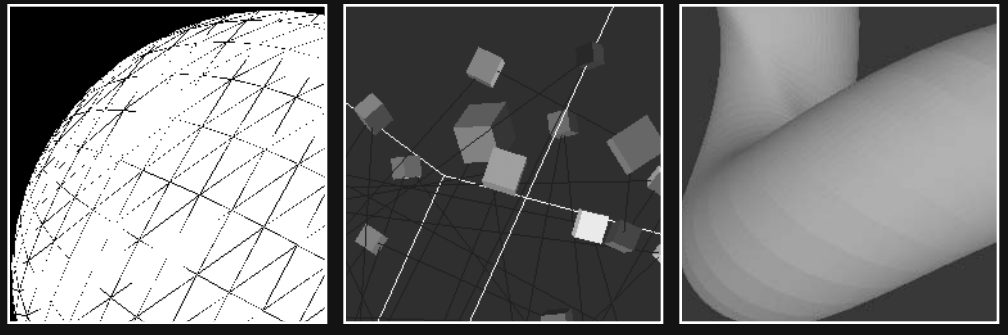
13 **3D**



The term *approach-avoidance* describes a psychological state in which people are attracted to and also repulsed by something. This tension aptly describes the relationship many of my past art students had with 3D animation. On the one hand, they were enamored by the cool 3D effects they watched in their favorite games and films. On the other hand, they became easily frustrated trying to learn the extremely dense and unintuitive software. Popular 3D modeling and animation applications such as LightWave, Maya, and 3ds Max (which handle the coding behind the scenes) are extremely complex, specialized pieces of software, presenting steep, drawn-out learning curves. Attempting to teach these same art students 3D programming would have been unthinkable. For this reason, coding 3D has been the domain of computer science types, requiring lots of scary math and very low-level programming—that is, until Processing came along. Processing has full 3D support and even includes two separate 3D renderers, and of course it's free. Most importantly, Processing greatly simplifies the process of coding 3D for creative folks, allowing us to begin "creating" in 3D almost immediately.

In this chapter, you'll learn about Processing's built-in 3D support, based on the custom P3D rendering engine. Working with some simple 3D functions, such as `box()`, you'll learn how to create that sexy spinning cube in no time. I'll revisit the concept of transformations, but in 3D space using Processing's `pushMatrix()` and `popMatrix()` functions. Then I'll go a little beneath the hood and teach you how to code your own 3D rotations as well as create some custom 3D geometry. First, I'll start with the basics.

Processing 3D basics

Here's the ubiquitous spinning cube I promised, using only nine lines of code (see Figure 13-1):

```
void setup(){
  size(400, 400, P3D);
}
void draw(){
  background(0);
  translate(width/2, height/2);
  rotateY(frameCount*PI/60);
  box(150, 150, 150);
}
```