Sound Processing Kit

Kai Lassfolk
Department of Musicology
P.O.Box 35 (Viroonkatu 1)
00014 University of Helsinki
Finland
E-Mail: klf@elisoc.helsinki.fi

ABSTRACT: Sound Processing Kit is an object-oriented software system for audio signal processing. The system includes a library of C++ classes for various sound processing tasks and – more importantly – introduces a model for writing new audio signal processing programs. Sound Processing Kit demonstrates the benefits of object-oriented programming by providing core re-use through inheritance and aggregation, and by maintaining a close analogy to signal processing theory.

Introduction

Sound Processing Kit (henceforth SPKit) is an object-oriented audio signal processing software system developed at the Department of Musicology, University of Helsinki. SPKit contains more than 20 classes written in the C++ programming language. They include signal routing and processing modules such as a multiplier, an amplifier, a signal inverter, sum units, delay lines and a set of filters. These basic units may be used to form more complex modules, such as reverberators or filter banks.

SPKit was originally designed as a framework for student projects on object-oriented digital audio signal processing. The system was first implemented in Objective-C under the NEXTSTEP operating system. However, the design proved to be elegant enough for further development. To ensure portability to other platforms, SPKit was re-written in C++. In the future, SPKit may grow into a full-featured sound synthesis and processing language.

Software Architecture

SPKit is based upon a class hierarchy containing a single base class: SPKitProcessor. It defines a protocol for connecting objects with their signal inputs and for transmitting samples between objects. All other classes inherit from either SPKitProcessor or classes derived from it.

SPKit objects are connected in a C++ program to form a patch, where the first object is a "reader" (an instance of the class SPKitReader) and the last one a "writer" (an instance of SPKitWriter). Other objects in a typical SPKit patch simulate conventional signal processing or flow control units. When all objects are connected and initialized, a "run" call is issued on the writer. It executes the call by requesting samples from its "input" in a loop, one sample on each iteration. The run routine terminates when the writer's input object runs out of samples. The reader object reads the processed audio samples from a data stream such as a sound file or a pasteboard. Respectively, the writer writes the resulting samples to a data stream. SPKitReader and SPKitWriter classes also hide sound file format specific details such as sample size. Samples are passed between objects in floating point format.

Implementing New SPKit Classes

A new SPKit class typically needs to implement a member function named getSample, which returns a single processed sample and acknowledgement of possible end of signal. In order to satisfy the getSample
call the object may request one or more samples from its signal input (another instance of an SPKit class) by calling its getSample function. The SPKitProcessor class implements a default getSample function, which may be used e.g. by container classes, such as reverberators.

In addition, an SPKit class may need to implement an initialization routine named setupInput, which connects the object to its signal input and performs other initialization routines. SPKitProcessor, through its default setupInput member function, automatically initializes some fundamental parameters, such as sampling rate and channel count. These parameters are accessible by derived classes as protected data members. A new class may also need additional member functions e.g. for setting and retrieving parameter data.

SPKit includes classes demonstrating code re-use through both inheritance and aggregation. Inheritance is demonstrated by a set of Butterworth filters, where an abstract base class implements the getSample member function implementing the sample processing algorithm common to all Butterworth filter variants, and the setupInput member function for initialization and memory allocation. These functions are inherited by derived concrete classes which implement the low-pass, high-pass, band-pass and band-reject filter variants. The derived classes differ only in the implementation of setCutoffFreq (set cutoff frequency) or setCenterFreqBW (set center frequency and bandwidth) member functions.

Aggregation is demonstrated by a Schroeder reverberator, which consists of comb filters and all-pass networks. They in turn consist of delay lines as well as signal routing modules including multiplexers, amplifiers and sum units. Each module in the Schroeder reverberator is implemented as a distinct class.

Conclusion and Acknowledgements

SPKit is designed to be usable by a person having little or no experience in digital signal processing techniques, such as filter design. On the other hand, DSP experts will find that signal processing theory is easily applied to the implementation of new SPKit objects.

SPKit is developed under the NEXTSTEP operating system but is designed to be portable to practically any platform providing an AT&T CFront 2.0 compatible C++ compiler. SPKit includes on-line documentation in HTML format with links to up-to-date manuals and software updates over the Internet. The first public release of the software is scheduled for September 1995.

Several students from the Department of Musicology have contributed to the design and implementation of SPKit. In particular, Janne Halmkrona and Jula Henriksson wrote several classes and helped to test and debug the Objective-C version. Mikko Rinne has implemented a phase vocoder in C++, including classes for signal processing in the frequency domain.