Music Synthesizer

By:
Frank Chen
Maxime Paul
Michael Lo
Tate Hornbeck

Music Synthesizer

By:
Frank Chen
Maxime Paul
Michael Lo
Tate Hornbeck

Online:
< http://cnx.org/content/col10618/1.1/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 18, 2008
PDF generated: February 5, 2011
For copyright and attribution information for the modules contained in this collection, see p. 17.

Table of Contents

Chapter 1

Introduction[1]

1.1 Introduction to a Music Synthesizer

1.1.1 Our Goal

The aim of the project was to create realistic instrument sounds by means of digital signal processing (DSP). Algorithms and theories already exist for mimicking various instrument families and all revolve around modeling the instrument structure and material as well as how the instrument is played. One of the simpler algorithms is the Karplus-Strong, which produces amazingly realistic guitar sounds. Using this algorithm in combination with attack-delay-sustain-release (ADSR) concepts, we were able to synthesize some decent instrument sounds.

1.1.2 Next Step

The first step would be to increase our instrument library to include different instrument families. Karplus-Strong mainly works for string and some percussion instruments, which greatly limited the diversity of our virtual orchestra. This would consist of implementing different algorithms to simulate the various types of instruments. Finally, how does mimicking instruments through DSP constitute music synthesis? The next objective would be to generate random pieces of music. A plausible scheme would be to implement a first or second order Markov model to generate random pieces with musical structure but leave enough room for creativity.

[1]This content is available online at <http://cnx.org/content/m19005/1.2/>.

Chapter 2

Karplus-Strong Algorithm[1]

2.1 Karplus-Strong Algorithm

2.1.1 How It Works

The Karplus-Strong algorithm was developed by Alexander Strong and analyzed by Kevin Karplus as a model of hammered or plucked string instruments. It simulates the sharp impact through a short wideband signal such as a burst of white noise. The signal is fed back through a delay line whose length depends on the frequency of the desired note. The delayed signal is sent through a lowpass filter to attenuate all other frequencies except the frequency of the note and its harmonics.

[1] This content is available online at <http://cnx.org/content/m19006/1.2/>.

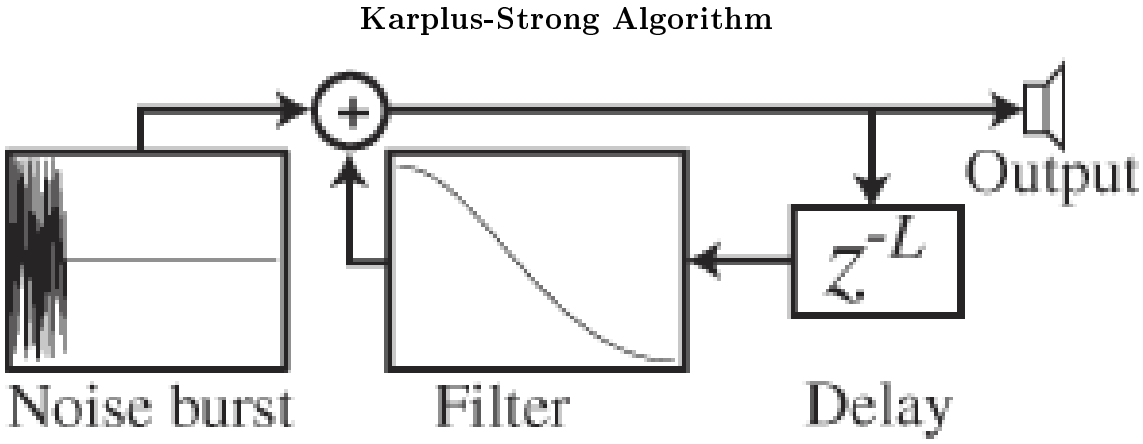

Figure 2.1: A diagram representation of a burst of white noise being delayed, filtered, and combined with the original burst. The output sounds like a realistic guitar string pluck.

2.1.2 Concepts

The main concept behind the algorithm is to model the sudden attack of a pluck with white noise containing equal energy in all frequencies. Due to the cavity of the instrument, the instrument material, and various other parameters, only a given frequency and its harmonics will resonate. This is simulated by recursively shaping the output signal. By matching the length of the time delay to correspond with the frequency of the note desired, the output will ultimately sound at the selected frequency given a short period of time. The feedback loop only reinforces the fundamental frequency and its harmonics. This technique is sometimes referred to as a comb filter because of the characteristic shape of the output.

Spectrum of a Plucked String

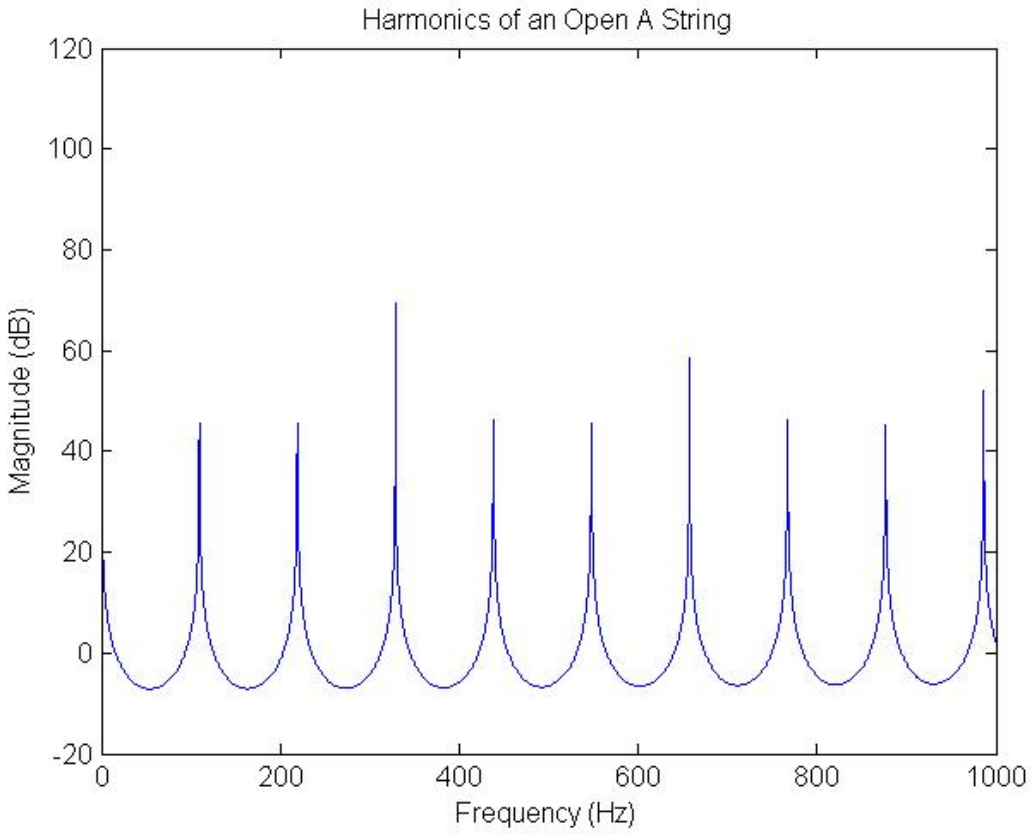

Figure 2.2: The output of the Karplus-Strong algorithm with its characteristic comb shape. Notice the rapid attenuation of all other frequencies not near a harmonic.

Chapter 3

ADSR[1]

3.1 ADSR

Attack-Decay-Sustain-Release (ADSR)

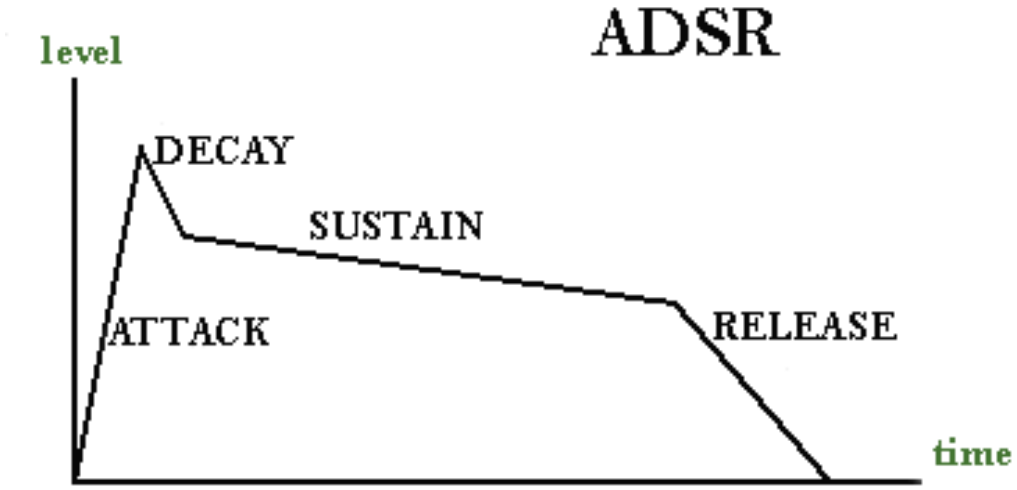

Figure 3.1: The ADSR curve models different instruments by their temporal characteristics. Instruments have varying degrees of the abruptness of the attack, the initial decay in sound, how long the sound resonates for without appreciable attenuation, and how quickly the sound fades away at the end.

[1]This content is available online at <http://cnx.org/content/m19007/1.2/>.

3.1.1 Introduction

ADSR stands for attack, decay, sustain, and release and is used to model the timbre of an instrument. The timbre or tone quality is determined by various factors such as the way the sound is produced and the material of the instrument. Different families of instruments have their own characteristic ADSR profiles. The attack refers to the phase in which the sound is initiated. This could be the fast attack of a strum on a guitar or the slower one of a pipe organ. The decay phase occurs immediately after the attack impulse and describes how rapidly the sound dies. Some instruments like a drum have extremely fast decay and the sound is virtually nonexistent after the attack. The sustain profile of an instrument refers to how long the sound resonates for when it is played. String instruments such as a violin have an extremely long sustain because the violin's sound box is receiving constant vibrational energy from the bowed string. Finally, the release phase describes how rapidly the sound fades away once the instrument is not being played.

Guitar ADSR Profile

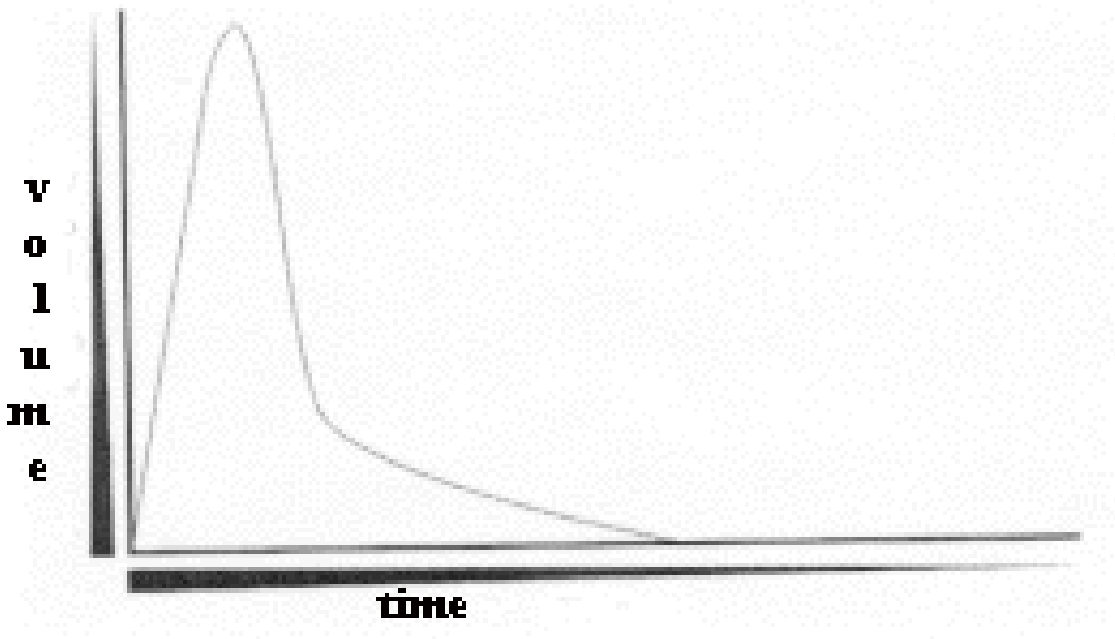

Figure 3.2: The guitar has a fairly abrupt attack due to the method of playing through plucks and strums. The rapid decay is a result of the nonharmonic frequencies fading quickly leaving only the fundamental frequency and its harmonics. The sustain and release phases are merged in this case since the sound just fades away slowly.

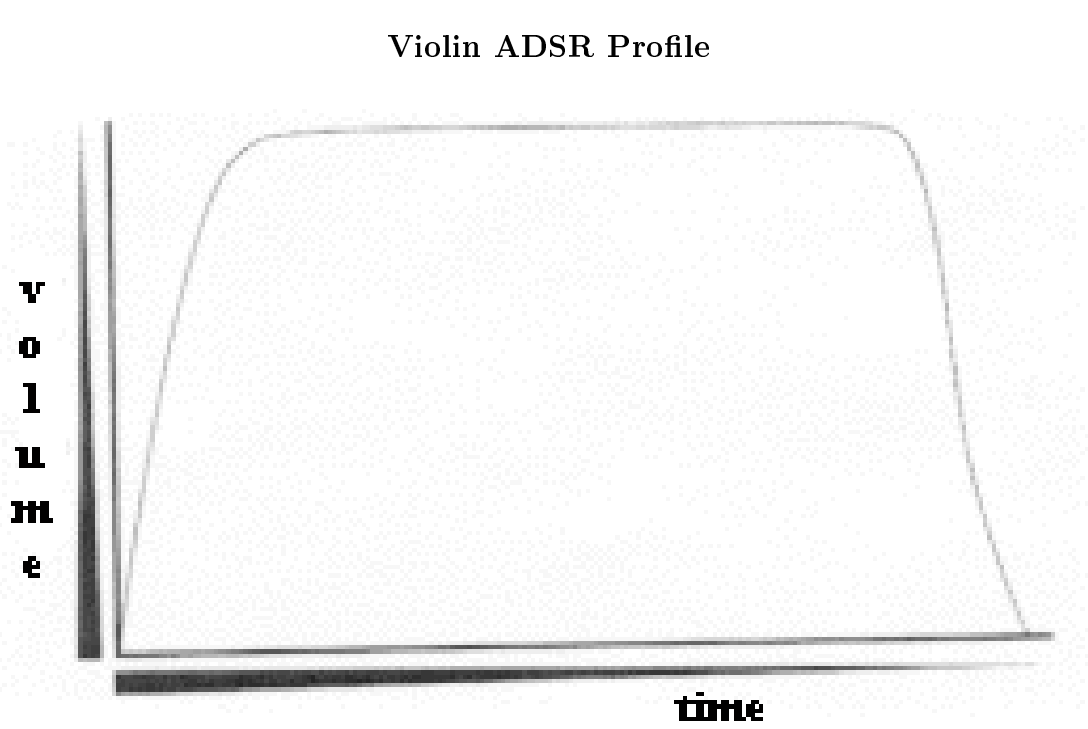

Figure 3.3: The violin has a slightly slower attack than the guitar but the other phases are drastically different. Because a violin is bowed to produce sound, there is virtually no decay and a very long sustain. Constant vibrational energy is delivered to the violin's sound box and the sound only fades once the bowing stops.

3.1.2 Implementation

In order to synthesize different sounding instruments, the ADSR envelope could be applied directly to the output of the Karplus-Strong algorithm. Since the algorithm models string and certain percussion instruments, there were limitations on the diversity of instruments that could be synthesized using this technique. After modeling an instrument's temporal characteristics with an ADSR envelope, one could apply it to the output of the Karplus-Strong by point-wise multiplication. By using ADSR, we were able to manipulate the guitar sounding output of the Karplus-Strong algorithm to sound like different instruments such as an organ or a bell.

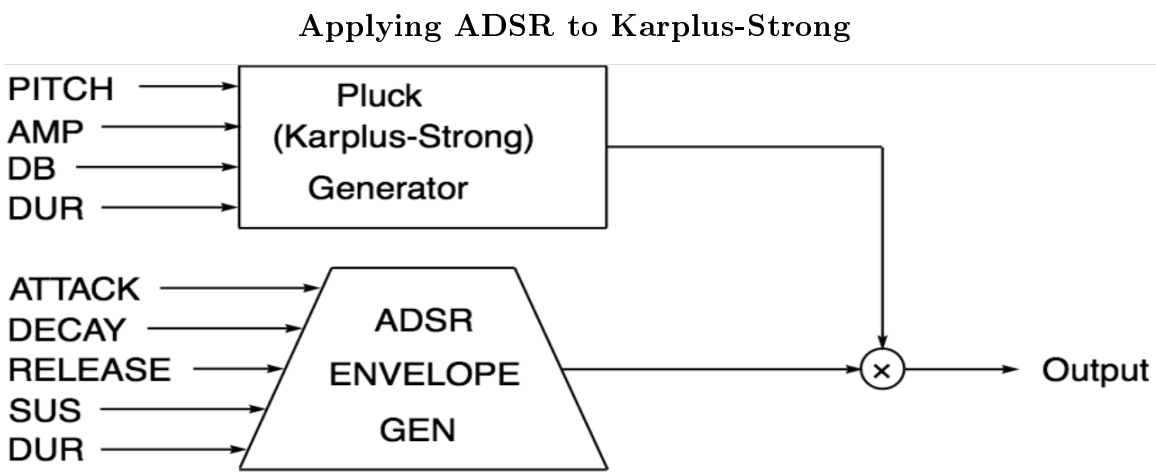

Figure 3.4: Synthesis of different instruments in an instrument family can be achieved through applying ADSR concepts to the Karplus-Strong algorithm. The output of the Karplus-Strong is point-wise multiplied with the modeled ADSR envelope to produce the final output sound.

Chapter 4

Results[1]

4.1 Results

4.1.1 Karplus-Strong Algorithm

The Karplus-Strong algorithm worked extremely well for producing realistic guitar sounds. We took advantage of this and synthesized chords as well simply by adding the outputs of the individual notes together. To simulate the physical strum of a guitar, we inserted a delay between each of the six strings of the guitar so that they would sound slightly later in time. An electric guitar sound could be easily synthesized by adding some noise to the output of the Karplus-Strong algorithm. The end product was a very pleasing imitation of a guitar.

[1] This content is available online at <http://cnx.org/content/m19008/1.2/>.

Spectrum of a Strummed Chord

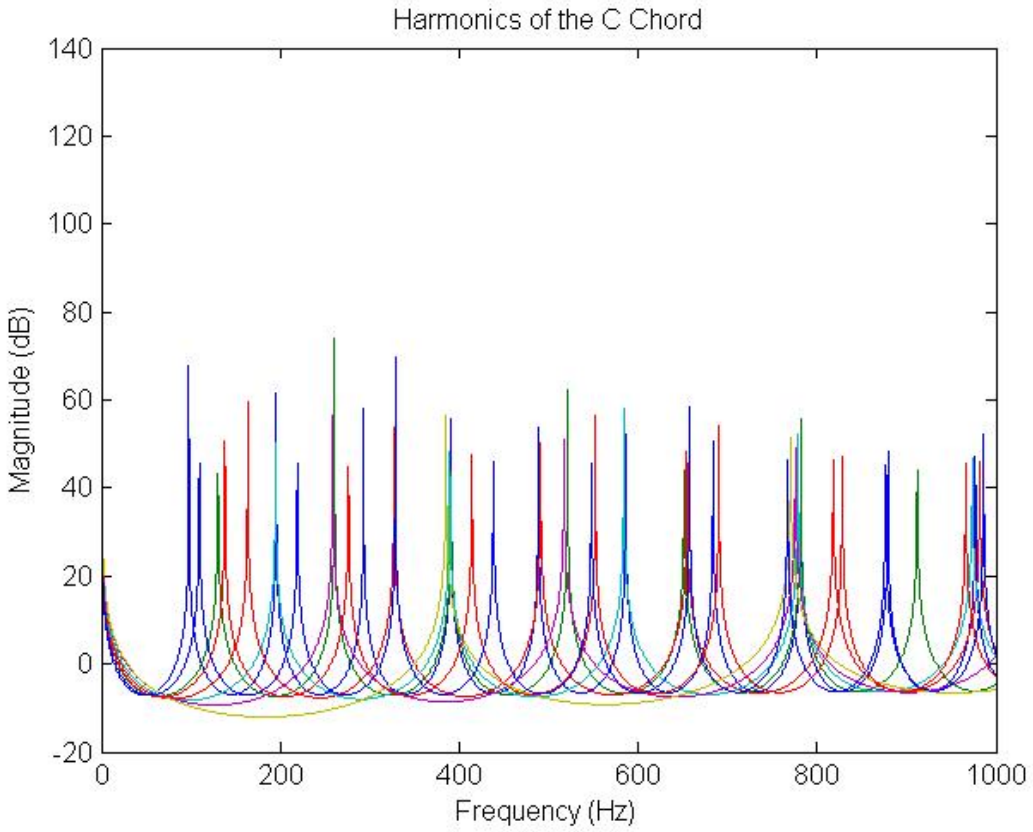

Figure 4.1: The result of combining the individual notes generated by the Karplus-Strong algorithm together to form a chord.

4.1.2 ADSR

Applying the ADSR envelope to the output of Karplus-Strong allowed us to alter the guitar sounds into ones that sounded like an organ, bell, and the pluck of a harpsichord piano. We created the ADSR curves through a mixture of research and our own intuition from having heard each instrument.

Chapter 5

Random Music Generator[1]

5.1 Random Music Generator

5.1.1 Introduction

Our original idea was to be able to generate random music once the instrument sounds were synthesized. Unfortunately, due to time constraints, the best we could do was to have a program randomly pick chords and note durations from an input library. The resulting output was, as expected, random chords and notes that most people would probably not consider as music. How does one teach a program how to write music? One technique is the Markov chain.

5.1.2 Markov Chains

Teaching a program music theory so that it can create music would be an extremely tedious task. You would have to teach chord structure, common progressions, the different musical styles, and so on. What if you could give the program examples of pieces you considered to be music and ask it, "write something like that for me." This is essentially how our Markov chain would work. The principle behind Markov chains in music is to generate a probability table to determine what note should come next. By feeding the program an example piece of music, the program can analyze the piece and create a probability table to determine which notes are more likely follow a given note. Below is an example of a first order Markov chain.

[1]This content is available online at <http://cnx.org/content/m19009/1.2/>.

Example of First Order Markov Chain Table for Music

	A	A#	B	C	D	E	F	G	G#
A	4/19		3/19		2/19	1/19		6/19	3/19
A#	1								
B	7/15		1/15	4/15		3/15			
C			6/15	3/15	6/15				
D				3/11	3/11	5/11			
E	4/19	1/19		3/19		5/19	4/19	1/19	1/19
F			1/5			1/5		3/5	
G	1/5		1/5	2/5				1/5	
G#			3/4			1/4			

Figure 5.1: The entries in each box indicate the probability or likelihood of the note in the column label to follow the note in the row label. Blank boxes indicate 0 probability, meaning that note will never be the next note given your current note. For example, if the current note is A#, the next note chosen will always be A.

With the probability table, one can generate random notes that still has some musical structure to it. By constructing a similar table for beats or note durations, one can complete the first order Markov chain for music generation. Increasing to a second order Markov chain simply means constructing a larger probability table where the row headings are now pairs of notes. The program will choose the next note according to the probability table, consequently updating the current note pair to include the newest note. The idea behind determining what order Markov chain to use is a balance between ensuring that the program has enough musical structure and allowing it enough freedom for creativity.

Chapter 6

Extension[1]

6.1 Extension

6.1.1 What Next?

Having synthesized a few instruments and come up with a method to generate music randomly, the next step would be to include different families of instruments by trying other algorithms to physically model them. This would expand the library of instruments for our synthesizer to make it more interesting. Our long-range goal is to be able to analyze music input such as a voice and generate random musical accompaniment. The inspiration for this project was the research done at Microsoft in developing a program called MySong that automatically chooses chords to accompany a singer. This allows casual singers to create songs complete with accompaniment on their own. The project helped us learn many of the tools we would need to build a program similar to MySong.

Link[2] to Microsoft's MySong page.

[1] This content is available online at <http://cnx.org/content/m19010/1.2/>.
[2] http://research.microsoft.com/en-us/um/people/dan/mysong/

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Music Synthesizer*
Edited by: Frank Chen, Maxime Paul, Michael Lo, Tate Hornbeck
URL: http://cnx.org/content/col10618/1.1/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Frank Chen
URL: http://cnx.org/content/m19005/1.2/
Page: 1
Copyright: Frank Chen
License: http://creativecommons.org/licenses/by/2.0/

Module: "Karplus-Strong Algorithm"
By: Frank Chen
URL: http://cnx.org/content/m19006/1.2/
Pages: 3-5
Copyright: Frank Chen
License: http://creativecommons.org/licenses/by/2.0/

Module: "ADSR"
By: Frank Chen
URL: http://cnx.org/content/m19007/1.2/
Pages: 7-10
Copyright: Frank Chen
License: http://creativecommons.org/licenses/by/2.0/

Module: "Results"
By: Frank Chen
URL: http://cnx.org/content/m19008/1.2/
Pages: 11-12
Copyright: Frank Chen
License: http://creativecommons.org/licenses/by/2.0/

Module: "Random Music Generator"
By: Frank Chen
URL: http://cnx.org/content/m19009/1.2/
Pages: 13-14
Copyright: Frank Chen
License: http://creativecommons.org/licenses/by/2.0/

Module: "Extension"
By: Frank Chen
URL: http://cnx.org/content/m19010/1.2/
Page: 15

Music Synthesizer
ELEC 301 PROJECT FALL 2008 MUSIC SYNTHESIZER

About Connexions

Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.