glass tube, and we can no longer increase the pitch no matter how far we reduce the water level below the half level, the pitch stays the same. The five interval scale created by the short glass tube gamelan is shown in the Figure 16.

Figure 16. The final short glass tube gamelan.

5.2.2 Long glass tube analysis and gamelan fabrication

Follows the similar procedure as in 5.2.1, we obtain the spectrum (Figure 17) of the long glass tube and the resultant Sensory dissonance curve (Figure 18) by using 5 principal components of Figure 17. Then perform the fine tune to derive the intervals of ratio 1.184, 1.27, 1.39, 1.44, 1.64 and 1.84. Table 4 is the frequency comparison chart between the long glass tube gamelan and 12-tet scale.

Figure 17. The spectrum of the sound created by striking the long glass tube with water level almost full.

Figure 18. Sensory Dissonance Curve of long glass tube.

In this set up, similarly, we can only find 6 intervals for the 7-note scale (Figure 19). Namely the highest pitch we can reach is only up to the sixth note. At this point, the water level has already reached half of the tube, and there is no way of pitch increase by reducing water level.

6. CONCLUDING REMARK

Scale of non-harmonic percussion instrument normally possesses fewer notes to the octave than 12-tet scale as that of a harmonic instrument does. Interval of scale has to be properly chosen to create consonant sound. We have shown that equal size glass-tube filled with different level of water can be served as a type of gamelan for recreation play ground applications. The levels are adjusted based on a modified dissonance measure. Two types of glass-tube were tested. Analysis shows that they resemble a Pelog type gamelan and possess a 7-note scale but with uneven intervals. Both tubes have similar spectrum pattern at different pitch range. We also found that in order to generate all 7 intervals we need to further increase the height of tubes we used. We had gained knowledge of sounding water filled glass-tube that higher water level creates lower pitch is quite opposite to people’s thinking, but it remains further investigation effort such as mathematical analysis of the glass vibration pattern related to tone, as well as the integration of control apparatus like Max/Msp and microprocessor (Arduino) in real-time interactive operation in the future.

7. REFERENCES


ELECTRONIC CIRCUIT BUILDING IN PERU: A SUBALTERN CASE ON PARTICIPATION AND TECHNOLOGY

José Ignacio Lopez Ramirez Gaston
Pontificia Universidad Catolica del Peru
Escuela de Musica
jilopez@pucp.pe

ABSTRACT

In this paper we take a brief look at the history of circuit building for musical performance in Peru and examine its current status, focusing particularly on the attempts made during the last decade to generate a local popular electronic musical scene derived from the use of homemade sound generating electronic devices. Acknowledging the historical shortages common to many subaltern societies and their role in the development of Peruvian sound arts, I examine here a social history from below that serves as social commentary for the Peruvian electronic musical scene and the development of particular musical identities. To this aim, I argue for the necessity to apply alternative educational strategies tailored to meet the needs of a new generation of Peruvian musicians; strategies that can make this shortages not a deficiency but an opportunity.

I present at the end of this paper a case report: a workshop I developed in 2012 at the Escuela de Musica of the Pontificia Universidad Catolica del Peru, PUCP), in Lima.2 This workshop was dedicated to the fabrication of basic electronic circuits for sound generation. The intention was twofold: (1) to provide the student with basic technical knowledge for the development of musical instruments non dependent on their economical possibilities; and (2) to open the students conceptually to forms of composition and performance not currently discussed or present in their educational or social environments.

1. INTRODUCTION

Musical communities and their identities are, in part, constructed based on the technological currency present in the environments they flourish. While particular marketing strategies produce a craving for specific musical tools, these strategies not always take under consideration the economical conditions in developing countries. Some historical ‘tools of the trade’ of the electronic musician never made it into the Peruvian scene, and the history of electroacoustic music in Peru, and related sound arts, is full of holes and struggles. Even thought we have had in the past professionally trained musicians, that could claim the have the credential to belong to the western culture of the educated sound arts;3 for the most part, the contemporary Peruvian electronic musician and sound artist is part of the age of “the rise of the amateur”, and is, at best: a professional amateur [7].

During the last decade, the constant drop on the price of consumer level computers, software piracy becoming a social standard, and a wider access to the Internet, have democratized the ‘right of entry’ for electronic musicians in Peru. However, lacking a complete intellectual and participative history of electronic music has kept Peruvian musicians from achieving a more conceptually ‘advanced’ understanding of the possibilities left to us by the electronic music revolution of the 1950s. Musicians in South America, and Peru in this case, interested in electronic music or related musical styles; find it difficult to buy the equipment necessary to replicate foreign musical trends that are based or dependent in technological gear,4 Also, if we consider that technological products could be linked to the social contexts in which they are imagined, constructed and used, many of these products would not belong in the Peruvian musical environment. Current academic discussions about issues related to the social, economical and technological relations between develop and developing countries are widespread. Postcolonial, subaltern, critical and cultural studies, as well as New Musicology, are some of the academic settings where the condition of non-elite social groups becomes an integral element in the construction and understanding of history. So far, still, the musical academic environments of Peru have lacked, for the most part, interest in these issues and maintain a conservative approach to academic training. Computer Music and related areas have yet to find a place in the Peruvian academic world. In recent years I have started to mention and discuss these concerns in academic

1 The term ‘history from below’, coined by Georges Lefebvre, is used, in this case, to refer to the condition of Peruvian electronic musicians in front of the electronic musical scenes and academic opportunities present only inside the developed world’s today.
2 The workshop, a summer class at PUCP, was moved afterwards to my home and became an open space where the students continue to experiment with circuit building.

1 For information on the history of Latin American electroacoustic music, see also Ricardo Dal Farra’s Latin American Electroacoustic Music Collection at: www.fondation-langlois.org/html/e/page.php?NumPage=556
2 While more research needs to be made about this, a similar case can probably be made about Ecuador and Bolivia, but not necessarily about Argentina and Chile.
territories such as the UCSD Computer Music program and the ICMC itself [1], [2], [3]. This work is an addition to my research and a progress report on both: the intention of including Peru in current academic international discussions, and the need to provide useful knowledge to the Peruvian musical communities (this is, electronic musicians, musical students, as well as institutions). Even as conversations related to Identity Politics or subaltern issues are many times misunderstood and its relevancy minimized at both sides of the cultural fence, the addition of Peru as a participant member of the electronic musical scene, both popular and academic, only enriches our understanding of the plural nature of the arts of sounds.

2. MADE IN PERU

The fabrication of machines for musical purposes is not entirely new in Peru. During the small craze of electronic experimental music at the beginning of the 2000s some homemade tone generators started to show up during different electronic performances. Musician Carlos García (a.k.a. Carlangas or Zetanga) was constructing in his room sound producing machines by recycling parts from old circuits bought in Paruro, a kind of parts cemetery market in the street of the same name in Downtown Lima [1]. While he did not perform with his tone generators he is single handedly responsible for producing a small sub culture within the electronic music community of Lima. Many musicians accustomed to buying their machines saw on these tone generators a new performative tool and an opportunity to develop new sounds and new styles for composition. These machines would free them from the use of loops, rhythm and traditional harmony. Also, the fact that these machines were made out of used parts added a Peruvian mystique to the instruments as objects in comparison to a mass produced instrument, and made it more affordable. Carlos called his company Zebranalogic and started to produce for the small experimental community of Lima in 2003 until moving to Sweden a few years later. Despite the fact that Zebranalogic does not build its machines with used parts anymore, the hand made logic was still and important concept for the company, at least until 2010. As the company’s blog states: “Zebranalogic specializes in the delicate elaboration of oscillators and pedals made entirely by hand, from the first to the last part.”

Following the example of Carlos Garcia, in 2005 Alfredo Aliaga started a company called Eclipse (currently known as Atomolabs) for the mass production of sound generators. If the first synth built by Alfredo was a traditional synth, keys included (see Figure 2), by the time he started his company he was producing sound generators in a style similar to Zebranalogic. Alfredo’s company, however, changed the conceptual direction taken by Zebranalogic by including fixtures like sequencers and by using new parts, moving away from the homemade street culture initially proposed by Carlos García’s machines, and towards a more professional look to Peruvian manufactured synths. During the last years Atomolabs have been building his machines again with a keyboard included, and they are as he calls them: compact analog synthesizer keyboards.

While Zebranalogic at its beginning had the Peruvian popular musician as its target client, after moving to Europe it has become a company mostly dedicated to the European and North American market, and the new generations of Peruvian musicians lack access to his instruments. Similarly, Atomolabs public target is not the Peruvian musician but the foreign market. Both companies have an interest in money making and in markets that are able to pay more for their products. While this is perfectly understandable for a company, for the Peruvian, politically charged, environment in which Zebranalogic emerged; this is a radical change of direction. Most of the initial buyers of Zebranalogic products had a strong political inclination, and viewed capitalism as an enemy, and money making enterprises negatively. The initial boom of electronic experimental popular music made with instruments made in Peru had declined considerably by the second part of the 2000s. It is important to mention that during this initial period the Alonso collective, a group of musicians mainly from the Cono Norte (the northern area of Lima), and dedicated to musical experimentation made several attempts to develop a culture of both experimentation and fabrication of instruments. This effort is still present though musicians like Gabriel Castillo Aguero and Rolando Apolo. Gabriel Castillo teaches workshops and Rolando Apolo builds oscillators for sale.

3. BEFORE THE WORKSHOP

Up until 2012, the fabrication of sound generators for musical performance had almost exclusively belonged to the popular realm. Construction, sales and teaching on the fabrication of sound circuits had been confined to the ghettos of the electronic music experimental suburban circles of underground popular musicians of Peru. The precarious condition that keeps the amateur popular musician from being able to declare knowledge, competence and credibility could be bridged after the ‘soon to be professional’ musician can understand his/her efforts and become a part of a mainstream knowledge musical environment. The companies fabricating the instruments were informal and the performers, untrained musicians. These ostracised subcultures, while an important contribution to musical experimentation in Peru, were not strong enough to produce a national scene. As a result there was an important decline on both fabrication and production of musical pieces made with Peruvian oscillators. After returning in 2010 to Peru, I decide that new strategies were necessary to develop a subculture of electronic musical experimentation and maintain alive the fabrication of electronic musical instruments. I considered that it was necessary for new musicians to learn how to use, for instance, a soldering gun, and how to fabricate a basic electronic musical tool. It was also essential for these musicians to expose themselves to the world of performance with these instruments. In addition to this, it was important to lobby for the inclusion of such topics in the curricula for the new students of music. If I managed to open this door and let sound circuit building enter the musical academic environment, new professional musicians would be exposed for the first time to a world that had been, and is still now, maintained within the boundaries of the very small circles of underground popular musicians of Peru. The precarious condition that keeps the amateur popular musician from being able to declare knowledge, competence and credibility could be bridged after the ‘soon to be professional’ musician can understand his/her efforts and become a part of a mainstream knowledge musical environment. The companies fabricating the instruments were informal and the performers, untrained musicians. These ostracised subcultures, while an important contribution to musical experimentation in Peru, were not strong enough to produce a national scene. As a result there was an important decline on both fabrication and production of musical pieces made with Peruvian oscillators. After returning in 2010 to Peru, I decide that new strategies were necessary to develop a subculture of electronic musical experimentation and maintain alive the fabrication of electronic musical instruments. I considered that it was necessary for new musicians to learn how to use, for instance, a soldering gun, and how to fabricate a basic electronic musical tool. It was also essential for these musicians to expose themselves to the world of performance with these instruments. In addition to this, it was important to lobby for the inclusion of such topics in the curricula for the new students of music. If I managed to open this door and let sound circuit building enter the musical academic environment, new professional musicians would be exposed for the first time to a world that had been, and is still now, maintained within the boundaries of the very small circles of underground popular musicians of Peru. The precarious condition that keeps the amateur popular musician from being able to declare knowledge, competence and credibility could be bridged after the ‘soon to be professional’ musician can understand his/her efforts and become a part of a mainstream knowledge musical environment. The companies fabricating the instruments were informal and the performers, untrained musicians. These ostracised subcultures, while an important contribution to musical experimentation in Peru, were not strong enough to produce a national scene. As a result there was an important decline on both fabrication and production of musical pieces made with Peruvian oscillators. After returning in 2010 to Peru, I decide that new strategies were necessary to develop a subculture of electronic musical experimentation and maintain alive the fabrication of electronic musical instruments. I considered that it was necessary for new musicians to learn how to use, for instance, a soldering gun, and how to fabricate a basic electronic musical tool. It was also essential for these musicians to expose themselves to the world of performance with these instruments. In addition to this, it was important to lobby for the inclusion of such topics in the curricula for the new students of music. If I managed to open this door and let sound circuit building enter the musical academic environment, new professional musicians would be exposed for the first time to a world that had been, and is still now, maintained within the boundaries of the very small circles of underground popular musicians of Peru. The precarious condition that keeps the amateur popular musician from being able to declare knowledge, competence and credibility could be bridged after the ‘soon to be professional’ musician can understand his/her efforts and become a part of a mainstream knowledge musical environment. The companies fabricating the instruments were informal and the performers, untrained musicians. These ostracised subcultures, while an important contribution to musical experimentation in Peru, were not strong enough to produce a national scene. As a result there was an important decline on both fabrication and production of musical pieces made with Peruvian oscillators. After returning in 2010 to Peru, I decide that new strategies were necessary to develop a subculture of electronic musical experimentation and maintain alive the fabrication of electronic musical instruments. I considered that it was
t erritories such as the UCSD Computer Music program and the ICMC itself [1], [2], [3]. This work is an addition to my research and a progress report on both: the intention of including Peru in current academic international discussions, and the need to provide useful knowledge to the Peruvian musical communities (this is, electronic musicians, musical students, as well as institutions). Even as conversations related to Identity Politics or subaltern issues are many times misunderstood and its relevancy minimized at both sides of the cultural fence, the addition of Peru as a participant member of the electronic musical scene, both popular and academic, only enriches our understanding of the plural nature of the arts of sounds.

2. MADE IN PERU

The fabrication of machines for musical purposes is not entirely new in Peru. During the small craze of electronic experimental music at the beginning of the 2000s some homemade tone generators started to show up at different electronic performances. Musician Carlos Garcia (a.k.a. Carlangas or Zetangas) was constructing parts from old circuits bought in Paruro, a kind of parts Garcia (´was constructing up at different electronic performances. Musician Carlos Garcia (a.k.a. Carlangas or Zetangas) was constructing in his room sound producing machines by recycling parts from old circuits bought in Paruro, a kind of parts cemetery market in the street of the same name in Downtown Lima [1]. While he did not perform with his tone generators he is single handedly responsible for producing a small sub culture within the electronic music community of Lima. Many musicians accustomed to buying their machines saw on these tone generators a new performative tool and an opportunity to develop new sounds and new styles for composition. These machines would free them from the use of loops, rhythm and traditional harmony. Also, the fact that these machines were made out of used parts added a Peruvian mystique to the instruments as objects in comparison to a mass produced instrument, and made it more affordable. Carlos called his company Zebranalogic and started to produce for the small experimental community of Lima in 2003 until moving to Sweden a few years later. Despite the fact that Zebranalogic does not build its machines with used parts anymore, the hand made logic was still and important concept for the company, at least until 2010. As the company’s blog states: “Zebranalogic specializes in the delicate elaboration of oscillators and pedals made entirely by hand, from the first to the last part.”

Following the example of Carlos Garcia, in 2005 Alfredo Alliaga started a company called Eclipse (currently known as Atomolabs) for the mass production of sound generators. If the first synth build by Alfredo was a traditional synth, keys included (see Figure 1), by the time he started his company he was producing sound generators in a style similar to Zebranalogic. Alfredo’s company, however, changed the conceptual direction taken by Zebranalogic by including fixtures like sequencing and by using new parts, moving away from the homemade street culture initially proposed by Carlos García’s machines, and towards a more professional look to Peruvian manufactured synths. During the last years Atomolabs have been building his machines again with a keyboard included, and they are as he calls them: compact analog synthesizer keyboards.1

While Zebranalogic at its beginning had the Peruvian popular musician as its target client, after moving to Europe it has become a company mostly dedicated to the European and North American market, and the new generations of Peruvian musicians lack access to his instruments. Similarly, Atomolabs public target is not

1 http://zebranologiclabs.blogspot.com/

2 As the Peruvian economy recovered from the crisis of the 1980s and early 1990s, conditions at Paruro Street have changed dramatically. Now few street vendors remain and stores mostly sale new products.

3. BEFORE THE WORKSHOP

Up until 2012, the fabrication of sound generators for musical performance had almost exclusively belonged to the popular realm. Construction, sales and teaching on the fabrication of sound circuits had been confined to the ghettos of the electronic music experimental subcultures of Peru, and in some particular cases to specific areas of Lima like the Cono Norte, as we mentioned before. The companies fabricating the instruments were informal and the performers, untrained musicians. These estranged subcultures, while an important contribution to musical experimentation in Peru, were not strong enough to produce a national scene. As a result there was an important decline on both fabrication and production of musical pieces made with Peruvian oscillators. After returning in 2010 to Peru, I decide that new strategies were necessary to develop a subculture of electronic musical tool. It was also essential for these musicians to expose themselves to the world of performance with these instruments. In addition to this, it was important to lobby for the inclusion of such topics in the curricula for the new students of music. If I managed to open this door and let sound circuit building enter the musical academic environment, new professional musicians would be exposed for the first time to a world that had been, and is still now, maintained within the boundaries of the very small circles of underground popular musicians of Peru. The precarious condition that keeps the amateur popular musician from being able to declare knowledge, competence and credibility could be bridged after the ‘soon to be professional’ musician can understand his/her efforts and become one of the constructors in the construction of a mainstream knowledge musical environment were technology and access work together in ways that make sense contextually in a Peruvian setting.

4. TALLER DE CIRCUITOS ELECTRONICOS SONOROS

In 2012, after becoming a professor at the Escuela de Musica of the Pontificia Universiatia Catolica del Peru I decided to test the waters by offering a summer workshop for the fabrication of electronic circuits for sound to the students entitled Taller de Circuitos Electrónicos Sonoros 1. Since this school was one of the first two in the country to bring us a unique compromise

Figure 1. Performing with a Zebranalogic oscillator, Tijuana, Mexico.

Figure 2. Max Salas performing with the first Eclipse (top)

Figure 3. Oscillators by Rolando Apolo (2013)

Figure 4. Advertisement for the workshop

1. “Zebranologic specializes in the delicate elaboration of oscillators and pedals made entirely by hand, from the first to the last part.”

2. As the Peruvian economy recovered from the crisis of the 1980s and early 1990s, conditions at Paruro Street have changed dramatically. Now few street vendors remain and stores mostly sale new products.

between the world of popular music performance and that of academic training at a university level, I thought it was the perfect environment to attempt to open the door to the basic world of sound producing circuitry to people trained mostly as performers. This workshop, I hoped, would allow them to develop a new palette of sounds and techniques for musical composition, as well as to allow them to have a better understanding of the world of electronic music and its sound sources.

The workshop was well received, and about 12 students enrolled. Few of them had any knowledge of circuit building, and none of them belonged to Lima’s academic training at a university level, I thought between the world of popular music performance and the arts. As we wait for the results of this experiment, we hope for the new generations of musical students to gain an understanding of the culture of the sound arts outside of traditional instrumentation, orchestration, harmony and counterpoint; and to understand the inner workings and possibilities offered by technology in the development of physical and conceptual tools for music making.

This initiative is far from having achieved the generation of any new subcultures for the construction and performance of electronic music. Its intentions were to open new possibilities for the students of music, and making spaces for what might be considered outside of the academy as standard or traditional, but as non-traditional sound arts in the academic environment of Peru. This might sound as nothing new in countries where however small, experimental communities of well trained electronic musicians have access to the tools of their trade, and where musicians can encounter training on electronic music at the university level. In Peru, it is common now to see this kind of workshops taught at cultural centres and art centres, but not at the academic institutions of higher education that work with performance and the arts. As we wait for the results of this experiment, we hope for the new generations of musical students to gain an understanding of the culture of the sound arts outside of traditional instrumentation, orchestration, harmony and counterpoint; and to understand the inner workings and possibilities offered by technology in the development of physical and conceptual tools for music making.

5. CONCLUSIONS

Kronos is a reactive-functional programming environment for musical signal processing. It is designed for musicians and music technologists who seek custom signal processing solutions, as well as developers of audio components.

The chief contributions of the environment include a type-based polymorphic system which allows for processing modules to automatically adapt to incoming signal types. An unified signal model provides a programming paradigm that works identically on audio, MIDI, OSC and user interface control signals. Together, these features enable a more compact software library, as user-facing primitives are less numerous and able to function as expected based on the program context. This reduces the vocabulary required to learn programming.

This paper describes the main algorithmic contributions to the field, as well as recent research into improving compile performance when dealing with block-based processes and massive vectors.

Kronos is a reactive-functional programming language[8] for signal processing tasks. It aims to be able to model musical signal processors with simple, expressive syntax and very high performance. It consists of a programming language specification and a reference implementation that contains a just in time compiler along with a signal I/O layer supporting audio, OSC[9] and MIDI.

The founding principle of this research project is to reduce the vocabulary of a musical programming language by promoting signal processor design patterns to integrated language features. For example, the environment automates signal update rates, eradicating the need for similar but separate processors for audio and control rate tasks.

Further, signals can have associated type semantics. This allows an audio processor to configure itself to suit an incoming signal, such as mono or multichannel, or varying sample formats. Together, these language features serve to make processors more flexible, thus requiring a smaller set of them.

This paper describes the state of the Kronos compiler suite as it nears production maturity. The state of the freely available beta implementation is discussed, along with issues that needed to be addressed in recent development work – specifically dealing with support for massive vectors and their interaction with heterogeneous signal rates.

ABSTRACT

Kronos is a reactive-functional programming environment for musical signal processing. It is designed for musicians and music technologists who seek custom signal processing solutions, as well as developers of audio components.

The chief contributions of the environment include a type-based polymorphic system which allows for processing modules to automatically adapt to incoming signal types. An unified signal model provides a programming paradigm that works identically on audio, MIDI, OSC and user interface control signals. Together, these features enable a more compact software library, as user-facing primitives are less numerous and able to function as expected based on the program context. This reduces the vocabulary required to learn programming.

This paper describes the main algorithmic contributions to the field, as well as recent research into improving compile performance when dealing with block-based processes and massive vectors.

1. INTRODUCTION

Kronos is a reactive-functional programming language[8] for signal processing tasks. It aims to be able to model musical signal processors with simple, expressive syntax and very high performance. It consists of a programming language specification and a reference implementation that contains a just in time compiler along with a signal I/O layer supporting audio, OSC[9] and MIDI.

The founding principle of this research project is to reduce the vocabulary of a musical programming language by promoting signal processor design patterns to integrated language features. For example, the environment automates signal update rates, eradicating the need for similar but separate processors for audio and control rate tasks.

Further, signals can have associated type semantics. This allows an audio processor to configure itself to suit an incoming signal, such as mono or multichannel, or varying sample formats. Together, these language features serve to make processors more flexible, thus requiring a smaller set of them.

This paper describes the state of the Kronos compiler suite as it nears production maturity. The state of the freely available beta implementation is discussed, along with issues that needed to be addressed in recent development work – specifically dealing with support for massive vectors and their interaction with heterogeneous signal rates.

As its main contribution, this paper presents an algorithm for reactive factorization of arbitrary signal processors. The algorithm is able to perform automatic signal rate optimizations without user intervention or effort, handling audio, MIDI and OSC signals with a unified set of semantics. The method is demonstrated via Kronos, but is applicable to any programming language or a system where data dependencies can be reliably reasoned about. Secondly, this method is discussed in the context of heterogeneous signal rates in large vector processing, such as those that arise when connecting huge sensor arrays to wide ugen banks.

This paper is organized as follows; in Section 2, Kronos Language Overview, the proposed language and compiler are briefly discussed for context. Section 3 describes an algorithm that can perform intelligent signal rate factorization on arbitrary algorithms. Section 4, Novel Features, discusses in detail the most recent developments. Finally, the conclusions are presented in Section 5.

2. KRONOS LANGUAGE OVERVIEW

Kronos programs can be constructed as either textual source code files or graphical patches. The functional model is well suited for both representations, as functional programs are essentially data flow graphs.

2.1. Functional Programming for Audio

Most of a Kronos program consists of function definitions, as is to be expected from a functional programming language. Functions are compositions of other functions, and each function models a signal processing stage. Per usual, functions are first class and can be passed as inputs to other, higher order functions.

This allows traditional functional programming stables such as map, demonstrated in Figure 1. In the example, a higher order function called Algorithm.Map receives from the right hand side a set of control signals, and applies a transformation specified on the left hand side, where each frequency value becomes an oscillator at that frequency. For a thorough discussion, the reader is referred to previous work[1].

2.2. Types and Polymorphism as Graph Generation

Kronos allows functions to attach type semantics to signals. Therefore the system can differentiate between, say, a stereo audio signal and a stream of complex numbers. In each case, a data element consists of two real numbers, but the semantic meaning is different. This is accomplished
between the world of popular music performance and that of academic training at a university level, I thought it was the perfect environment to attempt to open the door to the basic world of sound producing circuitry to people trained mostly as performers. This workshop, I hoped, would allow them to develop a new palette of sounds and techniques for musical composition, as well as to allow them to have a better understanding of the world of electronic music and its sound sources.

The workshop was well received, and about 12 students enrolled. Few of them had any knowledge of circuit building, and none of them belonged to Lima’s enrolment. Few of them had any knowledge of electronic music and its sound sources. This initiative is far from having achieved the generation of any new subcultures for the construction and performance of electronic music. Its intentions were to open new possibilities for the students of music, and making spaces for what might be considered outside of the country as standard or traditional, but as non-traditional sound arts in the academic environment of Peru. This might sound as nothing new in countries where however small, experimental communities of well trained electronic musicians have access to the tools of their trade, and where musicians can encounter training on electronic music at the university level. In Peru, it is common now to see this kind of workshops taught at cultural centres and art centres, but not at the academic institutions of higher education that work with performance and the arts. As we wait for the results of this experiment, we hope for the new generations of musical students to gain an understanding of the culture of the sound arts outside of traditional instrumentation, orchestration, harmony and counterpoint; and to understand the inner workings and possibilities offered by technology in the development of physical and conceptual tools for music making.

Kronos is a reactive-functional programming environment for musical signal processing. It is designed for musicians and music technologists who seek custom signal processing solutions, as well as developers of audio components.

The chief contributions of the environment include a type-based polymorphic system which allows for processing modules to automatically adapt to incoming signal types. An unified signal model provides a programming paradigm that works identically on audio, MIDI, OSC and user interface control signals. Together, these features enable a more compact software library, as user-facing primitives are less numerous and able to function as expected based on the program context. This reduces the vocabulary required to learn programming.

This paper describes the main algorithmic contributions to the field, as well as recent research into improving compile performance when dealing with block-based processes and massive vectors.

1. INTRODUCTION

Kronos is a reactive-functional programming language[8] for signal processing tasks. It aims to be able to model musical signal processors with simple, expressive syntax and very high performance. It consists of a programming language specification and a reference implementation that contains a just in time compiler along with a runtime that supports distributed computation. Kronos is a reactive-functional programming environment for musical signal processing.

The chief contributions of the environment include a type-based polymorphic system which allows for processing modules to automatically adapt to incoming signal types. An unified signal model provides a programming paradigm that works identically on audio, MIDI, OSC and user interface control signals. Together, these features enable a more compact software library, as user-facing primitives are less numerous and able to function as expected based on the program context. This reduces the vocabulary required to learn programming.

This paper describes the main algorithmic contributions to the field, as well as recent research into improving compile performance when dealing with block-based processes and massive vectors.

6. REFERENCES


5. CONCLUSIONS

This paper is organized as follows; in Section 2, Kronos Language Overview, the proposed language and compiler are briefly discussed for context. Section 3 describes an algorithm that can perform intelligent signal rate factorization on arbitrary algorithms. Section 4, Novel Features, discusses in detail the most recent developments. Finally, the conclusions are presented in Section 5.

2. KRONOS LANGUAGE OVERVIEW

Kronos programs can be constructed as either textual source code files or graphical patches. The functional model is well suited for both representations, as functional programs are essentially data flow graphs.

2.1. Functional Programming for Audio

Most of a Kronos program consists of function definitions, as is to be expected from a functional programming language. Functions are compositions of other functions, and each function models a signal processing stage. Per usual, functions are first class and can be passed as inputs to other, higher order functions.

This allows traditional functional programming stables such as map, demonstrated in Figure 1. In the example, a higher order function called Algorithm:Map receives from the right hand side a set of control signals, and applies a transformation specified on the left hand side, where each frequency value becomes an oscillator.

This paper describes the state of the Kronos compiler suite as it nears production maturity. The state of the freely available beta implementation is discussed, along with issues that needed to be addressed in recent development work – specifically dealing with support for massive vectors and their interaction with heterogeneous signal rates.

As its main contribution, this paper presents an algorithm for reactive factorization of arbitrary signal processors. The algorithm is able to perform automatic signal rate optimizations without user intervention or effort, handling audio, MIDI and OSC signals with a unified set of semantics. The method is demonstrated via Kronos, but is applicable to any programming language or a system where data dependencies can be reliably reasoned about. Secondly, this method is discussed in the context of heterogeneous signal rates in large vector processing, such as those that arise when connecting huge sensor arrays to huge computer banks.

This paper is organized as follows; in Section 2, Kronos Language Overview, the proposed language and compiler are briefly discussed for context. Section 3 describes an algorithm that can perform intelligent signal rate factorization on arbitrary algorithms. Section 4, Novel Features, discusses in detail the most recent developments. Finally, the conclusions are presented in Section 5.

2.1. Functional Programming for Audio

Most of a Kronos program consists of function definitions, as is to be expected from a functional programming language. Functions are compositions of other functions, and each function models a signal processing stage. Per usual, functions are first class and can be passed as inputs to other, higher order functions.

This allows traditional functional programming stables such as map, demonstrated in Figure 1. In the example, a higher order function called Algorithm:Map receives from the right hand side a set of control signals, and applies a transformation specified on the left hand side, where each frequency value becomes an oscillator at that frequency. For a thorough discussion, the reader is referred to previous work[1].

2.2. Types and Polymorphism as Graph Generation

Kronos allows functions to attach type semantics to signals. Therefore the system can differentiate between, say, a stereo audio signal and a stream of complex numbers. In each case, a data element consists of two real numbers, but the semantic meaning is different. This is accomplished...
3. REACTIVE FUNCTIONAL AS THE UNIVERSAL SIGNAL MODEL

3.1. Dataflow Analysis

Given an arbitrary user program, all signal data flows should be able to be reliably detected. For functional programming languages such as Kronos or Faust[7], this is trivial, as all data flows are explicit. The presence of any implicit data flows, such as the global buses in systems like SuperCollider[5] can pose problems for the data flow analysis.

3.2. Reactive Clock Propagation

The general assumption is that a node is active whenever any of its upstream nodes are active. This is because logical and arithmetic operations will need to be recomputed whenever any of their inputs change. However, this is not true of all nodes. If an operation merely combines unchanged signals into a vectored signal, it is appropriate to maintain separate clocking records for the components of the vectorized signal rather than having all the component clocks drive the entire vector. When the vector is unpacked later, subsequent operations will only join the activation states of the component signals they access.

Similar logic applies to function calls. Since many processors manifest naturally as functions that contain mixed rate signal paths, all function inputs should preferably have distinct activation states.

3.3. Stateful Operations and Clock

The logic outlined in section 3.2 works well for strictly functional nodes — that means all operations whose output is uniquely determined by their inputs rather than any state or memory. However, state and memory are important for many DSP algorithms such as filters and delays. Like Faust[7], Kronos deals with them by promoting them to language primitives. Unit delays and ring buffers can be used to peek into the signal graph from the previous time they were activated. This yields an elegant syntax for delay operations while maintaining strict functional style within each update frame.

For strictly functional nodes, activation is merely an optimization. For stateful operations such as delays, it becomes a question of algorithmic correctness. Therefore it is important that stateful nodes are not activated by any springs other than the ones that define their desired clock rate. For example, the unit delays in a filter should not be activated by the user interface elements that control their coefficients to avoid having the signal clock disrupted by additional update frames from the user interface. A resonator filter with a signal input and two control parameters freq and radius is shown in Figure 2. The nodes that see several clock sources in their upstream are indicated with a dashed border. Since these include the two unit delay primitives, it is unclear which clock should determine the length of the unit delay.

3.3.1. Clock Priority

The clocking ambiguities can be resolved by assigning priorities to the springs that drive the signal graph. This means that whenever a node is activated by multiple springs, some springs can preclude others.

The priority can be implemented by a strict-weak ordering criteria, where individual spring pairs can either have an ordered or an unordered relation. Ordered pairs will only keep the dominant spring, while unordered springs can coexist and both activate a node. The priority system is shown in Figure 3. The audio clock dominates the control signal clocks. Wires that carry control signals are shown hollow, while audio signal wires are shown solid black. This allows the audio clock to control the unit delays over any clock sources of lesser priority.

In general, data streams with a regular update interval such as audio should dominate event based streams such as MIDI or user interface elements.

3.3.2. Dynamic Clocking and Event Streams

The default reactivity scheme with appropriate spring priorities will result in sensible clocking behavior in most situations. However, sometimes it may be necessary to override the default clock propagation rules.

As an example, consider an audio analyzer, more specifically, the simple example of a transient detector. This process has an audio input and an event stream output. The output is activated by the input, but only sometimes; depending on whether the algorithm decides a transient occurred during that particular activation.

This can be implemented by a clock gate primitive. It allows a conditional masking of signal activation with dynamic activation, the reactive system can be used to model event streams — signals that do not have a regular update interval. This accomplishes many tasks that are handled with branching in procedural languages, and in the end results in similar machine code. A simple example is shown in Figure 4. The Reactive : Gate primitive takes a truth value and a signal, inhibiting any clock updates from the signal when the truth value is false. This allows an analysis algorithm to produce an event stream from features detected from an audio stream.

...
model imposes so little overhead that it is entirely suitable. The entire signal graph is synchronous and the reactive update allows the system to handle sampled audio streams and date events within continuous “staircase” signals. This means that whenever a node is activated by multiple springs, their activation states can coexist and both activate a node. The priority system is shown in Figure 3. The audio clock dominates the control signal clocks. Wires that carry control signals are shown hollow, while audio signal wires are shown solid black. This allows the audio clock to control the unit delays, while the audio clock drives the unit delays.

The priority can be implemented by a clock gate primitive. It allows a conditional masking of signal activation. With dynamic activation, the reactive system can be used to model event streams – signals that do not have a regular update interval. This accomplishes many tasks that are handled with branching in procedural languages, and in the end results in similar machine code. A simple example is shown in Figure 4. The Reactive : Gate primitive takes a truth value and a signal, inhibiting any clock updates from the signal when the truth value is false. This allows an analysis algorithm to produce an event stream from features detected from an audio stream.

3.3.2. Dynamic Clocking and Event Streams

The default reactivity scheme with appropriate spring priorities will result in sensible clocking behavior in most situations. However, sometimes it may be necessary to override the default clock propagation rules. As an example, consider an audio analyzer, more specifically the simple example of a transient detector. This process has an audio input and an event stream output. The output is activated by the input, but only sometimes; depending on whether the algorithm decides a transient occurred during that particular activation.

This can be implemented by a clock gate primitive. It allows a conditional masking of signal activation. With dynamic activation, the reactive system can be used to model event streams – signals that do not have a regular update interval. This accomplishes many tasks that are handled with branching in procedural languages, and in the end results in similar machine code. A simple example is shown in Figure 4. The Reactive : Gate primitive takes a truth value and a signal, inhibiting any clock updates from the signal when the truth value is false. This allows an analysis algorithm to produce an event stream from features detected from an audio stream.

3.3.1. Clock Priority

The clocking ambiguities can be resolved by assigning priorities to the springs that drive the signal graph. This means that whenever a node is activated by multiple springs, some springs can preclude others. The priority system is shown in Figure 3. The audio clock dominates the control signal clocks. Wires that carry control signals are shown hollow, while audio signal wires are shown solid black. This allows the audio clock to control the unit delays over any clock sources of lesser priority.

In general, data streams with a regular update interval such as audio should dominate event based streams such as MIDI or user interface elements.
### 3.3.3. Upsampling and Decimation

For generating several sequential activations from a single external activation, an upsampling mechanism is needed. A special purpose reactive node can be inserted in the signal graph to multiply the incoming clock rate by a rational fraction. This allows for both up- and downsampling of the incoming signal by a constant factor. For reactive priority resolution, clock multipliers sourced from the same external clock are considered unordered.

To synchronously schedule a number of different rational multiplies of an external clock, it is necessary to construct a super-clock that ticks whenever any of the multiplier clocks might tick. This means that the super-clock multiplier must be divisible by all multiplier numerators, yet be as small as possible. This can be accomplished by combining the numerators one by one into a reduction variable \( S \) with the formula in Equation (1):

\[
f(a,b) = \frac{ab}{\text{gcd}(a,b)}
\]

To construct an activation sequence from an upsam-pled external clock, let us consider the sequence of \( S \) super-clock ticks it triggers. Consider the super-clock multiplier of \( S \) and a multiplier clock \( \frac{1}{P} \). In terms of the super-clock period, the multiplier ticks at \( \frac{1}{ab} \). This is guaranteed to simplify to \( \frac{1}{k} \), where \( P \) is an integer— the period of the multiplier clock in super-clock ticks.

Within a period of \( S \) super-clock ticks, the multiplier clock could potentially activate once every \( \frac{1}{P} \) ticks. In the case of \( P = \text{gcd}(S,P) \) the activation pattern is deterministic. Otherwise, the activation pattern is different for each tick of the external clock, and counters must be utilized to determine which ticks are genuine activations to maintain the period \( P \). An activation pattern is demonstrated in Figure 1.

This system guarantees exact and synchronous timing for all rational fraction multipliers of a signal clock. For performance reasons, some clock jitter can be permitted to reduce the number of required activation states. This can be done by reducing the number of adjacent super-clock ticks. As long as the merge width is less than the smallest \( P \) in the clock system, the clocks maintain a correct average tick frequency with small momentary fluctuations. An example of an activation state matrix is shown in Figure 1. This table shows a clock and its multiplex by three and four, and the resulting activation combinations per super-clock tick.

### 3.3.4. Multiplexing and Demultiplexing

The synchronous multirate clock system can be leveraged to provide oversampled or subsampled signal paths, but also several less intuitive applications.

To implement a multiplexing or a buffering stage, a ring buffer can be combined with a signal rate divider. If the ring buffer contents are output at a signal rate divided by the length of the buffer, a buffering with no overlap is created. Dividing the signal clock by half of the buffer length yields a \( 50\% \) overlap, and so on.

The opposite can be achieved by multiplying the clock of a vectorized signal and indexing the vector with a ramp that has a period of a non-multiplied tick. This can be used for de-buffering a signal or canonical inserter/zero up-sampling.

### 3.4. Current Implementation in Kronos

The reactive system is currently implemented in Kronos as an extra pass between type specialization and machine code generation. An overview of the compilation process is described in Table 2.

The reactive analysis happens relatively early in the compiler pipeline, which results in some added complexity. For example, when a function is factored into several activation states, the factorizer must change some of the types inferred by the specialization pass to maintain graph consistency when splitting user functions to different activation states.

Further, the complexity of all the passes depends heavily on the data. During the specialization pass, a typed function is generated for each different activation state. For recursive call sequences, this means each iteration of the recursion. While the code generator is able to fold these back into loops, compilation time grows quickly as vector sizes increase. This hardwires matters for the original purpose of the compactor, as most of the vector sizes were in orders of tens or hundreds, representing parallel I/O banks.

However, the multirate processing and multiplexing detailed in Section 3.3.4 are well suited for block processes, such as FFT, which naturally need vector sizes from several thousand to orders of magnitude upwards. Such processes can currently cause compilation times from tens of seconds to minutes, which is not desirable for a quick development cycle and immediate feedback. The newest developments on Kronos focus on, amongst other things, decoupling compilation time from data complexity. The relationship of these optimizations to reactive factorization is explored in the following Section 4.

### 4. NEW DEVELOPMENTS

Before Kronos reaches production maturity, a final rewrite is underway to simplify the overall design, improve the features and optimize performance. This section discusses the improvements over the beta implementation.

#### 4.1. Sequence Recognition

Instead of specializing a recursive function separately for every iteration, it is desirable to detect such sequences as early as possible. The new version of the Kronos compiler has a special analysis algorithm for such sequences.

In the case of a recursion, the evolution of the induction variables is analyzed. Because Kronos is type deter-
mindistic, as explained in Section 2.2.1, the overload reso-
lution is uniquely determined by the types of the induction variables.

In the easy case, an induction variable retains the same type between recursions. In addition, the analyzer can handle homogeneous vectors that grow or shrink and com-
pile time constants with simple arithmetic evolutions. Such evolutions are detected and the evolution rules are lifted to the user program. The analyzer then attempts to continue these into recurrence relations, which can be solved in closed form. Successful analysis means that a sequence will have identical overload resolutions for \( N \) iterations, enabling the internal representation of the program to encode this efficiently.

Recognized sequences are then compiled in constant time, independent from the size of data vectors involved. This is in contrast to Kronost Beta, which did this in linear time. In practice, the analyzer works for functions that iterate over vectors of homogenous values as well as simple induction variables. It is enough to efficiently de-
tect and encode common functional idioms such as map, reduce, unfold and zip, provided their argument lists are homogenous.

#### 4.2. New LLVM Backend

As a part of Kronos redesign, a decision was made to push the reactive factorization further back in the com-

### Table 1. Activation State Matrix

<table>
<thead>
<tr>
<th>Clock</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock ( \times 3 )</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Clock ( \times 4 )</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Table 2. Compilation passes performed by Kronos Beta

<table>
<thead>
<tr>
<th>Pass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Specialization</td>
<td>Generic functions to typed functions and overload resolution</td>
</tr>
<tr>
<td>2. Reactively</td>
<td>Reactive analysis and splitting of typed functions to different activation states</td>
</tr>
<tr>
<td>3. Side Effects</td>
<td>Functional data flows to pointer side effects</td>
</tr>
<tr>
<td>4. Codegen</td>
<td>Selection and scheduling of x86 machine instructions</td>
</tr>
</tbody>
</table>

### Table 3. Compilation passes performed by Kronos Final

<table>
<thead>
<tr>
<th>Pass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Specialization</td>
<td>Generic functions to typed functions and overload resolution</td>
</tr>
<tr>
<td>2. Reactively</td>
<td>Reactive analysis</td>
</tr>
<tr>
<td>3. Copy Fusion</td>
<td>Dataflow analysis and copy elimination</td>
</tr>
<tr>
<td>4. Side Effects</td>
<td>Functional data flows to pointer side effects</td>
</tr>
<tr>
<td>5. LLVM Codegen</td>
<td>Generating LLVM IR with a specific activation state</td>
</tr>
<tr>
<td>6. LLVM Optimization</td>
<td>Optimizing LLVM IR</td>
</tr>
<tr>
<td>7. Native Codegen</td>
<td>Selection and scheduling of x86 machine instructions</td>
</tr>
</tbody>
</table>
### Table 1. Activation State Matrix

<table>
<thead>
<tr>
<th>Clock</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock × 3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Clock × 4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Table 2. Compilation passes performed by Kronos Beta

<table>
<thead>
<tr>
<th>Pass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Specialization</td>
<td>Generic functions to typed functions and overload resolution</td>
</tr>
<tr>
<td>2. Reactivity</td>
<td>Reactive analysis and splitting of typed functions to different activation states</td>
</tr>
<tr>
<td>3. Side Effects</td>
<td>Functional data flows to pointer side effects</td>
</tr>
<tr>
<td>4. Codegen</td>
<td>Selection and scheduling of x86 machine instructions</td>
</tr>
</tbody>
</table>

### Table 3. Compilation passes performed by Kronos Final

<table>
<thead>
<tr>
<th>Pass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Specialization</td>
<td>Generic functions to typed functions and overload resolution</td>
</tr>
<tr>
<td>2. Reactivity</td>
<td>Reactive analysis</td>
</tr>
<tr>
<td>3. Copy Elision</td>
<td>Dataflow analysis and copy elision</td>
</tr>
<tr>
<td>4. Side Effects</td>
<td>Functional data flows to pointer side effects</td>
</tr>
<tr>
<td>5. LIVM Codegen</td>
<td>Generating LIVM IR with a specific activation state</td>
</tr>
<tr>
<td>6. LIVM Optimization</td>
<td>Optimizing LIVM IR</td>
</tr>
<tr>
<td>7. Native Codegen</td>
<td>Selection and scheduling of x86 machine instructions</td>
</tr>
</tbody>
</table>

### 3.3.3. Upsampling and Decimation

For generating several sequential activations from a single external activation, an upsampling mechanism is needed. A special purpose reactive register can be inserted in the signal graph to multiply the incoming clock rate by a rational fraction. This allows for both up- and down-sampling of the incoming signal by a constant factor. For reactive priority resolution, clock multipliers sourced from the same external clock are considered unordered.

To synchronize a number of different rational multiplies of an external clock, it is necessary to construct a super-clock that ticks whenever any of the multipliers might tick. This means that the super-clock multiplier must be divisible by all multiplier numerators, yet be as small as possible. This can be accomplished by combining the numerators one by one into a reduction variable \( S \) with the formula in Equation (1)

\[
f(a, b) = \frac{ab}{\gcd(a, b)}
\]

To construct an activation sequence from an upsampled external clock, let us consider the sequence of \( S \) super-clock ticks it triggers. Consider the super-clock multiplier of \( S \) and a multiplier clock \( \frac{1}{b} \). In terms of the super-clock period, the multiplier ticks at \( \frac{1}{b} \). This is guaranteed to simplify to \( \frac{1}{k} \), where \( P \) is an integer – the period of the multiplier clock in super-clock ticks.

Within a period of \( P \) super-clock ticks, the multiplier clock could potentially activate once every \( \gcd(S, P) \) ticks. In the case of \( P = \gcd(S, P) \) the activation pattern is deterministic. Otherwise, the activation pattern is different for every tick of the external clock, and counters must be utilized to determine which ticks are genuine activations to maintain the period \( P \). An activation pattern is demonstrated in Figure 1.

This system guarantees exact and synchronous timing for all rational fraction multipliers of a signal clock. For performance reasons, some clock jitter can be permitted to reduce the number of required activation states. This can be done by raising the number of adjacent super-clock ticks. As long as the merge width is less than the smallest \( P \) in the clock system, the clocks maintain a correct average tick frequency with small momentary fluctuations. An example of an activation state matrix is shown in Figure 1. This table shows a clock and its multiplies by three and four, and the resulting activation combinations per super-clock tick.

### 3.3.4. Multiplexing and Demultiplexing

The synchronous multiplexing clock system can be leveraged to provide oversampled or subsampled signal paths, but also several less intuitive applications.

To implement a multiplexing or a buffering stage, a ring buffer can be combined with a signal rate divider. If the ring buffer contents are output at a signal rate divided by the length of the buffer, a buffering with no overlap is created. Dividing the signal clock by half of the buffer length yields a 50\% overlap, and so on.

The opposite can be achieved by multiplying the clock of a vectorized signal and indexing the vector with a ramp that has a period of a non-multipled tick. This can be used for de-buffering a signal or canonical inser-t-zero upsampling.

### 3.4. Current Implementation in Kronos

The reactive system is currently implemented in Kronos Beta as an extra pass between type specialization and machine code generation. An overview of the compilation process is described in Table 2.

The reactive analysis happens relatively early in the compiler pipeline, which results in some added complexity. For example, when a function is factored into several activation states, the factorizer must change some of the types inferred by the specialization pass to maintain graph consistency when splitting user functions to different activation states. Further, the complexity of all the passes depends heavily on the data. During the specialization pass, a typed function is generated for each different argument type. For recursive call sequences, this means each iteration of the recursion. While the code generator is able to fold these back into loops, compilation time grows quickly as vector sizes increase. This hardly matters for the original purpose of the compiler, as most of the vector sizes were in orders of tens or hundreds, representing parallel ugen banks.

However, the mutate processing and multiplexing detailed in Section 3.3.4 are well suited for block processes, such as FFT, which naturally need vector sizes from several thousand to orders of magnitude upwards. Such processes can currently cause compilation times from tens of seconds to minutes, which is not desirable for a quick development cycle and immediate feedback. The newest developments on Kronos focus on, amongst other things, decoupling compilation time from data complexity. The relationship of these optimizations to reactive factorization is explored in the following Section 4.

### 4. NEW DEVELOPMENTS

Before Kronos reaches production maturity, a final rewrite is underway to simplify the overall design, improve the features and optimize performance. This section discusses the improvements over the beta implementation.

#### 4.1. Sequence Recognition

Instead of specializing a recursive function separately for every iteration, it is desirable to detect such sequences as early as possible. The new version of the Kronos compiler has a special analysis algorithm for such sequences.

In the case of a recursion, the evolution of the induction variables is analyzed. Because Kronos is type deterministic, as explained in Section 2.2, the overload resolution is uniquely determined by the types of the induction variables.

In the easy case, an induction variable retains the same type between recursions. In addition, the analyzer can handle homogeneous vectors that grow or shrink and compute time constants with simple arithmetic evolutions. Such evolutions are detected and the evolution rules are lifted or separated from the user program. The analyzer then attempts to detect these into recurrence relations, which can be solved in closed form. Successful analysis means that a sequence will have identical overload resolutions for \( N \) iterations, enabling the internal representation of the program to encode this efficiently.

Recognized sequences are thus compiled in constant time, independent from the size of data vectors involved. This is in contrast to Kronost Beta, which did this in linear time. In practice, the analyzer works for functions that iterate over vectors of homogenous values as well as simple induction variables. It is enough to efficiently detect and encode common functional idioms such as map, reduce, unfold and zip, provided their argument lists are homogenous.

#### 4.2. New LIVM Backend

As a part of Kronos redesign, a decision was made to push the reactive factorization further back in the compilation pipeline. Instead of operating in typed Kronos functions, it would operate on a low level code representation, merely removing code that was irrelevant for the activation state at hand.

This requires some optimization passes after factorization, as well as an intermediate representation between Kronos syntax trees and machine code. Both of these are realized by the widely used LIVM, a compiler component capable of abstracting various low level instruction sets. LIVM includes both a well designed intermediate representation as well as industry strength optimization passes. As an extra benefit, Kronos can target a number of machine architectures without additional development effort.

In short, the refactored compiler includes more compilation passes than the beta version, but each pass is simple. In addition, the LIVM project provides several of them. The passes are detailed in Table 3, contrasted to Table 2.

#### 4.3. Reactive Factoring of Sequences

The newly developed sequence recognition creates some new challenges for reactive factorization. The basic functions of the two passes contrast; the sequence analysis combines several user functions into a compact representation for compile time performance reasons. The reactive factorization, in contrast, splits user functions in order to improve run time performance.

A typical optimization opportunity that requires cooperation between reactive analysis and sequence recognition would be a bank of filters controlled by a number of different control sources. Ideally, we want to maintain an efficient sequence representation for the audio section of those filters, while only recomputing coefficients when there’s input from one of the control sources.

If a global control clock is defined that is shared between the control sources, no special actions are needed. Since all iterations of the sequence see identical clocks at the input side, they will be identically factored. Thus, the sequence iteration can be analyzed once, and the analysis is valid for all the iterations. The LIVM Codegen sees a loop, and depending on the activation state it will filter out different parts of the loop and provide the plumbing between clock regions.

Forcing all control signals to tick at a global control rate could make the patches easier to compile efficiently. However, this breaks the unified signal model. A central motivation of the reactive model is to treat event-based and streaming signals in the same way. The global control clock is mandated, signal models such as MSTD streams could no longer maintain the natural relationship between
an incoming event and a clock tick. Therefore, event streams such as the user interface and external control interfaces should be considered when designing the sequence factorizer.

4.3.1. Heterogenous Clock Rates in Sequences

Consider a case where each control signal is associated with a different clock rate source. We would like to maintain the audio sequence as a section, but this is no longer possible for the control section, as each iteration responds to a different activation state. In this case, the reactive factorization must compute a distinct activation state for each iteration of the sequence. If there is a section of the iteration with an invariant activation state, this section can be factored into a sequence of its own.

Such sequence factorization can be achieved via Hylomorphism, which is the generalization of recursive sequences. The theory is beyond the scope of this article, but based on the methods in literature[6], any sequence can be split into a series of two or more sequences. In audio context, this can be leveraged so that any activation-invariant code as possible can be separated into a sequence that can be maintained throughout the compilation pipeline. The activation-invariant sections must then be wholly unrolled. This allows the codegen with activation state filtering to produce highly efficient code.

5. CONCLUSIONS

This paper presented an overview of Kronos, a musical signal processing language, as well as the design of its reactive signal model. Kronos is designed to increase the flexibility and generality of signal processing primitives, limiting the vocabulary that is requisite for programming. This is accomplished chiefly via the type system and the polymorphic programming method as well as the unified signal model.

The reactive factorization algorithm presented in this paper can remove the distinction between events, messages, control signals and audio signals. Each signal type can be handled with the same set of primitives, yet the code generator is able to leverage automatically deduced signal metadata to optimize the resulting program.

The concepts described in this paper are implemented in a prototype version of the Kronos compiler which is freely available along with a visual, patching-based interface[2]. For a final version, the compiler is currently being redesigned, scheduled to be released by the summer of 2013. The compiler will be available with either a GPL3 or a commercial license.

Some new developments of a redesigned compiler were detailed, including strategies for handling massive vector banks. As Kronos aims to be an environment where compilation should respond as quickly as a play button, this is critical for the feasibility of these applications.

As the compiler technology is reaching maturity, further research will be focused on building extensive, adaptable and learnable libraries of signal processing primitives for the system. Interaction with various software platforms is planned. This takes the form of OSC communication as well as code generation – Kronos can be used to build binary format extensions, which can be used as plugins or extensions to other systems. LLVM integration opens up the possibility of code generation for DSP and embedded devices. Finally, the visual programming interface will be pursued further.

6. REFERENCES


SPINDRIFT: A REAL-TIME SPATIALIZED GRANULAR SYNTHESIS ALGORITHM WITH PARTICLE SYSTEM PHYSICS & BEHAVIOURS

Michael Norris, New Zealand School of Music
Jason Post, New Zealand School of Music

ABSTRACT

This paper introduces spin/drift, an algorithm that applies a novel particle-systems approach to granular synthesis. spin/drift extends conventional granular synthesis techniques by using the concepts of an 'emitter' that generates 'particles', both of which can be endowed with independent, automated spatial trajectories. Trajectories may be circular ('spin') or linear ('drift'), or may combine aspects of both (e.g. spirals). Particles may be given absolute trajectories or trajectories relative to the angle of the emitter (e.g. to create 'centrifugal' forces). Particle trajectories may also be affected over their lifetime by environmental forces such as gravity and viscous drag. Spatialization is achieved with a combination of dynamic VBAH encoding, filtration and reverberation cues, and output to an equidistant octophonic speaker array, though this is easily extended to other speaker arrays. The potential implications for the fields of sonic art and design are also discussed, with particular references to Denis Smalley’s taxonomy of ‘cyclic/centric motion and growth processes’.

1. INTRODUCTION & PRIOR ART

Particle systems have been a key component of computer-generated imagery (CGI) since the early 1980s, most notably developed by William T. Reeves in the movie Star Trek: The Wrath of Khan [1]. These systems manage and control the behaviours of multiple ‘particles’ with independent, though related, spatial trajectories. Particle systems are most often used for simulating environmental effects such as clouds, mist, rain, fire and tornados. These systems are now available to ‘prosumers’ through off-the-shelf commercial packages such as Adobe After Effects or Apple Motion.

In the audio domain, the concept of an audio ‘particle’ or ‘grain’ was first clearly established in writings by Gabor [2] and Xenakis [3] and has now become a standard technique in digital sound processing. Details of the most common techniques and their applications are extensively covered in Curtis Roads’ Computer-Generated Imagery (CGI) since the early 1980s, most notably developed by William T. Reeves in the movie Star Trek: The Wrath of Khan [1]. These systems manage and control the behaviours of multiple ‘particles’ with independent, though related, spatial trajectories. Particle systems are most often used for simulating environmental effects such as clouds, mist, rain, fire and tornados. These systems are now available to ‘prosumers’ through off-the-shelf commercial packages such as Adobe After Effects or Apple Motion.

In the audio domain, the concept of an audio ‘particle’ or ‘grain’ was first clearly established in writings by Gabor [2] and Xenakis [3] and has now become a standard technique in digital sound processing. Details of the most common techniques and their applications are extensively covered in Curtis Roads’ Computer-Generated Imagery (CGI) since the early 1980s, most notably developed by William T. Reeves in the movie Star Trek: The Wrath of Khan [1]. These systems manage and control the behaviours of multiple ‘particles’ with independent, though related, spatial trajectories. Particle systems are most often used for simulating environmental effects such as clouds, mist, rain, fire and tornados. These systems are now available to ‘prosumers’ through off-the-shelf commercial packages such as Adobe After Effects or Apple Motion.

2. CONCEPTUAL AND TECHNICAL ISSUES

A number of issues arise in modelling CGI particle systems with granular synthesis:

1. The ear is less discriminating than the eye in tracking spatial trajectories of audio objects.
2. Most typical surround-sound speaker arrays (e.g. 5.1) are limited in the precision of sound location, and most lack the Z dimension.
3. CGI particles tend to have longer lives (often on the order of seconds) than audio grains, with clear trajectory changes to their properties.
4. Some visual characteristics do not map well to audio characteristics (e.g. particle ‘trails’, ‘glow’, colour, rotation, transparency, etc).

While the fine degree of control afforded by spin/drift may be seen as, in some cases, exceeding the psychoacoustic limits of human aural tracking abilities,