Nigel Powell - CM3070 Final Project - Final Report

Introduction

CODE REPOSITORY LINK:

https://github.com/sadsongco/JRGShow

My final project is based on the CM3065 Intelligent Signal Processing VJing System Template to create a "VJ engine(s), which are used by VJs to create or manipulate imagery in realtime through technological mediation in synchronization to music". This template defines an app with these characteristics:

- Loads video files, audio files, or a combination of those. It can also use a camera input and audio input as a video and audio source.
- Includes a modular node-based patching environment (similar to a graph flow language such as PureData) to process, mix and generate audio and video materials. Audio input and camera input can also be processed in the patching environment.
- Has separate control panel and visualiser windows so it can be used with a projector

I am shifting the focus of the app from pre-rendered or generated visuals to the processing of live camera feeds, for display in large formats, either via projection of LED screens, with a focus on performers and 'rock and roll' concerts, rather than DJ events, with the application being used by a lighting designer rather than the performer themselves. My intention is NOT for this to be a commercial prospect, but the focus is on it being a usable creative project. The main user base will be me, although I will seek input from other lighting designers in the rock and roll world.

Requirements

Working from the template requirements above, the stated requirements for this project are:

Requirement #	Description
U01.0	*processing video and audio sources chosen by the user*
U02.0	*selecting from a range of modules that will process the incoming video*
U03.0	*adjusting a selection of parameters for each module to allow a wide variety of different effects to be achieved*
U04.0	*modulate chosen parameters by selected characteristics of the incoming audio*
U05.0	*modulate chosen parameters with time-based fluctuations*
U06.0	*create chains of modules to allow a wide variety of different visualisations to be achieved*
U07.0	*save a module chain as a 'song'*
U08.0	*arrange the saved 'songs' into a setlist*
U09.0	*save and load setlists to / from disk*
U10.0	*display visualisations of the songs of the setlist in order in a separate window, with smooth controllable transitions between items*

Requirement #	Description
U11.0	*a clear easy to use UI for creating songs*
U12.0	*a clear easy to use UI for running the show*
U13.0	*maintain reasonable frame rates at the chosen output resolution while running the show*

In addition to these requirements, which can be grouped under the 'user requirements' definition, there are additional 'developer requirements':

Requirement #	Description
D1.0	*produce a piece of work that can achieve a satisfactory academic grade*
D2.0	*undertake an informative educational journey*

The user requirements are linked to technical requirements which are omitted to conform to page limit requirements of the report.

Literature Review

Live performance has become perhaps the most important aspect of most musician's careers from an economic standpoint. As far back as 18 years ago, it was noted that for rock concerts "average ticket price increased 82% from 1996 to 2003, while the Consumer Price Index (CPI) increased 17%" with evidence that "New technology that allows many potential customers to obtain recorded music without purchasing a record"[1] had produced the upwards pressure on these prices to replace lost revenue from recorded music. This has led to the situation that "performers generate around 75% of their income from live events and tours today, compared to less than one third in the '90s."[2]).

The importance of effective live performance also drives consumption of an artist's recorded music, with a study around multiple years of a Norwegian music festival showing a "clear spike in the streaming of festival artists during the festival week each year, compared to weeks before and after"[3]. This chimes with studies postulating that experiencing live music allows audience members to feel they are "connecting with the artists and experiencing the potential for spontaneity and unpredictability of live music as it unfolds over time", with results of measured head movements indicating audiences become more engaged and involved with music through live performance[4].

Lighting as part of a concert performance can further enhance the audience experience - "well-lighted environments produce an emotional response in the occupants that is positive and healthy" and "Elevated brightness has been associated with increases in an individual's heart and respiration rates" [5]. Alongside established lighting, "Visual imagery and the mobile components that enable the lighting designer to create and manipulate these images have become staples of most touring concert lighting systems"[6]. At concerts in larger venues, screens showing live camera footage of the performers (commonly referred to as IMAG, a contraction of 'image magnification') will provide an intimacy that might be lacking, but with the drawback that constant utilisation of IMAG can produce a visual sameyness that may become boring.

All of these drivers create a need for distinctive and creative visuals. The ubiquity of video projection / video walls has led to a ubiquity of the existing solutions (discussed below) and the functions they provide.

My project attempts to find creative ways to treat IMAG visuals such that the intimacy can be retained through extensive use of performer footage, while finding ways to maintain visual distinctiveness from song to song, and interest over the duration of a concert. The increase in price of concert tickets noted earlier may have resulted in a higher expectation of the visuals presented as a show - "A concert is an "experience good," as consumers do not know the utility they would derive from a concert unless they go to it. As a result, image and reputation are very important … especially if performers are trying to build a long-term audience"[1]. Merely presenting an image mixed camera feed would not add to perceived value. At the same time, the above-mentioned ubiquity of the industry standard solutions and the 'looks' they produce requires being able to dig deeper into experimentation to be able to achieve distinctive visuals.

My research indicates that academic studies into integration of visuals and concert lighting are not common. Perhaps this is to be expected - in the same way that moving lights were a practical concern where the rock and roll touring industry had "taken an existing technology and improved it tenfold"[6]long before it was adopted by theatre or TV, so the business of finding ways to present creative and engaging visuals has mostly been the remit of artists and lighting designers rather than academia to this point. However there are plenty of studies dealing with the psychological effects of lighting itself which I can refer to while developing the processing effects for this application - as Dunham notes, "After intensity, color is the next most likely element of a lighting design to shape the mood"[5], so even a simple tint on an image may be able to manipulate an audience's emotions in service of their reaction to a performance of a given song.

Existing Solutions

There are already existing solutions for concert focussed media servers, "a computer dedicated to the task of organizing and playing back video and still images, controlling aspect ratio, color, cropping, audio output (and much more) during a performance or presentation" [6]. There are no products that I currently feel fulfil my artistic requirements. Existing media servers are often too expensive, over-engineered and not sufficiently open to allow customisation and creative mangling. Max is a good solution, but is over-complicated - while I would be able to come up with exciting things working on my own without time constraints, within the limited period of production rehearsals with an artist I want to be able to combine and customise effects much quicker so I can be dynamically reactive to creative suggestions, while also having a clear UI for ease of use in the pressure situation of a performance

---

Catalyst Media Server

There are a number of media servers available, with broadly comparable features. I am focusing on the Catalyst media server[7], but much of the reasoning would apply to other solutions such as the Hippotizer by Green Hippo[8] or Resolume[9]

Pros

- an established industry standard, road proven on thousands of shows in hundreds of tours
- a powerful system that can deal with many layers, video streams and hi res outputs
- carried by rental houses worldwide, so a show designed on this can easily be recreated in other touring markets

Cons

- requires racks of dedicated hardware (fig 1) and either a dedicated expert crew member, or training
- unnecessary features for my proposed application, for instance it can work as a full vision mixer, functionality I wish to be offloaded to a separate system, so the lighting designer can concentrate on the image processing

- inflexible in creative image processing. Catalyst includes a standard set of filters and processing options, but the ability to go in and really mangle visuals down to a pixel / bit level is not there, something I wish to be encouraged in my application
- costly - the dedicated hardware puts it out of the reach of smaller tours / shows, whereas my application will run on a laptop and requires no extra financial outlay

---

Chamsys MagicQ

Similarly to the Catalyst, there are a number of lighting consoles that integrate some video control. I am concentrating on the Chamsys products[10] but the reasoning applies to offerings from GrandMA[11] and Avolites[12] as well.

Pros

- complete integration with lighting control. This is an area where these solutions are and will always be superior to anything I can do, since changes in visuals can be programmed as part of the show and operated alongside the lighting for perfect synchronisation
- familiar paradigm for lighting designers. Even though I want my application's UI to be clear and understandable, all lighting designers are comfortable with the operation of desks like these, and incorporating visuals is more natural for them in this way
- stable and mature software. A great deal of battle testing of all of these systems has happened, and is ongoing

Cons

- hardware requirements. While Chamsys provides their MagicQ software as a free download for PC and Mac, in order to integrate it with a lighting rig or screen some extra hardware is required, and in a professional situation a full desk would be expected (fig 2)
- out-of-the-box can only play pre-stored media on disk. My application is focussed on processing IMAG, which is something that these systems would require an external media server to be able to do
- very little processing customisation. In similar with Catalyst, experimental and creative video processing is not readily possible

---

Max / Jitter

Max[13] is a well-established signal processing graphical programming language, which now includes video and graphics tools with the Jitter expansion. This system was employed by lighting designer Leroy Bennet for visuals on Nine Inch Nails tours he designed[14][15]

Pros

- powerful object-based programming system for signal processing
- a well-developed graphical interface (fig 3)
- very flexible - allows for extensive experimentation and mangling in many of the ways that I would like my application to be able to

Cons

- originally built for audio processing, so video manipulation can be a little fussy
- flexibility can make for impenetrability. Max / Jitter's extraordinary power means there is a fairly significant learning curve, which can make it hard to understand, or extract the result you wish. My application will focus on specific processing modules in the visual domain to simplify operation
- not naturally focussed on the LD / setlist UI paradigm. While it's possible to customise it to follow the idea of a setlist of setups, it doesn't naturally fit into that. I would prefer my application to suit that simply and immediately, and fit LD expectations

Project Design

Domain

The domain of the project is visual arts and representation of music. My intention is NOT for this to be a commercial prospect, but the focus is on it being a usable creative project. The main user base will be me, although I will seek input from other lighting designers in the rock and roll world. However you could consider any artists for whom I do lighting to be secondary users of the product - while they won't have any direct interaction with the software, and therefore won't need to be considered in UI testing, they will be artistically represented by the output of the application, such output being decided mutually during a design consultation phase prior to any performance / tour, and dynamically during production rehearsals or tour dates.

Application Architecture

Fig 4 gives a diagrammatic overview of the architecture of the application, and fig 5 is the proposed way the main scripts will interact to provide the application functionality. I produced wireframe mockups of the various pages so I would have a UI reference to work from (figs 6 - 10).

Application Structure

As specified by the template requirements, I am going to build the application in a web browser, using HTML, CSS and Javascript. One of my goals is in performance, and this is an area where the tools mandated may prevent the application running as fast as I would like on full-res IMAG video. In the future I may look in to porting it to a compiled language to increase performance.

Proposed Software Implementation

The UI and application output will be built in HTML with CSS. The application logic will be written in Javascript, with the image manipulation taking advantage of the HTML canvas and built in browser or system camera / screen access. I will likely use the `p5.js` library to abstract some of the canvas functionality.

The main technological extensions from work we have done in the course already will be:

- using Broadcast Channels[16] to facilitate communication between the 'run show' and 'create visualiser' pages, and the visualiser output window. The visualiser output will be its own separate JS script, which will have its behaviour modified by Broadcast messages, returning its status via Broadcast channel for UI feedback to the user
- using Web Workers to allow threading of processor-intensive pixel manipulations, hopefully to improve performance.
- For the storage and retrieval of settings, local storage will be used via the IndexedDB javascript API[17]

Outside of `p5.js`, the creative / circuit bending focus of the visualisers means they will be built in vanilla js. It may be necessary to use a library for the Setlist Visual Creator (fig 8). Similarly, the presentation of the prototype will be built using vanilla CSS, but if it

becomes necessary to streamline development later on then a framework such as Bootstrap may be employed.

Potential Problem Assessment

In common with most image manipulation approaches, I will be "carefully considering a problem at the level of pixels and patches, and specifying the requirements in the most direct possible way"[18]. This does present a potential issue for me however. In order to output to large format screens (projection or LED), high resolutions will be required, 1920 x 1080 or more would be preferable. JS may be unable to sustain high frame rates while processing pixels at that resolution

While JS parsing takes "an average of 23% of the whole app loading time"[19] I don't anticipate this will be a bottleneck for my application - the parsing of it will take place on startup. Processing of raw frame data in real time will be much the greater challenge, so I am hoping that I will be able to make use of parallel processing: "Web Workers API allows multiple JavaScript codes to concurrently run in background threads"[20].

Outside of these considerations, I am intending to use the `p5.js` library as an abstraction layer on top of the HTML canvas, since it has been taught extensively in this course. During development I will compare performance with native JS / canvas to see if improvements can be had there. Some of the visualiser modules are intended to include reactivity to audio input. For this I will use `p5.js` inbuilt audio functionality for capturing and filtering / processing audio. If any more extensive processing is needed I will employ the `medya.js`, but initially I don't anticipate this being necessary.

Schedule and Testing

Workplan

See fig 11 for a Gantt chart of my proposed work plan, with explanation below.

Code Experimentation
Sandbox ad hoc implementation of various ideas, without having to integrate into a larger whole. Free-ranging and unstructured by nature. No unit or integration testing. No user testing.

Prototype
A basic working visualiser. This will include the 'setlist' functionality, but none of the UI features for creating visualisations or setlists. Various visualisations will be built, and a modular system for including and routing them in code, which can later be addressed by UI creation. The two window / user control paradigm will be built. Basic unit testing included using jest where possible, or ad-hoc where not (see Functional Testing below).

Codebase Iteration 1
Later codebase sprints will be flexible depending on previous progress. Intention:

- initial sandbox versions of filesystem and UI, independently unit tested, but not integrated
- update basic presentation UI of main application
- add basic settings landing page

- continue to develop creative visualiser modules

Codebase Iteration 2

- integrate filesystem and UI into main application
- integration testing
- offloading processing to web workers, with performance testing
- continue to develop creative visualiser modules
- deploy a version for user testing

Codebase Iteration 3

- complete application for final deployment
- bug and stress testing
- investigate viability of deploying as a Progressive Web App
- final user testing

Functional Testing

Unit testing of the application should use Jest[21], and through ad hoc TDD, where small amounts of code are written and tested at a time before moving on. The performance aims of the project will be monitored by a debug mode that will display a framerate for the video output. If I ensure that the application is tested with my target full-res (1920 x 1080) then this figure will provide a metric for assessing this aim.

User Testing

Even though the application is only intended to be used by me, I will deliver working prototypes to other professional lighting designers for their ad hoc comments on the UI, to prevent my myopia locking me into bad design patterns. One of the aims of the project is ease and rapidity of use, and gaining insight from other people in the same field may yield strategies I would not have been able to think of on my own.

As a creative project, requirements testing is a little more esoteric, and will need subjective judgement as to whether a visualiser is achieving the artistic aims. This will largely be down to my judgement, although I will also demonstrate the functionality of the application to professional performing musicians for their comments. The thrust of the project however is to build something that I can customise quickly in response to comments and ideas, which could easily involve quickly coding new visualiser modules to accommodate an artist's needs. Assessing whether this is something the application provides will inform the success of the creativity and flexibility aims.

Ethical Assessment

There are very few ethical concerns associated with this project.

- the project does not require disclosure of any private information by any user, so there are no identifiable privacy concerns
- the application will require no connectivity to the internet, further eliminating privacy concerns and negating net security concerns

- user testing will take place within a select group of professional people who are already known to me, with confidence that there is no exploitation of a vulnerable person or group. I will ask for their permission before enlisting them into testing
- since the identified primary user is me, I am happy to grant myself all necessary permissions for how the software may be used
- the secondary users will be musical artists who have hired me to create visuals for them, and the creation of the visuals will be in full consultation with them
- with the focus of the visuals being IMAG, there are no potential copyright breaches that may occur with pre-rendered / filmed material
- HOWEVER if any text is used in the text-generating visualisers, this should be ensured to be used with the proper permissions. As a real-world case study, I deployed a prototype version of this software at a show by the artist Jonny Greenwood, with the performance including many works he has composed for feature films[22]. I would have liked to include excerpts from the screenplay of one of these movies, but that material is copyrighted so was unavailable. Similarly the studios who own the rights to the filmed material have in the past made it clear that this artist cannot use clips from those films in his live performances without seeking (and likely paying for) specific permission.

Implementation

Algorithms, Techniques and Methods

Prototype

The application at prototype stage uses Javascript modules for a clear file structure (fig 12). The two main components were:

`pageManager.js`:

- provides a screen for control of the application
- provides visual on some aspects of the application
  - e.g. whether the visualiser is ready to fade in a new visualiser or if it is currently busy, indicated by text background colouring

`visMain.js`

- grabs pixel data from the video input, processes and outputs to window, using the `p5.js` wrapper on the HTML canvas
- reports status back to control screen

which communicated via the BroadcastChannel API, with common parameters stored in `setlist.js` and `visParams.js`, and in the `prepareVis.js` module (fig 11).

First Iteration

The architecture was developed from the prototype to follow fig 4.

For persistent storage of settings, setlist items and associated visuals I used the IndexedDB API, replacing some of the Broadcast Channels functionality and reducing data passed between windows. Some scripts required asynchronous database operations, so I used the OpenDB asynchronous wrapper for the API[23].

Visualiser chaining was overhauled with visualiser modules registered in `scripts/modules/visualisers/registeredVis.js` imported as a javascript Object of modules (`scripts/modules/common/importModules.js`, fig 22). Visualisers were rewritten as Classes inheriting from a custom Visualiser class, which allowed free chaining and saving of setlist item visualiser chains. Each visualiser class also provided adjustable parameters that can be passed to the visualiser each frame as keyword arguments.

For a computationally cheap pseudo-random bit array I generate a large random integer and convert it to a binary string once per frame (fig 25). Each frame the `dynamicGenerator()` function produces an array of values drawn from frequency offset sin signals dependent on the frame count (updated in a later iteration to a timestamp so the modulation would be consistent in creator and runshow).

Second Iteration

To better control asynchrony I rewrote the `VisOutputEngine` class to use the Canvas API directly, and abstracted the setup and management of the Web Audio API to the `AudioEngine` class, which created a media stream from the specified audio input device and created a frequency analyser to use with it.

I experimented implementing Web Workers in two ways.

- passing sliced pixel arrays for the current frame to each Web Worker for processing
- splitting the canvas into equal sized sub canvases for each Web Worker and control of this canvas being passed to the associated worker using the `transferControlToOffscreen()` Canvas API method

After performance was compared (see Evaluation) the second approach was pursued.

Third Iteration

For layering visualisers while pixel processing I implemented the `alphaBlend.js` module (fig 31).

The hub page initially used the draggable attribute[24] of an unordered list to sort setlist songs into the desired order, this was updated to use the Sortable library[30].

Code Explanations

Prototype

User interactions are abstracted to the `controlKeyEvents.js` script. `pageManager.js` passes all key events to this script, which processes them, updates parameters as necessary, and then passes any necessary messages to `visMain.js`. (fig 13 & 14) using the BroadcastChannel API.

In fig 13, the `controlKeyEvents.js` script subscribes to the `vis-comms` BroadcastChannel. When a keystroke of 't' is detected, it posts a message into the BroadcastChannel containing information for the `visMain.js` script. In fig 15, `visMain.js` reacts to messages received on the vis-comms BroadcastChannel.

`prepareVis.js` is passed the id of a setlist item and activates the required visualisers. As an example visualiser, a bitwise module is shown in fig 17.

First Iteration

`importModules()` imports the registered visualisers in `registeredVis.js`, then iterates through them using the asynchronous dynamic import syntax[25] to read in each visualiser module. It then instantiates each visualiser class and stores it in an object.

The `init.js` script called from the `index.html` page sets up the IndexedDB, creating object stores for output resolution, video input device, audio input device and setlist if they don't already exist. `navigator.mediaDevices.enumerateDevices` provides user-selectable options which are saved to the IndexedDB (fig 23) before continuing to the hub page. These settings are retrieved using the OpenDB async wrapper to ensure the values are available before the canvas is created. The setlist is also retrieved from the database asynchronously for scripts that require it (fig 24).

Second Iteration

For workers the dimensions of each sub-canvas are calculated from the size of the output canvas in the `VisOutputEngine` class and the defined number of workers. A `subcnvParams` attribute stores which part of the canvas each worker is working on, so

the main thread can extract the necessary data from the video input to pass to the worker each frame. Separate `drawStart` and `drawHeight` values were needed because I was overlapping the frames by a specified amount, so pixel transformations using a convolution matrix would still be able to work without making the sub-canvas boundaries obvious. Initially each worker was invoked for each frame returning a promise when completed (fig 28), with the main thread using `Promise.All`, to ensure all workers and canvases were ready before the draw loop began. (fig 30), though this was changed later as noted in the Evaluation.

Canvas processing was extracted to the `ProcessCanvas` class in `canvasWorker.js`. This class was also provided as an export so the `VisOutputEngine` class could access it for testing in the main thread. (fig 29)

Tooltips were added for visualiser parameters, enabling extra clarity and instruction to be given if necessary by hovering over the parameter name. It used a very elegant CSS-only implementation, using the data attribute of the element (fig 37). The text of tooltips was set in the `parameters` attribute of each visualiser class, and included dynamically when the parameters are displayed on the creator page.

Third Iteration

To enhance performance I found an approximate luminance calculation that was much quicker[28] (fig 27). Destructuring parameters in a visualiser method definition (e.g. `function foo ({ param1, param2, …kwargs}) {…}`) was much slower than assigning to variables within the method[29], so this was changed (fig 33).

Many floats needed rounding, but the application did not have a need for mathematical precision, so bitshifting by 0 places was used as a computationally cheaper method that `Math.floor()` (fig 34).

I implemented a noise visualiser, using javascript `random()`. To compare performance I wrote my own simple C++ functions, compiled to Web Assembly using Emscripten, to generate random grey or RGB values (fig 35).

Evaluation

Functional Prototype

Git commit: 'Final version before EOTR' 8694b896b4711acf2dddaaa9efc2f951162e71c7

Testing

There was no functional user testing with this version. Creative user testing was undertaken by employing the prototype for a performance with an artist, and involving them in the approval of the visuals being produced. Robustness testing took place under professional conditions, working reliably for a 1 hr 15m set headlining a festival.

Requirements testing

It allowed for creative processing of video input, and was controllable by a lighting designer alongside a show using the paradigm of a setlist of songs with associated visualiser settings. [requirements U04.0 - U8.0, U10.0]. Performance on 1920 x 1080 video was sometimes patchy, with frame rates falling well below 20 for processor intensive visualisers such as edge detection. The code architecture was very inflexible, and the visualiser chains were not easily edited. Selection of video and audio input devices, and the output resolution, were hard coded

First Iteration

Git commit: 'Test Version 1.0 permissions problem solved' 9ca14510014298edc7b91cd7ce5be3cec25552af

User Testing

The application was deployed to a web server along with an entry page outlining the intention and use of the project[1], which linked to the application and a Google questionnaire. This was circulated to friends who are professional lighting engineers, and posted to the UK Touring Crew Facebook group to invite users to provide feedback. At this point the visualisation side of the application had been scaled back to concentrate on the underlying code architecture and the user interface, so the questionnaire focused on usability issues to interrogate how intuitive the operation was. Unfortunately only one respondent made use of the questionnaire (fig 26 i-v).

The single helpful respondent also provided some comments on the interface and their use of the application:

> "There were some parameters missing for some effects, not sure if the are supposed to be there or not. If no parameters are available for a certain effect it helps if it says 'no parameters available'. Main issue I had was having to select the main window again to be able to start and stop the show playback, as I was running my output window on the same screen. A really interesting bit of software which other than actually playback, I found really intuitive."

[1] https://thesadsongco.com/UoL/usertesting1.html

I considered attempting to recruit a wider cohort, for instance from the student cohort, but decided that since the application was aimed solely at lighting designers this would not provide feedback that would inform that goal.

Functional Testing

The debug option displays the frame rate while the visualise is in operation. In the small preview canvas this was running at 60 fps, but this dropped when running the show in a full size window. For full HD, the frame rate for intensive visualisations dropped to 3 or 4.

Since the application was written exclusively in javascript / HTML / CSS, integrating unit testing (such as using Jest) was not straightforward, and I deemed attempting to develop using a TDD paradigm would impact upon my progress. In development I was sure to write very small amounts of code before saving and testing / observing functionality. This will not pick up potential fatal edge-cases that proper unit testing would identify, but I chose progress over robustness.

Requirements Testing

The application now fulfils the requirements for user selection of input and output [U01.0]. It provides a UI for creation of visualisers [U02.0, U03.0, U11.0, U12.0]. It allows for saving of songs under their name and organising them into a setlist for performance. [U09.0].

Second Iteration

Git commit: 'Code tidy up' 1bf3480e9768bb823d15e222749b21618c495527

User Testing

I didn't achieve everything I wanted to in this iteration due to the issues encountered, but I am ahead of schedule so some can safely be moved into the next iteration. I didn't test this version with any outside users because all of the changes were in code architecture.

Functional Testing

Transferring control of sub-canvases seemed the better approach. It allowed better code encapsulation, moving all of the visualiser processing routines into the `canvasWorker.js` script. When using this with the preview canvas on the Create Visualiser page it worked as intended. However as soon as it was opened in a full-size window from the Run Show page, it would run at fps of approximately 0.03, and eventually crash with an Out Of Memory error. After a ground up rebuild I discovered this was because I hadn't properly promisified the workers - the main thread was running at 60 fps, invoking new worker jobs each frame, but the workers were running much slower, meaning they were backing up with pixel arrays etc. I solved this by calling `requestAnimationFrame` from the worker with a callback to post a message back to the main thread, indicating it had completed drawing the frame. When all workers had reported completeness using `Promise.all()` the main thread could then call `requestAnimationFrame` on the main loop.

Requirements Testing

A benchmark visualiser chain was created, and the frames per second achieved by the visualiser measured for the different approaches (all processing in main thread, web workers processing passed pixel arrays, web workers using sub canvases under their control). The results are in fig 32, showing that the latter approach was the most performative, and that there was not much performance gain beyond 4 web workers. Subjectively, going beyond this caused the fans in my test machine to spin up very quickly, indicating that it was putting the processor under strain. [U13.0]

I tweaked some of the code to try and improve performance (e.g. by converting array iterations to their most efficient form[26]) but didn't achieve any particular advantage. It was possible that I may have achieved better performance by using WebGL and moving the processing onto a GPU, but this would require a major overhaul which would likely involve not finishing the project in the given timeframe.

Third Iteration

Git commit 'Submission version with documentation' c00aad858359b2bddae10b36709b8ba1a161b398

User Testing

The same user testing scenario as iteration 1 was used[2], with twice as many respondents. Results are shown in fig 36 i - vi, with 1 being easiest and 5 being hardest. While the cohort was still small, it did indicate that development had moved in a successful direction, and that the interface was intuitive for a lighting designer to work with [U11.0, U12.0]. I was very satisfied with the ad-hoc post-testing comments (included in the fig), which also included some interesting ideas for future development.

Functional Testing

Performance gains from code changes were monitored using the debug tools. Most of the tweaks in this iteration did not change the functional outcomes from the previous. I again looked at unit testing, but encountered an issue of not being able to import tests as modules, so again prioritised continuing development over solving this.

Requirements Testing

Performance was improved [U13.0], although the Web Assembly implementation didn't have a large effect. The main focus of this iteration was creative [U03.0, U06.0], integrating or re-integrating many visualisers and customising their parameters for the maximum range of possibilities. As the main user of this I was mainly aiming at fulfilling my own artistic aims. This was partially successful, which will be discussed more fully in the conclusion.

[2] https://thesadsongco.com/UoL/usertesting2.html

Conclusion

The application fulfils the stated user requirements, with the exception of U13.0, which will be discussed below. Requirement D2.0 has been fulfilled, and D1.0 will be decided by an external marker.

The iterative process of development of this project has brought out some interesting points. The final submitted version is user-friendly and easily modular, but at the expense of performance. In the prototype it was difficult to customise visualiser chains and change parameters due to the piecemeal way it was programmed, with each visualiser having a unique set of arguments and being called linearly via a series of conditionals dependent on an object that held the settings for each setlist item. It was unwieldy, but the customisation meant that each visualiser could be individually optimised for performance, resulting in some cases a better frames per second score for comparable visualisation chains than the final application.

This puts me in mind of compilers unrolling loops to improve execution times, as Lemire (2019) talks about [31]. The application software implementation that allows for a conveniently modular and adjustable environment carries unavoidable overhead that ad-hoc code does not. The final application has disappointing frame rates when the output resolution is high, which is necessary for display on large LED / projection screens in a live concert environment

I considered a rewrite of the `VisOutputEngine` and `ProcessCanvas` classes, and all pixel-processing visualisers, to make use of WebAssembly after my minimal first use of that technology for random noise generation. I concluded however that fundamentally the technologies required by the template are not appropriate for what I was trying to achieve. While HTML / CSS / Javascript greatly simplifies the creation of a UI, the time-sensitive and processor-intensive manipulation of live video will always be bottlenecked by attempting to implement in a browser-based environment. My research suggested that even employing WebAssembly may not bring sufficient performance gains, due to the overhead of pushing pixel arrays into shared memory every frame.

I experimented with using `OpenCV.js`, a WebAssembly binary of OpenCV. While this did provide some performance gain, it did so at the expense of the idiosyncratic control over the visualisations, which for me was central to the application - the OpenCV version produced visualisers that had a familiar ‘samey’ look that couldn’t be eliminated with the parameters that the library provided. For this reason I judged the artistic outcome of my version as written was more important than FPS performance.

Using WebWorkers gave performance gains, but also came with their own issues. Their most effective use was achieved by dividing the output canvas into a sub canvas for each worker to process. However this made implementations of some ‘whole canvas’ visualisation effects messy, both visually and in code.

The ASCII visualiser, by the nature of its display method, ended up with the gaps between sub-canvases being visible unless the rows and font size parameters were carefully set. Displaying text and external media required a partial re-write of `VisOutputEngine` to make use of special engines (`ExtMediaEngine`, `TextDisplayEngine`) that would draw to an offscreen canvas and then pass truncated information to each sub-canvas for

rendering to the visible output. This also hit performance, but there was no way I could devise for the workers to be able to do these operations in their own domains.

This restricted which visualisers that I had included in the prototype could be successfully re-integrated with the new application. Rather than undermine the coherence of the codebase architecture trying to incorporate some of the more esoteric ideas, I decided that would be more usefully held over to a proposed native implementation using a single output canvas (below).

Future Development

After my degree has concluded it is my intent to port the application over to Swift so it can run natively on MacOS. A compiled language that can address the GPU should be able to improve performance significantly.

With an increase in performance, the node patching functionality could be expanded. For this implementation the available slots for processors are deliberately limited to 3, since larger chains would impact severely on the frame rate. In a compiled implementation this can be expanded and made more flexible, perhaps moving towards the Max/Jitter environment discussed in Existing Solutions. This would also allow development of the modulation system. Currently visualisations are restricted to modulation sources set in the parameters of each visualiser. Being able to freely cross-patch any modulator to any destination would produce exciting possibilities.

Another development would be to have the application send and receive DMX512[27] data. If the application could be controlled this way it would allow tighter and more effective integration with lighting. Research indicates that this would require a partial hardware solution, which is beyond the remit of this project.

There are already plans for further festival appearances for the artist with whom I tested the prototype, and this application (whether remaining browser-based or implemented natively) will be employed for that purpose.

Appendix 1 - References

[1] Krueger, Alan B., 2005. The Economics of Real Superstars: The Market for Rock Concerts in the Material World. Journal of Labor Economics, Vol. 23, No. 1 (Jan 2005), pp. 1-30

[2] Krueger, Alan B., 2019. Rockonomics: What the Music Industry Can Teach Us About Economics (and Our Future). 4th ed USA: John Murray

[3] Maasø, A., 2018. Music Streaming, Festivals, and the Eventization of Music. Popular Music and society, Vol. 41, No. 2 (2018), pp. 154–175

[4] Swarbrick, D et al, 2019. How Live Music Moves Us: Head Movement Differences in Audiences to Live Versus Recorded Music. Frontiers In Psychology, Volume 9 (January 2019), Article 2682

[5] Dunham, Richard E., 2015. Stage Lighting Fundamentals and Applications. 3rd ed. London: Focal Press

[6] Moody, Dr. James L., Dexter, P, 2016. Concert Lighting: Techniques, Art and Business. 4th ed. London: Routledge

[7] Catalyst V5 Information [no author or date listed]. Retrieved November 18, 2021 from http://www.catalyst-v5.com/

[8] Design Starts Here [no author or date listed]. Retrieved November 18, 2021 from https://www.green-hippo.com/design-starts-here/design-starts-here/

[9] Resolume Media Server [no author or date listed]. Retrieved November 18, 2021 from https://resolume.com/software/avenue_arena

[10] MagicQ MQ500M Stadium Console [no author or date listed]. Retrieved November 18, 2021 from https://chamsyslighting.com/products/mq500m

[11] GrandMA3 full-size [no author or date listed]. Retrieved November 18, 2021 from https://www.malighting.com/product/grandma3-full-size-4010500/

[12] Our new flagship console, the Diamond 9 [no author or date listed]. Retrieved November 18, 2021 from https://www.avolites.com/product/diamond-9-330/

[13] A Playground for Invention [no author or date listed]. Retrieved November 18, 2021 from https://cycling74.com/products/max

[14] NIN | Lights in the Sky Tour | 2008 [no author or date listed]. Retrieved November 18, 2021 from http://www.leroybennett.com/portfolio/nin-lights-in-the-sky-tour-2008/

[15] Nine Inch Nails @ Fuji Rock Festival (HD Pro-Shot). Video. (30 Jul 2013). Retrieved November 18, 2021 from https://www.youtube.com/watch?v=scQtQ33TSY0

[16] MDN Web Docs. 2022. Broadcast Channel API (21 Jan 2022). Retrieved Jan 30 2022 from https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API

[17] Javascript Info. 2021. IndexedDB (12 Dec 2021). Retrieved Jan 30 2022 from https://javascript.info/indexeddb

[18] Adelson, E, 2015. Image Processing Goes Back to Basics. Communications Of The ACM, Vol 58, No.3 (March 2015), p 80

[19] Park, H, Cha, M, Moon, S-M, 2016. Concurrent JavaScript Parsing for Faster Loading of Web Apps. ACM Transactions on Architecture and Code Optimization, Vol. 13, No. 4 (November 2016), Article 41

[20] Verdú, J, Pajuelo, A, 2016. Performance Scalability Analysis of JavaScript Applications with Web Workers. IEEE Computer Architecture Letters, Vol. 15, No. 2 (December 2016), pp 105-108

[21] Hawkins, T, 2020 How to Unit Test HTML and Vanilla JavaScript Without a UI Framework. Retrieved 8 Jan 2022 from https://dev.to/thawkin3/how-to-unit-test-html-and-vanilla-javascript-without-a-ui-framework-4io

[22] IMDB - Jonny Greenwood [no author or date listed]. Retrieved 8 Jan 2022 from https://www.imdb.com/name/nm0339351/

[23] artecode, 2021. IndexedDB with usability. Retrieved 10 Jan 2022 from https://github.com/jakearchibald/idb

[24] MDN Web Docs, 2021. draggable (3 Oct 2021). Retrieved 10 Jan 2022 from https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/draggable

[25] MDN Web Docs, 2021. Dynamic Imports (4 Dec 2021). Retrieved 10 Jan 2022 from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import#dynamic_imports

[26] banerjeer2611 [no date listed]. JavaScript: Do you know the fastest way to iterate over arrays, objects? Retrieved 10 Jan 2022 from https://www.realpythonproject.com/javascript-do-you-know-the-fastest-way-to-iterate-over-arrays-objects/

[27] Wikipedia [no date listed]. DMX512. Retrieved 10 Jan 2022 from https://en.wikipedia.org/wiki/DMX512

[28] Multiple Contributors, 2009. Formula to determine perceived brightness of RGB color (27 Feb 2009). Retrieved 9 Feb 2022 from https://stackoverflow.com/questions/596216/formula-to-determine-perceived-brightness-of-rgb-color

[29] Multiple Contributors, 2017. Destructuring Variables Performance (3 Nov 2017). Retrieved 9 Feb 2022 from https://stackoverflow.com/questions/47099510/destructuring-variables-performance

[30] owen-m1, 2021. Sortable (2 Dec 2021). Retrieved 10 Feb 2022 from https://github.com/SortableJS/Sortable

[31] Lemire, D, 2019. Why are unrolled loops faster? (12 Apr 2019). Retrieved 24 Feb 2022 from https://lemire.me/blog/2019/04/12/why-are-unrolled-loops-faster/

Appendix 2 - Figures

fig 1: Catalyst Media Server hardware racks

fig 2: Chamsys lighting desk

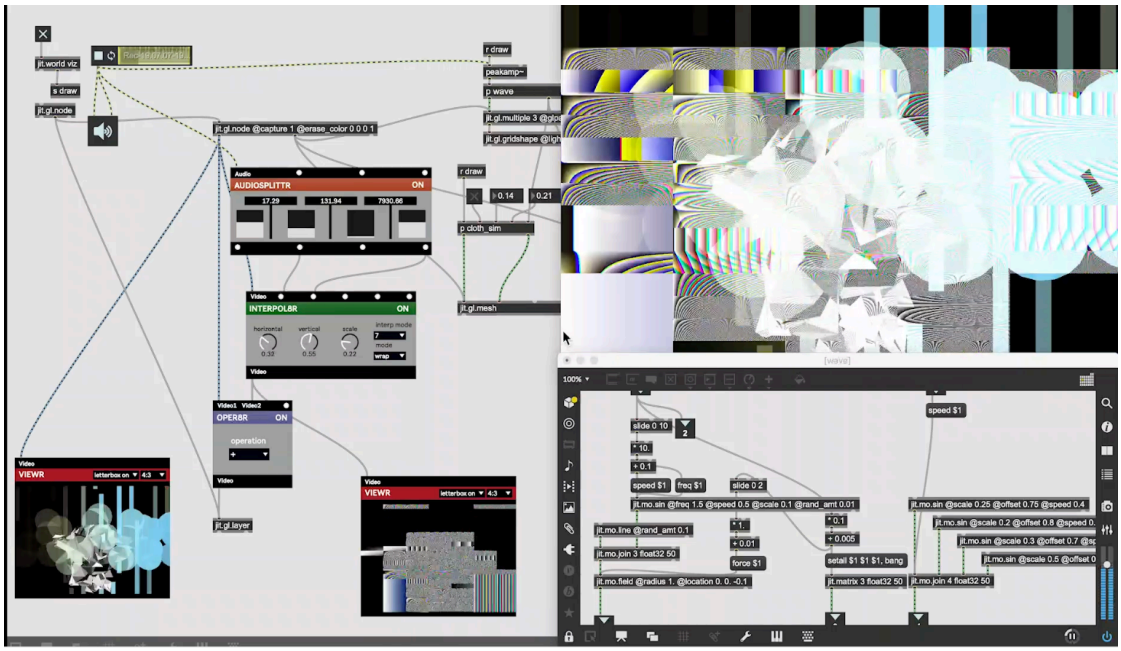

fig 3: Max programming environment and video output

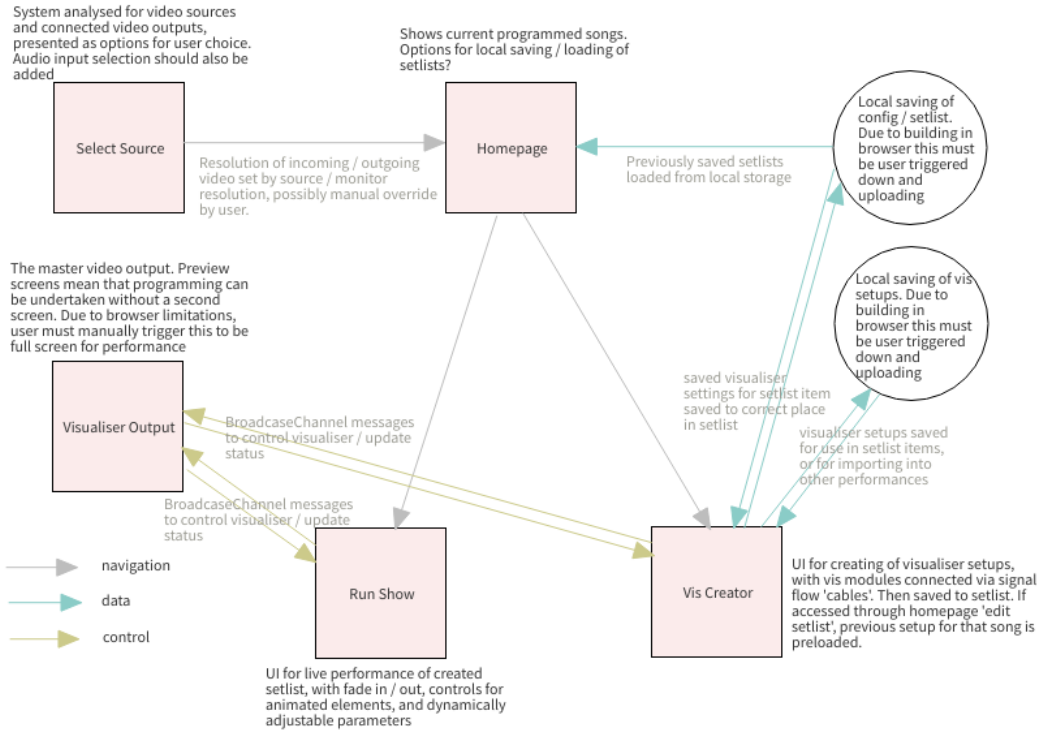

fig 4: diagram of application architecture

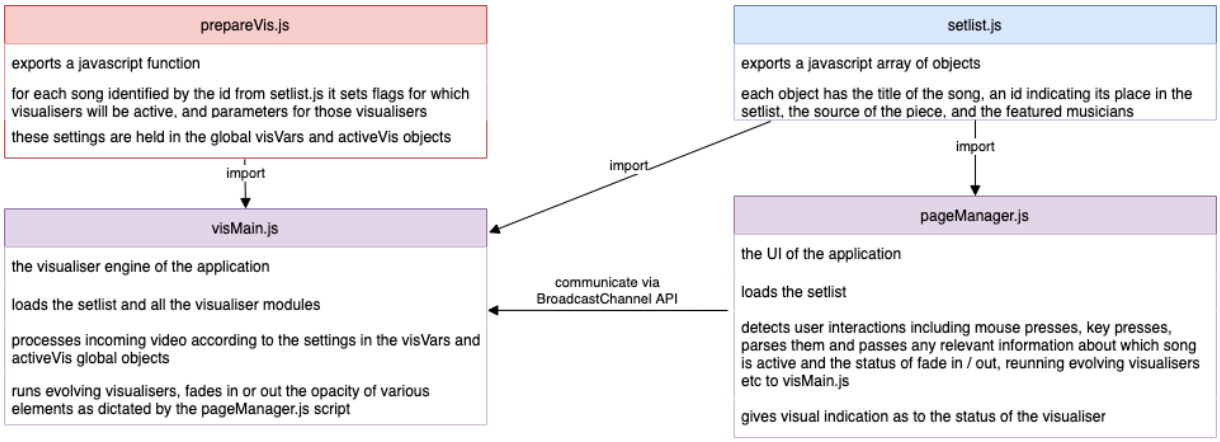

fig 5: architecture overview of application

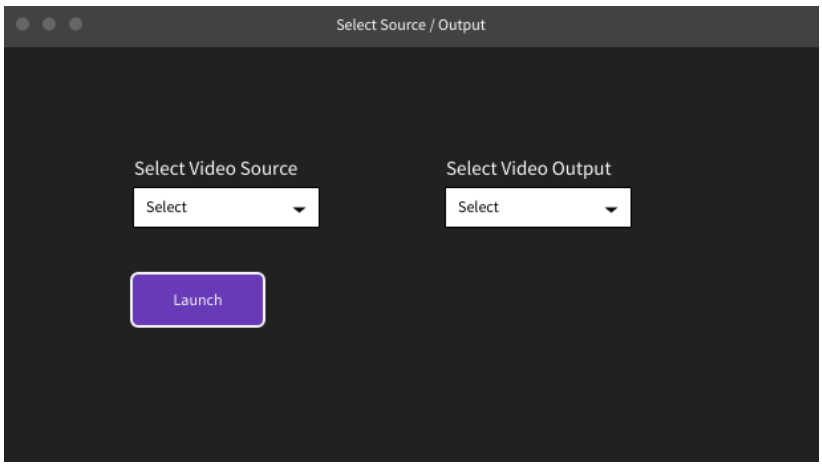

fig 6: Page 1 - on app opening
Allows choice of:

- video source: since the application is focussed on processing IMAG, this will likely come from a vision-mixed feed
- video output: during testing this will be a second monitor, for performance it will be a projector or LED screen for stage display

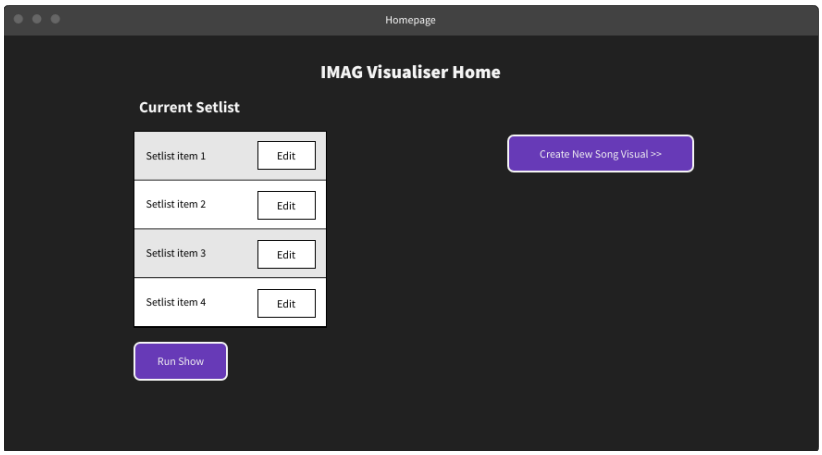

fig 7: Page 2 - Homepage
Central hub showing progress and saved setlist item visuals, along with links to run the show for live performance and to the create visuals page

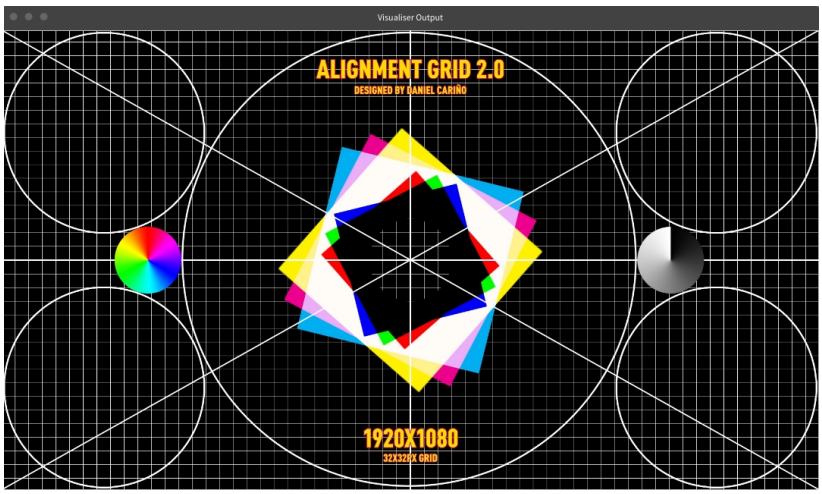

fig 8: Page 3 - Visualiser Output
On monitor while building, or output to fullscreen in performance

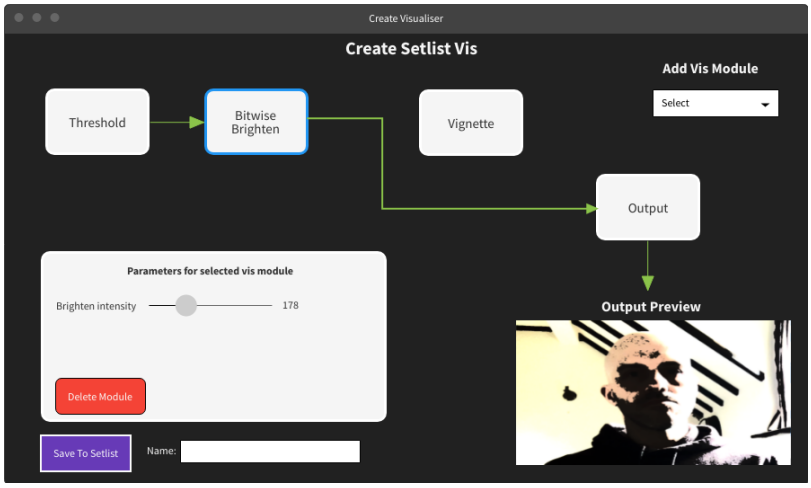

fig 9: Page 4 - Setlist Visual Creator
Visualiser modules can be added and chained together, their parameters adjusted and the chain saved as a setlist item

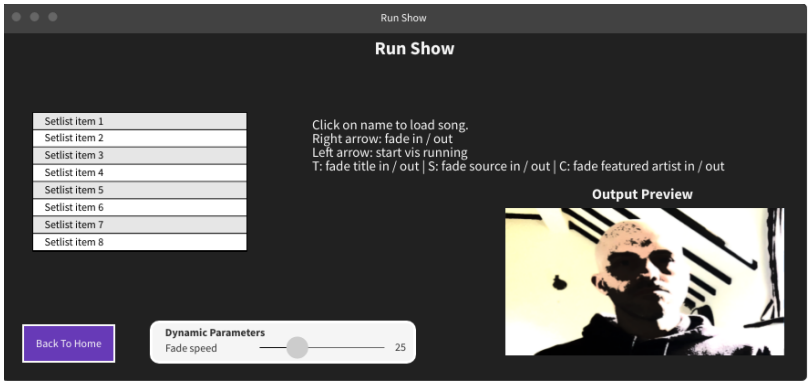

fig 10: Page 5 - Run Show
Simplified screen for easy running of the show, selecting and fading in visuals for each song / section of song as required, with dynamic parameters for each visualiser displayed, and an output preview

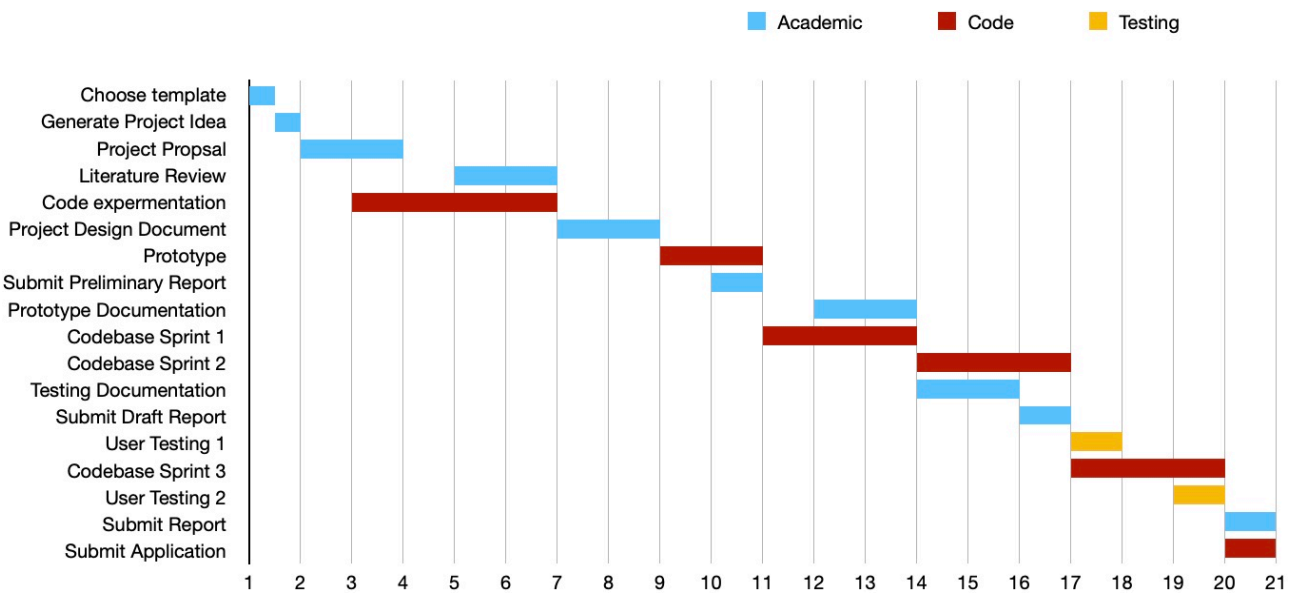

fig 11: Gantt Chart work plan. This is flexible by necessity, to account for schedule changes due to other university work, life commitments and roadblocks I might hit in the coding

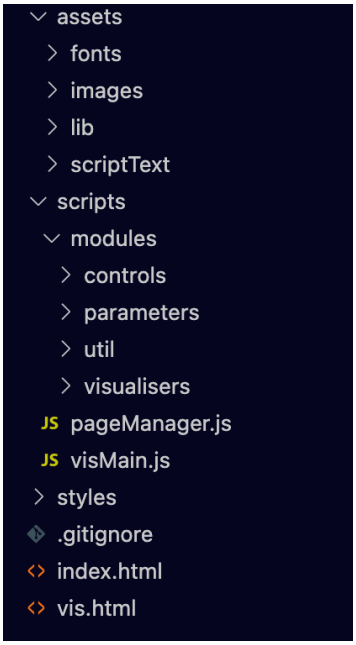

fig 12: modules allowing for a clear file hierarchy

```
const channel = new BroadcastChannel('vis-comms')

const keyEvent = function(e, currVisState, currSetState, currSourceState,
currFeatState, currSetId, run) {
    if (e.key == 't') { // toggle title
        e.preventDefault()
        switch(currSetState) {
            case 0:
                channel.postMessage({
                    setItemFadeIn: true,
                    setItem: currSetId
                })
                currSetState ++
                break
            case 1:
                channel.postMessage({
                    setItemFadeOut: true,
                    setItem: currSetId
                })
                currSetState = 0
                break
        }
    […]
    return [currSetState, currSourceState, currFeatState, currVisState, run]
}

export default keyEvent
```

fig 13: code extract from controlKeyEvents.js

```
// LOAD UTILITIES
import keyEvent from './modules/util/controlKeyEvents.js'
```

fig 14: code extract from pageManager.js

```
// CONTROLLER / VISUALISER COMMUNICATION
const channel = new BroadcastChannel('vis-comms')
channel.addEventListener('message', (e) => {
    // console.log(e.data)
    if (e.data.changeTrack) {
        // if this is our first track change from the test card,
        // make sure we're looping
        if (!isLooping())
            loop()
        channel.postMessage({
            visTransition: true
        })
        if (currTrack !== e.data.setItem)
            currTrack = e.data.setItem
```

fig 15: code extract of visMain.js receiving BroadcastChannel messages and acting on them

```
const prepareVis = function(setItem, activeVis, scriptVis, scriptText, visVars,
asciiVis) {
    // clear the canvas
    // console.log(setItem.id)
    switch(parseInt(setItem.id)) {
        case 0:
            // Intro
            break

        case 1:
            // House Of Woodcock
            visVars.bgOpacity = 180
            visVars.bw = true
            activeVis.showPixThru = true
            activeVis.showVignette = true
            break
```

fig 16: code extract from prepareVis.js

```
const bitwiseBrighten = function(pixIdx, iR, iG, iB, brighten=1, bw=false, pixels) {
    if (bw)
        iR = iG = iB =(0.28125*iR + 0.578125*iG + 0.109375*iB)
    pixels[pixIdx + 0] = iR << brighten
    pixels[pixIdx + 1] = iG << brighten
    pixels[pixIdx + 2] = iB << brighten
    pixels[pixIdx + 3] = 255

    return [pixels[pixIdx + 0], pixels[pixIdx + 1], pixels[pixIdx + 2]]
}
export default bitwiseBrighten
```

fig 17: using bitwise operations to process pixels

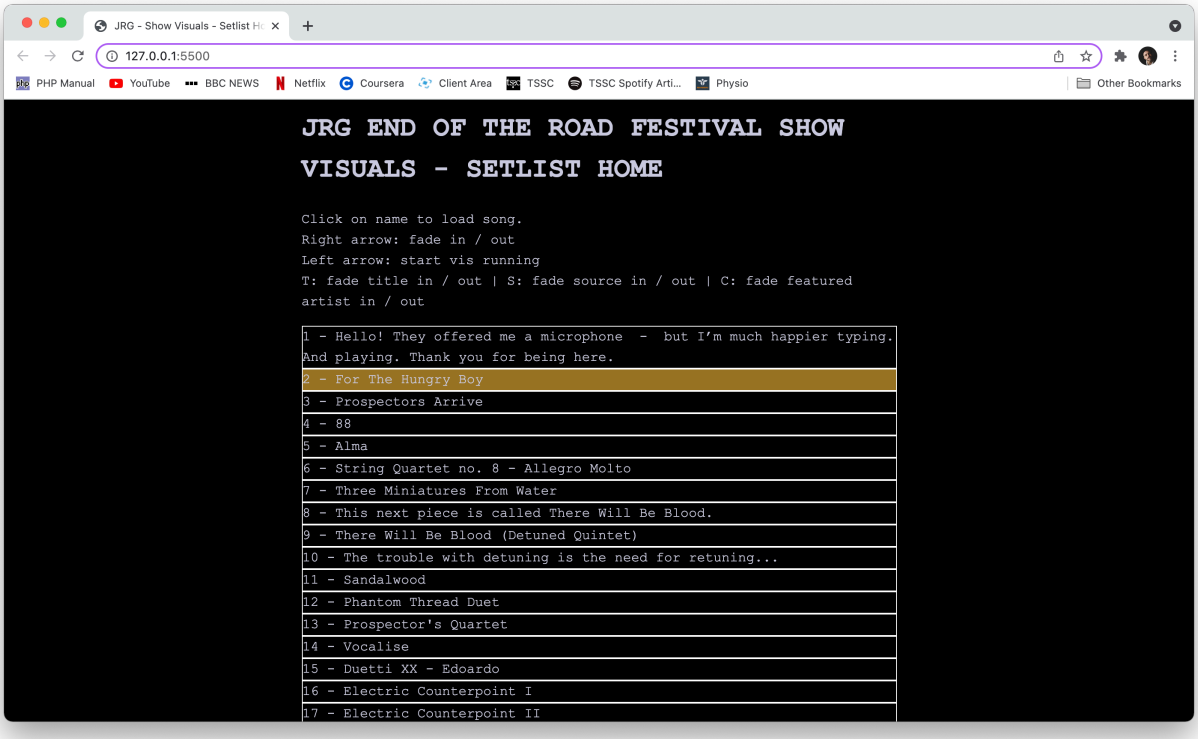

fig 18: prototype control screen showing setlist and instructions, with selected active song highlighted

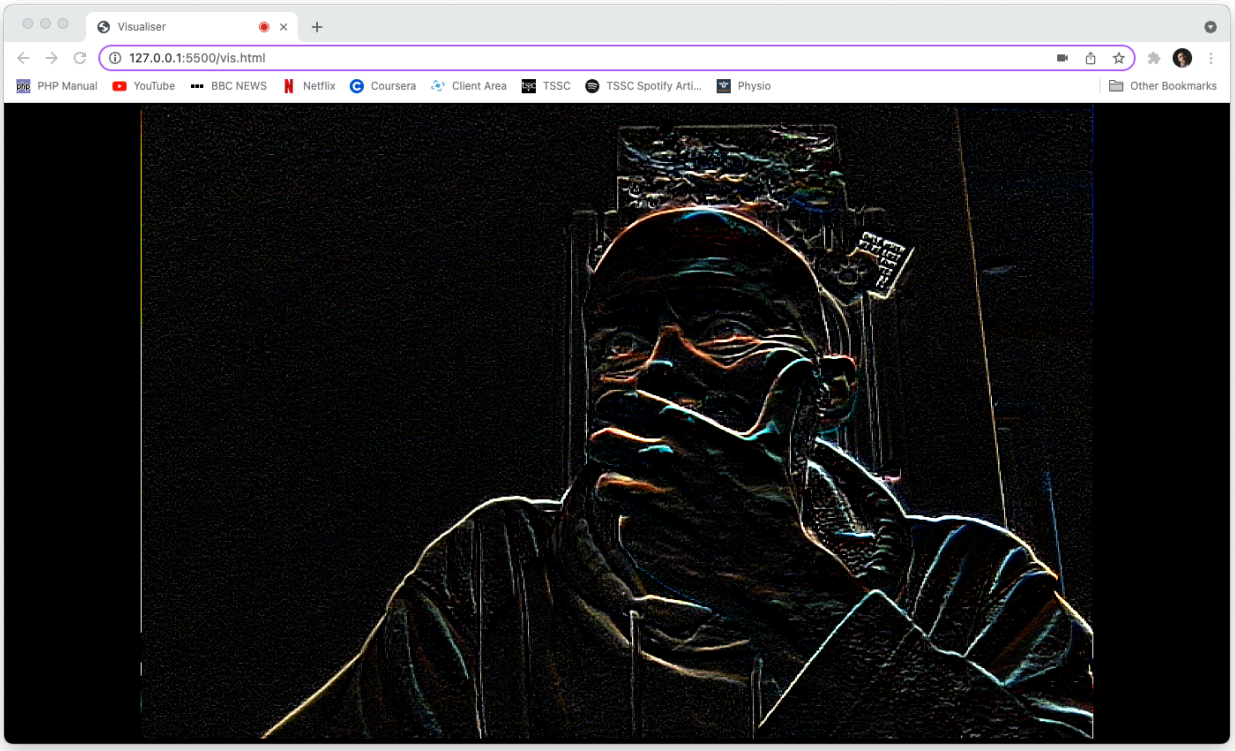

fig 19: prototype visualiser output window showing the edge detection visualiser working on a disgruntled student

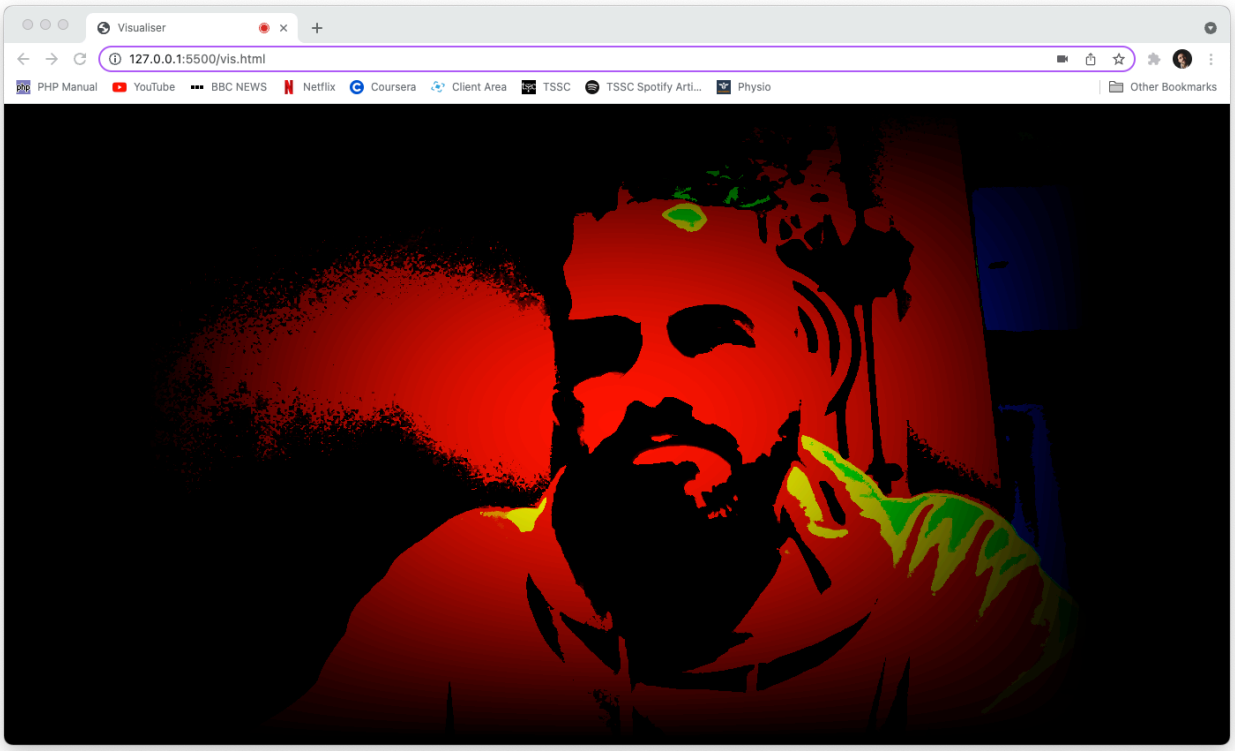

fig 20: prototype visualiser output window showing the bitwise1 and vignette visualiser

fig 21: revised front end file structure

```
import { visList } from "../visualisers/registeredVis.js";

/**
 * imports all of the registered visualiser modules into global visualiserModules array
 */
export const importModules = async function () {
    const visualiserModules = {};
    const path = '../visualisers/'
    for (let visGroup of visList) {
        for (let vis of visGroup.visualisers) {
            const modulePath = `${path}${visGroup.visGroup}/${vis.name}.js`;
            const currModule = await import(modulePath)
            visualiserModules[vis.name] = new currModule[vis.name]
        }
    }
    return  visualiserModules;
};
```

fig 22: code extract from scripts/modules/common/importModules.js

```
    let openRequest = indexedDB.open('visDB', 1);
    openRequest.onerror = () => {
        return 'Database error'
    }
    openRequest.onsuccess = () => {
        let db = openRequest.result;
        // write output resolution to db
        let transaction = db.transaction('outputResolution', 'readwrite');
        let writeSettings = transaction.objectStore('outputResolution');
        let outputSettings = {
            id: 1,
            outputResolution: resolutions[resSelector.value]
        }
        writeSettings.put(outputSettings);
        // write video input settings to db
```

fig 23: code extract from scripts/init.js

```
/**
 * Asynchronously retrieve current setlist from database
 * @returns {IDBObjectStore} - setlist retrieved from database
 */
export const getSetlist = async() => {
    let db = await openDB('visDB', 1, db => {
        if (db.oldVersion == 0) {
            console.log(`Error opening database: ${err.message}`);
            return null;
        }
    });
    return await db.getAll('setlist')
}
```

fig 24: code extract from scripts/modules/common/getSetlist.js

```
export const pseudoRandomGenerator = () => {
    return  d2b(Math.trunc(Math.random()*2147483647*2*2)+1);
}
...
export const d2b = x => x.toString(2);
```

fig 25: code extracts from scripts/modules/util/generators.js and scripts/modules/util/util.js

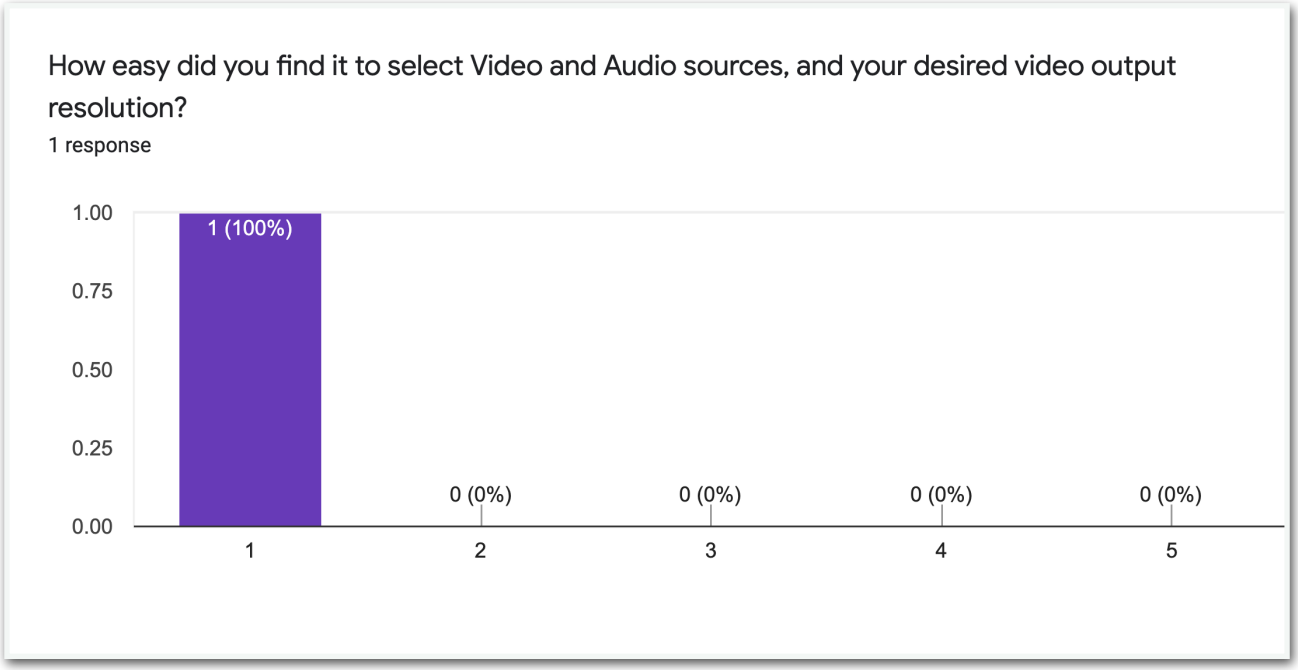

fig 26 i: Google questionnaire response

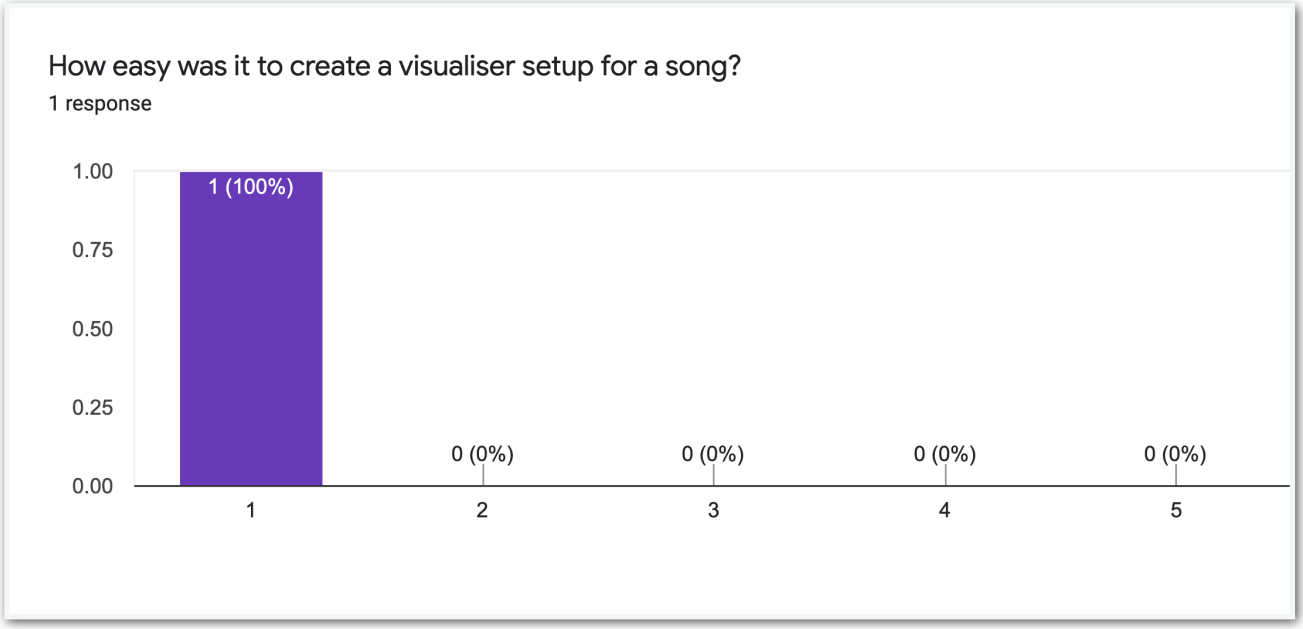

fig 26 ii: Google questionnaire response

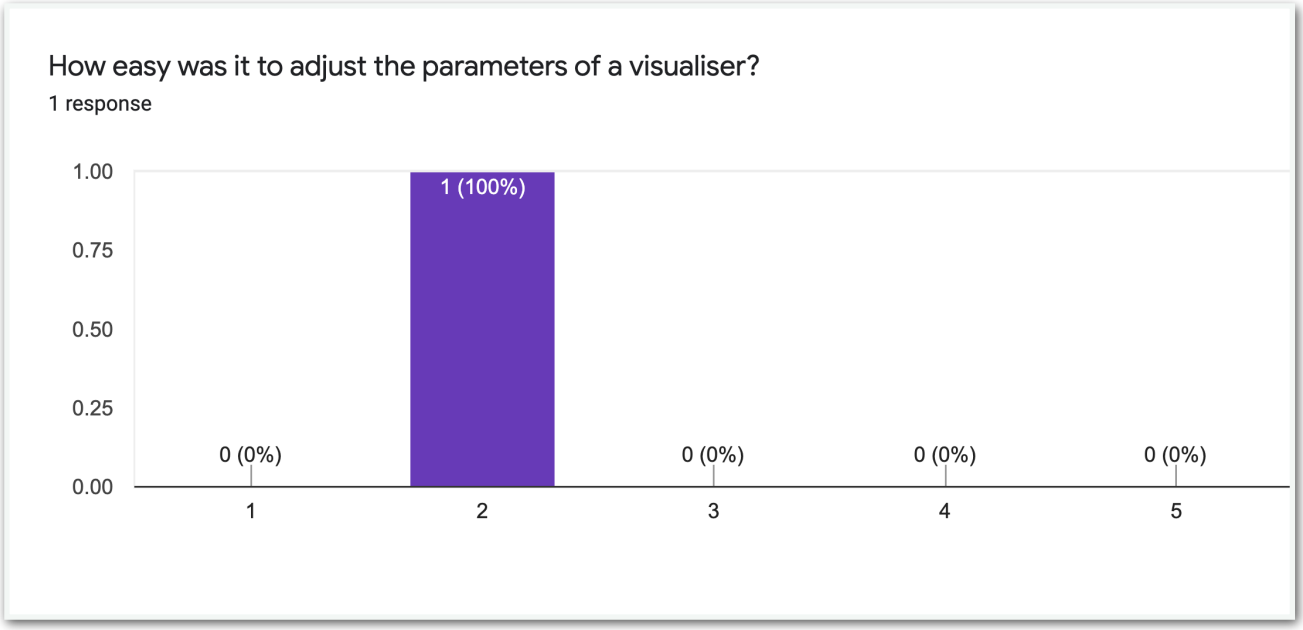

fig 26 iii: Google questionnaire response

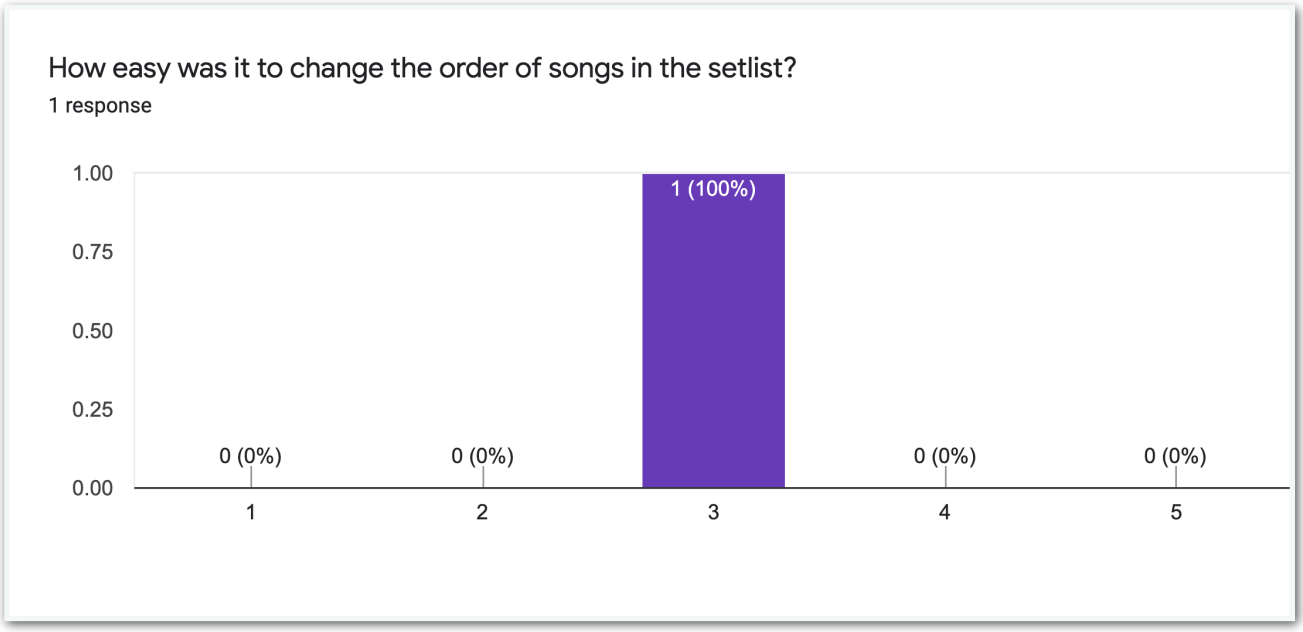

fig 26 iv: Google questionnaire response

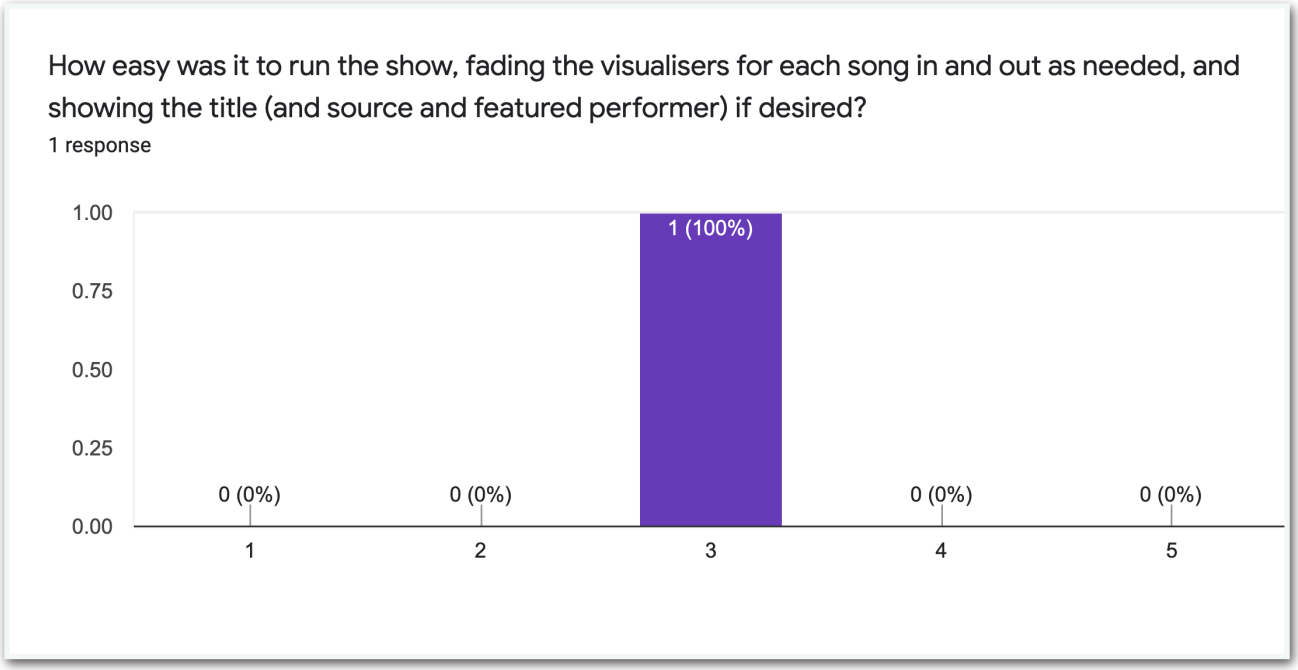

fig 26 v: Google questionnaire response

```
const greyscaleCalc = ([iR, iG, iB]) => {
  // return iR * 0.3 + iG * 0.59 + iB * 0.11;
  /* updated with performant formula from https://stackoverflow.com/questions/596216/
formula-to-determine-perceived-brightness-of-rgb-color */
  return ((iR << 1) + iR + (iG << 2) + iB) >> 3;
};
```

fig 27: code extract from utils.js

```
  invokeWorker = (worker, context) => {
    return new Promise((resolve) => {
      worker.postMessage(context);
      worker.onmessage = (e) => {
        resolve(this.onWorkEnded(e));
      };
    });
  };

  onWorkEnded = (e) => {
    const canvasData = e.data.result;
    const index = e.data.index;
    this.cnvContext.putImageData(canvasData, 0, this.blockSize * index);

    this.finished++;
    if (this.finished === this.workersCount) {
    }
    return Promise.resolve(true);
  };
```

fig 28: code extract from visOutputEngine.js in WebWorkers branch

```
const processCanvas = new ProcessCanvas();

onmessage = (e) => {
  switch (e.data.task) {
    case 'setup':
      processCanvas.setup(e.data);
      break;
    case 'draw':
      processCanvas.draw(e.data.data);
      break;
    default:
      const method = processCanvas[e.data.task];
      if (typeof method === 'function') method(e.data.data);
  }
};

export default ProcessCanvas; // used when testing with no workers
```

fig 29: code extract from canvasWorker.js

```
      this.subcnvs.push(subCnv.transferControlToOffscreen());
      this.subcnvParams.push({
        start: subCnvStart,
        drawStart: subCnvDrawStart,
        height: subCnvHeight,
        drawHeight: subCnvDrawHeight,
      });
      workerJobs.push(
        this.invokeWorker(
          this.workers[i], {
          task: 'setup',
          canvas: this.subcnvs[i],
          start: subCnvStart,
          drawStart: subCnvDrawStart,
          height: subCnvHeight,
          drawHeight: subCnvDrawHeight,
        },
        [this.subcnvs[i]]
        )
      );
      // next sub canvas
      subCnvStart = subCnvStart + subCnvHeight - (this.subcnvOverlap * 2);
      subCnvDrawStart = this.subcnvOverlap;
    }
```

fig 30: code extract from visOutputEngine.js

```
const alphaBlend = (bottomLayerPixVals, upperLayerPixVals) => {
  // considered math.js for vector operations, but it was overkill
  let oR =
    bottomLayerPixVals.r * (1 - upperLayerPixVals.a) +
    upperLayerPixVals.r * upperLayerPixVals.a;
  let oG =
    bottomLayerPixVals.g * (1 - upperLayerPixVals.a) +
    upperLayerPixVals.g * upperLayerPixVals.a;
  let oB =
    bottomLayerPixVals.b * (1 - upperLayerPixVals.a) +
    upperLayerPixVals.b * upperLayerPixVals.a;
  return { r: oR, g: oG, b: oB, a: 255 };
};
```

fig 31: code extract from /scripts/modules/common/blendModes/alphaBlend.js in WebWorkers branch

pixelProcessor worker / 1280 x 720 res

No. Web Workers	Reported FPS
0	6
2	8
4	8
8	8

original code before branch / 1280 x 720

No. Web Workers	Reported FPS
0	4

canvasProcessor worker / 1280 x 720 res

No. Web Workers	Reported FPS
0	5
1	5
2	9
4	14
6	14/15
8	15/16
10	14/15

fig 32: tables of fps testing results

```
  processPixels = function (pixIdx, pixVals, kwargs = {}, context) {
    // setup visualiser parameters with default values
    let { threshold = 100 } = kwargs;
    const { dynThresh = false } = kwargs;
    const { dynThreshMin = 0, dynThreshMax = 255 } = kwargs;
    const { dynThreshSpeed = 0 } = kwargs;
    const { bw = false } = kwargs;
```

fig 33: code extract from visualiser threshold.js

```
let loResStep = (context.cnv.width / resolution) << 0;
```

fig 34: code extract from visualiser boxes.js

```
(async () => {
  const fetchPromise = fetch('../../wasm/randomCols.wasm');
  const { instance } = await WebAssembly.instantiateStreaming(fetchPromise, {
    env: {
      malloc: (len) => wasmMalloc(len),
      free: (addr) => wasmFree(addr),
      emscripten_resize_heap: (arg) => console.log(arg),
    },
  });
  randGrey = instance.exports.rand_grey;
  randRgb = instance.exports.rand_rgb;
  malloc = instance.exports.malloc;
  free = instance.exports.free;
  mem = instance.exports.memory;
})();

export const getRandGrey = () => {
  return randGrey();
};

export const getRandRGB = () => {
  var output_ptr = malloc(6);
  randRgb(output_ptr);
  const result = new Uint16Array(mem.buffer, output_ptr, 3);
  // dealloc memory to avoid memory leaks
  free(output_ptr);
  return [...result];
};
```

fig 35: code extract from randGenerator.js using randomCols.wasm Web Assembly

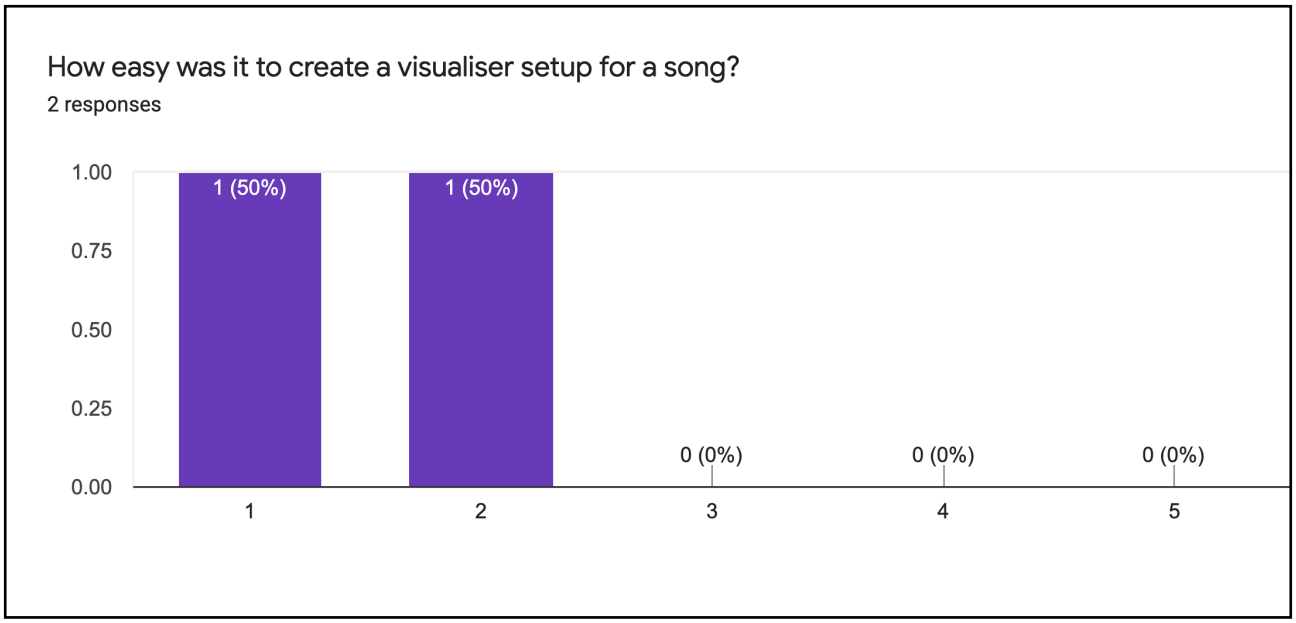

fig 36 i - response to google questionnaire

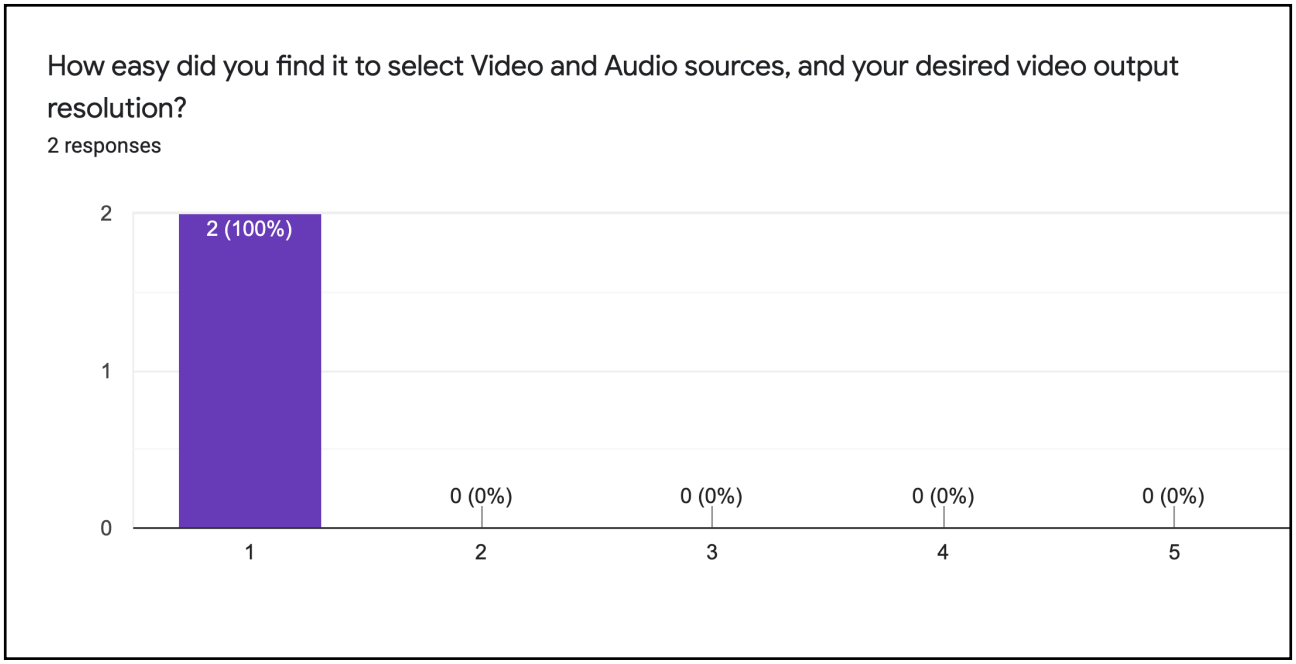

fig 36 ii - response to google questionnaire

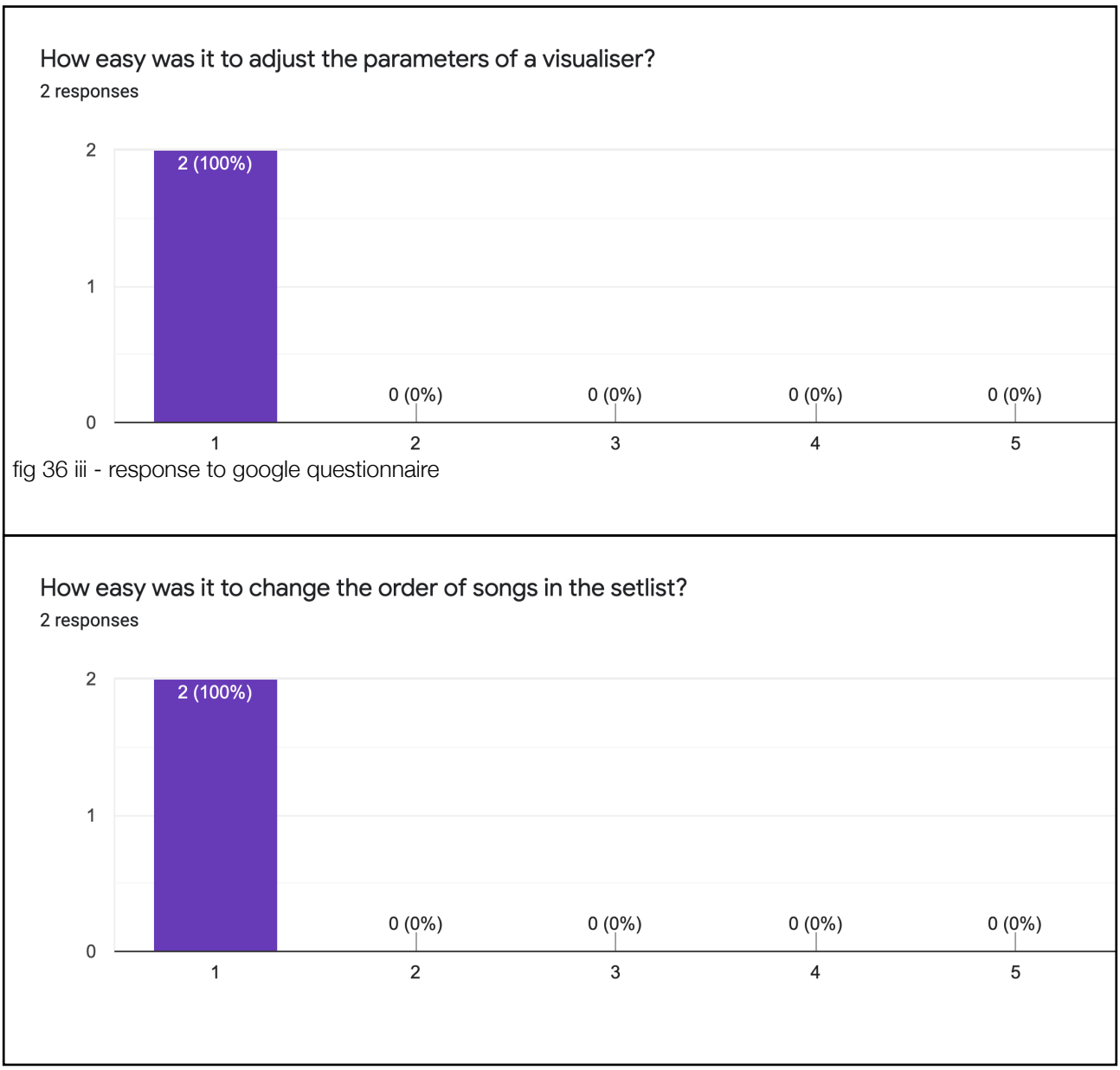

fig 36 iii - response to google questionnaire

fig 36 iv - response to google questionnaire

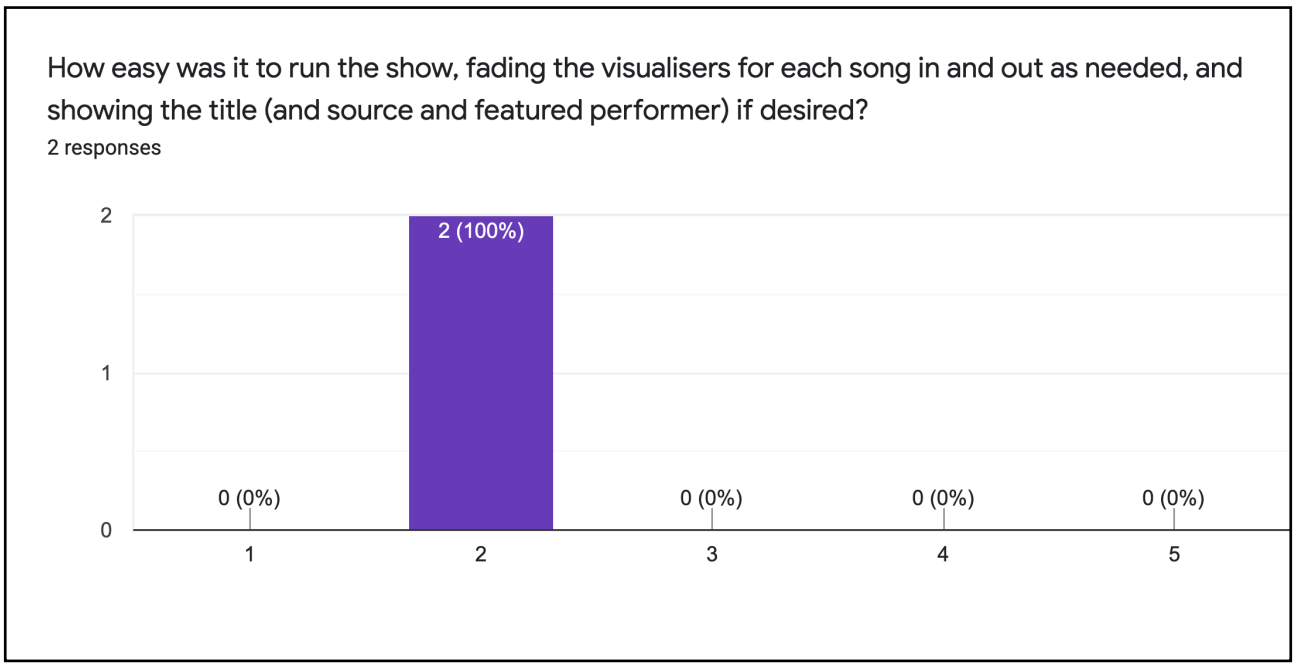

fig 36 v - response to google questionnaire

If you have time to write some comments about your experience with the app it will be very helpful. Were there things you thought were good about it? Did a feature come to mind that was missing? Was the user interface intuitive?

I had some issues using the content files, they seemed to preview with some effects and not others. I am not sure I understand the difference between pressing the left arrow to run the vis and the right arrow to fade it up? Maybe labelling the output layers in some way to show which is the top layer might be helpful? I love all the effects, they are really quirky and the level of control over each one is much more involved than you would expect from a relatively basic piece of video manipulation software. There is a lot of text over the preview window when building effects, not sure if that is just my laptop resolution etc but I would prefer to see more of the preview image and not have so much of it covered with text.
I love the audio manipulated effects, really nice addition.

Very impressed with the visualiser as demoed on the web interface. selecting inputs was simple (I have multiple video sources attached to the computer) and worked well. I had a lot of fun stacking the various effects to create a song setup and the parameters on the whole made sense. I couldn't quite get some of the effects to react to the audio inputs as I thought they should but i may have been making incorrect assumptions as to how they 'should' react rather than learn how they were programmed to react.

I really liked the implementation of some of the visual effects and the parameter range seemed really well thought out to give that balance between ease of accurate adjustment and range of effect. Very nicely done.

Arranging the set list and moving around it on the fly was fairly simple and made sense. I am hoping the next iteration of the software has a play button that cycles through the set list and loops. I think that would be fabulous for installation pieces.

I really liked the choice of effects for the three available layers. They worked really nicely together and also independently. The software allows for some beautiful effects combinations.

Overall I think the software could be really useful as a live tool and I would love to see how it develops further. I would be very happy to become a beta tester for ongoing development.

fig 36 vi - response to google questionnaire

```css
/* TOOLTIPS */
/* http://jsfiddle.net/AndreaLigios/jtLbpy62/2281/ */
[data-tooltip]:before {
  /* needed - do not touch */
  content: attr(data-tooltip);
  position: absolute;
  opacity: 0;

  /* customizable */
  transition: all 0.15s ease;
  padding: 10px;
  max-width: 30%;
  font: 0.8em Montserrat, sans-serif;
  color: #333;
  border-radius: 10px;
  box-shadow: 2px 2px 1px silver;
}

[data-tooltip]:hover:before {
  /* needed - do not touch */
  opacity: 1;

  /* customizable */
  background: #eaeaff;
  margin-top: -5%;
  margin-left: -7%;
}

[data-tooltip]:not([data-tooltip-persistent]):before {
  pointer-events: none;
}
```

fig 37: code extract from creator.css

Appendix 3 - Git Log

```
2022-02-22  36a49fe     Updated dynamic modulation to use time
2022-02-20  2acfe6d     wasm noise implementation
2022-02-20  025e93e     Vis container centering
2022-02-20  cb95520     Init UI
2022-02-20  76789cc     Noise visualiser 1.0, UI tweaks
2022-02-19  cead53c     Added README
2022-02-19  47c4102     Initialise VisChain issue solved
2022-02-19  594e5eb     ExtMediaEngine relative paths
2022-02-19  d7cd59f     Relative paths, grabbing style
2022-02-19  bb650fb     Clean up, relative path imports
2022-02-18  c00aad8     Submission version with documentation
2022-02-18  e970919     Sortable Setlist update
2022-02-18  d9e2b8f     Added bump change when running show
2022-02-18  d2ff331     VideoFile visualiser update
2022-02-18  3df78e2     Merge remote-tracking branch 'origin/New-Module-Approach'
2022-02-18  54b6e6a     Final before merge
2022-02-18  1e9197c     Cleanup Visualisers
2022-02-13  15949a8     Tweaks
2022-02-11  29b0b08     TextDisplay audio reactivity
2022-02-11  6eb1809     Controllable animations
2022-02-11  62d9609     Text Vis mk 2
2022-02-10  7f67940     TextDisplay more complete
2022-02-10  fd71197     TextDisplay first pass
2022-02-10  6deaecc     extMedia and new engine finished
2022-02-09  6f69a8a     Rebuilt VisOutputEngine - class based vis
2022-02-08  113fd6f     Way point before changing module management
2022-02-08  93b7307     Removed destructuring performance hit
2022-02-08  ca48da2     Reorganising and UI tweaks
2022-02-08  02b8ee4     Bitwise Brighten visualiser, some vis tweaks
2022-02-07  21c11f3     Solved repeated vis issue
2022-02-06  b7dbeea     Debug info
2022-02-06  4ecbee7     external video visualiser
2022-02-05  30fb3b5     Added tooltips, debug toggle button
2022-02-05  746222f     Solved vignette colour issue, UI tweaks
2022-02-05  a50ae78     Init page UI loading screen
2022-02-05  f6bf935     edgeDetect visualiser
2022-02-05  494dfc5     ASCII visualiser
2022-02-04  51cdebb     Ascii still not quite right
2022-02-04  d5ca0d0     ASCII vis partially working
2022-02-03  68ecce6     Changed openDB from dynamic to static for offline
2022-02-02  1c5d9c7     vignette and boxes visualiser modules added
2022-02-02  6c94a97     Vignette checkpoint
2022-02-02  fb81520     Audio integration, visualiser designing
2022-01-25  1bf3480     Code tidy up
2022-01-25  02c3bec     Canvas workers test final
2022-01-25  bf4fc4b     Rebuild to solve OOM issues, incomplete
2022-01-24  8086d55     Partial implementation
2022-01-23  d3099b0     Sub canvases defined and working
2022-01-22  d1b789d     Init commit
2022-01-18  8417f4d     Added visualisers and html colour pickers
```

```
2022-01-10  13b6b26     AudioEngine Class added
2022-01-10  6869975     Cleaned up documentation
2022-01-10  102d882     Save to and Load from disk
2022-01-09  d3a739c     Migration to canvas cont, styling of visMain
2022-01-07  7c34570     Started migration to canvas from p5.js
2022-01-06  9ca1451     Test Version 1.0 permissions problem solved
2022-01-06  e06f301     Test Deploy 1.0 updates for web bugs
2022-01-06  4b927a1     Test Version 1.0
2022-01-05  ee71c6a     Possible first user test candidate
2022-01-03  b656851     Visualisers to class-based
2021-12-31  087cf79     Audio Input integration 1
2021-12-31  b839d2d     Runshow controllers, added transition times
2021-12-30  04de982     Adjustable output settings
2021-12-29  b10a69e     Edit and delete setlist items, UI enhancements
2021-12-29  9bce385     Refactoring, started development on runshow / vis
2021-12-28  7b0acb8     Saving vis chains to DB
2021-12-24  e90b2b6     Reorder and save setlist to DB, happy christmas
2021-12-24  ab6ddaf     Refactoring, visualiser parameters
2021-12-21  784194b     Added docstrings to utils
2021-12-16  ced99b1     More drag and drop setlist
2021-12-14  7e28fdd     Sortable setlist, new hub page
2021-12-12  6cd8a15     Started updating modules for better modularity
2021-12-10  3cf5fff     Creator Preview first pass - not working
2021-12-10  38346b2     Init page and visDB
2021-12-10  0181eb9     Initial indexedDB sandbox - messy
2021-12-09  a41d0a2     Started on VisCreator
2021-12-06  5c714d5     File paths for deployment
2021-11-15  c2b1b6f     Tiny tweaks thinking about UoL prototype
2021-09-03  8694b89     Final version before EOTR
2021-09-02  fd1e9b4     Last tweaks tomorrow
2021-09-01  10ab3c3     Updated after rehearsal day
2021-08-31  3be06a3     Final, except camera choose
2021-08-30  a6c4d27     UI and all vis except score done,
2021-08-30  4932e91     Back to the starting line
2021-08-12  ca60e83     UI improvements
2021-08-12  cb9c9a8     Reorganised vis directory
2021-08-11  03ae2a3     Recode Ascii my way
2021-08-08  887f196     UI and control changes 1
2021-08-08  489f580     ScriptVis, loResVignette, som refactoring
2021-08-07  9a351cb     Tidied folder structure, added controls
2021-08-07  deaa67f     Control update 1 - fades
2021-08-05  d91388b     Lines, Motion, Threshold, tweaks
2021-08-05  a56a477     Circles, Boxes and tweaks
2021-07-28  334ac86     ascii module
2021-07-28  2f1519b     Added ascii module
2021-07-27  6170df8     Edge Detector implemented
2021-07-24  ddcb7c8     Added Modules, worked on vignetting
2021-07-22  2b98928     More Experimentation - boxes
2021-07-22  ea5a09a     Bitwise Experiments
2021-07-22  0e3bc44     Initial Commit
```