Alexei Masterov: am2268
Natasha Shamis: ni2104
Joshua Poritz: jsp2104

26 Sep 2005
COMS W4111
Prof. Edwards

ANIMO: Character Animation Language

I. Overview

A. Introduction

ANIMO is a language that is designed to control an animated character. It will allow a programmer to compose instructions specifying a series of body movements, enabling a character to perform a variety of motions. ANIMO will have an easy-to-use syntax, combining numerous features characteristic of higher-level languages with unique functions of its own.

B. Applications

There are three major domains in which our language would likely be employed.

a. Computer Animation

Computer animation is used very widely in the movie industry. ANIMO may become a standard for animators and an alternative for visual interfaces because it allows the reuse of code while providing the flexibility by means of parameters.

b. Computer Games

Computer games and virtual environments is another domain where our language can be used by both programmers and players. In places like the Metaverse (http://en.wikipedia.org/wiki/Snow_Crash), users can "learn" to dance by writing an ANIMO program and use it to impress their friends. The creators of such environments can use ANIMO for animating crowds by re-using the same code with different parameters.

c. Robot Control

Finally, ANIMO can be used in a real world for operating toy robots. It should be relatively easy to port it to allow control for a humanoid robot that has a set of joints arranged in a hierarchical fashion.

C. BVH File Format

The BVH file format was originally developed by Biovision, a motion capture services company, as a way to provide motion capture data to their customers. The name BVH stands for Biovision hierarchical data.

It contains the hierarchical structure of joints and end effectors, the motion capture data as a list of figure coordinates in space, and a list of joint rotation angles over a sequence of frames.

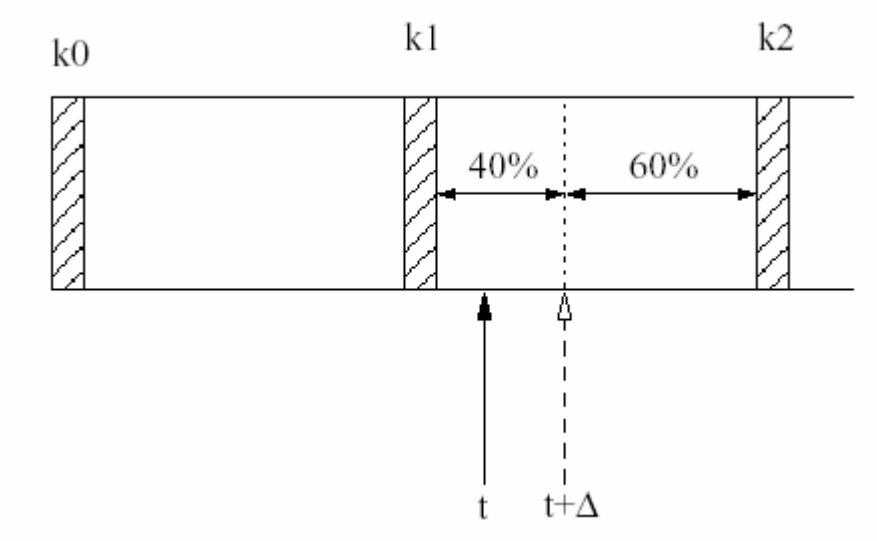
Sample BVH file:

```
HIERARCHY
ROOT Hips
{
 OFFSET 0.00 0.00 0.00
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
   JOINT Chest
   {
     OFFSET 0.00 5.21 0.00
     CHANNELS 3 Zrotation Xrotation Yrotation
       …
   }
}
MOTION
Frames: 2
Frame Time: 0.033333
8.03 35.01 88.36 -3.41 14.78 -164.35 13.09 40.30 -24.60 …
7.81 35.10 86.47 -3.78 12.94 -166.97 12.64 42.57 -22.34 …
```

The first three columns correspond to the X, Y, and Z coordinate of the root joint position in space, and the rest of the columns define the local rotation angle of each joint in the hierarchy along the X, Y, and Z-axis. Joints are rigidly attached; therefore, they can only rotate, and not move, in space.

If one rotates the joint in the hierarchy, all the joints in the hierarchy below get rotated with it:

The “Frame time” parameter specifies the time interval between frames. The rendering software then interpolates all the joint positions in between frames.



So how do we make the figure move in a consistent fashion? The hierarchy in the BVH file specifies the initial position of the figure (k0). Then each row specifies what will happen to the figure at the next time interval (k1, k2, …, kN). Each movement is a combination of joint rotations. In our language, we are going to allow the user to specify each joint rotation individually, and combine them using the “+” sign. This operation of combining different movements is called “blending”. By blending different joint rotations, the user can make the figure perform arbitrary complex movements through space.

D. Compiler: Inputs and Outputs

Motion will be simulated through the use of BVH files, which contain hierarchical information for the skeletal structure of the animated characters. Our compiler will accept an ANIMO file and a BVH file containing the character’s initial position, and output a new BVH file containing a set of instructions in the form of joint movements for the character.

II. Language Elements

A. Datatypes, Operators, Loops, and Conditionals

We’ll need conditionals such as “if” and loops such as “for”. Since all of our data represents joint rotations, we will limit our datatypes to doubles and have addition, subtraction, division, multiplication, and modulus operators defined for them with standard (mathematical) precedence.

B. Basic Commands: rotate() and move()

The most basic commands in ANIMO's default library will be:

rotate(joint, x, y, z);

which will map nicely to the 3 columns that correspond to a given joint rotation. To combine multiple joint rotations, the user will use the following syntax:

rotate(joint1, x1, y1, z1)+rotate(joint2, x2, y2, z2);

and

move (x, y);

which will map to the figure's position in space.

"rotate" and "move" comprise ANIMO's smallest building blocks. A programmer may combine these fundamental motions to create more complex movements.

C. Blending

One may say that "rotate" and "move" are such primitive commands that they provide no ostensible benefit over editing BVH files directly. Our compiler, however, provides the capability of blending several movements together to create fairly complex concurrent motions. These more complex motions, in turn, may be blended to form even more sophisticated character movements.

D. User-Defined Functions

Functions are fundamental to the purpose of our language. A function encapsulates a series of concurrent and sequential movements into a named block, such as kneel(), which may be called by any other function that seeks to perform the motion with that name. A program to make a skeleton jump, for example, may contain a main routine consisting solely of four function calls in succession — kneel(), straighten(), elevate(), and descend() — which results in a much more readable program than several dozen unnamed lines specifying the same joint transformations. Functions may take one or more parameters to control their behavior. kneel(), for example, might take an integer value from 0 to 1 specifying how steeply the skeleton in question will kneel.

III. Examples

A. Sample User-Defined Functions

Below is an example of a function that would move a character's left leg:

```
left_leg(..)
{
	rotate(LeftHip, ..)+rotate(LeftKnee, ..)+rotate(LeftAnkle, ..)+move(..);
	rotate(LeftHip, ..)+rotate(LeftKnee, ..)+rotate(LeftAnkle, ..)+move(..);
	rotate(LeftHip, ..)+rotate(LeftKnee, ..)+rotate(LeftAnkle, ..)+move(..);
	rotate(LeftHip, ..)+rotate(LeftKnee, ..)+rotate(LeftAnkle, ..)+move(..);
}
```

Or, equivalently:

```
left_leg(..)
{
	for (I = 0 to 4)
		rotate(LeftHip, ..)+rotate(LeftKnee, ..)+rotate(LeftAnkle, ..)+move(..);
}
```

By blending different functions together, users may create more complex movements:

```
step_right(..)
{
	left_hand(..)+right_leg(..)+move(x, y);
}

step_left(..)
{
	right_hand(..)+left_leg(..)+move(x, y);
}
```

B. **Sample Program**

```
march()
{
	for (I = 0 to 10)
	{
		if (I % 2 = 0)
			if (I % 4 = 0)
				step_right(..) + wave_right();
			else
				step_right(..);
		else
			step_left(..);
	}
}
```

REFERENCES:

[1] http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

[2] “Design and implementation of real-time character animation library”. Hakan Almer, Eric Erlandson.