1 Introduction

1.1 Introduction

1.2 Summary of work completed

1.3 Motivation

1.4 Thesis structure

2 Background to study

2.1 Introduction

2.2 The idea generation stage of music production

2.3 Discussion of DAW Metaphors

One of the primary tools used by electronic musicians today for the production of music is DAW and it's inherent metaphors based on analog system still reign supreme in the field [@bell_journal_2015]. The familiar concepts of analog tape machines and mixers benefit the novice user by offering a network of familiar and tangible real world metaphors in which to carry out their creative work. However, as well as the benefits that these types of metaphors bring, they also impose some limitations and bring about certain biases. Musical ideas that are difficult to realise can be left unexplored.

A particular criticism of the DAW is the difficulty in maintaining and managing the editing of complex automation information. Automation is the term given to the continuous altering of aspects of the sound and is usually represented in lanes separate to the primary note pitch information. It may be recorded in or drawn in by the producer. Difficulties can arise, when multiple subtly interacting lines of automation, such as pitch bends and filter changes are being manipulated. William Coleman gives a particularly clear example of this and outlines the difficulty of representing "portamento time", the time it takes a note to slide from one to the next. The visual results can be jarring, unintuitive and not reflective of the audio results.

Duignan (2008) describes a similar problem in his study that monitored professional producers working in DAW environments [@duignan_computer_2008, p. 156]. The particular problem identified by Duignan was that of processing one off effects for single musical events. A number of convoluted processes were observed, including bouncing the affected portion to audio, duplicating the track, setting up a particular auxiliary for the effect and controlling the effect with automation.

In these cases, the hierarchy imposed by the DAW gets in the way, where it could be modeled quite elegantly in a more open program such as Max Msp. This, unfortunately, raises the issue of drifting into the area of analytic thinking and away from creative thinking, a combination that John Cage advises against: "Don't try to create and analyse at the same time. They're different processes." [@popova_10_2012] The need to explore alternative metaphors is clear. A description of a promising alternative metaphor, that of drawing/sketching will now be discussed.

2.4 Discussion of more open systems

2.5 Music in the browser: a new frontier

Discuss drum machines, etc available on the web. Note the accessibility that they provide. Perhaps introduce tone.js here.

2.6 Conclusion

While the dominant metaphors used in DAWs have their uses they can lead to limitations in the creative process particularly at the early stage of ideas creation. More open system give too much power and impede the creative process.

3 Similar work

3.1 Introduction

As is pointed out by Levin (2002), the exploration of synchrony between audio and visuals is a practice going back centuries, being variously termed "ocular music, visual music, color music, or music for the eyes" [@levin_painterly_2000]. The twentieth-century technique of the optical soundtrack, however, brought these ideas to a new level of sophistication. The technique, which involved placing marks via photography or direct manipulation to specify audio properties, was explored by such luminaries as Oskar Fischinger, Norman McLaren and Daphne Oram. Oram's particular take on the technique will now be discussed as her system most closely resembles that of a piano roll.

3.2 Sonic sketching from a historical perspective

3.2.1 Oramics

A primary motivating factor behind Daphne Oram's development of the Oramic's machine was to bring more human-like qualities to the sounds generated by electronic means. The machine worked by playing back multiple lanes of film tape in unison, defining a monophonic series of notes as well as control signals to shape their timbre, pitch and amplitude. She details the thought process behind this in her hugely insightful and broad ranging journal style book, "An Individual Note" [@oram_individual_1972].

The aspects of the sound that she wishes to control are volume, duration, timbre, pitch, vibrato and reverb. In order to do this she describes a simple musical notation language based on the freehand drawing of lines combined with discrete symbols. The lines, which she describes as the analog control, are used to define volume envelopes. Interestingly, the default and preferred method for the parameters she wishes to control is the continuous line rather than discrete note symbols. For instance, she avoids the use of a static velocity per note and instead only specifies the use of a control envelope to change amplitude.

The discrete symbols, which she categorizes as digital control, are used to define individual pitches and are termed neumes. She highlights that notes shouldn't remain static and, thusly, an analog control of each note is also specified. Similarly to amplitude and vibrato, timbre is also defined by the freehand drawing of lines and is something that with practice the "inner ear" can develop an intuition as the sonic results of different line shapes. It is Oram's belief that the hand drawn nature of the lines make the results slightly inaccurate and to some extent unpredictable but in this also lies the possibility of bringing more humanity to the cold and precise machines generating the electronic signal.

3.3 Sonic sketching in the twenty first century

3.3.1 A Golan Levin's AVES

Golan Levin created the interactive audio visual system, AVES, a series of audio visual installations in the late nineties and represented a landmark in the field of visual music. It is an attempt to move away from the diagrammatic approach to musical interfaces and to present an interface that is painterly in approach. Taking strong influence from visual artists such as Paul Klee, he presents a system that maps user input from a graphics tablet and mouse to visuals and to music. The intention is to create a strong visual correlation between the visuals and the music. A variety of approaches are taken to achieve this, all of them involving an algorithmic approach to one degree or another. For instance, in the piece "Aurora", he maps visuals of a large amount of particles to a granulated sound synth sound source. He didn't take the approach of an exact mapping

of visual particles to audio particles however and instead used a statistical control approach to approximate the correlation in between the visual and aural. [@levin_painterly_2000]

For Levin, the digital pen input in combination with it's infinite variability represents an ideal instrument for creative expression in his digital temporal audio visual paintings. [@snibbe_interactive_2000] The reason he gives for this is, similar to a musical instrument such as a violin, the pen is instantly knowable in that a child can pick it up and start creating marks but infinitely masterable through practice and hard work, and ultimately a vehicle for creative expression after a certain amount of mastery. A set of criteria that he and John Maeda arrived at to evaluate the success of their experiments was: is it instantly knowable, how long did you use it, how much of your personality can be expressed through it and, finally, with practice is it possible to improve using it. These ideas foreshadow much of the ideas of the Natural User Interface and echo principles in game design and music instrument design.

Levin's work is largely realtime and transitional in nature with gestures giving rise to visual and audio reactions that rise, fall and dissipate. A description that he uses of some of work is that of creating ripples in a pond. Therefore his work is very much geared towards an instrument like experience and is not concerned with the recording or visualization of the temporal unfolding of musical events as would be the function of compositional tools such as DAWs and musical notation. Indeed it is a conscious design decision to avoid such representations. Many of the principles and ideas of his work can, however, be applied in the context of a composition tool.

3.3.2 William Coleman's sonicPainter

SonicPainter by William Coleman is a novel musical sequencer that seeks to address some of the shortcomings of traditional approaches to music sequencing found in commercial DAWs [@coleman_sonicpainter:_2015]. The focus of the line and node based interface (see figure) is to bring timbral shaping to the fore rather than being hidden away in miscellaneous automation lanes. The design takes influence from legacy musical systems, in particular, UPIC and incorporates ideas from visual music and embodied cognition.

Similarly to traditional sequencers the x axis represents time and the y axis pitch. Note information is input via keyboard and mouse but more as an intermediary prototype stage. The default mode for input is to click to create a node and follow that with an additional click to continue to shape the note. The note can be ended by clicking a keyboard shortcut. By enabling the drawing notes as lines in this manner, the unfolding of the note can be explicitly represented visually. Other timbral aspects such as vibrato are represented by further visual manipulation of the line. For instance, an overlaid sinewave line indicates the timing and amplitude of the vibrato. In addition, the system allows for freehand

input of notes.

Coleman recommends that the system could be further improved by multitouch input, specific elements for the control of other synthesis techniques, time/pitch grid quantization, and further visual timbre feedback representations. Many of these recommendations will be addressed and explored in SonicSketch through a system built using the guidelines of NUI (natural user interface).

3.4 Alternative music systems on the web

3.5 Summary of currently available music creation systems

3.6 Conclusion

4 My approach

4.1 Introduction

4.2 Appraisal of options

4.3 Approach - theory

4.3.1 HCI considerations

4.3.1.1 The natural user interface

NUI is an evolution of the concept of the graphic user interface and refers to an approach to human computer interaction beyond that of the traditional keyboard and mouse or what has been termed the WIMP model. It encompasses a set of guidelines and best practices which are set out most comprehensively by Wigdor (2014). Some of the basic tenets of NUI are as follows:

- Harness existing skills when possible without necessarily mimicing the real world tool or instrument [@wigdor_brave_2011, p.13].
- Be friendly and learnable by beginners but allow for mastery given enough practice (a sentiment shared by Levin, above) [@wigdor_brave_2011, p.13].
- Immediate feedback for all interactions should take place, most usually but not limited to, visual feedback. [@wigdor_brave_2011, p.87]

Furthermore, the interface should take advantage of the particular affordances offered by the input method. [@wigdor_brave_2011, p.115] An apt example of this is the early introduction of digital pens for windows laptops where the pen wasn't suited to the WIMP interface and failed to receive widespread usage. In this case, the interface forces the user to carry out awkward gestures for the

medium, including double clicking and right clicking, failing to take advantage of the stroke gesture much more suited to it.

The system CrossY, referenced in Wigdor (2014), uses a cross gesture stroke to interact with buttons, menus, and widgets, as well as the painting functionality [@apitz_crossy_2004]. The CrossY gesture system enables the user to, for instance, select brush size and colour in one stroke by dragging the pen from right to left across an icon and validating selection by dragging past the left or bottom selected leaf icon. This is illustrated clearly in the left-most diagram of the provided figure (fig. 3).

4.3.2 The Musical Interface Technology Design Space (MITDS)

4.3.3 Cross modal perception

4.4 Approach - practice

4.4.1 Delivery on Web Browser

4.4.2 Modern web browser as a delivery platform

4.4.3 Benefits of using tone.js cite:mann_interactive_2015

4.4.4 Paper.js for the graphics system

4.4.5 FM synthesis :: give a brief overview of fm synthesis and why it was a

good choice for the application

4.4.6 Live coding workflow

4.4.6.1 [Introduction]

4.4.6.2 React.js framework

react is a web framework built by facebook that aids the developer in updating the dom (document object model), a process that is required when the state of teh applciation changes. this was a role traditionally carrried out on the server and served to users as a static page. this all changed however with the rise of single page applications (spa) around the 2???s. the value proposition of the spa is increased interactivity and responsives to user input, allowing the look and contents of the page to update dynamically as the user interacst with the system. to aid in the construction of these spa's a number of frameworks to

help the process were introduced by the open source community. some popular early examples include *backbone.js* and *angular.js*. a technique that saw some popularity was a system called two way binding which created two way link between the current state in the model and the visual appearance of the view. this however has a number of issues including some serious performance issues, in addition to some conceptual problems (???ref). react offers a simpler one way bidning ssytem using what is termed the virtual dom. in this model a special virtual version of the dom is constructed and when the model changes is updated. the parts of the dom that require changing can thusly be pinpointed and the real dom can be efficiently updated. this system has proven to be particulalrly beneficial when paired with functional programming techniques, a style of programming that emphasizes the use of pure functions as the primary building block of programs. in the case of working with the dom, it can lead to not only an increase in efficiency in the rendering of the applications but also a simplification of the programming model. a number of projects have emerged that attempt to bring this benefits of the react model beyond the realm of the dom including writing console prgorams (???ref), writing web audio applications (???ref) and even arduino projects (???ref).

4.4.6.3 Clojurescript

4.4.6.4 Managing state with re-frame

4.5 Conclusion

5 Execution

5.1 Introduction

The following chapter gives an outline of the process that was undertaken to build out the final application. A description of early prototype work is given to give context to the construction of the final working prototype version. This is followed by a description of the technical architecture of the system as well as an outline of the sometimes tricky setup process of getting the live reload system working (as described in the previous chapter). Some detailed discussion of the the core functionality of the system is then given. The core of the system primarily consists of timeline events, which visually manifest themselves as strokes on the canvas and aurally as fm syntesized frequency modulated sounds. As will be discussed an important aspect of any well constructed software system is a good degree of seperation of concerns, a characteristic espoused to and evident in the resulting code. To this end, the code that brings the central functionality to life can conceptually divided into a data or entity layer, a business

logic or use case layer and an output layer which in this case consists of the visual output into a html canvas element and the audio output through the web audio api. In this sense the architecture conforms to the principles of the Clean Architecture as presented by Bob Martin (???).

5.2 Early prototype work

5.2.1 Melodypainter

Melodypainter is an early protoype built out in Max MSP that allows users to draw freehand lines, which are converted into break point function data and used to generate a melodic profiles using Bach for Max MSP. Bach is a suite of composition tools that allow for a number of computer aided composition techniques (CAC) and provides similar functionality to IRCAM's Open Music system. These melodic profiles are then filtered to only includes notes from a pentatonic scale, to give reasonably pleasing aural results. Some notable flaws in the system include the following. It is limited to strictly western tonal music styles. It has no allowance for rhythm and plays only eight notes giving results a noticeably bland and predictable quality. The freeform nature of sketched input however was quite a pleasing means of inputting the control information.

5.2.2 Sonicshaper [0/1]

A separate application was created in Processing which allowed users to draw shapes, using either mouse or ideally, pen input and have a sound that is associated with each shape played back. As the sound of each shape plays back, it is lit up using animation, creating a strong connection between the shape and it's resulting sound. The application uses the "gesture variation follower" system [@caramiaux_adaptive_2015], which while promising in principle, didn't have a high rate of accuracy in recognizing the shapes.

5.2.3 TODO Web version of William Coleman's SonicPainter [0/1]

A potential starting point that was considered was using the code from William's SonicPainter and porting it to the web platform. This process proved to be quite straightforward. The processing code could more or less be embedded in a webpage as is using "processing.js", a web version of the Processing library that enables users to run processing sketches in the Web Browser. Some notable changes that had to be made were removing the OSC functionality as this is not technically possible to use in a browser. In addition, some other pieces of code had to be commented out and tweaked. As it's not possible to run Max MSP patches in the browser, the audio system was re-implemented using Tone.js. As SonicPainter uses simple FM synthesis, a very close approximation to the

original version could be created. In the end, it was decided not to build on this codebase however as there were some issues with functionality and useability that would be difficult to resolve in an inherited codebase. A fundamental issue was that transitions between certain states would cause crashes or unpredicatable behaviour. An example of this is when a user attempts to use the vibrato tool while in the process of creating a note. Instead of either finishing the note and starting the vibrato tool or disallowing the behaviour, the program would crash. This is a common problem in software development and is evidence even in commercial products. To alleviate such issues in the new codebase a more disciplined approach would be taken to managing transitions between states. The process of porting the code did however give a more in depth incite into Coleman's implementation incuding the pitfalls mentioned above. In addition, the basic visual look and conceptual functionality would form the basis of the workings of SonicSketch.

5.3 Actual implementation

5.3.1 Setting up the architecture

5.3.1.1 Clojurescript and javascript npm modules

Despite the fact that clojurescript has existed for six years(???), some areas of the development process are still difficult, particularly when building out a more complex real world application. It should be noted that a good deal of work is being carried out to make this a smoother experience and thusly these pains are likely to become less of an issue in the near future (???ref). It should also be noted that building applicatons using plain javascript is not a trivial process and in all likelyhood will include a build process using a system like webpack or browserify. A primary issue that had to be resolved to allow the application to be built out was the incorporation of javascript npm modules. NPM is the module system used by node.js originally for more server oriented technologies but increasingly for rich clientside applications. For a purely javascript application, it would be a matter of simply adding the desired libraries as dependencies. However, with the use of clojurescript some extra steps needed to be carried out. In addition to adding the dependencies, a javascript file was created that imported these into a "deps" object. This deps object could then be referred to in clojurescript using the standard interop syntax `js/deps.myDependency`. At the time of development an alpha feature that allowed npm dependencies to be declared as part of the project.clj file was experimented with but was not used in due to difficulties getting it to work. While the project setup was not as elegant or succint as might be wished, it did provide a stable base to build on and a means to harness the rich resource that is the NPM ecosystem and use such tools as Paper.js and React.js.

5.3.1.2 Paper.js and react.js (paper.js bindings)

As has been outlined in the previous chapter, a declarative coding style would be employed to enable a live coding workflow and to avoid a building a codebase that is increasingly difficult to understand. In other words the code should as much as possible describe the "what" of the functionality rather than the "how". These qualities emerge quite naturally when using the *React.js* architecture. Paper.js however runs in the context of a canvas element and thusly it is not possible to directly use *React.js* with it. This shortcoming has been addressed in projects such as *three.js react bindings* and *pixi.js react bindings* which allow the use of react's declaritive programming style for 3d and 2d scenegraph oriented systems that run in the html canvas element. These solutions both work by creating dummy empty dom elements and hook into the *React.js* lifecycle events to the real work of updating the scenegraph. In many ways the scene graph structure of projects like these and indeed Paper.js exhibit a high resemblance to DOM structures and APIs making React a good fit for them. A similar approach to the above mentioned libraries approach was taken to integrate paper.js for use in SonicSketch and worked reasonably well but required quite a bit of setup. During the development of the project, a more suitable solution emerged from the open source community at an opportune time. This used the next version of *React.js* which has better support for render targets that are not the DOM and has the distinct advantage of not requiring the creation of redundant DOM nodes. The library was far from comprehensive and thusly a custom version of the library was used that included some custom functionality required for SonicSketch.

5.3.1.3 Tone.js and react.js

In some ways audio output can be thought of in a similar way to the visual output of the app and thusly can be treated in similar way by *React.js*. It can use the declarative data oriented system of react to configure the particular settings and connections in the audio graph and hook in to its lifecycle events to instanciate the various audio generating and processing web audio nodes. This addresses a notable (by design) ommission in Tone.js which does not allow the code to query the state of the audio graph once it has been setup. It is down to the userland code to keep track of this and manage it accordingly. The value proposal offered by introducing react.js into this part of the system is that it maintains the simple relationship between state and generated output. Conceptually the flow of change is:

1. The state updates
2. The react wrapper objects update their properties accordingly
3. The lifecycle events are triggered which takes care of altering, adding and removing web audio nodes (thus altering the audio being output)

The design of this part of the application is influenced by *react music*, a system that uses *React.js* with tuna.js, a web audio library similar to tone.js (???ref).

5.3.1.4 Reagent and react.js paper.js bindings

The final piece of the jigsaw in the underlying technology stack is the integration of react with clojurescript via the *Reagent* library. The core syntax of this system is simple clojurescript vectors similar to the following: This would result in the following html output: As can be seen the vectors begin with a keyword that corresponds to the tagname of the html. Additionally, instead of using html tag keywords, function calls can be made to generate html to allow for code reuse and logic. It was unclear how the paper.js bindings would work within this system due to the fact that it required a different version of react and uses non standard tag names for elements that can be drawn on screen such as "circle" and "rectangle". This however turned out to be much more straightforward than expected and the provided paper.js primitives could by simply using the relevant paper.js keywords. Complex scenegraphs could be constructed by using the following succint clojurescript syntax to describe the playback indicator:

As can probably be inferred from the code the `position` and `height` are properties that are passed into the hiccup and trigger updates to the visual display when they change: in the case of position, when the playback position changes and in the case of the height, when the user resizes the browser window. The path element describes the triangle that is places at the top of the screen.

The current state of the art in live code reloading in the browser is still not as comprehensive or as easy to setup as might be wished. Once it has been configured it is difficult to return to the compile and run workflow, and is in most cases a worthwhile investment of time. With these underlying elements in place the process of creating the core functionality application could begin and will now be described.

5.3.2 Core functionality - timeline events (or notes)

At the core of the application is the creation of timeline events which unfold in a looped fashion. These events are created based on the input of the user with a mouse or mouse like input device. On the production of a valid input gesture, the screen is updated immediately with a visual display of this content. The details of this gesture is stored in memory and the event that will eventually create the sound is registered with tone.js. Much of the events that occur in the system are captured in a main "View" component which houses the central html canvas element. To aid in organising the large amount of functionality associated with the component, higher order components are used to separate this out into logical groupings. A higher order component is a component that wraps a normal component to add functionality to it and accepts the same properties as the component it wraps (???ref-react). In this case the most logical grouping is by tool and so there are higher order components setup for each of the tools: draw, vibrato, delete, move, resize and probability.

5.3.2.1 Add timeline event

This is the default tool that is activated when the user opens the application and enables the user to add timeline events by drawing them onto the screen. It is added to the system when the "draw" tool is activated and a mouse drag operation is carried out from left to right within the bounds of the canvas element. The event is captured in the main canvas view and is initiated when the user left clicks the mouse triggering the following function:

This function receives a hashmap with a reference to a *ClojureScript* atom which can is `reset!` to contain a temporary visualisation of the newly created note. This function uses a heavy amount of javascript interop to directly instanciate paper.js objects and add them to a shared group.

As the user continues to move the cursor further points are added to the path created in the `pointer-down` function. Some constraints however are placed on the creation of the path and only points that are past the last previous from left to right are added. If the users backtracks it lead to a deletion of points, providing an intuitive undo like behaviour and implementing a recipricol HCI interaction pattern recommended by NUI principles. Completion of a note occurs when the user releases the button and triggers the `pointer-up` function: This function simplifies the path by calling the paper.js `simplify` method on the path object and dramatically reduces the amount of data captured while preserving the basic characteristic of the user's stroke (???ref-simplify-fn). Most importantly it calls the *re-frame* `dispatch` function to add the note to the app database. A `path->note` function is used to convert the stroke from the domain of euclidean space on the visual space of the canvas to the domain of time-pitch space for use with the audio synthesis system. The path->note function can be seen below: The domain of time-pitch is used to store the notes in memory and makes it possible to maintain a relative relationship between the screen size and the drawn notes.

The dispatched note event is handled by a *re-frame* `reg-event-db` handler which describes the alteration that is required to be made to the database. It also uses a series of interceptors, to perform validation of the database and to remove some of the boilerplate from the event handler functions. Interceptors are similar conceptually to middleware and is the place where all of the side effects arising from an event are actioned. Moving application side effects to this placed ensures that they are isolated and means that the majority of the program can be kept as pure logic improving to make it easier to test, debug and reason about. As can be seen the handler function is very simple: This does a simple check to make sure that note has a minimum duration and if so alters the notes vector to include the new vector which will instruct *re-frame* to update it's internal atom with this new state.

The structure of the note hashmap is defined using *clojure.spec*, a core library to perform data validation and a tool that may be used similarly to types in strongly typed languages. The note specs are defined as follows: Although note

specified here, notes also have an `envelopes` key that stores frequency and vibrato envelopes. For example: The update in state spins *re-frame*'s subscription system into action and any views that are subscribed to the application state are now re-rendered as needs be. The `graphics-notes` view for instance is subscribed to `:notes`, `:graphics-stage-width`, `:graphics-stage-height:` and `:mode`: When a note is added this render function will be ran which will call the `[graphics-note*]` component for each of the notes.

5.3.2.2 Add vibrato

5.3.2.3 Remove note

5.3.2.4 Move note

5.3.2.5 Resize note

5.3.2.6 Change sound (preset system)

5.3.2.7 Probability tool

5.3.3 Secondary functionality

5.3.3.1 Introduction

5.3.3.2 Transport controls

5.3.3.3 Animation (current play position & notes)

5.3.3.4 Undo and redo

5.3.3.5 Fullscreen

5.3.3.6 Outer UI

5.3.3.7 Save and load file

5.3.4 Performance issues

5.4 Conclusion

- Summarise the resulting artifact

6 Evaluation

6.1 Introduction

6.2 Initial pilot test

6.3 Exhibition

6.4 Conclusion

7 Conclusion and further work

7.1 Summary of work completed

7.2 Broader implications of development

7.3 Future work

7.3.1 Performance improvements

7.3.2 Broaden visual language

7.3.3 Allow for larger structures

7.3.4 3D spaces, VR, spatial audio