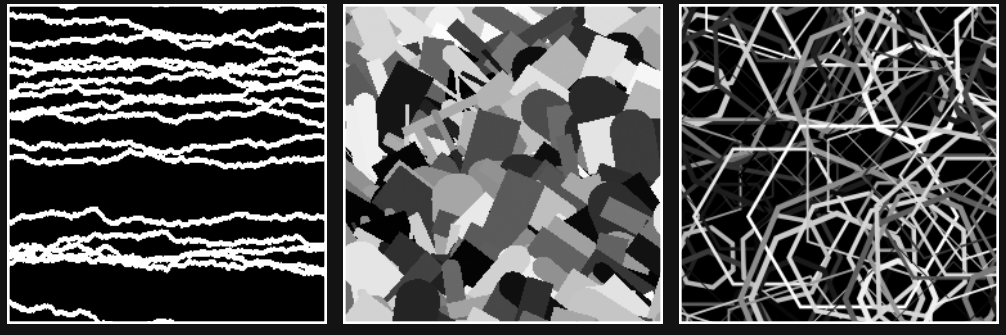
# 6 LINES



This chapter will concentrate on lines and how they are represented though code. You'll also look at how lines can represent more multidimensional space, such as shapes and volume.

# It's all about points

Before you can understand a line, you need to move back one dimension and take a look at points. Here is arguably the simplest graphic program you can write in any language (see Figure 6-1):

```
point(50, 50);
```

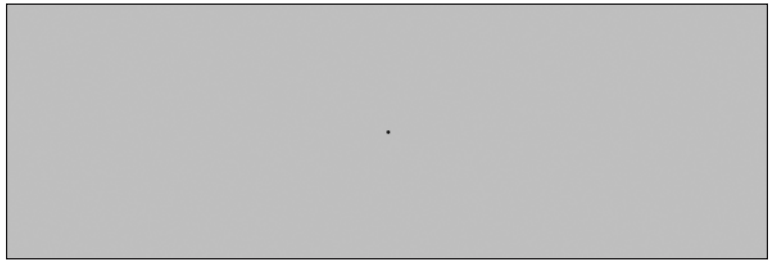


**Figure 6-1.** Single-point sketch

I mentioned before in the book that a point really has no dimension. Wikipedia says of points: "A point in Euclidean geometry has no size, orientation, or any other feature except position." Thus, your point, as output in Processing, is arguably a tiny square, with a width and height of a 1 pixel. However, what fun would it be to plot a point that didn't appear? In fact, your point is really a Processing line, with the same starting and ending points. The following line function call will yield the same result as the previous point function call:

```
line(50, 50, 50, 50);
```

However, this second approach is more complicated than necessary—it's certainly better to only have to pass two arguments instead of four—so stick with the `point()` function. Because `point(x, y)` is implemented internally with `line(x, y, x, y)`, if you want to change the color of a point, you need to use the `stroke()` function, not `fill()`.

There aren't many things you can do staring at a single static point, other than perhaps sitting lotus style and meditating on its oneness—which does have some value. However, enlightenment-seeking aside, let's add another point (see Figure 6-2):

```
point(33, 50);
point(66, 50);
```