The Synthesis ToolKit (STK)

Perry R. Cook[1] and Gary P. Scavone[2]

[1] *Department of Computer Science and Department of Music, Princeton University Princeton, New Jersey 08544-2087 USA prc@cs.princeton.edu*

[2] *Center for Computer Research in Music and Acoustics Department of Music, Stanford University Stanford, California 94305-8180 USA gary@ccrma.stanford.edu*

Abstract

This paper presents a cross-platform C++ programming environment designed for rapid prototyping of music synthesis and audio processing programs. The Synthesis ToolKit offers an array of unit generators for filtering, input/output, etc., as well as examples of new and classic synthesis and effects algorithms for research, teaching, performance, and composition purposes.

1 Introduction

A plethora of computer programming environments and applications exist for music synthesis. For the purpose of rapid prototyping of realtime synthesis and audio processing algorithms, environments with "drag and drop" graphical user interfaces (GUIs), such as Max and SynthBuilder, have generally been preferred. Such programs, however, often present problems to users and programmers alike, including cross-platform portability and user extensibility. Other non-GUI based environments, such as C-Sound, present powerful but complex programming paradigms that prove difficult for beginners to grasp. Development of the Synthesis ToolKit (STK) was motivated in large part by these problems (Cook, 1996).

In particular, fundamental design goals of STK have been:

- Cross-platform functionality
- Ease of use
- User extensibility
- Real-time synthesis and control
- Open source C and C++ code

Nearly all of STK is written in generic C and C++ and can be compiled on any system with a C++ compiler. Cross-platform functionality is further aided by encapsulating operating system dependencies, such as realtime sound and MIDI input/output, within a small number of classes. Portability problems are typically incurred with respect to GUI code. STK avoids this by using Tcl/Tk, a simple text-based scripting language that is freely available for nearly all current computer operating systems. Finally, STK runs completely on a computer's host processor, avoiding any specific hardware requirements other than a basic sound card.

STK achieves programming clarity and ease of use through an object-oriented structure. When convenient, coding optimization issues are addressed. In general, however, coding optimization is sacrificed for the sake of programming clarity. Given the "rapid prototyping" design goal, as well as the advent of gigahertz host processor speeds, such a tradeoff seems appropriate.

Realtime audio input/output and MIDI input functionality are currently supported for Irix, Linux, and Windows computer operating systems. Linux realtime support is accomplished using the Open Sound System (OSS) sound API, thus allowing further portability to other OSS supported systems (Solaris, HP-UX, etc.). Even when realtime support is not available for a given computer platform, however, it is easy to compile STK for traditional non-realtime functionality (the current level of NeXTStep support).

All source code for STK is freely available for non-commercial use, allowing full user extensibility and customization of its behavior. As a testament to its ease of use and pedagogical value, algorithms from the Synthesis ToolKit have been ported to various other sound synthesis systems such as Vanilla Sound Server, RTCMix, CSound, MSP, and SuperCollider.

Finally, the Synthesis ToolKit is first and foremost a set of C++ classes for music synthesis and digital audio programmers. A few example programs are provided with the distribution for demonstration purposes. However, these programs will most likely need modification to meet specific user requirements. Likewise, Tck/Tk GUIs are simple and functional but not necessarily robust. STK offers beginning and experienced programmers alike a great start at developing audio programs. STK is good for teaching and learning about synthesis algorithms. STK is an example of fairly careful C++ program design. And STK offers sound and MIDI support on lots of platforms. But, use of STK requires a basic understanding of C and C++ languages and a willingness to develop programming solutions to specific user requirements.

2 The Environment

All Synthesis ToolKit classes inherit from the *Object* master class. *Object* doesn't provide any program functionality but it does offer a convenient means for defining global program and operating system parameters. For example, operating system #defines in *Object.h* control proper byte swapping and the correct selection of audio/MIDI application programming interface (API) code during compilation. Further, STK's fundamental floating-point data parameter, MY_FLOAT, can be defined in *Object.h* as either a `float` or `double` value. Multi-channel data can be passed with the MY_MULTI parameter, which is defined as a pointer to MY_FLOAT.

All audio sample based STK classes implement a `tick()` method which, depending upon the given context, takes and/or returns MY_FLOAT or MY_MULTI data. Within this `tick()` method the runtime calculations of a class take place. With this structure, a soundfile can be read, delayed, and written to realtime output with the following program code:

```
SndWvIn *input = new SndWvIn("soundfile","oneshot");
DLineN *delay = new DLineN(maxLength);
RTWvOut *output = new RTWvOut(SRATE,channels);

input->normalize();   // normalize input data
delay->setDelay(50);  // delay by 50 samples

while (!input->isFinished()) {
  output->tick(delay->tick(input->tick()));
}

delete input;
delete delay;
delete output;
```

All objects which output audio samples implement a method `lastOut()`, which returns the last computed sample(s). This allows a single source to feed multiple sample consuming objects without necessitating an interim storage variable external to the object.

Currently, STK only implements single-sample `tick()` functions. This allows minimum memory usage, the ability to modularly build short (one sample) recursive loops, and guaranteed minimum latency through the system. Single-sample unit generator calculations, however, are nearly guaranteed to be sub-optimal in terms of computation speed. Vectorized unit generators take and/or yield pointers to arrays of sample values and improve performance significantly depending on the processor type and vector size. Though no specific support is planned for vectorized STK classes, all unit generators have been designed to allow easy conversion to vector functionality using arrays of MY_FLOAT and MY_MULTI values.

2.1 Unit Generators

At its core, STK makes use of the traditional unit generator paradigm originally introduced by Max V. Mathews in the Music *N* languages (Mathews, 1969). Current classes include envelopes, filters, noise generators, nonlinearities, and data input/output handlers. These unit generators form the fundamental building blocks for the various synthesis and audio effects algorithms discussed below.

The complementary base classes, *WvIn* and *WvOut*, and their associated subclasses handle data input and output to .wav, .snd, .mat (Matlab MAT-file), and .raw (STK raw) formatted files, as well as realtime audio input and output. File input subclasses can be configured for looping, oneshot, and/or interpolating behavior.

2.2 Synthesis Algorithms

A large variety of classic and new music synthesis algorithms are distributed with STK. These example classes were motivated by research, teaching, and music composition and performance demands, as well as a desire to create a set of different synthesis techniques which, wherever possible, share a common interface while still allowing the unique features of each particular algorithm to be exploited. Current sound synthesis techniques demonstrated include oscillator-based additive, subtractive, Frequency Modulation (FM), modal, sampling (Roads, 1996), physical models of string and wind instruments (Computer Music

Journal, 1992-1993), and physically inspired statistical particle models (Cook, 1997). Several models of the voice are provided and more vocal synthesis models are planned for the future. The particle-based models, combined with modal and PCM wavetable synthesis techniques, provide the framework for parametric synthesis of a large variety of real-world sounds and sound effects.

2.3 Effects Algorithms

The Synthesis ToolKit includes several simple delay-based effects algorithms for reverberation, chorus, flanging, and pitch shifting. The *RTDuplex* class provides simultaneous realtime sound input and output (when supported by hardware) for realtime effects processing. A simple effects demonstration application and control GUI are provided with the latest version of STK.

3 Realtime Control

STK control sources connect to synthesis programs via pipes and sockets, allowing for networked connections, and decoupling audio synthesis from control generation. An input handler, MD2SKINI, converts standard MIDI to SKINI (Synthesis toolKit Interactive Network Interface). Using SKINI, any language, system, or program capable of sending formatted text across a socket can control STK instruments. Wherever possible, STK algorithms share a common set of controllers which are mapped to standard MIDI controllers. This allows voicing and experimentation using Tcl/Tk GUIs, and expressive control using standard MIDI control sources.

3.1 SKINI

SKINI was created for the Synthesis ToolKit as a simple text-based extension to MIDI. SKINI supports a unified control interface across multiple platforms, multiple control signal sources such as GUIs of multiple flavors, MIDI controllers and score files, and simple text-based connections between processes on a single machine and across networks. SKINI extends MIDI in incremental ways, specifically in representation accuracy by allowing for floating point note numbers (micro-tuning for example), floating point control values, and double precision time stamps and delta-time values. Further, a text message basis for the control stream is used to allow for easy creation of SKINI files and debugging of SKINI control consumers and providers. Finally, SKINI goes beyond MIDI in that it allows for parametric control curves and functions to be specified and used. This allows continuous control streams to be potentially lower in bandwidth than MIDI (hence part of the name SKINI), yet higher in resolution and quality because the control functions are “rendered” in the instrument and/or in a performer-expert class which controls the instrument. Expressive figures like trills, drum rolls, characteristic pitch bends, heavy-metal guitar hammer-ons, etc. can all be specified and called up using text messages. To support SKINI scorefiles, the ToolKit provides SKINI11.cpp, which parses SKINI control data.

3.2 GUI Support

A number of Tcl/Tk control scripts are provided with the STK distribution. Tcl/Tk is a cross-platform scripting language that offers a simple means for creating sliders, radio buttons, etc. which generate control data that can be piped or socketed to an STK application. By formatting these control messages in the form of standard MIDI control messages, MIDI controllers can be exchanged for GUI control enabling real-time expressive synthesis control.

4 Summary

The Synthesis ToolKit offers a fast prototyping environment for audio DSP and computer music applications. STK’s cross-platform functionality and generic C/C++ development language minimize program obsolescence issues. STK’s object-oriented programming structure and “ease of use” design goals make it useful as a teaching aid. And STK’s open-source model allows for complete user extensibility. The Synthesis ToolKit is probably not for those that feel the need for GUI wrappers around their programs. But the benefits gained by avoiding such platform-dependent code far outweigh the inconvenience of having to port your algorithms every five years.

References

Computer Music Journal (1992-1993). Computer Music Journal. Special Issues on Physical Modeling, 16(4) and 17(1).

Cook, P. R. (1996). Synthesis ToolKit in C++, Version 1.0. In *SIGGRAPH 1996, Course #17 & 18, Creating and Manipulating Sound to Enhance Computer Graphics*. Available from ACM SIGGRAPH.

Cook, P. R. (1997). Physically informed sonic modeling (phism): Synthesis of percussive sounds. *Computer Music J.*, 21(3):38–49.

Mathews, M. V. (1969). *The Technology of Computer Music*. Cambridge, Massachusetts: MIT Press.

Roads, C., editor (1996). *The Computer Music Tutorial*. Cambridge: MIT Press.