Team SoundHammer: Language Proposal for Atomic Sound Manipulation Language (ASML)

Frank A. Smith
fas2114@columbia.edu

Tim Favorite
tuf1@columbia.edu

Introduction

Atomic Sound Manipulation Language is Team SoundHammer's solution to the problem of how to make a high level computer language that describes effects that can be applied to sound waves. The idea is to give the programmer access to the most elemental levels of a sound's composition in an easy, intuitive way. This elemental access to sounds is the idea from which our language derives its name, as we intended the language to operate upon sound at the "atomic" level.

ASML will feature typical computer language constructs such as loops, if statements, comparators, and arithmetic operators. It will use these features to manipulate new primitive types based upon the most fundamental components of sound: frequency, amplitude, and time. Programmers in ASML will be able to use these statements and operators to produce general purpose and specialized functions to act upon sound waves, their individual frequencies, and other available types. Programmers will then be able to aggregate these functions into even more robust effects that can be intelligently triggered.

Team SoundHammer would like ASML to bring power and ease to the manipulation of sound waves in the same way that a language like AWK makes it easy to write powerful text file manipulation programs. We would like to have the user be able to use simple commands to open wav files, isolate frequencies and time ranges from those files, and manipulate the data therein on either a micro or a macro scale.

Functional Aspects

Functionality

Any real world sound is, in essence, a sum of the amplitudes of the range of frequencies audible to the human ear (20 Hz - 20 kHz). Each frequency is represented by a simple sine wave with a unique wavelength. When a dog barks, or a violin plays, many frequencies are emitted, each at different amplitudes, and the combination of these (called a complex wave) is what gives the sound its unique character.

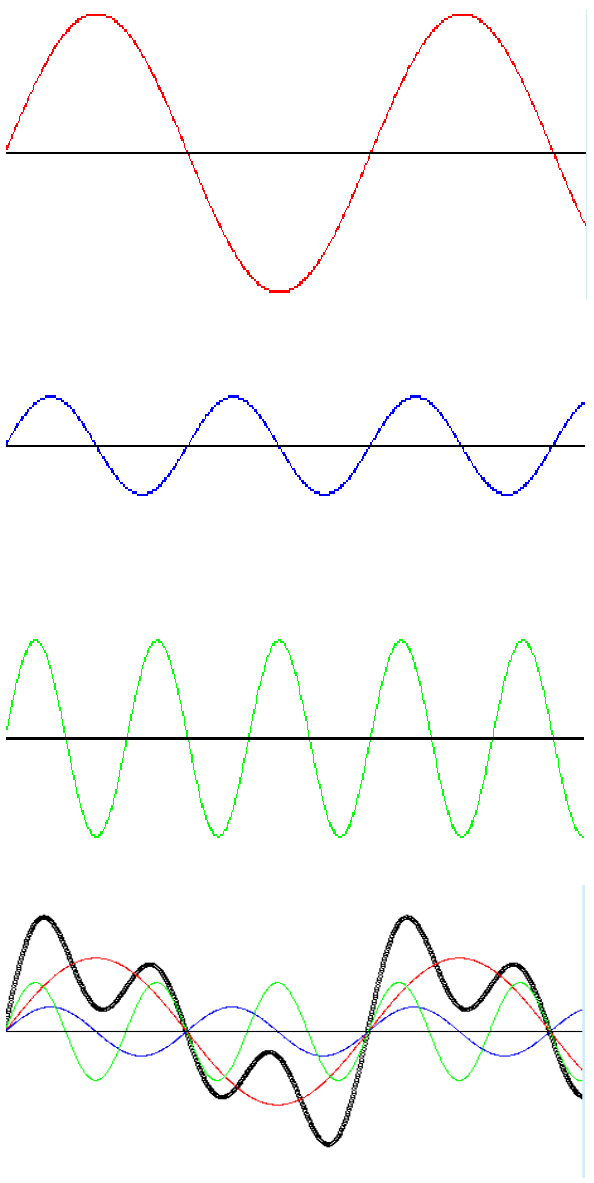

Figure 1: Three simple waves followed by a complex wave made up of the sum of the three simple waves (in black) (obtained from http://www.umanitoba.ca/faculties/arts/linguistics/russell/138/sec4/specgraf.htm)

So to work with a sound at the most basic level and perform functions such as pitch shifting or high-pass filtering, we must be able to pull out individual frequencies or ranges of frequencies from the complex wave of an input sound, and manipulate them. We'll represent input sound as an indexed list of frequencies, and users will be able to access an individual frequency as in the following example, where we are accessing the 440 Hz frequency:

```
wave f440 = get wave from input at 440hz;
```

Users will also be able to access a given sample time from the input sound, as in the following example, where we access a 1 second sample starting at 5 seconds into the input:

```
wave sec5 = get wave from input at 5000ms to 6000ms;
```

The "wave" type can represent simple waves (one frequency) or complex waves (many frequencies). All waves will be considered 2-dimensional matrices consisting of amplitudes, and indexed by frequency and time. For simple waves, all other frequencies besides the frequency represented by the simple wave will be considered to have an amplitude of zero at all times.

Major Characteristics

ASML will be implemented as a functional language. The program will accept wav files as input and move the input through a series of functions in order to accomplish the desired effects upon the sounds. We may stray from the notion of a purely stateless language where it allows for better ease of programming.

We will also be implementing ASML as a strongly typed language. While we will not be allowing users to create user defined types, we would also like to ensure against users doing actions that do not make sense within the context of sound manipulation. There is no reason for a user to write an equation for an amplitude as a function of a frequency, so we should prevent a user from doing this.

- For example:
    - freq f = 10000hz;
    - amp a = f/16;

Such an equation does not make sense from the perspective of sound manipulation because frequency and amplitude are measurements of two collaborating but unrelated characteristics of sound. What's more significant with respect to why we are making this a strongly typed language is that such a construction is not really useful. The numbers applying to these two different characteristics are on different scales and magnitudes and simply do not play well together for such a function, so we intend to make the program more strict in this respect. However, in case sound designers come up with ideas that we haven't thought of yet, we will try to incorporate a casting system between primitives that are based on numerals (e.g. frequency and time).

ASML will also be implemented with lazy evaluation of arguments in mind. The current rationale for this is the fact that sound engineering generally operates upon massive and repetitive data structures. If there is a situation where an argument in a function is only used in 40% of the calls made to the function, we would like to save that calculation. We may change our minds on this later depending on the difficulty of execution and the value of implementing it this way.

Another feature of the language will be the ability to create library files to store and share functions. It is beyond the scope of this project for Team SoundHammer to create more than just a few library functions, but we will create a small number for testing and demonstration purposes. This is a core feature of the language as it is what will allow programmers and sound designers to make large and interesting effects.

Although this paper has thus far focused on proposed new primitive types, the language will include such standbys as ints, floats, and strings as well. There is clear value in having arithmetic types in such a language, and we will allow new primitives to be coerced into these old types where that is logical.

- For example
    - amplitude time1 = get amplitude from wave1 at 500ms;
    - amplitude time2 = get amplitude from wave1 at 600ms;
    - float ratio = time1/time2;
    - set amplitude of wave1 at 500ms to ratio;

Furthermore, these types can be useful for purposes of debugging in the language, and this is a reason that we shall also try to include a console printing facility.
Finally, it is a goal of ours to try to implement a whole language syntax style that is closer to English than most existing high level languages. The small pieces of code so far are intended to represent our notion of how this could look and it is a standard that we will try to accomplish in our working version of the compiler.

Control Flow

Programs will always have at least one argument: an input file, specified by a -i in the command line. In the code, this is represented by the reserved word input. More than one input argument is possible, an output file, specified by a -o. Further arguments will be represented within the code by arg_0, arg_1, arg_2,...,arg_n for n additional arguments in the code. When the program is executed, it searches the code for a main function, and then executes all of the code inside of it. External function calls to functions defined in the same file or defined in an external library file included via an include statement are permitted. The main must end with a return statement that returns a data structure that can be written to a wav file, which will be written to the output file (if there is a -o argument) or back to the input file (if there is no -o).

An example program which applies reverb is below:

```
include "reverb.asml"

main
   double reverb_time = (double)arg_0;
   wave out = reverb(input, reverb_time);
   return out;

end main
```

Possible Applications

ASML allows for the creation of new sound effects by sound engineers in the realms of music, television and film, and telecommunications. Its support for libraries will make a standard set of effects readily available for sound engineers who are satisfied with the broad range of effects available in applications such as Pro Tools, but their open nature will allow for the potential of tweaking the effects as well, should that be desirable.

Challenges

Major challenges involve finding the tools in the Java Sound API to achieve the granularity desired in our language - it seems to have been designed with pre-set mixer functions already built in. Further research needs to be done to determine whether this API will be suitable or not. If it is the case that Java does not offer a suitable API, then it will be a major challenge for us to tackle the mathematical problems inherent in working with a sound based language. Naturally, as the focus of the class is not sound, we will probably only implement the features that require only the most casual knowledge of music theory and calculus.