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.
<As an example>

| Items | Format | Recipient(s) and Date |
|---|---|---|
| Deliverable 1, 2, 3 | Report | SSO (DD/MM/YY) |
| Participant consent forms | Signed forms in envelop | SSO (DD/MM/YY) |
| Deliverable 4 | Software codes or URL | Supervisor, Assessor (DD/MM/YY) |
| Deliverable 5 | User manuals | Client, Supervisor (DD/MM/YY) |

Type of project: _____

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) _____

## Summary

<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>

## Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>

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

# 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 [8].

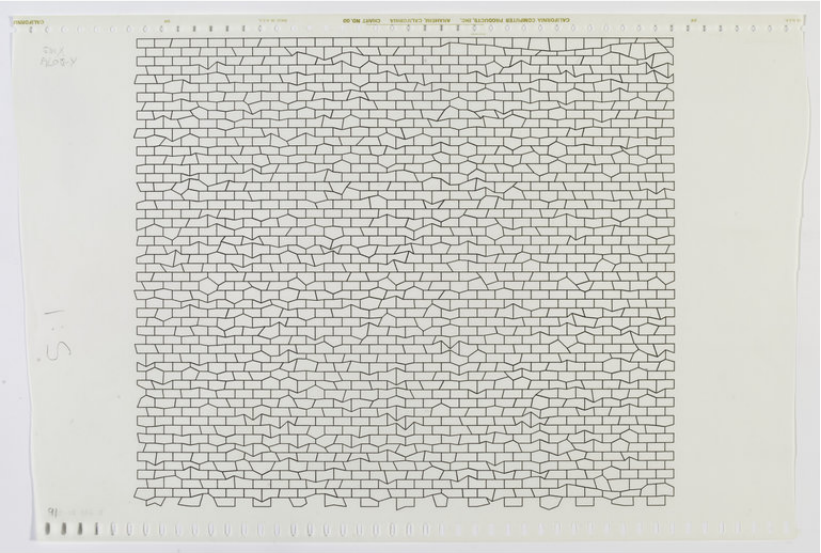


Figure 1.1: *Darrell Viner, Untitled (1974)* ©Victoria and Albert Museum, London.

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

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 various aspects 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." [13]

These parameters may for example control the spacing of the grid, the displacement of each point, the noise present at each vertex and so on. These parameters should be able to be changed to create some sort of 'landscape' or 'topology' that the user can explore through manipulating the program. Later on I will detail methods for users being able to manipulate the parameters and considerations that need to be made with respect to usability.

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 and work to create a dialogue with the original work and contextualise the user's understanding of his work.

## 1.1 Objectives & Deliverables

### 1.1.1 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.
- Develop a catalogue of techniques to be used in generative art and music, and to evaluate each for relevance to the project
- Create an interface which allows users to navigate generated spaces
- Create a system such that users are able to recall a state they were in, and find it again
- As an extension of that a way of seeing all previous states, or a map of what the user had seen in a session

### 1.1.2 Deliverables

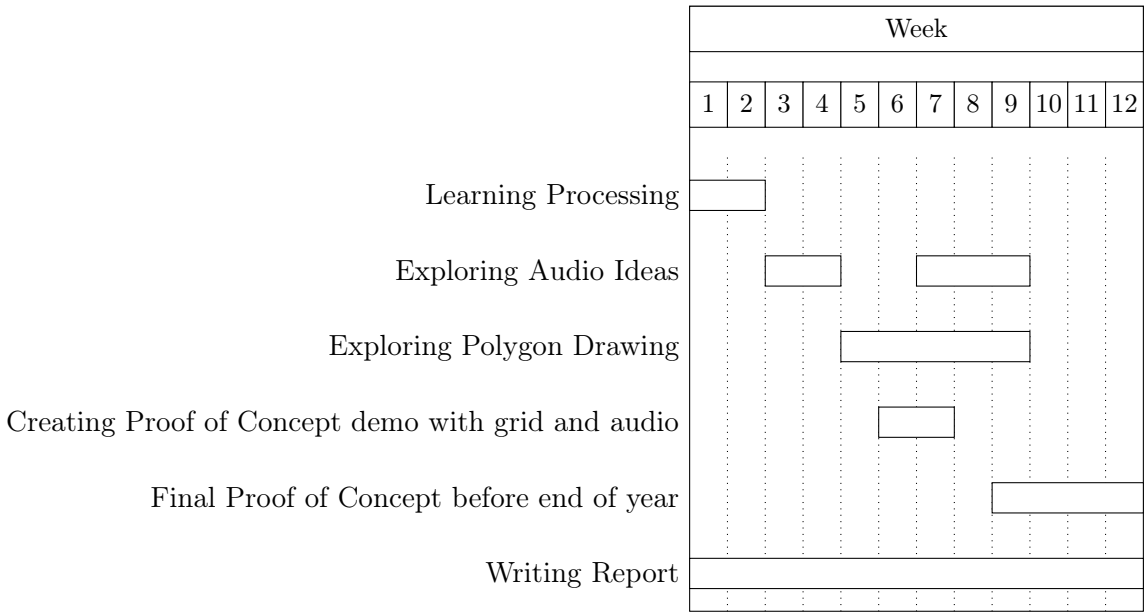
An application written for processing that should allow the user to explore a landscape, generated to feel like Viner's work. This application should be friendly to use and will include both graphics and sound. The feeling of some sort of topography should be conveyed and the visual should look similar 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.

There should be another mode within the program that allows users to explore the previous session's movements and recall what they saw. This should be easy to understand and allow for users to see configurations of parameters they like and save them somehow.

## 1.2 Initial Plan

Being that this project is exploratory in nature, an iterative approach to development makes sense, with small prototypes being created, explored, and built on quickly. As such I have allocated time for ideas to be developed in.

To start I will learn about processing, this library for Java makes it simple to get a framework for graphics, of which the logical can be easily moved to some other framework if need be. Since my tutor's previous work was written in processing, there is already some groundwork complete to build off of. It may be worth exploring other options like `p5.js` for portability and the ability to embed into webpages for the final implementation, `p5.js` offers some more mature sound libraries, as well as other web technologies which may be useful for future explorations such as Geo-location and portability to mobile devices.




## 1.3 Risk Mitigation

Since this project is fairly isolated in nature, 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 very easy to have many 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

There are two problems to consider here, how points on a grid are generated, and how they are displayed. To generate points there needs to be a method of procedural generation, i.e. creating an algorithm that can given a set of parameters tell me where a point should or shouldn't be and what other points it should or shouldn't connect to.

The problem of how they are displayed is another issue, how can we convey the idea of moving through a space when the space is simply a grid?

## 2.1 Anatomy of the 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) [3], this was essentially a line-drawing library, similar to what we are using processing / p5js for. `PICASO`'s use by Viner isn't well documented but the manual [12] has many subroutines for transforming vertices according to some rules. Some notes on Viner's work indicate there was definitely mathematical thinking going on in the development.

## 2.2 Polygons

One aspect the program should be able to do is to grow and shrink polygons between different number of verticies. To achieve this a polygon is inscribed on a circle. The points of this polygon of $n$ verticies 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 $n$.

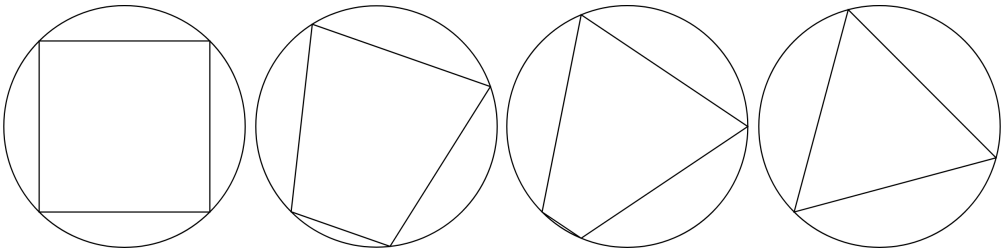


Figure 2.1: Starting at $n = 4$ the shape deforms through $n = 3.5$, $n = 3.2$ and finally $n = 3$

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;
dTheta = TWO_PI/n;

beginShape()
    vertex(x + r*cos(theta), y + r*sin(theta));
    for (i = 1; 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, calculated within the for loop, to transform the shape away from regular polyhedra (when $n$ is an integer).

## 2.3 The Grid

With the motif of the grid being the most obvious visible thing in Viner's art, there needs to be a way to actually draw a grid to the screen; ideally each vertex needs to be able to be separately controlled.

To do this I have adapted John Stell's work on Viner in his workshops [11], using an object-oriented approach; but have created a system where the grid is centred on a given `x,y` coordinate. Essentially there is a screen, and for every vertex at a column and row their relative coordinates need to be calculated. This is simple with the following statement:

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

Where `i,j` is the column and row value of the point, `cols,rows` are the total number of columns and rows, and `gridSize` is the pixel size of the grid (which is more of a guide than anything). The grid should also be centred on `x,y` so calling a translate before drawing any points should be done:

```
translate((width/2)+(0-x), (height/2)+(0-y));
```

All of this leads to a system where the centre point has a given coordinate and we can find that coordinate for all other points around it, this also means we can interpolate any parameters from the program to what should be expected at a given coordinate; or we can also generate a terrain away from the main graphics thread and draw them to the screen at a given coordinate.

### 2.3.1 Demo

Here a grid is prepared with fixed parameters that create distortion 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.

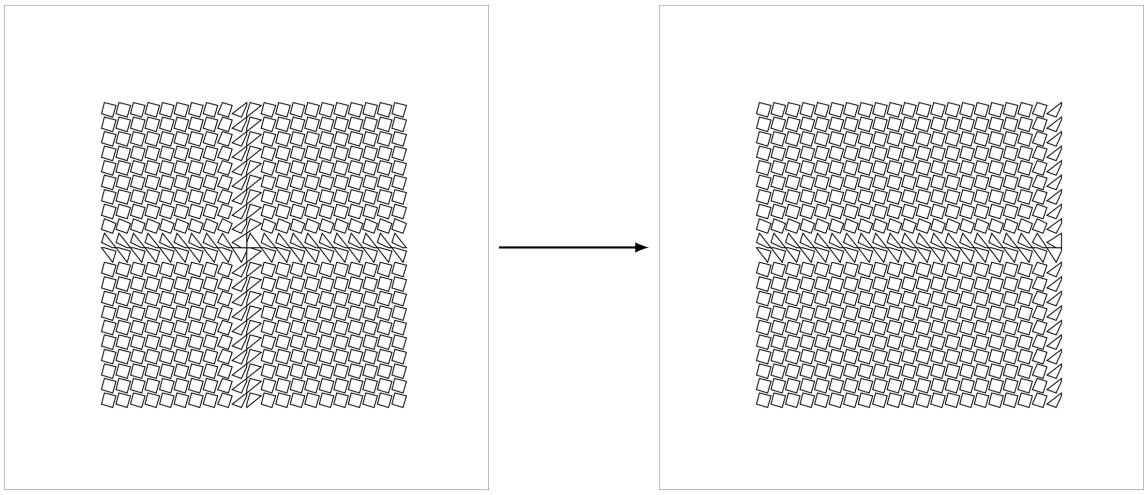


Figure 2.2: The user presses 'd' for a short time, increasing the 'x' parameter

## 2.4 Landscape Generation

The program should generate a 'landscape' or 'topology', this is an analogy for a space that has an continuous change of the parameters across them. 'Landscape' is apt because we're exploring a set of features generated by the program much like if you were to look at a

topographic map. The user should have the feeling of moving through a space like this, more than simply increasing some parameter and immediately seeing a change throughout the grid.

This is the main question of how the program will work on a technical level, how can we create an algorithm or mathematical model that explores the spaces that Viner's work set out to create?

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 considered to be aesthetically useful.

This leads to 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 ability to recall previous sessions, a seed would be given. It seems then that the random choice contains the static choice and should be the one to be carried out.

### 2.4.1 Noise

One possible function to consider here is a computer-graphics oriented noise implementation. A desireable property is that of a fairly smooth gradient between extremes, or defined regions in which values are high.

`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 relatively 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.

`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 $0 - 1$

[1] And won an Oscar for "allow[ing] computer graphics artists to better represent the complexity of natural phenomena" [6]

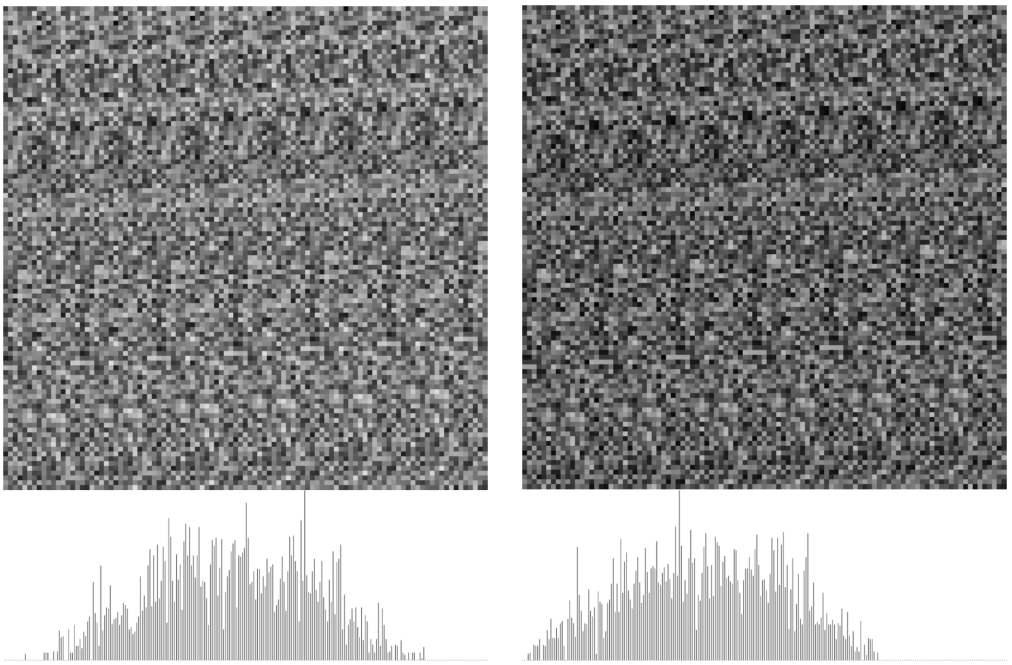


Figure 2.3: Two perlin noise samples (one default the other with noiseDetail(2, 0.5) with the same seed, histograms plotted beneath show that the adjustment makes the image overall darker

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 unimodal the most common values will be those towards the center of the range.

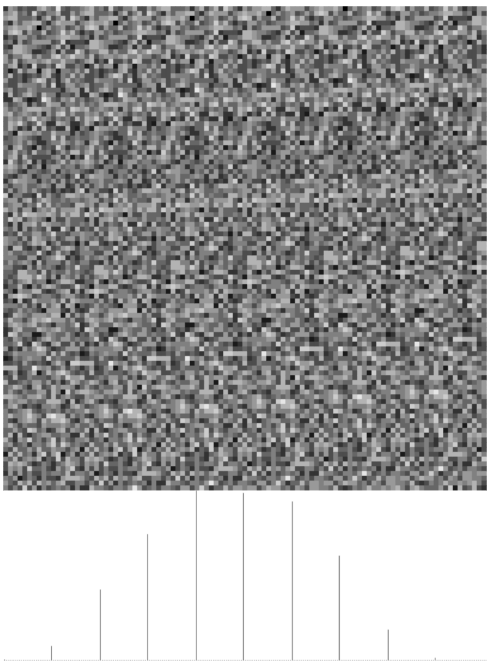


Figure 2.4: This sample contains only ten colours

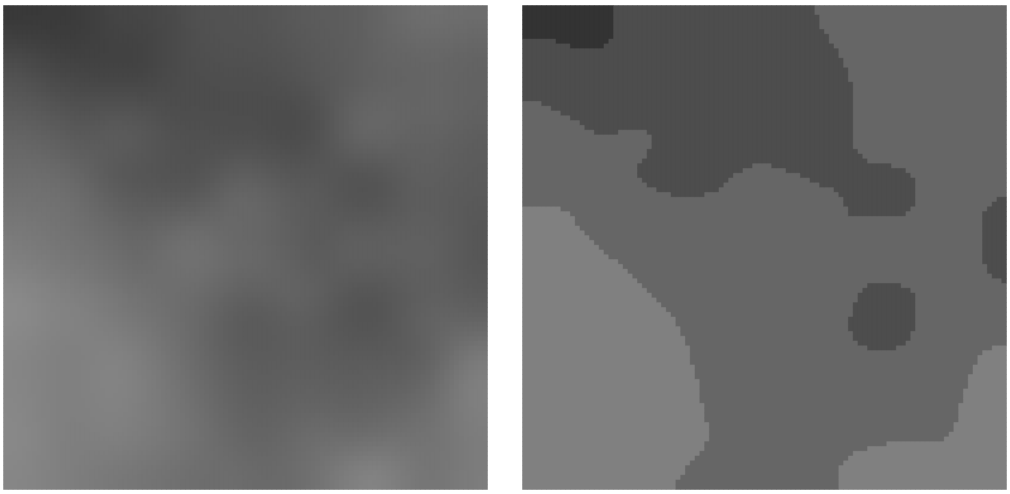


Figure 2.5: Unquantised and quantised noise at 0.001 noiseScale

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 multimodal 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;
}
```

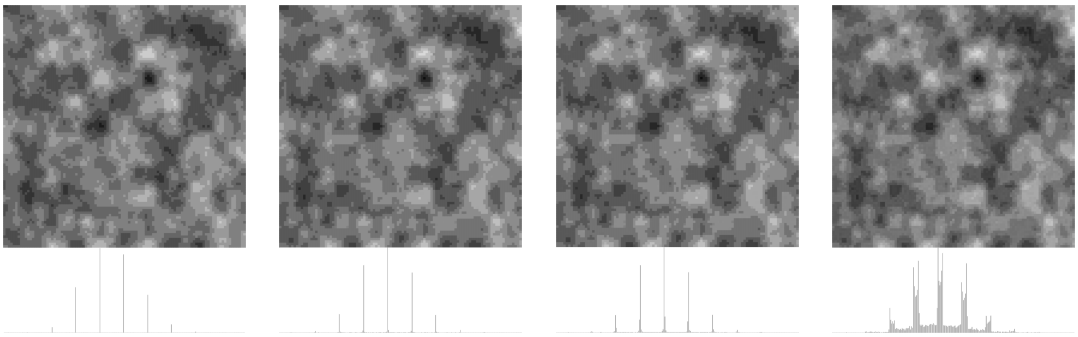


Figure 2.6: Rounded discretely, Terms = 100, Terms = 20, Terms = 1

At such a large scale the difference is hard to percieve but when using the example in Figure 2.5 there's a clear 'smoothing' in the boundaries.

Figure 2.7: Terms = 100, Terms = 10, Terms = 1

Implmenting this such that noise controlls the number $n$ (verticies 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 $n$)

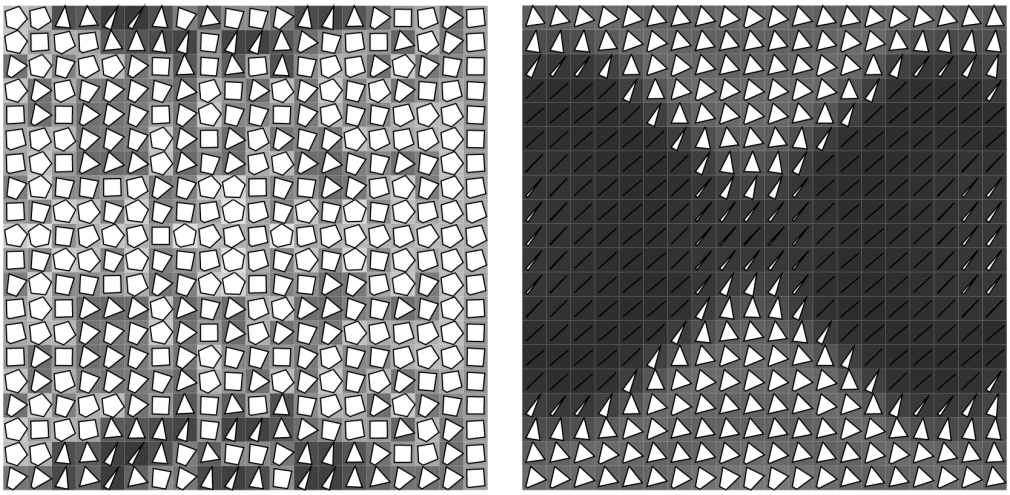


Figure 2.8: Terms = 20, Terms = 1

With user interaction this scheme creates the feeling of moving through regions of these

differernt 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.

### 2.4.2 Dynamical Systems

In a 2D space (x,y) we can find a set of functions that return a dimension higher. These are complex functions. For example the mandlebrot set,

# Chapter 3

# Navigating Higher-dimensional Space

## 3.1 Graphically

Navigating higher-dimensional space is of practical interest because a lot of data is higher-dimensional, for example the field of machine learning for example deals with very high dimensional data. Tools like '`ggobi`' exist to help explore multi-variate data [1] which tend to disregard spacial relations of points in favour of clustering based on shared properties.

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.

Video games also have explored non-euclidean and 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 a 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 [1] 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, 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.

### 3.1.1 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.

[1] Which the developer tells me is based on an astrolabe

### 3.1.2 Controls

The above mentioned video games allow for the user to move through the 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 2.2, 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.

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

### 3.1.3 Visualising Possible Shapes

As in [1] we could possibly plot sub-spaces of a shape to create a visualisation of the 'configuration space'. That is, where there are a potentially infinite (practically until overflow) variation in parameters there is a parameter space. But when we apply functions to limit this space we create a 'configuration space'.

## 3.2 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.

### 3.2.1 Intervals

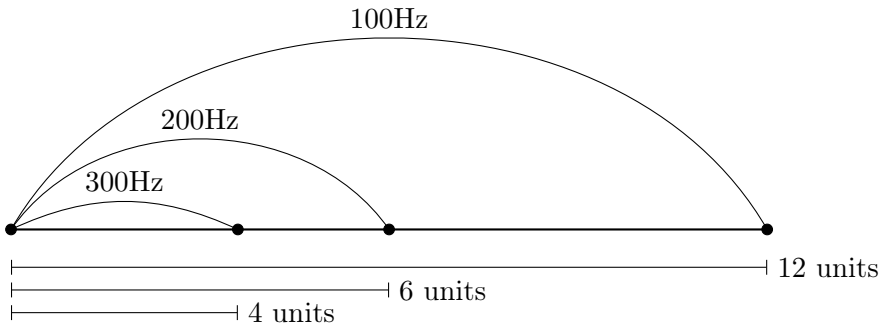



Figure 3.1: Diagram showing intervals being related to the idea of a ratio of lengths for example the 300Hz frequency is related to the 200Hz frequency by $\frac{6}{4} = \frac{3}{2}$

An *interval* is simply a ratio of two frequencies: $\frac{f_1}{f_2}$. This is a process to generate intervals based on a set of spacial coordinates, or simply a set of parameters. Taking inspiration from the *just-intonation* tuning method a ratio comprised of integers only sounds consonant. A *p-limit* tuning system is a just-intonation based system where every interval's highest prime factor is $p$. [5, p.76, 109]

3-limit tuning was the perhaps the first tuning system, being theorised by Pythagoras and can be recreated by the ratios $\frac{3}{2}, \frac{2}{3}, \frac{1}{1}$. These can be represented using an exponent vector, usually called a 'monzo' [4]; 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 $|*\ e_2\rangle$ 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.

### 3.2.2 Navigating Space

#### In 2D

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

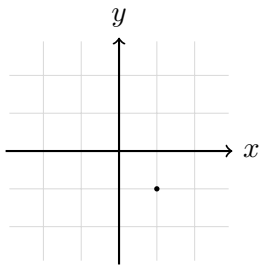



Figure 3.2

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.

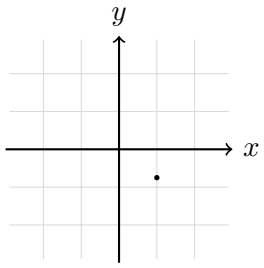



Figure 3.3

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, perhaps the image produced by the program will look disordered.

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.

### 3.2.3 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 less variables will still be possible so a user could still explore one or two parameters at a time and get the same results.

### 3.2.4 Processing Demo

I have created a demo of this concept in 2D in processing, the code is in section A.1. The program simply has a 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:

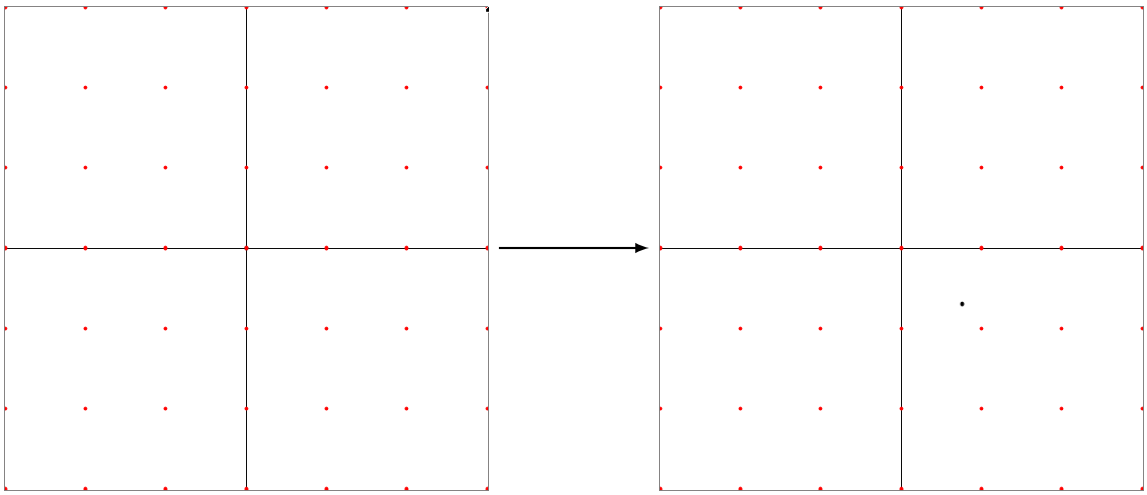


Figure 3.4: At first the black dot is hidden beneath the centre red dot, the user uses WASD to navigate, the audio is out of tune in the second state

This demo is very simple and only uses two sine waves so is very 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, there are points that aren't marked that *do* sound somewhat consonant, so this cannot be the only method for navigation, only a prompt to help the user.

## 3.3 Recall

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 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 in a neat way 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 first implementation of this was to create a json object that looked something like this:

```
{
  "seed": 230128038,
  "uuid": 102830128,
  "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 actually 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.

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.

```
let ceil = Math.ceil(index);
let floor = Math.floor(index);
let ceilParams = this.logObj.elements[ceil];
let floorParams = this.logObj.elements[floor];

let t = index - floor;
```

```
var state = {}
//create elements in an intersititial state using lerp
for (var key in ceilParams) {
    state[key] = lerp(ceilParams[key], floorParams[key], t);
}
```

This, of course, 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 the fact 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.

# Chapter 4

# Music

## 4.1 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 fairly vague as with generative art, the main requirement is that a system is setup 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.

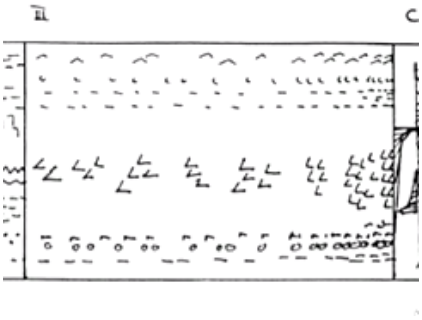
The immediate style of music to draw from Viner's work would probably be that of minimalism, the repetition and difference across an image is very similar in style to especially the percussive works of Reich, Glass, Riley, and even the drone works of La Monte Young to an extent. Another, less famous 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.

In fact, 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')

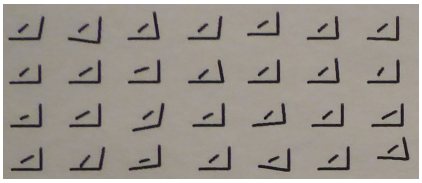
There is also a nice parallel from the graphical scores of Xenakis to Viner's works, at least aesthetically (see Figure 4.1). Xenakis is the so-called 'father of granular synthesis', this technique is made of playing many small, short sounds (grains) to create what is often described as a 'cloud' of sound. Xenakis was also a pioneer of ideas of stochasticism in music and could be said to be one of the first generative composers in that respect.

Overall the idea with the composition should be to enhance what is on the screen, if the image is disordered the sound should be too, if it is ordered and regular the sound should follow.

An idea here is to use a kernel or convolution matrix to be able to take the whole image's current parameters for each element in the grid and map it down to a smaller dataset for use as parameters to the synthesiser, essentially downscaling the image. This could be used powerfully



*Movement 3 of Xenakis' Metastasis*



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

Figure 4.1: Comparison of Xenakis and Viner

with the idea of granular synthesis as, for example each parameter could change something about a grain (playback speed, volume, effect intensity). This feels also almost like the feeling of the Moiré pattern mentioned by Eno.

Further this will be combined with the methods mentioned in section 3.2, including the ideas present in the mentioned minimalist composers.

## 4.2 Synthesis

There are many options for digital synthesis of which I've already mentioned granular synthesis, but frequency modulation, additive, wavetable, modal, and so on are also options.

Largely there are four approaches, that of the Processed Recording, the Spectral Model, the Physical Model, and the Abstract Algorithm [10].

For this project spectral and physical modelling are 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 who's 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 are fairly straightforward when there is already some digital signal processing in place; p5.js has a library called `Tone.JS` which handles a lot of the practical elements of sound generation.

## 4.3 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 very '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 [2] 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.[1] 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.

$$\begin{array}{c} \\ 1 \\ 1.2 \\ 1.25 \\ 1.5 \\ 1.6 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 1.2 & 1.25 & 1.5 & 1.6 & 2 \end{array} \\ \begin{pmatrix} 0 & 0.2 & 0.5 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0 & 0.5 & 0.1 & 0.2 & 0.2 \\ 0.3 & 0 & 0 & 0.6 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0 & 0 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.3 & 0 & 0.1 \\ 0.4 & 0.1 & 0.2 & 0.2 & 0.1 & 0 \end{pmatrix} \end{array}$$

This represented in code becomes:

```
const markovObject = {
                        1:    [0   , 0.2, 0.5, 0.1, 0.1, 0.1],
                        1.2:  [0.1, 0   , 0.5, 0.1, 0.2, 0.2],
                        1.25: [0.3, 0   , 0   , 0.6, 0.1, 0.1],
                        1.5:  [0.3, 0.2, 0.2, 0   , 0   , 0.2],
```

[1] 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).

```
        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;
    }
    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 very simple method for creating note sequences but is quite effective for limiting the number of possible transitions with a lot of flexibility.

# Chapter 5

# Ethics

This project does not handle any personal data, nor deals with identification of people from data. There is no subject matter that may offend or otherwise effect 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 own 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, also the provenance associated with the work is also completely different; in the case of Prado's Mona List 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 [9].

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"[7]. 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 completely different technology.

# References


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

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

[3] R. Lycett. Darrell viner: Materiality, process and the coded object, Jan. 2016. Unpublished.

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

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

[6] K. Perlin. An image synthesizer, July 1998. Retrieved 10/03/2021: `https://mrl.cs.nyu.edu/~perlin/doc/oscar.html`.

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

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

[9] 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`.

[10] J. O. Smith, III. Viewpoints on the history of digital synthesis, Dec. 2005.

[11] J. Stell. Notes, 2019. Unpublished.

[12] J. A. Vince. Picaso a computer language for art and design, 1975.

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

# Appendix A

# Listings

## A.1 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;
}
```