CSC 314 Programming Project
Harp Heroine

Spring, 2015

Assignment Overview

You work for a game company that is developing a tiny portable game. Past experience shows that players will be less stressed, and sales will be higher, if the game plays soothing harp-like music in the background. It is also important that the players be allowed to select their background music from a list of at least 50 tunes. Storing the tunes as compressed audio files will not work, because even when compressed as MP3 files, they would require more storage space than is available on the ultra-low-priced game system.

Therefore, the decision is made to store the tunes in a very simple format which only specifies the notes to be played, and use the Karplus-Strong algorithm[1] to generate the waveform for the tune, in the background, in real-time. The generated waveform will be sent to the audio device. Your task is to demonstrate that this idea will work. You have access to some code which implements the Karplus-Strong algorithm. It reads a file which specifies a series of notes to be played, generates a temporary file holding the wave form as a sequence of floating point numbers, and then converts the temporary file into a Wave file.

You must modify the program so that it generates the Wave file directly, rather than using a temporary file, and writes a Wave file to standard output. Furthermore, your program must generate the tune in real-time using less than 5% of the CPU on a Raspberry Pi model B. You are competing against Steve, who got his degree at MIT and thinks he is just *so* much smarter than you. You will receive a bonus and (possibly a raise), for using significantly less CPU time and/or memory than Steve.

Background

When a string is plucked, the string vibrates and creates sound. The length of the string determines its fundamental frequency of vibration. We model a string by sampling its displacement (a real number between $-\frac{1}{2}$ and $+\frac{1}{2}$) at N equally spaced points, where N equals the sampling rate (typically 44,100 samples per second) divided by the fundamental frequency (rounded to the nearest integer).

[1]The Karplus-Strong algorithm played a seminal role in the emergence of physically modeled sound synthesis (where a physical description of a musical instrument is used to synthesize sound electronically).

Plucking the string

The initial excitation of the string (plucking) contains energy at many different frequencies. We simulate the excitation by filling the buffer with white noise. In other words, we set each of the N sample displacements to a random real number between $-\frac{1}{2}$ and $+\frac{1}{2}$.

The resulting vibrations

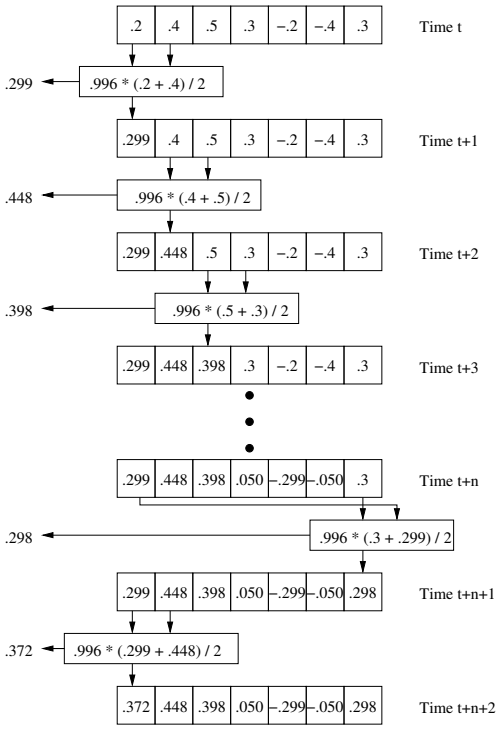
When the string is plucked, it begins to vibrate at many frequencies, but as time passes, the frequency of the string converges to the fundamental frequency of the string. The Karplus-Strong algorithm simulates the string vibration by maintaining a ring buffer of the N samples. The algorithm repeatedly deletes the next sample from the buffer and adds a new sample to the end of the buffer. The new sample is computed as the average of the first two samples, scaled by an energy decay factor of 0.996. The new sample is also used as the next value in the waveform generated by the string.

The fundamental frequency of the simulated string depends only on the size of the ring buffer, according to the following formula:

$$N = \frac{\tau}{\frac{2^{m-69}}{12.0} * 440}$$

where m is the MIDI number of the note to be played, and τ is the sample rate (typically 44100 samples per second).

The following figure illustrates how the Karplus-Strong algorithm is applied using a very small array as the ring buffer. Your program will need significatly larger arrays, but the method is the same.



Why it works

The two primary components that make the Karplus-Strong algorithm work are a ring buffer feedback mechanism and the averaging operation. The ring buffer models the medium (a string tied down at both ends) in which the energy travels back and forth. The length of the ring buffer determines the fundamental frequency of the resulting sound. Sonically, the feedback mechanism reinforces only the fundamental frequency and its harmonics (frequencies at integer multiples of the fundamental). The energy decay factor (0.996 in this case) models the slight dissipation in energy as the wave makes a round-trip through the string.

The averaging operation serves as a gentle low pass filter (which removes higher frequencies while allowing lower frequencies to pass, hence the name). Because it is in the path of the feedback, this has the effect of gradually attenuating the higher harmonics while keeping the lower ones, which is a reasonably good model for how plucked strings actually sound.

Program Specifications

The input file consists of a series of binary records. Each record has three fields, described by the following C structure:

```
struct filedat{
  int32_t time;  // time in ms at which the note is to be played.
  int16_t note;  // MIDI number of the note to be played.
  int16_t vol;   // volume of note: 0=silent to 32767=maximum
};
```

1. You can use as much or as little of the sample code as you wish, but your program must accept the same command line parameters, in the same order, as the original program.

2. Your program must write a Wave file on standard output. A Wave file consists of a file header followed by a stream of audio data, typically organized as a series of chunks.

3. The wave file must have 441 numbers for each millisecond of audio (44100 samples per second). The number that is written is the sum of the values returned by the Karplus-Strong algorithm for each of the 128 strings.

4. Your program must be as efficient as you can make it, in terms of both memory and CPU usage, but the output must differ by no more than ± 1 LSB from the output of the original program.

5. Your program must contain at least one useful function written in Assembly Language.

Comments and suggestions

- Do not delay. Start writing the program early. If you wait until a week before the due date, you will have a miserable week, and fail the assignment.

- Do not try to do everything at one time. Work on one improvement at a time, and make a backup after each step.
- Be sure to use good coding and documentation practices.
- Your program must correctly compile on a Raspberry Pi using the `make` command.
- Be sure your code files are readable and neat. Do not allow lines to extend past 80 characters, use appropriate white space and make sure to use a consistent and attractive indentation scheme.
- Provide plenty of helpful comments in your code.

Program Submission and Grading

Programming style is 25% of your total score, and the run-time test is 75% of your total score. Grading is performed using a script. You **must** submit a **single gnuzipped tar file**, with an extension of "`.tgz`". When extracted, the file must create a single directory named with **your last name, all lower-case**. The single directory, which is named with **your last name, all lower-case**, must contain all of your source code **and a `make` script, which must be named `Makefile`**. If your program fails to unpack properly, you will receive a zero for the project.

After your program is unpacked it will be tested by a script, which will execute the following shell command: "`cd <your_last_name_all_lower_case> ; make`" to build your program. Your `make` script, named `Makefile` must produce an executable file named `playmusic`. **If running `make` as descibed above does not create a file named `playmusic`, then your program has failed to build.** If your program fails to build, you will receive zero points for the run-time test.

If your program builds successfully, the script will run your program using the following command: "`cd <your_last_name_all_lower_case> ; playmusic <input file name>`". If your program builds, but fails to run properly, you will recieve ten points for the run-time test. If your program runs, then it will be ranked for output accuracy, execution time, and memory usage for three different input files. Your instructor will play the role of Steve. All students who beat or match Steve on accuracy, run-time, and memory usage, receive full run-time credit, regardless of the rankings.

Submit your program code using `http://www.mcs.sdsmt.edu/submit` before midnight of the due date.

DO YOUR OWN WORK.