School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCE


# Using Generative Art Techniques to Explore the Work of Darrell Viner

**Thomas Carroll**

**Submitted in accordance with the requirements for the degree of Computer Science with Artificial Intelligence (MEng)**

2020/21

**40 credits**

The candidate confirms that the following have been submitted.

| **Items** | **Format** | **Recipient(s) and Date** |
|---|---|---|
| Report | Report | SSO, Assessor, Supervisor (10/05/21) |
| Deliverable 1-6 | Github URL | Assessor, Supervisor (10/05/21) |
| Deliverable 7 | Demo URL | Assessor, Supervisor (10/05/21) |

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) – Thomas Carroll

## Summary

This project explores the work of Darrell Viner and produces a program that allows users to explore the space his pen-plotter work inhabits. To do this a variety of techniques are developed and deployed. Also, methods of exploring this space using sound and music are explored.

The project both provides potential users with a pleasant experience and a greater understanding of Viner's work, and provide techniques that can be applied to other computer art and music projects.

The techniques created in this project include: a point-wise method for approximating the rounding of Perlin noise; a method of using audio to demarcate space in higher-dimensions; and using Markov chains to generate melodies.

**Acknowledgements**

Thanks to John Stell for supervising this project and giving a lot of generally good advice!

Thanks to Rob Sturman for running 'The Mathematics of Music' (MATH2340) in which some of the ideas used in this report were introduced to me (tuning theory).

And finally, thanks to my friends and family for supporting me through this.

Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test"; see
http://www.leeds.ac.uk/gat/documents/policy/Proof-reading-policy.pdf.

# Contents

**1 Introduction** **2**

1.1 Problem Overview . . . . . 3

1.2 Aim . . . . . 4

1.3 Objectives . . . . . 4

1.4 Deliverables . . . . . 5

1.5 Plan . . . . . 5

1.6 Risk Mitigation . . . . . 6

**2 Graphics** **7**

2.1 Anatomy of Viner's Work . . . . . 8

2.2 Polygons . . . . . 8

2.3 The Grid . . . . . 10

2.4 Landscape Generation . . . . . 11

2.5 Integration . . . . . 15

2.6 Navigation . . . . . 16

2.7 User Interface . . . . . 18

**3 Music** **21**

3.1 Navigating Sonically . . . . . 21

3.2 Composition . . . . . 24

3.3 Synthesis . . . . . 25

3.4 Sequencing . . . . . 26

3.5 Chords . . . . . 28

**4 Recall** **30**

4.1 Saving the State . . . . . 30
4.2 Drawing and Interaction . . . . . 31
4.3 Recalling Parameters . . . . . 33
4.4 Limitations . . . . . 33

**5 Evaluation** **34**

5.1 User Testing . . . . . 34
5.2 Evaluation . . . . . 35

**6 Conclusion** **39**

6.1 Skill Development . . . . . 39
6.2 Personal Reflection . . . . . 39
6.3 Future Development . . . . . 39
6.4 Ethical and Legal Considerations . . . . . 40

**References** **41**

**A External Materials** **43**

**B User Testing Materials** **44**

B.1 Questionnaire . . . . . 44
B.2 Results . . . . . 46

**C Audio Demo** **50**

# Chapter 1

# Introduction

The purpose of this project is to use generative art techniques to explore the spaces created by the works of the artist *Darrell Viner (1947-2001)*. Viner's work included movement, sound, and light, and though primarily working with sculpture, he produced a series of pen plotter drawings. These drawings have been called pioneering works in the field of computer art [13].

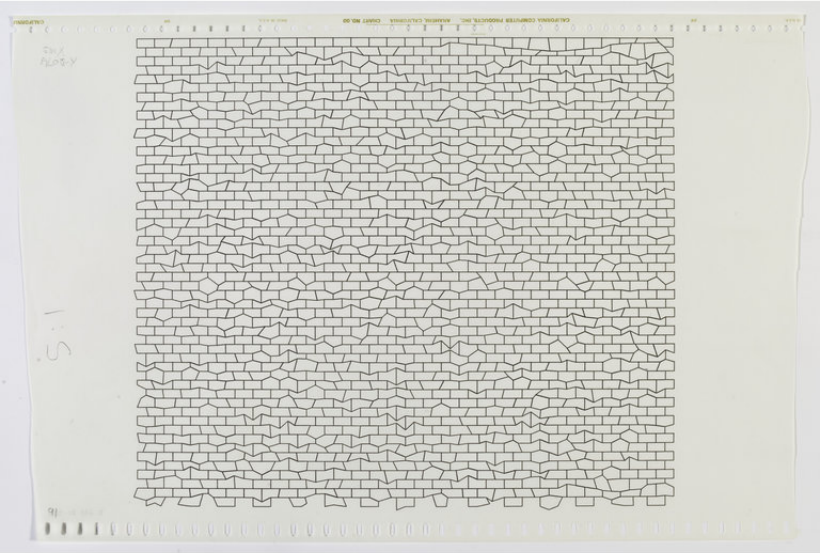


Figure 1.1: *Darrell Viner, Untitled (1974)* ©Victoria and Albert Museum, London.

Generative art is any art that is created with some system that produces an output, it can either be art that is completed by hand (e.g. a painting) with some system, or a stochastic process. Or, more commonly art that is generated by a computer program with some initial parameters.

The project will incorporate graphics and sound to create software that can be used to explore the landscapes present in Viner's works. The main problem that needs to be solved in this project is creating an interface that allows the user to interact meaningfully with the program and a set of parameters that might be changed to produce an image. As an extension of the graphics, a musical component will also be produced, this is less about Viner directly and more about creating an experience for the user that adds to the graphical aspects of the art.

## 1.1 Problem Overview

Viner spoke about his work being like a "townscape/landscape": "Basically it is a self-generating program which depends on the start conditions. Thus, by altering parameters of the program, changes in the final image can be achieved. Currently, I am after images which have the feel and scale of townscapes/landscapes. The program is modified depending upon whether I consider the images to be working or not: the program has become my personal aesthetic." [18]

These parameters may for example control the spacing of the grid, the displacement of each point, the number of vertices in a shape, so on. These parameters should be able to be changed to create some sort of 'landscape' or 'topography' that the user can explore through manipulating the program affecting both the entire grid and single points within it. Later on, I will detail methods for users being able to manipulate the parameters and considerations that need to be made concerning usability.

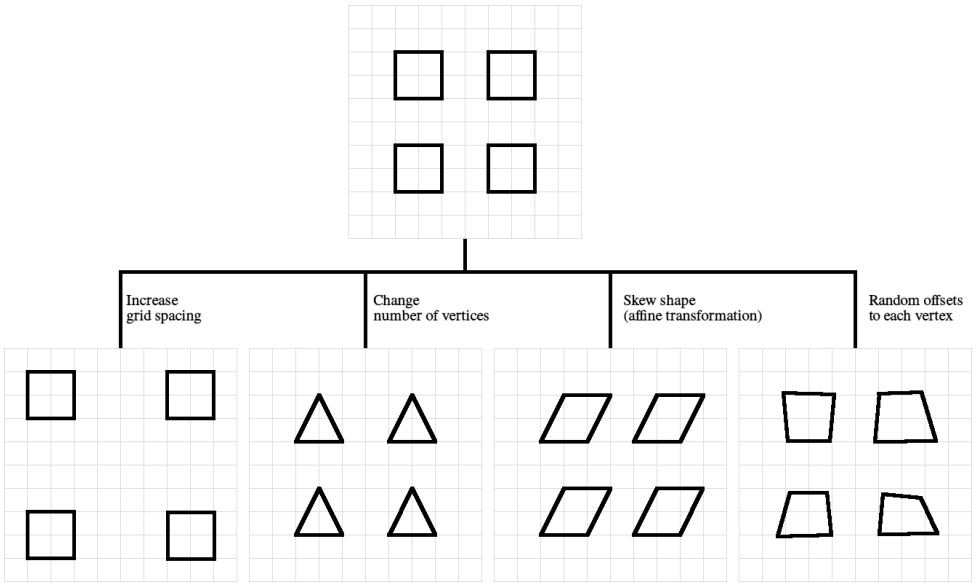



Figure 1.2: Example of changing parameters from an initial state

The program should allow for a user to generate such images and recall them later. Ideally, the program should be more than just a show of the potential configurations of Viner's work and itself be a unique experience for the user, working to create a dialogue with the original work and contextualising the user's understanding of his work.

Users of this program may be for example artists, art scholars, museum guests, museum curators. Each of these has different goals; the artist may be seeking to understand the work and find inspiration in it, the art scholar may be looking for an extended context to the work to help comprehend the thought process Viner used and the relationships between each piece. The museum guest may be at an exhibition (perhaps a virtual one) of Viner's work and have this presented as a piece of work that helps them understand the spaces that are explored by the work. A curator might be evaluating the program for display alongside Viner's work.

## 1.2 Aim

Using the idea of a virtual museum exhibition of Viner's work: this could be presented alongside a collection of images to show the relationships between each image where in a physical museum this may be able to be explained through the placement of the images in the physical space, online this is harder to convey with just a gallery on a web page.

The user should be engaged with the program and be able to become engaged with using it with minimal instruction. Through using the program the user should understand the concept of the 'landscape' described previously. To show the relationships between images and for the user to be able to explore these a method for seeing previous states of the program should be provided.

To create the graphics, a system needs to be created to display polygons in a grid of which each vertex can be offset separately. The calculation of these offsets and the number of vertices in each polygon is what will convey the idea of the 'landscape'. These offsets need to be calculated in a way that is repeatable so that the user can revisit parts of the 'landscape' both in the session and between different sessions. This will require the development of a set of tools to create an aesthetically pleasing image.

The audio component extends to the idea of an exhibit; a museum version of this would have a pair of headphones that the user can wear which would help isolate the piece of art from the rest of the exhibit. In the case of people accessing the piece online the same applies, with audio they can be more immersed in the art. Also, since the audio relates to the graphics on the screen, it can help users navigate the 'landscape' by providing cues.

Less focused on the exhibition example, through this report techniques that apply to other works of generative art and music will be outlined. The idea is that they may be reused. In this sense, the program also is for artists wishing to create pieces of work, and by presenting these techniques may use them for their ends; therefore, code should be reusable and understandable.

## 1.3 Objectives

- To create a program that allows the user to experience the works of Viner through the manipulation of parameters in a way that enables exploration and recall.
- Create an interface that allows users to navigate generated spaces.
- Create an audio component that works in tandem with the graphical aspect of the program to aid navigation and provide an extra layer to the experience.
- Create a system such that users can recall a state they were in, and find it again.
- Create a system such that users can download and upload the collection of states as described in the previous objective.

- Develop a catalogue of techniques to be used in generative art and music, and to evaluate each for relevance to the project.

## 1.4 Deliverables

An application written using `p5js` that should allow the user to explore a landscape, generated to feel like Viner's work. This application should be friendly to use and will include both graphics and sound. The feeling of some sort of topography should be conveyed and the visual should look similar to Viner's work. The audio will help the user navigate the space and correspond to what is on the screen graphically in some way.

The application should work in the browser, and be as compatible as possible with different screen sizes and web technologies, libraries allowing. Thus, not requiring hosting with a web browser, allowing for flexibility.

The structure of the final product will consist of three main components: the graphics which consist mainly of the grid; the audio which will be changed by the parameters that generate the graphics; and the history / recall part which will deal with data structures associated with saving the state of the parameters.

The graphics component is what will utilise the `p5js` library, although it will also be utilised in the history component. The audio part will use the `Tone.js` library. This is represented by the first five objectives (deliverables 1-5).

Also presented are two code examples of techniques used in this project that use p5js, in a `report-demos` directory. This is represented by the last objective (deliverable 6).

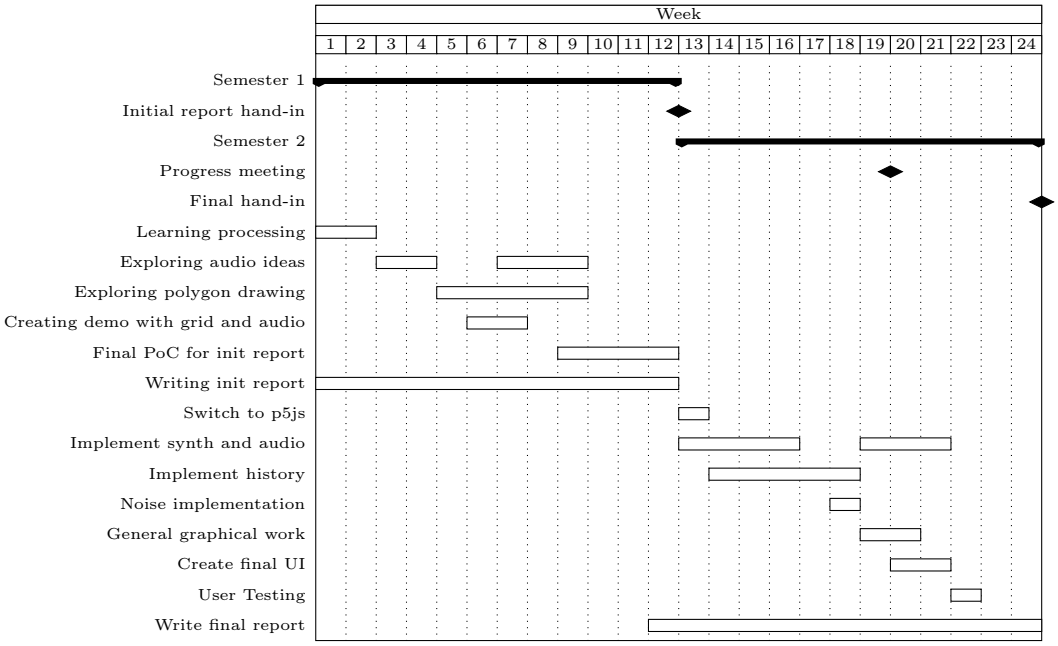
The final program demo will be hosted on `github.io` and sent as a deliverable (deliverable 7). But can be run from the files in the Github repository locally too.

## 1.5 Plan

Being that this project is exploratory, an iterative approach to development makes sense, with small prototypes being created, explored, and built on quickly. As such time has been allocated for ideas to be developed in, the idea is that smaller sub-systems (as in the objectives) are created and integrated together. Splitting the problem up will help avoid risk because systems can be integrated together without them being in some final state before starting the next.

To start I will learn about `processing`, a library for Java which is an artist-focused library for graphics and audio. Since my tutor's previous work was written in processing, there is already some groundwork complete to build on. I will then switch to `p5js`, a JavaScript library based on `processing` that is an offshoot and works in the browser using JavaScript. This switch will not take much time at all due to the program logic being identical. `p5js` in the browser has the

advantage of working with other JavaScript libraries well (such as `Tone.JS`, the audio library I will be using) as well as having the flexibility that comes with HTML and CSS for the user interface.




I will use `git` for version control, during development commits will be pushed to a private GitHub repository which, when complete will be made public and the `github.io` website-hosting feature will allow for the project to be available in full on GitHub for user testing and marking.

## 1.6 Risk Mitigation

Since this project is not dependent on external factors, there aren't a lot of risks to consider.

If the project had relied on access to the art directly, the COVID-19 pandemic would have affected it, but we mitigated this risk by setting the scope of the project to not include access to the art.

Another risk in this project is scope creep; since the project is artistic it can be easy to have lots of ideas whilst losing the original focus. To combat this I will set specific objectives and deliverables and reach those before exploring any further options.

# Chapter 2

# Graphics

To create the graphics we need to first analyse Viner's work to see the characteristics that need to be replicated; we then should create a method for drawing the grid of polygons. The vertices of the polygons need a method to be offset and the number of vertices in the polygon changed.

Where Viner's work was a static image and our is dynamic; we should also consider how the image changes as parameters change. The points must move smoothly between configurations to give the illusion of being in that 'landscape' rather than that of a series of static images.

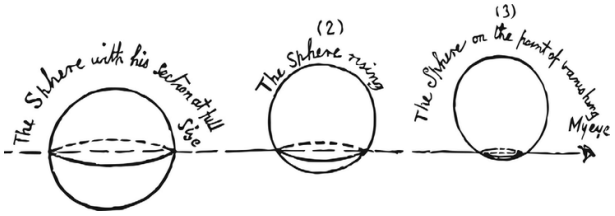



Figure 2.1: Figure reproduced from Flatland [1, p.112]

The landscape is not just 2D, parameters each constitute a dimension, when one is changing we see a 2D slice of the multi-dimensional world as in the book 'Flatland' "You cannot indeed see more than one of my sections, or Circles, at a time; for you have no power to raise your eye out of the plane of Flatland; but you can at least see that, as I rise in Space, so my sections become smaller. See now, I will rise; and the effect upon your eye will be that my Circle will become smaller and smaller till it dwindles to a point and finally vanishes." [1, p.112] A shape (polytope in this instance, sphere in the book) cuts through the plane to show 'slices' of itself. Here we are in 'flatland' and the parameter space we create allows us to see the work as slices of a higher-dimensional shape (than 2D).

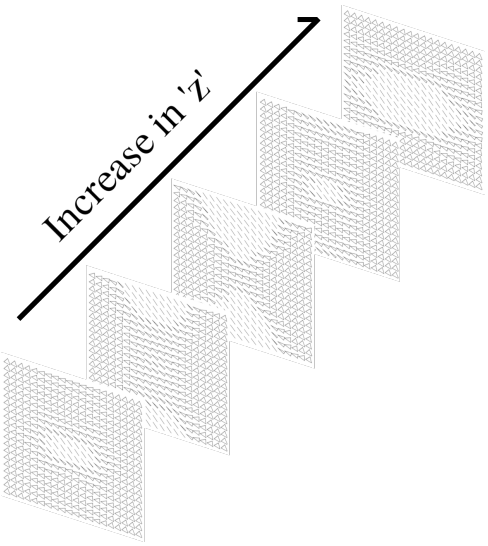



Figure 2.2: Here the 'z' parameter increases, each 'layer' is morphed between smoothly

## 2.1 Anatomy of Viner's Work

Viner's pen plotter work was created by a set of programs, created in `FORTRAN` using a set of subroutines called `PICASO` (PIcture Computer Algorithms SubroutineOriented) [7], this was essentially a graphics library, similar to what we are using processing / p5js for. This library was presented as part of J. A. Vince as part of their PhD. `PICASO`'s use by Viner isn't well documented but the manual has subroutines for transforming vertices according to some rules (affine transformations, morphs between shapes) [17]. Some pencil notes on Viner's work indicate there was mathematical thinking going on in the development, but without access to the code listings, it is hard to say exactly how the images were generated.

## 2.2 Polygons

One aspect the program should be able to do is to grow and shrink polygons between different numbers of vertices. Viner's work uses different numbers of vertices between images, so we should be able to reproduce this by allowing a smooth transition between polygons of a different number of vertices.

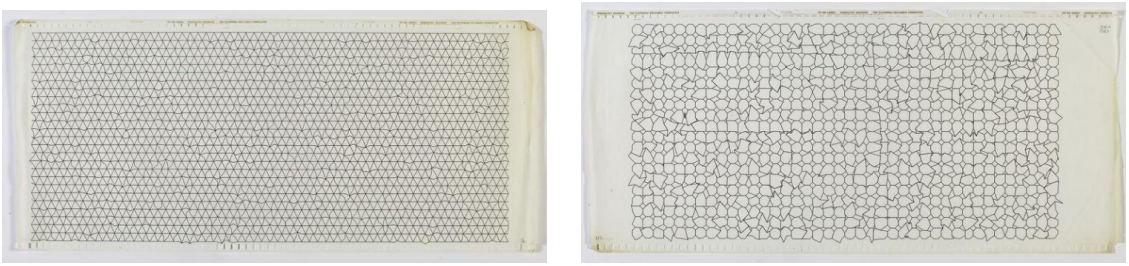


Figure 2.3: Two Viner pieces showing the use of polygons with different numbers of vertices

To achieve this a polygon is inscribed on a circle. The points of this polygon of $n$ vertices on the circle are simply given by

$$(x, y) = (\cos(\frac{2k\pi}{n}), \sin(\frac{2k\pi}{n})) \text{ where } k = 0, 1, \ldots, \lfloor n \rfloor$$

We can note that for integer values of $n$ the rotation will be complete, for non-integer values of $n$ this also works given that we used the floor of $n$ as the limit giving us an incomplete polygon, we can use the fact we have the first coordinate to close the shape. This gives us the smooth transition between integer values of $n$.

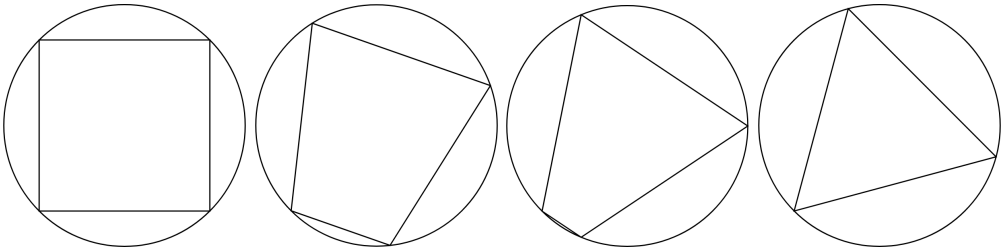


Figure 2.4: Starting at $n = 4$ the shape deforms through $n = 3.5$, $n = 3.2$ and finally $n = 3$

In code we can implement this as such:

```
// center
let x = innerWidth/2;
let y = innerHeight/2;
let n = 4;
let r = 300;

let theta = (3*QUARTER_PI) - TWO_PI/n;
dTheta = TWO_PI/n;

beginShape()
    for (i = 0; i <= n; i++) {
        theta += dTheta;
        vertex(x + r*cos(theta), y + r*sin(theta));
    }
endShape(CLOSE);
```

An offset to each $(x, y)$ pair can be made inside the loop. A function to limit the maximum offset should be created too. Here is an example of using noise to calculate the offsets.

```
ox = limit(noise(sx*cos(theta), sz)*10, 2*gridSize);
oy = limit(noise(sy*sin(theta), sz)*10, 2*gridSize);
```

The limit function is a simple sine calculation ensuring the offsets don't become larger than $\pm$ 2 times the grid size.

```
function limit(value, amplitude) {
    return amplitude * sin((PI / amplitude) * value);
}
```

## 2.3 The Grid

The motif of a grid with changes present across the image is central to Viner's work, so a method of drawing this should be created.

The centre point of the grid is assigned a coordinate `x,y,z` which is what is modified by the user when interacting with the program. The polygons around this point are calculated from that offset using the following calculation:

```
sx = (x + ((gridSize+gridSpacing) * ((cols/2) - i)));
sy = (y + ((gridSize+gridSpacing) * ((rows/2) - j)));
sz = (z + (gridSize * (cols/2)));
```

Where `i,j` is the column and row value of the point, `cols, rows` are the total number of columns, rows, and `gridSize` is the pixel size of the grid and the similarly for the `gridSpacing`.

This leads to the idea of a 'view window' of which the parameters in the program describe only the central polygon and the parameters for the others in the grid are calculated.

The grid should also be centred on `x,y` so calling a translate before drawing any points should be done:

```
translate((innerWidth/2) - x, (innerHeight/2) - y);
```

Here a grid is prepared with fixed parameters that distort around `x=0, y=0`. This point is fixed, and when the program is interacted with the world moves beneath the player rather than the point of distortion being changed, the grid displayed is reflecting the 'terrain' visible from the centre point.

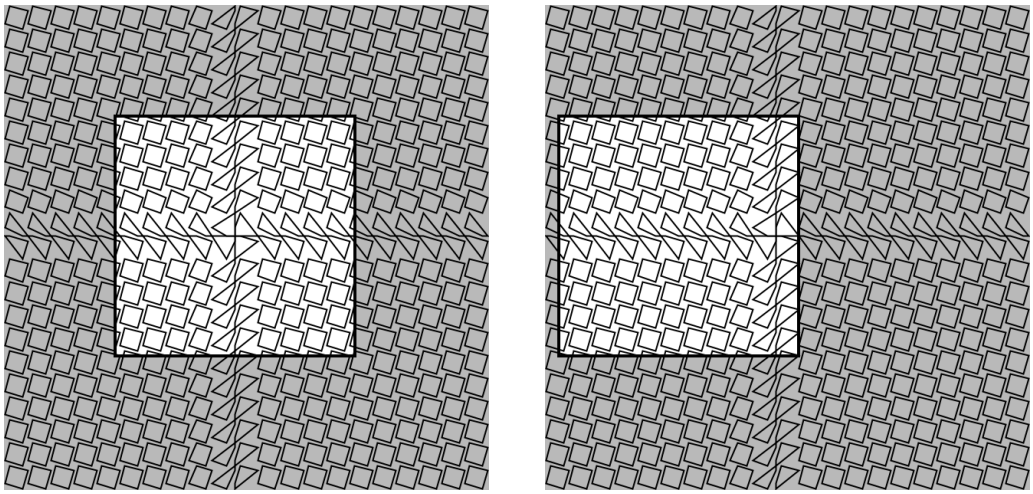


Figure 2.5: The user presses ‘a’ for a short time, decreasing the ‘x’ parameter

We then use `sx, sy` to draw a polygon using the method described previously, with the `sx, sy` as the centre.

Another possibility was considered to make the grid a simple series of vertices and have a method to draw lines between them according to some rule. This was ultimately rejected due to much more complexity when moving between a number of vertices in the polygon dynamically.

## 2.4 Landscape Generation

In the introduction we said that the program should generate a ‘landscape’ or ‘topography’. We can therefore understand the problem of creating offsets and setting the number of vertices in a shape to be that of creating a function that can produce different ‘regions’ with a fairly regular set of properties, then having a gradient between these regions. Looking at a topographic map we can see this, with the ocean being separated from the land by a beach, and the low-lands separated from the high with hills, other geographical features like valleys or cenotes and so on. This is opposed to the idea of a program that takes a change in the parameters and displays it throughout the entire grid with no concept of ‘regions’. This model needs to be able to take the parameters of the program and reduce them down to a single value for use to create an offset or number of vertices in a shape.

Dynamical systems may be of interest and allow for a system to be created where a state evolves into other states following some rules. These states can be deterministic which is important for the objective that we have of recall, but can also be chaotic, which may be aesthetically desirable. Similarly, fractals may be useful for their self-similarity. Given we’re working with a grid, the ability to have self-similar properties may be aesthetically useful. Neither of these however would be particularly flexible as one function would need to be picked and used throughout the whole program. Also, to model a landscape with the right-sized ‘regions’ may prove difficult.

There is then the choice between having each session using the software be either random in some sense or the same every time. Ideally given a random option to fulfil the requirement to recall previous sessions. A seed would be given and the program can have a deterministic outcome.

For this reason, noise is a good solution, the majority of noise implementations allow you to specify a seed and thus have this property of reproducibility between sessions. Graphical implementations of noise are often created with the express reason of looking natural and organic.

### 2.4.1 Noise

A desirable property is that of a smooth gradient between extremes, or defined regions in which values are high and low. This will help to demarcate 'regions' in the map analogy.

`p5js`'s built-in noise function is a Perlin noise generator. You can pass it up to three coordinates. Perlin noise was designed for computer graphics [1], and is fairly simple, generating a random grid of vectors and then computing the dot product vectors and their offsets, then finally interpolating to create a more smooth image [11].

`p5js` also provides a `noiseDetail()` method that provides some control over the 'texture' of the noise. Also for reproducibility `noiseSeed()` allows the programmer to set a seed value for the noise. The `noise()` function takes three coordinate arguments and outputs a number between $0 - 1$

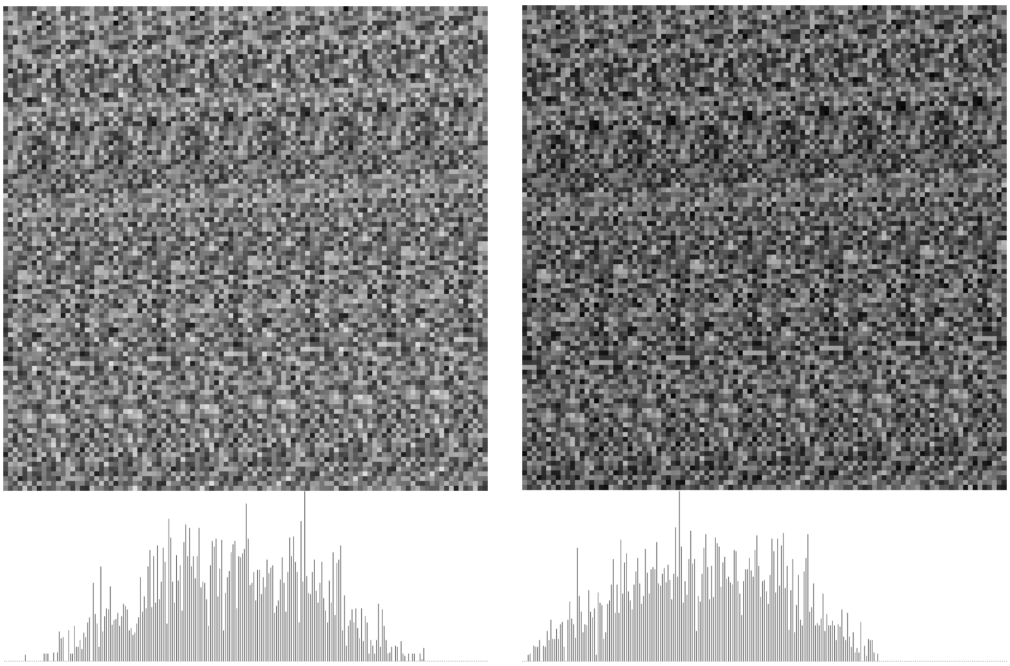


Figure 2.6: Two Perlin noise samples (one default the other with noiseDetail(2, 0.5) with the same seed, histograms plotted beneath show that the adjustment makes the image overall darker

[1] And won an Oscar for "allow[ing] computer graphics artists to better represent the complexity of natural phenomena" [10]

One option we may wish to be able to have is 'quantising' the noise to some set of values. For example, we may wish to have areas of a certain value of $n$ for a polygon. This can be achieved by mapping the noise from $0-1$ to $0-n$ and rounding to the closest integer. This was happening implicitly with the histogram above. We can see that because the distribution is uni-modal the most common values will be those towards the centre of the range.

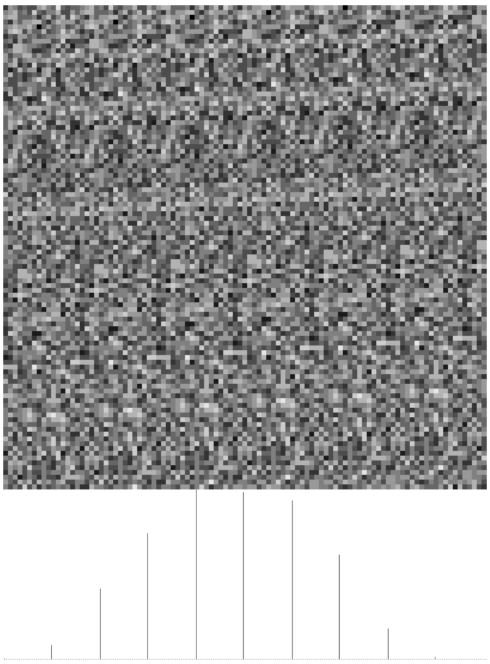


Figure 2.7: This sample contains only ten colours

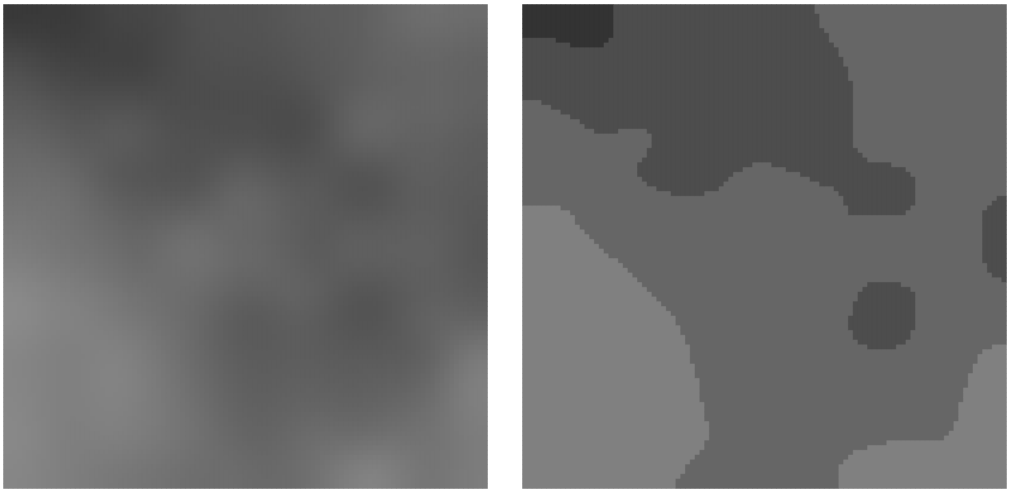


Figure 2.8: Unquantised and quantised noise at 0.001 noiseScale

This effect is much more obvious when the noise scale is a lot smaller, this would then allow for discrete regions between two integer values for a parameter. What if we don't want discrete regions and instead something more like a multi-modal distribution i.e. smoothing applied between edges?

We can use a waveform such as the sawtooth to achieve 'smoothing' by using it to estimate

rounding, adding the value to the wave produces something approximating the 'staircase' function of piece-wise rounding. To do this we can use additive synthesis:

$$x + \frac{1}{\pi}(-\sum_{k=1}^{\infty} \frac{\sin(2k\pi x)}{k})$$

To use this in code we take only some number of terms, this determines the accuracy of the rounding, the lower the more 'smooth', i.e. inexact the output:

```
function approx_round(value, terms) {
    let result = value;

    var innerSum = -sin(2 * PI * value);
    for (i = 2; i <= terms; i++)
        innerSum += (sin((i * 2) * PI * value) / i)

    result += innerSum / PI;

    return result;
}
```

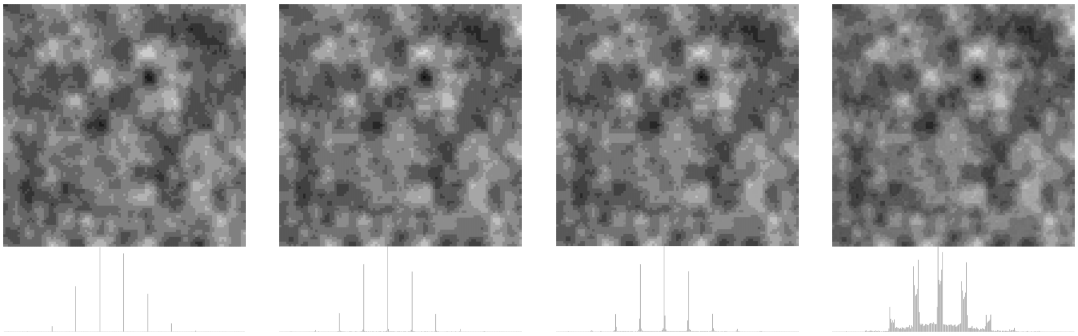


Figure 2.9: Rounded discretely, Terms = 100, Terms = 20, Terms = 1

At such a large scale the difference is hard to perceive but when using the parameters in Figure 2.8 there's a clear 'smoothing' in the boundaries.

Figure 2.10: Terms = 100, Terms = 10, Terms = 1

Implementing this such that noise controls the number $n$ (vertices in a polygon) we can get results like this (note that the background is shaded darker for lower values of $n$ and lighter for higher values of $n$)

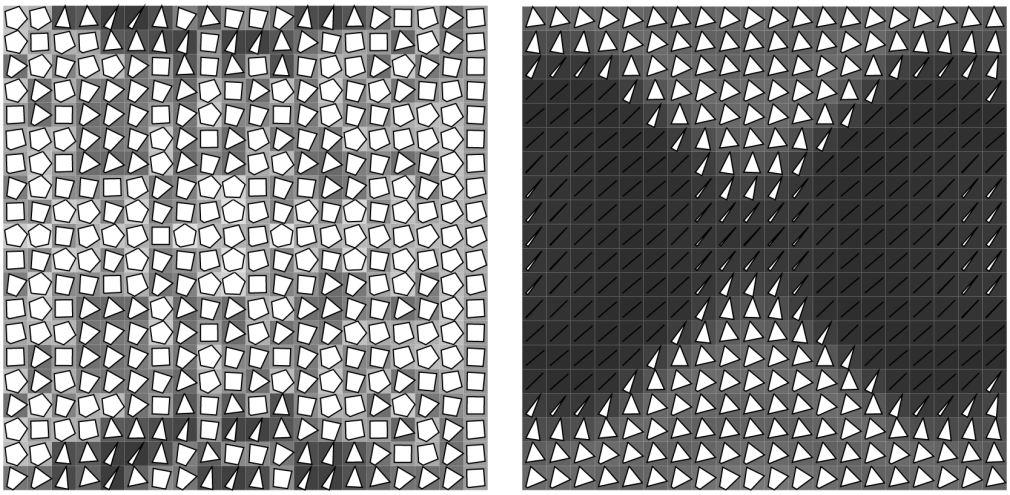


Figure 2.11: Terms = 20, Terms = 1

With user interaction, this scheme creates the feeling of moving through regions of these different values of $n$ and the parameter for the number of terms allows for adjustments to be made about how 'hard' or 'soft' the borders between these regions look.

This approach is, I believe somewhat novel, as most techniques for adjusting noise is usually done at a whole-image level. This method is point-wise and thus can work for each polygon or vertex in the grid we have defined. In the field of signal processing the concept of 'quantisation noise' is similar but reversed, taking a continuous signal and analysing the error in the digital discrete signal. Here we are designing a function for an artistic application, and I do not think that it has application to the field of signal processing as it works on discrete to discrete transformations and not continuous to discrete.

## 2.5 Integration

These parts then tie together to make the whole graphical system, the polygons are drawn at each point in the grid and given an offset by the noise; this is all drawn within a `push()` and `pop()` so as to not affect the history tree (detailed later). Here is one iteration of the loop through the columns and the rows:

```
// calculate x y and z values for the current polygon
sx = (x + ((gridSize+gridSpacing) * ((cols/2) - i)));
sy = (y + ((gridSize+gridSpacing) * ((rows/2) - j)));
sz = (z + (gridSize * (cols/2)));

// From section on polygons and noise, generate a number from 1 to 6 for
```

```
// the number of verticies
n = approx_round(map(noise(sx*0.001, sy*0.001, sz*0.001), 0, 1, 1, 6), 1);

let theta = (3*QUARTER_PI) - TWO_PI/n;
dTheta = TWO_PI/n;

beginShape()
    for (k = 0; k <= n; k++) {
        theta += dTheta;
        // Calculate offsets to points
        ox = limit(noise(sx*cos(theta)*0.001,
                         sz*0.001)*noiseMultiplier,
                   2*gridSize);

        oy = limit(noise(sy*sin(theta)*0.001,
                         sz*0.001)*noiseMultiplier,
                   2*gridSize);

        // Points on the circle
        xc = sx + ox + gridSize * cos(theta);
        yc = sy + oy + gridSize * sin(theta);

        // draw the vertex
        vertex(xc, yc);
    }
endShape(CLOSE);
```

In this code snippet we can see that most of the above code is replicated but with some changes in multipliers (e.g. values in noise are multiplied by 0.001) these are changes made to produce aesthetic results in the final output.

## 2.6 Navigation

Because this program deals with a set of parameters that change over time, it is a good idea to look at concepts for navigating higher-dimensional space that already exist; in this, we can learn how other people have approached this. If we imagine each parameter in the program as a spatial dimension, there is a body of knowledge available regarding this because many fields rely on large multivariate data; for example the field of machine learning for example deals with very high dimensional data in feature vectors.

Tools like '`ggobi`' exist to help explore multi-variate data [4] these are flexible with many visualisation methods but tend to disregard spacial relations of points in favour of clustering

based on shared properties. A novel method is that of the grand tour, through which a series of scatter plots are projected orthogonally into 2D subspaces from the higher-dimensional space and moved then between continuously to describe multi-variate data [2].

Whilst these approaches are useful for data, what we're dealing with here is generating spaces to explore, these spaces are less about trends in data and more about geometric exploration through many continuous parameters. In this, we want all of the data of the parameters to be visible at once, meaning these data visualisation methods are not as useful for our purposes.

Figure 2.12: Mario turns 90° around the z-axis (*Super Paper Mario (2007) pub. Nintendo*)

Video games also have explored higher-dimensional spaces. An interesting concept is that of 2D characters being able to navigate 3D spaces. *Super Paper Mario (2007)* for the Nintendo Wii is an example of this, the player can turn the world 90 degrees about the y-axis to explore the depth of the z-axis and progress through the game. An extension of this from a 3D character turning 90 degrees and being able to explore the 4th Dimension is present in the unreleased game *Miegakure* in which the player can move through the 4th Dimension using the controller, to help orient the player a graphic[2] is displayed below the controlled character showing the position of the slice of the 4D world, these slices are like the 2D slices we can see in MRI imaging but in 3D.

Whilst these worlds explore pre-made or procedurally generated assets (models, textures, sprites, etc.), and this program instead will explore moving through various parameters, ideas for how the user can interface with the program to understand where exactly they are.

### 2.6.1 Graphical Prompts

A simple idea to allow the user to recall where in the parameter space they were could be a spider / radar plot or parallel coordinates plot. For our intents, these are the same as we're only displaying a single set of parameters the spider plot is just a 'round' version of the parallel coordinates plot. This could exist on the screen somewhere and change as the user moved around the parameter space, and would allow the user to recall approximately where they were. The spider diagram could also be 'extruded' from how it looked at every point to create a 3D model that represented how you moved through the parameters.

[2]Of which the developer told me in an email is an Astrolabe

However, this method might be unintuitive and remove the user from the main focus of the program. The grid should reflect the changes in the parameters in a way that becomes apparent with some exploration; making this somewhat redundant. Other options like this can be considered for other projects, but here there is no need. The idea of history existing in a third axis is interesting for further applications, however.

### 2.6.2 Controls

The above-mentioned video games allow for the user to move through space by only letting the user worry about at most 3 dimensions at any one time and regarding the others as static when moving through them, this allows the user to control the movement with traditional controls (either a games controller or keyboard and mouse).

In Figure 2.5, the `WASD` style of control, popular in video games is used. This will be familiar to people who have played video games but not to people who haven't, for whom it may make sense to use the arrow keys. It's also important to note that in that particular demo the 'd' key increases `x`, this gives the feeling that the world is moving 'beneath' you instead of you moving the world itself; if the controls were flipped such that 'd' decreased `x`, it would instead feel like you were moving the world. This is extended for other buttons. For example, the use of 'e' and 'c' are common to move up and down in video games (e.g. Second Life). A similar pattern using 'ijkl' is present on the right-hand side of the keyboard that allows both hands to be used.

In the end to control all of the parameters the keys 'w, a, s, d, e, c, q, z, i, k' were used. Each controlling a different parameter. 'w, a' corresponding to y, 's, d' corresponding to x, 'e, c' corresponding to z, 'q, z' to the spacing of the grid, and 'i, k' to the amount of noise displacement given to each point.

Another option would be something like an array of knobs, these could encode one parameter each, and as a scientific instrument be 'tuned'. However, this relies on specialist hardware. This would probably feel less like 'moving' through space and more exploring a range of possibilities.

## 2.7 User Interface

The final graphical element to consider is how the user interacts with the HTML elements that control various functions. I created: an upload and download button for interaction with the history; a mute button and volume slider for the audio; and a help screen that explains the key controls and what the program is.

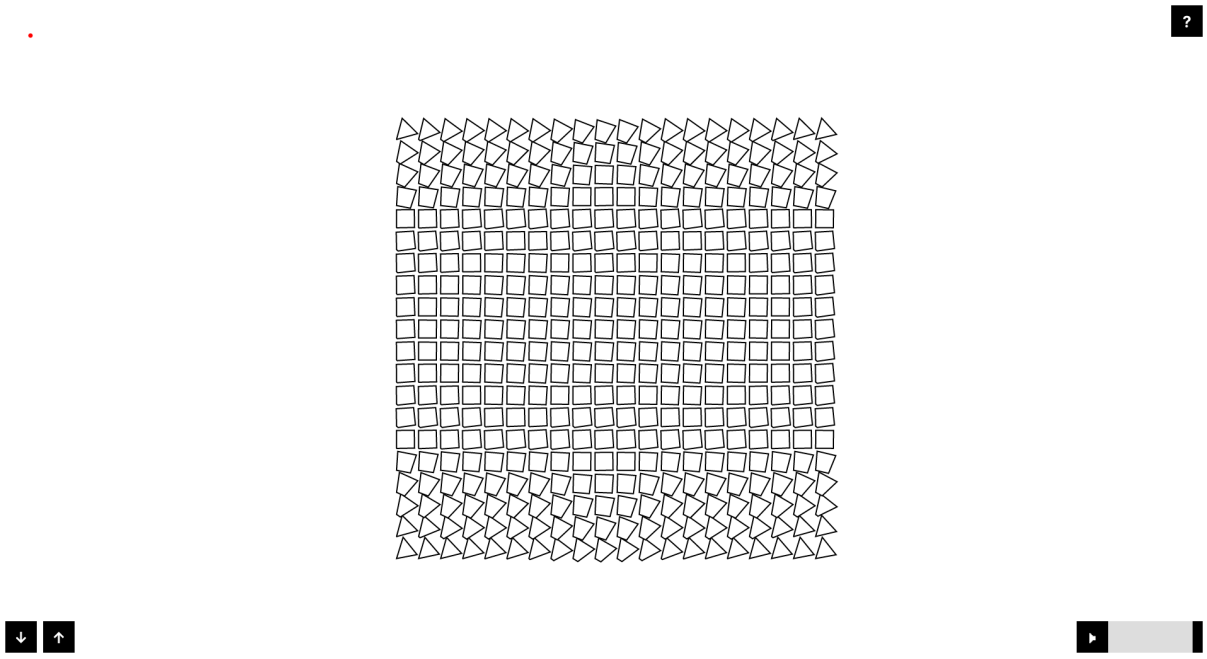



Figure 2.13: Web interface with buttons

I decided to keep the buttons simple, and without text to not distract from the main graphical part of the program. They also animate when the user hovers over them (they gain a black border and grey background) to indicate they are can be clicked.

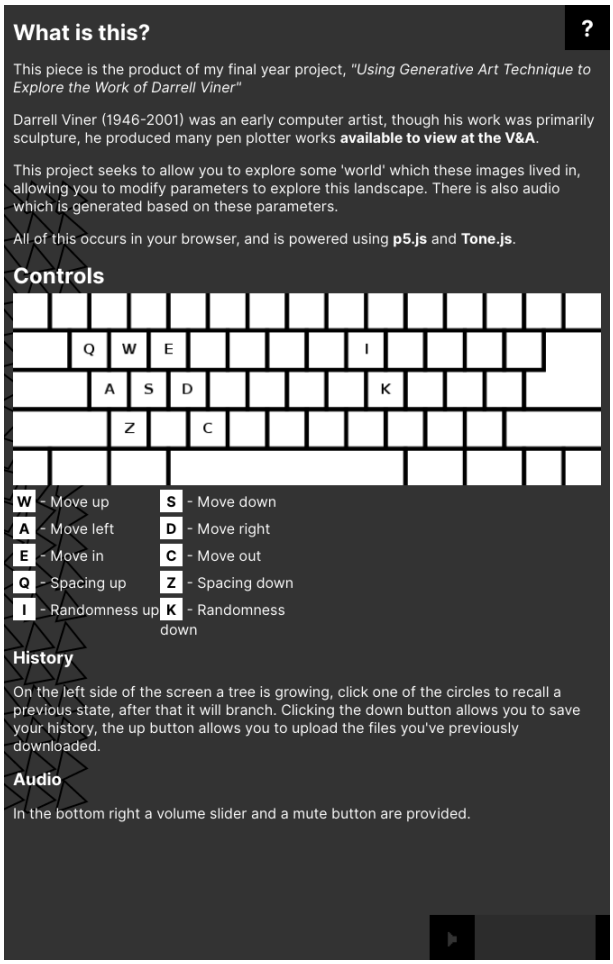



Figure 2.14: Help menu

The help menu expands when the '?' button in the top right is hovered and retracts when the user un-hovers the help menu. The background is slightly transparent to not block the rest of the program. I produced a graphic of a keyboard to help explain where the keys are located on the users' computer and a flex-box list of keys (that adapt to different screen widths). I also included a short explanation of how the history tree works and what the audio controls do.

# Chapter 3

# Music

## 3.1 Navigating Sonically

I have developed a theoretical system for using sound to navigate a higher-dimensional space, using ideas from *Harry Partch* and *Joe Monzo's* approached to tuning theory.

This is a process to generate intervals based on a set of spatial coordinates, or simply a set of parameters. Taking inspiration from the *just-intonation* tuning method a ratio comprised of integers only sounds consonant.

Tuning theory is a fundamental question when it comes to developing musical systems, the common myth as told by Iamblichus is that Pythagoras was "walking near a brazier's shop, he heard from a certain divine casualty the hammers beating out a piece of iron on an anvil, and producing sounds that accorded with each other." [16, p.62] Pythagoras had discovered that hammers of certain weights were consonant with each other musically. This leads to the concept of intervals.

### 3.1.1 Intervals

An *interval* is simply a ratio of two frequencies: $\frac{f_1}{f_2}$. Pythagoras' hammers were consonant as their weights were 12, 9, 8, and 6. The ratios here of $12:6$ correspond to $\frac{2}{1}$ or an octave, similarly $12:9=\frac{4}{3}, 12:8=\frac{3}{2}$ for the other two. Whilst this myth is probably apocryphal, the use of ratios of strings for music has been used since antiquity. What Pythagoras proved is that intervals are these ratios and that materials have properties from which these frequencies arise.

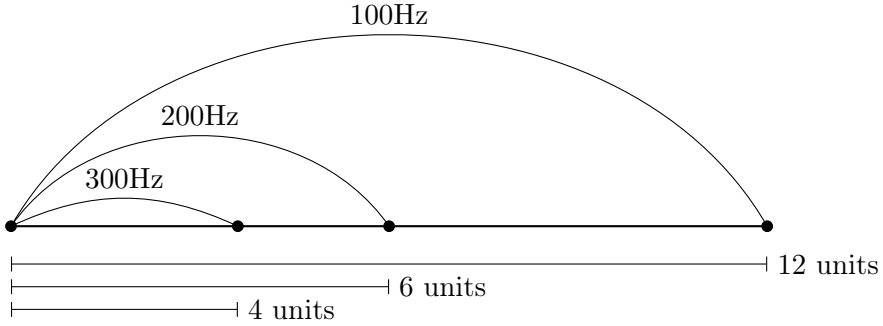



Figure 3.1: Diagram showing intervals being related to the idea of a ratio of lengths for example the 300Hz frequency is related to the 200Hz frequency by $\frac{6}{4}=\frac{3}{2}$

Pythagoras then set about creating a tuning system based around the $\frac{3}{2}$ ratio (the perfect fifth). The perfect fifth is the most consonant ratio that is not unison or an octave [1]. Using this ratio we can combine the numbers 2 and 3 to different powers to generate intervals. For example $2^0 3^0 = 1$ or unity, other intervals can be generated but we run into a problem. Using this method of moving in $\frac{3}{2}$ increments if we try to go 7 octaves i.e. $(\frac{2}{1})^7 = 128$ vs 12 fifths [2] we get $(\frac{3}{2})^{12} = 129.7463379\ldots$ the ratio of these two different values is $\frac{(1.5)^{12}}{2^7} \approx 1.01364$. In essence the comma's existence can be surmised by: there exists no $n \neq 0$ such that $2^n = 3^n$. This is important as it marks the difference from the system we use in the west today called '12-tone equal temperament' which divides an octave into 12 equal tones of $2^{\frac{1}{12}}$ to solve this problem.

For our purposes we will be using intervals, ratios of frequencies, to generate new frequencies or notes for use in the program. If we generate a ratio then we may simply multiply a frequency to find the frequency of the other note $x \cdot \frac{f_1}{f_2} = y$.

A *p-limit* tuning system is a just-intonation based system where every interval's highest prime factor is $p$. [9, p.76, 109] Pythagoras' tuning system can be said to be 3-limit. These intervals can be represented using an exponent vector, usually called a 'monzo' [8]; for example, $3:2$ is represented as such: $|-1\ 1\rangle$. This is simply a shorthand for $2^{-1}3^1$, and can be extended: $|e_1\ e_2\ \cdots\ e_n\rangle$ where each $e_i$ in the vector is an exponent of a prime number $2^{e_1}3^{e_2}\cdots p_n^{e_n}$.

Often these vectors are shown as $|*\ e_2\ \cdots\ e_n\rangle$ The $*$ represents the idea of octave equivalence, a musical idea that a doubling of the frequency produces a note that is 'the same', and as such any value of exponent for the 2 can be placed there. As an example $|1\ 1\rangle = 2^0 3^1 = \frac{3}{1}$ but this is equivalent to $\frac{3}{2}$, as you can simply halve the frequency produced by the ratio to get back to it. To normalise the interval the same octave, set the constraints $1 \leq \frac{f_1}{f_2} < 2$ and similarly for other octaves higher or lower, just halving and doubling the bounds.

### 3.1.2 Navigating Space

#### In 2D

Assume a base frequency, 440Hz $= f_1$. As an example, if the user was at $x = 1$, $y = -1$.

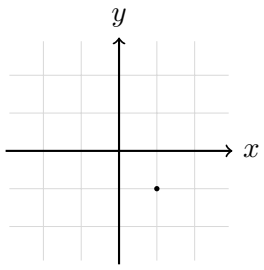



Figure 3.2

[1] It is named so because of musical convention, it being five notes between notes in a scale, it has no direct relation to the ratio of 1.5

[2] These should be the same see: the circle of fifths

This is represented in the vector $|0\,1\,-1\rangle$ which corresponds to $2^0 3^1 5^{-1} = \frac{6}{5}$. $\frac{6}{5}$ is a minor-third, and represents 528Hz, this is consonant.

Or in general:

$$2^n \cdot 3^x \cdot 5^y \cdot f_1 = f_2$$

If the user is at non-integer numbers for $x$ and $y$ we can calculate the frequencies too.

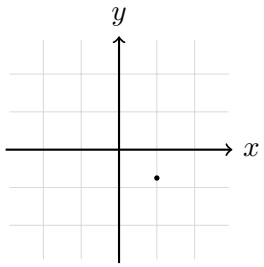



Figure 3.3

Where $x = 1$ and $y = -0.75$ we can simply find the frequency as
$2^0 \cdot 3^1 \cdot 5^{-0.75} \cdot 440\text{Hz} = 394.772\text{Hz}$. This will sound dissonant and not particularly nice.

The idea here is simple, generating intervals like this allows for some points in space to be consonant and others to be dissonant, these could correspond visually with some output being produced by the program but should hopefully allow a user to differentiate between points in space using their sense of hearing.

#### In Higher Dimensions

This idea extends into higher dimensions naturally, simply adding to the number of terms in the vector $|*\ e_3\ e_5\ \cdots e_p\rangle$. Each of these exponents could represent some parameter added to the program.

### 3.1.3 Limitations

As more variables are introduced integer ratios will sound less 'strongly consonant' and may be harder to understand naturally. However, every interval that is possible with fewer variables will still be possible so a user could still explore one or two parameters at a time and get the same results.

### 3.1.4 Processing Demo

I have created a demo of this concept in 2D in processing, the code is in Appendix C. The program simply has an integer-marked grid that the user can navigate using `WASD` and will calculate the interval based on the coordinates of the point. Similarly to Figure 3.3:

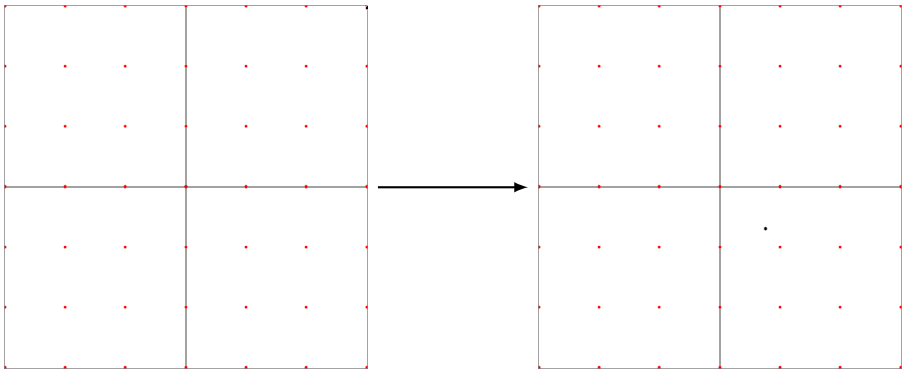


Figure 3.4: At first the black dot is hidden beneath the centre red dot, the user uses WASD to navigate, the audio is out of tune in the second state

This demo is simple and only uses two sine waves so is unpleasant to listen to, but illustrates the point, at points outside those marked there is a 'beating' sound to the waves as they do not harmonise correctly. However, as some points aren't marked that *do* sound somewhat consonant, so this cannot be the only navigation method , only a prompt to help the user.

## 3.2 Composition

To create a sound accompaniment to the visual aspect we first need to consider what style of music we should explore, that is to say, create at least an idea of composition. Generative music is the technique we're exploring here, with musical ideas being emergent from a set of parameters, and created from a system that processes them. The definition is vague as with generative art, the main requirement is that a system is set up and creates the music, this doesn't need to be a computer, but often is.

Generative music is rather recent, with Brian Eno being major figure popularising its use, he would often use the analogy of a Moiré pattern to describe how the programs would work at the time.

The immediate style of music to draw from Viner's work would probably be that of minimalism, the repetition and difference across an image is similar in style to especially the percussive works of Reich, Glass, Riley, and even the drone works of La Monte Young to an extent. Another, less famous example that I feel conveys the feeling well is *Jon Gibson - Cycles (1977)*, the cover for the recording is a Moiré pattern, and the work modulates a 7-note pattern that comes into and out of phase with itself.

Eno refers to his inspiration of generative music to be triggered by hearing *It's Gonna Rain* by Reich, linking generative music as a concept pretty solidly to Minimalism. Reich used the analogy of fabric work and weaving, featuring on the cover of Music for 18 Musicians is a woven piece of fabric; this seems similar to the idea of a grid (given that weaving takes place on a matrix of strings, perhaps the crossing points could be seen as 'vertices')

Overall, the composition should enhance what is on the screen, using the parameters that are being manipulated by the user to inform what sounds are created for example: if the number of vertices in the shapes on the screen changes the music should change accordingly.

Further, this will be combined with the methods mentioned in section 3.1, including the ideas present in the mentioned minimalist composers. This will be an element in the work that is unique from the art and perhaps helps distance itself from being purely a replica of Viner.

## 3.3 Synthesis

Broadly there are four approaches to digital synthesis, that of: the processed recording; the spectral model; the physical model; and the abstract algorithm [15].

For this project, spectral and physical modelling is out of the scope and would require more specialist audio software frameworks. Processed recording, includes more sample-based audio and manipulation of that, granular synthesis is an example. The 'abstract algorithm' methods include things like FM synthesis, which in its most basic form is comprised of a carrier waveform whose frequency is modulated by another waveform, this can be extended by things like including feedback at various stages of the processing.

On top of these methods, there should also be a consideration to audio effects, processing and p5.js both have sound libraries with built-in audio effects, reverb and delay perhaps being the most important to create the idea of a space in the sound.

Creating these methods in software is straightforward when there is already some digital signal processing in place; a library called `Tone.JS` handles the practicalities of generating sound in JavaScript and works alongside `p5.js` nicely.

Additive synthesis makes sense for use here there are a lot of controls over parameters, it is easy to implement and can reproduce many different qualities of sound. As in the composition section, work like 'Cycles' is performed on the organ for example. This is an easy sound to replicate in additive synthesis as you can achieve the effect by playing multiple sine waves at the same time; this is what the earliest pipe-less organs did. Whilst not in software, they would have series of plates or disks that encoded the waveforms and played them all at once through a mixer to create complex waveforms [3].

### 3.3.1 Additive Synthesis in Tone.JS

First I have created a bank of oscillators that can be tuned to any frequency and triggered. I can loop through these and use an 'envelope' consisting of to control the volume over time of all of them at once, this is simply set as such:

```
const env = new Tone.AmplitudeEnvelope({
    attack:1,
    decay: 2,
    sustain: 1,
    release: 0.3
});
```

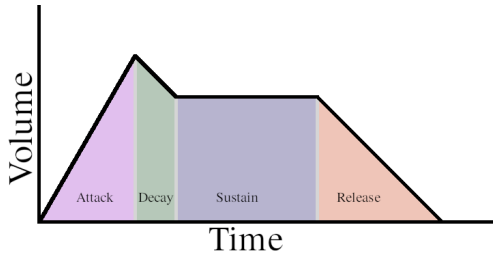



Figure 3.5: Typical attack decay sustain release (ADSR) curve

The number of oscillators in a bank can be chosen at will and for our purposes are tuned as per the harmonic series this is to create a pleasing timbre (the 'summation' or additive part is when we feed them into a mixer). Each oscillator is labelled with a number that corresponds to the term of the harmonic series, when the attack for each oscillator is triggered the frequency is calculated as the incoming frequency times the number of the term. Also, each oscillator will be quieter than the last according, this is again to create the timbre.

Since we are adding these oscillators' signals together, consideration needs to be given to the phase angle that each are set to. Setting them to `(i / NUM_OSCS) * 360` allows for there to be no harsh constructive or destructive interference between the oscillators.

## 3.4 Sequencing

To create a sequence I considered a couple of options, to have a fixed note sequence for example. This would perhaps be too static and not fit with the idea of the project being generative art. Another simple option was to make notes completely random, this tends to sound 'robotic' like a Sci-Fi movie control panel. There could also be a random choice within a set of consonant notes (i.e. that of a chord) this is a better idea because there's less chance of a 'bum note' (one that sounds out of place).

An extension of this idea is to use Markov Chains to model music in [6] for example, they analyse Bach, Mozart, Palestrina, and Beethoven to create a set of transition matrices for their

chord progressions. In this example, chord progressions are used but the method can easily be adapted to note progressions too.

For my sequence, I picked six possible tones, expressed as intervals from a base tone, as $1, 1.2, 1.25, 1.5, 1.6, 2$ these represent unison, a minor third, a major third, perfect fifth, and a minor sixth, and the octave (all just-intonation). These tones were picked because they were the intervals in 5-limit that didn't have recurring decimals.[3] Each tone is also then affected by the interval generated above to create a possible series of tones that depend on the parameters of the program.

To create a Markov chain sequence first a matrix of probabilities that represent the chain needs to be created. This defines the possible transitions between notes and the probabilities of them, each row should add to give a total probability of 1. Changing the values in this matrix can therefore be thought of as composing what series of tunes are possible and probable.

$$\begin{array}{c} \\ 1 \\ 1.2 \\ 1.25 \\ 1.5 \\ 1.6 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 1.2 & 1.25 & 1.5 & 1.6 & 2 \end{array} \\ \begin{pmatrix} 0 & 0.2 & 0.5 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.3 & 0 & 0.1 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0 & 0.1 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.3 & 0 & 0.1 \\ 0.4 & 0.1 & 0.2 & 0.2 & 0.1 & 0 \end{pmatrix} \end{array}$$

This represented in code becomes:

```
const markovObject = {
                        1:    [0   , 0.2, 0.5, 0.1, 0.1, 0.1],
                        1.2:  [0.1, 0   , 0.4, 0.1, 0.2, 0.2],
                        1.25: [0.3, 0   , 0.1, 0.4, 0.1, 0.1],
                        1.5:  [0.3, 0.2, 0.2, 0   , 0.1, 0.2],
                        1.6:  [0.2, 0.2, 0.2, 0.3, 0   , 0.1],
                        2:    [0.4, 0.1, 0.2, 0.2, 0.1, 0   ]
                    };
```

Using an object allows for fast lookup, then each index in the object refers to an index of the keys of the object. So picking the next is simple, generate a random number from 0 to 1 and then accumulate probability until we reach it:

```
probList.forEach((e, i) => {
    if (picked >= acc && picked < acc + e) {
        intervalIndex = i;
    }
```

[3] Ultimately this is arbitrary but ends up fitting the definition of an augmented scale, this type of scale was used more extensively in the 20th Century in Jazz, it was also used in Listz's 'Faust's Symphony'. Another way of thinking about it is two major triads above a base pitch (e.g. C E G and E Eb Ab).

```
    acc += e;
});
```

Where `probList` is the array referenced by the key of the current note in the object and `intervalIndex` is the index of the array of keys in the object.

Overall this is a simple method for creating note sequences but is effective for limiting the number of possible transitions with a lot of flexibility.

## 3.5 Chords

For the music to not sound one-dimensional we need to pick more than one note to play in the additive synthesis. This leads to the idea of chords, again we can build these with intervals, this time using a bit more music theory.

I chose that for each vertex that is present in the middle of the screen there should be another note in the chord. The structure of the chord I chose is simple with the notes being the base note, the minor third, the fifth, the minor seventh, the octave, and the whole tone (or 2nd). Represented by the ratios $1, \frac{6}{5}, \frac{3}{2}, \frac{9}{5}, 2, \frac{9}{8}$ respectively. This is purely an artistic choice. For example, if a triangle is in the centre of the screen the first three intervals in the chord will be played $1, \frac{6}{5}, \frac{3}{2}$ resulting in a minor fifth chord. If the user then moves on top of a square the $\frac{9}{5}$ ratio is added to the chord making a minor seventh, similarly, if there's a pentagon it simply adds an octave and for a hexagon, a whole tone which will sound strange but hexagons are rarer to find, making it feel 'special'.

To implement this I simply store the chord in a function expression that returns the frequency to set each voice (oscillator bank) to in an array up to the index `i`. Then another function expression that triggers the release on voice and attack with the new frequency.

```
const getChord = (i) => [
    base*currentNote,
    ...
    base*currentNote*1.125
];

const playVoice = (note, time) => {
    voiceIndex++;
    voiceIndex = voiceIndex % bvoices.length;
    bvoices[voiceIndex].triggerRelease(time);
    mvoices[voiceIndex].triggerRelease(time);
    bvoices[voiceIndex].triggerAttack(note, time);
    mvoices[voiceIndex].triggerAttack(note*interval, time);
};
```

Since the number of possible notes to play at once is 6, the voices array contains six oscillator banks, each with 4 oscillators in them leading to 24 sine waves oscillators at once at maximum. There are two arrays of voices for the base and movable voice so overall there are 48 sine oscillators.

Extensions of this method may lend themselves to being able to control the waveform types and phase of each of the waves as well as what harmonics are present in each voice. Also the ability to change the intervals in a chord. These seem less important for an application such as this primarily static artistic use.

# Chapter 4

# Recall

## 4.1 Saving the State

To be able to recall past states a system needs to be in place that saves parameters periodically. Every 100 frames a new 'snapshot' of the state is generated; this is an element in a JSON tree. Given the program is running at around 60Hz (it is, however, not fixed) this means that approximately every 1.6 seconds the state is saved.

However, if we left the program running memory use would grow, so when saving the state the program should check to see if it's changed or not. This has the effect of compressing time neatly because you can interact with the program, then step away and come back, begin interaction again and still be able to use the recall as if there was no large gap between times.

The naive implementation of this is to create a JSON object that looks something like this:

```
{
  "seed": 230128038,
  "elements": [
    {
      "x": 28,
      "y": 0,
      "z": 0,
      "noiseLevel": 0.4,
      "gridSpacing": 0.3
    },
    {
      "x": 31,
      "y": 10,
      "z": 0,
      "noiseLevel": 0.4,
      "gridSpacing": 0.3
    }
  ]
}
```

But, this approach of only using an array doesn't work, as when you 'rewind' you're branching from the progression that is ahead in the array. If you only insert at the end of the array then this history gets mixed up leading to a mess of non-chronological states.

So to fix this, a method using a tree should be devised. Each time you 'rewind' the tree should create a new branch from which you can begin control and generate state elements in that branch. To do this I used the library `Data-Tree`. This helps manage the tree structure by using references and can export and import the structure to and from JSON. The library also provides methods to search and traverse the tree, operating on each node.

Exporting the tree to a file is then simple, create a method that exports the JSON tree and downloads it to the computer, the filename can be used to store the seed. This works by creating a blob (binary large object), URL to the blob and then a link element that downloads the JSON file, then using JavaScript to click the link and finally removing the link.

## 4.2 Drawing and Interaction

Drawing the tree to the screen requires traversal. If we use depth-first traversal we can work out where a new branch is formed by checking if the depth of the node is the previous node's depth plus one. Then we can simply move down and draw a point. If not, we branch, however it becomes more complicated; at every node, we store the $(x, y)$ values into a JSON object with the key being the node's unique key. Then when we see that the tree has branched we can find the parent node (`Data-Tree` stores this) and set the $y$ value to this and the $x$ values increase by two times the diameter of the point. We then also need to draw a line from the new branch's first node to the point where it branches, this can be done by simply recalling the $x$ value of the parent node. Finally, the most recent node is coloured red. In the end, it looks like this:

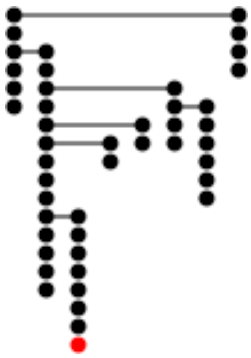


Figure 4.1

To interact with the tree, each point is modelled as a button. These buttons have a `display()` method that checks if the mouse is hovering and then draws a point. The point becomes twice the size if hovered and uses the key of the node to check whether to make itself red or not. The button can be used to check when the mouse is clicked if the button is hovered at the time of clicking and then the state can be loaded.

```
tree.traverser().traverseDFS(function(node) {
    if (prevDepth + 1 === node['_depth']) {
        b += treeButtonDiameter+1;
        values[node.data().key] = {'a':a, 'b':b};
```

```
        prevDepth = node['_depth'];
    } else {
        // get the parent node only when we branch
        parentNode = node.parentNode();
        b = values[parentNode.data().key].b;
        a += treeButtonDiameter*2;

        prevDepth = node['_depth'];

        // draw a line from the node to it's parent's
        push();
        strokeWeight(1);
        values[node.data().key] = {'a':a, 'b':b};
        line(values[parentNode.data().key].a + (treeButtonDiameter/2), b, a,b);
        pop();
    }

    let currentButton = new TreeButton(a, b, node.data().key,
                                               treeButtonDiameter);
    currentButton.display(logger.getIndex());
    buttons.push(currentButton);
});
```

This method was inspired by various interfaces for git that show a tree view of commits / forks, however in this case there is no merge. `gittk` is an example of this but many other similar visualisations exist.

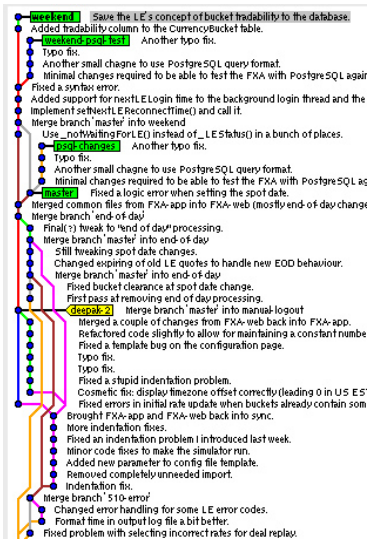



Figure 4.2: gittk tree view

## 4.3 Recalling Parameters

Recalling parameters is simple, when we know the key of the node to recall (from the button that was clicked on) we can search the tree and then return the values stored at that node. The program then sets each of the variables in the main script to these values.

Importing a tree from JSON is similar, except we use `Data-Tree` to import the file into its data structure first and then load the state at the last node that was imported. We also call `noiseSeed(filename)` (with the `.json` stripped) to set the seed correctly.

## 4.4 Limitations

Since the state is only saved every 1.6 seconds but we want to have a 'fluid' method of playback, we can use a linear interpolation between two values in the tree. This can be found by letting the user choose a non-integer index and using `p5js`'s `lerp` function. Pseudo-code:

```
let ceil = Math.ceil(index);
let floor = ceil - 1;
let ceilElement = tree[ceil];
let floorElement = ceil.parent();

let t = index - floor;
var state = {}
//create elements in an interstitial state using lerp
for (var key in ceilParams) {
    state[key] = lerp(ceilElement.params[key], floorElement.params[key], t);
}
```

This assumes that the user is moving linearly between states which may be the case if they're just holding buttons down. Other methods of interpolation could be considered, but for the increased computing cost of a polynomial interpolation or spline interpolation and that users are unlikely to be moving in a way that either of these methods could fit either. For example, if the user decides to move back and forth between two steps there is no way to recover that data. Ultimately the user's inputs will never be able to be modelled by interpolation; this method however saves having to keep the state every frame and is 'good enough' to recall a previous state as likely the user cannot remember the exact configuration anyway.

The problem in integrating this into the program however is the user interface, with the current model in which each history element is represented as a button, there is no way to have the user click on a partial value. This is included in the report because it is a technique that could be used in the future with a different interface design.

# Chapter 5

# Evaluation

Evaluating an artistic project is difficult as there is a need to separate aesthetic goals from more technical ones. Testing against the objectives I set in the introduction may be potentially *subjective* due to the nature of the project. For example, how can we test if the program "Allows the user to experience the works of Viner", here some comparison of the methods used in this project, the possibilities of the images they generate, and the existing work of Viner needs to be made. User testing is possible to compare against the user interaction elements, but would not tell the whole story of the more artistic style.

I have therefore tried to talk about some of the more aesthetic goals and evaluate them as technically as possible concerning the original objectives.

## 5.1 User Testing

The current situation with COVID-19 and meeting people in person made in-person testing impossible. In my introduction I outlined a use case of an online exhibition where this work may be displayed; I decided to try and find users online via social media. This fits the 'spirit' of an online exhibition well and allows the user to explore the program without my influence and without them asking me questions which should hopefully help reflect on if the program is intuitive to use, and makes sense generally. Further, my social media 'influence' contains a lot of people who are generally interested in computers, art, and music; so I feel this is as best of a sample group as possible.

I prepared a questionnaire for completion after the user had been instructed to play with the program for 5-10 minutes. This included questions on whether it felt like the user was exploring a landscape, if the audio felt connected to the graphics, how intuitive the program was to control, how it felt to use the history function, and two optional free text fields about how it felt to control the program and a generic 'other' field for capturing any other feelings the user might have had. I kept the questionnaire short to encourage a greater number of responses. The questionnaire is attached in Appendix B. In the end, I collected 20 user's responses most of which left text in the free text fields.

## 5.2 Evaluation

### 5.2.1 Overall Experience

Users tended to share a sentiment that the program was slightly confusing at first but with exploration of the features it began to make sense. One user said: “I found it unintuitive at first but I think that was made it engaging at first. If I felt like I knew how it worked straight away I wouldn’t have played with it for as long. I felt like it was a puzzle.”

This is good for a program made to engage people, the fact that users were puzzled but then with some user it began to make sense shows that there is some learning moment going on; which may cause a feeling of ‘mastery’ over the program and overall a greater sense of accomplishment than if a full tutorial was given. This technique is well documented in video games, the most famous example being World 1-1 of Super Mario Bros for the Nintendo Entertainment System where concepts are gently introduced without instruction to make the player feel like they are in control [5].

Another user said that they “left this open for like 45 mins zoning out to the noise” which I think is another sign of overall success and engagement.

### 5.2.2 Reproduction of Viner’s Style

The program cannot reproduce every image that Viner’s pen plotter pieces span. Ultimately this is due to fundamental design decisions in how the program operates. For example, some of Viner’s work includes non-closed polygons the method outlined in section 2.2 cannot produce polygons that are not closed.

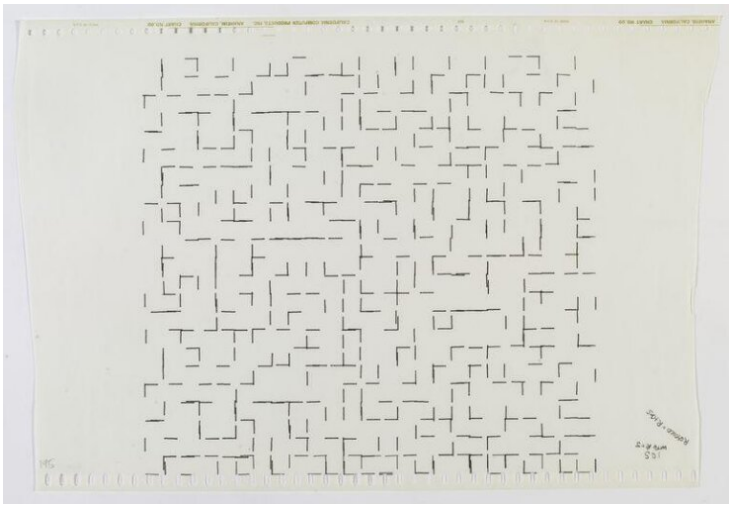
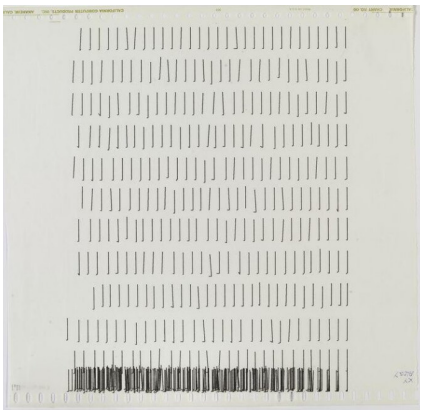


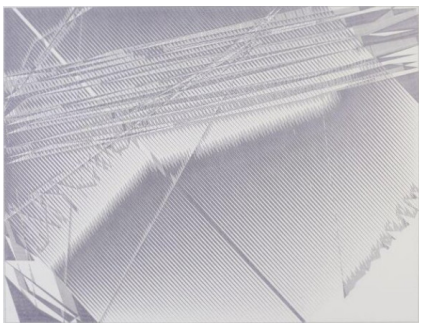
Figure 5.1: An example of an image that the program cannot reproduce

Some of Viner’s work contains artefacts from the pen plotter method, which is also not represented in the final program. This is an example of the medium and the process he worked

with, even if it was not ideal. Perhaps a program that truly sought to reproduce the totality of his work should reproduce this as well. Later plotter art by Viner also took on a radically different style, which isn't explored in this project either.



(a) 'Bunching' at the bottom of the page, likely due to coordinates greater than that of the plotter's maximum range



(b) An example of Viner's later pen plotter work

Figure 5.2

This is not necessarily a failure outright but does bear mentioning that the program only explores a subset of Viner's work; which still provides the user with the sense of exploring a possible 'landscape' of which these images are part of.

Fixing this would require some fundamental reworking to the way the graphics are generated, perhaps creating the grid as a set of points and finding a way to work out if the points should be drawn or lines between points should be drawn.

### 5.2.3 Methods Used

I believe that the methods outlined in section 2.4 and chapter 3 (specifically: section 3.1 and section 3.4) were successful in the sense that they are tools that could be reused in any project.

The methods for auditory location outlined in section 3.1 are of interest to me to develop further as I believe they could be expanded to work well with existing data visualisation programs and perhaps be used as a novel way to explore multi-variate datasets as an extra part of, for example, the grand tour method as in [2]; a chord could play with the intervals marking the subsets you are viewing and move between chords as the image on the screen changes. More exploration could also be completed with rhythm which is a good use of temporal data, perhaps using 'beats' as marks on a 'ruler' in space you are travelling through in time, with acceleration these beats get closer together.

The noise 'approximate rounding' method in section 2.4 is an example of a technique that can be fully integrated into another project at will, this was a successful example of creating a technique that is applies to other projects that utilise point-wise noise.

### 5.2.4 User Interface

Some users in testing pointed out that the help menu would be cut off on smaller screens (e.g. older laptops) so I implemented some responsive CSS using `@media` queries to allow for it to be used on smaller screens. I did this during user testing as I believe it was important to the user's experience of other elements of the program for evaluation. Similarly, an issue with some elements (the volume slider and mute button) not repositioning themselves if the screen resized. I also fixed this during user testing.

One user commented that they used an AZERTY keyboard which this program does not support so they had to switch to QWERTY. Whilst not a problem for a physical exhibition allowing for international or alternative keyboard layouts should be considered before making this part of an online exhibition.

### 5.2.5 Navigation

80% of users felt that they were in a 'landscape'. A few interesting comments about how it felt to control the program were:

> "Once I decided to ignore the descriptions of the controls, and sought to come up with my own names for them, it began to be intuitive. The discovery of the landscape-nature was quite interesting, and the experience was overall enjoyable."

> "It was simple and rewarding to explore the landscape."

> "The layout of the keys was a little confusing to me, and it wasn't clear why some squares moved but not others – even when I turned the randomness down, they seemed to still move randomly but more slowly."

Some users agreed that the image changed too slowly, perhaps the idea of acceleration as you hold down a button could be introduced to allow people to move around faster.

### 5.2.6 Audio

From an artistic perspective, I think that the musical aspect was a success, this is hard to evaluate as it is disconnected from even Viner's work. Ideally, I would have included some more depth. I had looked into using various field recordings from around Leeds I have made as backing for the music but the technical aspects of this made it impossible to implement without using a web server meaning the whole application was less portable.

Figure 5.3: Field recording setup

The majority of users thought that the audio was connected to the graphics but 35% did not. Perhaps a more explicit explanation or some change to make it more immediately obvious to the user could help. Recalling section 3.2 one user said that the music reminded them of "90s Brian Eno quite a bit".

A few users commented that the audio did not sound at all, it seemed to be mostly Google Chrome users. This would require extra testing before deployment to a remote gallery but, again, might not be a problem on a single installation where only one computer is used. I believe this is an issue about how auto-playing sound is allowed to work in the browser (it needs to be triggered from a user-controlled context first), so I added an extra instruction to press the space bar to play sound (which was mapped to toggle Tone.JS's transport), but I could not reproduce this issue and therefore cannot test that it works.

### 5.2.7 History

Users generally understood the history function, only 10% said that it did not function as expected. For some users the tree ended up going off screen; to fix this more work would need to be done to create a method to scroll up and down it.

One user suggested that there should be the ability "[...] to decide when a node on the tree was created, rather than have this happen automatically." Which is an interesting idea, perhaps two modes could be provided 'automatic' and 'manual' with a button to add a new node at will, or maybe a different colour be used for manually saved states. This would allow users to fully save and recall states that they wish to revisit.

Some users commented that they found the upload and download confusing and that perhaps I should limit the possible uploaded files to only the 'application/json' MIME type to not let the user think any arbitrary file is uploaded. Also, being in a web browser the user might think that the uploaded files are stored on an external server which is not the case; The file never leaves their computer. Some method to let the users know this is the case would be helpful

# Chapter 6

# Conclusion

## 6.1 Skill Development

This project helped me develop skills relating to graphics and coordinate geometry a lot more, before partaking in the project I was not confident in my skills doing any programming that drew to the screen. I also gained a good understanding of novel methods for visualising multi-variate data and a better understanding of the field of data visualisation which I think is important to explore further especially with uses in AI and Machine Learning where multi-variate data is common.

## 6.2 Personal Reflection

The nature of this project was such that at times it was hard to find a goal to aim towards, the idea that an art project could be fit into the shape required to complete this report was tough to grapple with. Overall my time working on this however has been enjoyable and it has been pleasing to create a piece I am happy with and that, in user testing, users have seemed to enjoy and engage with; this is perhaps the artist in me speaking.

I'd like to imagine that one day I'll be able to work on similar projects to this more and that this project has helped create a base for any number of web toy-esque computer art creations I'd like to complete.

Finally, with the pressure of the COVID-19 pandemic, it has been hard to judge my progress with peers and the like as I haven't seen any classmates in uni for over a year now. Time management has also generally been more of an issue with an open timetable but my plan was still roughly stuck to.

## 6.3 Future Development

One concept for an extension of this I enjoy but wasn't possible to complete due to budgetary constraints and access to materials would be to control the program using a dance pad for rhythm games. Perhaps at an event like Leeds Light Night, a large projection of the image could be made and people invited to step onto a dance pad to control the artwork.

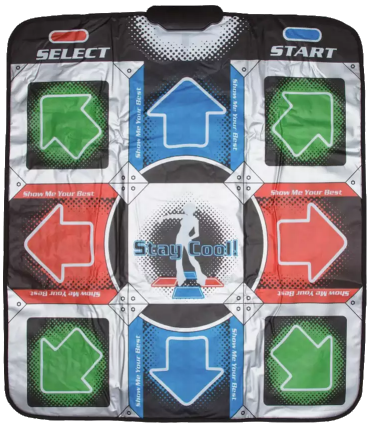



Figure 6.1: An eight-direction dance pad would map to most of the controls in this program

Other options I considered were having 3D models represent the image. Similar to the giant's causeway, these would be extrusions of polygons through space. Also, a full map overview where the user could choose two points on the history tree and view, in 2D or 3D a 'map' of what they saw in that time frame.

Outside of the space of the art itself, I would like to develop the technique in section 3.1 further to see if it would work for other users and perhaps extend it to encode more information, this would require more in-depth study.

## 6.4 Ethical and Legal Considerations

This project does not handle any personal data, nor deals with the identification of people from data. No subject matter may offend or otherwise affect groups of people.

However, there is a question over ownership and who owns images produced by this program if they are indistinguishable from Viner's work. One point to consider is that where his work is pen-plotter images on listing paper, this is purely computer-generated imagery.

Of course, the provenance associated with the work is also different; in the case of Prado's Mona Lisa, for example, it's clear that despite being made around the same time, in the same workshop, and of the same subject it is a separate piece of work with a different story. This perhaps is not the best example however the creator of the Prado version likely worked in da Vinci's studio at the time [14].

It's also worth mentioning the fair dealing exception to UK copyright law which says that you are allowed to "copy limited extracts of work for non-commercial research or private study"[12]. However, I am unsure if this legal protection extends to *the potential for generating* copies of work. It is clear to me however that I am not attempting to reproduce directly and instead explore the techniques of creation of this work using different technologies.

# References


[1] E. Abbott. *Flatland: A Romance of Many Dimensions*. Roberts Brothers, 1885.

[2] D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985.

[3] P. Comerford. Simulating an organ with additive synthesis. *Computer Music Journal*, 17(2):55–65, 1993.

[4] D. T. L. Deborah F. Swayne, Andreas Buja. Exploratory visual analysis of graphs in GGobi. In *Exploratory Visual Analysis of Graphs in GGobi*, 2003.

[5] Eurogamer. Miyamoto on world 1-1: How nintendo made mario's most iconic level. Youtube interview with Miyamoto (designer of SMB). Retrieved: 2021-04-22 `https://youtu.be/zRGRJRUWafY`.

[6] P. Kiefer and M. Riehl. Markov chains of chord progressions. *Ball State Undergraduate Mathematics Exchange*, 10:16–21, 2016.

[7] R. Lycett. Darrell viner: Materiality, process and the coded object, Jan. 2016. Abstract Retrieved 05/04/2021: `https://eprints.hud.ac.uk/id/eprint/28205/`.

[8] J. Monzo. Monzo / exponent vector / prime-exponent vector, 2005. `http://tonalsoft.com/monzo/lattices/lattices.htm`.

[9] H. Partch. *Genesis Of A Music*. Da Capo Press, 1974.

[10] K. Perlin. Noise and turbulence, 1998. Retrieved 10/03/2021: `https://mrl.cs.nyu.edu/~perlin/doc/oscar.html`.

[11] K. Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. Association for Computing Machinery.

[12] gov.uk. Exceptions to copyright. `https://www.gov.uk/guidance/exceptions-to-copyright\#overview`.

[13] Henry Moore Institute. Darrell viner: Early work, 2011. `https://www.henry-moore.org/whats-on/2011/07/27/darrell-viner-early-work`.

[14] Museo Nacional del Prado. The Mona Lisa - The Collection - Museo Nacional del Prado. `https://www.museodelprado.es/en/the-collection/art-work/the-mona-lisa/80c9b279-5c80-4d29-b72d-b19cdca6601c`.

[15] J. O. Smith, III. Viewpoints on the history of digital synthesis, Dec. 2005. Keynote paper from proceedings of international computer music conference pp. 1-10, Oct. 1991. Retrieved 2021/04/23: `https://ccrma.stanford.edu/~jos/kna/kna.pdf`.

[16] T. Taylor. *Iamblichus' Life of Pythagoras, or Pythagoric Life.* J.M. Watkins, 1818.

[17] J. A. Vince. *Picaso A Computer Language For Art And Design.* PhD thesis, Brunel University, 1975. Available: `https://eprints.mdx.ac.uk/10169/`.

[18] D. Viner. Artist statment. Retrieved 14/12/2017: `http://www.darrellviner.org/artiststatement/`.

# Appendix A

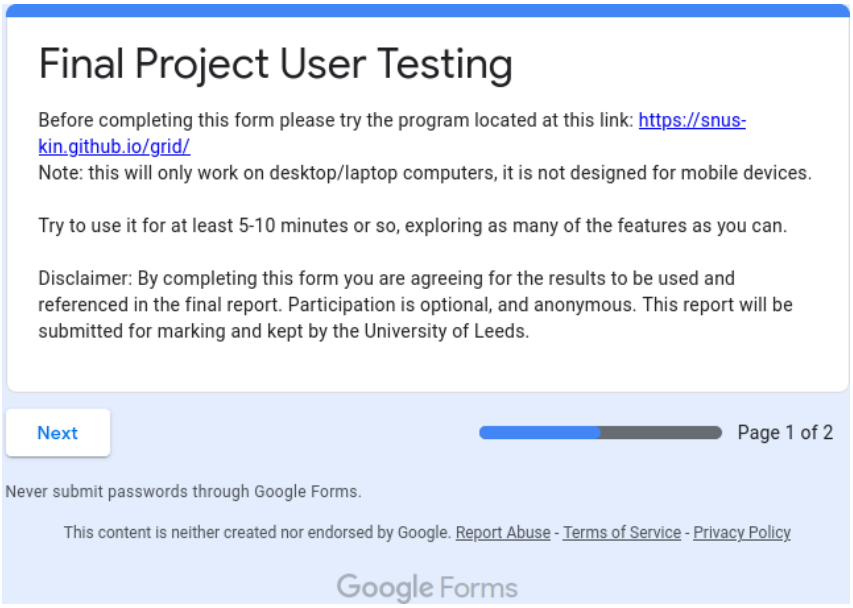
## External Materials

- Libraries
  - `p5.js` javascript library: `https://p5js.org/`
  - `tone.js` javascript library: `https://tonejs.github.io/`
  - `Data-Tree` javascript library: `http://cchandurkar.github.io/Data-Tree/`
- Other External Materials
  - Some of the initial graphics code was based on notes by my supervisor, John Stell
  - The code for the additive synth was based on and heavily modified from `https://github.com/ejarzo/additive-synth`

# Appendix B

# User Testing Materials

## B.1 Questionnaire



Final Project User Testing

Before completing this form please try the program located at this link: https://snus-kin.github.io/grid/
Note: this will only work on desktop/laptop computers, it is not designed for mobile devices.

Try to use it for at least 5-10 minutes or so, exploring as many of the features as you can.

Disclaimer: By completing this form you are agreeing for the results to be used and referenced in the final report. Participation is optional, and anonymous. This report will be submitted for marking and kept by the University of Leeds.

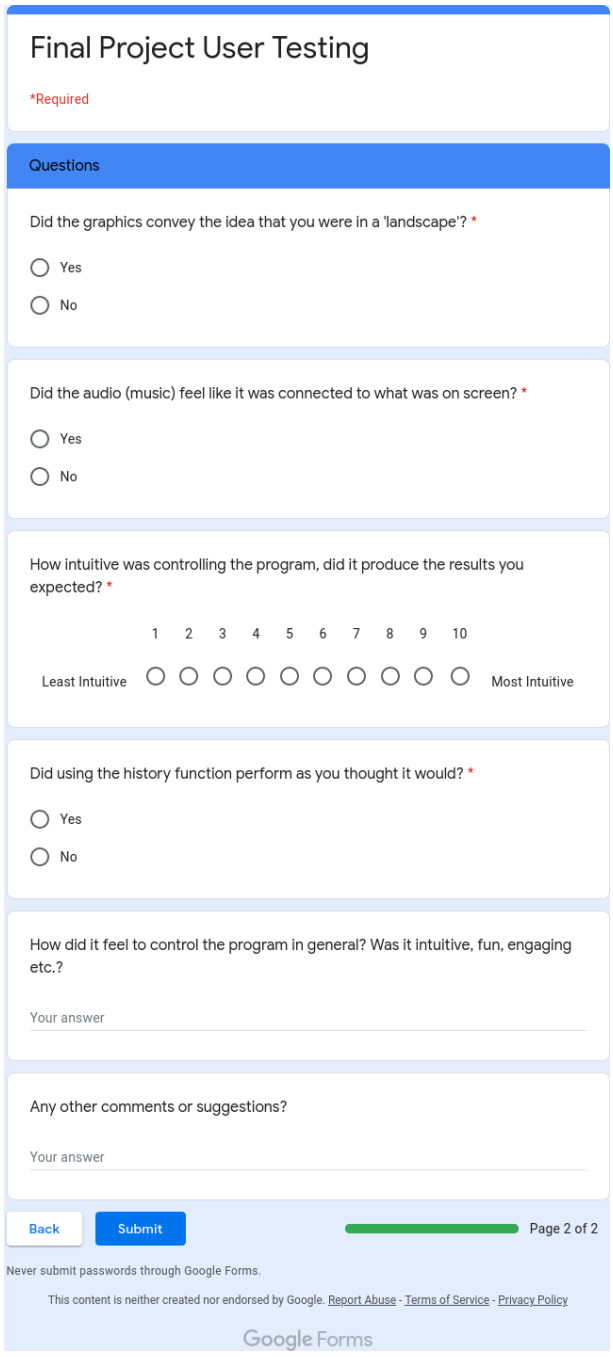
Next

Page 1 of 2

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms



Final Project User Testing

*Required

Questions

Did the graphics convey the idea that you were in a 'landscape'? *

- Yes
- No

Did the audio (music) feel like it was connected to what was on screen? *

- Yes
- No

How intuitive was controlling the program, did it produce the results you expected? *

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Least Intuitive | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | Most Intuitive |

Did using the history function perform as you thought it would? *

- Yes
- No

How did it feel to control the program in general? Was it intuitive, fun, engaging etc.?

Your answer

Any other comments or suggestions?

Your answer

Back Submit Page 2 of 2

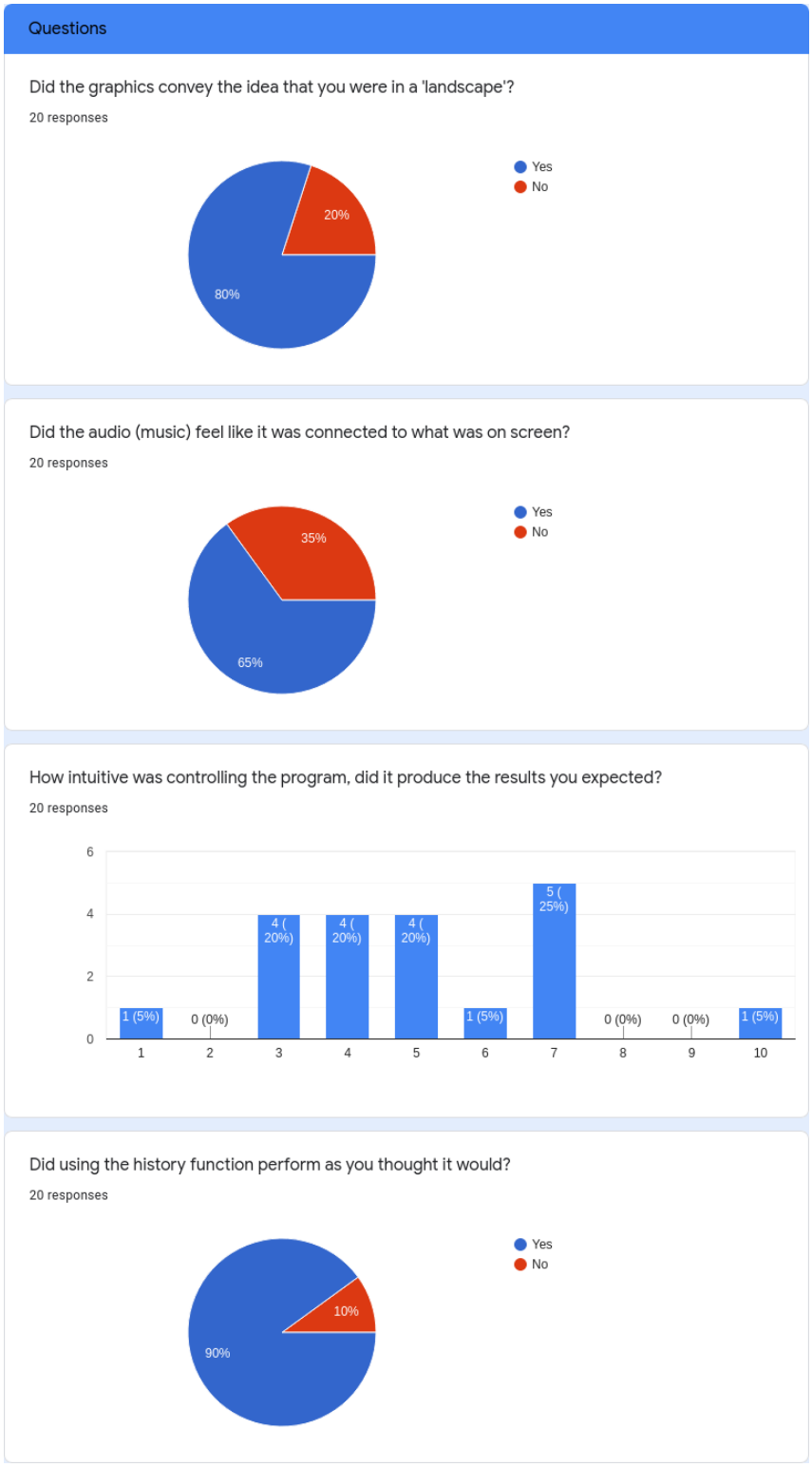
Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

## B.2 Results

### B.2.1 Restricted Response



### B.2.2 How did it feel to control the program in general? Was it intuitive, fun, engaging etc.?

- Nice watching everything transform, I think it's good that you added the option to zoom in and out as it creates really interesting effects in terms of how the shapes overlap and move with each other in their wave-like clusters. The music is great, very immersive, makes me think of 90s Brian Eno quite a bit.
- All in all, an enjoyably evocative and surreal experience. The music was very pleasingly eerie
- I found it unintuitive at first but I think that was made it engaging at first. If I felt like I knew how it worked straight away I wouldn't have played with it for as long. I felt like it was a puzzle.
- was a bit confusing at first, but I got a hang of it after a minute or so
- A little difficult to use the arrow keys, also the low sensitivity of the keys made it seem a little laggy in response. The sound response was exciting but it wasn't completely clear how movement was influencing it.
- It was really neat, could definitely see myself trekking across a landscape as the polygons warped and shifted. A really hypnotizing experience. Love the musical choices as well.
- it was not initially clear what anything did, but after repeatedly consulting the controls I could explore a bit
- It was fascinating, a very interesting interactive project. It was simple and rewarding to explore the landscape.
- needs more labels on the buttons and more dramatic / quick change upon doing things. keyboard navigation was glacial
- It was pretty fun once i got it going but it took some time for the controls to have any effect - very slow movement at first
- Once I decided to ignore the descriptions of the controls, and sought to come up with my own names for them, it began to be intuitive. The discovery of the landscape-nature was quite interesting, and the experience was overall enjoyable.
- It would be easier if the controls were always visible. I didn't understand how to use it until I opened the help menu, and I wasn't able to clearly remember the options. I can get a sense of the controls after a while, but it's hard to notice immediate effects after the initial state.
- fun
- Controls were smooth. Maybe make spacing up/down a bit faster?
- I felt like the changes were too slow to notice them. It was not intuitive but it was fun

- The layout of the keys was a little confusing to me, and it wasn't clear why some squares moved but not others – even when I turned the randomness down, they seemed to still move randomly but more slowly. I liked the tree branching, and the overall effect is very pretty and calming.
- Once the controls were down, it was fun to play around with.
- it was cool once i got it into a state where WASDing around really changed the landscape yea actually i just left this open for like 45 mins zoning out to the noise

### B.2.3 Any other comments or suggestions?

- I found myself wanting to decide when a node on the tree was created, rather than have this happen automatically. I didn't use the upload/download features.
- Using the mouse for some input may feel more natural, it is hard to remember what keys do what, etc. The sound also stopped at some point.
- i couldnt read the help text, since it overflowed off the bottom of the screen (1366x768). personally i would rather the help button was a toggle rather than a hover at first glance i didnt know what the two arrow buttons do. best to say what they are, since when i clicked the down button i was interrupted by a download dialogue the UI doesnt scale properly with zoom, since they are positioned absolutely at load rather than using position: relative and letting the browser place them doesnt work great with browser vim plugins. but that's my fault ;)
- 1. Should probably by default limit what backup uploaded by users, it asks for all files when it just needs json. 2. History can extend past edge of screen and cannot be obtained once it does. Should make it scrollable if possible. 3. The instructions are fantastic, love the aesthetic and everything is clear and concise. However intuitively it would be nice to just click the help button to show and hide the instructions instead of moving the mouse over to the edge past the render. Otherwise, a really great art piece, loved playing around with it.
- I felt like the encoding of the landscape as shapes that slowly move to reflect some underlying state made it difficult to know what sort of movement was occurring. Before consulting the help menu, and even for a bit after, I interpreted the controls as manipulating the shapes rather than as moving around a landscape viewed through the shapes.
- Having an AZERTY keyboard, I had to change the keyboard to QWERTY to use the program. Not very important but something to consider for online works.
- didn't actually find the history function but the question was mandatory
- Sound didn't work for me :(

- The "history" tree on the left goes off-screen if one carries on too long, and becomes inaccessible, unfortunately. Perhaps a scroll function? (On Chrome, if that helps.)
- The music only seems to play in Firefox.
- lol funny volume button
- I couldn't actually hear any music, so I don't know what it was supposed to sound like. I had my volume up and messed with the slider and mute button, but nothing happened for me.
- lacking some "affordances" (ux term), i get you're going for artsy look font, favicon

# Appendix C

## Audio Demo

```
import processing.sound.*;

SinOsc base;
SinOsc variable;
float baseFreq = 440;
// they are in unity to start with
float variableFreq = 440;

float volume = 0.2;
float s_x,s_y,x,y = 0;
float interval;

boolean isUp, isDown, isLeft, isRight;

void settings() {
  size(501, 501);
}

void setup() {
  base = new SinOsc(this);
  base.play();
  base.freq(baseFreq);
  base.amp(volume);

  variable = new SinOsc(this);
  variable.play();
  variable.amp(volume);
}

void draw() {
  background(255);
  translate(width/2, height/2);
  scale(1, -1);

  // make some axes
  stroke(0);
  strokeWeight(1);
```

```
  line(0,height/2,0,-height/2);
  line(width/2,0,-height,0);

  strokeWeight(4);
  point(x,y);

  // mark where the integer values will be, for demo reasons
  stroke(255,0,0);
  strokeWeight(4);
  for(int i = -3; i <= 3; i++) {
    for(int j = -3; j <= 3; j++) {
      point(map(i, -3, 3, -width/2, width/2),
            map(j, -3, 3, -height/2, height/2));
    }
  }

  // scale x and y to some reasonable numbers -3 -> 3
  s_x = map(x, -width/2, width/2, -3, 3);
  s_y = map(y, -height/2, height/2, -3, 3);
  // now we need to find a way to scale until
  // we're in the same octave
  interval = pow(3,s_x) * pow(5,s_y);

  while (interval < 1 || interval >= 2) {
    if (interval < 1) {
      interval = interval * 2;
    } else if (interval > 2) {
      interval = interval / 2;
    }
  }

  variableFreq = interval * baseFreq;
  variable.freq(variableFreq);

  if (isUp) y++;
  if (isDown) y--;
  if (isRight) x++;
  if (isLeft) x--;
}

void keyPressed() {
  setMove(key, true);
}
```

```
void keyReleased() {
  setMove(key, false);
}

void setMove(char k, boolean b) {
  if (key == 'w') isUp = b;
  if (key == 's') isDown = b;
  if (key == 'a') isLeft = b;
  if (key == 'd') isRight = b;
}
```