**Optimizing Program Performance**

The goal of this assignment is to optimize, in stages, the performance of a convolution reverb program. Convolution reverb is an audio digital signal processing technique where a “dry” recording of an instrument (i.e., a recording without reverberation) is convolved with the impulse response of an acoustical space such as a concert hall. The result of the convolution is a sound file where the instrument sounds as if it were playing in the actual concert hall. This is a commonly used, but computationally intensive, technique for adding natural-sounding reverberation to recorded sounds.

You are to create a command-line program that takes in a dry recording and an impulse response, and produces the convolved signal. It should be invoked from the command line as follows:

convolve inputfile IRfile outputfile

where convolve is the name of your program, inputfile is the dry recording in .wav format, IRfile is the impulse response in .wav format, and outputfile contains the convolved signal in .wav format. All .wav files should be monophonic, 16-bit, 44.1 kHz sound files (at least initially, before attempting the bonus part of the assignment).

**Baseline Program**

Create an initial version of your program where the convolution is implemented directly in the time domain (i.e., use the input-side convolution algorithm found on p. 112-115 in the Smith text). You will find this version of your program quite slow. Measure the run-time performance of your program using a dry recording that is at least thirty seconds long, and an impulse response that is at least 2 seconds long. You will reuse these same inputs for timing measurements after each optimization that you do in the later stages of the assignment. There are many suitable dry recordings and impulse responses available on the Internet as well as on the course D2L site. You can use utility programs such as *sox* to convert sound files of different types (e.g. .aiff or .snd) to the .wav format.

Although the program may be implemented in any programming language, it would be best to use a language supported by the *gcc* compiler, since it is available on our servers, or can be installed on a home machine. The *gcc* compiler supports the C, C++, and Objective-C languages.

**Algorithm-Based Optimization**

Create a second version of your program where you re-implement the convolution using a frequency-domain convolution algorithm. A good candidate is the overlap-add FFT convolution algorithm described on p. 311-318 in the Smith text, although you can try other algorithms if you wish. You can also experiment with different Fast Fourier Transform (FFT) algorithms. Measure the run-time performance of this second version of program using the same inputs that you used for the baseline program. Be sure to use version control and regression testing as part of a disciplined process of optimization.

**Compiler-Level Optimization and Code Tuning**

Use compiler-level optimization and manual code tuning to further optimize the performance of your program. Do your improvements step by step, measuring and testing at each stage. Be sure to use version control and profiling as you make changes. Use at least 5 different manual code-tuning techniques and at least one compiler optimization.

**Report**

Create a formal written report that describes how you optimized your code at each step. You must show each version of your program, and describe what changes you made at every stage of the process. You must also quantify the improvements with your timing measurements, and show the regression tests you did. Use tables and/or graphs to help illustrate how you improved performance at each stage of your work.

**Bonus #1 (up to 5%)**

Elaborate your program so that it can handle stereo (i.e., 2-channel) impulse response files, and produce the appropriate stereo (2-channel) output file. In other words, your program will convolve a monophonic dry input sound with a stereo impulse response, and output a stereo sound file. Your program should be able to recognize automatically if the impulse response file has one or two channels.

**Bonus #2 (up to 5%)**

Elaborate your program so that it can deal with .aiff and .snd file formats, both for input and output. These formats should be automatically recognized using the suffixes appended to the filename and the headers in the files. For example, convolve input.aiff ir.snd output.wav specifies .aiff format for the dry signal, .snd format for the impulse response, and .wav for the output file.

**Submit the following to the D2L Assignment 4 Dropbox:**

1. An electronic copy of your report (in Word or PDF format). Be sure it includes all supporting material, including all versions of your program code, version control log reports, timings, profiler reports, and regression tests.
2. An electronic copy of your source code. Your TA will compile and run your program to confirm it does the convolution reverberation processing correctly. Be sure it can be run from the command line as described above.
3. If you do any of the bonus parts, a copy of the corresponding source code for your program. Indicate in your report if you are doing any of the bonus parts.

Baseline program 4 \_\_\_\_\_\_

Algorithm-Based Optimization 8 \_\_\_\_\_\_

Minimum of 5 manual code tunings 10 \_\_\_\_\_\_

Minimum of one compiler optimization 2 \_\_\_\_\_\_

Profiling 6 \_\_\_\_\_\_

Timing measurements 6 \_\_\_\_\_\_

Regression Tests 6 \_\_\_\_\_\_

Version control 5 \_\_\_\_\_\_

Program Design Quality 5 \_\_\_\_\_\_

Report 10 \_\_\_\_\_\_

**Total 62 \_\_\_\_\_\_ \_\_\_\_\_%**

**Bonus #1 5% \_\_\_\_\_%**

**Bonus #2 5% \_\_\_\_\_%**

**Grand Total** **\_\_\_\_\_%**