Introduction

This chapter outlines the approach that will be taken in the realisation of the SonicSketch application. An assessment of SonicPainter will be given to identify the features that will be incorporated, improved on or omitted. Some consideration will be given to theoretical aspects that will underly the design and development of the software. The major technologies being used in the project will then be introduced along with some justifications for their usage. These include the *Web Audio API*, *Tone.js*, *Paper.js*, *ClojureScript* and Facebook's *React* framework.

Assessment of SonicPainter

As has been mentioned previously, the SonicPainter application developed by William Coleman would form the basis of the work on SonicSketch. While the basic design of SonicPainter works very well and is aligned with the goals of SonicSketch in the following ways:

- Provides a simple immersive score space that represents time on the x-axis and frequency on the y-axis.
- Allows multiple voices or timbres to be represented in this space so that all notes can be seen at a glance without having to click into separate tracks.
- Visualisation of vibrato effect using overlaid sine wave is suitable in theory but perhaps could be implemented better.
  - Uses FM synthesis as the synthesis engine. This versatile and efficient is a good fit particularly for use in the web browser where performance can be an issue.

Some issues that have been identified with SonicPainter will now be addressed. It should be noted that as the software is only at a prototype stage some of the shortcomings outlined here may not be design decisions and instead may be implementation issues.

Usability issues

While the application is conceptually straightforward and beginner friendly there are some stumbling blocks that would make it difficult for a new user to begin using it. Starting a note is obvious and involves simply clicking on the screen to anchor the start point. A line now becomes attached to from this point to the mouse pointer location, indicating that another press will finish the note. However, this instead adds a point to the note, indicated by the fact that a new line appears between this new point and the pointer position. It is unclear how to finalise the note. The solution is to press the **shift** modifier key and click. Some simple labeling could alleviate this issue, by perhaps using a small toggleable information box. Unfortunately, this issue repeats itself in a good deal of the functionality with a good deal of additional features hidden behind keyboard shortcuts that would be difficult or impossible to discover.

The intention of SonicPainter and SonicSketch is to create a tool for musicians. Some basic usability additions greatly improve its potential utility in this regard. With this in mind, some additional features would be added to improve usability. This included an undo and redo a feature that can help greatly to increase the confidence of using the system. Save and load functionality makes it possible to resume sketches and goes some way to making it a more viable tool for musicians. In addition, the editing capabilities to existing content would be greatly expanded on. Basic editing operations such as moving and deletion would be added. Some other basic features that would be added are tempo adjustment, to allow the user to slow down and speed up playback; velocity adjustment, to make notes play at varying amplitudes and an adaptive sizing system to allow the application to use the maximum screen real estate.

Issues with audio visual connection

Some deficiencies were found with the mapping between visuals and audio. It is possible to draw multiple concurrent notes that share the same timbre suggesting that the audio would polyphonically sound the notes together. This is not the case however and only one note can be played back at the same from the same timbre. To achieve this, a separate voice must be used by clicking the up arrow. To remedy this issue, it could either prevent the creation of concurrent notes visually or allow multiple notes to sound together by making each voice polyphonic.

Invalid states

A serious technical issue with SonicPainter is the presence of invalid application states that are not properly handled in the code. These arise when the user performs a series of actions that are not properly handled either by catering for them or by disallowing them. An example of this is when the user starts a note and then changes mode to, for instance, vibrato. When this happens, the application allows the mode to change and carry out the vibrato change all the while keeping a line connected to the pointer position. From here it is not possible to complete the note without returning to the original mode and clicking multiple times.

This unexpected behaviour could be avoided by either disallowing mode changes during note creation or by ending the note when the mode is changed. These type of issues are common even in commercial applications as is illustrated in Harel (???). An example he gives is that of a calculator built by Microsoft that enters an invalid state in a number of different situations. To counter these issues a disciplined approach would be taking to managing state in SonicSketch by performing *validation* operations and the use of *finite-state Machines* (FSM). Validation is a technique used to ensure that data is in the correct format and that any required information is present and in an allowable format. A finite-state machine is a computational model that can be in only one of a finite number of states. These states may change due to external input which triggers transitions from one state to the next. They can prove useful in avoiding invalid states and to reduce the amount of conditional logic in the program.

Linear representation of frequency

The SonicPainter represents a frequency range of 20-500 Hertz (Hz) on its Y-axis. This is mapped linearly from the on-screen coordinates to frequency values. It is not made clear why this range and mapping was chosen and may have been for aesthetic reasons. Psychoacoustically it may be suitable to map this linearly as this corresponds to the linear perception of frequency at the lower frequencies. This phenomenon is represented in the Bark Scale, which is a psychoacoustic frequency scale on which equal distances correspond to perceptually equal distances. However, the range of 20-500 Hz frequency is quite limiting given that the highest note of a piano can reach to 4186 Hz. In addition, there is a strong cultural expectation of logarithmic mapping of frequency owing to the semitone being a logarithmic scale.

NUI considerations

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 mimicking the real world tool or instrument (Wigdor and Wixon, 2011, p. 13). Be friendly and learnable by beginners but allow for mastery given enough practice (Wigdor and Wixon, 2011, p. 13). Immediate feedback for all interactions should take place, most usually but not limited to, visual feedback. (Wigdor and Wixon, 2011, p. 87) Furthermore, the interface should take advantage of the particular affordances offered by the input method. (Wigdor and Wixon, 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 (???). 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 leftmost diagram of the provided figure (fig. 3).

Audio in the modern Web Browser

Tone.js

Tone.js is a Web Audio framework that provides several helpful abstractions and libraries to help interaction with the Web Audio API. A central aim of Tone.js is to enable some of the conveniences of DAWs and is formed on three tenets: musicality, modularity, and synchronization. An example of this is the flexibility it allows to express time values, eg. "`4n`" for a quarter note in metrical notation, `1` for a second and "`100hz`" to express 100 hertz, etc. These are all converted to seconds before scheduling them with the Web Audio API. A system called "just in time scheduling" ensures that tempo relative values are not scheduled until the latest possible time, thus ensuring that they reflect the latest tempo value.

The primary reason that Tone.js was chosen was for the signal system that it uses to make working with parameter modulation easier and more efficient than working directly with Web Audio API. This uses a constant signal generator running at audio rate connected to a GainNode. tempo-relative In addition, this value can be multiplied and summed using GainNodes native capabilities enabling performant signal processing operations on all parameters. For instance, *FM synthesis* generally requires that a relative relationship is maintained between the two oscillators, the carrier and the modulator. When directly using the Web Audio API Param, a *processor node* would need to be setup to calculate this which is not as efficient or as straightforward as carrying out the calculations using Tone.js signals. A processor node allows developers to run arbitrary javascript on audio signals which provides some flexibility at the cost of performance.

Paper.js

Paper.js is a descendant of Scriptographer, a scripting tool for Adobe Illustrator a vector graphics program. It runs in the web browser canvas element, running in its 2d mode. Paper.js adds a number of features to the browsers native canvas element including a *scene graph*, geometry and vector abstractions as well as tools to draw and animate these shapes on-screen. The central abstraction in Paper.js and any vector system is the *path*. This allows for the specification of any shape by describing start points and endpoints for a series of paths. Curves can be added to these paths by adjusting an additional set of points associated with a path, the `handleIn` and `handleOut` points. These define control handles that alter the curvature of the line using Bezier mathematics.

A primary reason that Paper.js was chosen for SonicSketch is the path simplification algorithm that allows the data captured with freehand input to be simplified and smoothed. Instead of just plotting every point captured, an optimized subset of these points is used to reduce memory usage and speed up drawing. It is based on an algorithm by Philip J. Schneider published in Graphics Gems (1990). In addition, these simplified paths were more suited to mapping to control data for the audio system.

FM synthesis

React framework

React is a web framework built by Facebook that aids the developer in updating the document object model (DOM), a process that is required when the state of the application changes. This was a role traditionally carried out by the web server and delivered to users as a static page. This saw a significant change however with the rise of single page applications (SPA) around 2010. The advantage of the SPA is increased interactivity and responsiveness to user input, allowing the look and contents of the page to update dynamically as the user interacts 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 a 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 binding system using what is termed the virtual dom. in this model a special virtual version of the dom is constructed and when the model changes. 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 particularly beneficial when paired with *functional programming* techniques, a style of programming that encourages 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 these benefits of the react model beyond the realm of the DOM. This includes writing console programs (???ref), writing web audio applications (???ref) and even Arduino (???ref).

Clojurescript

Clojurescript is a compile to javascript programming language that is based on Clojure, a modern *Lisp* that runs on the Java virtual machine. Lisp is a programming language that was invented by John McCartney in the 1960's and is known for its minimal syntax consisting primarily of parens. The word Lisp is derived from the term "List Processor" as Lisps source code and data structures are built around lists. Clojure and ClojureScript promote a functional programming style, a style of programming that encourages the use of pure functions as the core building blocks of programs. Clojurescript and other functional programming languages such as *Elm* have seen an increase in usage in the past number of years as this paradigm has proved useful in managing complex stateful UIs. Some annotated examples of ClojureScript follow that show the basic building blocks of the language and will help in understanding the code walkthrough in the next chapter.

Reagent & Re-frame

Reagent is a library that provides an idiomatic ClojureScript interface to React, allowing ClojureScript to harness the DOM manipulation facilities provided by React. This delegate the *side-effects* of rendering and manipulating DOM to React's reconciler algorithm. Side-effects is functional programming term to denote anything that isn't related to the supplied arguments or return value of a function. This is normally object mutation (to change the state of the program) or input/output (I/O) operations, e.g. writing a file to disk, displaying graphics, playing a sound. [**?**] In addition to the interface to React, it provides a special reactive atom that efficiently re-renders React components when state changes.

Re-frame is a framework that uses Reagent's interface to React to manage views and it's reactive atom to manage state. It proposes a program architecture consisting of the following 6 elements:

1. Event dispatch
2. Event handling
3. Effect handling
4. Query
5. View
6. DOM

The majority of events that are dispatched are due to users interacting with the system (for instance by clicking with the mouse). Event handling is the code that is run in response to these events. Re-frame proposes that these event handlers should supply data to describe the side-effects that should be made to change the application. Re-frame carries out this work which is typically to update the application state stored in a single reactive atom and managed by the framework.

A subscription system allows the view system to update when the state that it depends on changes. Finally, React updates the DOM to complete the cycle.

Conclusion

SonicPainter was discussed in some depth