Java Music Systems
Professor Nicholas Didkovsky
Due Date: 12/20/2011

A Sieve Calculator: Java Music Specification Language Meets the *Sieves Like Teen Spirit* Project

James Keary

New York University, Steinhardt, Department of Music Technology
New York, New York

jpk349@nyu.edu

Abstract

The creation of a sieve calculator has the potential to serve and aid the modern day composer. It has the capability of breaking us free from the confines past composers have faced. One disadvantage to being a human is that our memory is not "a blank slate": we choose musical elements such as pitch, rhythm, and note duration, based on subjective factors of taste. A computer "thinks" differently: unlike humans, a calculator/computer has *no taste*! The connotation of the term intends that having *no taste* is having *bad taste*. However, *good taste* can confine us as composers. The creation of a sieve calculator, a computer program that can objectively choose series of musical elements for us, can break us free to new worlds of compositional though that we as composers may not have seen before.

1. Introduction to Sieves

The use of sieves in composition has a longer history then one might imagine. Considered to be the father of sieve theory, composer Iannis Xenakis (1922-2001) would contend this to be true. The theory behind sieves finds its origins in the historical attempts of Western European composers to rationally explain and assign axioms to musical elements.[1] The earliest documented approach in Western Europe would be that of medieval music theorist and Italian monk Guido of Arezzo. Guido of Arezzo devised a system of melody generation from a religious text and published his system in his 1026 work *Micrologus*[2]. The text was 8th century religious Latin hymn, *Ut queant laxis* (*The Hymn of St. John*). Arezzo assigned specific notes values to specific syllables: *ut, re, mi,* etc.[3]. Throughout the medieval ages, the system was developed; for example *ut* became *do*. This series of notes is referred to as solfège, and is what we know today to be the major scale. Most of western music and music theory is based on Arezzo's sieve, or series of musical notes.

Only recently in the time line of western music have alternative series and orderings of notes been presented. The most notable example is Schoenberg's 12-tone series based on the 12 notes of the chromatic scale. As opposed to Arezzo who placed his faith in God, Schoenberg

[1] Iannis Xenakis, *The Origins of Stochastic Music,* Tempo, New Series, No. 78, pp. 9
[2] Nierhaus, G. *Algorithmic Composition: Paradigms of Automated Music Generation*. p. 21
[3] Burkholder, P., Grout, D., & Palizca, C. *A History of Western Music*, 8th Edition, p 43-45

placed his faith in geometry when creating his series. The 12-tone system did not have a hierarchy of note importance. Every note was as important as the next. This was achieved through symmetry of intervals where all 12 tones are 1 semitone apart. From this ordering of notes one soon realizes that the hierarchy of importance, with the tonic and dominant at the top, is gone. The idea of the tonic is replaced with the idea of *pitch center*. The term *pitch center* was defined as the note where the cycle or series of notes begins; however, it had no importance to it besides a place marker[4]. Through a completely ordered series of notes came sounds of disorder.

So, since sieves are the ordering of series, both Arezzo's solfège series and Schoenberg's 12-tone series are sieves. If one define a series of notes through symmetrical interval leaps around a pitch center, as Schoenberg does, then one could create a plethora of different series of notes just based on the 12-tone system alone, just by choosing different intervals and or pitch centers. You could also make your cycle as long or as short as you want, not just a cycle of 12 notes. You could also get into microtonal notes. The possibilities for tone series creation based on symmetry are numerous indeed.

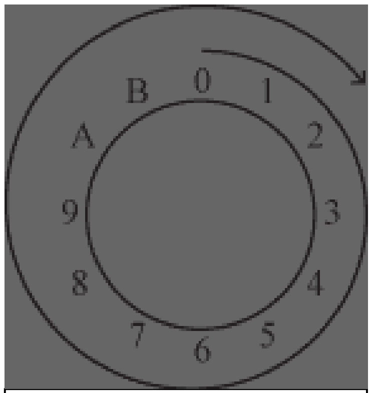
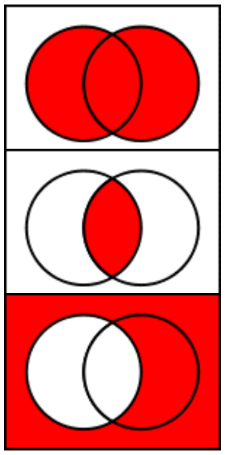


Figure 1: mod 12

Xenakis took this idea and ran with it. He applied this idea of series creation not only to pitches, but also to other musical elements and the objects that define them as well. This includes, but is not limited to, notes, duration, rhythm, pitch, timbre, resonance, amplitude, velocity, hold, chords, scales, etc. In his book *Formalized Music*, Xenakis lays out the way these elements become usable mathematic axioms. A simple sieve can be written as this: $x \equiv n \pmod{m}$. Note x is the *pitch center*, the cycle of m notes is a *modulus*, and the symmetrical and intervallic leaps by number n around the modulus starting at the pitch center is known as the *residue class*.[5] For example, if the pitch center is C, The exponent 'x' is 0, if the pitch center is C#/Db, x = 1, and around the cycle (see Figure 1). The objects that comprise the sieve are written this way: $n_x = \{\ldots x\text{-}3, x\text{-}2, x\text{-}1, x, x\text{+}1, x\text{+}2, x\text{+}3\ldots\}$, stopping at the end of modulus m. As you can see, the symmetry of the hops is reflected around x. Hoping around a cycle is what is known as *modulo arithmetic* and it is the basis of creation for all of Xenakis's sieves.



A prime example of non-symmetrical interval spacing between notes would be the major scale. In the major scale, the series of intervals goes; whole (2 semitones), whole, half (1 semitone), whole, whole, whole, half, at which point the cycle repeats itself. Xenakis applies Boolean algebra as a way to relate simple sieves to each other to achieve non-symmetrical series of intervals. The Boolean operators Xenakis used are: AND (intersection, symbol: $\wedge$); OR (union, symbol: $\vee$) and NOT (negation, symbol: $\neg$). OR, meaning union or disjunction, takes grouped sieves and uses all their elements to create a complex sieve. AND, meaning intersection or conjunction, takes like elements from the grouped sieves. NOT meaning complimentary or negation, takes no elements one simple sieve, and only uses elements outside of that series. There are other Boolean operators that relate to set theory, but Xenakis

[4] Perle, G., & Lansky, P. *Atonality* entry. Oxford Music Online, Source: Grove Music Online.

[5] Xenakis, I. *Formalized Music, Thoughts and Mathematics in Composition*, Revised Edition.

does not employ them because these three cover all the basics. Through Boolean operations, Xenakis could create complex sieves with non-symmetrical interval leaps[6]. For example, the sieve of the C major scale is as follows:

$$(\neg 3_2 \wedge 4_0) \vee (\neg 3_1 \wedge 4_1) \vee (3_2 \wedge 4_2) \vee (\neg 3_0 \wedge 4_3)$$

As one can see, sieves can reach a high complexity fairly quickly. So, Xenakis and other composers using sieve theory employ the computer to explore complex sieves faster than one can do by hand. Today, there are many computer composition programs the employ the use of sieves to aid in the composer's compositional process. Notable ones that this researcher has explored include Xenakis's GENDYN program (coded in BASIC and C)[7], OpenMusic by Carlos Agon, Gérard Assayag and Jean Bresson (coded in CommonLisp)[8], and NYU graduate Christopher Ariza's amazingly complex athenaCL program (coded in Python)[9]. Which brings us to the current work of this researcher.

2. The Project: A Sieve Calculator and Sieves Like Teen Spirit

The goal of this project is to create a tool to go into the toolbox of the modern day composer. Unlike some of the programs mentioned above, the Sieve Calculator program is not intended to write entire musical pieces based on user input. Just as a scientist may need a calculator to do complex mathematics, a mathematical or algorithmic composer would need something similar. A sieve calculator would serve to aid in the pursuit of musical expression, bringing the composer into under or unexplored realms. The Sieve calculator is different however from a normal calculator in that its output is not just the raw math: the modulus arithmetic to create simple sieves and the Boolean operations to create complex sieves. This particular calculator must aid the composer in other ways as well. Being able to hear or see the resulting output would be the logical next step. However, this presents many issues and questions.

The main challenge faced is the inherent challenge of all algorithmic composition. How much "artistic" control should the program have and how much control should the user have? By making the Sieve program a calculator, most of the control remains in the hands of the composer. The only part of the compositional process for the program is in the calculation of the mathematics. So, if the program generates an example of that output, what exactly should it be doing? The computer algorithmic composition process has the side affect of creating a distance between the composer and the piece they are creating. Sieve algorithmic composer Dr. Sever Tipei however believes that this is the point, categorizing this type of composition as 'un-romantic'[10].

However, composers have been using algorithms in composition for a long time; what has changed?

[6] Noll, T., Andreatta, M., & Agon, C. *Computer-Aided Transformational Analysis with Tone Sieves*, p 2.
[7] Hoffmann, P. *The New GENDYN Program*.
[8] Agon, C., Assayag, G., Laurson M., Rueda, C. *Computer Assisted Composition at Ircam: PatchWork & OpenMusic*, p 59.
[9] Ariza, Christopher. *Compositional Algorithms: The Xenakis Sieve as Object: A New Model and a Complete Implementation*
[10] Tipei, S. *Solving Specific Compositional Problems with MP1*

"… The process of musical composition itself has been described by many writers as involving a series of choices of musical elements from an essentially limitless variety of musical raw materials. The act of composing… can be thought of as the extraction of order out of a chaotic environment." (Neirhaus. p 238)

With the addition of the computer to the composer's toolbox, the process Neirhaus described is actually inverted. The environment is now a computer-programming language, which is completely ordered. Disorder and chaos are extracted from the order. The Romantic composers of the 17th and 18th centuries were known for breaking down the structural ideas of composition that were so ingrained at the time of the Renaissance. The use of computer programming languages in composition is diametrically opposed to the idea of breaking away from a set of rules: instead rules are imposed through sieve theory, and or other algorithms, onto the musical form.

For this researcher, the answer was to have the Sieve calculator generate a melody to the rhythm of one of the most iconic and well-known rock songs of all time. The user inputs the modulus, pitch center and residue class of 2 simple sieves, and then through whichever Boolean operator they want creates a complex sieve. Complex sieves can then be added together as much as the user wants to create whichever series they desire. Then through a random selection of members of that series a melody is generated based to the set rhythm and song structure of Nirvana's *Smells Like Teen Spirit*. This particular implementation of the sieve calculator is called the Sieves Like Teen Spirit Project.

3. Technical Description: JMSL Implementation of the Project

The Sieve Calculator applet was written in Java. The platform for the Sieves Like Teen Spirit experiment was the Java Application Platform Interface Java Music Specification Language (JMSL) created by Nick Didkovsky and Phil Burke. The two applets were written in the Eclipse IDE program.

The Sieve applet was authored by Alan Johnson and James Keary. It is separate from the Sieves Like Teen Spirit applet so that the Sieve class is changeable for the future when newer functionalities are added and has the ability to be implemented by other classes, not just the Sieves Like Teen Spirit class. The sieve object maintains a collection of methods that can be called upon and acted out by other programs / applet. Those methods perform the modulus arithmetic and Boolean operations of the creation of the simple and complex sieves. The applet main method is the creation of a sieve as an array of Boolean relating values. Another method finds the length, and another counts the members of the array. The filter method determines the starting pitch and the interval leaps around the modulus, rather through the array of members. The chooseValue method determines how the values from the sieve are chosen, and it extends (uses) the JMSLRandom.choose class, as a behavior or choosing numbers randomly from the sieve. There is method for sending the sieve array as a string. And then there are 3 methods pertaining to the Boolean operations that relate the simple sieves together to create complex sieves. These are all the methods that called upon in the sieve class. If other methods need to be created in the future, they can easily be added to increase the functionalities of the sieve object.

The Sieves Like Teen Spirit applet was coded by James Keary based on code by Nick Didkovsky. This applet uses a Sequential Collection of MusicShapes to play a composition built on the song structure and rhythm of Smells Like Teen Spirit by Nirvana. The JSyn SynthNote class FilteredSawtoothBL plays the melody based on the sieves input by the user. The GUI layout of the applet is fairly simple to be reminiscent of a calculator with several buttons and text fields and areas. Action Listeners are attached to the buttons, fields and areas that call particular methods from the sieve applet. The complex sieve, or sieve c, values are passed into the MusicShapes in a random sequence into the pitch parameter. Thus the melody changes over time. This information could be added to whichever of the MusicShape dimensions to manipulate other elements, such as duration, amplitude and hold.

4. Artistic Analysis

It is hard to artistically analyze a calculator. If the composer's creative process lies in he creation of a computer algorithm, then how does one artistically analyze programming code? Regardless, I was definitely pleased with the melody outputs of the program. The Sieves Like Teen Spirit project was successful in that regard. However, I must admit that all the melodies created by experiments with the program did not exceed my skeptical expectations. None of the melodies that were output by the program were melodies that I thought a human could not have written. Perhaps humanity arrived at the creation of that melody faster with the aid of the sieve calculator. But, tools are made to do two things; 1) to perform actions that humans can do but faster, and 2) to perform actions that humans cannot do. In the second regard, this researcher believes that this project failed. At the risk of sounding too harsh, suffice it to say this project proved to only be a preliminary exploration, and ideas of how sieves can be used further in the future were discovered.

5. Conclusion: The Future of the Sieve Calculator

For the Sieve Calculator program to become an actual useful tool in the composer's toolbox, additions need to be made. The Sieves Like Teen Spirit project only explored pitch series, specifically Western European pitch series. The calculator could potentially be made to explore microtonal pitches. It could also be made to do a different note selection process as well. Right now the calculator picks members from the series in random order. But what if there was some sort of artistic reasoning behind the selection process?

Also, why not make the calculator explore all the areas that Xenakis intended it to? Future versions of the Sieve Calculator would include sieves for other musical elements such as rhythm, duration, amplitude, frequency, phase, hold, and more. Given the wide scope of possibilities, this researcher foresees a future for the Sieve Calculator as a useful tool in the composer's toolbox.

Acknowledgements

I must first extend thanks and praise to Alan Johnson. Without his expertise and genius in the Java programming environment, the coding of the Sieve Calculator would not have been possible.

Next, thanks goes to Phil Burk and Professor Nick Didkovsky for the creation of JMSL. The language provided a perfect way to express these musical ideas.

References

Ariza, Christopher. *Compositional Algorithms: The Xenakis Sieve as Object: A New Model and a Complete Implementation*, Computer Music Journal, Volume 29, Issue: 2. pp. 40-60. 2005

Agon, C., Assayag, G., Laurson M., Rueda, C. *Computer Assisted Composition at Ircam: PatchWork & OpenMusic*, Computer Music Journal. Volume 23, No. 5. 1999.

Burkholder, P., Grout, D., & Palizca, C. *A History of Western Music*, 8th Edition. New York, W.W. Norton & Company. 2010.

Hoffmann, P. *The New GENDYN Program*. Computer Music Journal, Volume 24: No. 2, pp. 31-38. 2000.

Nierhaus, G. *Algorithmic Composition: Paradigms of Automated Music Generation*. New York, SpringerWien. pp. 1 – 65. 2009.

Noll, T., Andreatta, M., & Agon, C. *Computer-Aided Transformational Analysis with Tone Sieves*, Computer Music Journal. pp 1–6. 2006

Perle, G., & Lansky, P. *Atonality* entry. Oxford Music Online, Source: Grove Music Online.

Pohlmann, K. C. *Principles of Digital Audio*, 6th Edition. New York, McGraw Hill. 2011.

Tipei, S. *Solving Specific Compositional Problems with MP1*, Proc. 1981 International Computer Music Conference, (North Texas State Univ., Denton, Texas; November 1981), International Computer Music Association, pp. 101-109. 1989.

Xenakis, I. *Formalized Music, Thoughts and Mathematics in Composition*, Revised Edition. Stuyvesant, New York, Pendragon Press. Chapters 7 – 12. 1992

Xenakis, I. *The Origins of Stochastic Music*, Tempo, New Series, No. 78, pp. 9-12. 1966.