4
Representing the Animation

Traditional 3D animation techniques require that several (and sometimes all) of the frames be stored [Wikipedia 2009], so that they can later be displayed sequentially during animation playback (see Figure 4.1).

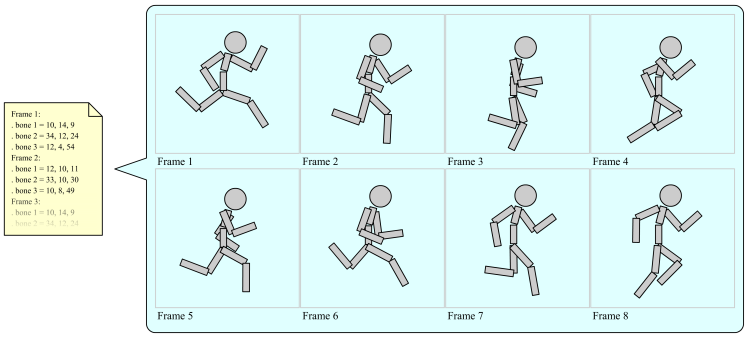

Figure 4.1: "Canned animations" are written in files that store the position and orientation of each bone in each frame.

More recent techniques rely instead on a set of rules that, when applied, create the animation's frames [Gritz and Hahn 1997]. Why, that is exactly what we are trying to achieve! The character moves as a result of the commands issued by its brain, which modify physical properties of its body, which is propelled forwards. In other words, the character's motion is *implicitly* described by its physical structure and its intelligence. Our system, then, will need the following data in order to play back the animation:

- A description of the character's body.
- A description of the character's brain.

Models for describing the character's body were covered in Chapter 3. This chapter will cover two models for describing the character's brain, enabling us to fully represent the character animation.

4.1 Sequence of Commands

Our first model is the simplest: A *sequence of instructions* that are issued one after the other. This is very similar to imperative programming languages (such as assembly, or machine code), so we will refer to these commands as *op-codes*. The animation can be considered a program that is executed one op-code at a time.

We decided to employ programs of variable length, that is, the number of op-codes in the animation is *not* pre-defined. Our research focuses on walking animations, so it is also necessary to loop the program — restarting from the first op-code when the end of the program is reached — to simulate an endless cycle.

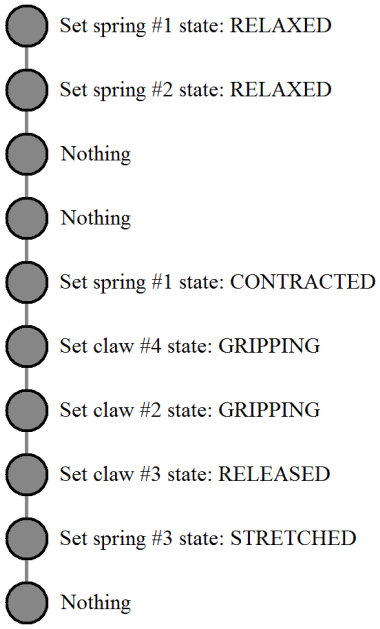

Figure 4.2: A short animation with ten op-codes. Op-codes marked with "Nothing" represent a short interval of time during which the character's physical parameters are not modified.

4.2 Expression Trees

Our second model is an implementation of the *genetic programming* paradigm, as described in [Koza 1992]. The program, in this case, is a tree-like data structure that represents a mathematical expression. Each leaf (or *terminal*) in the tree is a variable that "reads" a state of the character's body; each node in the tree is a unary, binary, or ternary operator. Evaluating the entire expression tree yields a real number, which is applied to one of the

character's muscles. In other words, the expression tree is a *controller* for the muscle, and each muscle has its own controller.

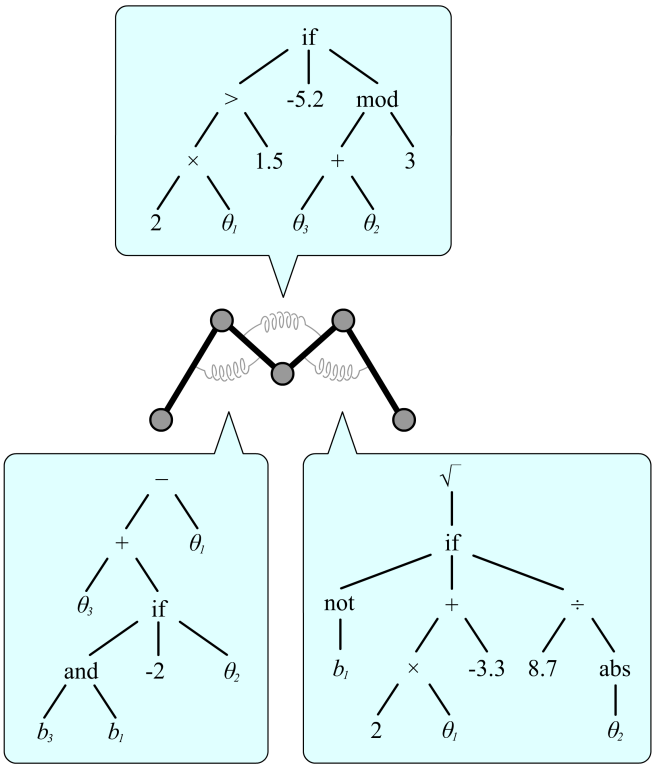

Figure 4.3: Each of the three muscles is controlled by an expression tree.

Figure 4.3 is a visual description of this structure. In that example, muscles are angular springs, the terminals b_1–b_4 are Boolean values indicating whether a bone is touching the floor, and the terminals θ_1–θ_3 are real numbers indicating each joint's current angle. The result of each expression tree is directly applied to an angular spring by setting its natural angle to the evaluated value.