Development of Compositional Tools in Max/MSP

Michael Sweeton

A portfolio of original software and commentary submitted to the University of Huddersfield in partial fulfilment of the requirements for the MA by Research.

January 2011

Abstract

The aim of the research is to build a suite of software tools that use a unique combination of techniques to generate specific audio material. Emphasis will be put on the methods used to create the desired sounds, how to achieve sophisticated control over relevant parameters quickly and easily, and how to provide functionality that cannot be replicated with existing software. The driving force behind the creation of each piece of software is the type of sound created and how this can be used within my own compositions. The development of application will begin with a specific sound or compositional method in mind, which will be worked towards through a process of refining various audio and control processes. I have become increasingly interested in drone music and two of the three applications will be ideally suited to creating audio material for this genre. The final application stems from my interest in the cut up style compositions of Akufen and is designed to simplify the creation of samples in this style.

An important part of the research is the development of the graphical user interfaces for each piece of software. I want to create not just Max/MSP patches, but standalone applications that I find enjoyable and easy to use. I feel it is important to have continuity throughout the suite and will demonstrate this by using a similar design aesthetic in each application. Through my research of other applications created in Max, I recognise that the design of the user interface has a large impact on whether I enjoy using the software. With this in mind I intend to organise the interfaces in a logical way, and to keep them as clear as possible.

Another main research aim is using the computer to cut down the time it takes to achieve my compositional goals. Through actively using the applications as they are being developed, I will be able to identify areas that I can refine, which will save time when composing. One area I will investigate is the application of random number procedures and how they can be used to provide interesting variation of the sound within bounded parameters. Each of the final applications will demonstrate their role in my compositional process.

I will use Max/MSP as the development environment as I have had extensive experience with it during my undergraduate degree, and also for independent projects. As a prototyping environment it has an advantage over traditional code based programming because any changes are instantly usable, giving the environment a ‘live’ feel. The diagrammatical nature of the patching window also lends itself to experimentation and quick changes, which are not as feasible in text based programming languages. When working towards a specific sound, this ease of experimentation is a vital characteristic of the software and will undoubtedly save time during the development process.

Table of Contents:

Chapter 1 – Stand Alone Max/MSP Applications

1.1 Ambient 5

1.2 Gleetchlab 8

1.3 Forester 12

1.4 Thoughts on Stand Alone Max/MSP Applications 14

Chapter 2 – My Software

2.1 Feedback 16

2.2 Generative 36

2.3 Sampler..........49

Chapter 3 – Conclusions

3.1 Conclusions..........60

Bibliography, Discography, & Online References..........62

Enclosures:

A DVD containing the compiled applications, the Max/MSP patches, three demonstration videos, and the musical examples.

Standalone Max/MSP Applications

To begin my research, I chose to see how other developers had approached the task of building standalone applications in Max/MSP. I found three separate applications, each with a unique focus and sound output. I reviewed the user interface design and how this made the software either easy, or difficult to use. I was also interested to see how much control the user was given over the sound output and how this affected their interaction.

Ambient

> A unique ambient soundscape generator. AMBIENT is capable of producing a vast array of ambient textures, from the bizarre to the beautiful. AMBIENT processes any sound you care to load into it. The possibilities are endless.[1]

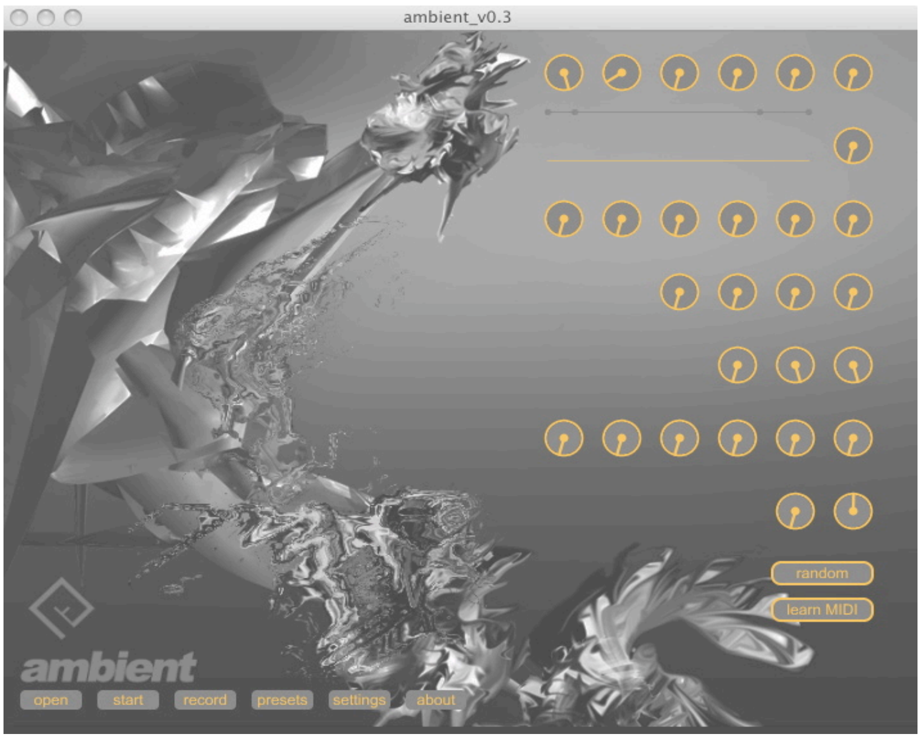


Figure 1 Interface for the software *Ambient*, created by Christopher Hipgrave.

Interface Design and Functionality

Created by Christopher Hipgrave, *Ambient's* interface is minimal in design with a digital artwork forming the background on which the controls are laid out. The

[1] http://www.audiobulb.com/create/Ambient/Ambient-PR.pdf (accessed 2.6.11)

bottom left section of the interface is clear and the information associated with the buttons is well displayed and in keeping with the rest of the design. Any dialogue boxes are framed within the main software window, preventing too many windows being open at the same time and giving the software a compact feel. I felt that the minimal design was taken too far with the exclusion of clear labelling for the various controllable parameters to the right of the window. The parameters associated with the controls are only revealed by hovering over them with the mouse, but as a first time user I found this made learning to use the software more difficult than if clear labelling had been used. There is also a randomizing button that generates a list of random values for all controllable parameters. I found this function a very useful addition to the software as it produced many unexpected and interesting results that may have not been discovered manually.

Sound Output Control

Ambient produces its sound by granular synthesis of a loaded audio file. There are controls for altering the length, pitch, and size of the grains. There are also controls to randomise these parameters. Whilst the controls do alter the sound, there is a feeling of detachment that stems from the constant granular synthesis of the loaded file. I found that by altering the parameters with the mouse, it seemed as if I was guiding the application rather than directly controlling it. The randomisation parameters lessened this effect, but a familiar pattern to the granulation emerged after around 10 to 15 seconds.

The application also has MIDI input. Using a MIDI controller added a new dimension to the software and gave me the opportunity to play the application more as an instrument, as opposed to using the mouse to alter parameters sequentially. Whilst using the software I noticed that on occasion, the sound would stop for a few seconds before starting up again. I am unsure what caused this but it could become problematic in a live situation.

Conclusion

The application presents itself as highly configurable, with many controls available to the user. However, I feel that Christopher has retained enough control over the internal processes for the application to have a specific sound. From a users perspective I felt that I wanted more control over the granulation, particularly the frequency at which the grains are triggered. For novice users this may lessen the appeal, as adding controls that affect the applications parameters at a deeper level would detract from the simplicity of creating interesting sounds with minimal interaction.

I also noted that without direct interaction from the user the sound output was fairly static and became repetitive after a short period of time. I thought that the random function of the application was the most useful feature and could be used to find new parametrical starting points that may not have been discovered when using the controls manually.

Unfortunately the software was let down by the minimal design, which I felt was an example of aesthetics over functionality, making the software more difficult to use than if clear labelling had been used.

Gleetchlab 3

A realtime modular composition and processing software for sound design and experimental music like glitch, IDM, electroacoustic, ambient...[2]

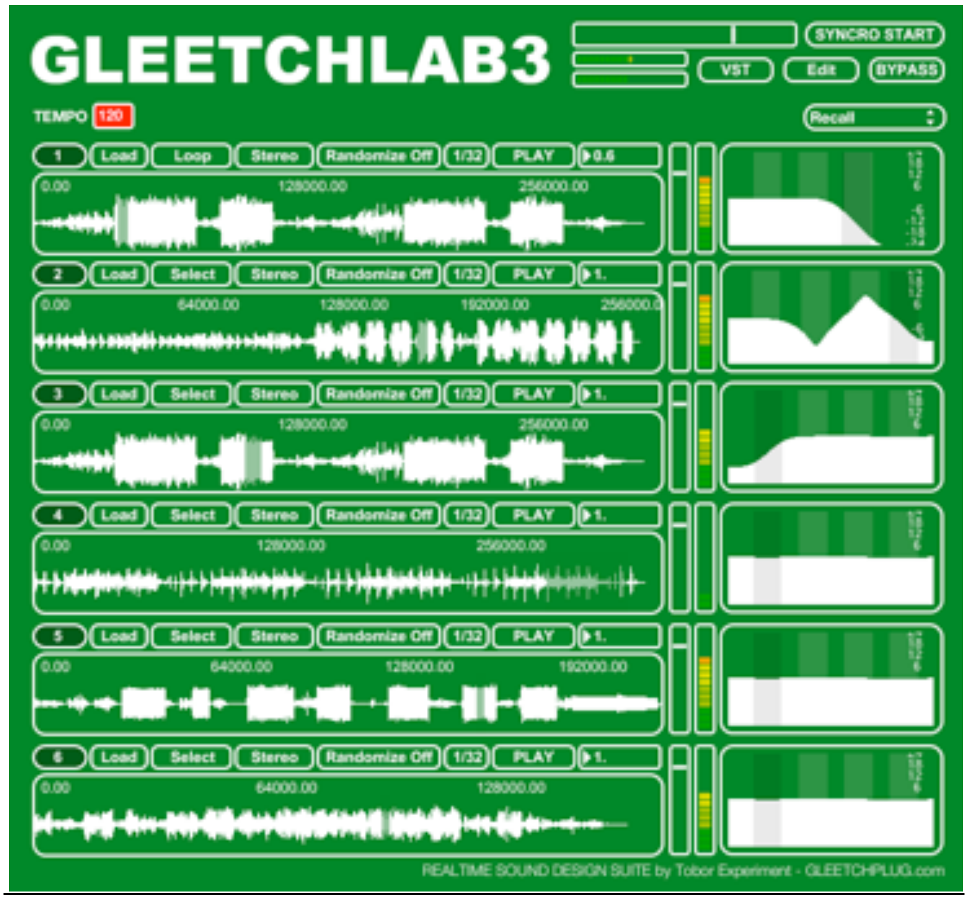


Figure 2 The main interface for the software '*Gleetchlab 3*' created by Giorgio Sancristoforo.

Interface Design and Functionality

Created by Giorgio Sancristoforo, *Gleetchlab's* main window is where the audio to be used with the application is loaded, viewed, and operated on. Up to six audio files are used as source material, which can be played at different speeds and routed to various effects. Unlike *Ambient* which requires minimal interaction to generate sound, *Gleetchlab* requires its users to connect sound sources and effects via a matrix control. As a user I found it cumbersome to use the matrix,

2 http://www.gleetchplug.com/gleetchplug/Software.html (accessed 2.6.11)

which has just under 600 possible connections. It took me some time to become comfortable routing the various sound sources and effects. The matrix forms an integral part of the software and I feel that the developer could have spent more time making it clearer and easier to use. It would have been improved vastly by some form of hint next to the cursor, detailing the connection, or a more obvious graphical display that makes greater use of colours or lines to differentiate between effects and sources.

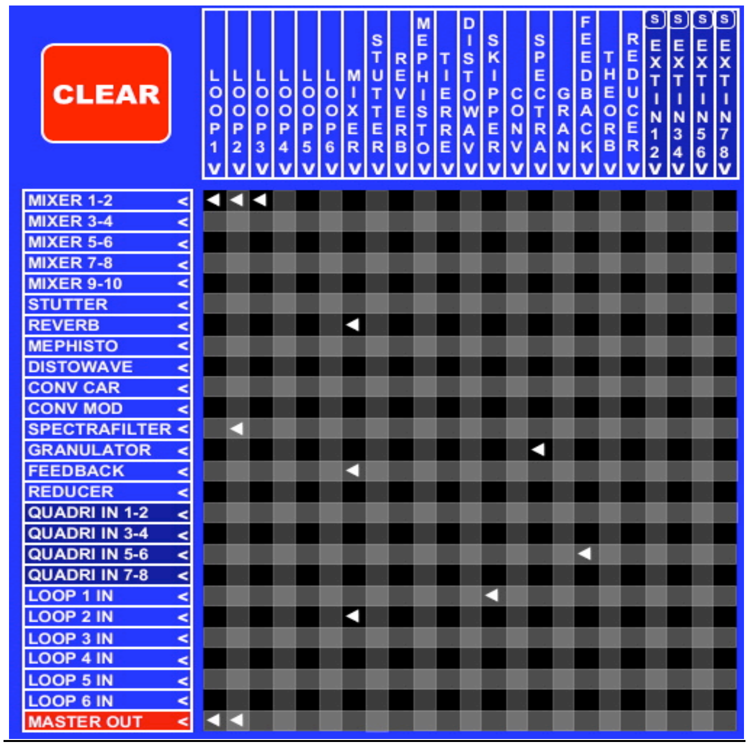


Figure 2.1 *Gleetchlab's* matrix for routing sources and effects.

The software also generates a random background colour on load. This can be changed via a colour palette located in the main window. I thought this was an interesting feature at first, but after opening the software a few times, I felt that it took away some continuity from the interface and I would have preferred the developer to have used a single colour.

Strangely for such a configurable piece of software, there is no save function, meaning that the user has to start from scratch every time the software is loaded. The developer explains the reason for this on his website:

> 'In my analog synthesizer days there were no save functions at all but pencil and paper. If you approach each time a reset machine, you are forced to do something new and with little time and patience, you can master the software much better.'[3]

Whilst he raises a valid point about learning to use the software, there are many situations where a save function would save a lot of time. For example, a live performance may require complex routing of the matrix and six different audio files that would be time consuming to set up. To change quickly from one configuration to another would allow the software to be more flexible in performance situations and would generally save time when wanting to recreate a specific configuration.

The application also has MIDI input, which allows the user to control various parameters in the software. The input is limited to fixed channel and controller numbers, meaning any external MIDI device needs to be programmed specifically to use with the software. Having used *Ambient*, with its MIDI learn feature, this seemed a laborious way of sending MIDI information to the application.

Sound Output Control

As the audio files loop continually most of the user interaction is by altering the parameters of the various effects. The controls for these are generally straightforward and the effects are of a good quality. The pitch of the loops can be altered and there is also a function for randomising the loop area. I found that the loop randomisation function was unreliable, as it does not always show the

[3] http://www.gleetchplug.com/gleetchplug/Software.html (accessed 2.6.11)

portion of the file being looped. I also noted that there was no option for fading between settings when changing the routing of the matrix. This meant that any changes were abrupt, and ruled out the possibility of gradual fades from one effect to the other.

Conclusion

The effects are the best feature of the program and it would be interesting to see these incorporated in to a different approach with a more flexible and varied sound file playback system. The most interesting part of the software is its flexibility, but this is negated by a routing system that is difficult to use, and the sound source, which is essentially a looping audio file that requires combinations of effects to offer an interesting sound output. I think that a large amount of the sound output could be replicated in a DAW, but this would focus more on a studio style of working as opposed to a live performance tool.

To use the application to its full potential I feel that users would have to spend a lot of time using the matrix and be willing to program a MIDI device specifically to use with the application. I also feel that the lack of a focus is detrimental to the application. With *Ambient* the focus is on granular synthesis, and it is clear that the application is for creating granular textures. As a user I found it difficult to gauge what the purpose of the application was.

Forester

Magical sound creation For Mac & PC[4]

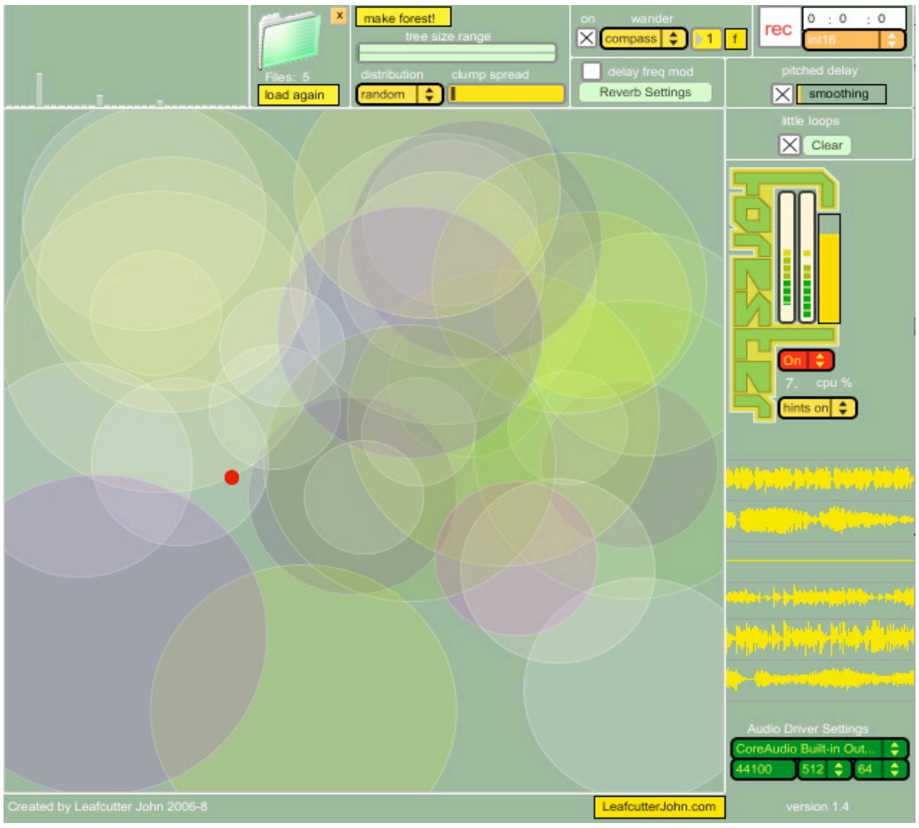


Figure 3 The user interface for *Forester*, created by John Burton (Leafcutter John).

Interface Design and Functionality

Created by John Burton, *Forester* has a unique design that employs circles to represent a forest, within which a point is moved. When the point moves inside a circle, playback of one of six user specified audio files begins. The playback can be pitched, looped, delayed, or played back normally with reverb. Audio files are loaded from a folder, and *Forester* takes six, eight second samples from each file to create the source material used with the application. The point can be dragged around the interface manually, or it can be set to 'wander' in either 'forage' or 'compass' mode. The compass mode allows the point to travel in straight lines

[4] http://leafcutterjohn.com/?page_id=14 (accessed 2.6.11)

until it comes in to contact with the edge of the window that the forest is contained in. The 'forage' mode has a more random movement and does not follow a set path. Both modes have a speed control that alters how fast the point travels in the interface.

I thought that the visual representation of the various playback modes was a great original idea and made the application more engaging than if a visual representation had not been used.

Sound Output Control

The amount of control given to the user over the sound output is minimal. Using the mouse to change the location of the point allows for gestures to be represented, and the speed at which the cursor enters the circle can sometimes have an effect on the playback. For example when the audio is pitchshifted, I found that moving the point in to the circle at a faster pace made the pitch go higher. The greatest control the user has over the sound comes from which audio files they select to be used with the application.

Conclusion

As a composer I felt that while I enjoyed using the program, I did not have enough control over the various sound processes. This was primarily because the application is designed to produce an interesting and varied sound output with minimal user interaction. I do not think the application is best viewed as a compositional tool, but rather a form of interactive installation. In this sense I feel the developer has been successful, and has created a unique application that is enjoyable to use.

Thoughts on Stand Alone Max/MSP Applications

I found that *Ambient* was the most useful application that I researched. It was a good example of using Max/MSP to create a compact and stable application, with a rich and varied sound output. Like many Max/MSP projects, the applications that I researched were created by one developer. This brings a certain level of autonomy to the development process and results in software that a team of developers creating commercial applications would be less likely to create. The advantage of this kind of software to the user is that they are often using a tool that the developer has created out of a compositional need, implying that the software does not already exist. The disadvantage of this is that some features can often be overlooked or negated. This was evident with *Gleetchlab*, whose developer continues to stand by his decision to exclude a preset function[5], when in such a configurable environment it would be a useful feature. I wanted to avoid these problems during my research and so maintained a constant dialogue with a small group of alpha testers throughout the project who suggested improvements or new features to the software[6]. As a developer I wanted to create software that while performing a specific task well, did not overlook any logical features or controls that may be of use to the user.

Reviewing the applications also made me more aware of what aspects I wanted to include in my own software. I enjoyed the random aspects of *Ambient* and *Forester* and felt that there was a middle ground between the almost completely random aspects of *Forester* and the variation used in *Ambient* that I wanted to

[5] http://splendidbeats.com/gleetchlab-30/ (accessed 2.6.11)

[6] Most notably Monty Adkins, the supervisor of my research.

explore. This aspect, combined with an idea from my undergraduate studies formed the basis of my second application ‘Generative’.

I recognised that it was important to spend time making the applications easy to use. The majority of the problems that I had with the applications I reviewed were related to the user interfaces, which made some parts of the applications difficult to use. During the development process I was conscious of this and tried to keep the user interfaces as clear as possible. In relation to this I also spent time developing help sections that were integrated in to the applications.

Principles of Graphical User Interface design

To finish my preliminary research I decided to look at some fundamental principles of graphical user interface design in order to guide me when developing my own interfaces. Wendy L. Martinez[7] outlines 3 basic interface design principles:

‘1. Focus on the users and their tasks, not the technology’

With this in mind I only wanted to include controls that were relevant in a compositional sense when interacting with the application. As a developer and composer I was in a unique position to make informed decisions about these controls and each time I added a control to the interface I reviewed whether it was necessary and how useful it would be during my compositional process.

‘2. Consider function first, presentation later’

It was important that the applications were to be designed first and foremost with ease of use mind, with this taking precedence over how the application

[7] Martinez, W.L., *Graphical User Interfaces*, Wiley Interdisciplinary Reviews (2011)

looked. As I had encountered problems during my review of other Max/MSP applications that were related to the user interface, I wanted to make sure that I did not make the same mistakes when designing the interfaces of my own applications.

‘3. Conform to the user’s view of the task, and do not make it more complicated’

This principle was slightly more difficult to adhere to, as the interfaces were new and not similar to many other applications. However, I did use this principle as a point of reference, which informed my decisions when making the applications as clear and easy to use as possible.

2.1 ‘Feedback’

Foundations and development process

During my undergraduate degree I became increasingly interested in drone music. For one of my undergraduate projects I used sine waves and reverb to create drones that I used within my compositions. I would export the audio and then layer the files manually in a DAW. This process was fairly laborious and did not allow for much spontaneity during the compositional process. I wanted to create an application that allowed for a more fluid approach when creating drone-based material. Initially I envisaged a system that used a granular playback method. The user would record a note or sound from an external source and this would be recorded in to a buffer within which the playback location would continually change. More notes or sounds could be recorded allowing sounds to be layered on top of each other. Whilst working on this prototype I was also revising my knowledge of Max/MSP. I came across a tutorial based around controlling feedback of a delay[7]. When experimenting with the settings in the tutorial I found that a loop could be created that played indefinitely. The level of the feedback was controlled by a compressor, which regulated the amplitude of the audio that was routed back in to the delay. The compressor ensured that instead of the feedback overloading or dying out, it stayed at a constant level.

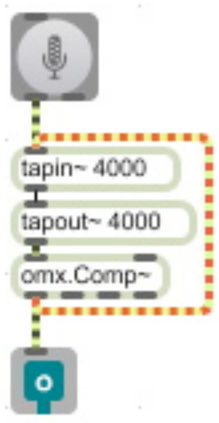


Figure 4. A simplified version of the technique. The input from 'ezadc~' is fed in to the delay objects 'tapin~' and 'tapout~' and then in to the compressor 'omx.Comp~'. From the compressor the delay is connected back in to the input of the delay line causing the feedback effect.

What I found appealing about this approach was the ability to add to the loop simply by gating the input signal. I quickly decided that this method was much simpler than the granular playback method, and offered the same ability to create drones, and additionally loops depending on the length of the delay. I found this technique to be interesting for a variety of reasons. Firstly, when dealing with sample based looping it is relatively complicated to program the recording and playback of each loop in relation to a master loop or BPM. When using delay based looping the loop length is already defined making it easier to define delay times relative to a global BPM. Recording and playback can occur at the same time and audio can easily be added to the loop without overwriting the existing

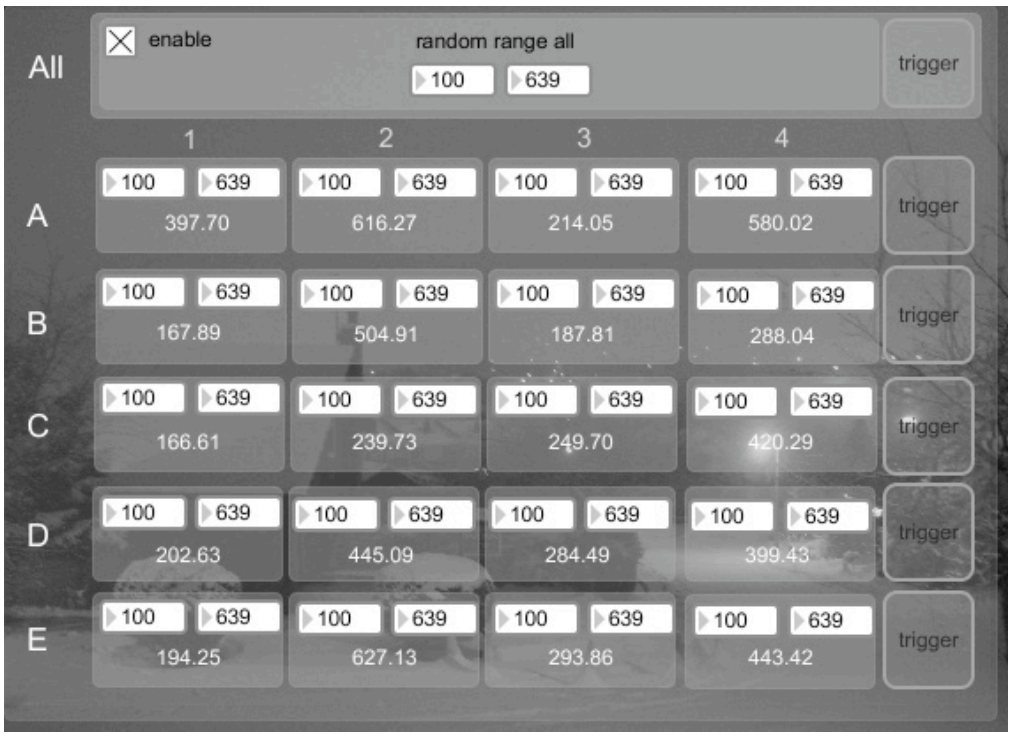
[7] MSP Compression Tutorial 10

audio. Also, when using sample based looping the playback has no inherent variation. When using delay based looping the high frequency content gradually diminishes with each successive loop until it reaches a plateau. This effect is quite subtle but adds a small variation that in comparison to sample based looping gives the sound a more organic nature. I began working with multiple copies of this basic technique and quickly found that I could create dense textures by setting short delay times combined with compressor settings that reduced distortions, producing a clear reproduction of the input audio.

I decided to create 20 delay units that I grouped into banks of four. Each delay had a millisecond time control and also a measure control. I implemented a global BPM which when altered would provide the correct times in milliseconds when a measure was specified. I often used this feature when I wanted to work with loop-based motifs, as it was an easy way to specify delay times relative to the global BPM. I also included a variation control that randomised the delay time by using the specified time as a centre point for the randomisation. I linked this to a global randomisation control that saved time when I wanted to alter all of the variation parameters. This variation control created an effect similar to tape saturation and was useful to alter the character of the delays individually or as a whole.

I created a gate for the input audio that was connected to each delay. In addition to this I created a master gate, which simplified the process of recording in to more than one bank, or all the delay banks simultaneously. For the master gate I built in a ramp that fades the signal in and out when the record enable toggle is used. I created a dialog so that I could edit the duration and curve of the ramp, allowing the input audio to fade in slowly, or more abruptly.

Through using the software I noted that I was often using small delay times for each delay to create the dense textures I wanted from a simple input source. Setting 20 delay times manually was time consuming, so I created a function that allowed me to generate a random time in milliseconds for each delay. Like the random function in *Ambient*, this allowed me to alter many parameters at once. The random time was between ranges that I could specify, meaning that I could influence whether the times were shorter or longer.


All ☒ enable random range all 100 639 trigger

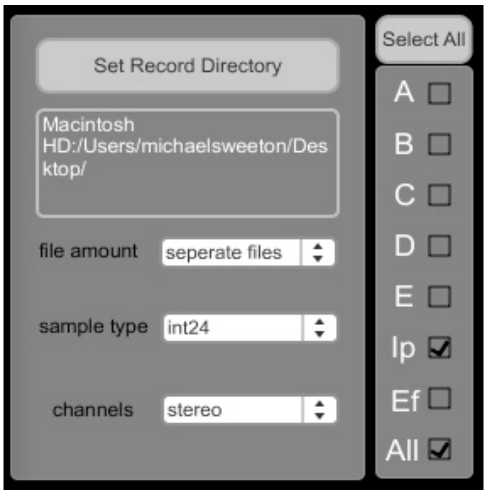
	1	2	3	4	
A	100 639 397.70	100 639 616.27	100 639 214.05	100 639 580.02	trigger
B	100 639 167.89	100 639 504.91	100 639 187.81	100 639 288.04	trigger
C	100 639 166.61	100 639 239.73	100 639 249.70	100 639 420.29	trigger
D	100 639 202.63	100 639 445.09	100 639 284.49	100 639 399.43	trigger
E	100 639 194.25	100 639 627.13	100 639 293.86	100 639 443.42	trigger

Figure 4.1 The random dialog allowed me to generate random delay times for each delay in the different delay banks.

As well as saving time, I felt that this was a significant development in relation to the sound of the application as it facilitated the creation of a compositionally rich sound. By generating 20 random delay times I opened up a range of combinations that would have been laborious to explore using manual processes.

I then created volume controls for each of the 20 delays and also a control to scale the amplitude of each bank as a whole. I also added a panning control for each delay bank. By using these controls in conjunction with the random delay time values I was able to create dense drones by only playing one or two notes in to the delay banks.

At this stage in the development process I was creating a lot of audio material that I began to use in my compositions. I was recording a stereo output of all the delay banks combined with the input audio. I felt that while I enjoyed using the software in this real-time manner, it would be beneficial to explore the option of recording various parts of the audio produced by the application for more flexible post-production options. I created a recording function that allowed me to record all of the delay banks, each delay bank individually, and the input audio.



Macintosh HD:/Users/michaelsweeton/Desktop/

file amount: seperate files

sample type: int24

channels: stereo

A ☐ B ☐ C ☐ D ☐ E ☐ Ip ☑ Ef ☐ All ☑

Figure 4.2 The record function allowed me to record a variety of sounds made by the application with varying bit depths and a choice of mono or stereo file types.

My first composition to utilise this feature was 'Amb 03' which was created by recording each delay bank and the input audio separately. I utilised the panning control of each delay bank to enhance the stereo effect and also the random delay time function to provide the settings for the composition. In this composition I recorded into every delay bank simultaneously using the 'record enable' control as a gate for the input audio. I started with one note and began to play motifs over the top of the drone. At intervals I would introduce another note to all of the delay banks, slowly evolving the piece by creating a denser drone. The technique I used initially was only to introduce the end of a note. By not introducing the attack of the guitar into the delay lines I was able to create a smooth sounding drone. At 3:04 in 'Amb 03' I introduce the attack of the note, which is an obvious change from the previous drone based material. It was useful to have the original input audio to work with in Cubase and I employed further delays, reverb, and distortion to it during the post-production process. I enjoyed this process of composing by layering the notes one by one and thought that the drone provided an interesting and varied backdrop to the guitar motifs. I was pleased with how the application had developed so far as it allowed me to create the core of a piece in the application and then use the DAW to finish off the compositional process.

At this point in the development the core of the application was complete and I began to think about what functions were needed to turn the patch in to a more varied compositional tool. Firstly I did not want the application to be limited to an audio input, so I began to devise a file playback system to add another dimension to the tool. I wanted to allow for expressive playback with variable pitches and also a form of automatic playback where I could select parts of the audio to introduce to the delay lines by gating the input. I chose to use the 'waveform~' object as the focal point for the interaction with the file and created three different types of playback.

The first mode 'Play' was the simplest representation of the file. It allowed me to select a portion of the audio file to play normally or in reverse. The selection could be played once or looped. I found the looping tool useful as it allowed a continuous stream of audio that I could use as a source when recording the input audio in to the delay banks. The second playback mode 'Play Free' allowed me to click on a point of the selected waveform as a destination for the playhead[8]. The duration of the playhead's movement from one point to the next was fixed, meaning that the pitch of the playback changed depending on the time specified and which part of the waveform was clicked. This mode was particularly good at providing interesting changes to pitch of the file. Finally I created a granular playback mode. This had typical parameters for granular synthesis and allowed the user to click on the waveform as a starting point for the playback. Within this mode I included various randomisation processes that gave a more organic feel to the sound. These were a randomisation of the start time, and a deviation in pitch derived from a sampled 'noise~' object. I also included a randomisation of the length of the grains and the direction of the playback (either forwards or backwards). With this mode I included an 'envelope type' option which used the mouse's vertical position on the waveform as a way of scaling the amplitude. This meant that I could effectively make gestures with the mouse by clicking and dragging on the waveform, simultaneously altering the playback position and

[8] In the first two pieces of software, I displayed the playback position via the 'waveform~' objects line message. This shows a line, or playhead related to the current playback position.

amplitude of the granular synthesis. The pitch of the playback in the 'Play' and 'Grain' modes could also be altered via the transposition dialog, which gave me the option of a transposition in semitones or as a division of the original pitch.

When working with any audio I often find that filtering or equalisation is required to make sounds more presentable. I built filters in to the two audio inputs with bandpass, lowpass, and highpass filter types. I copied these filters to each delay bank and also to the output.

Whilst I found the input audio usable I felt that on occasion I wanted to add some character to it, particularly when using the guitar input without any processing. I found that when I was working with the audio in Cubase I often added some form of distortion or delay to the input audio. I decided that as Max/MSP has a number of interesting effects available, it would be interesting to try and replicate the effects I was using during post-production inside the application. I decided on this for two reasons: firstly, by using varied inputs to record into the delays, it may be more inspiring and lead to more creative use of the input audio when used in application; and secondly, to see if I could create improvisational compositions that sounded similar to the compositions finished using the DAW.

As the main window for the application had many controls on it already, I decided to create a separate window for the effects section. I created three channels, each with up to three effects consisting of various distortions, a filter, a delay, and a delay ramp. The delay ramp was built specifically to replicate the effect that I had been using in Cubase. This effect changes the time of the delay without changing the pitch. In Cubase the delay changes over time by automation. I wanted to replicate this so I created a delay time ramp using the function object. The X-axis of the ramp indicates the time and the Y-axis the delay time. The minimum and maximum delay time can be set and also the length of the ramp in seconds. I created an option so that the delay would continually trigger or could be triggered manually. The delay ramp was the most complex effect in terms of development as it employed a windowing technique to allow the pitch of the delay to stay the same whilst the delay time is changed. I chose to use part of Emmanuel Jourdan's abstraction 'ej.vdb~[9]' which facilitates this effect.

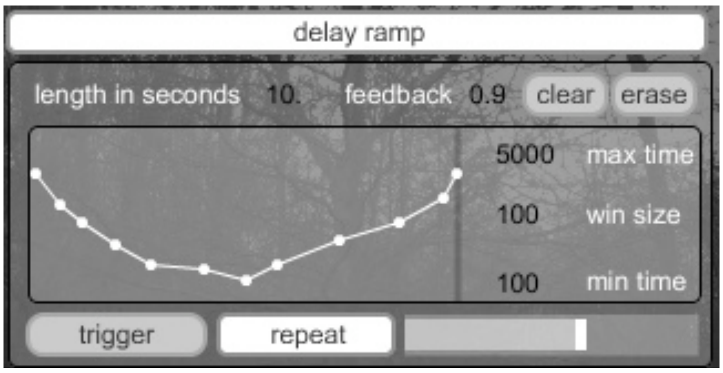


Figure 4.3 The interface for the delay ramp effect. Designed to replicate automation of the delay time I had been using in Cubase.

Creating a smooth distortion is difficult in Max/MSP but the 'tanh~' object provides a usable version. I chose to use 'overdrive~' and 'downsamp~' for harsher and more digital distortions respectively. Finally, I created a static delay which included three delays with controls for the delay time, feedback amount, and dry/wet mix. Each effect also had a master dry/wet mix and a volume control for the incoming and outgoing audio. I chose to include this volume control as the volume of the audio could change quite significantly when using the distortion effects.

9 http://www.e--j.com/?page_id=165 (accessed 2.6.11)

Each of the three channels allow for the input of the file playback, the ADC, or a combination of the two. The output of the effects section allows each channel to be routed to the delays in the main window, or to the output. I included this option as it is possible that a user would want to use one of the inputs solely for the effects section, and one input solely for a dry input in to the delays.

At this stage of development, one of the alpha testers Mark Bokoweic suggested that it would be useful to include controls for the compressor's parameters. Previously I had decided against this as I felt that the settings were an important part of the sound made by the application. However, adding these controls would make the application more flexible allowing the user to overload the looping feedback, or conversely fade away after a certain length of time. I decided that I would add the controls, but in order to retain the concept of the original idea I chose to include an option of setting the compressor's parameters to the default values I had originally specified when first using the patch.

While I was comfortable using the application in a studio setting I recognised that it could be beneficial to add MIDI control for use in a live setting. I created a MIDI dialog that allowed control over many of the adjustable parameters in the patch, this included a MIDI learn function that allowed the user to easily set the controller they wanted to use with the parameter. I also included a preset section to allow the user to recall the various delay, filter, compressor, and effects settings in the software.

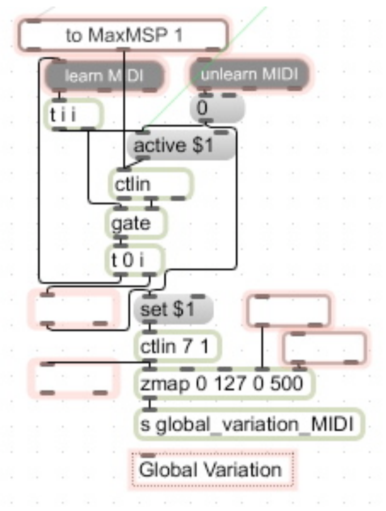
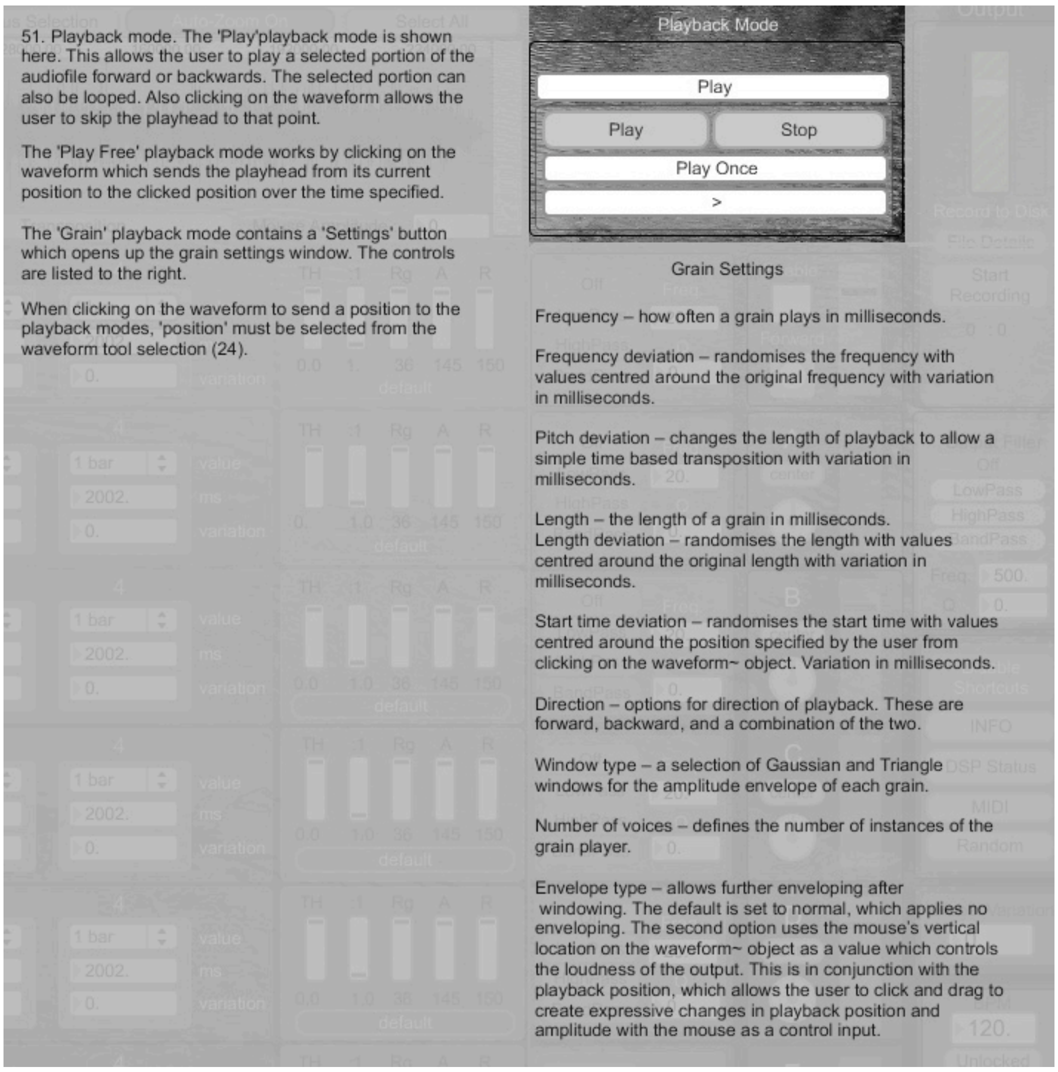


Figure 4.4 The programming involved for the application's MIDI learn function. In this instance the 'Global Variation' parameter is shown. Note that when the first 'ctlin' object is not in use its processing is turned off.

One of the final additions to the application was the help section. As the application has many controls I decided to create this section which allowed the user to mouse over a picture of the main interface, revealing a number assigned to each control and an explanation of the control. I found that this interactive approach was a simple and clear way of showing the user what each control or parameter did, as the user could simply mouse over a control they were unsure about, revealing its function.


51. Playback mode. The 'Play'playback mode is shown here. This allows the user to play a selected portion of the audiofile forward or backwards. The selected portion can also be looped. Also clicking on the waveform allows the user to skip the playhead to that point.

The 'Play Free' playback mode works by clicking on the waveform which sends the playhead from its current position to the clicked position over the time specified.

The 'Grain' playback mode contains a 'Settings' button which opens up the grain settings window. The controls are listed to the right.

When clicking on the waveform to send a position to the playback modes, 'position' must be selected from the waveform tool selection (24).

Grain Settings

Frequency – how often a grain plays in milliseconds.

Frequency deviation – randomises the frequency with values centred around the original frequency with variation in milliseconds.

Pitch deviation – changes the length of playback to allow a simple time based transposition with variation in milliseconds.

Length – the length of a grain in milliseconds.
Length deviation – randomises the length with values centred around the original length with variation in milliseconds.

Start time deviation – randomises the start time with values centred around the position specified by the user from clicking on the waveform~ object. Variation in milliseconds.

Direction – options for direction of playback. These are forward, backward, and a combination of the two.

Window type – a selection of Gaussian and Triangle windows for the amplitude envelope of each grain.

Number of voices – defines the number of instances of the grain player.

Envelope type – allows further enveloping after windowing. The default is set to normal, which applies no enveloping. The second option uses the mouse's vertical location on the waveform~ object as a value which controls the loudness of the output. This is in conjunction with the playback position, which allows the user to click and drag to create expressive changes in playback position and amplitude with the mouse as a control input.

Figure 4.5 A part of the help section. In this view, showing the user information about the various parameters of the different playback modes. The rest of the main interface can be seen behind the grey panels. Using the mouse to hover over different sections of the interface reveals the information related to that section, hiding the others from view.

To complete the application I created a number of keyboard shortcuts that allowed me to toggle the record states of the delay banks, delete or clear the audio from specific banks, and enable or disable recording. While this was a small addition, I feel it saved a lot of time and made the process of creating audio in the application more enjoyable.

<u>*User Interface*</u>

Originally, I wanted to create an interface with few controls. However, the more functions I built in to the software, the more I realised it would be beneficial to have control over them. I wanted to keep the most important controls in the middle of the screen, so I arranged the delay banks in this space. I chose to keep the input section on the left, the main processing in the middle, and the output and recording section on the right. By doing this I wanted to create a form of visual hierarchy whereby the component parts of the application were in separate sections that were easy to remember.

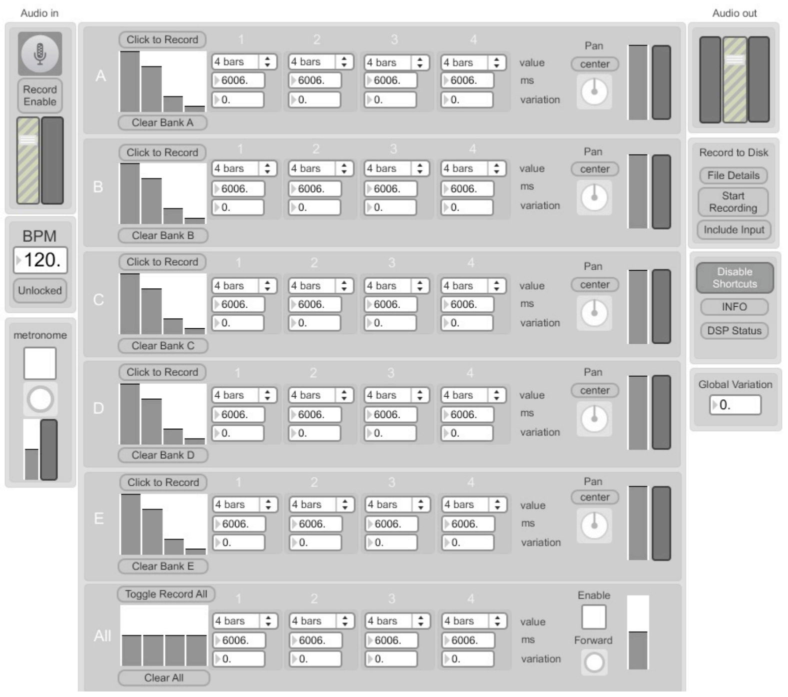


Figure 4.6 An early version of the application.

When adding the file playback capabilities I wanted the waveform to be large enough to operate on, without detracting from the rest of the display. I placed it at the top of the interface and kept it within the vertical boundaries of the delay banks. To make the tool more precise I added the ability to zoom in and out of the selected audio waveform. I also added controls to change the mouse behaviour, allowing the audio to be selected, to move this selection, and also to send the position, which was sent to the file playback system.

At this stage I felt that the background was too simplistic. Taking a cue from the *Ambient* application I experimented with adding images to the background. I chose a suitable picture that I felt was in keeping with the colour scheme and used this as the background. To increase the sense of depth in the interface I chose to alter the 'alpha' setting of Max 5's[10] object colours to make the Max objects on the interface appear translucent.

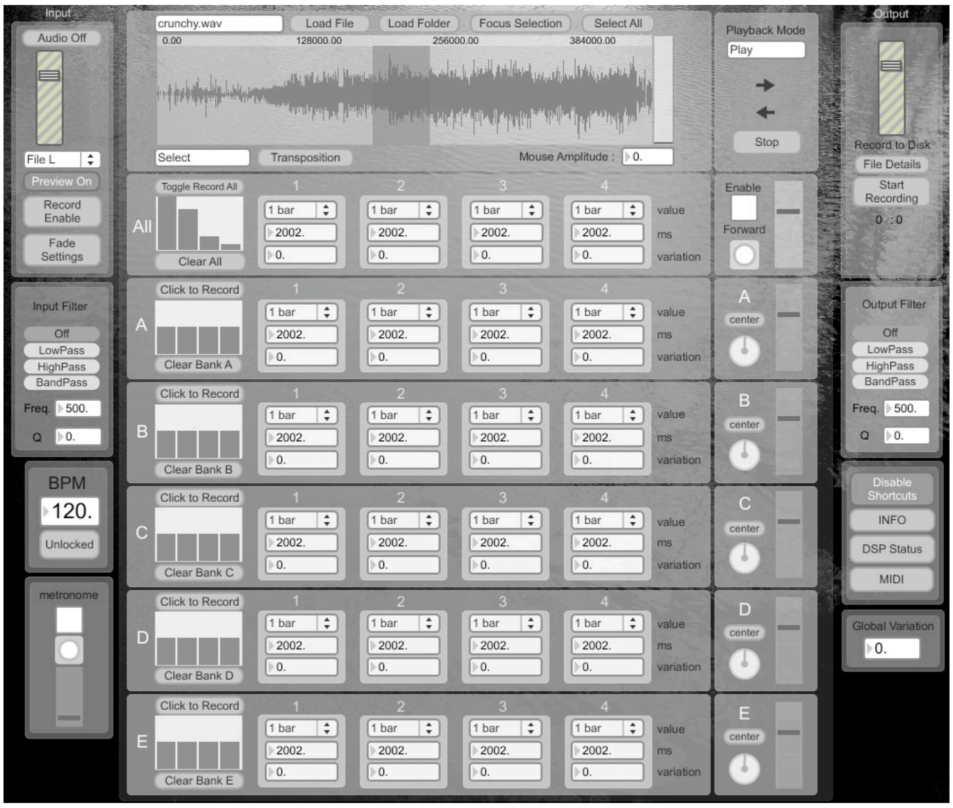


Figure 4.7 Adding file playback capabilities, filters, MIDI, and a background picture.

I also decided to save a small amount of space on the interface, by combining a 'multislider' and 'meter~' object, in to a gain control and display. When I added the filter and compression sections to each delay bank, it seemed logical to

[10] A list of 'what's new' in Max/MSP 5 can be found at: http://www.cycling74.com/docs/max5/vignettes/intro/docnew.html (accessed 2.6.11)

expand the application horizontally. By doing this I was also able to increase the size of the file playback ‘waveform~’ object, and also the playback mode selection area. I felt that visually this made the application easier to use as the extra width allowed for more space between the various interface objects, giving the software a more expansive feel.

As a common action when using the application is recording into the delay banks, I decided to make it as clear as possible when the record mode was enabled. I did this by making the ‘Record Enable’ and delay bank record functions pulse red to indicate that recording was taking place. I also added this feature for the control that recorded the various audio outputs of the application.

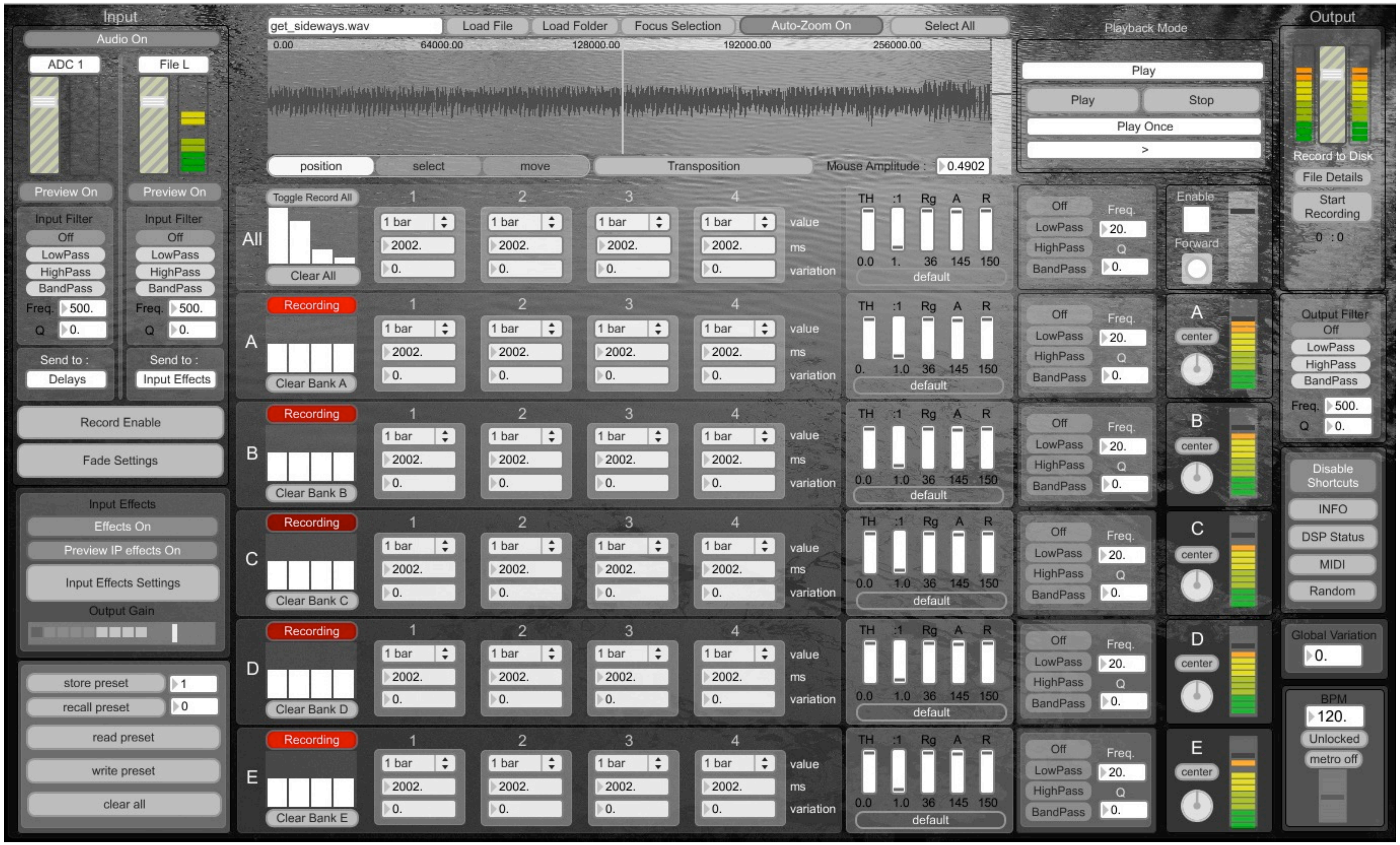


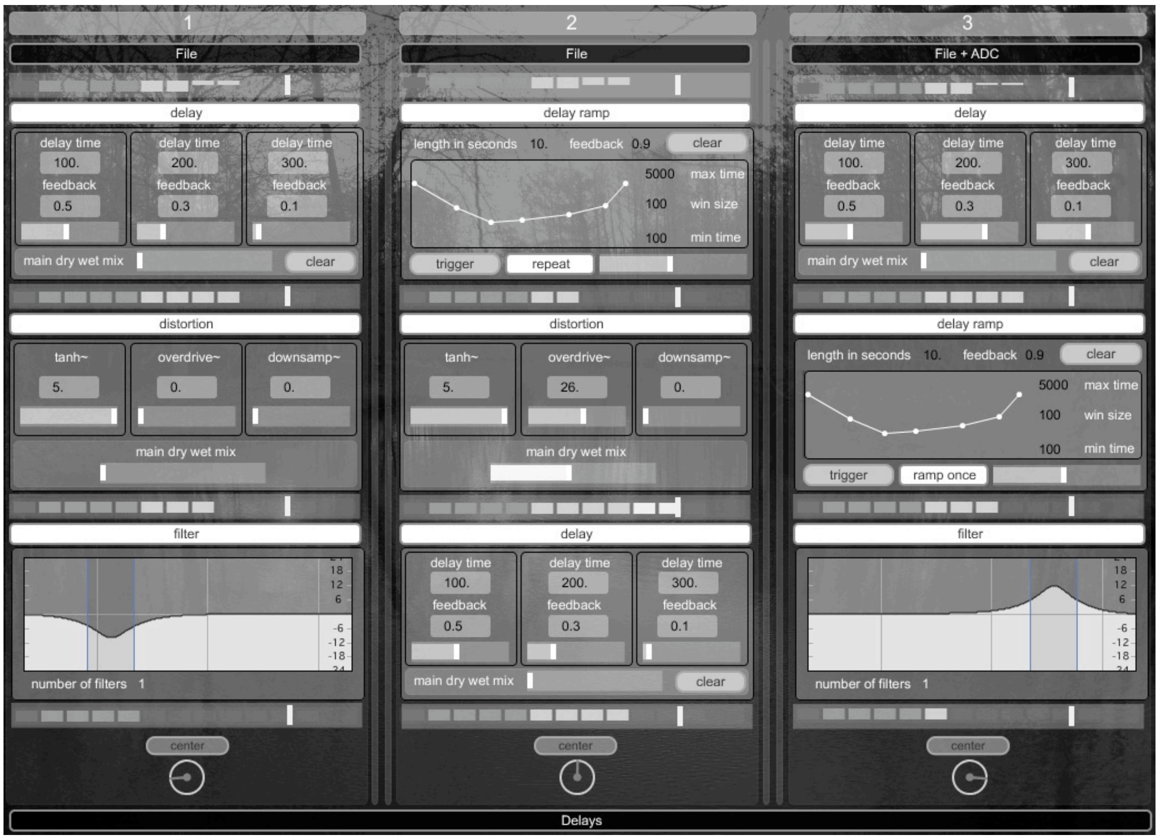
Figure 4.8 The final interface for the main window of the software, with the delay banks recording.

As I was using the ‘tab’ object for the filter selections, I noted that the default behaviour was to highlight any selection. I decided to make the whole object grey when the selection was off. My reasoning for this was to avoid any needless colouration and to highlight only what was relevant in the interface.

During the period where the software was being tested, I noted that it would be useful to build a help section into the software. I decided to use Max/MSP’s ‘ubutton’ object to define areas on a screenshot of the application. When the mouse is moved over the areas, only that area remains displayed, with the rest of the screenshot being almost hidden with the ‘panel’ object. I then used this blank space to detail any information related to the specified area.

When designing the effects section I adopted a similar approach using a different picture for the background and making many of the objects see-through. I also incorporated the gain controls from the main window but positioned them horizontally before and after each effect.

In order to be able to display the varied effects chain, I used the ‘bpatcher’ feature in Max/MSP. This allows the developer to embed one patch within another. In this case I embedded nine copies of an effects patch into the effects section of the software. Within each patch there are the various effects, which can be configured in a way so that when the user selects an effect, the correct effect is shown.



1 File
delay: delay time 100. feedback 0.5 | delay time 200. feedback 0.3 | delay time 300. feedback 0.1 | main dry wet mix
distortion: tanh~ 5. | overdrive~ 0. | downsamp~ 0. | main dry wet mix
filter: number of filters 1

2 File
delay ramp: length in seconds 10. feedback 0.9 | 5000 max time | 100 win size | 100 min time
distortion: tanh~ 5. | overdrive~ 26. | downsamp~ 0. | main dry wet mix
delay: delay time 100. feedback 0.5 | delay time 200. feedback 0.3 | delay time 300. feedback 0.1 | main dry wet mix

3 File + ADC
delay: delay time 100. feedback 0.5 | delay time 200. feedback 0.3 | delay time 300. feedback 0.1 | main dry wet mix
delay ramp: length in seconds 10. feedback 0.9 | 5000 max time | 100 win size | 100 min time
filter: number of filters 1

Figure 4.9 The interface for the effects section of the software, showing all available effects.

Other Musical Examples

In the composition 'Loop1' I use the application to create a loop that has its component parts recorded in to separate delay banks. I use a loop as the backbone of the composition over which field recordings are layered. I utilised the recording function to record the component parts of the loop separately and faded in and out parts of the loop as the composition progressed. The audio recorded from the application was left running in the delay banks for a period of two to three hours. I found that this enhanced the high frequency loss and gave the audio a rounded character that would have been a stark contrast to the precision of sample based looping.

The composition ‘Jigglr’ uses the application in exactly the same way. Only a bass guitar is overdubbed to provide a focal point to the composition. I found that using the application in this way was particularly rewarding from a compositional perspective. I became fascinated with creating short loops that were interesting enough to be played for the duration of a composition and how the parts of these loops could be faded in and out to highlight different parts of the composition.

The compositions ‘RiddimDraft’ and ‘ForJess’ use sounds created in the application as a backdrop to beat based experimentation. In the composition ‘ForJess’ I used both loops and drones, and all of the harmonic sounds except the obvious synth bass were from the application. I employed pitch shifting techniques in Cubase which are particularly evident at 1:20 where the main drone is pitched to provide a chord sequence.

Specific programming issues

As the patch grew in size I became aware of the need to save processing power. The main way of saving processing power in Max/MSP is to turn a functions audio processing off when it is not being used. There are two ways to do this. Firstly, if the process takes place inside ‘poly~’ object, the processing can be stopped by sending the message ‘mute 0 1’ to the object. This turns the processing off for all instances or voices of the patcher. I implemented this behaviour for the granular playback which used the ‘poly~’ object. When the granular playback is not being used, the processing is turned off.

The other way to turn to minimize CPU usage is by using the ‘enable’ message to a subpatch. This has the same effect as the ‘mute’ message to the ‘poly~’ object, but also turns of MIDI processing in the patch. As I had built the effects section in a subpatch, it was obvious to provide an option of turning the processing in the subpatch on and off to the user. Also, each effect was loaded in to a bpatcher, making nine effects available at any given time. I encapsulated the processing of each effect in to a 'poly~' object so that selecting one effect turned off the processing of the other effects. This meant that when all nine effects were used, they were the only processes activated, instead of all thirty-six. I also implemented the same process on each of the filters in the application, creating an abstraction that used the 'poly~' object. This abstraction also faded between the filter states preventing any clicks that I became apparent when working with lower frequencies.

When I was creating the record function, I became aware that I had to set the file name of each channel that I wanted to record. This was time consuming, so I devised a way of setting a file name automatically so that I only needed to set the directory, which the recorded audio was being saved to. This involved using the time and date to form a filename which was sent to the various 'sfrecord~' objects. When this was complete I could record any number of different takes without having to set a filename each time I wanted to record them.

2.2 – Generative

Foundations and development process

Whilst I appreciate methods of composition that allow the user to control many aspects of the creative process, I am also interested in building systems that require minimal user input to generate interesting and diverse results within bounded parameters (unlike the more varied output of *Forester*). It was this interest in random functions within clearly delineated boundaries that lead me to create an installation during my undergraduate degree, which had no emphasis on user interaction with the computer itself. The original project required the participant to speak, sing, or play an instrument into a microphone while various aspects of the audio were analysed. The data from this analysis was used to generate music using oscillators and granular synthesis of the analysed audio. On completion of this project I felt that there were components of the project I wanted to explore more deeply, specifically the granular synthesis of the input audio. Also, I felt that while the installation worked well, there was no real scope to use it as a compositional tool.

Similarly when reviewing the *Ambient* and *Forester* applications the features that impressed me most where the random aspects, and how the sound created by the applications could vary with little or no user input. I knew that with this project I wanted to create an application with similar randomisation, but one that was more applicable to my compositional needs. Whilst I enjoyed the variations in pitch of the *Ambient* application I felt that I wanted to explore a more natural reproduction of the input audio and focus on variations in playback position and other granular parameters.

With the *Feedback* application complete, I had a useful tool for creating drones. However, it produced a specific type of drone that was fairly static without user interaction. With this second application, I was interested in building an application that created evolving drones automatically. I wanted to focus on immersing the user in the sound of the specified audio using granular synthesis. To do this I took the fundamental idea of analysing an audio signal from my undergraduate project and applied it to build a tool that could create a continually evolving soundscape. I wanted to create a type of granular playback that did not affect the input audio too drastically, and that kept the character of the audio intact. My initial goal was to work towards creating a system that allowed me to play a short riff or motif of around 10-15 seconds in length that immersed the listener in the sound recorded. I did not want to use small grains typically associated with granular synthesis for this purpose, but longer ones that played back enough of the analysed audio to create a faithful representation of the audio. Like the *Feedback* application I was interested in both file and ADC audio inputs, so I added the ability to select the input audio from a loaded file, or by simultaneously recording and analysing an audio signal from the ADC input.

The first step in the development was to decide what information to analyse. Like my undergraduate project, I decided to focus on the amplitude of the audio. I created a peak detection system that would record each sample position that was above a certain threshold, and the sample position when the amplitude fell below a certain level. I decided to call these sample positions ‘segments’. When the end point of a segment was found, a new segment could be defined. During testing this system I noted that certain audio files or audio inputs did not easily generate segments when there were peaks in the audio. To combat this problem I decided to build two controls that were displayed on the user interface. The first control I named 'sensitivity'. This changed the amplitude threshold allowing more segments to be defined during the analysis stage. The second control 'release time' was created because during the analysis I occasionally generated many segments that were close to each other in time. This release time control meant that a new segment could not be defined until the release time had elapsed, which helped to gain more significant amplitude data from the analysed audio.

I chose to have four separate granular devices that would be represented by playheads indicating the playback position on a waveform display of the analysed audio. I decided on four devices because I wanted to be able to accentuate the evolving aspect of the synthesis. With one device, the resulting audio is not only less eventful but it also is not as rich in texture. Also, this number created an acceptable level of CPU usage when the application was running.

The way I used the segments was to split them between the four devices. The start and end times of the segments were then used as points for each devices playback location to travel between during the granular synthesis. To facilitate the evolving aspect of the synthesis, I began to think about some form of sequencer that would drive the movement of the playheads and the parameters of the granular synthesis. In broad terms I wanted to create a movement from a calm quiet state, through to an erratic loud state. To do this I outlined six states, with lengths from 15 to 35 seconds that were to send various values to the granular synthesis devices that would change the sound.

At this stage I needed to define which parameters would be altered and also how they would be altered. The granular synthesis component had parameters for the interval between grains, the length of the grain, the playback start point, and the pitch of the grain. Firstly, I wanted to define the ranges that these parameters could fall between. To do this I created a control that allowed me to effectively audition the parameters by using the segment start points as the playback position and altering the other parameters manually. Using this manual process I was able to refine the parameters in real-time whilst listening to their effect on the synthesis. This allowed me to specify accurately the ranges I wanted the parameters to fall between. During this process I was conscious of the original idea of trying to create an immersive effect to the synthesis and set the ranges of the parameters to best replicate this effect.

My next task was to facilitate the evolving aspect to the sound that I desired. To do this I divided the parameters between the six states, with the longer less erratic parameters assigned to state one, and as the states progressed to state six the grains changed more in pitch and the interval between each grain became shorter. I wanted the progression through the states to be sequential, so I created a system whereby each state was preceded by a neighbouring state. This meant that the progression of the variation retains a character that is independent of the source audio. I defined the new states with a probability weighting that allowed me to influence whether a certain state moved up or down the number scale. This was configured in a way that biased the progression towards the middle states, meaning that states one or six provide a contrast to the most common middle states.

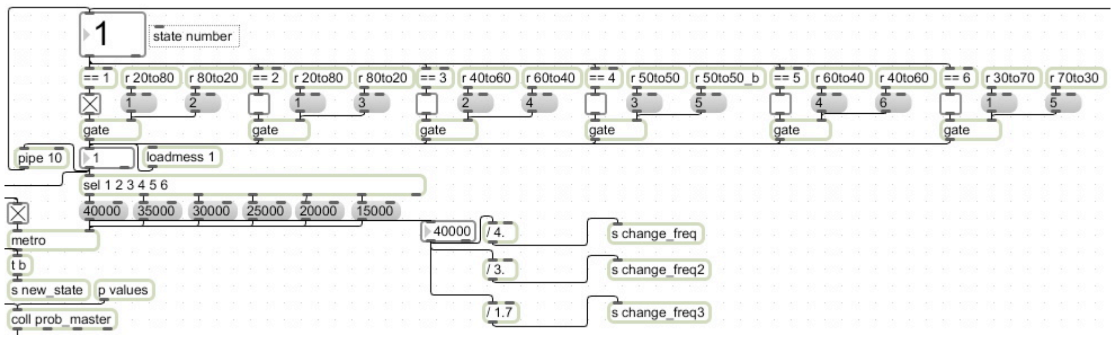


Figure 5. A section of the probability sequencer that drives the granular playback of the analysed audio. The different weightings for each state are visible as well as 'change frequencies' which alter the speed at which new playback locations are given, and the length of the interpolation between them.

An obvious way of achieving the evolving sound I was interested in was to interpolate between the values. This worked well for the grain interval and grain length, but I felt that the playback location was too static. To combat this problem I devised a system that utilised both the start and end times of each segment. When a new segment number was given, the start and end times are interpolated so that the playback location is continually evolving. In addition to this I created a value that interpolated from 1-10 that was then scaled to the start and end points of the segment. This value was derived from a low-pass filtered 'noise~' object whose cutoff frequency was altered in keeping with the states. This provided a sophisticated variation of the playback position and came to be the single most important development in achieving the immersive, evolving soundscapes that I had intended to create using the application. The interpolation between playback locations also provided a pseudo time-stretching effect[11], with longer interpolations seeming to stretch the playback of the audio, and shorter ones speeding the playback up. This time stretching aspect was possible by overlapping the grains, which meant that a new grain was triggered before an old

[11] Time-Stretching: An effect that lengthens or shortens the playback of an audio file, without altering the pitch.

grain had finished. When combined with a continually changing playback location this time stretching effect was achieved. I feel that in this particular part of the application the use of the random values derived from the 'noise~' object were essential to the organic nature of the sound created.

Through observing the behaviour of the application I noticed that there were occasions when a pattern emerged in the variation of the playback location. I had programmed the segments to be changed via the 'drunk' object, which is designed to move sequentially from one number to another in either direction. I realised that this was causing the pattern that I observed and built an additional feature in to the segment selection. The step size of the object was scaled so that during the more erratic states there were larger steps. The effect of this was to add more randomisation to the application, and it successfully eradicated the pattern that I viewed originally. In relation to this development I also created a control that allowed the user to randomise the current state. I named this control 'Jump State' and I thought that on occasion it was a useful for breaking up the sequential progression of the states.

With these developments I also began to think about how I would control the amplitude of the playback. I found that when using the mono input from the ADC, the playback was missing a vital stereo aspect that was present when using the file input. To solve this I created a panning control for each device. I linked this panning control into the states so that the higher the state, the wider the range within which the panning control would be set. This effect was a simple way of giving the ADC audio more depth and I feel that it was an important development in the overall sound of the application. I created an option for turning the automatic panning off so that this effect could be bypassed if I decided to use fixed panning values. I was also interested in the overall volume of the devices. I did not want the volume level to be static, so a simple interpolation of an amplitude value was used to vary the overall volume of each device. Again this control was linked with the states, with the calmer states being the quietest, and the more erratic states the loudest.

To save time during the development, I chose to incorporate the gain and waveform display sections from my previous project and also the filters. I tailored the record function from the previous project so I had the ability to record each device and the ADC input individually. Initially I did not feel the need to include the ADC input, but towards the end of my research I began to improvise over the analysed audio using my electric guitar. As this is was an addition that happened late on in the development stage I have yet to produce any compositions utilising it, however I do hope to explore compositions that do so in the future.

During a discussion with one of the alpha testers, the lack of user interaction was put forward as one of the main drawbacks of the application. I recognised that at times it would be useful to take control of one of the playheads and use it in a manual capacity. Originally, I had designed the software to have minimal user input but I felt that the addition of manual control would allow the user to have more of an influence over the audio created and increase the flexibility of the application. I wanted to retain the character of the synthesis, so I chose to keep the granular parameters hidden from the display and created controls to alter the position of the playback heads. For this control, I included two options: firstly, the playback location could be altered by clicking on the waveform with the mouse - this sent the playhead to the location clicked; the second method was by using a MIDI input that was scaled to the length of the analysed audio. When using this manual control I noted that the playback location was too static and built a variation control that randomised the position of the playhead when no user input was specified. This was achieved by sampling the output of a filtered noise signal. By using a low pass filter I was able to create a random variation that could be changed via the cutoff frequency and the amount applied to the playback position. I created controls for the cutoff, amount, and a 'time' control that allowed the user to smooth the signal resulting in less erratic variations. I also created a 'slide time' control to specify a time in milliseconds for the duration of the interpolation from one playback location to another. I created 'user' or 'automatic' modes for each device so that they each granular device's playhead could be toggled between the automatic playback defined by the application, or manual control.

Finally I created a MIDI section, which also used the MIDI learn function from the previous application. This allowed me to control the output gain of each device, the playback position, and also to switch between the 'automatic' and 'user' playback types.

Other Musical Examples

In the composition 'Surgeon Grains' I sampled a composition of the electronic artist Surgeon. I chose to use the audio from the beatless section of 'Floorshow 1.2' at 1:54 as the source for the granulation. I wanted to extend this part, and use the application to create a soundscape from it. I analysed the part ensuring that I found enough segments to generate an interesting output. For the composition I mainly wanted to let the 'automatic' functions of the granular devices play out over the piece's duration. Towards the end of the composition I employed the 'user' playback mode on one of the devices, which I controlled via a Korg Nano Kontrol using the applications MIDI learn function.

For the composition 'Gliese' I decided to use the ADC input to realise my original idea of immersing the listener in a motif of around 10-15 seconds in length. I used an electric guitar to provide this riff and let the 'automatic' playback occur for ten minutes. I added some post-production to the audio, using a small amount of EQ and reverb to finish off the piece. This piece is perhaps the best example I have produced of how the application can be used to transform a relatively small motif in to an extended soundscape that is continually evolving both in playback position and granular synthesis.

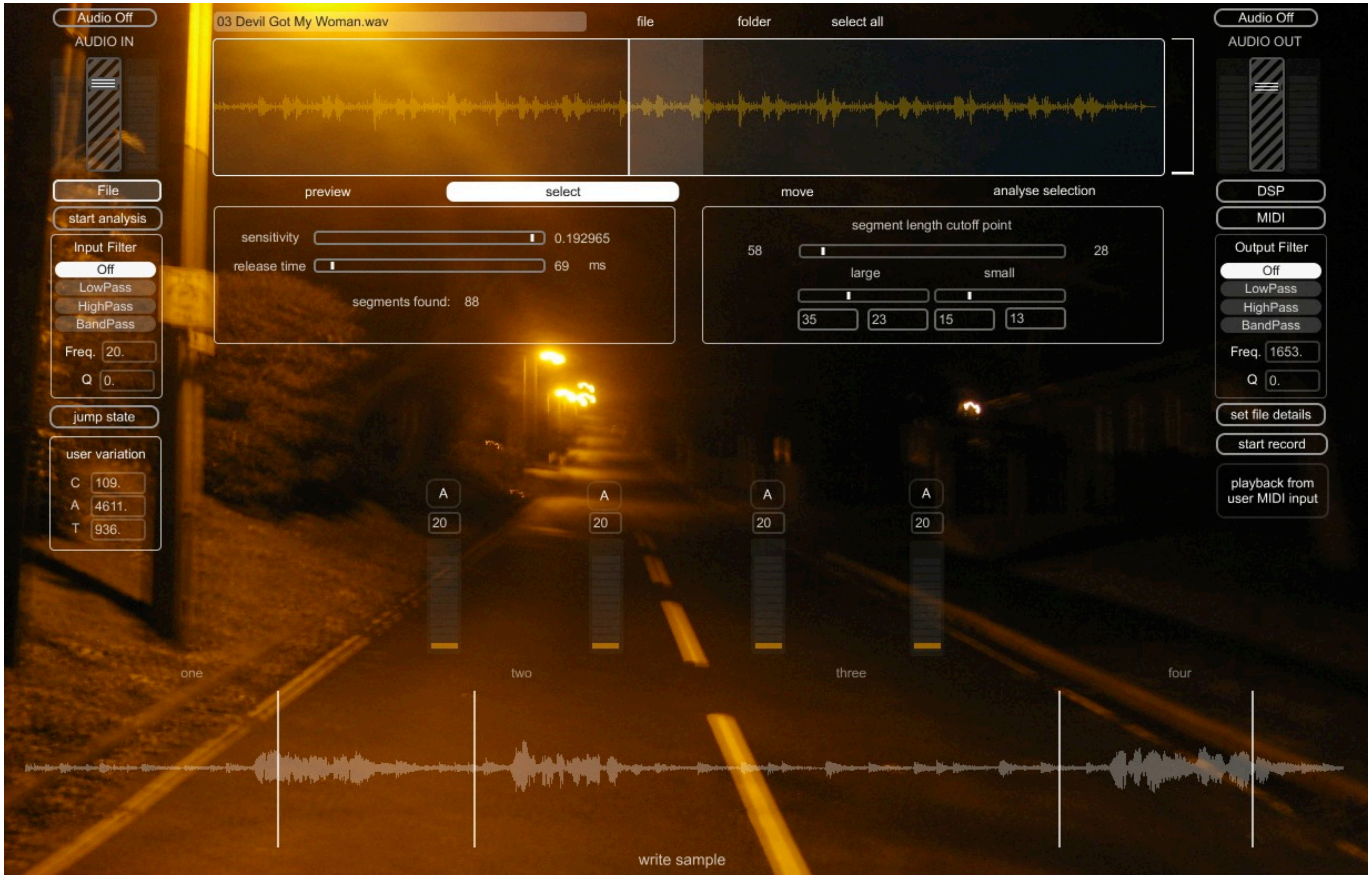
The application also had a less significant role in the composition 'Amb 03' where I used one device in 'user' mode to alter the playback position of a field recording, which was used by the application. The main drone in this composition was created by the *Feedback* application demonstrating that the two applications were not exclusive and could be used together in one composition.

User Interface

In order to make the application as clear as possible, I assigned a fairly large portion of the screen to the waveform displays. I positioned the file waveform at the top of the interface and the analysed waveform at the bottom.

I decided again to use a picture as the background to the software. I auditioned a number of pictures and settled on a picture taken at night. As this picture was striking in its colour, many of the standard colours of the Max objects did not work well with it. I chose to take cues from the colours of the photograph and use the orange colour of the streetlights for some of the colours of the objects. Similar to the last software I also chose to make many of the objects see-through to include more of the background picture, increasing the sense of depth.

I replicated the layout of the previous application, with the input on the left, the main focus of the application in the middle, and the output section on the right. I saw no reason to vary this theme, and feel that it creates a sense of continuity between the two pieces of software. When positioning the user interface objects I wanted to use the picture as a guide. My reasoning for this was that the picture has many features, and is not a simple background image like the previous application. By placing the objects relative to the features in the picture I feel that the user interface is clearer than if I had ignored the background picture.
In figure 5.1, the lamppost on the left separates the input section and the waveform and sensitivity controls, and that the gain controls are placed on the road, with the central white line splitting the four controls in the middle.


03 Devil Got My Woman.wav

sensitivity 0.192965

release time 69 ms

segments found: 88

segment length cutoff point 58 28

large 35 23

small 15 13

Input Filter Off

Freq. 20.

Q 0.

user variation

C 109.

A 4611.

T 936.

Output Filter Off

Freq. 1653.

Q 0.

Figure 5.1 The final user interface showing a selected portion of the input file at the top, and audio from the ADC input that was analysed below. The four playheads are shown in white. These playheads change to an orange colour when the device they are linked to is in the 'user' mode.

I also included an interactive help section, with the same design as the *Feedback* software. This displayed a picture of the background and allowed the user to mouse over various parts of the software to reveal the functions of the various controllable parameters.

Specific programming issues

I had originally scaled the sliders that distributed the segments using the length of the segments. This actually made distributing the segments evenly fairly difficult, because if two or three segments were of a similar length they could not be separated easily. Looking for a solution to this problem I found that the 'coll' object had an option of sorting its contents via an index. As I was storing the start time, the end time, and the duration, I was able to sort the contents of the 'coll' by the duration. This allowed the segment length cutoff controls to be far more precise, as they were scaled the number of segments found. In addition to this I often found myself trying to separate the segments equally to all four devices. As this was time consuming, I automated this process so that once an analysis is finished, the application separates the segments between each granular device as equally as possible.

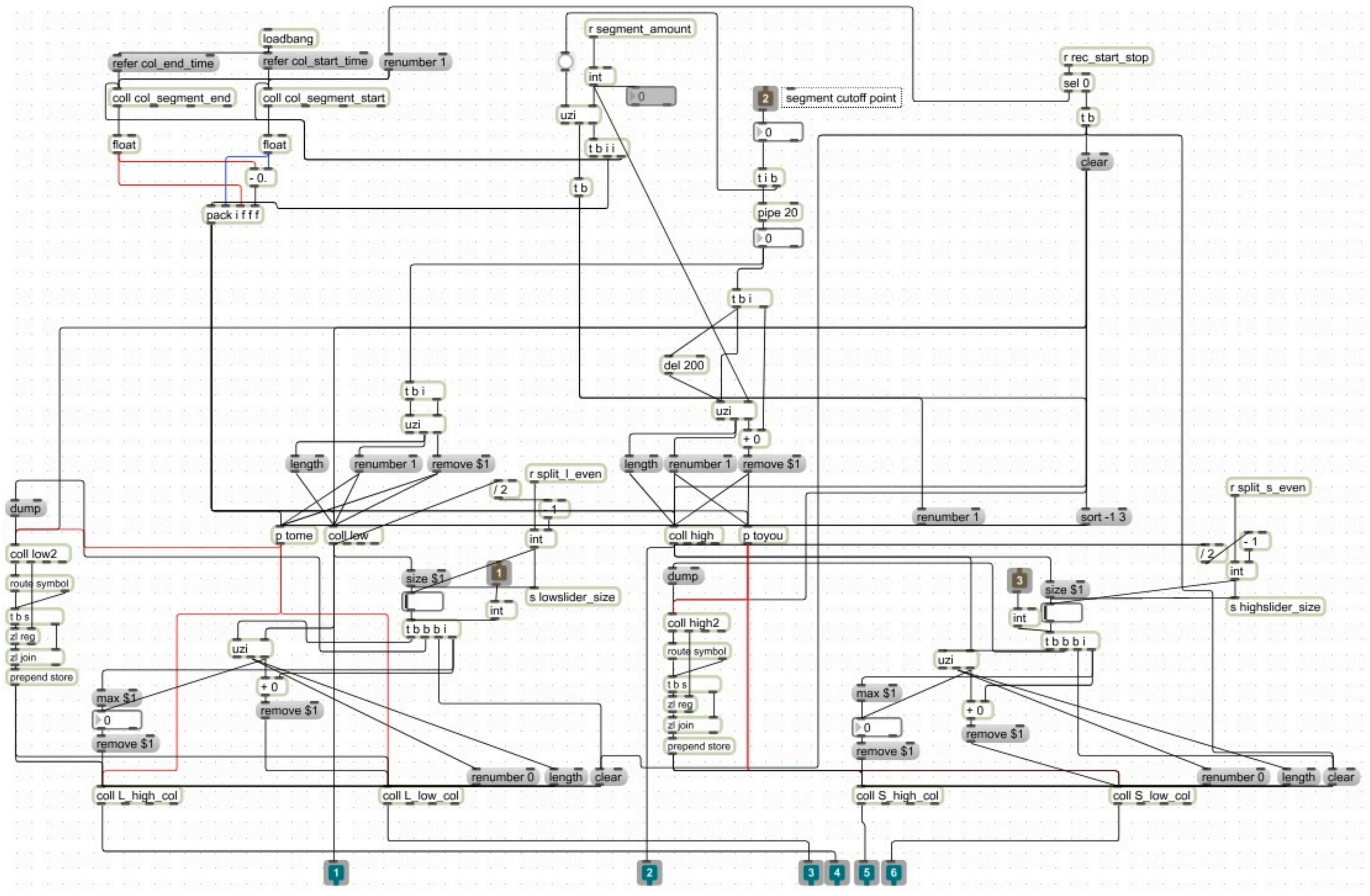


Figure 5.2 A view of the programming of inside the 'segment sorting' subpatch. This method of distributing the segments made using the application much easier.

Towards the end of my research I decided to see what objects were taking up the most CPU. I noticed that the 'waveform~' objects I was using to draw the playheads for each device took up a large portion of the applications CPU usage. I decided to use the 'LCD' object to draw the playheads as this reduced the CPU usage significantly. Initially it appeared not to have much of a difference, however when I limited the drawing rate to update once every 50 milliseconds it became a useful refinement.

My final addition to the program was the ability to turn the processing of each device off. I added an option to toggle this feature next to each devices gain slider on the main user interface. Primarily this development was to save CPU when not all four devices were needed, but it was also useful as a way of muting the devices.

2.3 – Sampler

Foundations and development process

As the previous two pieces of software were tailored to my interest in ambient styles of composition, I decided to make a piece of software that was designed to complement my more rhythmical beat-based music. My preference is to use synthesis for the main focal point of this type of music, but I am also interested in using samples in a rhythmical way. One artist who exemplified this technique at the start of the 21st century was Akufen, the musical pseudonym of the Canadian artist, Marc Leclair. His style of 'microsampling' is a technique using small fragments of often unrelated audio material as samples that are used within the context of a dance music composition. As I have used samples extensively in my own work, I know that it can be time consuming to find usable fragments of sound to use. I became interested in creating a piece of software that automated this process. I was less concerned with the human process of defining what constituted a usable sample, as this brings with it many subjective connotations. My idea was to create a system that allowed me to generate random samples, meaning that I would have no control over where the sample began. The attraction of this idea was that samples could be generated that I may never have found via a manual process, opening up all parts of the audio file as potential sample material.

I chose to have sixteen 'voices' of the sampler. Meaning that sixteen different samples could be played back independently of each other. As I planned to use the application in conjunction with a sequencer, it was logical to use MIDI to trigger the samples. I created a MIDI input system whereby I could specify the MIDI input and channel that triggered the samples. The samples are triggered by MIDI notes that start at C3 (or 60) and the finish at D#4 (or 75). Whilst a standard MIDI input from a sequencer or MIDI keyboard is useful, I decided it would also be beneficial to allow a MIDI control input to trigger the samples. This is because I use a Korg NanoKontrol, which has pad style buttons that I could use for triggering the samples.

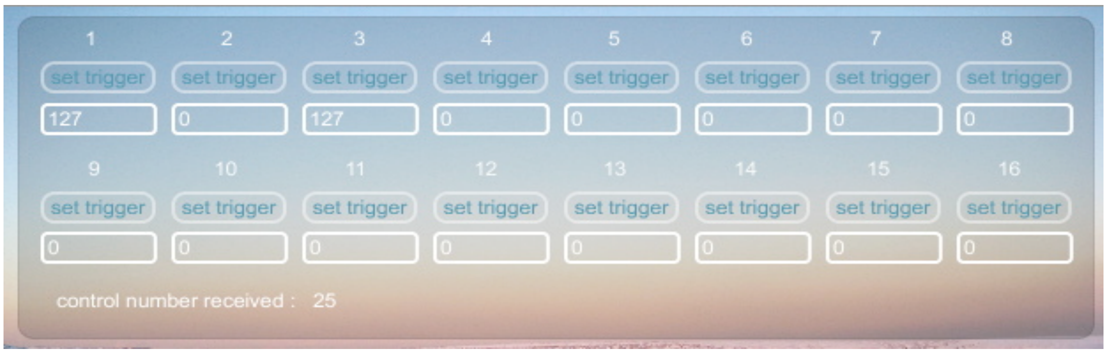


Figure 6. The control input dialog allows MIDI control data from Pads to be used to trigger the samples. In this picture, samples one and three are being triggered.

At this stage in the development I was simply triggering the playback of the loaded file and whilst this provided interesting sample start points, I felt that I needed more control over the playback of the sample. The first feature I implemented was the ability to loop the sample. This meant that if the note that triggered the sample was held for a duration longer than the sample's length, it would begin to play from the beginning. I found that this effect was interesting when applied to multiple samples of different lengths as it created a continually changing rhythmical structure as the different samples looped together. This effect can be heard in the composition 'Aphex Plink' which was created using the control input of my Nano Kontrol MIDI controller. To create samples of varying lengths I set the random range to between 300 and 1000 milliseconds and randomised all of the voices. In the composition I tried to create a sense of continuity by triggering four samples that started the piece, while I triggered other samples to vary this main motif. When I had recorded the audio from the application I added some post-production in Cubase, which involved layering the audio file created, adding delays, equalisation, and reverb. I feel that buy using a single take as the core of the composition I retained a 'live' aspect which gave it a slightly unpredictable element that I enjoyed.

The next stage was to focus on the amplitude of the sample. As the software was conceived with rhythmical applications in mind, it was obvious to add an amplitude envelope that was triggered when the sample was played. I used the 'function' object, which I scaled to the length of the sample. I also used the object's sustain feature which allowed me to specify a point that would be used to define the amplitude level if the sample was looped for longer than the duration of the sample. I also added a control named ‘velocity scale’. I created this function because I wanted to allow for variations in the overall amplitude of the sample. For this I used the MIDI velocity data, which has a range of 0-127 to scale the volume of the sample. This could be set within a range of zero to one, allowing the full range of amplitudes or a reduced range to keep the volume above a certain level. I implemented this function for each sample, which gave me the option of different scaling settings for each voice. To finish the variation of amplitude I created a panning control. On completion of this section I had much more control over the playback of the sample, and felt that I could use these new features in a creative way.

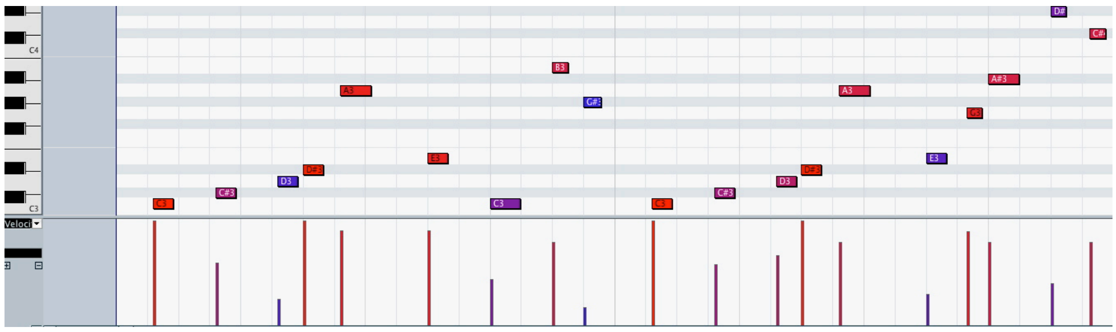


Figure 6.1 MIDI from the program Cubase triggers the samples. In this example, the velocities are varied, allowing me to play voices back at different amplitudes using the scaling feature.

I designed the application so that when a file is loaded 16 random sample start points in the loaded file are assigned to the 16 voices. I created a function to randomise the start point of each voice individually or to randomise all voice’s start points together. This function of randomising all voices was to become the focus of my next developments, which were designed to generate a diverse range of samples from the loaded audio.

As much of Akufen's cut-up style works involve contrasting samples, I allowed a folder of files to be loaded so that each voice could have a random file during randomisation. In addition to this I also added pitch shifting, which was set either by selecting a note on the 'kslider' object which changed the pitch relative to the equal tempered scale, or by altering a floating point number to access the pitches between the notes. I also included a filter with filter type, cutoff, and resonance settings. With this variety of processes I recognised a need to automate the randomisation process so that I could randomise all of the voices, generating a varied list of parameters. To do this I created a 'random settings' section that allowed me to specify ranges that the parameters would fall between. This included setting the range of the length of the sample, the range of the filter cutoff and resonance settings, and also the range of the panning. The range of the pitch shifting could also be set, with a choice of a pitch that was changed via the 'kslider' object, or via the floating point control. Whether the loop, and envelope were activated could also be randomised, as well as varying the filter type. Also, if a folder was loaded, a random file could be selected for each voice. With this section finished I was able to randomise each voice within defined ranges and I feel that I successfully integrated enough control over the randomisation processes to guide the computer in generating samples that I wanted to use.

When using the application with certain files I noticed that the randomisation of the sample start point would be at the beginning or end of an audio file, which was silent. I thought that with certain rhythmical applications this produced an interesting effect of leaving a space in the audio pattern where there would normally be a sample. However, in general this effect was problematic, so I created a function that allowed the audio file to be cropped so that the silence could be removed. This could also be used to crop the audio file to a specific part of the file with which to take samples from. Again, through using the software I noted that when randomising every voice to generate a new set of samples, I would sometimes overwrite a sample that I wanted to keep. To solve this problem I built a feature that allowed the exclusion of a sample from the randomisation process. This meant that I could keep the samples I liked and use the 'randomise all' process again to generate new samples in place of the old samples that I deemed unusable. By creating this feature I exercised more control over the randomisation process, and the application became more of a tool to generate usable samples, as opposed to generating completely random audio material.

Occasionally, I generated a sample that I wanted to play back at different pitches, like a traditional pitched sampler. To facilitate this I created a feature that allowed copying of one sample to a different voice or to all voices. Once copied, I could then set the pitch via the 'kslider' object. I did not use this feature very often but felt that it was a worthwhile addition to the application.

As the software is designed with studio based work in mind, it was important for me to be able to recall the source audio files and all the various parameters of each voice. Using the 'pattr' family of Max/MSP objects allowed me to do this, and ensured that I was able to recall presets effectively.

Like the other two projects, the sound output is based largely on the sample used. When using the features from the random settings section of the program, a wide variety of sounds can be created from the input audio. The most notable changes being via pitch shifting, filtering, and looping. Whilst the sample playback is fairly simple and is ultimately the aim of the software, it is the way in which the user arrives at specific samples that is the more important process.

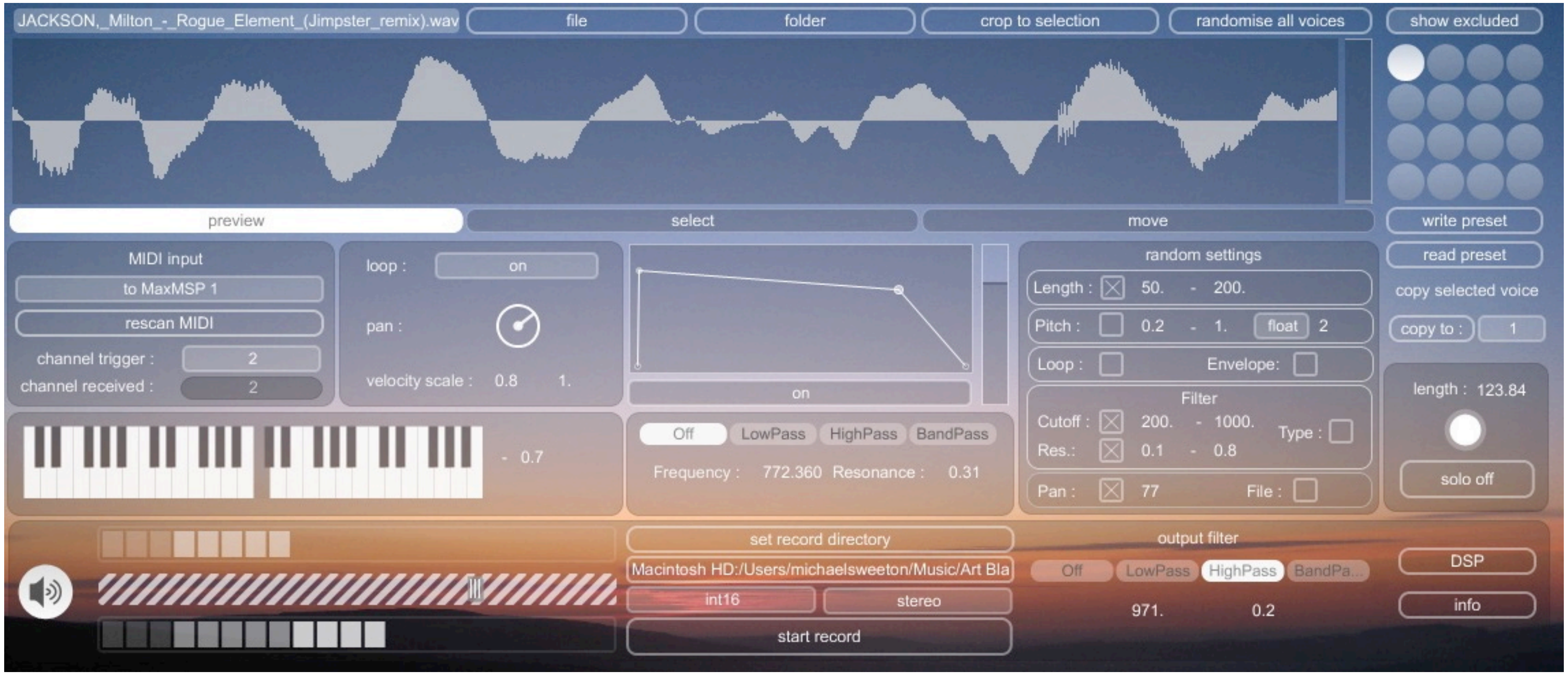
Other Musical Examples

I include the audio 'Sampler Strings' as an example of the 'randomise all' function and how this can be used to generate useful samples. In this example I use this function repeatedly to generate 16 new samples that are triggered by a looping MIDI from a sequencer, similar to figure 6.1. As the example progresses I alter the length of the samples and finally the pitch of the samples to show how these functions can be used to provide usable sounds from a compositional perspective.

User Interface

In keeping with the other two pieces of software, I chose the similar design of overlaying the user interface objects on to a background picture. I chose to colour many of the user interface objects white as I felt this gave the software a look that was more attractive, and also provided a less fatiguing contrast compared with darker interface colours in relation to the background.

To provide a way of recalling each samples parameters I used the 'preset' object, which also doubled as a control to navigate to the different samples. This allowed the user to click on a circle numbered from one to sixteen, which recalled the settings of the corresponding voice. To help the user navigate the samples, a MIDI note within the range of the triggered samples makes the corresponding preset blink white. This feature was implemented by positioning a 'bang' object behind each circle of the preset object, which blinked when the corresponding sample was played. I had originally planned to use a playhead display with the ‘waveform~’ object, like the previous two pieces of software, but as many of the samples were short, the speed at which the playhead was moving made it jump between parts of the waveform. Instead, I chose to highlight the background of the ‘waveform~’ object when the sample was triggered.



channel trigger : 2
channel received : 2
loop : on
pan :
velocity scale : 0.8 1.
on
- 0.7
Off LowPass HighPass BandPass
Frequency : 772.360 Resonance : 0.31
Length : ☒ 50. - 200.
Pitch : ☐ 0.2 - 1. float 2
Loop : ☐ Envelope: ☐
Cutoff : ☒ 200. - 1000. Type : ☐
Res.: ☒ 0.1 - 0.8
Pan : ☒ 77 File : ☐
copy to : 1
length : 123.84
Macintosh HD:/Users/michaelsweeton/Music/Art Bla
int16 stereo
Off LowPass [✓ HighPass] BandPa...
971. 0.2

Figure 6.2 The final user interface, with the first sample selected.

When working with looping MIDI from the sequencer, it became difficult to identify which sample corresponded to which voice as many of the presets could flash at short intervals. To remedy this, I created a solo function that muted all other voices, leaving only the selected voice playing.

Finally I created a help section with the same design as the other two pieces of software that allowed the user to mouse over a picture of the interface, revealing information about the controllable parameters of the software.

Specific programming issues

During the start of the development I was fortunate enough to discover a new object in development that was offered by Cycling 74 on their online forum[12]. 'polybuffer~' is an object that groups a collection of 'buffer~' objects in to one object. It allows the user to load a folder of audio files that can be referenced via other objects without having to create a collection of buffers in which to store the audio. The advantage of this is that when using the object, the developer does not need to create an unknown number of 'buffer~' objects in which to store the audio, and does not need to limit the amount of files in a folder that the user can load.

I found that using the 'groove~' object to drive the playback of the samples was particularly useful for the looping function, as the object was designed with this application in mind. In relation to this I used parts of an abstraction included with the examples of Max 5 called 'grooveduck2'[13] which helped to reduce the clicks

[12] http://cycling74.com/forums/topic.php?id=26356 (accessed 2.6.11)

[13] On a macintosh computer with Max/MSP loaded, the file is located here: /Applications/Max5/examples/sequencing-looping/audio-rate-sequencing-looping/lib/grooveduck2.maxpat

caused by the jump in sample values from the final sample of the loop, to the first sample of the new loop.

I designed the interface of the application so that I had one set of user interface objects that would display the parameters of the selected voice. When a voice was selected the parameters were recalled, but were also saved when they were altered. To do this I used the 'poly~' object's instance number to formulate a message which was sent to the various 'receive' objects inside the instance. This allowed me to target the instance from the main user interface using the 'forward' object which was given a message containing a number that correlated with the 'receive' objects inside the 'poly~' object. I also designated a slot in the 'pattrstorage' object for the files loaded, the output filter settings, the MIDI input settings, the overall volume settings and the control input settings. All of this information needed to be saved and reloaded using the 'pattrstorage' object, which was not completely straightforward. When loading a preset I had to run through the voices in sequence, similar to the 'randomise all' function in order to send the various parameters to the correct instances inside the 'poly~' object. I also had to recall the slot, which contained the other parameters listed above.

In order to facilitate the 'randomise all' function I employed a process of automatically changing to a voice, triggering the random parameter generation, and so on until the 16 samples had been randomised. If a voice was excluded this was removed from the randomisation process.

Conclusion of research

On completion of the research I feel that I have created three useful software tools that are relevant to my compositional needs. I often use the *Feedback* software when composing drone or loop based music, and have been increasingly using the other two pieces when composing both ambient and beat-based music. The link between the three projects is that they all use samples as a source and random number features to create a sense of organic evolution or variance in the final sound that is compositionally rich. This is in contrast with a number of other prototypes I worked on during my research, which utilised oscillator-based synthesis. I created a conventional equal tempered scale synthesiser, and a more experimental synthesiser using sustained sine waves. Whilst I enjoy using these prototypes and include them in my compositions, I chose to exclude them from my submission as I do not think they are of the same quality as the other pieces of software, both in the originality of the idea, and the range of the sound output. In addition to this, I feel that oscillator based synthesis has a narrower application than sample based processes, which have more potential for manipulation and variation.

Almost as important as the function of the tools, is the fact that have proved genuinely useful to create music and are highly stable stand alone applications. When using Max/MSP, I have always been interested in creating interfaces that are attractive to the user and easy to use, and as I have many Max/MSP prototypes left unfinished, it is fulfilling to see my ideas developed to the stage of a completed application.

I also feel strongly about distributing the applications in the future and I am currently in the process of establishing a large user base for which future pieces of software could be targeted.

Bibliography

Holmes, T., *Electronic and Experimental Music,* Routledge (New York, 2002).

Roads, C., *The Computer Music Tutorial,* MIT Press (U.S.A, 1996).

Roads, C., *Microsound,* MIT Press (U.S.A, 2004).

Licht, A., *Sound Art,* Rizzoli (New York, 2007).

Cox, C. & Warner, D., *Audio Culture: Readings in Modern Music,* Continuum (New York, 2004).

Young, R., *Undercurrents: The Hidden Wiring of Modern Music,* Continuum (New York, 2002).

Discography

Akufen, *My Way*, Force Inc. Music Works (2002).

Taylor Deupree, *Comma,*, 12k, (1998).

Taylor Deupree, *Stil.*, 12k, (2002).

Baby Ford & The Ifach Collective, *Sacred Machine*, Klang Elektronik (2002).

Brian Eno With Daniel Lanois & Roger Eno, *Apollo – Atmospheres & Soundtracks*, E'G Records, Polydor, (1983).

Basic Channel, *BCD*, Basic Channel, (1995).

Steven R. Smith, *Cities*, Immune, (2009).

Mika Vainio, *Behind The Radiators*, Touch, (2008).

Godspeed You Black Emperor!, *F# A# ∞*, Constellation, (1997).

Max Richter*, The Blue Notebooks*, 130701, (2004).

Twine, *Violets*, Ghostly International(2008).

Surgeon, *Floorshow 1.2*, Counterbalance(2006).

Online References

Christopher Hipgrave, *Ambien*t - http://www.audiobulb.com/create/Ambient/AB-Ambient.htm

Giorgio Sancristoforo, *Gleetchlab* - http://www.gleetchplug.com/gleetchplug/Software.html

John Burton, *Forester* - http://leafcutterjohn.com/?page_id=14

Cycling 74 Forums - http://www.cycling74.com/forums/

12k Forums - http://www.12k.com/forum/index.php

Martinez, W.L., *Graphical User Interfaces* Wiley Interdisciplinary Reviews (2011) - http://onlinelibrary.wiley.com.librouter.hud.ac.uk/doi/10.1002/wics.150/pdf