Laptop Improvisation in an Multi-Dimensional Space

Sam Pluta

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Musical Arts

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Laptop Improvisation in a Multi-Dimensional Space

Sam Pluta

Using information theory as a foundation, this paper defines virtuosity in the context of laptop performance, outlines a number of challenges that face laptop performers and software designers, and provides solutions that have been implemented in the author's own software environment. A summary of the argument is that by creating a multi-dimensional environment of Sonic Vector Spaces (see page 17) and implementing a method for quickly traversing that environment, a performer is able to create enough information flow to achieve laptop virtuosity. At the same time, traversing this multi-dimensional environment produces a perceptible sonic language that can add structural signposts for the listener to latch on to in performance. Specifics of the author's personal approach to this problem, a software environment coded in SuperCollider, are then shared. Lastly, Mihály Csíkszentmihályi's concept of flow psychology is applied to the three stages of creation in the laptop performance process - software design, patch design, and performance.

Table of Contents

1)	List of Graphs, Charts, and Illustrations	ii
2)	Acknowledgements	iii
3)	Dedication	iv
4)	Introduction	1
5)	Part I: Virtuosity, Complexity, Modularity	3
	1. Virtuosity	3
	2. Perception - What the Listener Hears	4
	3. Complexity and Unpredictability	5
	4. Complexity from Nested Arrays of Modules	7
	5. Grouping Structure, Language, and Form	12
	6. The Sound Module Vector Space	17
	7. Sound Module Vectors in Action	19
	8. The Complex Vector Space	23
	9. The Interaction Gray-Scale	24
6)	Part II: Implementation	29
	1. Why Modular? - David Foster Wallace and Total Noise	29
	2. Original Concept	31
	3. Software Features	34
	4. Goals of Performance - Flow	38
	5. Good Plumbing, Good Flow	40
7)	Conclusion	41
8)	Works Consulted	44

9) Appendix I - Core Components of the Live Modular Interface 46

10) Appendix II - Selected Modules from the Live Modular Interface 118

List of Charts, Graphs, and Illustrations

	Figure Name	Page
1)	Five-Point Listening Space	5
2)	Simple Patch	7
3)	Simple Patch with Switch	9
4)	Simple Patch with Switch and Interrupt Modules	10
5)	Augmented Transition Network	12
6)	Formal Structure I	14
7)	Formal Structure II	17
8)	Analysis/Resynthesis	20
9)	Harmonic Shifter X	21
10)	Cycle Gripper	22
11)	Loop Machine	23
12)	Complex Vector Space	24
13)	The Interaction Gray-Scale	27
14)	Software Features	33
15)	Multi-Dimensional Array	34
16)	Four Simultaneous Layers	35
17)	Interaction of Layers	36
18)	Module Connection	41

Acknowledgements

I am indebted to many individuals and groups who have led me towards this research. I would especially like to thank Brad Garton and George Lewis for their priceless insights during my time at Columbia University and Russell Pinkston for starting me down the road of live-processing software at the University of Texas at Austin. I would also like to thank the developers of SuperCollider for their outstanding programming language that has allowed me to be so expressive in code and Jeff Snyder for his hardware controller, the Manta. Many of the ideas in this paper were developed concurrently while I was creating software for the Manta. In our time together at the Computer Music Center at Columbia University, Jeff and I also worked extensively with the Buchla 100/200 modular synthesizer, using it to create our *exclusiveOr* album in 2007. Working with modular synthesizers changed the way I look at sound creation and manipulation, and was a driving force in initializing my work on a modular live-processing software system. Finally, without instrumental performers, my role in live-processing improvisation would not exist. I would like to thank Jim Altieri, Caroline Mallonee, Alex Ness, and Meighan Stoops of Glissando bin Laden for our three years of music-making as a band. Our weekly rehearsals, constant sonic discovery, and friendship were some of the finest music-making I have ever experienced. I would like to thank the Wet Ink Ensemble for taking me on as Technical Director and giving me the opportunity to use my software in chamber music settings. I would like to thank the Peter Evans Quintet for letting me play jazz, even though I know nothing about it. Lastly I would like to thank all of the performers that I play with in duo performance, mostly: Jim Altieri, Peter Evans, Anne LaBerge, and George Lewis, who provide constant new challenges and exciting performance opportunities.

Dedication

To my teachers, who have been the most supportive group I could possibly imagine:

Brad Garton, George Lewis, Russell Pinkston, Tristan Murail, Fabien Levy, Fred Lerdahl, Pat Plude, Leo Wannanchak, Kevin Putts, Donald Grantham, Tom Lopez, Lynn Shurtleff, Teresa McCollough, Nancy and Leroy Kromm, and Hans Boepple

Introduction

> *Ultimately music is, at its core, a means of communication; computers offer ways of enhancing interconnection.*
>
> -- Scott Grasham-Lancaster, "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music"

Throughout history, technology has a been a driving force behind musical invention. One cannot imagine the Symphonies of Bruckner without the development of the valved horn in the 19th century or the string music of Vivaldi without the development of the violin family of instruments in the 16th and 17th centuries. Such is the case with my argument. Recent developments in technology make the software this paper describes and thus the music it creates possible. While the theories presented in this paper derive much of their substance from previous research, the specific implementation that I have realized would not have been possible but ten years ago. There are three main reasons for this. First of all, ten years ago computers were simply not fast enough to deal with the large amount of information that I process with my software. Second, the relative stability of computer music software environments, in my case SuperCollider, has given me the ability to work with the same programming language for almost a decade. Prior to this, the constant turnover of software environments forced musical software engineers to not only learn new techniques in mastering their trade, but to constantly learn new languages and new software. The relative stabilization of SuperCollider, Max/MSP, and Pure Data around the turn of the 21st century, with the addition of few new processing techniques over the past decade, has allowed me to focus on one software environment for almost ten years, and to build up the substantial cache of

techniques necessary for my argument. Lastly, the shift of focus of the research community away from processing techniques and into the realm of controllers has released a floodgate of new hardware interfaces for musical expression, giving laptop performers newer and better ways to control software with hardware. Because of these three developments I believe we are standing on a watershed moment where live laptop performance is moving away from quaint novelty and into the realm of serious musical achievement.

Please consider this paper as a philosophical discussion of the nature of live-processing-based laptop performance, the challenges faced by the designers of laptop-based improvisation software interfaces, and an outline of how I have designed my own software. A number of topics are discussed simply to illuminate my thinking, but should not confused with my actual implementation. For instance, I talk a bit about my software's similarity to an Augmented Transition Network. My software does not deal with artificial intelligence and does not employ any kind of scheme to aid in phrase generation. However, the network of modules I create is similar in look and feel to an ATN, and thus I use this as a metaphor to help describe how I was thinking in creating the software.

Lastly, while I believe the problems presented in this paper are universal, the solutions presented here are not to be seen as final. Rather, they are simply as the solutions that I have realized in my research. In the end, this paper exists simply to convey my thinking on the subject of laptop improvisation software and display some of the methods I have employed in my own software. My hope is that others find it helpful in finding their own methods and techniques to achieve laptop virtuosity.

Part I: Virtuosity, Complexity, Modularity

Virtuosity

One way to define instrumental virtuosity would be to say that it is the ability to instantly access any technique, sound, note, fingering, or timbre available on one's instrument. It is the facility to move between loud and soft, high and low, noisy and clear, rich and thin sounds - at will and at any moment. A trumpet virtuoso, for instance, can move between a crystal clear note in the highest register to a subtone to a split tone in an instant and with total ease.

To rephrase this, the instrumental virtuoso brings everything he has to the table at all times. In any performance situation he can execute anything he is able to execute at any given moment. The goal of the laptop performer should be to have the same ability. If performing using a granular synthesis tool, the laptop performer should be able to change to a delay tool at literally the flip of a switch. However, this has not historically been the case. In most situations, the delay tool is only available in another patch or piece of software. The laptop performer has to close the current program and open a new one to access the desired tool. To do this he needs to disengage with the performance, re-engage with the mouse and keyboard, close the program, find the new program, open the program, activate the appropriate setup routine, and finally re-engage with his hardware interface; at which point the performance may have already moved to a new space where the delay tool is not appropriate. To continue the trumpet analogy, the performer's plunger mute is in a bag off stage, and he needs to leave the performance to go get it. This is obviously not ideal.

To be clear, virtuosity is not to be confused with musicality, so I am not saying that this is a prerequisite for good music-making. Musicality can be achieved with little or no element of virtuosity. However, the ideal situation would be the ability to do everything one can do, at any time, with the musicality to do what is necessary to make great art.

Perception - What the Listener Hears

Quantifying human perception of music is a difficult task, to say the least. Traditional approaches focus on pitch, rhythm, and duration as their quanta. This is likely because of the historical importance of notation in music from the Baroque through Modern Eras. As Trevor Wishart claims, “it is notability that determines the importance of pitch, rhythm, and duration.” (Wishart 1996:6-7) However, when music moves beyond notes and more into the worlds of timbre and gesture, there needs to be further criteria from which to describe how we perceive sound. Denis Smalley, in his excellent essay on Spectro-Morphology expands the descriptive criteria beyond pitch, rhythm, and duration, to include timbre and space, claiming that “the wide-open sonic world of electro-acoustic music encourages imaginative and imagined extrinsic connections because of the variety and ambiguity of its materials, because of its reliance on the motion of colorful spectral energies, its emphasis on the acousmatic, and not least through its exploration of spatial perspective.” (Smalley 1997:110)

I will claim that in an improvisational setting, one further element surfaces. This is interaction – defined by the OED as “action or influence of persons or things on each other.” (OED 1989) How much are the performers interacting with one another and in what way? Who is in the lead and who is following? What relationship does one performer's material have to another's?

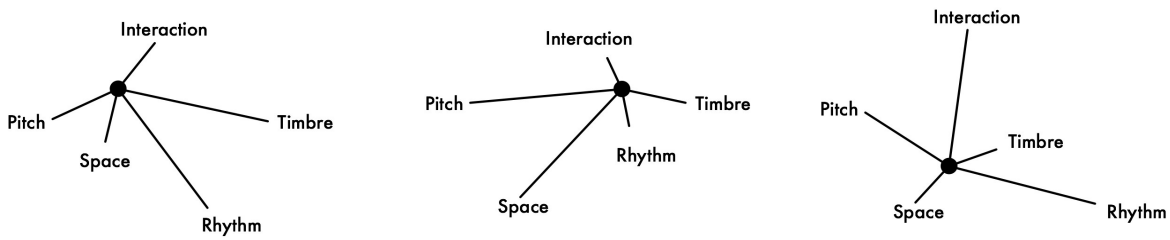


figure 1

By adding interaction into the listening space defined above, we can visualize our listening experience as a five point star surrounding a single point of focus. This map pertains to both listeners and performers. At any point in a musical performance the listener/performer is shifting his focus between these five elements, leaning more towards one and away from another. Any given performance may focus heavily on one element or another, but a virtuosic performer or listener is able to move around the entire space at will, while at the same time keeping all elements in his awareness.

Complexity and Unpredictability

> *Uncertainty is the basis for a lot of my work. One always operates somewhere between the totally predictable and the totaly unpredictable*
>
> -- Don Buchla, *The Buchla 100 Series*

How do we quantify a situation through which a laptop performer is able to easily traverse our five point perception map? How do we quantify the situation in which he is able to achieve virtuosity?

“The simpler an object is, the less information it conveys. The more complexity, the more information.” (Gleick, 2011:loc 5953) The virtuosic performer strives for the possibility of creating a great deal of information and thus the possibility of complexity. The basic problem is that most computer music processes are not capable of producing complex results. Given an input, they are not capable of producing a multiplicity of perceptibly different sonic outputs. Most digital processes are likely have one state in which they are performed, and can be turned on or off. Given a specific input, the output is highly predictable.

There are, of course, processes which are very complex. An example of this is a ring modulator, which has two analog inputs multiplied together, combining to create a continuous and constantly changing output. Any slight variation in either of the two inputs results in a change in the output. Therefore, the infinite possibilities of the two inputs result in infinite possible outputs.

The goal of the virtuosic performer is to create a setup where the user can inhabit a gray scale from simple to complex, as “everything we care about lies somewhere in the middle, where pattern and randomness intersect.” (Gleick, 2011:loc 6257) On the listener's end, the software environment should be able to be unpredictable at times and predictable at others. The challenge is in creating unpredictability. There are two solutions to this problem. The first solution is to make each and every process in a setup as complex as possible, essentially simulating the complexity of a ring-modulator with each and every patch one uses. This is a viable solution. While it does not facilitate a means of moving between ideas, it does give the user a multiplicity of sonic outputs, which could result in interesting music.

The second solution, and the solution of this paper, is to take the modular approach. If we are to assume that all processing modules are equally simple and essentially one bit, the solution is to create complexity by setting up a system with access to a variety of modules. This would result in a range of possible outputs with any given input, thus facilitating the possibility of more complexity. I will once again emphasize the word "possibility." An overly complex system is just as uninteresting as a simple system. An unmusical performer creates more problems than he solves.

Complexity from Nested Arrays of Modules

To continue with the argument, we will assume that all sound processing modules available to a laptop performer are equally minimally complex. Since the goal of this paper is to create a complex setup, we will build our complexity not in each sound producing module we use, but rather by creating a complex matrix of simple modules. Here I will outline the process of building up such a system, starting with the simplest design and growing to a setup that resembles a multi-layered Augmented Transition Network (Bates, 1978) of modules.

The first and most obvious solution to creating a complex setup would be to create a system where more than one process is available to the performer. In the following figure, each box represents a sound producing or manipulating module.

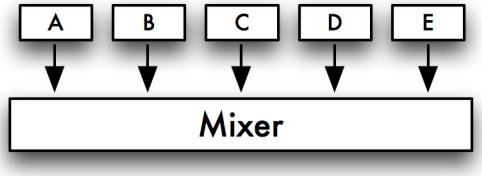


figure 2

This setup gives us five available unique processing options, with the possibility of mixing down multiple processes into combined sonic spaces. For the sake of simplifying the argument, we will assume that the performer will not want to mix more than two processes together at once. Furthermore, in a real-world situation, too many processes sounding simultaneously are likely to create a muddy sonic space, especially when other signals like instrumental performers are present. Therefore, we will assume a maximum of two simultaneous process. The result is an environment with fifteen available spaces - (A B C D E AB AC AD AE BC BD BE CD CE DE).

The above system is appropriate for many situations, but it has its limitations. In this setup, moving between sound spaces is clunky. It would have to be achieved either using five faders that can be moved up and down, or five switches that turn each module on and off. The fader method is graceful but slow, and is likely to take up all of the laptop performer's focus during the performance. No focus is left to control different elements of the modules themselves. The switch method is fast but gruff, and requires a complicated five way switching mechanism, where the performer needs to remember which modules are on and which are off.

To solve the problem of the previous setup, the next level in the modular design uses both methods (faders and switches), adding a switching mechanism between a set of predefined mixes, each of which can be mixed down individually with faders. This setup takes advantage of one of the defining characteristics of modular systems - the ability to create complex structures out of many simpler structures. In this case, the advantage of the design is not necessarily complexity; it is speed. Now there are two simple systems that can be toggled between at the push of a button. Fewer setups are available, but predefined groupings are more

quickly accessed. Each of the available groupings can be mixed gracefully, and once the performer is controlling a group, he can manipulate each module in that group at will.

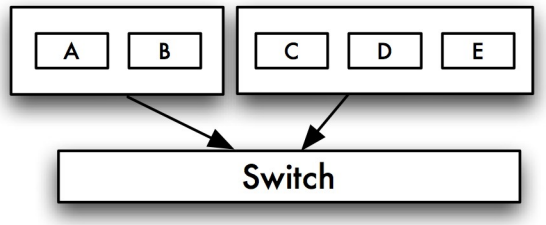


figure 3

Complexity comes through the combination of multiple simple spaces, while the switching mechanism simplifies how the performer reacts to his environment. Rather than having to look at the screen to know where he is, the performer can aurally recognize which module group he is currently playing in, thereby increasing visceral connection to the performance system.

As I continue to build the modular matrix in size, any base grouping of modules will remain small. This gives the user access to the maximum number of sonic spaces while maintaining maximum control over the modules that are sounding at any one time. The result is complexity without being complicated.

Adding a third layer to the system involves what I will call “interrupt modules.” An interrupt module is added to the audio chain in series, and its normal state it does nothing, letting sound pass unaffected. When turned on, however, it can interrupt a signal with its own process. In a digital system an interrupt module might 1) do nothing, 2) filter or distort incoming sound, 3) grab a portion of incoming sound and repeat it, or 4) replace the current output with something else. The classic interrupt module on the Buchla 200 was the Programmable Spectral Processor

Model 296. This module takes a complex input and adjusts the filtration on that input to a different state upon receiving a trigger.

The complexity of interrupt modules comes from a combination of multiple processes; a complex sonic input plus additional filtration/distortion. This creates an unpredictable and complex sonic result. The interrupt module has the possibility of adding to the sound it receives, replacing it altogether with new audio, or doing nothing, thus greatly adding to the complexity of the system without enforcing any kind of prescribed change.

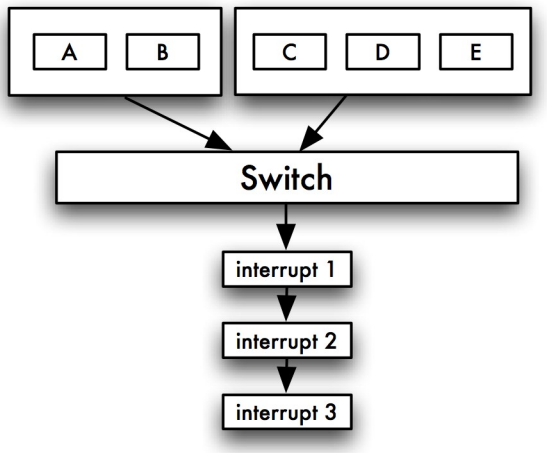


figure 4

The complexity of the system is greatly increased. The setup's possible states are (A B AB C D E CD CE DE)*(Int123) = 36. There are thirty-six possibilities in a system with eight modules, with the ability to move quite quickly between different setups and quickly engage and disengage interrupt modules.

The final step in creating a complex modular setup is to encapsulate multiple smaller groups of modules into separate layers. In the example below (which is an abstraction of an actual system that I use in live performance), there are four layers of modules. The user can only access one

layer of modules at a time, though modules can and do exist in multiple layers. The user switches layers with a switching mechanism, then uses internal switches to access different groups of modules within any layer.

Once again, the complexity of the system has increased greatly, while it is no more complicated than the previous setup. The available states are (A B AB C D J B JB K L KL M)*(IntEFGHI)+(N O P)+(Q R S T U). There 68 available states in the system. Because modules are nicely packaged into small groups, the system is easily traversable and the user can generally know which module group is sounding based on the audio stamp of the currently enacted layer.

The diagram below gives an augment transition network representation of the final modular matrix. Not only does the diagram show the signal path of the system - black lines with arrows represent probable signal flow - but it also shows probable switching decisions that will be made by the user. Certain decisions by the user are more likely than others simply because of the layout of the software. Thus, solid black lines represent more probable changes within layers and dotted lines represent less likely switches between layers. Because a switch from one layer to another is just as likely to result in any module group in the second layer, all switches between layers can be represented with single dotted lines. Probable phrase structres result from the layout, which favors certain changes of modules over others. The resulting musical language is discussed in the following section.

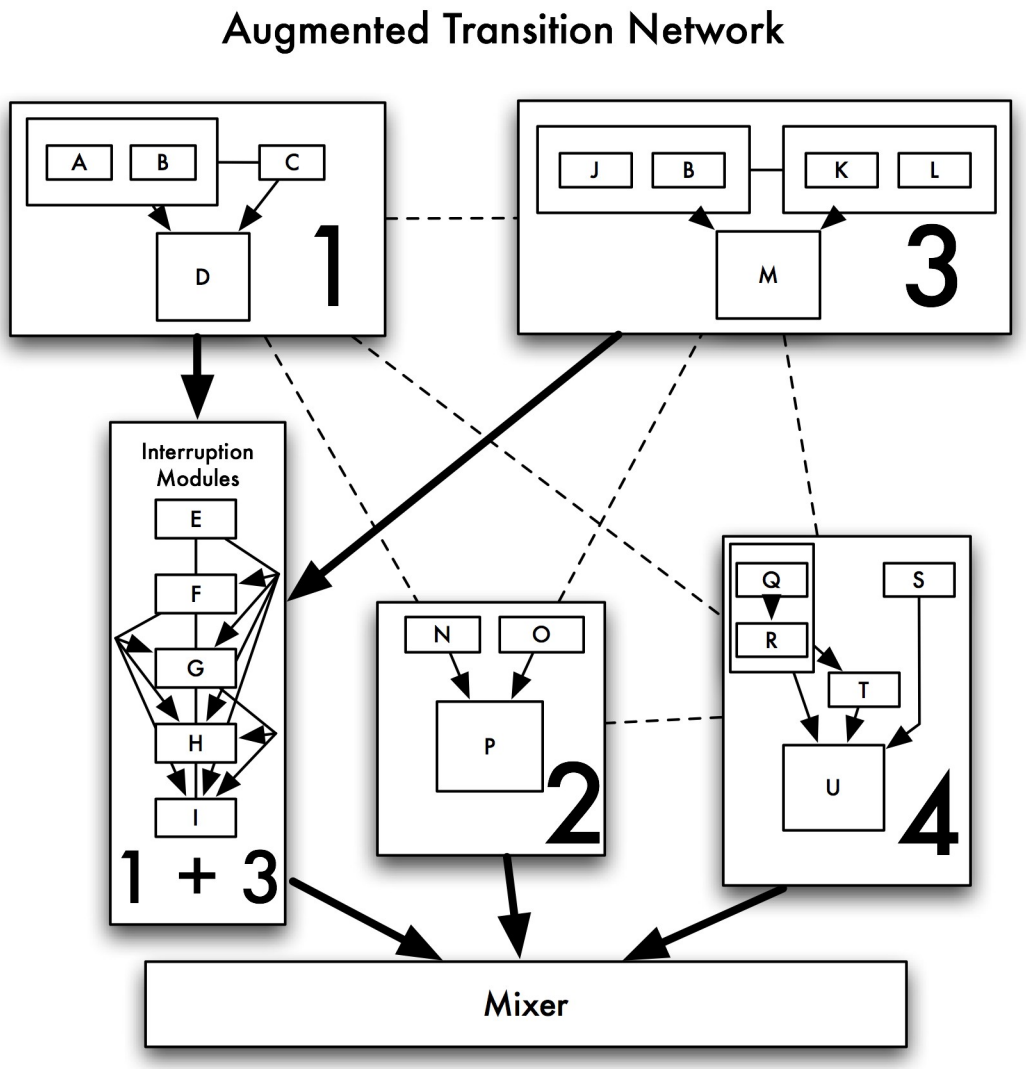


figure 5

Grouping Structure, Language, and Form

If confronted with a series of elements or a sequence of events, a person spontaneously segments or 'chunks' the elements or events into groups of some kind.

-- Fred Lerdahl and Ray Jackendoff, *A Generative Theory of Tonal Music*

One of the main impetuses for developing this framework for improvisation was the desire to create larger formal structures inside the constraints of laptop improvisation. My experience was that most laptop improvisation was one dimensional. Performers had the ability to make great sounds, but moving between sounds was slow and clunky. Envelopes were limited to fades in and fades out and in general most improvisations had a golden arch form, where ideas would slowly accumulate over time, build to a climax, and then decrescendo to the end. I wanted to create a situation where I could quickly move through a large number of processing modules at will, developing a musical language out of these rapid changes in ideas. In other words, I wanted to create my own personal musical language out of the timbres associated with specific combinations of sound processing and synthesis modules.

As Fred Lerdahl and Ray Jackendoff state in *A Generative Theory of Tonal Music*, "If confronted with a series of elements or a sequence of events, a person spontaneously segments or 'chunks' the elements or events into groups of some kind." Though Lerdahl and Jackendoff are dealing with tonal music, the same can be said for any kind of sound organized in time. The exact way a listener might group a noise improvisation, however, would be very difficult to quantify. Even with a single performer, bifurcation points in an improvisation are far from objective, and are likely different from one listener to the next. A listener focusing on timbre could hear changes very differently from one focusing on pitch or envelope. Furthermore, bifurcation points are likely to be the result of the interaction of all elements of our five point focus star. Adding a second or third performer muddies the situation considerably. One performer may pivot, changing to a new section. The second performer may continue in the previous space, slowly moving to the new space over time. Where the change from one section

to the next actually is perceived is subjective, and likely different from one listener to the next.

The grouping structure created by the proposed software design is but one layer of many in a performance, contributing to but not dictating what the listener perceives. What the modular design described in this paper affords is the ability to suggest bifurcation points that may or may not be perceived as phrases and formal sections of music. The software allows the performer to stop on a dime and move to a new improvisational space. How the listener separates pivot points is influenced by many factors: phrase structure of the instrumental performer, resultant envelope/gestural structure created by the interaction of instrumental and laptop performers, which performer is in the lead, interplay between modules, and time-spans between changes of modules, etc. The power of the software is that it provides the laptop performer with the possibility of creating an underlying phrase structure which will contribute to how the improvising ensemble is interacting, plus the ability to change along with the ensemble as they move into new sound spaces. Instant access to change facilitates expressive and dynamic free improvisation.

Using the same modular setup found in figure 5, the following is a possible phrase structure grouping that might occur in performance:

Layer1(
[AB]
[D C D C]
[D H D H D H]
)
Layer3(
[JB KL JB KL]
[H JB F KL]
[JB KL]
)
MultiLayer Grouping(

```
        [Layer1(D) Layer3(KL M) Layer1(D)]
)
Layer2(
        [N -> P]
)
Layer1(
        [AB]
        [D C AB C D]
        [AB]
        [D G D AB G D AB G I G I]
)
```

figure 6

In my experience performing with the software, this might be 30 seconds to one minute of music. Once again quoting Lerdahl/Jackendoff, "A hierarchical structure...is an organization composed of discrete elements or regions related in such a way that one element or region subsumes or contains other elements or regions" (Lerdahl 1983:13). Looking at the diagram, the highest level grouping structure is the Layer. Layers encapsulate groups of modules which encapsulate modules. Once inside a layer, the performer can quickly move between modules encapsulated in that layer. This is likely to create phrases that are perceived by the listener. If the performer focuses on a certain set of modules, then moves to a second set, two different phrases exist. If the perform plays one module for an extended period of time, then moves to a set of modules which he plays in rapid succession, two phrases exist.

The proximity effect in Gestalt Grouping Theory tells us that elements placed close together tend to be perceived as a group (Wolfe 2009). The laptop performer improvising within a Layer can create phrase groups by quickly moving between modules and groups of modules. When a sound event is short enough, the listener does not perceive it as a phrase, but instead groups this sound object into a set with other sound objects. When the performer moves away from a perceived method of switching between objects, this can signify the end of one phrase and the

beginning of the next. Phrase groups do not have to contain multiple modules. If a module is distinct enough or lasts long enough, it can be perceived as its own phrase or even may create a number of phrases.

Phrases are not the only elements the listener perceives. It is likely that some modules will pop out of the texture and be perceived as motifs that recur over time. These modules will be part of other phrases, but their repetition over time can also be heard. The listener will likely group these modules as a single object that is repeated over time.

Larger formal structures can also be created within the software's framework. If, in a given performance, the laptop performer inhabits Layer 1 for an extended period, then moves away into other Layers, a return to Layer 1 can, but does not necessarily, indicate a return to a previously explored sonic environment. Depending on how the performers interact with the software when returning to an already explored area, relationships between larger sections in the music may be perceived by the listener, thus facilitating the creation of large formal structures.

In this compositional scheme, the resultant music consists of a large number of distinct ideas chopped up, with the possibility of repetition and return available throughout the improvisation. These repetitions provide the listener items to latch on to and associate with one another inside a sea of constant change. In the figure below, which is a flattened version of figure 6, four types of repetition are present:

1) Large formal repetition, in this case of Layer 1, which bookends the structure.
2) Layer repetition, similar to type 1, but which does not create a larger formal

structure. In this case, Layer 3 repeats inside the structure, but in one of these cases it exists as part of another phrase, thus it is merely referencing the previous instantiation of the Layer without differentiating itself as a phrase.

3) Groups of modules may repeat inside different phrases. The phrases themselves are built out of many different groups of modules, but portions of these phrases can be related to one another.

4) And lastly, there is the repetition of a single module or module group which can occur in multiple contexts over time.

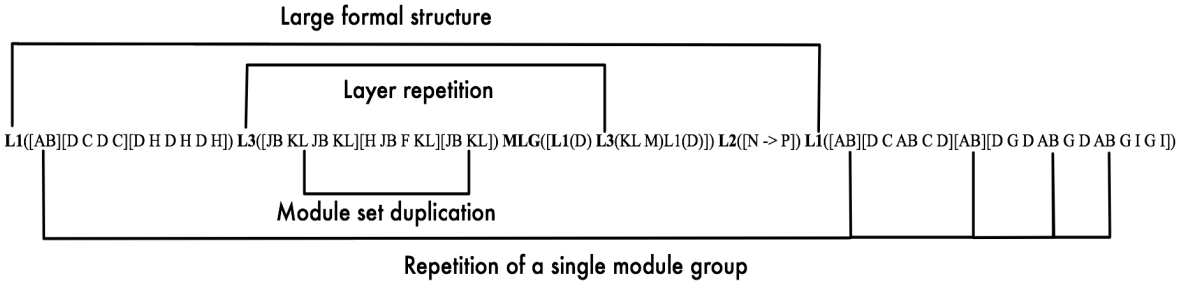


figure 7

The Sound Module Vector Space

Ideally the technology should be transparent, or at least the music needs to be composed in such a way that the qualities of its invention override any tendency to listen primarily in a technological manner.

-- Denis Smalley, “Spectromorphology: Explaining Sound Shapes”

The syntax described in the previous section would likely not be perceived if all modules in a given setup were sonically similar. It only works if each sound module creates a distinct sonic object, unique from the others in the setup. To describe the sonic uniqueness of a sound module, I will present a set of descriptors that can be given a value from 1 to 10. The set of descriptor

values for each sound processing module results in a feature vector that describes that module. Uniqueness among processing modules could then be measured by plotting the feature vectors for a group of modules in an multi-dimensional space and comparing their closeness. In the end, this is probably too extreme. The vector also provides and easily readable description of modules, which can then be compared by a knowledgable programmer or musician.

In this section I will present some sound processing descriptors. I will preface this description by noting that some of the following descriptors are difficult if impossible to quantify and can only be described subjectively. We are not dealing with digital signal processing values like Spectral Centroids or MFCCs, which can be measured by a computer with perfect accuracy. These are values described by humans, not computers. For example, a module might add a great amount of delay or very little, but giving that description a value from one to ten is subjective. We can always add more delay to the system, so it is difficult to say what would constitute a ten on this scale. Furthermore, any given module may allow a range of delay values, making it even more difficult to quantify. Interaction is even less objective. There is no way to mechanically measure this value. However, it is a vital descriptor when differentiating between modules. Finally, this is a rubric of some of the possible descriptors. These are simply the ones I have come up with. I can imagine others exist which could be added to the argument.

Delay - The amount of time between when a source sound first occurs and when it reemerges from the software. A sound with minimal delay has a delay value of 0, while a sound with eight seconds or more of delay has a value of 10.

Space Change - In general, Space Change refers to added reverb. Other factors can also influence space change. Filters can move a sound further away. Distortion and

synthesized elements can move a sound closer up. A large amount of added distance has a value of 6 to 10, while no added distance has a value of 5. Removal of space has a value from 0 to 4.

Instrumental Interaction Value - This is a measure of how much interaction an instrumental performer has with a module. If the instrumental performer is in total control of a module, this has a value of 10. If he has no control, this has a value of 0.

Laptop Interaction Value - This is a measure of how much interaction a laptop performer has with a module. If the laptop performer is in total control of a module, this has a value of 10. If he has no control, this has a value of 0.

Timbral Distortion - If the timbre of a sound is significantly changed by a module, the Timbral Distortion has a value of 10. If the timbre is unchanged, it has a value of 0.

Envelope Distortion - If the envelope of a sound is significantly affected by a module, the Envelope Distortion has a value of 10. If the envelope is unchanged, it has a value of 0.

Synthesis/Processed Value - The Synthesis/Processed Value measures how much of the signal is synthesized and how much is processed. A value of 10 represents a highly synthesized sound and a value of 0 represents a purely processed sound.

Sound Module Vectors in Action

In this section I will make a slight digression away from the abstract and into the real world. I believe it is necessary to clarify the concept of the Sound Module Vector by providing four concrete examples of different modules in my setup and explaining their particular vector.

Analysis/Resynthesis - This module takes an incoming signal, tracks the pitch of the signal and also monitors it for amplitude peaks. On an amplitude peak the pitch of the signal is analyzed. The resultant pitch is then treated as if it were a harmonic of a false fundamental between 7 and 50 hertz. The software creates four other harmonics of the fundamental above the analyzed pitch. If the analyzed pitch were 250 hertz and the fundamental were 10 hertz and the software is creating every other harmonic, the five harmonics would be 250, 270, 290, 310 and 330 hertz. The software then sends these values into five sine wave oscillators. A sound only results when the volume of the input is above a threshold, at which point its amplitude is controlled by the amplitude of the incoming signal.

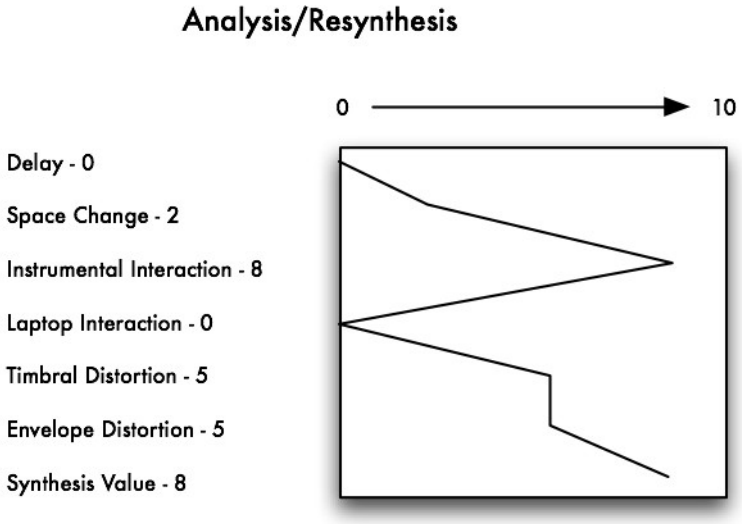


figure 8

The Analysis/Resythesis module adds no delay to the signal; creates sound in a space with no reverb, thereby reducing the space of the signal; is highly effected by the instrumentalist but not at all by the laptop performer; adds significant timbral distortion to the signal; has an envelope effected by the instrumentalist, but not exactly the same; and is highly synthesized.

Harmonic Shifter X - This module takes a signal, records it into a buffer, then asynchronously

plays the recorded buffer back with a granular synthesis unit generator. The playback is pitch shifted to just intervals above and below the incoming signal in ratios of 1/4, 1/2, 5/4, 3/2, 7/4, 5/2, 3, and 7/2.

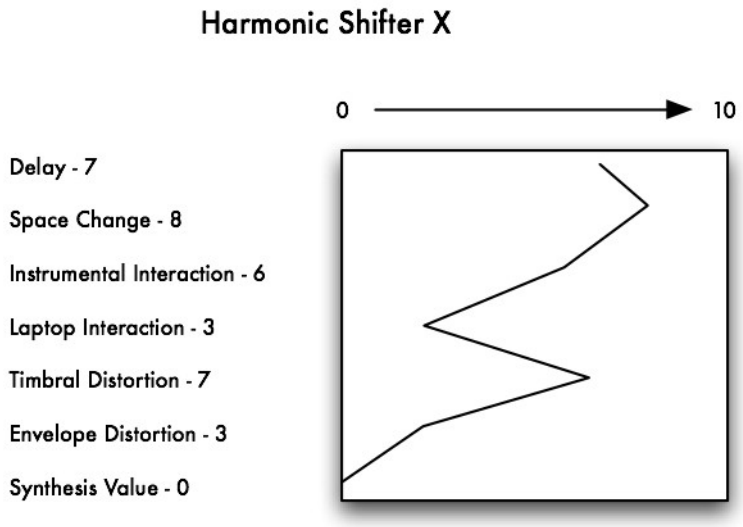


figure 9

Harmonic Shifter X adds significant delay to the signal; the delay and pitch shifting add a large amount space; the instrumentalist has a large amount of control over the signal, but the delay reduces his overall interaction with it; the laptop player has some control, being able to control which harmonies result; there is significant timbral distortion due to the granular synthesis and pitch shifting; the envelope of the input is largely kept in tact, with some distortion present due to the asynchronous granular playback; and there is no synthesis present.

Cycle Gripper - This module takes the input signal, grabs a small portion of audio (as small as one block size), and replaces the input signal by looping the small portion of recorded audio. It also adds an envelope to the output by slightly boosting the amplification on each cycle of the looping output, resulting in a crescendo.

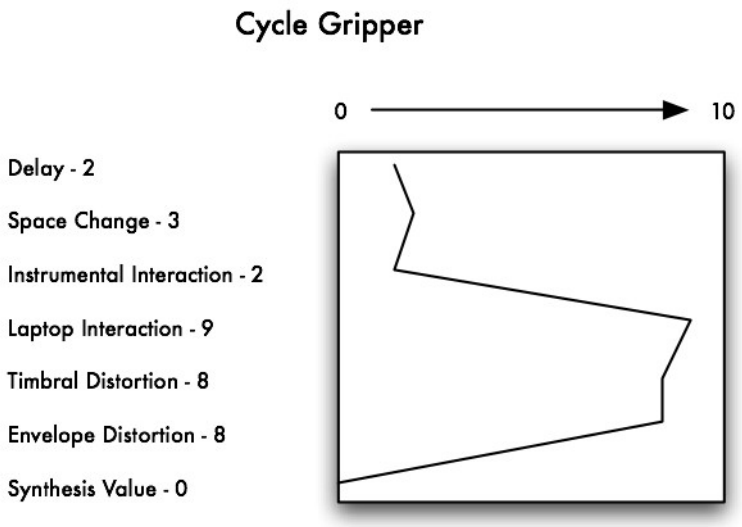


figure 10

The Cycle Gripper adds very little delay to the system; removes space; has very little interaction with the instrumental performer, but a great deal of interaction with the laptop performer; greatly distorts the signal and adds its own envelope to the signal, essentially destroying the original envelop; and adds no synthesis.

Loop Machine - This module records a buffer containing the previous eight seconds of sound at its input, then once engaged plays back the buffer at a rate anywhere from four times faster to zero to four times faster in reverse. Playback rate is controlled by a slider. The input signal is muted when the module is engaged, and replaced with the sound of the Loop Machine module.

The loop machine adds a variable amount of delay, up to eight seconds or more; does not change the space in which sound exists; has very little interaction from the instrumentalist, but a great deal from the laptop performer; adds some timbral distortion when playback rates are different from normal; greatly distorts the envelope of sounds, especially when sounds are reversed; and has no synthesis value.

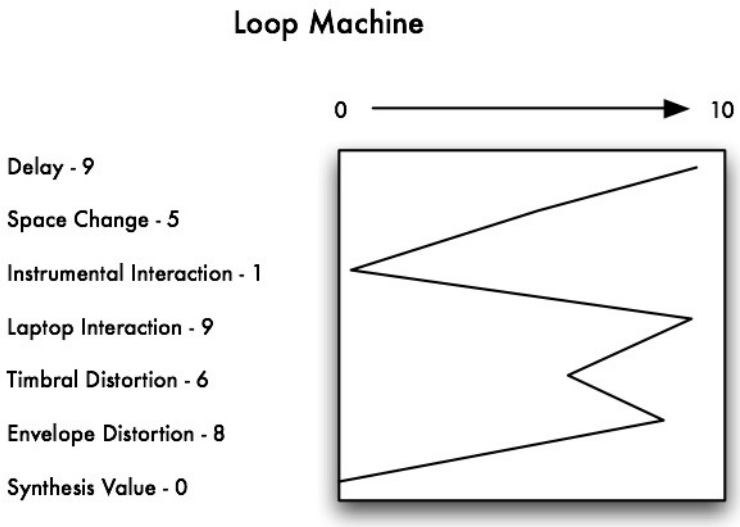


figure 11

The Complex Vector Space

The four examples presented in this section should give a sense of how to create a Complex Vector Space, a system of Sound Module Vectors that the performer can traverse during a performance. Each of these modules has a different vector, thus it interacts with sound in a unique way. By creating a setup where the performer is able to easily switch between combinations of sound modules, as described above, we create a system where multiple ways of interacting with sound are available to the performer at all times. It is as if the performer is able to move around a multi-dimensional space with different parameter values available to him wherever he goes. Thus we have laptop performance in a mulit-dimensional space.

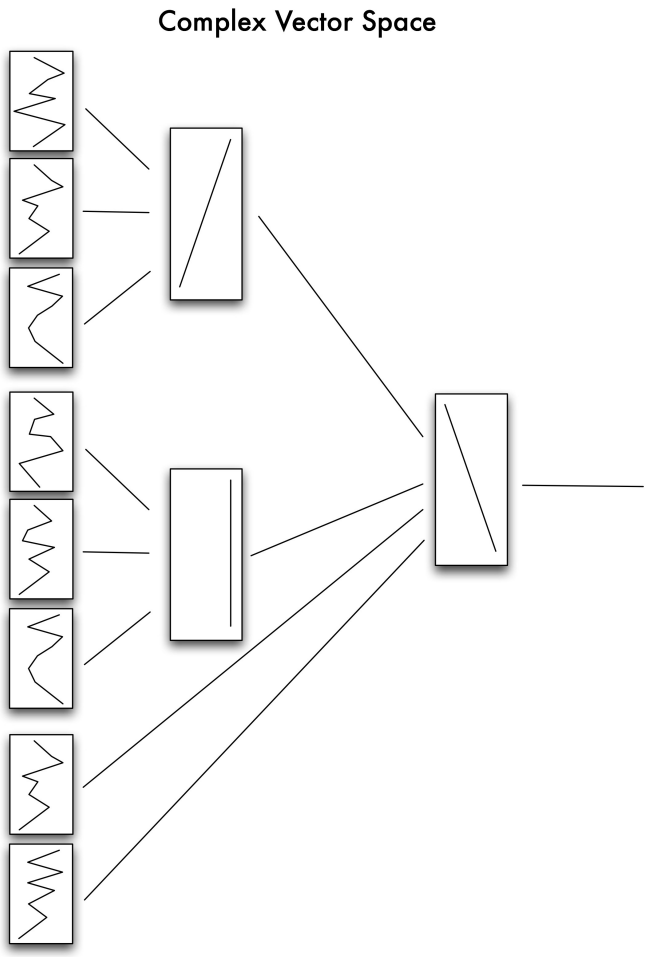


figure 12

The Interaction Gray-Scale

It is likely obvious from the preceding sections that timbral and temporal differentiation between sound modules is vital to constructing an effective modular design. Differentiation of these elements also has a significant effect on how laptop performers and instrumentalists interact with each other during performance. The reason for this is what Denis Smalley calls *source bonding*, or "the natural tendency to relate sounds to supposed sources and causes, and to relate sounds to each other because they appear to have shared or associated origins" (Smalley 1997:110).

Source bonding is what makes laptop improvisation with an instrumentalist or instrumentalists work. To the listener, the instrumental performer is a known quantity with a known and/or visible sound production method. The live-processing patch is not only a foreign entity, meaning the audience is unlikely to know how it works, but in most cases, unlike instrumentalists, the performer's interaction with his instrument does not physically convey its mechanism for sound creation. Therefore, if there is any similarity between the sounds produced by the two performers, the audience member will more than likely link the generation of all or most of the sounds directly to the instrumental performer, even in the case where the two are completely unrelated.

While this may at first seem like a disadvantage, it gives the laptop performer a large distortion space in which to improvise without destroying the audience member's sense of a relationship between the performers on stage. In effect, the audience member's sense of source bonding can be stretched from perfectly obvious to almost imperceptible to the imagined. This stretching and relaxing puts the relationship between performers in a constant state of flux, keeping the audience on their toes and focused on the performance at hand. Whether the laptop performer is distorting the instrumentalists incoming signal, playing samples, or synthesizing sounds for the instrumentalist to play against, this element is always at play.

At the same time, the standard goal of most any improviser is the ability to switch roles between leader and follower, guiding the improvisation at points and following a leader at others. Historically, however, the tendency of the laptop performer has always been to play the role of the follower. This is primarily due to the temporal relationship the laptop has with an the

real world. In most cases, sound from the instrumental performer is informing the responses of the computer component in the improvisation.

Therefore, in both live-processing and analysis/reaction models of interaction, the computer is by nature temporally behind the instrumentalist. There is always a latency between when the sound is made, when the sound is available for processing or analysis, and when the processed or synthesized sound re-emerges from the system. The smallest possible delay is the block size of the digital to audio converter times two - around 3 to 6 milliseconds for a 64 or 128 sample block size. This is a tiny amount of time, but one whose presence is unavoidable. In any situation where information comes from the audio signal, the computer is always behind or at least cannot take the lead.

Aside from a situation where the laptop performer completely disassociates himself from the instrumentalist by using no information from the audio signal, as is George Lewis's occasional method in his *Voyager* program (Lewis 2000), it would therefore follow that the computer improviser cannot take the lead in a performance situation. However, I believe the situation is more complex than this. Let us take into account that there are three possible temporal relationships between the instrumentalist and the laptop:

1) With the smallest possible delay as close as possible the block size times two), maybe in the case of a distortion or pitch shifting module, the sound coming from the laptop is perceived as an effect. The laptop and the instrumentalist become one instrument, contributing to an immediate and temporally fused sonic result.

2) With a slightly larger amount of delay, somewhere between 10 milliseconds and 4 to 8 seconds, the laptop signal is heard as an echo of the instrumentalist, resulting in a

sort of canon of ideas.

3) With a large amount of delay (more than 4 to 8 seconds), there is a bit of a paradox at play. At this point the laptop performer can go far enough behind the instrumentalist to break the perception of source bonding, effectively allowing himself the possibility of taking the lead in performance.

Distortion magnifies the process. With more distortion on the signal, the break in source bonding caused by delay is amplified. Even though the instrumentalist may be performing next to a delayed version of himself, the addition of substantial delay and distortion liberates the sound object from source bonding, giving the laptop performer the independence to take the lead and influence in the improvisation by producing new sounds.

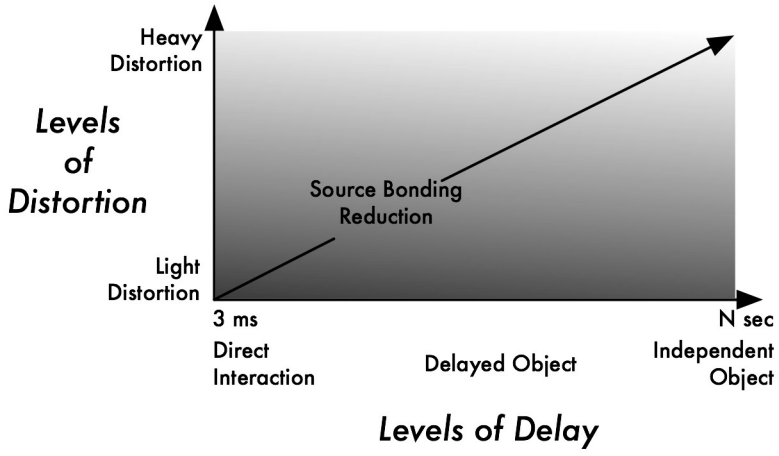


figure 13

In essence, this is how the laptop performer is able to contribute to the interaction value that is present in Figure 1 of this paper. The ability to occupy multiple points on concurrent scales of distortion and delay creates a significantly complex relationship between the laptop performer and instrumentalist(s). The resultant continuum of source bonding reduction allows the laptop performer the possibility of instantaneously changing roles and relationships in the middle of

performance. In my mind, a good improvisation is one where performers are engaged in a dialog of sounds and ideas. To quote George Lewis (1991:203), “Improvisation must be open - that is, open to inputs, open to contingency - a real-time, real-world mode of production.” The dialog must go both ways, with the laptop performer on equal footing with any other member of the performing ensemble; able to contribute new ideas to any improvisation dialog. Oddly enough, these new ideas can simply be distortions of the past.

Part II: Implementation

Why Modular? - David Foster Wallace and Total Noise

In his brief essay *Deciderization 2007 - A Special Report*, David Foster Wallace defines Total Noise as “the tsunami of available fact, context, and perspective” (Wallace 2007:xii). We live in an age of information overload, being hit from all sides with constant streams of information. Wallace's full throttle prose, in my view, can be seen as an artistic and philosophical realization of what our lives have become in the information age. His works not only describe and portray worlds of over-information; his books and essays actually mimic this hypertext barrage of information in prose. Aside from the perdurable need for dictionary consultation, the most obvious composition method is his telescoping story-within-a-story-within-a-story...etc formal style accentuated by hypertextual endnotes and endnotes within endnotes. Multitudes of plot-lines, subplots, asides, digressions, elaborations, and distractions intersect, creating an onslaught of information that often takes multiple highly stimulated readings to digest. Gargantuan sentences are split in the middle by twelve page, five point font endnotes, which also have multiple footnotes. Three days later, upon returning to the main text the reader has to trace back a page and a half to remember where he is. Even though *Infinite Jest*, his largest and most ambitious novel, was written during the decade preceding the internet's rise to world domination, it is clear that Wallace is mimicking its essential modus operandi - more information is always just one click away.

Wallace's work is one of the strongest influences on my music and therefore on my software design. I began exploring his formal structures first in my acoustic composition (*American*

Tokyo Daydream IV (data structures/monoliths), 2007), and then began applying this structure to software based works (*data structures/monoliths ii (for chion),* 2008). Over the past three years I have developed the software described in this paper to create those structures in a real-time live processing environment. It is my attempt at total noise, virtuosity, the tsunami of available timbre and gesture, and complexity through over-information.

The idea of using a multi-dimensional matrix or hyperspace environment for live performance is not new. As far back as 1997, in *Improvisation, Hypermedia, and the Arts since 1945*, Hazel Dean and Roger Smith claim that "the application of hyperspaces for improvised performance is overwhelmingly attractive. Instead of the space dictating limitations of the improvisation, the improvisation will be able to dictate the nature, disposition of, and access to, the space. Each 'transition' in an improvisation could be associated with a corresponding transition of the event spaces in which it occurs." (Smith 1997:53) Furthermore, the method of juxtaposing cut up parts to create a whole is certainly not new in music. As early as 1920, Stravinsky was cutting up scores with scissors to create works like *Symphonies of Wind Instruments*. One of the earliest masterpieces of musique concrète is John Cage's *Williams Mix*, a cut-up mix of nearly 600 samples that took the composer and assistants over a year to splice together. Yet the idea is new in live laptop performance, at least in its execution. I believe this is simply because, as a culture, we are still in awe of the sonic possibilities of live manipulation and have not thought to move beyond this base approach and into the formal structures that can arise through complex modular interaction.

In the musical language that results from traversing the modular matrix, the basic units are block timbres, not notes. This makes sense. In a language that is trying to convey total noise,

the basic unit of sound should be as dense as possible, while at the same time being easily differentiated from items beside it. Timbres are unique building blocks, especially those with ample internal motion. They have the distinction of being able to act not only as objects inside a phrase, but also when long enough, as phrases themselves. Thus they are perfect for a language of noise, a language that conveys meaning by presenting too much information, which is in turn sculpted by its practitioners in real-time.

Original Concept

The Modular Live Interface has been developed between the summer of 2007 and the present, and it continues to undergo changes and updates. It is written in the SuperCollider programming language and runs on Apple Macintosh computers. With subtle reworking it would also run in Linux. Because of the modular design of the interface I have been able to incorporate all of my live-processing algorithms into the program, some written as part of my original endeavors into SuperCollider 2, as far back as 2002.

Prior to 2007, I had written a number of interface programs, each created for a particular composition or installation. I came to understand that interface design was a major part of each of these pieces of software, and I was wasting significant portions of time designing the control, audio routing, and MIDI/HID connection software to make these interfaces work. Furthermore, each time I made a new piece of music, any older processing software I wanted to incorporate into the new piece had to be rewritten to fit the new environment. Forced to focus on these lower-level processes, sound, the most important element to me, often had to take a back seat to interface. My goal became to write a piece of software that could do this lower-level grunt work

for me while at the same time adapting to each new composition or improvisation situation I found myself in.

The result is the Modular Live Interface, a program designed for high level performance and infinite expandability. Because the software is written in SuperCollider, it can be programmed to do almost any task, but because the code is hidden behind a graphical user interface, lower-level features do not burden the user in performance.

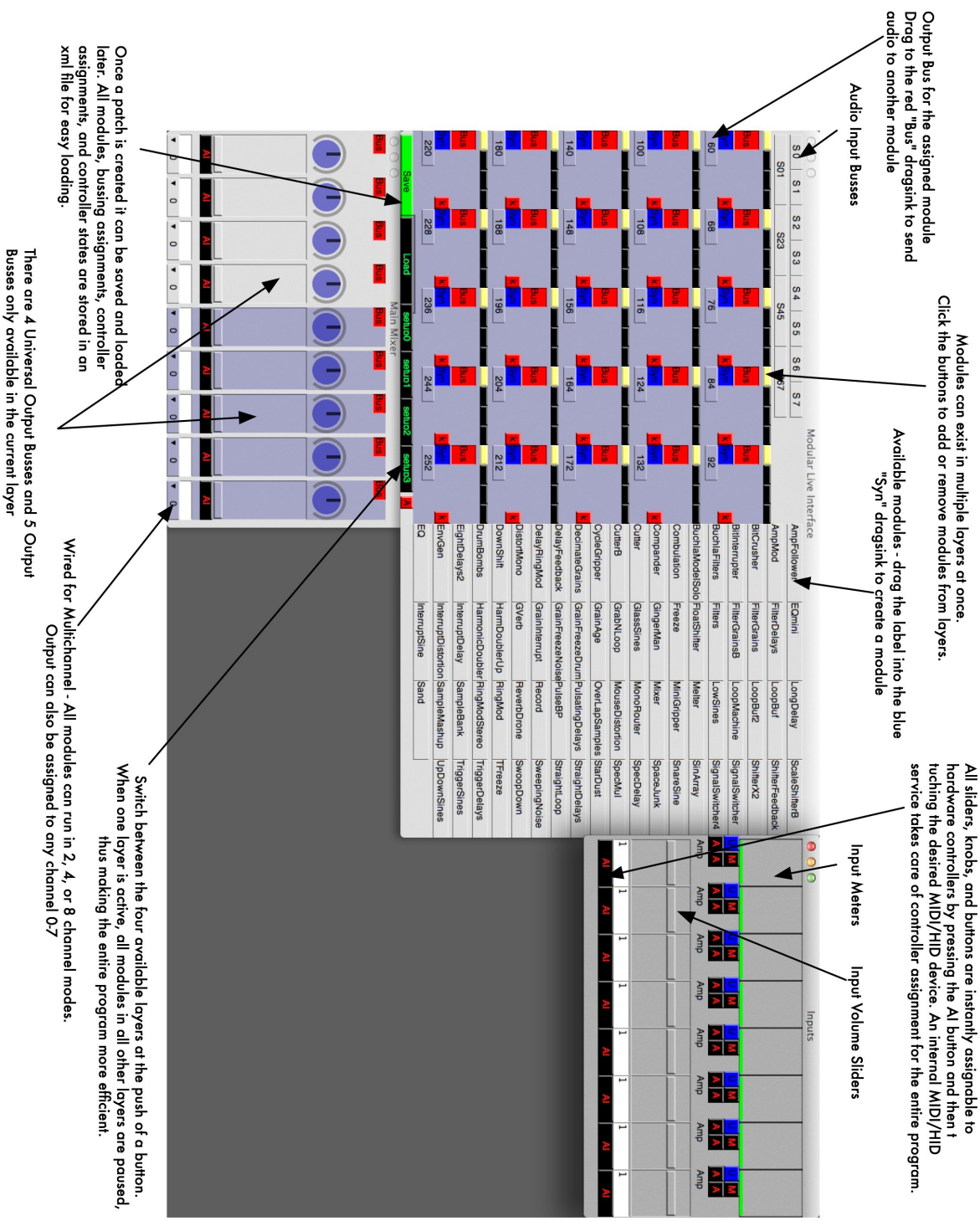

Output Bus for the assigned module
Drag to the red "Bus" dragsink to send
audio to another module
Audio Input Busses
Modules can exist in multiple layers at once.
Click the buttons to add or remove modules from layers.
Available modules - drag the label into the blue
"Syn" dragsink to create a module
All sliders, knobs, and buttons are instantly assignable to
hardware controllers by pressing the AI button and then t
tuching the desired MIDI/HID device. An internal MIDI/HID
service takes care of controller assignment for the entire program.
Input Meters
Input Volume Sliders
Once a patch is created it can be saved and loaded
later. All modules, bussing assignments, controller
assignments, and controller states are stored in an
xml file for easy loading.
Switch between the four available layers at the push of a button.
When one layer is active, all modules in all other layers are paused,
thus making the entire program more efficient.
Wired for Multichannel - All modules can run in 2, 4, or 8 channel modes.
Output can also be assigned to any channel 0-7
There are 4 Universal Output Busses and 5 Output
Busses only available in the current layer

figure 14

Software Features

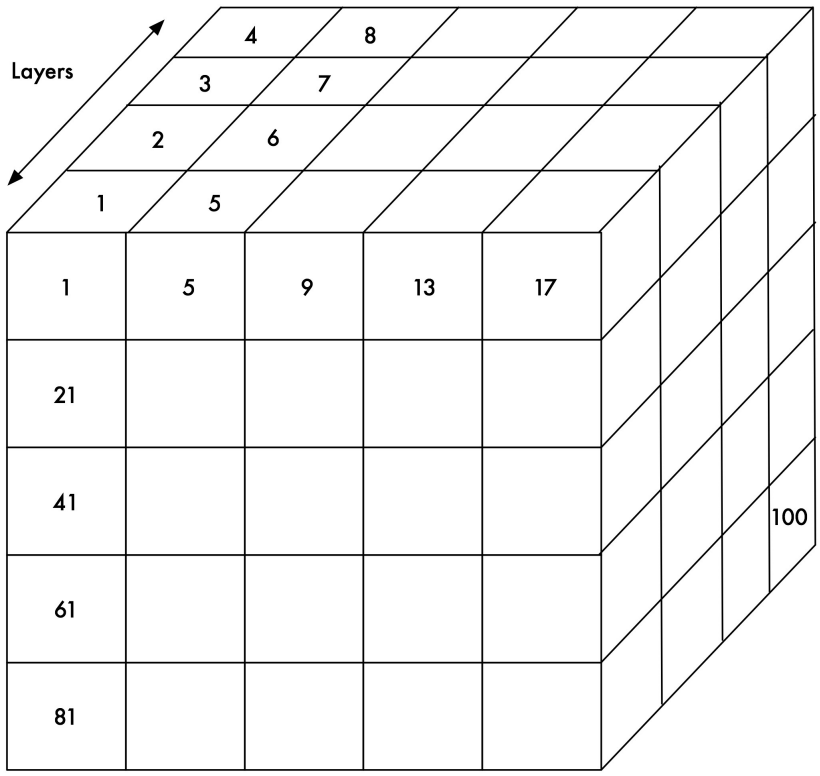
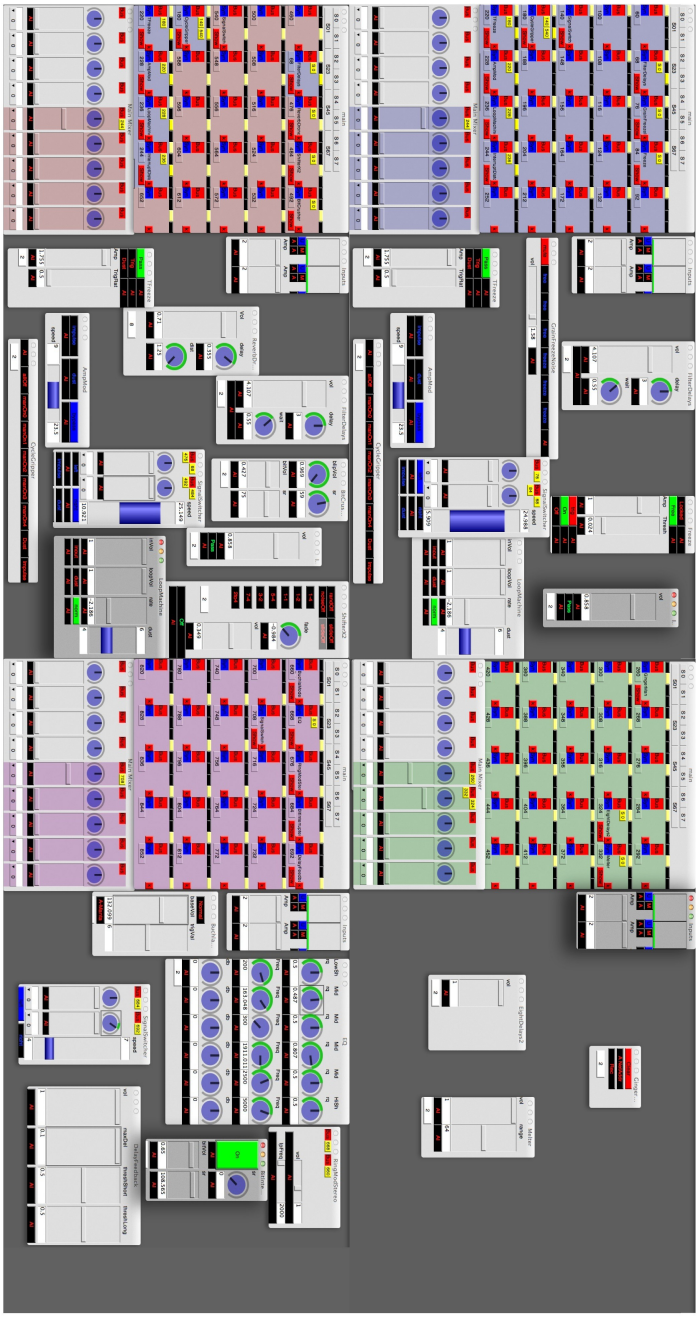


figure 15

1) The Multi-Dimensional Array - In order to facilitate the easily traversable matrix of modules that were described in Part I of this paper, the software is set up so that modules exist in an array of layers, with each layer a five by five array of module slots. In essence, the result is a five by five by four array of modules with the possibility of one hundred different processes interconnected through pre-assigned busses. The multi-dimensional array takes care of one of the nastiest problems the designer faces in dealing with digital processing software - order of operations. Order of operations is enforced by a top left to bottom right queuing scheme, following a z-x-y ordering to module slots.

2) Four Simultaneous Layers - While there are four separate layers of module slots, only one layer is active at any one time. When a layer is activated, all of the modules of that layer are brought to the front. All modules not in that layer are hidden and paused.



Four Simultaneous Layers in one Live Modular Interface Patch

figure 16

3) Layer Pausing - Only one layer is available to the user at any one time. Items and processes not in the current layer are paused and can be unpaused when a layer in which they exist is enabled. This efficiency consideration allows the software to run nearly four times as many unit generators as it otherwise could. This allows incredibly dense patches without overtaxing the CPU.

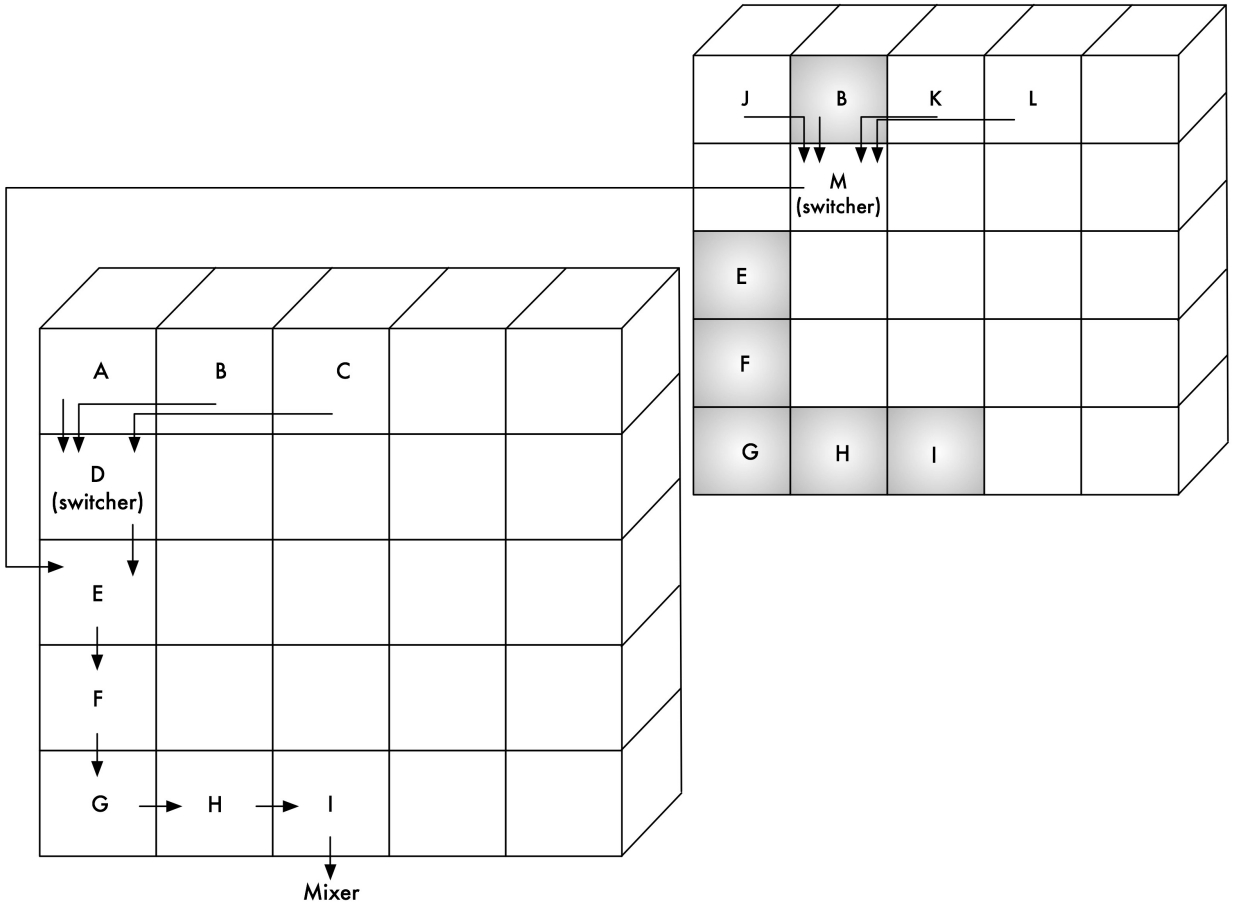


figure 17

4) Modules Can Exist in More Than One Layer at a Time - The z-x-y ordering described above may at first seem odd, however this allows a specific function of the software that is essential to its operation. In order to create some similarity between module groups in different layers, some modules may exist in two or more layers simultaneously. Putting modules in multiple layers allows audio to flow in three directions: left to right, top to

bottom, and front to back. This allows complex bussing strategies that would look complicated in two dimensions, but are simple in three.

5) Drag-able Module Creation and Click-able Removal - In order to create a module the user simply drags the name of the desired module from the list of available modules, into the “Syn” dragsink of the desired module slot. Modules are removed by pressing the “K”ill button. All MIDI assignments and internal processes are also removed when a module is deactivated.
6) Automatic Bussing - When a module is created in a module slot, its output is automatically routed to the output bus of that module slot. The output of a module slot can then be routed to any other module slot by dragging the output bus number for the slot to the “Bus” dragsink of the desired target slot.
7) Multichannel Support - Each and every module in the setup can run in 2, 4 or 8 channel mode. Because the busses between objects are already 8 channels, there no extra steps necessary when running in multichannel mode.
8) Automatic Controller Assignment - Every slider, knob, and button in every module can be automatically assigned to any slider, knob, and button on any controller. Attached to each control object is an AI or “Assign Instantly” button. The user touches the AI button, then touches the desired controller and the controller is instantly assigned to the slider or button.
9) Distinct Controller Layout in Different Layers - Controller assignments in one layer do not transfer to another unless the associated module also exists in the second layer. This means that each layer can have a distinct controller layout with hardware buttons and sliders used differently in each of the four different layers. This minimizes the number of controllers needed while maximizing control.

10) Saving and Loading Patches - All setups can be saved into xml format files to be loaded at any time. All module assignments, bussing assignments, controller assignments, and controller states are stored in the saved file.

Goals of Performance - Flow

> *The translation of performance-gesture into gestural-structure of the sound object is most complete and convincing where the technology of instrument construction does not present a barrier.* --Trevor Wishart, *On Sonic Art*

While my goals as a laptop performer are to create a musical language consisting of a varied sonic output by accessing multiple Sonic Vector Spaces, and to provide a suitable interaction with my instrumental colleagues by traversing the Interaction Gray-Scale, these are not things I want to be thinking about in performance. My goal in performance is to enter a flow state (Csíkszentmihályi, 2008), a fully immersed state of focus, where all I am thinking about is the sound being produced by my setup and by those around me. Distractions from the sound environment are to be avoided at all cost. These push the user out of a flow state, and into a world of technical issues, where performance becomes a side note. The computer is a problematic instrument. Before the performability of the performance software even comes into the picture, it presents a variety of anxiety inducing and distracting problems: Did I boot the server? Is the MIDI controller on? Will the software update I ran yesterday make my computer crash during the gig? This list could go on for pages.

A musical software environment should reduce the user's anxiety about performance, not

increase it. The Live Modular Interface does this by providing a multi-dimensional space for the user to traverse. The user is then not thinking about controller assignments, correct bussing, phrase creation, whether the correct sounds are available in the current patch, source bonding, etc. The interface gives the user access to a flow state, because with proper preparation, the performer only needs to think about interacting with hardware controllers and sound and nothing else. In performance, my setup is such that I hardly ever look at the screen. The screen is distracting. I avoid the mouse and keyboard, two very poor interfaces for sound manipulation. In my setup, faders and buttons on hardware controllers represent interactions with sound. This is all I have to think about. A complex system of layers and controls exists behind the abstraction of my interface, but it is not what I am thinking about.

The modular design allows my setup to be complex while the focus of my current sound world is simple. Many manipulation objects are available at the flip of a switch, but only a few are acting on sound at any one time. My ear tells me where I am in the setup and guides my fingers to the correct controllers for that part of the setup. I am not thinking. I am performing. Because of this, I maintain a visceral connection to the sound at all times.

Good Plumbing, Good Flow

Flow is not only an important element of the performance situation. It is important in every step of the design process, especially in two steps along the way: 1) the creation of modules and 2) the patching together of a module layout to be used in performance.

In a modular design, or one with independent parts, an engineer is able to focus on details of a

single item of the product with no effect on the design as a whole. In the Modular Live Interface, order of operations, MIDI assignment, bussing, and mixing are taken care of by the larger software package. When the engineer is working on a new sound module, lower-level operations are not a distraction. Sound is the only focus, and because of this the designer is more capable of achieving a flow state during the design process. In a flow state the engineer is more capable of producing interesting sounding modules that will be useful in a performance situation.

The Modular Live Interface is also an outstanding environment for creating performance patches. Modules and audio connections between modules are easily created by dragging sources and dropping them into destinations. Complex routing is simplified, without the presence of hundreds of messy wires. In the current setup, over 800 audio busses are in use before a single module is created. However, they are completely invisible and do not get in the way. Furthermore, they are easily traceable with numbered labels. Modules can easily be added or removed from layers with a unique button interface. Control objects are assigned to hardware controller with the push of a button. A completed patch can be saved to be loaded later. When the user sits down to create a performance patch, the software provides an environment to create the interface that is immediately ready for performance, without the distraction of lower-level elements that would interrupt the flow state. In performance, the user can also modify a patch without interrupting the performance to pause audio and change the software.

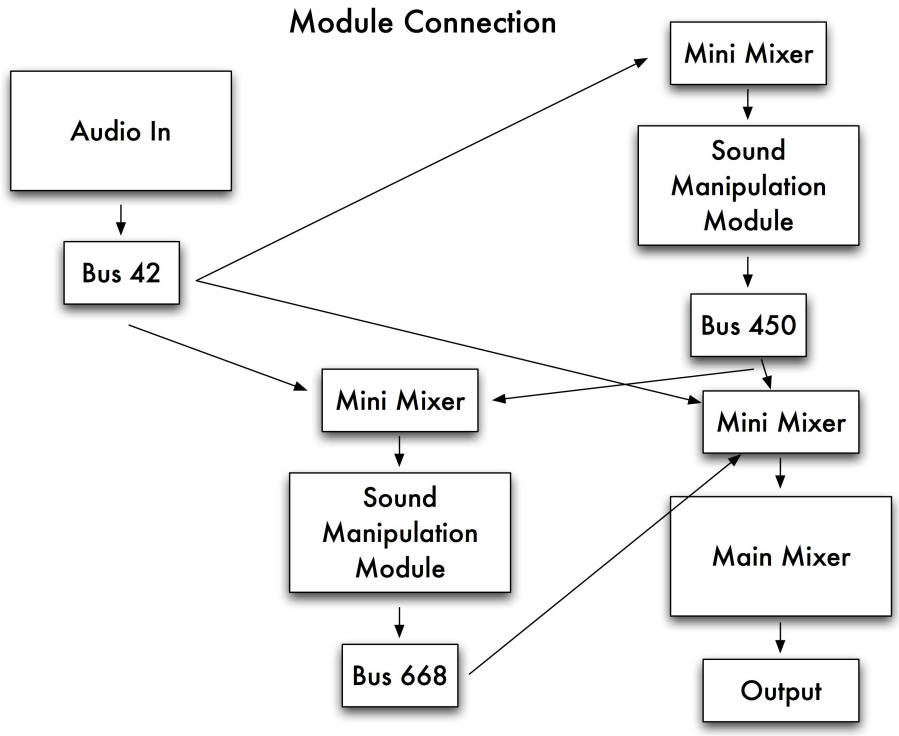


figure 18

Because modules serve specific functions, new and interesting combinations can be made on the fly by daisy-chaining different modules with different Vector Spaces. The results can be outstanding or terrible, but are seldom predictable. In a flow state of patching, the user is able to expand his performance vocabulary by improvising new connections and interactions. Feedback is immediate. As soon as the modules are are instantiated, they are able to make sound. The user knows in an instant whether the combination works or not. He is able to interact with it as soon as he assigns his controllers, which takes just a moment.

Conclusion

The goal of virtuosic interactive laptop improvisation is to produce a dynamic performance of an evocative sound world in an open creative environment. This paper theorizes that in order to

achieve laptop virtuosity, a software environment should allow a multiplicity of possible outputs with a method of quickly accessing each of those possibilities. By providing the user with multiple Sonic Vector Spaces arranged in a multi-dimensional matrix, and giving him a creative approach to moving between different Vector Spaces, the software described in this paper provides a system where the performer can virtuosically adapt to different performance situations and produce varied sonic results. At the same time and through the same method, the user is able to produce a perceptible sequence of events that gives the performance audible phrases and formal shape and contributes to the larger structure perceivable by the audience.

Concurrently, interactive improvisation software should give a performer the tools necessary to be an integral part in the developing dialog of an improvisation. By manipulating levels of distortion and delay, a software improvisation environment provides a method of traversing the Interaction Gray-Scale, affording the performer the ability to act in roles from follower to leader and everywhere in between, thus putting him on equal footing with instrumental performers in steering the course of an improvisation.

Finally, a musical software environment should facilitate a flow state in every step in the process of software/sound creation – module design, patch design, and performance. The Modular Live Interface described in this paper provides the laptop performer with a framework for dynamic performance in a flow state by providing a multi-dimensional matrix of Sonic Vector Spaces with varying levels of interaction between instrumental and laptop performers. The user does not have to think about the elements of laptop performance during an improvisation, thus these technical matters do not hinder the performer's ability to improvise. He simply needs to traverse the multi-dimensional matrix and react to the sonic results to create

the necessary variance in parameters. Opportunities for uninhibited creation in a flow state are also available while creating modules and performance patches. By isolating the software framework design from the sound-module and patch design processes, the software structure allows the engineer to focus on sound module creation and patch design as separate from low-level features of the software.

While I believe Sonic Vector Spaces and the Interaction Gray-Scale are universal to all live-processing situations, the solution of the modular matrix provided by this paper is personal, developed over multiple years of trial and error. The specific software structure of the Modular Live Interface works well with my aesthetic considerations of total noise and over-information, therefore the resulting musical language is not necessarily something I would impose on other performers and composers. I imagine there are other solutions to these problems that will result in equally unique musical languages, and I hope that this paper, if nothing else, will encourage others find these solutions.

Works Consulted

Armstrong, Newton. "An Enactive Approach to Digital Musical Instrument Design." Diss. Princeton University, 2006.

Bates, M. “The theory and practice of augmented transition network grammars.” *Natural Language Communication with Computers*. Springer Verlag, Berlin: 191-259, 1978.

Blackwell, Tim and Young, Michael. "Live Algorithms." LAM Manifesto. Goldsmiths College, London. Web. 05 Nov. 2011. <http://igor.gold.ac.uk/~mas01tb/papers/AISB.pdf>.

Buchla, Don. "Buchla 100 Series." Vasulka.org. 1964. Web. <http://www.vasulka.org/Kitchen/PDF_Eigenwelt/pdf/096-099.pdf>.

Chion, M. 1983. *Guide des objets sonores*. Paris: Buchet Chastel INA-GRM.

Cover, T. M., and Joy A. Thomas. *Elements of Information Theory*. Hoboken, NJ: Wiley-Interscience, 2006.

Csíkszentmihályi, Mihály. *Flow*. New York: Harper Perennial, 2008.

"David Foster Wallace And Hypertext." *Raise High the Roofbeam, Carpenters!* Web. 20 Dec. 2011. <http://bmackie.blogspot.com/2009/05/david-foster-wallace-and-hypertext.html>.

Gresham-Lancaster, Scot. "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music." *Leonardo Music Journal* 8 (1998): 39-44. Print.

Gleick, James. *The Information: a History, a Theory, a Flood*. New York: Pantheon, 2011.

“interaction, *n.”* The Oxford English Dictionary. 2nd ed. 1989. OED Online. Oxford University Press. 2 March 2012 <http://dictionary.oed.com/>.

Lerdahl, Fred, and Jackendoff, Ray. *A Generative Theory of Tonal Music*. Cambridge, MA: MIT, 1983.

Lerdahl, Fred. "Timbral Heirarchies." *Contemporary Music Review 2* (1987): 135-60.

Lewis, George E. “The Secret Love between Interactivity and Improvisation, or Missing in Interaction: A Prehistory of Computer Interactivity.” In Fähndrich, Walter, ed. *Improvisation V: 14* Beiträge. Winterthur: Amadeus (2005).

Lewis, George E. "Too Many Notes: Computers, Complexity and Culture in Voyager." *Leonardo Music Journal 10* (2000): 33-39.

Nørretranders, Tor. "Chapter 6, The Bandwidth of Consciousness." The User Illusion: Cutting

Consciousness down to Size. New York: Viking, 1998.

Oliveros, Pauline. *Software for People: Collected Writings 1963-80*. Baltimore, MD: Smith Publications, 1984.

Pluta, Sam. "Maximize Information Flow: How to Make Successful Live Electronic Music." New Music Box. American Music Center, 18 June 2008. Web. <http://www.newmusicbox.org/articles/Maximize-Information-Flow-How-to-Make-Successful-Live-Electronic-Music/>.

Smalley, Denis. "Spectromorphology: Explaining Sound-shapes." *Organised Sound 2.2* (1997): 107-26.

Smith, Hazel, and Roger Dean. *Improvisation, Hypermedia and the Arts since 1945*. London: Harwood Academic, 1997.

Wallace, David Foster. *Infinite Jest: a Novel*. Boston: Little, Brown and, 1996.

Wallace, David Foster. "Deciderization 2007 - A Special Report." *The Best American Essays 2007*. Boston: Houghton Mifflin, 2007.

Wishart, T. 1996. *On Sonic Art*, S. Emmerson (ed.). Amsterdam: Harwood Academic Publishers.

Wolfe, Jeremy M. *Sensation & Perception*. Sunderland, Mass: Sinauer Associates, 2009.

Appendix I

Core Components of the Live Modular Interface

```
ModularSynthBar {
	var group, win, point, mainProcessingWindow, midiHidControl, manta, bcf2000, soundInBusses, stereoSoundInBusses, objectBusses, synthRow, modularClassList, modularObjects, panel, outputBusses, tempObject, numObjects, xmlModules, floppedBusses, objectBusNums, tempList, temp;

	*new {arg group, win, point, mainProcessingWindow, midiHidControl, manta, bcf2000, soundInBusses, stereoSoundInBusses, objectBusses, synthRow, modularClassList;
		^super.newCopyArgs(group, win, point, mainProcessingWindow, midiHidControl, manta, bcf2000, soundInBusses, stereoSoundInBusses, objectBusses, synthRow, modularClassList).init;
	}

	init {
		objectBusNums = List.new;
		objectBusses.flatten.do{|item| objectBusNums.add(item.index)};
		floppedBusses = objectBusses.flop;
		numObjects = 5;
		panel = SCCompositeView.new(win, Rect(point.x, point.y, (100*5), 90));
		modularObjects = List.new;
		numObjects.do{arg i; modularObjects.add(NewMainModularObject(Group.tail(group), panel, Point(point.x+(100*i), 0), i, this, midiHidControl, manta, bcf2000, soundInBusses, stereoSoundInBusses, floppedBusses[i], mainProcessingWindow, synthRow, i, modularClassList))};
	}

	changeSetup {arg num;
		modularObjects.do{arg item; item.changeSetup(num)}
	}

	requestBusValidity {arg object, bus;
		^mainProcessingWindow.requestBusValidity(object, bus);
	}

	requestBusAssignInfo {arg object, inputBusses;
		mainProcessingWindow.requestBusAssignInfo(object, inputBusses);
	}

	getOutputBusses {
		^objectBusNums;
	}

	getOutputBussesBeforeX {arg x;
		tempList = List.new;
		objectBusses.do{arg item;
			if(x>0,{
				(0..(x-1)).do{arg i; tempList.add(item[i].index)};
			})
```

```
                };
                ^tempList;
        }

        includes {arg object;
                ^modularObjects.includes(object);
        }

        requestObjectXCoor {arg object;
                ^modularObjects.indexOf(object);
        }

        killMe {
                modularObjects.do{arg item; item.killMe};
        }

        save {arg xmlDoc, xmlSynthBar;
                xmlModules = List.new;
                modularObjects.do{arg item, i;
                        xmlModules.add(xmlDoc.createElement("MainModularObject"++i.asString));
                        xmlSynthBar.appendChild(xmlModules[i]);
                        item.save(xmlDoc, xmlModules[i]);
                }
        }

        load {arg xmlSynthBar, busMap;
                modularObjects.do{arg item, i;
                        item.load(xmlSynthBar.getElement("MainModularObject"++i.asString),
busMap);
                }
        }
}

MainProcessingWindow : MIDIHIDObject {
        var <>group, <>soundInBusses, <>stereoSoundInBusses, <>liveInterface, <>whichClassList,
win, synthList, availableModules, synthLocation, availableBusses, paths, file, text, num;
        var saveButton, loadButton, xmlSynthBars, xmlBusses, xmlSoundInBusses, xmlInternalBusses,
soundInBussesTemp, stereoSoundInBussesTemp, availableBussesTemp, <>busMap, soundInBoxes,
stereoSoundInBoxes, classBoxes, modularClassList, xmlBounds;
        var setupButtons, objectBusses, assignMantaButton, mantaIsSet, smallWin;
        var xmlLocalBusses, xmlLocalBussesSize, extraLocalBusses, temp;

        *new {arg group, soundInBusses, stereoSoundInBusses, midiHidControl, manta, bcf2000,
liveInterface, whichClassList;
```

```
^super.new.group_(group).soundInBusses_(soundInBusses).stereoSoundInBusses_(stereoSoundInBusses).midiHidControl_(midiHidControl).manta_(manta).bcf2000_(bcf2000).liveInterface_(liveInterface).whichClassList_(whichClassList).init;
	}

	init {
		win = SCWindow("Modular Live Interface", Rect(0, SCWindow.screenBounds.height, 100*5+300, 90*5+60));
		win.alwaysOnTop_(true);

		soundInBoxes = List.new;
		soundInBusses.do{arg item, i;
			soundInBoxes.add(SCDragSource(win,Rect(5+(45*i), 0, 45, 16)));
			soundInBoxes[i].setProperty(\align,\center);
			soundInBoxes[i].object = [item, "S"+i.asString];
			soundInBoxes[i].string="S"+i.asString;
			soundInBoxes[i].dragLabel="S"+i.asString;
		};

		stereoSoundInBoxes = List.new;
		stereoSoundInBusses.do{arg item, i;
			stereoSoundInBoxes.add(SCDragSource(win,Rect(5+(90*i), 20, 90, 16)));
			stereoSoundInBoxes[i].setProperty(\align,\center);
			stereoSoundInBoxes[i].object = [item, "S"++(i*2).asString++((i*2)+1).asString];
			stereoSoundInBoxes[i].string="S"++(i*2).asString++((i*2)+1).asString;
			stereoSoundInBoxes[i].dragLabel="S"++(i*2).asString++((i*2)+1).asString;
		};

		objectBusses = List.new;
		4.do{arg i;
			objectBusses.add(List.new);
			5.do{arg i2;
				objectBusses[i].add(List.new);
				5.do{arg i3;
					objectBusses[i][i2].add(Bus.audio(group.server,8));
				}
			}
		};

		classBoxes = List.new;
		modularClassList = ModularClassList.new(whichClassList);
		modularClassList.classArray.do{arg item, i;
			classBoxes.add(SCDragSource(win,Rect(500+((i/20).floor*100), i%20*25, 100, 20)));
			classBoxes[i].object = item;
		};
```

```
		synthList = List.new;
		5.do{arg i; synthList.add(ModularSynthBar(Group.tail(group), win, Point(0,90*i+40),
this, midiHidControl, manta, bcf2000, soundInBusses, stereoSoundInBusses, objectBusses.flop[i], i,
modularClassList))};

		saveButton = SCButton(win, Rect(5, 90*5+45, 120, 16))
			.states_([["Save", Color.black, Color.green]])
			.action_{
				CocoaDialog.savePanel({ arg path;
					liveInterface.save(path);
				},{
					"cancelled".postln;
				});
			};

		loadButton = SCButton(win, Rect(110, 90*5+45, 120, 16))
			.states_([["Load", Color.green, Color.black]])
			.action_{
				CocoaDialog.getPaths({ arg paths;
					paths.do({ arg path;
						liveInterface.load(path);
					})
				},{
					"cancelled".postln;
				});
			};

		setups = [\setup0, \setup1, \setup2, \setup3];

		this.initControlLists(4);
		controls = List.new;
		assignButtons = List.new;

		4.do{arg i;
			controls.add(SCButton(win, Rect(220+(60*i), 90*5+45, 60, 16))
				.states_([["setup"++i, Color.green, Color.black],["setup"++i, Color.black,
Color.green]])
				.action_{arg butt;
					setupButtons.do{arg item; item.value = 0;};
					butt.value = 1;
					liveInterface.changeSetup(i)
				};
			);
			this.addDoubleAssignButton(Rect(220+(60*i), 90*5+65, 60, 16),i,0);
		};
```

```
		assignMantaButton = SCButton(win, Rect(465, 90*5+45, 16, 16))
			.states_([["A", Color.black, Color.red],["C", Color.red, Color.black]])
				.action_{arg butt;
					if(butt.value==1,{
						mantaIsSet = true;
						4.do{|i| manta.addGlobal(\noteOn, 49+i, {|val|
{controls[i].valueAction_(controls[i].value+1)}.defer})}
					},{
						mantaIsSet = false;
						4.do{|i| manta.removeGlobal(\noteOn, 49+i)}
					})
				};
		mantaIsSet = false;
		win.front;

		win.onClose = {
			group.free;
			liveInterface.killMe;
			synthList.do{arg item; item.killMe}
		}
	}

	changeSetup {arg num;
		synthList.do{arg item; item.changeSetup(num)};
	}

	requestBusValidity {arg requestObject, bus;

		//add routing busses

		synthList.do{arg item, i;
			if(item.includes(requestObject),{
				synthLocation = [item.requestObjectXCoor(requestObject), i];
			});
		};
		availableBusses = List.new;
		if(synthLocation[1]>0,{
			synthLocation[1].do{arg i;
				availableBusses.addAll(synthList[i].getOutputBusses);
			};
		});


availableBusses.addAll(synthList[synthLocation[1]].getOutputBussesBeforeX(synthLocation[0]));
```

```
                //this is bad, but hopefully works for now
                availableBusses.addAll(ModularBusCenter.getBusses);

                if((soundInBusses.indexOfEqual(bus)!=nil) or:
(stereoSoundInBusses.indexOfEqual(bus)!=nil),{
                        ^true;
                },{
                        if(availableBusses.indexOfEqual(bus)!=nil,{^true},{^false})
                });
        }

        requestMixerBusValidity {arg requestObject, bus, location;
                availableBusses = List.new;
                if(location==nil,{
                        synthList.size.do{arg i;
                                availableBusses.addAll(synthList[i].getOutputBusses);
                        };
                },{
                        location[1].do{arg i;
                                availableBusses.addAll(synthList[i].getOutputBusses);
                        };

        availableBusses.addAll(synthList[location[1]].getOutputBussesBeforeX(location[0]));
                });

                availableBusses.addAll(ModularBusCenter.getBusses);

                if((soundInBusses.indexOfEqual(bus)!=nil) or:
(stereoSoundInBusses.indexOfEqual(bus)!=nil),{
                        ^true;
                },{
                        if(availableBusses.indexOfEqual(bus)!=nil,{^true},{^false})
                });
        }

        setName {arg name;
                win.name_(name);
        }

        save {arg xmlDoc, xmlMainProc;
                xmlBusses = xmlDoc.createElement("Busses");
                xmlMainProc.appendChild(xmlBusses);
                xmlSoundInBusses = xmlDoc.createElement("soundInBusses");
                8.do{arg i;
                        xmlSoundInBusses.setAttribute("soundInBus"++i.asString,
soundInBusses[i].asString);
```

```
		};
		xmlBusses.appendChild(xmlSoundInBusses);

		xmlSoundInBusses = xmlDoc.createElement("stereoSoundInBusses");
		4.do{arg i;
			xmlSoundInBusses.setAttribute("stereoSoundInBus"++i.asString,
stereoSoundInBusses[i].asString);
		};
		xmlBusses.appendChild(xmlSoundInBusses);

		xmlInternalBusses = xmlDoc.createElement("internalInBusses");
		availableBusses = List.new;
		synthList.do{arg item;
			availableBusses.addAll(item.getOutputBusses);
		};
		availableBusses.do{arg item, i;
			xmlInternalBusses.setAttribute("internalBus"++i.asString, item.asString);
		};
		xmlBusses.appendChild(xmlInternalBusses);
		xmlSynthBars = List.new;
		synthList.do{arg item, i;
			xmlSynthBars.add(xmlDoc.createElement("ModularSynthBar"++i.asString));
			xmlMainProc.appendChild(xmlSynthBars[i]);
			item.save(xmlDoc, xmlSynthBars[i]);
		};

		mantaData.do{arg item, i;
			xmlMainProc.setAttribute("manta"++i.asString, item.asString);
		};
		midiData.do{arg item, i;
			xmlMainProc.setAttribute("midi"++i.asString, item.asString);
		};

		xmlMainProc.setAttribute("bounds", win.bounds.asString);
		xmlMainProc.setAttribute("mantaIsSet", mantaIsSet.asString);
	}

	load {arg xmlMainProc;
		xmlBusses = xmlMainProc.getElement("Busses");
		xmlSoundInBusses = xmlMainProc.getElement("soundInBusses");
		soundInBussesTemp = List.newClear(8);
		8.do{arg i;
			soundInBussesTemp.put(i,xmlSoundInBusses.getAttribute("soundInBus"+
+i.asString));
		};
```

```
		xmlSoundInBusses = xmlMainProc.getElement("stereoSoundInBusses");
		stereoSoundInBussesTemp = nil;
		if(xmlSoundInBusses!=nil,{
			stereoSoundInBussesTemp = List.newClear(4);
			4.do{arg i;

	stereoSoundInBussesTemp.put(i,xmlSoundInBusses.getAttribute("stereoSoundInBus"+
+i.asString));
			};
		});

		availableBusses = List.new;
		synthList.do{arg item;
			availableBusses.addAll(item.getOutputBusses);
		};

		xmlInternalBusses = xmlMainProc.getElement("internalInBusses");
		availableBussesTemp = List.newClear(availableBusses.size);
		availableBusses.size.do{arg i;
			availableBussesTemp.put(i,xmlInternalBusses.getAttribute("internalBus"+
+i.asString));
		};

		busMap = IdentityDictionary.new;
		soundInBussesTemp.do{arg item, i;
			busMap.add(item.asSymbol -> soundInBusses[i]);
		};
		soundInBussesTemp.do{arg item, i;
			busMap.add(item.asSymbol -> soundInBusses[i]);
		};
		if(stereoSoundInBussesTemp!=nil,{
			stereoSoundInBussesTemp.do{arg item, i;
				busMap.add(item.asSymbol -> stereoSoundInBusses[i]);
			}
		});
		availableBussesTemp.do{arg item, i;
			item.postln;
			7.do{|i2|
				temp = item.asInteger+i2;
				busMap.add(temp.asSymbol -> (availableBusses[i].asInteger+i2))
			};
		};

		synthList.do{arg item, i;
			item.load(xmlMainProc.getElement("ModularSynthBar"++i.asString), busMap);
		};
```

```
                xmlBounds = xmlMainProc.getAttribute("bounds");
                if(xmlBounds!=nil,{win.bounds_(xmlBounds.interpret)});
                mantaIsSet = xmlMainProc.getAttribute("mantaIsSet");
                if(mantaIsSet!=nil,{
                        mantaIsSet = mantaIsSet.interpret;
                        if(mantaIsSet,{assignMantaButton.valueAction_(1)});
                },{
                        mantaIsSet = false;
                });

                midiData.size.do{arg i;
                        midiHidTemp = xmlMainProc.getAttribute("midi"++i.asString);
                        if(midiHidTemp!=nil,{
                                midiHidTemp = midiHidTemp.interpret;
                                if(midiHidTemp!=nil,{
                                        this.setMidi(midiHidTemp[0],midiHidTemp[1],i);
                                });
                        });
                };

                mantaData.size.do{arg i;
                        midiHidTemp = xmlMainProc.getAttribute("manta"++i.asString);
                        if(midiHidTemp!=nil,{
                                midiHidTemp = midiHidTemp.interpret;
                                if(midiHidTemp!=nil,{
                                        this.setManta(midiHidTemp[0], midiHidTemp[1], i);
                                });
                        });
                };
        }
}

ModularLiveInterface {
        var server, inBus, outBus, whichClassList, mainGroup, inGroup, synthGroup, mixerGroup,
volumeInRack, mainMixer, mainProcessingWindow;
        var manta, midiHid, mantaWindow, uc33Window, oxygen49Window, inBusses, inBusIndexes,
stereoInBusses, stereoInBusIndexes;
        var xmlDoc, xmlRoot, xmlMainProc, xmlMainMixer, xmlVolumeRack, file, bcf2000;

        *new {arg server, inBus=0, outBus=0, whichClassList;
                ^super.newCopyArgs(server, inBus, outBus, whichClassList).init;
        }

        init {
                server.waitForBoot({
                        //set up groups
```

```
mainGroup = Group.tail(server);
inGroup = Group.tail(mainGroup);
synthGroup = Group.tail(mainGroup);
mixerGroup = Group.tail(mainGroup);

//modularAudioRouter = ModularAudioRouter
ModularBusCenter.reset;

//set up the control devices and control GUI
manta = Manta.new;
(3..0).do{arg i; manta.store('setup'++i.asSymbol)};
manta.setup(\setup0);
MIDIResponder2.currentSetup_(\setup0);

if(MIDIClient.initialized.not,{ MIDIIn.connectAll });
bcf2000 = MIDIClient.destinations.detect({ |ep,epi|
        ep.device == "BCF2000" and: {ep.name == "Port 1"}
});
if(bcf2000!=nil,{bcf2000 = MIDIOut.newByName("BCF2000", "Port 1")});

midiHid = MIDIHIDControl.new;

//set up the inputs and outputs
inBusses = List.new;
inBusIndexes = List.new;
8.do{arg i;
        inBusses.add(Bus.audio(server,1));
        inBusIndexes.add(inBusses[i].index);
};
stereoInBusses = List.new;
stereoInBusIndexes = List.new;
4.do{arg i;
        stereoInBusses.add(Bus.audio(server,2));
        stereoInBusIndexes.add(stereoInBusses[i].index);
};

volumeInRack = ModularVolumeRack(inGroup, [inBusses, stereoInBusses],
midiHid, manta, bcf2000, [\setup0, \setup1, \setup2, \setup3]);
volumeInRack.init2(inBus);

if(whichClassList == nil, {whichClassList = 'normal'});

mainProcessingWindow = MainProcessingWindow(synthGroup, inBusIndexes,
stereoInBusIndexes, midiHid, manta, bcf2000, this, whichClassList);
mainMixer = MainMixer.new(mixerGroup, inBusIndexes, stereoInBusIndexes,
outBus, "Main Mixer", nil, 8, midiHid, manta, bcf2000, [\setup0, \setup1, \setup2, \setup3],
```

```
mainProcessingWindow);
		});
	}

	changeSetup {arg num;
		manta.setup('setup'++num.asSymbol);
		MIDIResponder2.currentSetup_(('setup'++num.asSymbol).asSymbol);
		mainProcessingWindow.changeSetup(num);
		mainMixer.changeSetup(num);
	}

	save {arg path;
		mainProcessingWindow.setName(path.basename);

		xmlDoc = DOMDocument.new; // create empty XML document
		xmlRoot = xmlDoc.createElement("ModularLiveInterface");
		xmlDoc.appendChild(xmlRoot);

		xmlMainProc = xmlDoc.createElement("MainProcessingWindow");
		xmlRoot.appendChild(xmlMainProc);
		mainProcessingWindow.save(xmlDoc, xmlMainProc);

		xmlMainMixer = mainMixer.saveMainMixer(xmlDoc);
		xmlRoot.appendChild(xmlMainMixer);

		xmlVolumeRack = volumeInRack.save(xmlDoc);
		xmlRoot.appendChild(xmlVolumeRack);

		file = File(path, "w");
		xmlDoc.write(file); // output to file with default formatting
		file.close;
	}

	load {arg path;
		xmlDoc = DOMDocument.new(path);

		mainProcessingWindow.setName(path.basename);

		xmlMainProc = xmlDoc.getDocumentElement.getElement("MainProcessingWindow");
		mainProcessingWindow.load(xmlMainProc);

		xmlMainMixer =  xmlDoc.getDocumentElement.getElement("ModularMainMixer");
		mainMixer.load(xmlMainMixer, mainProcessingWindow.busMap);

		xmlVolumeRack = xmlDoc.getDocumentElement.getElement("ModularVolumeRack");
		if(xmlVolumeRack!=nil,{
```

```
                volumeInRack.load(xmlVolumeRack);
            });
        }

        killMe {
            mainMixer.killMe;
            volumeInRack.killMe;
            manta.clearDicts;
            mainGroup.free;
            inBusses.free;
            MIDIResponder.removeAll;
        }
}

ModularVolumeObject {
        var <group, <win, <bounds, <inBus, <outBus, <stereoOutBus, top, synth, volInBus, volIn;

        *new {arg group, win, bounds, inBus, outBus, stereoOutBus;
            ^super.newCopyArgs(group, win, bounds, inBus, outBus, stereoOutBus).init;
        }

        *initClass {
            {
                SynthDef("modularInput", {arg inBus, outBus, stereoOutBus, volInBus, vol=1,
gate=1;
                    var in, env;

                    env = EnvGen.kr(Env.asr(0,1,0), gate);

                    in = SoundIn.ar(inBus)*env;

                    Out.kr(volInBus, LagUD.kr(Amplitude.kr(in), 0.01, 0.5));
                    Out.ar(outBus, in*Lag.kr(vol,0.01));
                    Out.ar(stereoOutBus, in*Lag.kr(vol,0.01));
                }).writeDefFile;
            }.defer(1);
        }

        init {
            volIn = 0;
            volInBus = Bus.control(group.server);
            synth = Synth("modularInput", [\inBus, inBus, \outBus, outBus, \stereoOutBus,
stereoOutBus, \volInBus, volInBus.index], group);
        }

        update {
```

```
                volInBus.get({arg val;
                        volIn = val;
                });
                Color.black.set;
                Pen.strokeRect(bounds);
                if(volIn>=1,{Color.red.set},{Color.green.set});
                top = bounds.top + (bounds.height-((volIn*bounds.height)));
                Pen.fillRect(Rect(bounds.left+2, top, bounds.width-4, volIn*bounds.height-4));
        }

        setVol {arg vol;
                synth.set(\vol, vol);
        }

        mute {
                synth.set(\gate, 0);
        }

        unmute {
                synth.set(\gate, 1);
        }
}

ModularVolumeRack : Module_Mod {
        var dispArray, win, <>outBusses, run;

        init {}

        init2 {arg inBusOffset;
                outBus.postln;

                run = true;
                win = SCWindow("Inputs", Rect(500, SCWindow.screenBounds.height,
60*outBus.size+10, 120));
                win.userCanClose_(false);
                win.alwaysOnTop_(true);

                modName = "ModularVolumeRack";
                this.initControlLists(outBus[0].size*3);

                controls = List.new;
                assignButtons = List.new;

                outBus[0].do{arg item, i;
                        dispArray = dispArray.add(ModularVolumeObject(group, win,
Rect(i*60+5,0,60,80), i+inBusOffset, item.index, outBus[1][(i/2).floor.asInteger].index+(i%2)));
```

```
                        //for lightpipe
                        //dispArray = dispArray.add(ModularVolumeObject(group, win,
Rect(i*60+5,0,60,80), i+14, item));

                        controls.add(SCButton.new(win,Rect(60*i, 80, 20, 20))
                                .states_([ [ "U", Color.red, Color.black ] ,[ "U", Color.black,
Color.blue ] ])
                                .action_{|v|
                                        dispArray[i].unmute;
                                        controls[i*3+1].value = 0;
                                        controls[i*3].value = 1;
                                });
                        this.addAssignButton(Rect(60*i, 100, 20, 20),i*3);
                        controls.add(SCButton.new(win,Rect(60*i+20, 80, 20, 20))
                                .states_([ [ "M", Color.red, Color.black ] ,[ "M", Color.black,
Color.blue ] ])
                                .action_{|v|
                                        dispArray[i].mute;
                                        controls[i*3].value = 0;
                                        controls[i*3+1].value = 1;
                                });
                        this.addAssignButton(Rect(60*i+20, 100, 20, 20),i*3+1);

                        controls.add(EZSlider(win, Rect(60*i+5, 120, 60, 120), "Amp",
ControlSpec(0.001, 2, \amp),
                                {arg slider;
                                        dispArray[i].setVol(slider.value);
                                }, 1, true, layout:\vert));
                        this.addDoubleAssignButton(Rect(60*i+5, 240, 60, 20),2+(3*i),1);

                        controls[i*3].value = 1;
                };

                win.front;
                win.drawHook = {
                        dispArray.do{arg item, i;
                                item.update;
                        }
                };
                { while { run } { win.refresh; 0.1.rand.wait } }.fork(AppClock);
        }

        killMe {
                group.free;
                run = false;
```

```
                win.close;
        }
}

ModularMixerStrip {
        var group, soundInBusses, stereoSoundInBusses, outBus, location, win, point, color,
<>midiHidControl, manta, bcf2000, <>setups, mainProcessingWindow, assignedChannelsArray, mixer,
index, ampSpec, ccResponders, midiData, mantaData, inputBusses, discardBusses, inputBusString,
assignInputButton, inputDisplay, volSlider, assignMidiButton, muteButton, sepInputBusses,
xmlMixerStrip, inBusTemp, inBusTempList, counter, label, numBussesMenu, numBusses,
channelOutMenu, temp, channelOutTemp, panKnob, busAssignSink, panel, waitForSet, waitForType;

        *new {arg group, soundInBusses, stereoSoundInBusses, outBus, location, win, point, color,
midiHidControl, manta, bcf2000, setups, mainProcessingWindow;
                ^super.newCopyArgs(group, soundInBusses, stereoSoundInBusses, outBus, location,
win, point, color, midiHidControl, manta, bcf2000, setups, mainProcessingWindow).init;
        }

        init {

                inputBusses = List.new;

                mixer = ModularMixer(group, soundInBusses, stereoSoundInBusses);
                mixer.setMainMixerOutBusses([outBus]);

                panel = SCCompositeView.new(win, Rect(point.x, point.y, 50, 300));
                if(color!=nil,{
                        panel.background_(color)
                });

                ampSpec = ControlSpec(0,1,'amp');

                busAssignSink=MixerBusAssignSink(this, panel, Point(0,0));

                panKnob = EZKnob(panel,Rect(0, 50, 50, 50), nil, ControlSpec(-1,1,'linear'),
                        {arg knob; mixer.setPan(knob.value)}, 0, true, labelHeight:0);

                volSlider = SCSlider.new(panel,Rect(0, 100, 50, 120))
                        .action_{|v|
                                mixer.setVol(ampSpec.map(v.value));
                        };
                this.addDoubleAssignButton(Rect(0, 220, 50, 16), 0, 1);

                label = SCTextField.new(panel, Rect(0, 246, 50, 16));

                numBusses = 8;
```

```
        channelOutMenu = SCPopUpMenu.new(panel, Rect(0, 262, 50, 16));
        channelOutTemp = Array.series(8,0,1);
        channelOutTemp.do{arg item, i;  channelOutTemp.put(i, item.asString)};
        channelOutMenu.items = channelOutTemp;
        channelOutMenu.action = {arg pop;
                mixer.setMainMixerOutBusses([outBus+pop.value]);
                mixer.mixers.do{arg item; item.set(\outBus, outBus+pop.value)};
        };

        waitForSet = false;

        ccResponders = (
                setup0: nil,
                setup1: nil,
                setup2: nil,
                setup3: nil
        );
}

addDoubleAssignButton {|rect, num, type, layout = \horz, specialClearAction = nil|
        assignMidiButton = DoubleAssignButton.new(panel,rect,layout)
                .instantAction_{|butt|
                        if(butt.value==1,{
                                midiHidControl.requestInstantMidiAssignInfo(this, num);
                                waitForSet = true;
                                waitForType = type;
                                if(type == 3, {waitForType = 0});

                        },{
                                midiHidControl.clearInstantMidiAssignInfo;
                                if(specialClearAction.isNil,{
                                        this.clearMidiHid(num);
                                },{
                                        specialClearAction.value;
                                });
                                waitForSet = false;
                        })
                };
}

setControlWNext {arg controllerType, dataType, controlsIndex, controlInfo;
        //dataType - 0 is onOff info, 1 is continuous
        //controllerType - 0 is Manta, 1 is Midi
        "setControlWNext".postln;
        [controllerType, dataType, controlsIndex, controlInfo].postln;
```

```
		if(dataType.postln==waitForType, {
			waitForSet = false;
			if(controllerType.postln ==0,{
				this.setManta(controlInfo[0], controlInfo[1], controlsIndex);
			},{
				this.setMidi(controlInfo[0], controlInfo[1], controlsIndex);
			});
			midiHidControl.clearInstantMidiAssignInfo;
		})
}

mute {
		mixer.mute;
}

unmute {
		mixer.unmute;
}

hide {
		panel.visible = false;
}

unhide {
		panel.visible = true;
}

requestBusValidity {arg bus;
		^mainProcessingWindow.requestMixerBusValidity(this, bus, location);
}

createSepInputBusses {arg inputBusses;
		sepInputBusses = List.new;
		2.do{sepInputBusses.add(List.new)};
		inputBusses.do{arg item, i;
			index = soundInBusses.indexOfEqual(item);
			if(index==nil,{
				sepInputBusses[1].add(item)
			},{
				sepInputBusses[0].add(index)
			});
		};
}

setInputBusses {arg inputBussesIn;
		"setInputBusses".post;[inputBussesIn, numBusses].postln;
```

```
            inputBusses = mixer.setInputBusses(inputBussesIn, numBusses);
        }

        updateInputBusGUI { arg inputBussesIn;
            inputBussesIn[0].do{arg item, i; item = "S"++item.asString; inputBussesIn[0].put(i,
item)};
            inputBussesIn = inputBussesIn.flatten;
            inputBusString = "";
            inputBussesIn.do{arg item; inputBusString = inputBusString+item.asString};
            inputDisplay.string = inputBusString;
        }

        clearMidiHid {
            ccResponders.do{|ccResponder|
                if(ccResponder!=nil,{
                    ccResponder.remove;
                    ccResponder=nil;
                });
            };
            midiData = nil;
            if(manta!=nil,{
                if(mantaData!=nil,{
                    switch(mantaData[1],
                        0,{
                            setups.do{arg setup;
manta.removeNoteOnSetup(setup.asSymbol, mantaData[0])};
                        },
                        1,{
                            setups.do{arg setup;
manta.removePadSetup(setup.asSymbol, mantaData[0])};
                        },
                        2,{
                            setups.do{arg setup;
manta.removeSliderSetup(setup.asSymbol, mantaData[0])};
                        }
                    );
                    mantaData = nil;
                });
            });
        }

        assignChannel {arg channelIn;
            index = assignedChannelsArray.indexOfEqual(channelIn);
            if(index==nil,{
                assignedChannelsArray.add(channelIn);
                mixer.addBusPair([channelIn, 0]);
```

```
        });
}

removeChannel {arg channelIn;
        index = assignedChannelsArray.indexOfEqual(channelIn);
        if(index==nil,{
                assignedChannelsArray.removeAt(channelIn);
                mixer.removeBusPair([channelIn, 0]);
        });
}

setDefaultMidi {arg data, dataType, controlsIndex;
        this.clearMidiHid;
        this.setMidi(data, dataType, controlsIndex);
        assignMidiButton.value_(1);
}

setMidiForSetup {|setup, data|
        "data".post;data.postln;
        ccResponders.put(setup.asSymbol, CCResponder2({ |src,chan,num,val|
                        {volSlider.value_(val/127)}.defer;
                        mixer.setVol(ampSpec.map(val/127));
                },
                nil, // any source
                data[0], // desired channel
                data[1], // desired number
                nil, // any value
                setup.asSymbol
        ))
}

setMidi {arg data, dataType;
        midiData = [data,0];   //0 means control data
        setups.do{arg setup;
                switch(dataType,
                        0,{
                                this.setMidiForSetup(setup, data);
                        },
                        1,{
                                "you shouldn't do that!".postln;
                        }
                )
        };
}

setStoredMIDI {
```

```
                "setStoredMIDI".postln;
                if(bcf2000!=nil,{
                        if(midiData!=nil,{
                                bcf2000.control(midiData[0][0],midiData[0][1],
(ampSpec.unmap(volSlider.value)*127).ceil);
                        })
                })
        }

        removeSetup {arg setup;
                setups.remove(setup);
                if(mantaData!=nil,{
                        switch(mantaData[1],
                                0,{
                                        manta.removeNoteOnSetup(setup.asSymbol, mantaData[0]);
                                },
                                1,{
                                        manta.removePadSetup(setup.asSymbol, mantaData[0]);
                                },
                                2,{
                                        manta.removeSliderSetup(setup.asSymbol, mantaData[0]);
                                }
                        )
                });
                if(midiData!=nil,{
                        ccResponders[setup.asSymbol].remove;
                });
                setups.postln;
        }

        addSetup {arg setup;
                "addSetupMixer".postln;
                midiData.postln;
                setups.add(setup);
                if(mantaData!=nil,{this.setMantaForSetup(setup, mantaData)});
                if(midiData!=nil,{this.setMidiForSetup(setup, midiData[0])});
                setups.postln;
        }

        setMantaForSetup {arg setup, item;
                [setup, item].postln;
                if(item!=nil,{
                        switch(item[1],
                                0,{
                                        "whaaaaat?  don't do that".postln;
                                },
```

```
                                1,{
                                        manta.addPadSetup(setup.asSymbol, item[0], {|val|
{volSlider.valueAction_(ampSpec.map(val/180))}.defer});
                                },
                                2,{
                                        manta.addSliderSetup(setup.asSymbol, item[0], {|val|
{volSlider.valueAction_(ampSpec.map(val/4096))}.defer});
                                }
                        );
                });
        }

        setManta {arg buttonNum, buttonType;
                if(manta!=nil,{
                        mantaData=[buttonNum.deepCopy, buttonType.deepCopy];
                        mantaData.postln;
                        setups.do{arg setup;
                                this.setMantaForSetup(setup, mantaData)
                        }
                })
        }

        getMidiHid {
                ^[midiData, mantaData];
        }

        save {arg xmlDoc, xmlSynth, index;
                xmlMixerStrip = xmlDoc.createElement("Mixer"++index.asString);
                xmlMixerStrip.setAttribute("manta", mantaData.asString);
                xmlMixerStrip.setAttribute("midi", midiData.asString);
                xmlMixerStrip.setAttribute("volSlider", volSlider.value.asString);
                inputBusses.do{arg item,i;
                        xmlMixerStrip.setAttribute("inBus"++i.asString, item.asString);
                };
                xmlMixerStrip.setAttribute("outBus", channelOutMenu.value.asString);

                if(label.string!=nil,{xmlMixerStrip.setAttribute("label", label.string)});
                xmlSynth.appendChild(xmlMixerStrip);
        }

        load {arg xmlMainMixer, busMap, index;
                soundInBusses.postln; busMap.postln;
                if(xmlMainMixer!=nil,{
                        xmlMixerStrip = xmlMainMixer.getElement("Mixer"++index.asString);

                        temp = xmlMixerStrip.getAttribute("manta").interpret;
```

```
                        if (temp!=nil, {this.setManta(temp[0], temp[1], 0)});
                        temp = xmlMixerStrip.getAttribute("midi").interpret;
                        if (temp!=nil, {this.setMidi(temp[0], temp[1], 0)});
                        if(xmlMixerStrip.getAttribute("label")!=nil,{label.string =
xmlMixerStrip.getAttribute("label")});
                        counter = 0;

                        channelOutTemp = xmlMixerStrip.getAttribute("outBus").interpret;
                        if(channelOutTemp!=nil,{
                                "channelOutTemp".post; channelOutTemp.postln;

                                channelOutMenu.valueAction_(channelOutTemp)
                        });

                        temp = xmlMixerStrip.getAttribute("volSlider");
                        if (temp!=nil, {volSlider.valueAction = temp.interpret});

                        //loads the desired bus
                        inBusTemp = xmlMixerStrip.getAttribute("inBus"++counter.asString);

                        inBusTempList = List.new;
                        while({inBusTemp!=nil},{
                                counter.postln;
                                inBusTempList.add(busMap[inBusTemp.interpret.asSymbol]);
                                counter = counter + 1;
                                inBusTemp = xmlMixerStrip.getAttribute("inBus"+
+counter.asString).postln;
                        });
                        inBusTempList.do{arg item;
                                "bus".post;item.postln;
                                soundInBusses.postln;
                                if(soundInBusses.indexOf(item).postln!=nil,
                                {
                                        "soundInBus".postln;
                                        busAssignSink.assignBus(item,
"S"+soundInBusses.indexOf(item).asString);
                                },{
                                        if(stereoSoundInBusses.indexOf(item)!=nil,
                                        {
                                                "stereoSoundInBus".postln;
                                                busAssignSink.assignBus(item, "S"+
+stereoSoundInBusses.indexOf(item).asString++((stereoSoundInBusses.indexOf(item)+1).asString));
                                        },{
                                                "assignNormalBus".postln;
                                                busAssignSink.assignBus(item, item.asString);
                                        })
```

```
                        })
                    };
                });
        }
}

MainMixer {
        var group, soundInBusses, stereoSoundInBusses, outBus, name, location, numMixers,
midiHidControl, manta, bcf2000, <>setups, mainProcessingWindow, mixerStrips, window,
assignDefaultsButton, midiHidArray, xmlSynth, currentLayer, xmlMixerLayer;

        *new {arg group, soundInBusses, stereoSoundInBusses, outBus, name, location, numMixers,
midiHidControl, manta, bcf2000, setups, mainProcessingWindow;
                ^super.newCopyArgs(group, soundInBusses, stereoSoundInBusses, outBus, name,
location, numMixers, midiHidControl, manta, bcf2000, setups, mainProcessingWindow).init;
        }

        init {
                window = SCWindow(name, Rect(0, 0, 10+(numMixers+1*55), 278));
                window.userCanClose_(false);
                window.alwaysOnTop_(true);

                mixerStrips = List.new;
                4.do{|i|
                        mixerStrips.add(List.new);
                        ((numMixers/2)..numMixers).do{arg i2;
mixerStrips[i].add(ModularMixerStrip(group, soundInBusses, stereoSoundInBusses, outBus, location,
window, Point(5+(i2*55), 0), [Color.blue, Color.green, Color.red,
Color.magenta].at(i).multiply(0.5).alpha_(0.25), midiHidControl, manta, bcf2000, setups[i],
mainProcessingWindow))};
                };
                mixerStrips.add(List.new);
                (numMixers/2).do{arg i2; mixerStrips[4].add(ModularMixerStrip(group,
soundInBusses, stereoSoundInBusses, outBus, location, window, Point(5+(i2*55), 0), nil,
midiHidControl, manta, bcf2000, setups, mainProcessingWindow))};

                (1..3).do{|i| mixerStrips[i].do{|item|item.mute;item.hide}};
                currentLayer = 0;

                window.front;
        }

        changeSetup {arg num;
                mixerStrips[currentLayer].do{|item| item.mute; item.hide};
                currentLayer = num;
                mixerStrips[currentLayer].do{|item| item.unmute; item.unhide; item.setStoredMIDI};
```

```
	}

	killMe {
		"kill the mixer";
		group.free;
		window.close;
	}

	saveMainMixer {arg xmlDoc;
		xmlSynth = xmlDoc.createElement("ModularMainMixer");
		xmlSynth.setAttribute("bounds", window.bounds.asString);

		5.do{arg i;
			xmlMixerLayer = xmlDoc.createElement("MixerLayer"++i.asString);
			mixerStrips[i].do{arg item, i2; item.save(xmlDoc, xmlMixerLayer, i2)};
			xmlSynth.appendChild(xmlMixerLayer);
		}
		^xmlSynth;
	}

	load {arg xmlMainMixer, busMap;
		if(xmlMainMixer.getElement("MixerLayer0")!=nil,{
			5.do{arg i;
				xmlMixerLayer = xmlMainMixer.getElement("MixerLayer"++i.asString);
				xmlMixerLayer.postln;
				mixerStrips[i].do{arg item, i; item.load(xmlMixerLayer, busMap, i)};
			};
		},{
			mixerStrips[0].do{arg item, i; item.load(xmlMainMixer, busMap, i)};
		});
		window.bounds_(xmlMainMixer.getAttribute("bounds").interpret);
		window.front;
	}


	show {
		window.visible = true;
	}

	hide {
		window.visible = false;
	}
}

ModularMainMixer : MainMixer {
```

```
	init {
		window = SCWindow(name, Rect(0, 0, 10+(numMixers+1*55), 280));
		window.userCanClose_(false);
		window.alwaysOnTop_(true);

		mixerStrips = List.new;
		numMixers.do{arg i; mixerStrips.add(ModularMixerStrip(group, soundInBusses,
stereoSoundInBusses, outBus, location, window, Point(5+(i*55), 0), nil, midiHidControl, manta,
bcf2000, setups, mainProcessingWindow))};

		window.front;
	}

	setStoredMIDI {arg num;
		mixerStrips.do{|item| item.setStoredMIDI};
	}

	load {arg xmlMainMixer, busMap;
		mixerStrips.do{arg item, i; item.load(xmlMainMixer, busMap, i)};

		window.bounds_(xmlMainMixer.getAttribute("bounds").interpret);
		window.front;
	}

	addSetup {|setup|
		mixerStrips.do{|item| item.addSetup(setup.asSymbol)}
	}

	removeSetup {|setup|
		mixerStrips.do{|item| item.removeSetup(setup.asSymbol)}
	}

	save {arg xmlDoc;
		xmlSynth = xmlDoc.createElement("Mixer");
		xmlSynth.setAttribute("bounds", window.bounds.asString);
		mixerStrips.do{arg item, i; item.save(xmlDoc, xmlSynth, i)};
		^xmlSynth;
	}

	pause {
		mixerStrips.do{|item| item.mute};
	}

	unpause {
		mixerStrips.do{|item| item.unmute};
	}
```

```
	show {
		window.visible = true;
	}

	hide {
		window.visible = false;
	}
}

SignalSwitcher_Mod : Module_Mod {
	var localBusses, synths, controls, soundInBusses, stereoSoundInBusses, location,
mainProcessingWindow, mixerGroup, synthGroup, mixerStrips, pulseRate, impulseOn, dustOn,
currentOnChan;

	*initClass {
		{
			SynthDef("signalSwither_mod", {arg inBus0, inBus1, outBus, whichModulator
= 0, pulseRate0=0, pulseRate1=0, onBypass=0, gate = 1, pauseGate = 1;
				var in0, in1, env, out, impulse, dust, whichSignal, pauseEnv;

				impulse = Impulse.kr(pulseRate0);
				dust = Dust.kr(pulseRate1);

				whichSignal = Lag.kr(Select.kr(whichModulator, [0, 1,
Stepper.kr(impulse+dust, 0, 0, 1, 1, 0)]), 0.001);

				//Out.ar(0,In.ar(inBus0, 2));

				out = (In.ar(inBus0, 8)*(1-whichSignal))+(In.ar(inBus1, 8)*whichSignal);

				env = EnvGen.kr(Env.asr(0,1,0), gate, doneAction:2);
				pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

				Out.ar(outBus, out*env*pauseEnv);
			}).writeDefFile;
		}.defer(1);
	}

	init {}

	init2 {|soundInBusses, stereoSoundInBusses, location, mainProcessingWindow|
		[soundInBusses, stereoSoundInBusses, location, mainProcessingWindow].postln;

		this.makeWindow("SignalSwitcher",Rect(0, 0, 10+(3*55), 294));
		this.initControlLists(5);
```

```
		synths = List.new;

		controls = List.new;
		assignButtons = List.new;

		mixerGroup = Group.tail(group);
		synthGroup = Group.tail(group);

		localBusses = List.new;
		2.do{localBusses.add(Bus.audio(group.server, 8))};
		localBusses.postln;

		mixerStrips = List.new;
		2.do{arg i; mixerStrips.add(SwitcherMixer_Mod(mixerGroup, soundInBusses,
stereoSoundInBusses, localBusses[i].index, location, win, Point(5+(i*55), 0), nil, midiHidControl,
manta, bcf2000, setups, mainProcessingWindow))};

		"outBus".post;outBus.postln;

		synths.add(Synth("signalSwither_mod", [\inBus0, localBusses[0].index, \inBus1,
localBusses[1], \outBus, outBus], synthGroup));
		currentOnChan = 0;

		dustOn = false;
		impulseOn = false;

		controls.add(SCButton(win, Rect(5, 280, 75, 16))
			.states_([["left", Color.blue, Color.black],["left", Color.black, Color.blue]])
			.action_({arg butt;
				impulseOn = false;
				dustOn = false;
				currentOnChan = 0;
				synths[0].set(\pulseRate0, 0, \pulseRate1, 0, \whichModulator, 0);
				butt.value_(1);
				controls[1].value_(0);
				controls[2].value_(0);
				controls[3].value_(0);
			})
		);

		controls.add(SCButton(win, Rect(85, 280, 75, 16))
			.states_([["right", Color.blue, Color.black],["right", Color.black, Color.blue]])
			.action_({arg butt;
				impulseOn = false;
				dustOn = false;
```

```
                        currentOnChan = 1;
                        synths[0].set(\pulseRate0, 0, \pulseRate1, 0, \whichModulator, 1);
                        butt.value_(1);
                        controls[0].value_(0);
                        controls[2].value_(0);
                        controls[3].value_(0);
                    })
                );

                controls[0].value = 1;

                this.addDoubleAssignButton(Rect(5, 300, 75, 16), 0, 0);
                this.addDoubleAssignButton(Rect(85, 300, 75, 16), 1, 0);

                controls.add(SCButton(win, Rect(5, 320, 75, 16))
                    .states_([["impulse", Color.blue, Color.black],["impulse", Color.black,
Color.blue]])
                    .action_({arg butt;
                        if(impulseOn,{
                            impulseOn = false;
                            if(dustOn,{
                                synths[0].set(\pulseRate0, 0, \pulseRate1,
rrand(pulseRate[0], pulseRate[1])*2, \whichModulator, 2);
                            },{
                                synths[0].set(\pulseRate0, 0, \pulseRate1, 0,
\whichModulator, currentOnChan);
                                if(currentOnChan==0,{controls[0].value_(1)},
{controls[1].value_(1)});
                            });
                            butt.value_(0);
                        },{
                            impulseOn = true;
                            if(dustOn,{
                                synths[0].set(\pulseRate0, rrand(pulseRate[0],
pulseRate[1]), \pulseRate1, rrand(pulseRate[0], pulseRate[1]), \whichModulator, 2);
                            },{
                                synths[0].set(\pulseRate0, rrand(pulseRate[0],
pulseRate[1])*2, \pulseRate1, 0, \whichModulator, 2);
                            });
                            butt.value_(1);
                            controls[0].value_(0);
                            controls[1].value_(0);
                        })
                    })
                );
```

```
		controls.add(SCButton(win, Rect(85, 320, 75, 16))
			.states_([["dust", Color.blue, Color.black],["dust", Color.black, Color.blue]])
			.action_({arg butt;
				if(dustOn,{
					dustOn = false;
					if(impulseOn,{
						synths[0].set(\pulseRate0, rrand(pulseRate[0],
pulseRate[1])*2, \pulseRate1, 0, \whichModulator, 2);
					},{
						synths[0].set(\pulseRate0, 0, \pulseRate1, 0,
\whichModulator, currentOnChan);
						if(currentOnChan==0,{controls[0].value_(1)},
{controls[1].value_(1)});
					});
					butt.value_(0);
				},{
					dustOn = true;
					if(impulseOn,{
						synths[0].set(\pulseRate0, rrand(pulseRate[0],
pulseRate[1]), \pulseRate1, rrand(pulseRate[0], pulseRate[1]), \whichModulator, 2);
					},{
						synths[0].set(\pulseRate0, 0, \pulseRate1,
rrand(pulseRate[0], pulseRate[1])*2, \whichModulator, 2);
					});
					butt.value_(1);
					controls[0].value_(0);
					controls[1].value_(0);
				})
			})
		);


		this.addDoubleAssignButton(Rect(5, 340, 75, 16), 2, 0);
		this.addDoubleAssignButton(Rect(85, 340, 75, 16), 3, 0);

		controls.add(EZRanger(win, Rect(115, 5, 50, 275), "speed", ControlSpec(0.25, 30,
'linear'),
			{arg val;
				pulseRate = val.value.postln;
				if(impulseOn&&dustOn,{
					synths[0].set(\pulseRate0, rrand(pulseRate[0], pulseRate[1]),
\pulseRate1, rrand(pulseRate[0], pulseRate[1]));
				},{
					if(impulseOn,{
						synths[0].set(\pulseRate0, rrand(pulseRate[0],
pulseRate[1])*2);
```

```
                                        },{
                                                if(dustOn,{
                                                        synths[0].set(\pulseRate1, rrand(pulseRate[0],
pulseRate[1])*2);
                                                })
                                        })
                                })
                        }, [4, 7], true, layout:\vert));

                win.front;
        }

        addSetup {|setup|
                mixerStrips.postln;
                mixerStrips.do{|item| item.postln; item.addSetup(setup.asSymbol)};

                setups.add(setup);
                mantaData.postln;
                midiData.postln;
                mantaData.do{arg item, i;
                        this.setMantaForSetup(setup, item, i);
                };
                midiData.do{arg item, i;
                        if(item!=nil,{
                                this.setMidiForSetup(setup, item[0], item[1], i);
                        });
                };
                setups.postln;
        }

        setStoredMIDI {
                mixerStrips.do{|item| item.setStoredMIDI};
        }

        save {arg xmlDoc;
                xmlSynth = xmlDoc.createElement(modName);
                mantaData.do{arg item, i;
                        xmlSynth.setAttribute("manta"++i.asString, item.asString);
                };
                midiData.do{arg item, i;
                        xmlSynth.setAttribute("midi"++i.asString, item.asString);
                };
                controls.do{arg item, i;
                        xmlSynth.setAttribute("controls"++i.asString, item.value.asString);
                };
                xmlSynth.setAttribute("bounds", win.bounds.asString);
```

```
		mixerStrips.do{arg item, i; item.save(xmlDoc, xmlSynth, i)};
		^xmlSynth;
	}

	load {arg xmlSynth, busMap;
		this.loadMidiData(xmlSynth);
		mantaData.size.do{arg i;
			midiHidTemp = xmlSynth.getAttribute("manta"++i.asString);
			if(midiHidTemp!=nil,{
				midiHidTemp = midiHidTemp.interpret;
				if(midiHidTemp!=nil,{
					this.setManta(midiHidTemp[0], midiHidTemp[1], i);
				});
			});
		};
		controls.do{arg item, i;
			midiHidTemp = xmlSynth.getAttribute("controls"++i.asString);
			if(midiHidTemp!=nil,{
				controls[i].valueAction_(midiHidTemp.interpret);
			});
		};
		mixerStrips.do{arg item, i; item.load(xmlSynth, busMap, i)};
		win.bounds_(xmlSynth.getAttribute("bounds").interpret);
		win.front;
	}

	killMeSpecial {
		"kill the mixer";
		localBusses.do{arg item; item.free};
		synthGroup.free;
		mixerGroup.free;
	}

}

SignalSwitcher4_Mod : Module_Mod {
	var localBusses, synths, controls, soundInBusses, stereoSoundInBusses, location,
mainProcessingWindow, mixerGroup, synthGroup, mixerStrips, pulseRate, impulseOn, dustOn,
currentOnChan, randArray, seqArray, seqArray2, seqArray3, temp, arrayIsGood, stepperOn, randOn,
seqOn, whichSeq;

	*initClass {
		{
			SynthDef("signalSwither4_mod", {arg inBus0, inBus1, inBus2, inBus3, outBus,
stepTo = 3, whichModulator = 0, whichSelector=0, pulseRate=0, onBypass=0, gate = 1, pauseGate = 1,
```

```
whichSeqArray=0, seqArray0 = #[0,1,2,3], seqArray1 = #[0,1,2,3,0], seqArray2 = #[0,1,2,3,0,1,2];
                                var in0, in1, env, out, impulse, dust, whichSignal, pauseEnv, stepper,
rand, pattern, selector, env0, env1, env2, env3;

                                impulse = Impulse.kr(pulseRate);

                                stepper = Stepper.kr(impulse, 0, 0, stepTo, 1, 0);
                                rand = Demand.kr(impulse, 0, Dxrand([0,1,2,3], inf));

                                pattern = Select.kr(whichSeqArray, [Demand.kr(impulse, 0,
Dseq(seqArray0, inf)), Demand.kr(impulse, 0, Dseq(seqArray1, inf)), Demand.kr(impulse, 0,
Dseq(seqArray2, inf))]);

                                selector = Select.kr(whichSelector, [stepper, rand, pattern.poll]);

                                whichSignal = Select.kr(whichModulator, [0, 1, 2, 3, selector]).poll;

                                env0 = Select.kr(whichSignal, [1,0,0,0]);
                                env1 = Select.kr(whichSignal, [0,1,0,0]);
                                env2 = Select.kr(whichSignal, [0,0,1,0]);
                                env3 = Select.kr(whichSignal, [0,0,0,1]);

                                out = (In.ar(inBus0, 8)*Lag.kr(env0, 0.01))+(In.ar(inBus1,
8)*Lag.kr(env1, 0.01))+(In.ar(inBus2, 8)*Lag.kr(env2, 0.01))+(In.ar(inBus3, 8)*Lag.kr(env3, 0.01));

                                env = EnvGen.kr(Env.asr(0,1,0), gate, doneAction:2);
                                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

                                Out.ar(outBus, out*env*pauseEnv);
                        }).writeDefFile;
                }.defer(1);
        }

        init {}

        init2 {|soundInBusses, stereoSoundInBusses, location, mainProcessingWindow|
                [soundInBusses, stereoSoundInBusses, location, mainProcessingWindow].postln;

                this.makeWindow("SignalSwitcher4",Rect(0, 0, 10+(5*55), 345));
                this.initControlLists(5);

                synths = List.new;

                controls = List.new;
                assignButtons = List.new;
```

```
		mixerGroup = Group.tail(group);
		synthGroup = Group.tail(group);

		localBusses = List.new;
		4.do{localBusses.add(Bus.audio(group.server, 8))};
		localBusses.postln;

		mixerStrips = List.new;
		4.do{arg i; mixerStrips.add(SwitcherMixer_Mod(mixerGroup, soundInBusses,
stereoSoundInBusses, localBusses[i], location, win, Point(5+(i*64), 0), nil, midiHidControl, manta,
bcf2000, setups, mainProcessingWindow))};

		"outBus".post;outBus.postln;

		synths.add(Synth("signalSwither4_mod", [\inBus0, localBusses[0].index, \inBus1,
localBusses[1], \inBus2, localBusses[2].index, \inBus3, localBusses[3], \outBus, outBus],
synthGroup));
		currentOnChan = 0;

		dustOn = false;
		impulseOn = false;

		controls.add(SCButton(win, Rect(5, 260, 75, 16))
			.states_([["0", Color.blue, Color.black],["0", Color.black, Color.blue]])
			.action_({arg butt;
				stepperOn = false;
				seqOn = false;
				randOn = false;
				currentOnChan = 0;
				synths[0].set(\pulseRate0, 0, \whichModulator, 0);
				butt.value_(1);
				this.setValue0(0);
			})
		);

		controls.add(SCButton(win, Rect(85, 260, 75, 16))
			.states_([["1", Color.blue, Color.black],["1", Color.black, Color.blue]])
			.action_({arg butt;
				stepperOn = false;
				seqOn = false;
				randOn = false;
				currentOnChan = 1;
				synths[0].set(\pulseRate0, 0, \whichModulator, 1);
				butt.value_(1);
				this.setValue0(1);
			})
```

```
		);

		controls.add(SCButton(win, Rect(165, 260, 75, 16))
			.states_([["2", Color.blue, Color.black],["2", Color.black, Color.blue]])
			.action_({arg butt;
				stepperOn = false;
				seqOn = false;
				randOn = false;
				currentOnChan = 2;
				synths[0].set(\pulseRate0, 0, \whichModulator, 2);
				butt.value_(1);
				this.setValue0(2);
			})
		);

		controls.add(SCButton(win, Rect(245, 260, 75, 16))
			.states_([["3", Color.blue, Color.black],["3", Color.black, Color.blue]])
			.action_({arg butt;
				stepperOn = false;
				seqOn = false;
				randOn = false;
				currentOnChan = 3;
				synths[0].set(\pulseRate0, 0, \whichModulator, 3);
				butt.value_(1);
				this.setValue0(3);
			})
		);

		controls[0].value = 1;

		this.addDoubleAssignButton(Rect(5, 280, 75, 16), 0, 0);
		this.addDoubleAssignButton(Rect(85, 280, 75, 16), 1, 0);
		this.addDoubleAssignButton(Rect(165, 280, 75, 16), 2, 0);
		this.addDoubleAssignButton(Rect(245, 280, 75, 16), 3, 0);

		controls.add(SCButton(win, Rect(5, 300, 75, 16))
			.states_([["stepper", Color.blue, Color.black],["stepper", Color.black,
Color.blue]])
			.action_({arg butt;
				randOn = false;
				seqOn = false;
				if(stepperOn,{
					stepperOn = false;
					synths[0].set(\pulseRate, 0, \whichModulator, currentOnChan);

					controls[currentOnChan].value_(1);
```

```
                                butt.value_(0);
                        },{
                                stepperOn = true;
                                synths[0].set(\pulseRate, rrand(pulseRate[0], pulseRate[1])*2,
\whichModulator, 4, \whichSelector, 0);
                                butt.value_(1);
                                this.setValue0(4);
                        })
                })
        );

        controls.add(SCButton(win, Rect(85, 300, 75, 16))
                .states_([["rand", Color.blue, Color.black],["rand", Color.black, Color.blue]])
                .action_({arg butt;
                        stepperOn = false;
                        seqOn = false;
                        if(randOn,{
                                randOn = false;
                                synths[0].set(\pulseRate, 0, \whichModulator, currentOnChan);

                                controls[currentOnChan].value_(1);
                                butt.value_(0);
                        },{
                                randOn = true;
                                synths[0].set(\pulseRate, rrand(pulseRate[0], pulseRate[1])*2,
\whichModulator, 4, \whichSelector, 1);
                                butt.value_(1);
                                this.setValue0(5);
                        })
                })
        );

        controls.add(SCButton(win, Rect(165, 300, 75, 16))
                .states_([["seq", Color.blue, Color.black],["seq", Color.black, Color.blue]])
                .action_({arg butt;
                        randOn = false;
                        stepperOn = false;
                        if(seqOn,{
                                seqOn = false;

                                synths[0].set(\pulseRate, 0, \whichModulator, currentOnChan);


                                controls[currentOnChan].value_(1);
                                butt.value_(0);
                        },{
```

```
                                        seqOn = true;
                                        whichSeq = [0,1,2].choose;

                                        controls[currentOnChan].value_(1);

                                        switch(whichSeq,
                                                0, {
                                                        synths[0].set(\pulseRate, rrand(pulseRate[0],
pulseRate[1])*2, \whichModulator, 4, \whichSelector, 2, \whichSeqArray, 0);
                                                        synths[0].set(\seqArray0,
this.makeSeqArray(4).postln);
                                                },
                                                1, {
                                                        synths[0].set(\pulseRate, rrand(pulseRate[0],
pulseRate[1])*2, \whichModulator, 4, \whichSelector, 2, \whichSeqArray, 1);
                                                        synths[0].set(\seqArray1,
this.makeSeqArray(5).postln);
                                                },
                                                2, {
                                                        synths[0].set(\pulseRate, rrand(pulseRate[0],
pulseRate[1])*2, \whichModulator, 4, \whichSelector, 2, \whichSeqArray, 2);
                                                        synths[0].set(\seqArray2,
this.makeSeqArray(7).postln);
                                                }
                                        );
                                        butt.value_(1);
                                        this.setValue0(6);
                                })
                        })
                );

                stepperOn = false;
                randOn = false;
                seqOn = false;

                this.addDoubleAssignButton(Rect(5, 320, 75, 16), 4, 0);
                this.addDoubleAssignButton(Rect(85, 320, 75, 16), 5, 0);
                this.addDoubleAssignButton(Rect(165, 320, 75, 16), 6, 0);

                controls.add(EZRanger(win, Rect(261, 5, 50, 255), "speed", ControlSpec(0.25, 30,
'linear'),
                        {arg val;
                                pulseRate = val.value.postln;

                                synths[0].set(\pulseRate0, rrand(pulseRate[0], pulseRate[1]));
                        }, [4, 7], true, layout:\vert));
```

```
		win.front;
}

makeSeqArray {arg size;
		^this.makeArray(size);
}

makeArray {arg size;
		temp = List.newClear(0);
		size.do{|i| temp.add(i.wrap(0,3))};
		temp = temp.scramble;
		^temp;
}

setValue0 {|notThisOne|
		7.do{arg i;
			if(i!=notThisOne,{
				controls[i].value_(0);
			})
		}
}

addSetup {|setup|
		mixerStrips.postln;
		mixerStrips.do{|item| item.postln; item.addSetup(setup.asSymbol)};

		setups.add(setup);
		mantaData.postln;
		midiData.postln;
		mantaData.do{arg item, i;
			this.setMantaForSetup(setup, item, i);
		};
		midiData.do{arg item, i;
			if(item!=nil,{
				this.setMidiForSetup(setup, item[0], item[1], i);
			});
		};
		setups.postln;
}

setStoredMIDI {
		mixerStrips.do{|item| item.setStoredMIDI};
}

save {arg xmlDoc;
```

```
            xmlSynth = xmlDoc.createElement(modName);
            mantaData.do{arg item, i;
                  xmlSynth.setAttribute("manta"++i.asString, item.asString);
            };
            midiData.do{arg item, i;
                  xmlSynth.setAttribute("midi"++i.asString, item.asString);
            };
            controls.do{arg item, i;
                  xmlSynth.setAttribute("controls"++i.asString, item.value.asString);
            };
            xmlSynth.setAttribute("bounds", win.bounds.asString);

            mixerStrips.do{arg item, i; item.save(xmlDoc, xmlSynth, i)};
            ^xmlSynth;
      }

      load {arg xmlSynth, busMap;
            this.loadMidiData(xmlSynth);
            mantaData.size.do{arg i;
                  midiHidTemp = xmlSynth.getAttribute("manta"++i.asString);
                  if(midiHidTemp!=nil,{
                        midiHidTemp = midiHidTemp.interpret;
                        if(midiHidTemp!=nil,{
                              this.setManta(midiHidTemp[0], midiHidTemp[1], i);
                        });
                  });
            };
            controls.do{arg item, i;
                  midiHidTemp = xmlSynth.getAttribute("controls"++i.asString);
                  if(midiHidTemp!=nil,{
                        controls[i].valueAction_(midiHidTemp.interpret);
                  });
            };
            mixerStrips.do{arg item, i; item.load(xmlSynth, busMap, i)};
            win.bounds_(xmlSynth.getAttribute("bounds").interpret);
            win.front;
      }

      killMeSpecial {
            "kill the mixer";
            localBusses.do{arg item; item.free};
            synthGroup.free;
            mixerGroup.free;
      }

}
```

```
SwitcherMixer_Mod : ModularMixerStrip {
}

DiscreteInput_Mod : ModularMixerStrip {

	init {
		mainProcessingWindow.postln;

		inputBusses = List.new;

		panel = SCCompositeView.new(win, Rect(point.x, point.y, 50, 300));

		mixer = ModularMixer(group, soundInBusses, stereoSoundInBusses);
		mixer.setOutBusses([outBus]);
		mixer.setVol(1);

		ampSpec = ControlSpec(0,1,'amp');

		busAssignSink=MixerBusAssignSink(this, panel, Point(0,0));

		numBusses = outBus.numChannels;

		waitForSet = false;

		ccResponders = (
			setup0: nil,
			setup1: nil,
			setup2: nil,
			setup3: nil
		);
	}

	setInputBusses {arg inputBussesIn;
		"setInputBusses".post;[inputBussesIn, numBusses].postln;
		inputBusses = mixer.setInputBusses(inputBussesIn, numBusses);
	}

	save {arg xmlDoc, xmlSynth, index;
		xmlMixerStrip = xmlDoc.createElement("Mixer"++index.asString);
		xmlMixerStrip.setAttribute("numBusses", numBusses.asString);
		inputBusses.do{arg item,i;
			xmlMixerStrip.setAttribute("inBus"++i.asString, item.asString);
		};
		xmlMixerStrip.setAttribute("outBus", channelOutMenu.value.asString);
		xmlSynth.appendChild(xmlMixerStrip);
```

```
	}

	load {arg xmlMainMixer, busMap, index;
		xmlMixerStrip = xmlMainMixer.getElement("Mixer"++index.asString);



		numBusses = xmlMixerStrip.getAttribute("numBusses");
		if(numBusses!=nil,{
			numBusses = xmlMixerStrip.getAttribute("numBusses").interpret;
		},
		{
			numBusses = 2;
		});

		counter = 0;

		inBusTemp = xmlMixerStrip.getAttribute("inBus"++counter.asString);
		inBusTempList = List.new;
		while({inBusTemp!=nil},{
			counter.postln;
			inBusTempList.add(busMap[inBusTemp.interpret.asSymbol]);
			counter = counter + 1;
			inBusTemp = xmlMixerStrip.getAttribute("inBus"++counter.asString).postln;
		});

		inBusTempList.do{arg item;
			if(soundInBusses.indexOf(item)!=nil,
			{
				busAssignSink.assignBus(item,
"S"+soundInBusses.indexOf(item).asString);
			},
				if(stereoSoundInBusses.indexOf(item)!=nil,
				{
					busAssignSink.assignBus(item, "S"+
+soundInBusses.indexOf(item).asString++(soundInBusses.indexOf(item)+1.asString));
				},{
					busAssignSink.assignBus(item, item.asString);
				})
			)
		};
	}
}

ModularBusCenter {
	classvar <>monoRouterBusses;
```

```
	*initClass {
		monoRouterBusses = List.newClear(0);
	}

	*reset {
		monoRouterBusses = List.newClear(0);
	}

	*addMonoRouterBus {arg busNum;
		monoRouterBusses.add(busNum);
	}

	*removeMonoRouterBus {arg busNum;
		monoRouterBusses.remove(busNum);
	}

	*getBusses {
		^monoRouterBusses
	}

	*monoRouterMatch {arg num;
		if(monoRouterBusses.indexOf(num)!=nil,{
			^true;
		},{^false});
	}


}

MIDIHIDControl {
	var midiRequestObject, setNextMidiNum, instantMidiRequestObject, instantNextMidiNum,
instantCCResponder, instantValueResponder, instantSliderResponder, instantNoteOnResponder,
instantNoteOnMantaResponder;

	*new {
		^super.newCopyArgs().init;
	}

	init {
	}

	requestMidiAssignInfo {arg object, num;
		midiRequestObject = object;
		setNextMidiNum = num;
	}
```

```
	requestInstantMidiAssignInfo {arg object, controlNum;
		[object, controlNum].postln;

		instantMidiRequestObject = object;
		instantNextMidiNum = controlNum.postln;

		instantCCResponder = CCResponder2({|src,chan,num,val|
			instantMidiRequestObject.setControlWNext(1,1,controlNum,[[chan,num],0]);
		},nil,nil,nil,nil,nil);

		instantNoteOnResponder = NoteOnResponder2({|src,chan,num,val|
			instantMidiRequestObject.setControlWNext(1,0,controlNum,[[chan,num],1]);
		},nil,nil,nil,nil,nil);

		instantValueResponder = OSCresponderNode(nil, '/manta/value', {|t, r, msg|
instantMidiRequestObject.setControlWNext(0,1,controlNum,[msg[1],1])}).add;
		instantSliderResponder = OSCresponderNode(nil, '/manta/slider', {|t, r, msg|
instantMidiRequestObject.setControlWNext(0,1,controlNum,[msg[1],2])}).add;
		instantNoteOnMantaResponder = OSCresponderNode(nil, '/manta/noteOn', {|t, r, msg|
instantMidiRequestObject.setControlWNext(0,0,controlNum,[msg[1],0])}).add;
	}

	clearInstantMidiAssignInfo {
		instantMidiRequestObject = nil;
		instantNextMidiNum = nil;
		instantCCResponder.remove;
		instantNoteOnResponder.remove;
		instantValueResponder.remove;
		instantSliderResponder.remove;
		instantNoteOnMantaResponder.remove;
	}

	sendMidiAssignInfo {arg data, dataType;
		[midiRequestObject, data].postln;
		if(midiRequestObject!=nil,{
			midiRequestObject.setMidi(data, dataType, setNextMidiNum);
			midiRequestObject = nil;
		});
	}

	sendMantaAssignInfo {arg buttonNumber, buttonType;
		[midiRequestObject, buttonNumber, buttonType].postln;
		if(midiRequestObject!=nil,{
			midiRequestObject.setManta(buttonNumber, buttonType, setNextMidiNum);
			midiRequestObject = nil;
```

```
		});
	}
}

DoubleAssignButton {
	var win, rect, layout, regButton, instantButton, <>regAction, <>instantAction, <bounds;

	*new {arg win, rect, layout;
		^super.newCopyArgs(win, rect, layout).init;
	}

	init {
		bounds = rect;
		//regAction = {};
		instantAction = {};
		if(layout.asSymbol==\vert,
		{
			instantButton = SCButton.new(win,Rect(rect.left, rect.top+(rect.height),
rect.width, rect.height))
				.states_([ [ "AI", Color.red, Color.black ] ,[ "C", Color.black,
Color.red ] ])
				.action_{|v|
					instantAction.(v);
				};
		},{
			instantButton = SCButton.new(win,Rect(rect.left, rect.top, rect.width,
rect.height))
				.states_([ [ "AI", Color.red, Color.black ] ,[ "C", Color.black,
Color.red ] ])
				.action_{|v|
					instantAction.(v);
				};
		})
	}

	bounds_ {arg rect;
		instantButton.bounds_(Rect(rect.left+(rect.width/2), rect.top, rect.width/2, rect.height));
	}

	setInstBut {arg val; instantButton.value = val}
}

ModularClassList {
	var <>whichArray, <>classArray, classDictionary;

	*new {arg whichArray;
```

```
		^super.new.whichArray_(whichArray).init;
	}

	init {
		("whichArray"+whichArray).postln;
		switch(whichArray,
			'normal', {
				classArray = ["GlassSines", "RingMod", "FilterDelays",
"PulsatingDelays", "BitCrusher", "TriggerDelays", "OverLapSamples", "LoopBuf", "Combulation",
"BuchlaFilters", "BuchlaModelSolo", "ReverbDrone", "ShifterFeedback", "Compander",
"MouseDistortion", "CycleGripper", "Mixer", "Freeze", "AmpMod", "SignalSwitcher",
"SignalSwitcher4", "LoopMachine", "LoopBuf2", "GrainAge", "GingerMan", "SwoopDown", "EQ",
"DistortMono", "GrainFreezeNoise", "TFreeze", "PulseBP", "DownShift", "InterruptDistortion",
"StraightLoop", "Sand", "GrabNLoop", "FloatShifter", "HarmDoublerUp", "GrainFreezeDrums",
"StarDust", "AmpFollower", "StraightDelays", "EightDelays2", "SpecMul", "Melter", "GVerb",
"HarmonicDoubler2", "Cutter", "LongDelay", "Filters", "ShifterX2", "Record", "RingModStereo",
"InterruptSine", "MiniGripper", "BitInterrupter", "SampleBank", "DecimateGrains", "InterruptDelay",
"DelayFeedback", "TriggerSines", "DelayRingMod", "SnareSine", "CutterB", "EQmini", "SpecDelay",
"LowSines", "EnvGen", "DrumBombs", "GrainInterrupt", "SampleMashup", "FilterGrains",
"FilterGrainsB", "ScaleShifterB","UpDownSines","SinArray", "SweepingNoise", "SpaceJunk",
"MonoRouter"].sort;
			},
			'wubbels', {
				classArray = ["WubbelsSine", "WubbelsSine2", "WubbelsSine3",
"FilterDelays", "AutoTune", "RhythmicDelays", "Mixer", "EQ", "Compander", "LoopBuf"].sort;
			},'neuwirth', {
				classArray = ["NeuwirthSine", "RingModStereo", "Mixer", "EQ",
"Compander"].sort;
			},
			'atdV', {
				classArray = ["GlassSines", "RingMod", "FilterDelays",
"PulsatingDelays", "BitCrusher", "TriggerDelays", "OverLapSamples", "LoopBuf", "Combulation",
"BuchlaFilters", "BuchlaModelSolo", "ReverbDrone", "ShifterFeedback", "Compander",
"MantaBuffers", "MouseDistortion", "CycleGripper", "Mixer", "Freeze", "AmpMod",
"SignalSwitcher", "SignalSwitcher4", "LoopMachine", "LoopBuf2", "GrainAge", "GingerMan",
"SwoopDown", "EQ", "DistortMono", "GrainFreezeNoise", "TFreeze", "PulseBP", "DownShift",
"InterruptDistortion", "StraightLoop", "Sand", "GrabNLoop", "AtdV", "FloatShifter",
"HarmDoublerUp", "GrainFreezeDrums", "StarDust", "AmpFollower", "StraightDelays",
"EightDelays2", "SpecMul", "Melter", "GVerb", "HarmonicDoubler2", "Cutter", "LongDelay",
"Filters", "ShifterX2", "Record", "RingModStereo", "InterruptSine", "MiniGripper", "BitInterrupter",
"SampleBank", "DecimateGrains"].sort;
			}
		);

		classDictionary = IdentityDictionary.new;
		classDictionary.add('WubbelsSine'->{arg synthGroup, bus, midiHidControl, manta,
```

```
bcf2000, setups; WubbelsSine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('WubbelsSine2'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; WubbelsSine2_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GlassSines'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GlassSines_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('RingMod'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; RingMod_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('FilterDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; FilterDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('PulsatingDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; PulsatingDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('BitCrusher'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; BitCrusher_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('TriggerDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; TriggerDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('OverLapSamples'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; OverLapSamples_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LoopBuf'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; LoopBuf_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Combulation'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; Combulation_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('BuchlaModelSolo'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; BuchlaModelSolo_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('BuchlaFilters'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; BuchlaFilters_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ReverbDrone'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ReverbDrone_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ShifterFeedback'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ShifterFeedback_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('EightDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; EightDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('EightDelays2'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; EightDelays2_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Compander'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; Compander_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('MantaBuffers'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; MantaBuffers_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('MouseDistortion'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; MouseDistortion_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('CycleGripper'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; CycleGripper_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Freeze'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Freeze_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('TFreeze'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; TFreeze_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('AmpMod'->{arg synthGroup, bus, midiHidControl, manta,
```

```
bcf2000, setups; AmpMod_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LoopVidBuf'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; LoopVidBuf_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LoopMachine'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; LoopMachine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LoopBuf2'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; LoopBuf2_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GrainAge'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GrainAge_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GingerMan'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GingerMan_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SwoopDown'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SwoopDown_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('EQ'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; EQ_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DistortMono'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DistortMono_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GrainFreezeNoise'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GrainFreezeNoise_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SignalSwitcher'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SignalSwitcher_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SignalSwitcher4'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SignalSwitcher4_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('PulseBP'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; PulseBP_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DownShift'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DownShift_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('StraightLoop'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; StraightLoop_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Sand'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Sand_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GrabNLoop'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GrabNLoop_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('AtdV'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; AtdV_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('InterruptDistortion'->{arg synthGroup, bus, midiHidControl,
manta, bcf2000, setups; InterruptDistortion_Mod(synthGroup, bus, midiHidControl, manta, bcf2000,
setups)});
		classDictionary.add('FloatShifter'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; FloatShifter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('HarmDoublerUp'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; HarmDoublerUp_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GrainFreezeDrums'->{arg synthGroup, bus, midiHidControl,
manta, bcf2000, setups; GrainFreezeDrums_Mod(synthGroup, bus, midiHidControl, manta, bcf2000,
setups)});
		classDictionary.add('StarDust'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
```

```
setups; StarDust_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('AmpFollower'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; AmpFollower_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('StraightDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; StraightDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SpecMul'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; SpecMul_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Melter'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Melter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GVerb'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; GVerb_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('HarmonicDoubler2'->{arg synthGroup, bus, midiHidControl,
manta, bcf2000, setups; HarmonicDoubler2_Mod(synthGroup, bus, midiHidControl, manta, bcf2000,
setups)});
		classDictionary.add('Cutter'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Cutter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LongDelay'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; LongDelay_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('AmpInterrupter'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; AmpInterrupter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Filters'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Filters_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ShifterX2'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ShifterX2_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('Record'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; Record_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('RingModStereo'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; RingModStereo_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('InterruptSine'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; InterruptSine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('MiniGripper'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; MiniGripper_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('BitInterrupter'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; BitInterrupter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SampleBank'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SampleBank_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DecimateGrains'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DecimateGrains_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('InterruptDelay'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; InterruptDelay_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DelayFeedback'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DelayFeedback_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('AutoTune'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; AutoTune_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('RhythmicDelays'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; RhythmicDelays_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
```

```
		classDictionary.add('WubbelsSine3'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; WubbelsSine3_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('TriggerSines'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; TriggerSines_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DelayRingMod'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DelayRingMod_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SnareSine'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SnareSine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('CutterB'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; CutterB_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('NeuwirthSine'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; NeuwirthSine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('EQmini'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; EQmini_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SpecDelay'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SpecDelay_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('LowSines'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; LowSines_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('EnvGen'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; EnvGen_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('DrumBombs'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; DrumBombs_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('GrainInterrupt'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; GrainInterrupt_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});

		classDictionary.add('SampleMashup'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SampleMashup_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('FilterGrains'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; FilterGrains_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('FilterGrainsB'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; FilterGrainsB_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ScaleShifter'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ScaleShifter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ScaleShifterB'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ScaleShifterB_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ScaleShifterC'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ScaleShifterC_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('UpDownSines'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; UpDownSines_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('ReverseMachine'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; ReverseMachine_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SinArray'->{arg synthGroup, bus, midiHidControl, manta, bcf2000,
setups; SinArray_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SweepingNoise'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; SweepingNoise_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('SpaceJunk'->{arg synthGroup, bus, midiHidControl, manta,
```

```
bcf2000, setups; SpaceJunk_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		classDictionary.add('MonoRouter'->{arg synthGroup, bus, midiHidControl, manta,
bcf2000, setups; MonoRouter_Mod(synthGroup, bus, midiHidControl, manta, bcf2000, setups)});
		}

	initModule {arg className, synthGroup, bus, midiHidControl, manta, bcf2000, setups;
		"initModule".postln;
		^classDictionary[className.asSymbol].value(synthGroup, bus, midiHidControl, manta,
bcf2000, setups);
	}

	initMixer {arg group, soundInBusses, stereoSoundInBusses, outBus, name, location,
numMixers, midiHidControl, manta, bcf2000, setups, mainProcessingWindow;
		"initMixer".postln;
		location.postln;
		^ModularMainMixer(group, soundInBusses, stereoSoundInBusses, outBus.index, name,
location, numMixers, midiHidControl, manta, bcf2000, setups, mainProcessingWindow)
	}
	checkSynthName {arg name;
		if(classArray.indexOfEqual(name)!=nil,{^true},{^false})
	}
}

ModularSetInputWindow {
	var <>availableBusIns, <>requestingObject, inputBusses, window, applyButton,
soundInButtons, busInButtons, returnList, index;

	*new {arg availableBusIns, requestingObject, inputBusses;
		^super.newCopyArgs(availableBusIns, requestingObject, inputBusses).init;
	}

	init {
		returnList = List.new;
		2.do{
			returnList.add(List.new);
		};
		window = SCWindow.new("Assign Inputs",Rect(654, 690, 5+(35*8), 60+
((availableBusIns[1].size/5).ceil*35)));
		window.alwaysOnTop_(true);

		soundInButtons = List.new;
		busInButtons = List.new;

		availableBusIns[0].do{arg item, i;
			soundInButtons.add(SCButton.new(window,Rect(5+(35*i), 5, 30, 20))
				.states_([ [ "S"++i, Color.blue, Color.black ], [ "S"++i, Color.black,
```

```
Color.blue ] ])
                        .action_{|v|
                                if(v.value==1,{
                                        returnList[0].add(i);
                                },{
                                        returnList[0].remove(i);
                                });
                                returnList.postln;
                        });
                };

                availableBusIns[1].do{arg item, i;
                        busInButtons.add(SCButton.new(window,Rect(5+((i%5)*35), 30+
((i/5).floor*25), 30, 20))
                                .states_([ [ item.asString, Color.blue, Color.black ], [ item.asString,
Color.black, Color.blue ] ])
                                .action_{|v|
                                        if(v.value==1,{
                                                returnList[1].add(item);
                                        },{
                                                returnList[1].remove(item);
                                        });
                                        returnList.postln;
                                });
                };

                applyButton = SCButton.new(window, Rect(5, (availableBusIns[1].size/5).ceil*25+30,
50, 30))
                        .states_([ [ "Apply", Color.blue, Color.black ]])
                                .action_{|v|
                                        requestingObject.setInputBusses(returnList);
                                        window.close;
                                };


                inputBusses[0].do{arg i; soundInButtons[i].valueAction = 1};
                inputBusses[1].do{arg i;
                        index = availableBusIns[1].indexOfEqual(i);
                        if(index!=nil,{
                                busInButtons[index].valueAction = 1;
                        })
                };

                window.front;
        }
}
```

```
ModularMixerInfoObject {
	var <>numInChannels, <>numOutChannels, <>inBus, <>outBus, <>volBus, <>muteBus;

	*new {arg numInChannels, numOutChannels, inBus, outBus, volBus, muteBus;
		^super.newCopyArgs(numInChannels, numOutChannels, inBus, outBus, volBus,
muteBus);
	}
}

ModularMixer {
	var <>group, soundInBusses, stereoSoundInBusses, <>mixers, modularMixerInfoObjects,
index, <>volBus, <>muteBus, <>panBus, singleMixer, <>inputBusses, outBusses, outInNums,
outBussesMonoStereo, outBusTemp;

	*new {arg group, soundInBusses, stereoSoundInBusses;
		^super.newCopyArgs(group, soundInBusses, stereoSoundInBusses).init;
	}

	*initClass {
		{
			SynthDef("modularMixer1-1", {arg inBus, outBus, volBus, muteBus, gate=1;
				var env, vol, muteVol;

				vol = In.kr(volBus);
				muteVol = Lag.kr(In.kr(muteBus), 0.01);

				env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

				Out.ar(outBus, In.ar(inBus)*env*vol*muteVol);
			}).writeDefFile;
			SynthDef("modularMixer1-2", {arg inBus, outBus, volBus, muteBus, panBus,
gate=1;
				var env, vol, muteVol, in;

				vol = In.kr(volBus);
				muteVol = Lag.kr(In.kr(muteBus), 0.01);

				env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
				in = In.ar(inBus);

				Out.ar(outBus, Pan2.ar(in, In.kr(panBus))*env*vol*muteVol);
			}).writeDefFile;
			SynthDef("modularMixer1-3", {arg inBus, outBus, volBus, muteBus, gate=1;
				var env, vol, muteVol, in;
```

```
        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
        in = In.ar(inBus);

        Out.ar(outBus, [in,in,in,]*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer1-4", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol, in;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
        in = In.ar(inBus);

        Out.ar(outBus, [in,in,in,in]*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer1-5", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol, in;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
        in = In.ar(inBus);

        Out.ar(outBus, [in,in,in,in,in]*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer1-6", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol, in;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
        in = In.ar(inBus);

        Out.ar(outBus, [in,in,in,in,in,in]*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer1-7", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol, in;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);
```

```
                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
                in = In.ar(inBus);

                Out.ar(outBus, [in,in,in,in,in,in,in,in]*env*vol*muteVol);
        }).writeDefFile;
        SynthDef("modularMixer1-8", {arg inBus, outBus, volBus, muteBus, gate=1;
                var env, vol, muteVol, in;

                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
                in = In.ar(inBus);

                Out.ar(outBus, [in,in,in,in,in,in,in,in]*env*vol*muteVol);
        }).writeDefFile;


        SynthDef("modularMixer2-1", {arg inBus, outBus, volBus, muteBus, gate=1;
                var env, vol, muteVol;

                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

                Out.ar(outBus, Mix.ar(In.ar(inBus, 2))*env*vol*muteVol);
        }).writeDefFile;
        SynthDef("modularMixer2-2", {arg inBus, outBus, volBus, muteBus, panBus,
gate=1;
                var env, vol, muteVol, in;

                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

                in = In.ar(inBus, 2);

                Out.ar(outBus,
Balance2.ar(in[0],in[1],In.kr(panBus))*env*vol*muteVol);
        }).writeDefFile;
        SynthDef("modularMixer3-3", {arg inBus, outBus, volBus, muteBus, gate=1;
                var env, vol, muteVol;
```

```
        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

        Out.ar(outBus, In.ar(inBus, 3)*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer4-4", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

        Out.ar(outBus, In.ar(inBus, 4)*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer5-5", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

        Out.ar(outBus, In.ar(inBus, 5)*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer6-6", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

        Out.ar(outBus, In.ar(inBus, 6)*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer7-7", {arg inBus, outBus, volBus, muteBus, gate=1;
        var env, vol, muteVol;

        vol = In.kr(volBus);
        muteVol = Lag.kr(In.kr(muteBus), 0.01);

        env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

        Out.ar(outBus, In.ar(inBus, 7)*env*vol*muteVol);
```

```
}).writeDefFile;
SynthDef("modularMixer8-8", {arg inBus, outBus, volBus, muteBus, gate=1;
	var env, vol, muteVol;

	vol = In.kr(volBus);
	muteVol = Lag.kr(In.kr(muteBus), 0.01);

	env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

	Out.ar(outBus, In.ar(inBus, 8)*env*vol*muteVol);
}).writeDefFile;

//N-1 mixers

SynthDef("modularMixer3-1", {arg inBus, outBus, volBus, muteBus, gate=1;
	var env, vol, muteVol;

	vol = In.kr(volBus);
	muteVol = Lag.kr(In.kr(muteBus), 0.01);

	env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

	Out.ar(outBus, Mix.ar(In.ar(inBus, 3))*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer4-1", {arg inBus, outBus, volBus, muteBus, gate=1;
	var env, vol, muteVol;

	vol = In.kr(volBus);
	muteVol = Lag.kr(In.kr(muteBus), 0.01);

	env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

	Out.ar(outBus, Mix.ar(In.ar(inBus, 4))*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer5-1", {arg inBus, outBus, volBus, muteBus, gate=1;
	var env, vol, muteVol;

	vol = In.kr(volBus);
	muteVol = Lag.kr(In.kr(muteBus), 0.01);

	env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

	Out.ar(outBus, Mix.ar(In.ar(inBus, 5))*env*vol*muteVol);
}).writeDefFile;
SynthDef("modularMixer6-1", {arg inBus, outBus, volBus, muteBus, gate=1;
	var env, vol, muteVol;
```

```
                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

                Out.ar(outBus, Mix.ar(In.ar(inBus, 6))*env*vol*muteVol);
            }).writeDefFile;
            SynthDef("modularMixer7-1", {arg inBus, outBus, volBus, muteBus, gate=1;
                var env, vol, muteVol;

                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

                Out.ar(outBus, Mix.ar(In.ar(inBus, 7))*env*vol*muteVol);
            }).writeDefFile;
            SynthDef("modularMixer8-1", {arg inBus, outBus, volBus, muteBus, gate=1;
                var env, vol, muteVol;

                vol = In.kr(volBus);
                muteVol = Lag.kr(In.kr(muteBus), 0.01);

                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);

                Out.ar(outBus, Mix.ar(In.ar(inBus, 8))*env*vol*muteVol);
            }).writeDefFile;
        }.defer(1);
}

init {
        volBus = Bus.control;
        volBus.set(0);
        muteBus = Bus.control;
        muteBus.set(1);
        panBus = Bus.control;
        panBus.set(0);

        inputBusses = List.new;

        mixers = IdentityDictionary.new;
        modularMixerInfoObjects = IdentityDictionary.new;
}

//outIn is a number here, signifying the number of channels that the mixer outputs
```

```
	setMainMixerOutBusses {arg outIn;
		if(outIn.size>0,{
			outBussesMonoStereo = Pseq(#[2], inf).asStream;
			outBusses = Pseq(outIn, inf).asStream;
		})
	}

	//outIn is a Bus here, signifying the bus that is being used as a transfer bus
	setOutBusses {arg outIn;
		if(outIn.size>0,{
			outInNums = List.new;
			outIn.do{arg item, i; outInNums.add(item.index)};
			outBussesMonoStereo = List.new;
			outIn.do{arg item, i; outBussesMonoStereo.add(item.numChannels)};
			outInNums.postln;
			outBussesMonoStereo.postln;
			outBussesMonoStereo = Pseq(outBussesMonoStereo, inf).asStream;
			outBusses = Pseq(outInNums, inf).asStream;
		})
	}

	add11InputMixer {arg inputBusIndex, outBusIndex;
		outBusIndex.postln;
		singleMixer = mixers[inputBusIndex.asSymbol];
		if(singleMixer==nil,{
			outBusTemp = outBusses.next;
			mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-1", [\inBus,
inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group));
			modularMixerInfoObjects.add(inputBusIndex.asSymbol ->
ModularMixerInfoObject(1, 1, inputBusIndex, outBusTemp, volBus, muteBus));
		});
	}

	add12InputMixer {arg inputBusIndex, outBusIndex;
		outBusIndex.postln;
		singleMixer = mixers[inputBusIndex.asSymbol];
		if(singleMixer==nil,{
			outBusTemp = outBusses.next;
			mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-2", [\inBus,
inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index, \panBus,
panBus.index], group));
			modularMixerInfoObjects.add(inputBusIndex.asSymbol ->
ModularMixerInfoObject(1, 2, inputBusIndex, outBusTemp, volBus, muteBus));
		});
	}
```

```
	add21InputMixer {arg inputBusIndex, outBusIndex;
		singleMixer = mixers[inputBusIndex.asSymbol];
		if(singleMixer==nil,{
			outBusTemp = outBusses.next;
			mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer2-1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group));
			modularMixerInfoObjects.add(inputBusIndex.asSymbol -> ModularMixerInfoObject(2, 1, inputBusIndex, outBusTemp, volBus, muteBus));
		});
	}

	add22InputMixer {arg inputBusIndex, outBusIndex;
		outBusIndex.postln;
		singleMixer = mixers[inputBusIndex.asSymbol];
		if(singleMixer==nil,{
			outBusTemp = outBusses.next;
			mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer2-2", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index, \panBus, panBus.index], group));
			modularMixerInfoObjects.add(inputBusIndex.asSymbol -> ModularMixerInfoObject(2, 2, inputBusIndex, outBusTemp, volBus, muteBus));
		});
	}

	addNNInputMixer {arg inputBusIndex, numBusses;
		"addNNInputMixer".postln;
		singleMixer = mixers[inputBusIndex.asSymbol];
		if(singleMixer==nil,{
			outBusTemp = outBusses.next;
			switch(numBusses,
				2, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer2-2", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index, \panBus, panBus.index], group))},
				3, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer3-3", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group))},
				4, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer4-4", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group))},
				5, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer5-5", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group))},
				6, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer6-6", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index], group))},
				7, {	mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer7-
```

```
7", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                8, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer8-
8", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))}
                        );
                        modularMixerInfoObjects.add(inputBusIndex.asSymbol ->
ModularMixerInfoObject(numBusses, numBusses, inputBusIndex, outBusTemp, volBus, muteBus));
                });
        }

        add1NInputMixer {arg inputBusIndex, numBusses;
                "add1NInputMixer".postln;
                singleMixer = mixers[inputBusIndex.asSymbol];
                if(singleMixer==nil,{
                        outBusTemp = outBusses.next;
                        switch(numBusses,
                                2, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
2", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index,
\panBus, panBus.index], group))},
                                3, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
3", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                4, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
4", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                5, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
5", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                6, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
6", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                7, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
7", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                8, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer1-
8", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))}
                        );
                        modularMixerInfoObjects.add(inputBusIndex.asSymbol ->
ModularMixerInfoObject(1, numBusses, inputBusIndex, outBusTemp, volBus, muteBus));
                });
        }

        addN1InputMixer {arg inputBusIndex, numBusses;
                "addN1InputMixer".postln;
```

```
                singleMixer = mixers[inputBusIndex.asSymbol];
                if(singleMixer==nil,{
                        outBusTemp = outBusses.next;
                        switch(numBusses,
                                2, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer2-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index,
\panBus, panBus.index], group))},
                                3, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer3-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                4, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer4-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                5, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer5-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                6, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer6-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                7, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer7-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))},
                                8, {    mixers.add(inputBusIndex.asSymbol -> Synth("modularMixer8-
1", [\inBus, inputBusIndex, \outBus, outBusTemp, \volBus, volBus.index, \muteBus, muteBus.index],
group))}
                        );
                        modularMixerInfoObjects.add(inputBusIndex.asSymbol ->
ModularMixerInfoObject(numBusses, 1, inputBusIndex, outBusTemp, volBus, muteBus));
                });
        }

        setInputBusses {arg inputBussesIn, numBusses;
                "numBusses ".post; numBusses.postln;
                inputBusses.postln;
                "bussesIn".post;inputBussesIn.postln;

                //remove the extra mixers
                inputBusses.do{arg item, i;
                        item.postln;
                        index = inputBussesIn.indexOfEqual(item);
                        if(index==nil,{
                                this.removeMixer(item.postln);
                        });
                };

                //add the new mixers
```

```
		inputBussesIn.do{arg item;
			this.addABus(item, numBusses);
		};

		^inputBusses = inputBussesIn.deepCopy;
	}

	addABus {arg item, numBusses;
		if((soundInBusses.indexOf(item)!=nil)||(stereoSoundInBusses.indexOf(item)!=nil),{
			if((soundInBusses.indexOf(item)!=nil),{
				("settingSoundInBus"+item).postln;
				if(outBussesMonoStereo.next.postln == 1,{
					this.add11InputMixer(item);
				},{
					this.add1NInputMixer(item, numBusses);
				})
			},{
				"stereo input bus".postln;
				if(outBussesMonoStereo.next.postln == 1,{
					this.add21InputMixer(item);
				},{
					this.add22InputMixer(item);
				})
			})
		},{
			ModularBusCenter.getBusses.postln;
			if(ModularBusCenter.monoRouterMatch(item),{

				if(outBussesMonoStereo.next.postln == 1,{
					"add11Mixer".postln;
					this.add11InputMixer(item);
				},{
					"add1NMixer".postln;
					this.add1NInputMixer(item, numBusses);
				})
			},{
				("settingNormalBus"+item).postln;
				//this is for the cockimamey scheme i was using to deal with the ring
modulator
				if(outBussesMonoStereo.next.postln == 1,{
					//force 8 channel input mixed down to 1 channel - hopefully this
doesn't add too much noise to the signal
					"8-1?".postln;
					this.addN1InputMixer(item, 8);
				},{
```

```
                        this.addNNInputMixer(item, 8);
                    })
                })
            });
}

resetNumBusses {arg numBusses;
        "resetNumBusses".postln;
        mixers.do{arg item, i;
            item.set(\gate, 0);
        };
        mixers = IdentityDictionary.new;
        modularMixerInfoObjects.do{arg item;
            this.addABus(item.inBus, numBusses);
        };
        modularMixerInfoObjects.do{arg item;
            item.numInChannels.postln; item.numOutChannels.postln;
        };
}

removeMixer{arg index;
        "removing".postln;
        mixers[index.asSymbol].set(\gate, 0);
        mixers.removeAt(index.asSymbol);
}

removeAllMixers{arg index;
        mixers.do{arg item;
            "removing".postln;
            item.set(\gate, 0);
        };
        mixers = IdentityDictionary.new;
        inputBusses = List.new;
}

mute {
        muteBus.set(0);
}

unmute {
        muteBus.set(1);
}

setVol {arg val;
        volBus.set(val);
}
```

```
	setPan {arg val;
		panBus.set(val);
	}
}

MIDIHIDObject {
	var <>midiHidControl, <>manta, <>bcf2000, <>setups, waitForSet, waitForType,
assignButtons, win, midiHidTemp, midiData, mantaData, noteOnResponders, ccResponders, controls;

	initControlLists {arg num;
		mantaData = List.newClear(num);
		midiData = List.newClear(num);
		ccResponders = (
			setup0: List.newClear(num),
			setup1: List.newClear(num),
			setup2: List.newClear(num),
			setup3: List.newClear(num)
		);
		noteOnResponders = (
			setup0: List.newClear(num),
			setup1: List.newClear(num),
			setup2: List.newClear(num),
			setup3: List.newClear(num)
		);
	}

	setMidi {arg data, dataType, controlsIndex;
		if(controlsIndex<midiData.size,{
			setups.do{arg setup;
				this.setMidiForSetup(setup, data, dataType, controlsIndex)
			}
		},{
			this.setSpecialMidi(data, dataType, controlsIndex);
		});
	}

	setMidiForSetup {|setup, data, dataType, controlsIndex|
		if(controlsIndex<midiData.size,{
			switch(dataType,
				0,{
					midiData.put(controlsIndex, [data,0]);   //0 means control data
					ccResponders[setup.asSymbol].put(controlsIndex,
						CCResponder2({ |src,chan,num,val|
```

```
{controls[controlsIndex].valueAction_(controls[controlsIndex].controlSpec.map(val/127))}.defer;
                                        },
                                        nil, // any source
                                        data[0], // desired channel
                                        data[1], // desired number
                                        nil, // any value
                                        setup.asSymbol  //signifies the setup of the
responder
                                    )
                                )
                            },
                            1,{
                                midiData.put(controlsIndex, [data,1]);  //1 means noteOn data
                                noteOnResponders[setup.asSymbol].put(controlsIndex,
                                    NoteOnResponder2({ |src,chan,note,vel|

        {controls[controlsIndex].valueAction_(controls[controlsIndex].value+1)}.defer;
                                        },
                                        nil, // any source
                                        nil, // desired channel
                                        data, // desired note
                                        nil, // any value
                                        setup.asSymbol  //signifies the setup of the
responder
                                    )
                                )
                            },
                            2,{
                                midiData.put(controlsIndex, [data,2]);   //2 onOff control data
                                ccResponders[setup.asSymbol].put(controlsIndex,
                                    CCResponder2({ |src,chan,num,val|

        {controls[controlsIndex].valueAction_(controls[controlsIndex].value+1)}.defer;
                                        },
                                        nil, // any source
                                        data[0], // desired channel
                                        data[1], // desired number
                                        nil, // any value
                                        setup.asSymbol  //signifies the setup of the
responder
                                    )
                                )
                            }
                        )
                    },{
                        this.setSpecialMidi(setup, data, dataType, controlsIndex);
```

```
        });
}

removeSetup {arg setup;
        setups.remove(setup);
        mantaData.do{arg item;
                if(item!=nil,{
                        switch(item[1],
                                0,{
                                        manta.removeNoteOnSetup(setup.asSymbol, item[0]);
                                },
                                1,{
                                        manta.removePadSetup(setup.asSymbol, item[0]);
                                },
                                2,{
                                        manta.removeSliderSetup(setup.asSymbol, item[0]);
                                }
                        );
                });
        };
        ccResponders.do{arg setup;
                setup.do{arg item;
                        if(item!=nil,{
                                item.remove;
                        });
                }
        };
        noteOnResponders.do{arg setup;
                setup.do{arg item;
                        if(item!=nil,{
                                item.remove;
                        });
                }
        };
}

addSetup {arg setup;
        setups.add(setup);
        mantaData.do{arg item, i;
                this.setMantaForSetup(setup, item, i);
        };
        midiData.do{arg item, i;
                if(item!=nil,{
                        this.setMidiForSetup(setup, item[0], item[1], i);
                });
        };
```

```
	}

	setMantaForSetup {arg setup, item, i;
		if(item!=nil,{
			switch(item[1],
				0,{
					manta.addNoteOnSetup(setup.asSymbol, item[0], {|val|
{controls[i].valueAction_(controls[i].value+1)}.defer});
				},
				1,{
					manta.addPadSetup(setup.asSymbol, item[0], {|val|
{controls[i].valueAction_(controls[i].controlSpec.map(val/180))}.defer});
				},
				2,{
					manta.addSliderSetup(setup.asSymbol, item[0], {|val|
{controls[i].valueAction_(controls[i].controlSpec.map(val/4096))}.defer});
				},
				3,{
					manta.addNoteOnSetup(setup.asSymbol, item[0], {|val|
{controls[i].valueAction_(1)}.defer});
					manta.addNoteOffSetup(setup.asSymbol, item[0], {|val|
{controls[i].valueAction_(0)}.defer});
				}
			);
		});
	}

	setStoredMIDI {
		if(bcf2000!=nil,{
			midiData.do{arg data, i;
				if(data!=nil,{
					if(data[1]==0,{
						bcf2000.control(data[0][0],data[0][1],
(controls[i].controlSpec.unmap(controls[i].value)*127).ceil);
					})
				})
			}
		});
	}

	setManta {arg buttonNum, buttonType, controlsIndex;
		if(controlsIndex<mantaData.size,{
			if(manta!=nil,{
				mantaData.put(controlsIndex, [buttonNum.deepCopy,
buttonType.deepCopy]);
				setups.do{arg setup;
```

```
                                        this.setMantaForSetup(setup, mantaData[controlsIndex],
controlsIndex)
                                }
                        })
                },{
                        this.setSpecialManta(buttonNum, buttonType, controlsIndex);
                });
        }

        clearMidiHid {arg num;
                ccResponders.do{arg setup;
                        if(setup[num]!=nil,{
                                setup[num].remove;
                                setup.put(num, nil);
                        });
                };
                noteOnResponders.do{arg setup;
                        if(setup[num]!=nil,{
                                setup[num].remove;
                                setup.put(num, nil);
                        });
                };
                if(midiData[num]!=nil,{
                        midiData.put(num, nil);
                });
                if(manta!=nil,{
                        if(mantaData[num]!=nil,{
                                switch(mantaData[num][1],
                                        0,{
                                                setups.do{arg setup;
manta.removeNoteOnSetup(setup.asSymbol, mantaData[num][0])};
                                        },
                                        1,{
                                                setups.do{arg setup;
manta.removePadSetup(setup.asSymbol, mantaData[num][0])};
                                        },
                                        2,{
                                                setups.do{arg setup;
manta.removeSliderSetup(setup.asSymbol, mantaData[num][0])};
                                        },
                                        3,{
                                                setups.do{arg setup;
manta.removeNoteOnSetup(setup.asSymbol, mantaData[num][0])};
                                                setups.do{arg setup;
manta.removeNoteOffSetup(setup.asSymbol, mantaData[num][0])};
                                        }
```

```
                    );
                });

                mantaData.put(num, nil);
        });
}

addAssignButton {|rect, num|
        assignButtons.add(SCButton.new(win,rect)
                .states_([ [ "A", Color.red, Color.black ] ,[ "C", Color.black, Color.red ] ])
                .action_{|v|
                        if(v.value==1,{
                                midiHidControl.requestMidiAssignInfo(this, num);
                        },{
                                this.clearMidiHid(num);
                                waitForSet = false;
                        })
                });
}

setControlWNext {arg controllerType, dataType, controlsIndex, controlInfo;
        //controllerType - 0 is Manta, 1 is Midi
        //dataType - 0 is onOff info, 1 is continuous

        if(dataType==waitForType, {
                waitForSet = false;
                if(controllerType ==0,{
                        this.setManta(controlInfo[0], controlInfo[1], controlsIndex);
                },{
                        this.setMidi(controlInfo[0], controlInfo[1], controlsIndex);
                });
                midiHidControl.clearInstantMidiAssignInfo;
        },{
                //this should only happen with midi buttons
                if(controllerType==1,{
                        this.setMidi(controlInfo[0], 2, controlsIndex);
                        midiHidControl.clearInstantMidiAssignInfo;
                })
        });
}

addDoubleAssignButton {|rect, num, type, layout = \horz, specialClearAction = nil|
        assignButtons.add(DoubleAssignButton.new(win,rect,layout)
                .instantAction_{|butt|
                        if(butt.value==1,{
                                midiHidControl.requestInstantMidiAssignInfo(this, num);
```

```
                                        waitForSet = true;
                                        waitForType = type;
                                        if(type == 3, {waitForType = 0});

                                },{
                                        midiHidControl.clearInstantMidiAssignInfo;
                                        if(specialClearAction.isNil,{
                                                this.clearMidiHid(num);
                                        },{
                                                specialClearAction.value;
                                        });
                                        waitForSet = false;
                                })
                        });
        }
}

Module_Mod : MIDIHIDObject {
        var <>group, <>outBus, mixerToSynthBusses, synths, assignAllButtons, xmlSynth, modName,
numChannels = 2, rout;

        *new {arg group, outBus, midiHidControl, manta, bcf2000, setups;


^super.new.group_(group).outBus_(outBus).midiHidControl_(midiHidControl).manta_(manta).bcf200
0_(bcf2000).setups_(setups).init;
        }

        makeWindow {arg name, rect;
                win = SCWindow.new(name, rect);
                win.userCanClose_(false);
                win.front;
                win.alwaysOnTop_(true);
                modName = name;
        }


        getInternalBusses {
                ^mixerToSynthBusses;
        }

        pause {
                synths.do{|item| if(item!=nil, item.set(\pauseGate, 0))};
        }

        unpause {
```

```
		synths.do{|item| if(item!=nil,{item.set(\pauseGate, 1); item.run(true);})};
}

show {
		win.visible = true;
}

hide {
		win.visible = false;
}

numBusses {
		^mixerToSynthBusses[0].numChannels;
}

loadMidiData {arg xmlSynth;
		midiData.size.do{arg i;
				midiHidTemp = xmlSynth.getAttribute("midi"++i.asString);
				if(midiHidTemp!=nil,{
						midiHidTemp = midiHidTemp.interpret;
						if(midiHidTemp!=nil,{
								this.setMidi(midiHidTemp[0],midiHidTemp[1],i);
						});
				});
		};
}

load {arg xmlSynth;
		this.loadMidiData(xmlSynth);
		mantaData.size.do{arg i;
				midiHidTemp = xmlSynth.getAttribute("manta"++i.asString);
				if(midiHidTemp!=nil,{
						midiHidTemp = midiHidTemp.interpret;
						if(midiHidTemp!=nil,{
								this.setManta(midiHidTemp[0], midiHidTemp[1], i);
						});
				});
		};

		rout = Routine({
				group.server.sync;
				controls.do{arg item, i;
						midiHidTemp = xmlSynth.getAttribute("controls"++i.asString);
						if(midiHidTemp!=nil,{
								controls[i].valueAction_(midiHidTemp.interpret);
						});
```

```
                };
        });
        AppClock.play(rout);
        win.bounds_(xmlSynth.getAttribute("bounds").interpret);
        win.front;
    }

    save {arg xmlDoc;
        xmlSynth = xmlDoc.createElement(modName);
        mantaData.do{arg item, i;
            xmlSynth.setAttribute("manta"++i.asString, item.asString);
        };
        midiData.do{arg item, i;
            xmlSynth.setAttribute("midi"++i.asString, item.asString);
        };
        controls.do{arg item, i;
            xmlSynth.setAttribute("controls"++i.asString, item.value.asString);
        };
        xmlSynth.setAttribute("bounds", win.bounds.asString);
        ^xmlSynth;
    }

    killMe {
        midiData.size.do{arg i; this.clearMidiHid(i)};
        win.close;
        if(synths!=nil,{
            synths.do{arg item; if(item!=nil,{item.set(\gate, 0)})};
        });
        mixerToSynthBusses.do{arg item; item.free};
        this.killMeSpecial;
    }

    killMeSpecial {

    }
}
```

Appendix II

Selected Modules

Delayed-Controlled Feedback	119
Loop Machine	128
Cycle Gripper	132
Freeze	138
Harmonic Shifter X	144
Analysis/Resynthesis	157

```
//Delayed-Controlled Feedback

MulArray {var <>size, <>mulBus, <>multi0, <>multi1, <>multi4, <>depth, <>calculateMethod, temp,
<>threshold=0.5, depth, previousVolArrays, avgVolArray, <>mulArray, counterArray, incrementArray,
task, maxes, mulArray2, <>numBinsLong = 10, clipArray;

        *new {arg size, mulBus, multi0, multi1, multi4, depth, calculateMethod;


^super.new.size_(size).mulBus_(mulBus).multi0_(multi0).multi1_(multi1).multi4_(multi4).depth_(dep
th).calculateMethod_(calculateMethod).init;
        }

          init {
                incrementArray = Array.newClear(0);
                size.do{incrementArray = incrementArray.add(rrand(0.04,0.06))};

                previousVolArrays = Array.newClear(0);
                    depth.do{|i|
                            temp = Array.fill(size, 0);
                        previousVolArrays = previousVolArrays.add(temp);
                    };
                mulArray = List.fill(size, 1);
                    mulArray2 = List.fill(size, 1);
                mulBus.setn(mulArray);
                counterArray = List.fill(size, 0);
                    maxes = List.newClear(0);

                    clipArray = multi4.value;

                multi4.action = {arg array;
                        array.value.postln;
                            clipArray=array.value;
                    };

          }

        addVolArray {arg inArray;
                previousVolArrays = previousVolArrays.shift(1);
                    previousVolArrays.put(0, inArray);
                    avgVolArray = (previousVolArrays.sum)/(previousVolArrays.size);
                    if(calculateMethod==0,{
                        this.calculateMulArray0;
                },{
                        this.calculateMulArray1;
                });
```

```
}

calculateMulArray0 {
    avgVolArray.do{|item, i|
        if(item>(threshold*clipArray[i]),{
                counterArray.put(i, 10);
            })
        };
    counterArray.do{|item,i|
        if((item>0)&&(mulArray[i]>0),{
                mulArray.put(i, (mulArray[i]-incrementArray[i]).abs);
                counterArray.put(i, item-1);
            },{
                if(mulArray[i]<1,{
                    mulArray.put(i, (mulArray[i]+incrementArray[i]).abs);
                    incrementArray.put(i, rrand(0.04,0.06));
                });
                if(mulArray[i]>1, {mulArray.put(i, 1)});
            });
        };
        mulBus.setn(mulArray);
}

calculateMulArray1 {
    avgVolArray.do{|item, i|
        if(item>(threshold*clipArray[i]),{
                counterArray.put(i, 10);
                if((maxes.size<numBinsLong)&&(maxes.includes(i).not),
{maxes.add(i)});
            })
        };
    counterArray.do{|item,i|
        if((item>0)&&(mulArray[i]>0),{
                mulArray.put(i, (mulArray[i]-incrementArray[i]).abs);
                counterArray.put(i, item-1);
            },{
                if(mulArray[i]<1,{
                    mulArray.put(i, (mulArray[i]+incrementArray[i]).abs);
                    incrementArray.put(i, rrand(0.02,0.03));
                    maxes.remove(i);
                });
                if(mulArray[i]>1, {mulArray.put(i, 1)});
            });
    };
    mulArray2.fill(1);
    maxes.do{|item| mulArray2.put(item, mulArray[item])};
```

```
            mulBus.setn(mulArray2);
        }

        turnDispOn {
            task = Task({ {
                {multi0.value_(avgVolArray)}.defer;
                    if(calculateMethod==0,{
                            {multi1.value_(mulArray.asArray)}.defer;                    },{
                            {multi1.value_(mulArray2.asArray)}.defer;
                    });
                    0.25.wait;
                }.loop });
                task.start;
        }

        turnDispOff {
                task.stop
        }

        killMe {
                task.stop;
        }

}

FeedbackDetection {
        var <>group, <>inBus, <>outBus, <>size, <>multis, thresholdShort=0.5, thresholdLong = 0.5,
temp, mulBus0, mulBus1, synth, task, getArray, finalVals, tot, sepArray, buffer, inc, waitTime=0.5,
mulArray0, mulArray1;

      *new {arg group, inBus, outBus, multis;
                ^super.new.group_(group).inBus_(inBus).outBus_(outBus).multis_(multis).init;
        }

         *initClass {
                {
                   SynthDef("spectralLimiter_mod", {arg inBus, outBus, bufnum, threshold, mulBus0,
mulBus1, pauseGate = 1, gate=1;
                                var mulArray0, mulArray1, chain0, chain1, env, pauseEnv, in, fftSize,
clip;

                                fftSize = 4096;

                        in = In.ar(inBus, 2);

                        mulArray0 = In.kr(mulBus0, 560);
```

```
                mulArray1 = In.kr(mulBus1, 560);

            FFT(bufnum, Mix.ar(in));

            chain0 = FFT(LocalBuf(fftSize), in[0]);
                chain1 = FFT(LocalBuf(fftSize), in[1]);

            chain0 = chain0.pvcollect(fftSize, {|mag, phase, index|
                mag*mulArray0[index]*mulArray1[index]
                }, frombin: 0, tobin: 559, zeroothers: 0);

            chain1 = chain1.pvcollect(fftSize, {|mag, phase, index|
                mag*mulArray0[index]*mulArray1[index]
                }, frombin: 0, tobin: 559, zeroothers: 0);

            env = EnvGen.kr(Env.asr(2,1,0), gate, doneAction:2);
                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

            Out.ar(outBus, [IFFT(chain0), IFFT(chain1)]*env*pauseEnv);
        }).writeDefFile;
    }.defer(1);
}

init {
    size = 560;

    mulBus0 = Bus.control(group.server, size);
    mulBus1 = Bus.control(group.server, size);

    mulArray0 = MulArray(size, mulBus0, multis[0], multis[1], multis[4], 25, 0);
    mulArray1 = MulArray(size, mulBus1, multis[2], multis[3], multis[4], 250, 1);

    {
        buffer = Buffer.alloc(group.server, 4096, 1);
        group.server.sync;
            0.4.wait;

        synth = Synth("spectralLimiter_mod", [\inBus, inBus, \outBus, outBus, \bufnum,
buffer.bufnum, \threshold, 0.5, \mulBus0, mulBus0.index, \mulBus1, mulBus1.index], group);

        task = Task({ { buffer.getn(2, size*2, { arg buf;
                var z, x;
            z = buf.clump(2).flop;
            z = [Signal.newFrom(z[0]), Signal.newFrom(z[1])];
                x = Complex(z[0], z[1]);
                getArray = (Array.newFrom(x.magnitude)* 0.05).squared;
```

```
                    mulArray0.addVolArray(getArray);
                        mulArray1.addVolArray(getArray);
                    }); waitTime.wait;}.loop });
                    task.start;
            }.fork;
    }

setThresholdShort {arg in;
        mulArray0.threshold_(in);
    }

setThresholdLong {arg in;
        mulArray1.threshold_(in);
    }

setNumBinsLong {arg in;
        mulArray1.numBinsLong_(in);
    }

setOverallThresh {arg thresh;
            synth.set(\threshold, thresh);
}

    pause {
        if (task!=nil,{
                task.pause;
                synth.set(\pauseGate, 0);
        });
    }

    unpause {
            task.resume;
        synth.set(\pauseGate, 1);
            synth.run(true);
}

setAnalSec {arg analSec;
        waitTime = 1/analSec;
    }

    turnDispOn {
            mulArray0.turnDispOn;
            mulArray1.turnDispOn;
    }

    turnDispOff {
```

```
                mulArray0.turnDispOff;
                mulArray1.turnDispOff;
        }

        killMe {
                mulArray0.killMe;
                mulArray1.killMe;
                task.stop;
            synth.set(\gate, 0);
        }
}

DelayFeedback_Mod : Module_Mod {
        var volBus, feedbackDetection, synthGroup, limiterGroup, multis, tempArray, transferBus;

        *initClass {
                {

                    SynthDef("delayFeedback_mod", {arg inBus, outBus, volBus, delay = 0.02158,
varAmount=0.5, varFreq=0.02, pauseGate = 1, gate = 1;
                            var in, convolveAudio, volume, env, pauseEnv;

                        volume = In.kr(volBus);

                        in  = LPF.ar(In.ar(inBus), 3000);

                        in = Compander.ar(in, in,
                                thresh: 0.5,
                                        slopeBelow: 1,
                                        slopeAbove: 0.5,
                                        clampTime: 0.01,
                                        relaxTime: 0.01
                                );

                        env = EnvGen.kr(Env.asr(0,1,0), gate, doneAction:2);
                                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

                        in = DelayC.ar(in, 0.5, (LFNoise2.kr(varFreq).range(0, varAmount)+delay));

                        Out.ar(outBus, Limiter.ar(in, 0.1, 0.03)*volume);
                    }).writeDefFile;
                }.defer;
        }

        init {
            this.makeWindow("DelayFeedback", Rect(318, 645, 345, 250));
```

```
        this.initControlLists(3);

    this.createMultis;

    mixerToSynthBusses = List.new;
        mixerToSynthBusses.add(Bus.audio(group.server, 2));

    transferBus = Bus.audio(group.server,2);

    volBus = Bus.control(group.server);

    synths = List.new;

    limiterGroup = Group.tail(group);
        synthGroup = Group.tail(group);

    feedbackDetection = FeedbackDetection(limiterGroup, mixerToSynthBusses[0].index,
transferBus, multis);

    4.do{|i|synths.add(Synth("delayFeedback_mod",[\inBus, transferBus.index+(i%2), \outBus,
outBus.index+(i%2), \volBus, volBus.index, \delay, 0.025+(i*0.1234)], synthGroup))};

    controls = List.new;
        assignButtons = List.new;

    controls.add(EZSlider.new(win,Rect(5, 5, 80, 220), "vol", ControlSpec(0,1,'amp'),
            {|v|
             volBus.set(v.value);
        }, 0, layout:\vert));
        this.addDoubleAssignButton(Rect(5, 230, 80, 20),0,1);

    controls.add(EZSlider.new(win,Rect(90, 5, 80, 220), "maxDel", ControlSpec(0.02,
0.1,'linear'),
            {|v|
                //feedbackDetection.setOverallThresh(v.value);
                synths.do{arg item; item.set(\varFreq, v.value)};
            }, 0.02, layout:\vert));
        this.addDoubleAssignButton(Rect(90, 230, 80, 20),1,1);

    controls.add(EZSlider.new(win,Rect(175, 5, 80, 220), "threshShort",
ControlSpec(0,1,'linear'),
            {|v|
             feedbackDetection.setThresholdShort(v.value);
        }, 0.5, layout:\vert));
        this.addDoubleAssignButton(Rect(175, 230, 80, 20),2,1);
```

```
            controls.add(EZSlider.new(win,Rect(260, 5, 80, 220), "threshLong",
ControlSpec(0,1,'linear'),
                        {|v|
                         feedbackDetection.setThresholdLong(v.value);
                    }, 0.5, layout:\vert));
                this.addDoubleAssignButton(Rect(260, 230, 80, 20),3,1);

            controls.add(EZSlider.new(win,Rect(345, 5, 80, 220), "numBinsLong",
ControlSpec(2,15,'linear',1),
                        {|v|
                         feedbackDetection.setNumBinsLong(v.value);
                    }, 5, layout:\vert));
                this.addDoubleAssignButton(Rect(345, 230, 80, 20),4,1);

            controls.add(EZSlider.new(win,Rect(430, 5, 80, 220), "anal/Sec",
ControlSpec(2,15,'linear',1),
                        {|v|
                         feedbackDetection.setAnalSec(v.value);
                    }, 2, layout:\vert));
                this.addDoubleAssignButton(Rect(430, 230, 80, 20),5,1);

            controls.add(SCButton(win, Rect(5, 260, 90, 20))
                        .states_([["dispOff", Color.blue, Color.black],["dispOn", Color.black,
Color.blue]])
                        .action_({arg butt;
                         if(butt.value==0,{
                                    feedbackDetection.turnDispOff
                            },{
                                    feedbackDetection.turnDispOn
                            })
                        })
                );


            //multichannel button
            numChannels = 2;
        }

    createMultis {arg size = 560;

            multis = List.newClear(0);

            multis.add(SCMultiSliderView(win, Rect(515, 5, size, 350)));
                tempArray = Array.new;
                size.do({arg i;
                    tempArray = tempArray.add(i/size);
```

```
    });
    multis[0].value_(tempArray);
multis[0].readOnly = false;
multis[0].xOffset_(1);
    multis[0].thumbSize_(1);
multis[0].strokeColor_(Color.black);
    multis[0].drawLines_(true);
    multis[0].drawRects_(false);
multis[0].indexThumbSize_(0.5);
    multis[0].valueThumbSize_(0.1);
multis[0].isFilled_(false);
multis[0].gap_(1);


multis.add(SCMultiSliderView(win, Rect(515+size, 5, size, 350)));
multis[1].value_(tempArray);
multis[1].readOnly = false;
multis[1].xOffset_(1);
    multis[1].thumbSize_(1);
multis[1].strokeColor_(Color.black);
    multis[1].drawLines_(true);
    multis[1].drawRects_(false);
multis[1].indexThumbSize_(0.5);
    multis[1].valueThumbSize_(0.1);
multis[1].isFilled_(false);
multis[1].gap_(1);

multis.add(SCMultiSliderView(win, Rect(515, 355, size, 350)));
multis[2].value_(tempArray);
multis[2].readOnly = false;
multis[2].xOffset_(1);
    multis[2].thumbSize_(1);
multis[2].strokeColor_(Color.black);
    multis[2].drawLines_(true);
    multis[2].drawRects_(false);
multis[2].indexThumbSize_(0.5);
    multis[2].valueThumbSize_(0.1);
multis[2].isFilled_(false);
multis[2].gap_(1);

multis.add(SCMultiSliderView(win, Rect(515+size, 355, size, 350)));
multis[3].value_(tempArray);
multis[3].readOnly = false;
multis[3].xOffset_(1);
    multis[3].thumbSize_(1);
multis[3].strokeColor_(Color.black);
```

```
                multis[3].drawLines_(true);
                multis[3].drawRects_(false);
            multis[3].indexThumbSize_(0.5);
                multis[3].valueThumbSize_(0.1);
            multis[3].isFilled_(false);
            multis[3].gap_(1);

            multis.add(SCMultiSliderView(win, Rect(5, 355, size, 350)));
                tempArray = Array.newClear(0);
                size.do{|i|tempArray = tempArray.add(1-(i/(size*2)))};
            multis[4].value_(tempArray);
            multis[4].readOnly = false;
            multis[4].xOffset_(1);
                multis[4].thumbSize_(1);
            multis[4].strokeColor_(Color.black);
                multis[4].drawLines_(true);
                multis[4].drawRects_(false);
            multis[4].indexThumbSize_(0.5);
                multis[4].valueThumbSize_(0.1);
            multis[4].isFilled_(false);
            multis[4].gap_(1);
        }

        pause {
            synths.do{|item| if(item!=nil, item.set(\pauseGate, 0))};
            feedbackDetection.pause;
        }

        unpause {
            synths.do{|item| if(item!=nil,{item.set(\pauseGate, 1); item.run(true);})};
            feedbackDetection.unpause;
        }

        killMeSpecial {
                feedbackDetection.killMe;
        }
}

//Loop Machine

LoopMachine_Mod : Module_Mod {
        var volBus0, volBus1, phasorBus, rateBus, recordGroup, playGroup, buffer, dustRate,
rateSwitch, controlIndex;

        *initClass {
                {
```

```
		SynthDef("loopBufRecord_mod", {arg inBus, outBus, bufnum, phasorBus, volBus, smallGate = 1, gate=1, pauseGate=1;
			var in, phasor, vol, env, smallEnv, pauseEnv;

			phasor = Phasor.ar(0, BufRateScale.kr(bufnum)*smallGate, 0, BufFrames.kr(bufnum));
			Out.kr(phasorBus, A2K.kr(phasor));

			in = In.ar(inBus,8);

			vol = In.kr(volBus);

			smallEnv = EnvGen.kr(Env.asr(0.02,1,0.02), smallGate);
				env = EnvGen.kr(Env.asr(0.02,1,0.02), gate, doneAction: 2);
				pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

			BufWr.ar(in, bufnum, phasor, loop:1);

			Out.ar(outBus, in*vol*env*smallEnv*pauseEnv);
		}).writeDefFile;
		SynthDef("loopBufPlay_mod", {arg outBus, bufnum, rateBus, phasorBus, volBus, rateSwitch = 0, gate=1, pauseGate=1;
			var playBack, phaseStart, phase, env, rate, vol, pauseEnv, dust;

			env = EnvGen.kr(Env.asr(0.02,1,0.02), gate, doneAction: 2);
				pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

			vol = In.kr(volBus);

			dust = Dust.kr(5);
				rate = Select.kr(rateSwitch, [In.kr(rateBus), TRand.kr(2,4,dust)*Select.kr(TIRand.kr(0,1,dust), [-1, 1])]);

			phaseStart = Select.kr(rateSwitch, [Latch.kr(In.kr(phasorBus),1), TRand.kr(0,BufFrames.kr(bufnum),dust)]);
			phase = (Phasor.ar(0, BufRateScale.kr(bufnum)*rate, 0, BufFrames.kr(bufnum))+phaseStart).wrap(0, BufFrames.kr(bufnum));

			playBack = BufRd.ar(8, bufnum, phase, loop:1)*env*vol*pauseEnv;

			XOut.ar(outBus, env, playBack);
		}).writeDefFile;
		SynthDef("loopBufDuster_mod", {arg outBus, bufnum, rateBus, phasorBus, volBus, dustRate, gate=1, pauseGate=1;
			var playBack, phaseStart, phase, env, rate, vol, pauseEnv, dust, stepper;
```

```
                        env = EnvGen.kr(Env.asr(0.02,1,0.02), gate, doneAction: 2);
                            pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

                        vol = In.kr(volBus);

                        dust = Dust.kr(dustRate);
                            stepper = Demand.kr(dust, 0, Dseq([0,0,1],inf));

                        rate = TRand.kr(2,4,dust)*Select.kr(TIRand.kr(0,1,stepper), [-1, 1]);

                        phaseStart = TRand.kr(0,BufFrames.kr(bufnum),stepper);
                            phase = (Phasor.ar(0, BufRateScale.kr(bufnum)*rate, 0,
BufFrames.kr(bufnum))+phaseStart).wrap(0, BufFrames.kr(bufnum));

                        playBack = BufRd.ar(8, bufnum, phase, loop:1)*env*vol*pauseEnv;


                        XOut.ar(outBus, Lag.kr(stepper, 0.05)*env, playBack);
                    }).writeDefFile;
                }.defer(1);
        }

        init {
            this.makeWindow("LoopMachine", Rect(700, 700, 270, 235));
                this.initControlLists(7);

            mixerToSynthBusses = List.new;
                mixerToSynthBusses.add(Bus.audio(group.server, 8));

            buffer = Buffer.alloc(group.server, group.server.sampleRate*8, 8);

            volBus0 = Bus.control(group.server);
                volBus1 = Bus.control(group.server);
                phasorBus = Bus.control(group.server);
                rateBus = Bus.control(group.server);

            synths = List.newClear(2);
                recordGroup = Group.head(group);
                playGroup = Group.tail(group);
                synths.put(0, Synth("loopBufRecord_mod", [\inBus, mixerToSynthBusses[0].index,
\outBus, outBus, \bufnum, buffer.bufnum, \phasorBus, phasorBus.index, \volBus, volBus0.index],
recordGroup));

            controls = List.new;
                assignButtons = List.new;
                assignAllButtons = List.new;
```

```
		controls.add(EZSlider(win, Rect(5, 5, 60, 160),"inVol", ControlSpec(0.0,1.0,\amp),
				{|v|
				 volBus0.set(v.value);
			}, 0, true, 40, 40, 0, 16, \vert));
		this.addDoubleAssignButton(Rect(5, 165, 60, 16), 0, 1);

		controls.add(EZSlider(win, Rect(70, 5, 60, 160),"loopVol", ControlSpec(0.0,1.0,\amp),
				{|v|
				 volBus1.set(v.value);
			}, 0, true, 40, 40, 0, 16, \vert));
		this.addDoubleAssignButton(Rect(70, 165, 60, 16), 1, 1);

		controls.add(EZSlider(win, Rect(135, 5, 60, 160),"rate", ControlSpec(-4.0,4.0),
				{|v|
				 rateBus.set(v.value);
			}, 0, true, 40, 40, 0, 16, \vert));
		this.addDoubleAssignButton(Rect(135, 165, 60, 16), 2, 1);

		controls.add(EZRanger(win, Rect(200, 5, 60, 180), "dust", ControlSpec(2, 10),
				{arg vals;
				 dustRate = vals.value;
				 synths[1].set(\dustRate, dustRate);
				}, [4,6], true, layout:\vert));

		controls.add(SCButton(win, Rect(5, 185, 60, 16))
				.states_([["input", Color.red, Color.black],["loop", Color.black, Color.green]])
				.action_{arg butt;
					synths[1].set(\gate, 0);
					synths.put(1, Synth("loopBufPlay_mod", [\outBus, outBus, \bufnum,
buffer.bufnum, \rateBus, rateBus.index, \phasorBus, phasorBus.index, \volBus, volBus1.index,
\dustRate, rrand(dustRate[0],dustRate[1]), \rateSwitch, rateSwitch], playGroup));
					synths[0].set(\smallGate, 0);
					this.setButtons(4);
			});
		this.addDoubleAssignButton(Rect(5, 205, 60, 16), 4, 0);

		rateSwitch=0;
		controls.add(SCButton(win, Rect(70, 185, 60, 16))
				.states_([["dust", Color.red, Color.black],["dust", Color.black, Color.green]])
				.action_{arg butt;
					synths[1].set(\gate, 0);
					synths[0].set(\smallGate, 1);
					synths.put(1, Synth("loopBufDuster_mod", [\outBus, outBus,
\bufnum, buffer.bufnum, \rateBus, rateBus.index, \phasorBus, phasorBus.index, \volBus,
volBus1.index, \dustRate, rrand(dustRate[0],dustRate[1])], playGroup));
```

```
                                        this.setButtons(5);
                        });
                this.addDoubleAssignButton(Rect(70, 205, 60, 16), 5, 0);

                controls.add(SCButton(win, Rect(135, 185, 60, 16))
                                .states_([["norm", Color.red, Color.black],["norm", Color.black, Color.green]])
                        .action_{
                                synths[1].set(\gate, 0);
                                        synths[0].set(\smallGate, 1);
                                        this.setButtons(6);
                        });
                this.addDoubleAssignButton(Rect(135, 205, 60, 16), 6, 0);
                controls[6].valueAction = 1;
        }

        setButtons {arg val;
                controlIndex = val;
                (4..6).do{|i| controls[i].value=0};
                controls[val].value=1;
        }

        killMeSpecial {
                volBus0.free;
                volBus1.free;
                phasorBus.free;
                rateBus.free;
        }
}

//Cycle Gripper

CycleGripper_Mod : Module_Mod {
        var bufs, bufnums, bufPlayers, mantaIsSet, midiIsSet, lastValue, seqs, repeatSeq, numPlays,
countToPlays, tempList;

        *initClass {
                {
                   SynthDef("cycleGripper_mod", {arg inBus, outBus, trigRateDust=0,
trigRateImpulse=0, mode=0, inDelay = 0.02, t_trig = 0, gate = 1, pauseGate = 1;
        var trig, div0, div1, switch0, switch1, source, local, delay, delayTime;
                                        var triga, div0a, div1a, switch0a, switch1a, env, pauseEnv;

                                env = EnvGen.kr(Env.asr(0.01,1,0.01), gate, doneAction:2);
                                        pauseEnv = EnvGen.kr(Env.asr(0.01,1,0.01), pauseGate, doneAction:1);

                                trig = Dust.kr(trigRateDust) + Impulse.kr(trigRateImpulse);
```

```
                        div0 = PulseDivider.kr(trig, 2, 0);
                                div1 = PulseDivider.kr(trig, 2, 1);
                                switch0 = SetResetFF.kr(div0,div1);
                                switch1 = SetResetFF.kr(div1,div0);

                        div0a = Trig.kr(t_trig, 0.01);
                                div1a = Trig.kr(TDelay.kr(t_trig, inDelay), 0.01);
                                switch0a = SetResetFF.kr(div0a,div1a);
                                switch1a = SetResetFF.kr(div1a,div0a);

                        switch0 = Select.kr(mode, [switch0, switch0a, 1]);
                                switch1 = Select.kr(mode, [switch1, switch1a, 0]);

                        source = In.ar(inBus, 2);

                        delayTime = Select.kr(mode, [TRand.kr(64/44100, 1024/44100, trig), inDelay,
inDelay]);

                        delay = DelayN.ar(LocalIn.ar(2), 8192/44100, delayTime);

                        delay = Compander.ar((switch1*delay), (switch1*delay), 1, 1, 0.5, 0.01,
0.01).distort.clip2(0.8);
                                //delay = (delay+PitchShift.ar(delay, 0.02, TRand.kr(0.9, 1.1, switch1),
0.01, 0));

                        local = Mix.ar([(switch0*source),delay]);

                        LocalOut.ar(local.reverse*1.2);

                        Out.ar(outBus, local*env*pauseEnv);
                    }).writeDefFile;
                }.defer(1);
        }

        getDelay {arg num;
                numPlays.put(num, numPlays[num]+1);
                    if(numPlays[num]>countToPlays[num], {
                            numPlays.put(num, 0);
                            countToPlays.put(num, rrand(12,17));
                        tempList = List.newClear;
                        rrand(3,5).do{tempList.add(rrand(2**(9+num), 2**(10+num))/44100)};
                        seqs.put(num, Pseq.new(tempList, inf).asStream);
                });
                    lastValue.put(num, seqs[num].next);
                    ^lastValue[num].postln;
```

```
}

init {
    this.makeWindow("CycleGripper",Rect(946, 618, 60*9, 20));
        this.initControlLists(8);

    mixerToSynthBusses = List.new;
        mixerToSynthBusses.add(Bus.audio(group.server, 8));

    synths = List.newClear(4);
        synths.put(0, Synth("cycleGripper_mod", [\inBus, mixerToSynthBusses[0].index,
\outBus, outBus, \trigRateDust, 0, \trigRateImpulse, 0, \mode, 2], group));

    seqs = List.newClear(0);
        5.do{arg i; seqs.add(Pseq.new(#[0.1], inf).asStream)};
        lastValue = List[seqs[0].next, seqs[1].next, seqs[2].next, seqs[3].next, seqs[4].next];
        numPlays = List[0,0,0,0,0];
        countToPlays = List[0,0,0,0,0];
    repeatSeq = 0;

    controls = List.new;
        assignButtons = List.new;
        assignAllButtons = List.new;

    this.addDoubleAssignButton(Rect(0, 0, 60, 20), 0, 0);

    controls.add(SCButton.new(win,Rect(60, 0, 60, 20))
            .states_([["allOff", Color.red, Color.black ]])
            .action_{|v|
                "allOff".postln;
                synths.do{arg item;
                    if(item!=nil,{
                        item.set(\trigRateDust, 0);
                        item.set(\trigRateImpulse, 0);
                        item.set(\mode, 2);
                })
                }
            });
    controls.add(SCButton.new(win,Rect(120, 0, 60, 20))
            .states_([["manOn0", Color.red, Color.black ]])
            .action_{|v|
                synths.do{arg item;
                    if(item!=nil,{
                        item.set(\trigRateDust, 0);
                        item.set(\trigRateImpulse, 0);
                        item.set(\mode, 1);
```

```
                                item.set(\inDelay, this.getDelay(0));
                                item.set(\t_trig, 1);
                        })
                        }
                });
controls.add(SCButton.new(win,Rect(180, 0, 60, 20))
                .states_([["manOn1", Color.red, Color.black ]])
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                item.set(\trigRateDust, 0);
                                item.set(\trigRateImpulse, 0);
                                item.set(\mode, 1);
                                item.set(\inDelay, this.getDelay(1));
                                item.set(\t_trig, 1);
                        })
                        }
                });
controls.add(SCButton.new(win,Rect(240, 0, 60, 20))
                .states_([["manOn2", Color.red, Color.black ]])
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                item.set(\trigRateDust, 0);
                                item.set(\trigRateImpulse, 0);
                                item.set(\mode, 1);
                                item.set(\inDelay, this.getDelay(2));
                                item.set(\t_trig, 1);
                        })
                        }
                });
controls.add(SCButton.new(win,Rect(300, 0, 60, 20))
                .states_([["manOn3", Color.red, Color.black ]])
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                item.set(\trigRateDust, 0);
                                item.set(\trigRateImpulse, 0);
                                item.set(\mode, 1);
                                item.set(\inDelay, this.getDelay(3));
                                item.set(\t_trig, 1);
                        })
                        }
                });
controls.add(SCButton.new(win,Rect(360, 0, 60, 20))
                .states_([["manOn4", Color.red, Color.black ]])
```

```
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                        item.set(\trigRateDust, 0);
                                        item.set(\trigRateImpulse, 0);
                                        item.set(\mode, 1);
                                        item.set(\inDelay, this.getDelay(4));
                                        item.set(\t_trig, 1);
                        })
                        }
                });
        controls.add(SCButton.new(win,Rect(420, 0, 60, 20))
                .states_([["Dust", Color.red, Color.black ]])
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                        item.set(\trigRateImpulse, rrand(5,12));
                                        item.set(\trigRateDust,rrand(5,25));
                                        item.set(\mode, 0);
                        })
                        }
                });
        controls.add(SCButton.new(win,Rect(480, 0, 60, 20))
                .states_([["Impulse", Color.red, Color.black ]])
                .action_{|v|
                        synths.do{arg item;
                                if(item!=nil,{
                                        item.set(\trigRateImpulse, rrand(5,25));
                                        item.set(\trigRateDust, 0);
                                        item.set(\mode, 0);
                        })
                        }
                });

        //multichannel button
        numChannels = 2;
        controls.add(SCButton(win,Rect(10, 25, 60, 20))
                .states_([["2", Color.black, Color.white],["4", Color.black, Color.white],["8",
Color.black, Color.white]])
                .action_{|butt|
                 switch(butt.value,
                                0, {
                                        numChannels = 2;
                                3.do{|i| synths[i+1].set(\gate, 0)};
                        },
                                1, {
```

```
                                        synths.put(1, Synth("cycleGripper_mod", [\inBus,
mixerToSynthBusses[0].index+2, \outBus, outBus.index+2, \trigRateDust, 0, \trigRateImpulse, 0,
\mode, 2], group));
                                        numChannels = 4;
                                          },
                                          2, {
                                                  if(numChannels==2,{
                                              synths.put(1, Synth("cycleGripper_mod", [\inBus,
mixerToSynthBusses[0].index+2, \outBus, outBus.index+2, \trigRateDust, 0, \trigRateImpulse, 0,
\mode, 2], group));
                                        });
                                        synths.put(2, Synth("cycleGripper_mod", [\inBus,
mixerToSynthBusses[0].index+4, \outBus, outBus.index+4, \trigRateDust, 0, \trigRateImpulse, 0,
\mode, 2], group));
                                                  synths.put(3, Synth("cycleGripper_mod", [\inBus,
mixerToSynthBusses[0].index+6, \outBus, outBus.index+6, \trigRateDust, 0, \trigRateImpulse, 0,
\mode, 2], group));
                                        numChannels = 8;
                                          }
                                )
                        };
                );

                mantaIsSet = false;
                    midiIsSet = false;
        }

        setManta {arg buttonNum, buttonType, controlsIndex;
                8.do{arg i;
                    mantaData.put(i, [buttonNum.deepCopy+i, buttonType.deepCopy]);
                    setups.do{arg setup;
                                this.setMantaForSetup(setup, mantaData[i], i)
                    }
                };

                mantaIsSet = true;
        }

        setMidi {arg data, dataType, controlsIndex;

          }

          clearMidiHid {
                this.clearAllManta
        }
```

```
	clearAllManta {
		mantaData.postln;
		8.do{arg i;
			if(mantaData[i]!=nil,{manta.removeNoteOn(mantaData[i][0])});
			mantaData.put(i, nil);
		};
		mantaIsSet = false;
	}

	clearAllMidi {
		8.do{arg i;
			noteOnResponders[i].remove;
			};
		midiData.put(0, nil);
			midiIsSet = false;
	}

	load {arg xmlSynth;
			this.loadMidiData(xmlSynth);
			//see if the manta is set
			midiHidTemp = xmlSynth.getAttribute("manta0").interpret;
			if(midiHidTemp!=nil,{
				this.setManta(midiHidTemp[0], midiHidTemp[1], 0);
		});
		//change the number of channels if necessary
			midiHidTemp = xmlSynth.getAttribute("controls"++(controls.size-1).asString);
			if(midiHidTemp!=nil,{
			controls[(controls.size-1)].valueAction_(midiHidTemp.interpret);
			});

		win.bounds_(xmlSynth.getAttribute("bounds").interpret);
		win.front;
	}
}

//Freeze

RDVolumeDisplay_Mod {
	var <win, <bounds, top;

	*new {arg win, bounds;
			^super.newCopyArgs(win, bounds);
	}

	init {
	}
```

```
        update {arg val;
                Color.black.set;
                Pen.strokeRect(bounds);
                if(val>=1,{Color.red.set},{Color.green.set});
            top = bounds.top + (bounds.height-((val*bounds.height)));
            Pen.fillRect(Rect(bounds.left+2, top, bounds.width-4, val*bounds.height-4));
        }
}


Freeze_Mod : Module_Mod {
        var group, outBus, midiHidControl, manta, buffer, win, frozenAudioBus, fftBus, levelBus,
updateDisplayRout, volDisplay, displayVol, volumeDisplay, onOff, rout, volBus, threshBus, onOffBus,
muteGateBus, buffers;

        *initClass {
                {
                    SynthDef("rdFreeze_mod", { arg audioInBus, audioOutBus, levelBus, t_keyTrig,
volBus, threshBus, muteGateBus, onOffBus, buffer, gate = 1, pauseGate = 1;
                            var audioIn, fftIn, chain, outSig, trig1, trig2, trig, amp, sin, peak, env,
pauseEnv, muteEnv, vol, thresh, muteGate, onOff;

                            vol = In.kr(volBus);
                                thresh = In.kr(threshBus);
                                muteGate = In.kr(muteGateBus);
                                onOff = In.kr(onOffBus);

                            audioIn = In.ar(audioInBus, 1);

                            amp = Amplitude.kr(audioIn)*EnvGen.kr(Env.asr(0.001, 1, 0.001), onOff);

                            trig1 = Trig1.kr((amp>thresh),0.01);

                            trig2 = Trig1.kr(t_keyTrig, 0.01);

                                trig = (trig1+trig2);

                            chain = FFT(buffer, audioIn);

                            chain = PV_Freeze(chain, 1 - trig);
                                outSig = IFFT(chain);

                            peak = PeakFollower.ar(outSig, 0.99);
                                Out.kr(levelBus, peak);
```

```
                sin = SinOsc.kr(Rand(0.09,1.1))*0.25;

                outSig = Compander.ar(outSig, outSig,
                        thresh: 0.8,
                                slopeBelow: 1,
                                slopeAbove: 0.5,
                                clampTime: 0.01,
                                relaxTime: 0.01
                        );

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);
                        env = EnvGen.kr(Env.asr(0.1,1,0.1), gate, doneAction:2);
                        muteEnv = EnvGen.kr(Env.asr(0.1,1,0.1), muteGate);

                        outSig = outSig*vol*env*pauseEnv*muteEnv;

                Out.ar(audioOutBus, outSig);
            }).writeDefFile;
        }.defer(1);
}

init {
    this.makeWindow("Freeze",Rect(946, 618, 130, 330));
        this.initControlLists(8);

    mixerToSynthBusses = List.new;
        mixerToSynthBusses.add(Bus.audio(group.server, 8));

    volBus = Bus.control(group.server);
        threshBus = Bus.control(group.server);
        onOffBus = Bus.control(group.server);
        muteGateBus = Bus.control(group.server);

    synths = List.newClear(8);

    buffers = List.new;
        8.do{buffers.add(Buffer.alloc(group.server, 2048, 1))};

    controls = List.new;
        assignButtons = List.new;

    levelBus = Bus.control(group.server, 1);

        onOff = 0;

    controls.add(SCButton(win, Rect(0, 0, 65, 20))
```

```
                .states_([
                        ["Locked", Color.red, Color.black],
                                ["Locked", Color.black, Color.green]
                ])
                .action = {arg butt;
                        onOff = 1;
                                onOffBus.set(onOff);
                                butt.value_(1);
                                controls[1].value = 0;
                        });
        this.addDoubleAssignButton(Rect(65, 0, 65, 20), 0, 0);

        controls.add(SCButton(win, Rect(0, 20, 65, 20))
                .states_([
                        ["Free", Color.red, Color.black],
                                ["Free", Color.black, Color.green]
                ])
                .action = {arg butt;
                        onOff = 1;
                                onOffBus.set(onOff);
                                butt.value_(1);
                                controls[0].value = 0;
                        });
        this.addDoubleAssignButton(Rect(65, 20, 65, 20), 1, 0);
        controls[0].valueAction_(1);

        controls.add(EZSlider(win, Rect(0, 40, 40, 200), "Amp", ControlSpec(0.001, 2, \amp),
                        {arg slider;
                         volBus.set(slider.value);
                }, 1, true, layout:\vert));
            this.addDoubleAssignButton(Rect(0, 240, 40, 20),2, 1);

        controls.add(EZSlider(win, Rect(45, 40, 40, 200), "Thresh", ControlSpec(0.0, 1.0, \cos),
{arg slider;
                        threshBus.set(slider.value);
                }, 0.1, true, layout:\vert));
            this.addDoubleAssignButton(Rect(45, 240, 40, 20), 3, 1);

            controls[2].value = 1;
        volDisplay = RDVolumeDisplay_Mod(win, Rect(90, 40, 40, 200));

        controls.add(SCButton(win, Rect(0, 260, 65, 20))
        .states_([
                ["trig", Color.black, Color.red],
                        ["trig", Color.black, Color.blue]
        ])
```

```
            .action_{arg butt;
                        synths.do{arg item; item.set(\t_keyTrig, 1)};
            });

            this.addDoubleAssignButton(Rect(65, 260, 65, 20), 4, 0);

            controls.add(SCButton(win, Rect(0, 280, 65, 20))
            .states_([
                  ["On", Color.red, Color.black],
                        ["On", Color.black, Color.green]
            ])
            .action_{arg butt;
                  muteGateBus.set(1);
                        butt.value = 1;
                        controls[6].value = 0;
                  });
            this.addDoubleAssignButton(Rect(65, 280, 65, 20), 5,0);

            controls.add(SCButton(win, Rect(0, 300, 65, 20))
            .states_([
                  ["Off", Color.red, Color.black],
                        ["Off", Color.black, Color.green]
            ])
            .action_{arg butt;
                  muteGateBus.set(0);
                        butt.value = 1;
                        controls[5].value = 0;
                  });
            this.addDoubleAssignButton(Rect(65, 300, 65, 20), 6,0);

                  controls[1].valueAction_(1);
                  controls[5].valueAction_(1);

            //multichannel button
            numChannels = 2;
            controls.add(SCButton(win,Rect(0, 325, 60, 20))
                        .states_([["2", Color.black, Color.white],["4", Color.black, Color.white],["8",
Color.black, Color.white]])
                        .action_{|butt|
                         switch(butt.value,
                                    0, {
                                          numChannels = 2;
                                    6.do{|i| synths[i+2].set(\gate, 0)};
                              },
                                    1, {
                                    2.do{|i| synths.put(i+2, Synth("rdFreeze_mod", [\audioInBus,
```

```
mixerToSynthBusses[0].index+i+2, \audioOutBus, outBus.index+i+2, \levelBus, levelBus.index,
\onOffBus, onOffBus, \muteGateBus, muteGateBus, \volBus, volBus, \threshBus, threshBus, \buffer,
buffers[i+2]], group))};
                                        numChannels = 4;
                                          },
                                          2, {
                                                  if(numChannels==2,{
                                              2.do{|i| synths.put(i+2, Synth("rdFreeze_mod", [\audioInBus,
mixerToSynthBusses[0].index+i+2, \audioOutBus, outBus.index+i+2, \levelBus, levelBus.index,
\onOffBus, onOffBus, \muteGateBus, muteGateBus, \volBus, volBus, \threshBus, threshBus, \buffer,
buffers[i+2]], group))};
                                        });
                                        4.do{|i| synths.put(i+4, Synth("rdFreeze_mod", [\audioInBus,
mixerToSynthBusses[0].index+i+4, \audioOutBus, outBus.index+i+4, \levelBus, levelBus.index,
\onOffBus, onOffBus, \muteGateBus, muteGateBus, \volBus, volBus, \threshBus, threshBus, \buffer,
buffers[i+4]], group))};
                                        numChannels = 8;
                                          }
                                  )
                          };
                  );

                rout = Routine({
                        group.server.sync;
                              0.4.wait;
                        synths.add(Synth("rdFreeze_mod", [\audioInBus, mixerToSynthBusses[0].index,
\audioOutBus, outBus, \levelBus, levelBus.index, \onOffBus, onOffBus, \muteGateBus, muteGateBus,
\volBus, volBus, \threshBus, threshBus, \buffer, buffers[0]], group));
                                synths.add(Synth("rdFreeze_mod", [\audioInBus,
mixerToSynthBusses[0].index+1, \audioOutBus, outBus.index+1, \levelBus, levelBus.index,
\onOffBus, onOffBus, \muteGateBus, muteGateBus, \volBus, volBus, \threshBus, threshBus, \buffer,
buffers[1]], group));
                });
                AppClock.play(rout);

                    win.front;
          }

        load {arg xmlSynth;
                    rout = Routine( {
                        group.server.sync;
                              0.4.wait;
                        this.loadMidiData(xmlSynth);
                              mantaData.size.do{arg i;
                                        midiHidTemp = xmlSynth.getAttribute("manta"++i.asString).interpret;
                                        if(midiHidTemp!=nil,{
```

```
                                        this.setManta(midiHidTemp[0], midiHidTemp[1], i);
                                });
                            };
                        {[2,3,7].do{arg i;
                                midiHidTemp = xmlSynth.getAttribute("controls"++i.asString);
                                if(midiHidTemp!=nil,{
                                controls[i].valueAction_(midiHidTemp.interpret);
                                });
                            }}.defer;
                        win.bounds_(xmlSynth.getAttribute("bounds").interpret);

                            controls[1].valueAction_(1);
                            controls[5].valueAction_(1);
                    });
                AppClock.play(rout);
        }

          killMeSpecial {
                buffers.do{arg item; item.free};
        }
}

//Harmonic Shifter X


HarmonicShifter_Mod {
        var buffer, shiftBus, shiftByArray, <>shiftWeightArray, loopLength, inBus, group,
<>randomPitch;
          var playID, shiftRout, length, delayTime, windowSize, pitchRatio, pitchDisp, volume;
          var largeEnv, largeEnvBus, largeEnvBusNum, transBus, transBusNum, playRout, resp,
bussesOut1, bussesOut2;
          var outBus1, outBus2, outBus3, outBus4;
          var volGroup, recordGroup, playGroup, synthGroup;
          var playBufSynth, writeBufSynth;

        *new {arg buffer, shiftBus, shiftByArray, shiftWeightArray, loopLength, inBus, group;
                ^super.newCopyArgs(buffer, shiftBus, shiftByArray, shiftWeightArray, loopLength,
inBus, group).init;
        }

          *initClass {
                {
                    SynthDef("largeEnvShifter_mod",{arg outBusNum, attack, decay, gate;
                            var env, out;

                        env = Env.asr(attack,1,decay);
```

```
                Out.kr(outBusNum, EnvGen.kr(env, gate, doneAction: 2));
        }).writeDefFile;
        SynthDef(\playBufShifter_mod, {arg bufNum, outBusNum, length, gate = 1.0;
                var out, env;

            env = Env.new([0.001, 1, 1, 0.001], [3, length-6.1, 3], 'sine');
                out = EnvGen.kr(env, gate, doneAction: 2)*PlayBuf.ar(1, bufNum);
                Out.ar(outBusNum, out);
        }).writeDefFile;
        SynthDef(\writeBufShifter_mod, {arg inBusNum=0, bufNum, rate=1, length;
                var in, out, env;
                in = In.ar(inBusNum);
                env = Env.new([0,1,1,0],[0.1, length-0.2, 0.1], 'linear');
                out =  BufWr.ar(EnvGen.ar(env,1.0,doneAction: 2)*in, bufNum,
Phasor.ar(0, BufRateScale.kr(bufNum) * rate, 0, BufFrames.kr(bufNum)), 0);
        }).writeDefFile;
        SynthDef(\shifterX_mod, {arg inBusNum, outBus1, outBus2, outBus3, outBus4,
length, delayTime, windowSize, pitchRatio,                pitchDisp, xStart, xEnd, yStart, yEnd,
largeEnvBusNum;
                var in, in2, out1, out2, out3, out4, addToSlope, env, bigEnv, largeEnv;

                addToSlope = length/4;
            env = Env.new([0.001,1,1,0.001], [addToSlope+1,length-
(2+(2*addToSlope)),1+addToSlope], 'linear');
                bigEnv = Env.new([0.001, 1, 1, 0.001], [0.001, length + addToSlope +
delayTime +2, 0.001], 'linear');

                largeEnv = In.kr(largeEnvBusNum, 1);

                in = In.ar(inBusNum, 1);
                in2 =  in*EnvGen.kr(bigEnv, doneAction: 2)*EnvGen.kr(env,
doneAction: 0)*largeEnv;

                # out1, out2, out3, out4 = Pan4.ar(
                                PitchShift.ar(DelayL.ar(in2, 0.5, delayTime), windowSize,
pitchRatio, pitchDisp),                                 Line.kr(xStart, xEnd,
length+2.1+delayTime),                                  Line.kr(yStart, yEnd,
length+2.1+delayTime)
                );

                Out.ar(outBus1, out1);
                Out.ar(outBus2, out2);
                Out.ar(outBus3, out3);
                Out.ar(outBus4, out4);
        }).writeDefFile;
    }.defer(1);
```

```
	}

	init {
		volGroup = Group.tail(group);
			recordGroup = Group.tail(group);
			playGroup = Group.tail(group);
			synthGroup = Group.tail(group);

		largeEnvBus = Bus.control(group.server, 1);
			largeEnv = Synth("largeEnvShifter_mod", [\outBusNum, largeEnvBus.index, \attack,
10, \decay, 10, \gate, 1.0], volGroup);

		transBus = Bus.audio(group.server, 1);

		writeBufSynth = Synth("writeBufShifter_mod", [\inBusNum, inBus.postln, \bufNum,
buffer.bufnum, \length, loopLength], recordGroup);

		playRout = Routine.new({{
				playBufSynth = Synth("playBufShifter_mod", [\bufNum, buffer.bufnum,
\outBusNum, transBus.index, \length, loopLength, \gate, 1.0], playGroup);
			loopLength.wait;
		}.loop});

		SystemClock.play(playRout);

		bussesOut1 = Pxrand(#[0,2,4,6], inf).asStream;
			bussesOut2 = Pxrand(#[1,3,5,7], inf).asStream;

		randomPitch = false;
			shiftRout = Routine.new({{
				length = 5.0.rand + 5;
					delayTime = 0.05+0.25.rand;
					windowSize = 0.5+2.0.rand;
					if(shiftWeightArray.sum>0,{
							pitchRatio = shiftByArray.wchoose(shiftWeightArray.normalizeSum);
					},{
							pitchRatio = shiftByArray.choose
					});
					if(pitchRatio == 2, {pitchRatio = shiftByArray.choose*[2,4].choose});
					if(randomPitch, {pitchRatio = pitchRatio*(rrand(0.75,1.25))});
					pitchDisp = 0.025.rand;
					volume = 0.5.rand+0.5;

					outBus1 = shiftBus.index+bussesOut1.next;
					outBus2 = shiftBus.index+bussesOut2.next;
					outBus3 = shiftBus.index+bussesOut1.next;
```

```
                    outBus4 = shiftBus.index+bussesOut2.next;


                Synth("shifterX_mod", [\inBusNum, transBus.index, \outBus1, outBus1, \outBus2,
outBus2, \outBus3, outBus3, \outBus4, outBus4, \length, length, \delayTime, delayTime, \windowSize,
windowSize, \pitchRatio, pitchRatio, \pitchDisp, pitchDisp, \xStart, 1.0.rand2, \xEnd, 1.0.rand2,
\yStart, 1.0.rand2, \yEnd, 1.0.rand2, \largeEnvBusNum, largeEnvBus.index], synthGroup);
                (1.5 + (3.5.rand)).wait;
            }.loop});

        SystemClock.play(shiftRout);
    }

     killMe {
            shiftRout.stop;
        SystemClock.sched(15,
        {
                playRout.stop;
            playBufSynth.set(\gate, 0);
            largeEnvBus.free;
            nil
        });
     }
}

FeedBackLooper_Mod {
    var shiftBus, distortBus, group;
    var busOut, busOutStream, playRout, waitLine1, waitLine2, durLine1, durLine2, startLine,
endLine, lineDeviation, grainyMouse, resp, resp2;

    *new {arg shiftBus, distortBus, group;
            ^super.newCopyArgs(shiftBus, distortBus, group).init;
    }

     *initClass {
            {
            SynthDef("tapeFeedback_mod", {arg inBus, outBus, length, startLine, endLine;
                    var local, in, amp, initEnv, envLength, delayTime, env, vol;

                in = In.ar(inBus);

                in = Compander.ar(in, in, thresh: 0.5, slopeBelow: 1, slopeAbove: 0.5,
clampTime: 0.01,relaxTime: 0.01);

                amp = Amplitude.kr(in);
                    in = in * (amp > 0.02); // noise gate
```

```
                local = LocalIn.ar(1);

                local = DelayN.ar(local, 0.5, Line.kr(startLine, endLine, length));

                local = LeakDC.ar(local);
                local = ((local + in) * 1.25).softclip;

                LocalOut.ar(local);

                env = EnvGen.kr(Env.sine(length), doneAction:2);

                Out.ar(outBus, local * 0.1 * env);
            }).writeDefFile;
        }.defer(1);
    }

    init {
        waitLine1 = LangLine(0.5, 0.25, 30);
            waitLine2 = LangLine(3.0, 1.5, 30);

        durLine1 = LangLine(5.0, 7.5, 50);
            durLine2 = LangLine(10.0, 15.0, 50);

            lineDeviation = 0;

        busOutStream = Pxrand(#[0,1,2,3,4,5,6,7], inf).asStream;

        playRout = Routine.new({{
                startLine = rrand(0.15,0.5);
                    endLine = startLine+(([(startLine-0.15).rand.neg, (startLine-
0.5).neg.rand].choose)*lineDeviation.value);

                Synth("tapeFeedback_mod", [\inBus, shiftBus.index+(8.rand), \outBus,
distortBus.index+busOutStream.next, \length, rrand(durLine1.value,durLine2.value), \startLine,
startLine, \endLine, endLine], group);
                rrand(waitLine1.value,waitLine2.value).wait;
            }.loop});

        SystemClock.play(playRout);
    }

    setDeviation {arg bool;
        if (bool,{
            lineDeviation = LangLine(0.0, 1.0, 30);
        },{
```

```
                        lineDeviation = 0;
                })
        }

        killMe {
                playRout.stop;
        }
}

ShifterFeedback_Mod : Module_Mod {
        var shiftBus, shiftGroup, tapeGroup, mixGroup, topButtons, sideButtons, randomPitch,
shiftSlide, shiftWeightArray, shiftByArray, harmonicShifters, bombVol;
        var distortBus, feedBackLooper, shiftButtons, bufferArray, bombVol, mainVol, rout;

        *initClass {
                {
                  SynthDef("busToOuts2_mod", {arg outBus, bus1, bus2, bus3, bus4, bus5, bus6, bus7,
bus8, bus1a, bus2a, bus3a, bus4a, bus5a, bus6a, bus7a, bus8a, fade, volBus, gate=1, pauseGate = 1;
                        var out, out1, out2, out3, out4, out5, out6, out7, out8, fade2, pauseEnv,
env, vol;

                        vol = In.kr(volBus);

                        fade2 = Lag.kr(fade, 0.05);

                        out1 = XFade2.ar(In.ar(bus1), In.ar(bus1a), fade2);
                                out2 = XFade2.ar(In.ar(bus2), In.ar(bus2a), fade2);
                                out3 = XFade2.ar(In.ar(bus3), In.ar(bus3a), fade2);
                                out4 = XFade2.ar(In.ar(bus4), In.ar(bus4a), fade2);
                                out5 = XFade2.ar(In.ar(bus5), In.ar(bus5a), fade2);
                                out6 = XFade2.ar(In.ar(bus6), In.ar(bus6a), fade2);
                                out7 = XFade2.ar(In.ar(bus7), In.ar(bus7a), fade2);
                                out8 = XFade2.ar(In.ar(bus8), In.ar(bus8a), fade2);

                        out = Pan2.ar(out1,1,0.5)+ Pan2.ar(out2,-0.75,0.5)+ Pan2.ar(out3,0.75,0.5)+
Pan2.ar(out4,-0.6,0.5)+ Pan2.ar(out5,0.6,0.5)+ Pan2.ar(out6,-0.2,0.5)+
Pan2.ar(out7,0.2,0.5)+Pan2.ar(out8,-1,0.5);

                        pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);
                                env = EnvGen.kr(Env.asr(0.2,1,0.2), gate, doneAction:2);

                        Out.ar(outBus, pauseEnv*env*LeakDC.ar(out*Lag.kr(vol, 0.05)));
                }).writeDefFile;

                SynthDef("busToOuts4_mod", {arg outBus, bus1, bus2, bus3, bus4, bus5, bus6, bus7,
bus8, bus1a, bus2a, bus3a, bus4a, bus5a, bus6a, bus7a, bus8a, fade, volBus, gate=1, pauseGate = 1;
```

```
                var outa, outb, out1, out2, out3, out4, out5, out6, out7, out8, fade2,
pauseEnv, env, vol;

                vol = In.kr(volBus);

                fade2 = Lag.kr(fade, 0.05);

                out1 = XFade2.ar(In.ar(bus1), In.ar(bus1a), fade2);
                        out2 = XFade2.ar(In.ar(bus2), In.ar(bus2a), fade2);
                        out3 = XFade2.ar(In.ar(bus3), In.ar(bus3a), fade2);
                        out4 = XFade2.ar(In.ar(bus4), In.ar(bus4a), fade2);
                        out5 = XFade2.ar(In.ar(bus5), In.ar(bus5a), fade2);
                        out6 = XFade2.ar(In.ar(bus6), In.ar(bus6a), fade2);
                        out7 = XFade2.ar(In.ar(bus7), In.ar(bus7a), fade2);
                        out8 = XFade2.ar(In.ar(bus8), In.ar(bus8a), fade2);

                        outa = [out1, out3, out5, out7];
                        outb = [out2, out4, out6, out8];

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);
                        env = EnvGen.kr(Env.asr(0.2,1,0.2), gate, doneAction:2);

                Out.ar(outBus, pauseEnv*env*LeakDC.ar((outa+outb)*Lag.kr(vol, 0.05)));
        }).writeDefFile;

        SynthDef("busToOuts8_mod", {arg outBus, bus1, bus2, bus3, bus4, bus5, bus6, bus7,
bus8, bus1a, bus2a, bus3a, bus4a, bus5a, bus6a, bus7a, bus8a, fade, volBus, gate=1, pauseGate = 1;
                var out, out1, out2, out3, out4, out5, out6, out7, out8, fade2, pauseEnv,
env, vol;

                vol = In.kr(volBus);

                fade2 = Lag.kr(fade, 0.05);

                out1 = XFade2.ar(In.ar(bus1), In.ar(bus1a), fade2);
                        out2 = XFade2.ar(In.ar(bus2), In.ar(bus2a), fade2);
                        out3 = XFade2.ar(In.ar(bus3), In.ar(bus3a), fade2);
                        out4 = XFade2.ar(In.ar(bus4), In.ar(bus4a), fade2);
                        out5 = XFade2.ar(In.ar(bus5), In.ar(bus5a), fade2);
                        out6 = XFade2.ar(In.ar(bus6), In.ar(bus6a), fade2);
                        out7 = XFade2.ar(In.ar(bus7), In.ar(bus7a), fade2);
                        out8 = XFade2.ar(In.ar(bus8), In.ar(bus8a), fade2);

                        out = [out1, out2, out3, out4, out5, out6, out7, out8];

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);
```

```
                env = EnvGen.kr(Env.asr(0.2,1,0.2), gate, doneAction:2);

            Out.ar(outBus, pauseEnv*env*LeakDC.ar(out*Lag.kr(vol, 0.05)));
        }).writeDefFile;

        SynthDef("delayBomb2_mod", {arg buf, outBus, volBus;
                var out, env, env1, dur, vol;

            vol = In.kr(volBus);

            env = EnvGen.kr(Env.sine(Rand(0.5, 0.9), 1));

            out = PlayBuf.ar(1, buf, startPos: MouseX.kr*BufFrames.kr(buf))*env;

            dur = Rand(7.0, 11.0);

            env1 = EnvGen.kr(Env.new([0,1,0],[0.1, dur]), doneAction:2);

            Out.ar(outBus, Pan2.ar(CombC.ar(out, 0.3, Rand(0.1, 0.3), dur*4)*env1,
Rand(-1,1), vol));

        }).writeDefFile;

        SynthDef("delayBomb4_mod", {arg buf, outBus, volBus;
                var out, env, env1, dur, vol;

            vol = In.kr(volBus);

            env = EnvGen.kr(Env.sine(Rand(0.5, 0.9), 1));

            out = PlayBuf.ar(1, buf, startPos: MouseX.kr*BufFrames.kr(buf))*env;

            dur = Rand(7.0, 11.0);

            env1 = EnvGen.kr(Env.new([0,1,0],[0.1, dur]), doneAction:2);

            Out.ar(outBus, PanAz.ar(4, CombC.ar(out, 0.3, Rand(0.1, 0.3), dur*4)*env1,
Rand(-1,1), vol));

        }).writeDefFile;

        SynthDef("delayBomb8_mod", {arg buf, outBus, volBus;
                var out, env, env1, dur, vol;

            vol = In.kr(volBus);
```

```
                    env = EnvGen.kr(Env.sine(Rand(0.5, 0.9), 1));

                    out = PlayBuf.ar(1, buf, startPos: MouseX.kr*BufFrames.kr(buf))*env;

                    dur = Rand(7.0, 11.0);

                    env1 = EnvGen.kr(Env.new([0,1,0],[0.1, dur]), doneAction:2);

                    Out.ar(outBus, PanAz.ar(8, CombC.ar(out, 0.3, Rand(0.1, 0.3), dur*4)*env1,
Rand(-1,1), vol));

                }).writeDefFile;
            }.defer(1);
        }

        init {
            this.makeWindow("ShifterFeedback",Rect(490, 510, 170, 490));

            mixerToSynthBusses = List.new;
                mixerToSynthBusses.add(Bus.audio(group.server));

            this.initControlLists(5);

            synths = List.new;

            bufferArray = Array.with(
                    Buffer.alloc(group.server, 44100*30, 1),
                    Buffer.alloc(group.server, 44100*30, 1),
                    Buffer.alloc(group.server, 44100*30, 1),
                    Buffer.alloc(group.server, 44100*30, 1)
            );

            bombVol = Bus.control(group.server);
                mainVol = Bus.control(group.server);

            topButtons = List.new;

            randomPitch = false;
                shiftSlide = false;

            topButtons.add(SCButton.new(win,Rect(10, 10, 70, 20))
                    .states_([ [ "randOff", Color.red, Color.black ], [ "randOn", Color.black,
Color.green ] ])
                    .action_{|v|
                        if(v.value==1,{randomPitch = true},{randomPitch = false});
                        harmonicShifters.do{arg item; item.randomPitch = randomPitch};
```

```
                });
        topButtons.add(SCButton.new(win,Rect(80, 10, 70, 20))
                .states_([ [ "slideOff", Color.red, Color.black ], [ "slideOn", Color.black,
Color.green ] ])
                .action_{|v| if(v.value==1,{shiftSlide = true},{shiftSlide = false})});
            topButtons.add(SCButton.new(win,Rect(10, 30, 70, 20))
                .states_([ [ "noiseOff", Color.red, Color.black ], [ "noiseOn", Color.black,
Color.green ] ])
                .action_{|v|
                        feedBackLooper = FeedBackLooper_Mod(shiftBus, distortBus,
tapeGroup);
                        v.enabled_(false);
                        topButtons[3].enabled_(true);
                });
        topButtons.add(SCButton.new(win,Rect(80, 30, 70, 20))
                .states_([ [ "slideOff", Color.red, Color.black ], [ "slideOn", Color.black,
Color.green ] ])
                .action_{|v|
                        if(v.value==1,{feedBackLooper.setDeviation(true)},
{feedBackLooper.setDeviation(false)});
                });
        topButtons[3].enabled_(false);

        shiftButtons = List.new;

        shiftButtons.add(SCButton.new(win,Rect(10, 60, 50, 20))
                .states_([ [ "1-4", Color.red, Color.black ], [ "1-4", Color.black, Color.green ] ])
                .action_{|v|
                 shiftWeightArray.put(0, v.value);
                 harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
        shiftButtons.add(SCButton.new(win,Rect(10, 90, 50, 20))
                .states_([ [ "1-2", Color.red, Color.black ], [ "1-2", Color.black, Color.green ] ])
                .action_{|v|
                 shiftWeightArray.put(1, v.value);
                 harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
        shiftButtons.add(SCButton.new(win,Rect(10, 120, 50, 20))
                .states_([ [ "1-1", Color.red, Color.black ], [ "1-1", Color.black, Color.green ] ])
                .action_{|v|
                 shiftWeightArray.put(2, v.value);
                 harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
        shiftButtons.add(SCButton.new(win,Rect(10, 150, 50, 20))
                .states_([ [ "5-4", Color.red, Color.black ], [ "5-4", Color.black, Color.green ] ])
                .action_{|v|
```

```
                    shiftWeightArray.put(3, v.value);
                    harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
            shiftButtons.add(SCButton.new(win,Rect(10, 180, 50, 20))
                    .states_([ [ "3-2", Color.red, Color.black ], [ "3-2", Color.black, Color.green ] ])
                    .action_{|v|
                    shiftWeightArray.put(4, v.value);
                    harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
            shiftButtons.add(SCButton.new(win,Rect(10, 210, 50, 20))
                    .states_([ [ "7-4", Color.red, Color.black ], [ "7-4", Color.black, Color.green ] ])
                    .action_{|v|
                    shiftWeightArray.put(5, v.value);
                    harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });
            shiftButtons.add(SCButton.new(win,Rect(10, 240, 50, 20))
                    .states_([ [ "2to4", Color.red, Color.black ], [ "2to4", Color.black,
Color.green ] ])
                    .action_{|v|
                    shiftWeightArray.put(6, v.value);
                    harmonicShifters.do{arg item; item.shiftWeightArray = shiftWeightArray};
                });

            shiftWeightArray = [0,0,0,0,0,0,0];
            shiftByArray = [0.25, 0.5, 1.0, 1.25, 1.5, 1.75, 2];

            controls = List.new;
            assignButtons = List.new;

            controls.add(EZKnob.new(win,Rect(90, 70, 60, 100), "fade", ControlSpec(-1,1,'linear'),
                    {|v|
                            synths[0].set(\fade, v.value)
                    },-1, true));
            this.addDoubleAssignButton(Rect(90, 170, 60, 20),0,1);

            controls.add(EZSlider.new(win,Rect(90, 190, 60, 160), "vol", ControlSpec(0,1,'amp'),
                    {|v|
                    mainVol.set(v.value)
                }, 0, layout:\vert));
            this.addDoubleAssignButton(Rect(90, 350, 60, 20),1,1);

            harmonicShifters = List.new;
                controls.add(SCButton(win, Rect(10, 370, 160, 20))
                    .states_([ [ "Go0", Color.green, Color.black ], [ "Go1", Color.black,
Color.green ], [ "Go2", Color.green, Color.black ], [ "Go3", Color.black, Color.green ], [ "Done",
Color.black, Color.red ] ])
```

```
                        .action_({arg but;
                         if(but.value<4, {
                                harmonicShifters.add(HarmonicShifter_Mod(bufferArray[but.value-
1].postln, shiftBus, shiftByArray, shiftWeightArray, 30, mixerToSynthBusses[0].index, shiftGroup));
                         },{
                                but.enabled_(false);
                         });
                        }));
                this.addDoubleAssignButton(Rect(10, 390, 160, 20),2,0);

                controls.add(SCButton(win, Rect(10, 400, 160, 20))
                        .states_([ [ "delayBomb", Color.green, Color.black ], [ "delayBomb",
Color.black, Color.green ] ])
                        .action_({arg but;
                         switch(numChannels,
                                        2, {
                                        4.do{arg i; Synth("delayBomb2_mod", [\buf,
bufferArray[i].bufnum, \outBus, outBus, \volBus, bombVol.index], group)};
                                },
                                        4, {
                                        4.do{arg i; Synth("delayBomb4_mod", [\buf,
bufferArray[i].bufnum, \outBus, outBus, \volBus, bombVol.index], group)};
                                },
                                        8, {
                                        4.do{arg i; Synth("delayBomb8_mod", [\buf,
bufferArray[i].bufnum, \outBus, outBus, \volBus, bombVol.index], group)};
                                }
                                )
                        }));

                this.addDoubleAssignButton(Rect(10, 420, 160, 20),3,0);

                controls.add(EZSlider.new(win,Rect(10, 440, 160, 20), "bVol", ControlSpec(0,4,'amp'),
                        {|v|
                         bombVol.set(v.value)
                        }, 8));
                this.addDoubleAssignButton(Rect(10, 460, 160, 20),4,1);

                //multichannel button
                    controls.add(SCButton(win,Rect(10, 485, 60, 20))
                        .states_([["2", Color.black, Color.white],["4", Color.black, Color.white],["8",
Color.black, Color.white]])
                        .action_{|butt|
                                synths[0].set(\gate, 0);
                         switch(butt.value,
                                        0, {
```

```
                                        numChannels = 2;
                                    synths.put(0, Synth("busToOuts2_mod", [\outBus, outBus.postln,
\bus1, shiftBus.index, \bus2, shiftBus.index+1, \bus3, shiftBus.index+2, \bus4, shiftBus.index+3, \bus5,
shiftBus.index+4, \bus6, shiftBus.index+5, \bus7, shiftBus.index+6, \bus8, shiftBus.index+7, \bus1a,
distortBus.index, \bus2a, distortBus.index+1, \bus3a, distortBus.index+2, \bus4a, distortBus.index+3,
\bus5a, distortBus.index+4, \bus6a, distortBus.index+5, \bus7a, distortBus.index+6, \bus8a,
distortBus.index+7, \vol, 0, \fade, -1, \volBus, mainVol.index], mixGroup));
                                },
                                        1, {
                                        numChannels = 4;
                                    synths.put(0, Synth("busToOuts4_mod", [\outBus, outBus.postln,
\bus1, shiftBus.index, \bus2, shiftBus.index+1, \bus3, shiftBus.index+2, \bus4, shiftBus.index+3, \bus5,
shiftBus.index+4, \bus6, shiftBus.index+5, \bus7, shiftBus.index+6, \bus8, shiftBus.index+7, \bus1a,
distortBus.index, \bus2a, distortBus.index+1, \bus3a, distortBus.index+2, \bus4a, distortBus.index+3,
\bus5a, distortBus.index+4, \bus6a, distortBus.index+5, \bus7a, distortBus.index+6, \bus8a,
distortBus.index+7, \vol, 0, \fade, -1, \volBus, mainVol.index], mixGroup));
                                },
                                        2, {
                                        numChannels = 8;
                                    synths.put(0, Synth("busToOuts8_mod", [\outBus, outBus.postln,
\bus1, shiftBus.index, \bus2, shiftBus.index+1, \bus3, shiftBus.index+2, \bus4, shiftBus.index+3, \bus5,
shiftBus.index+4, \bus6, shiftBus.index+5, \bus7, shiftBus.index+6, \bus8, shiftBus.index+7, \bus1a,
distortBus.index, \bus2a, distortBus.index+1, \bus3a, distortBus.index+2, \bus4a, distortBus.index+3,
\bus5a, distortBus.index+4, \bus6a, distortBus.index+5, \bus7a, distortBus.index+6, \bus8a,
distortBus.index+7, \vol, 0, \fade, -1, \volBus, mainVol.index], mixGroup));
                                }
                                )
                        };
                );

                shiftGroup = Group.tail(group);
                    tapeGroup = Group.tail(group);
                    mixGroup = Group.tail(group);

                shiftBus = Bus.audio(group.server, 8);
                    distortBus = Bus.audio(group.server, 8);

                synths.add(Synth("busToOuts2_mod", [\outBus, outBus.postln, \bus1, shiftBus.index,
\bus2, shiftBus.index+1, \bus3, shiftBus.index+2, \bus4, shiftBus.index+3, \bus5, shiftBus.index+4,
\bus6, shiftBus.index+5, \bus7, shiftBus.index+6, \bus8, shiftBus.index+7, \bus1a, distortBus.index,
\bus2a, distortBus.index+1, \bus3a, distortBus.index+2, \bus4a, distortBus.index+3, \bus5a,
distortBus.index+4, \bus6a, distortBus.index+5, \bus7a, distortBus.index+6, \bus8a,
distortBus.index+7, \vol, 0, \fade, -1, \volBus, mainVol.index], mixGroup));
        }

        load {arg xmlSynth;
```

```
                this.loadMidiData(xmlSynth);
                mantaData.size.do{arg i;
                        midiHidTemp = xmlSynth.getAttribute("manta"++i.asString);
                        if(midiHidTemp!=nil,{
                         midiHidTemp = midiHidTemp.interpret;
                         if(midiHidTemp!=nil,{
                                this.setManta(midiHidTemp[0], midiHidTemp[1], i);
                         });
                        });
                };
            rout = Routine.new({
                group.server.sync;
                controls.do{arg item, i;
                        if(i!=2,{
                                midiHidTemp = xmlSynth.getAttribute("controls"++i.asString);
                                        if(midiHidTemp!=nil,{
                                    controls[i].valueAction_(midiHidTemp.interpret);
                                        });
                                });
                        };
                });
            AppClock.play(rout);
                win.bounds_(xmlSynth.getAttribute("bounds").interpret);
            win.front;
        }

        killMeSpecial {
            harmonicShifters.do{arg item; item.killMe;};
                if(feedBackLooper!=nil,{feedBackLooper.killMe});
      }
}


//Analysis/Resynthesis

BitCrusher_Mod : Module_Mod {
      var sr1Bus, sr2Bus, distVolBus, sineVolBus;

        *initClass {
                {
                   SynthDef("bitCrusher2_Mod",{ arg inbus, outbus, sr1Bus, sr2Bus, distVolBus,
sineVolBus, gate = 1, pauseGate = 1;
                                var input, fx1, fx2, sines, sine0, sine1, sine2, sine3, sine4, fund, skip, freq,
hasFreq, env, pauseEnv;
                                var sr1, sr2, distVol, sineVol;
```

```
                sr1 = In.kr(sr1Bus);
                        sr2 = In.kr(sr2Bus);
                        distVol = In.kr(distVolBus);
                        sineVol = In.kr(sineVolBus);

                env = EnvGen.kr(Env.asr(0.1,1,0.1), gate, doneAction:2);

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

                input=In.ar(inbus, 1);
                        fx1=Latch.ar(input.round(0.125),Impulse.ar(sr1));
                        fx2=Latch.ar(input.round(0.1),Impulse.ar(sr2));

                #freq, hasFreq = Pitch.kr(fx1, ampThreshold: 0.02, median: 7);

                fund = TIRand.kr(7, 50, hasFreq);
                        skip = TIRand.kr(1, 5, hasFreq);

                sine0 = SinOsc.ar(freq, 0, hasFreq);
                        sine1 = SinOsc.ar(freq+(fund*skip), 0, hasFreq);
                        sine2 = SinOsc.ar(freq+(2*fund*skip), 0, hasFreq);
                        sine3 = SinOsc.ar(freq+(3*fund*skip), 0, hasFreq);
                        sine4 = SinOsc.ar(freq+(4*fund*skip), 0, hasFreq);

                sines = Pan2.ar(sine0, TRand.kr(-1,1,hasFreq)) + Pan2.ar(sine1, TRand.kr(-
1,1,hasFreq)) + Pan2.ar(sine2, TRand.kr(-1,1,hasFreq)) + Pan2.ar(sine3, TRand.kr(-1,1,hasFreq)) +
Pan2.ar(sine4, TRand.kr(-1,1,hasFreq));

                Out.ar(outbus, sines*0.25*sineVol*env*pauseEnv);
                        Out.ar(outbus,([fx1,fx2])*distVol*env*pauseEnv);
        }).writeDefFile;

        SynthDef("bitCrusher4_Mod",{ arg inbus, outbus, sr1Bus, sr2Bus, distVolBus,
sineVolBus, gate = 1, pauseGate = 1;
                var input, fx1, fx2, fx3, fx4, sines, sine0, sine1, sine2, sine3, sine4, fund,
skip, freq, hasFreq, env, pauseEnv;
                var sr1, sr2, distVol, sineVol;

                sr1 = In.kr(sr1Bus);
                        sr2 = In.kr(sr2Bus);
                        distVol = In.kr(distVolBus);
                        sineVol = In.kr(sineVolBus);
                        env = EnvGen.kr(Env.asr(0.1,1,0.1), gate, doneAction:2);

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);
```

```
                input=In.ar(inbus, 1);
                        fx1=Latch.ar(input.round(0.125),Impulse.ar(sr1));
                        fx2=Latch.ar(input.round(0.1),Impulse.ar(sr2));
                        fx3=Latch.ar(input.round(0.1125),Impulse.ar(sr1-50));
                        fx4=Latch.ar(input.round(0.15),Impulse.ar(sr2+50));

                #freq, hasFreq = Pitch.kr(fx1, ampThreshold: 0.02, median: 7);

                fund = TIRand.kr(7, 50, hasFreq);
                        skip = TIRand.kr(1, 5, hasFreq);

                sine0 = SinOsc.ar(freq, 0, hasFreq);
                        sine1 = SinOsc.ar(freq+(fund*skip), 0, hasFreq);
                        sine2 = SinOsc.ar(freq+(2*fund*skip), 0, hasFreq);
                        sine3 = SinOsc.ar(freq+(3*fund*skip), 0, hasFreq);
                        sine4 = SinOsc.ar(freq+(4*fund*skip), 0, hasFreq);

                sines = Pan4.ar(sine0, TRand.kr(-1,1,hasFreq), TRand.kr(-1,1,hasFreq)) +
Pan4.ar(sine1, TRand.kr(-1,1,hasFreq), TRand.kr(-1,1,hasFreq)) + Pan4.ar(sine2, TRand.kr(-
1,1,hasFreq), TRand.kr(-1,1,hasFreq)) + Pan4.ar(sine3, TRand.kr(-1,1,hasFreq), TRand.kr(-
1,1,hasFreq)) + Pan4.ar(sine4, TRand.kr(-1,1,hasFreq), TRand.kr(-1,1,hasFreq));

                Out.ar(outbus, sines*0.25*sineVol*env*pauseEnv);
                        Out.ar(outbus,([fx1,fx2,fx3,fx4])*distVol*env*pauseEnv);
        }).writeDefFile;

        SynthDef("bitCrusher8_Mod",{ arg inbus, outbus, sr1Bus, sr2Bus, distVolBus,
sineVolBus, gate = 1, pauseGate = 1;
                var input, fx1, fx2, fx3, fx4, fx5, fx6, fx7, fx8, sines, sine0, sine1, sine2,
sine3, sine4, fund, skip, freq, hasFreq, env, pauseEnv;
                var sr1, sr2, distVol, sineVol;

                sr1 = In.kr(sr1Bus);
                        sr2 = In.kr(sr2Bus);
                        distVol = In.kr(distVolBus);
                        sineVol = In.kr(sineVolBus);
                        env = EnvGen.kr(Env.asr(0.1,1,0.1), gate, doneAction:2);

                pauseEnv = EnvGen.kr(Env.asr(0,1,0), pauseGate, doneAction:1);

                input=In.ar(inbus, 1);
                        fx1=Latch.ar(input.round(0.125),Impulse.ar(sr1));
                        fx2=Latch.ar(input.round(0.1),Impulse.ar(sr2));
                        fx3=Latch.ar(input.round(0.1125),Impulse.ar(sr1-25));
                        fx4=Latch.ar(input.round(0.15),Impulse.ar(sr2+25));
                        fx5=Latch.ar(input.round(0.125),Impulse.ar(sr1-50));
```

```
                fx6=Latch.ar(input.round(0.1),Impulse.ar(sr2+50));
                fx7=Latch.ar(input.round(0.1125),Impulse.ar(sr1-75));
                fx8=Latch.ar(input.round(0.15),Impulse.ar(sr2+75));

            #freq, hasFreq = Pitch.kr(fx1, ampThreshold: 0.02, median: 7);

            fund = TIRand.kr(7, 50, hasFreq);
                skip = TIRand.kr(1, 5, hasFreq);

            sine0 = SinOsc.ar(freq, 0, hasFreq);
                sine1 = SinOsc.ar(freq+(fund*skip), 0, hasFreq);
                sine2 = SinOsc.ar(freq+(2*fund*skip), 0, hasFreq);
                sine3 = SinOsc.ar(freq+(3*fund*skip), 0, hasFreq);
                sine4 = SinOsc.ar(freq+(4*fund*skip), 0, hasFreq);

            sines = PanAz.ar(8, sine0, TRand.kr(-1,1,hasFreq)) +
                        PanAz.ar(8, sine1, TRand.kr(-1,1,hasFreq)) +
                        PanAz.ar(8, sine2, TRand.kr(-1,1,hasFreq)) +
                        PanAz.ar(8, sine3, TRand.kr(-1,1,hasFreq)) +
                        PanAz.ar(8, sine4, TRand.kr(-1,1,hasFreq));

            Out.ar(outbus, sines*0.25*sineVol*env*pauseEnv);
                Out.ar(outbus,
([fx1,fx2,fx3,fx4,fx5,fx6,fx7,fx8])*distVol*env*pauseEnv);
                }).writeDefFile;
            }.defer(1);
        }

        init {
            this.makeWindow("BitCrusher",Rect(718, 645, 135, 270));

            mixerToSynthBusses = List.new;
                mixerToSynthBusses.add(Bus.audio(group.server));

            sr1Bus = Bus.control(group.server);
                sr2Bus = Bus.control(group.server);
                distVolBus = Bus.control(group.server);
                sineVolBus = Bus.control(group.server);

            this.initControlLists(4);

            synths = List.new;
                synths.add(Synth("bitCrusher2_Mod", [\inbus, mixerToSynthBusses[0].index, \outbus,
outBus, \sr1Bus, sr1Bus, \sr2Bus, sr2Bus, \distVolBus, distVolBus, \sineVolBus, sineVolBus], group));

            controls = List.new;
```

```
        assignButtons = List.new;

    controls.add(EZKnob.new(win,Rect(5, 0, 60, 100), "blipVol", ControlSpec(0,1,'amp'),
            {|v|
             sineVolBus.set(v.value)
        }, 0, true));
        this.addDoubleAssignButton(Rect(5, 100, 60, 20), 0, 1,\horz);


    controls.add(EZSlider.new(win,Rect(5, 120, 60, 120), "bitVol", ControlSpec(0,1,'amp'),
            {|v|
             distVolBus.set(v.value)
        }, 0, layout:\vert));

    this.addDoubleAssignButton(Rect(5, 240, 60, 20),1,1,\horz);

    controls.add(EZKnob.new(win,Rect(70, 0, 60, 100), "sr", ControlSpec(0,127,'linear'),
            {|v|
             sr1Bus.set(v.value*40+400);
                sr2Bus.set(v.value*40+300);
            }, 0));
    this.addDoubleAssignButton(Rect(70, 100, 60, 20),2,1,\horz);

    controls.add(EZSlider.new(win,Rect(70, 120, 60, 120), "sr", ControlSpec(0,127,'linear'),
            {|v|
             sr1Bus.set(v.value*40+400);
                sr2Bus.set(v.value*40+300);
        }, 0, layout:\vert));
        this.addDoubleAssignButton(Rect(70, 240, 60, 20),3,1,\horz);


    //multichannel button
        controls.add(SCButton(win,Rect(5, 265, 60, 20))
            .states_([["2", Color.black, Color.white],["4", Color.black, Color.white],["8",
Color.black, Color.white]])
            .action_{|butt|
                synths[0].set(\gate, 0);
             switch(butt.value,
                    0, {
                        numChannels = 2;
                    synths.put(0, Synth("bitCrusher2_Mod", [\inbus,
mixerToSynthBusses[0].index, \outbus, outBus, \sr1Bus, sr1Bus, \sr2Bus, sr2Bus, \distVolBus,
distVolBus, \sineVolBus, sineVolBus], group));
                },
                    1, {
                        numChannels = 4;
```

```
                                        synths.put(0, Synth("bitCrusher4_Mod", [\inbus,
mixerToSynthBusses[0].index, \outbus, outBus, \sr1Bus, sr1Bus, \sr2Bus, sr2Bus, \distVolBus,
distVolBus, \sineVolBus, sineVolBus], group));
                                    },
                                            2, {
                                                    numChannels = 8;
                                        synths.put(0, Synth("bitCrusher8_Mod", [\inbus,
mixerToSynthBusses[0].index, \outbus, outBus, \sr1Bus, sr1Bus, \sr2Bus, sr2Bus, \distVolBus,
distVolBus, \sineVolBus, sineVolBus], group));
                                    }
                                    )
                            };
                    );
            }

        setManta {arg data;

            }
}
```