Collegial Computation

Processing Architectural form

Jordan Anderson

Figure 0.01 Photograph of ‘Collegial Computation’ at the NZIA Student Design Awards 2020
David St. George (2020)

Collegial Computation

Processing Architectural form

by Jordan Anderson

A 120-point thesis submitted to the Victoria University of Wellington in partial fulfillment of the requirements for the degree of Master of Architecture (Professional)

Victoria University of Wellington
School of Architecture
2021

Acknowledgments

First, I want to acknowledge my supervisors; Tane Moleta, Andre Brown, and Marc Aurel Schnabel for your unwavering support throughout this year. With your guidance, I have reached a point in my studies that I never thought were possible, so thank you all.

Also, to my peers, and fellow DARA'ites, thank you all for your support and guidance throughout this entire year. We have each produced interesting and compelling projects that we should all be proud of.

I want to say a huge thank you to my parents. Mum and Dad, thank you for everything you have done for me over the last five years at university. You both have supported me every step of the way, whether it was a home cooked meal, or helping me fulfil my aspiration to do an international exchange to the United States. This publication marks the end of my Master's of Architecture and my studies, now onto the next chapter.

Lastly, I want to thank my partner, Brittany. You have been there every step of the way through this research project, through the highs and lows, and everything in between. Thank you for listening to my architecture rants everywhere we go, and for that, I thank you for everything.

Thank you, Victoria University of Wellington, The Bach Boys, Tane, Andre and Marc.

Sec 00
PREFACE

My journey began on exchange at the 'Rhode Island School of Design' (RISD) in Spring 2019. The Department of Architecture, where I was studying, had a map of the different courses. This was much like a road map often used on road trips with roads forked and spread out in different directions. The courses all lead to different destinations; however, they all commonly lead all to an eventual 'understanding' of the course material, which was in this case, Architecture.

In selecting my courses, the head of the school asked me to describe my interests in three words: Architecture, computers, and drawing to which I responded. As I began to explain this in more detail an associate professor within the Department of Architecture walked in. By way of coincidence, he happened to be the professor of computational drawing research, Carl Lostritto. He explained his course encourages students to form a "relationship" with the computer, to view the computer as a work colleague.

Carl mentioned he was in the process of publishing a book titled 'Computational Drawing - From foundational exercises to theories of representation' (Lostritto, 2019). The book teaches and documents drawing with code. It highlights the ways in which designers can capture the processing power of the computers to 'draw' in a way that had not previously been explored. This sounded like an obscure idea, but my interest was piqued and I was eager to explore further.

In his computational drawing course, Carl works alongside his students and research assistants hacking machines, writing computer instructional language or code, sampling from history, designing tools and adapting technology. This acts to augment human authorship in pursuit of architecture. Much of this work and that of his student's is representational. It invokes an interpretation of 'drawing', as the drawings themselves are situated within architectural practice. The course teaches reflection, analysis and critique of drawing relative to form and space to achieve this link.

I began to consider how I could apply this thinking, combine my interest in computers, and use it to design. The course sounded perfect! Unfortunately, that year the course was taught in Fall, which was when I was due back in New Zealand. Nonetheless, I couldn't help but think how I could use this idea for my thesis.

Programming was being taught in a school of architecture. Could I use this back home as a thesis research field? Would this be considered by the Victoria School of Architecture in their course road map?

This thesis began with this talk.

Sec 00
PREFACE

This diagram provides the road map illustrating the relationships between project families and the thought / action threads across this thesis journey. The x and y axes connect the trajectory of ideas over time, with the main concepts spanning the full journey. It can be read horizontally as a series of correlations, revealing the evolution of ideas, or vertically as a snapshot in time. Highlighted in yellow are the ideas that are crucial to the end product of this thesis.

After a long discussion about courses and eventually choosing what I was going to be enrolled in, I set off back to my accommodation with a mission to obtain a copy of the book. Once the book had arrived, there was no hesitation in getting to work. This involved teaching myself to code, learning from my mistakes, and ultimately, finding a love for coding which I still carry today.

Using code to create architecture was still at the forefront of mind upon returning to New Zealand and continuing with fourth year second semester architecture. However, I still did not know how to bring this idea to fruition. I needed to learn how to code from scratch, to produce something that was not 'random' and that resembled architecture. My aim was not to produce a house or a building but to elaborate on how we, as architects, work with spatial conditions to invoke a mood or feeling within the given space. I decided to set off on my journey to find the place and position for code within architecture.

One thing Carl said to me still rang in my mind, "we teach students to work with the computer as a friend, not a slave" (C. Lostritto, personal communication, 2019).This became a source of inspiration throughout this thesis. However, in the meantime, I needed to get a grasp on how to produce that code.

Computing and drawing have intertwined histories. The first computer art was drawn by A. Michael Noll. Noll programmed a digital computer at the Bell Telephone Laboratories in Murray Hill, New Jersey (Noll, 1994). Noll generated patterns solely for artistic purposes, although his later computer generated works simulated paintings done by Piet Mondrain and Bridget Riley. These became 'classics' (Dietrich, 1986). Limits on processing power and memory storage inhibited pixels and images. Instead, computers were programmed to control machines that moved pens to make marks on paper. Initially, artists saw a utilitarian advantage to using computers to generate art, the computer was seen to be an 'accelerator' for high speed visual thinking (Mohr, 1983).

Soon thereafter, the trajectories of computing and drawing diverged. In addition to exploring the meaning of computing and drawing, this thesis will explore what computing and drawing can collectively do. In most cases, combining computing and drawing involves merging cultures and methods. Typically, the "computing territory" and "drawing territory" are largely distinct. Deciding to 'draw with a computer' or 'compute in the space of drawing' may be considered complex and abstract. Nonetheless, such tasks are creative, exciting, and productive.

Prior to meeting Carl, I was fascinated and excited by coding. My focus was animation and graphic design using code, alongside my architectural studies. I was seeking computational methods that might inform architecture by way of provoking animation to perform in ways other than the simulation or the diagram.

STARTING TO CODE

Firstly, anyone who is looking to learn how to program, you are 99% of the way there already believe it or not. If you can read and write, you already have the foundation. Contemporary programming languages such as Java and JavaScript and libraries that augment their basic capacities to establish “fussy and complex” environments are often associated with the term “code.” The term essentially means to write in a language that can be interpreted and executed by computers. The language can also be read (interpreted and executed) by humans.

Secondly, you can start making computational art immediately. You do not need to be an expert in computer programming before you can start applying computation to drawing. In fact, learning to code within the context of ‘drawing’ will help the artist, architect or designer learn to code more efficiently and effectively. It will be more fun and more satisfying.

Readers within this discipline - architects and architecture students - will recognise particular concerns and questions. For example, an interest in space and form, the question of scale, the capacity of the drawing to become a work of architecture through mental inhibition and interpretation. These questions are not unique to architecture.

Those outside the discipline may not recognise the quirks, hang-ups, and particular obsessions of architecture when they appear. They may be surprised by the lack of 'buildings' in this thesis. Strictly defining a construct or product as "architectural" versus "artistic" is avoided in this thesis, with the hope that readers avoid the temptation to "read-in" their own disciplines.

Architecture and Other Disciplines

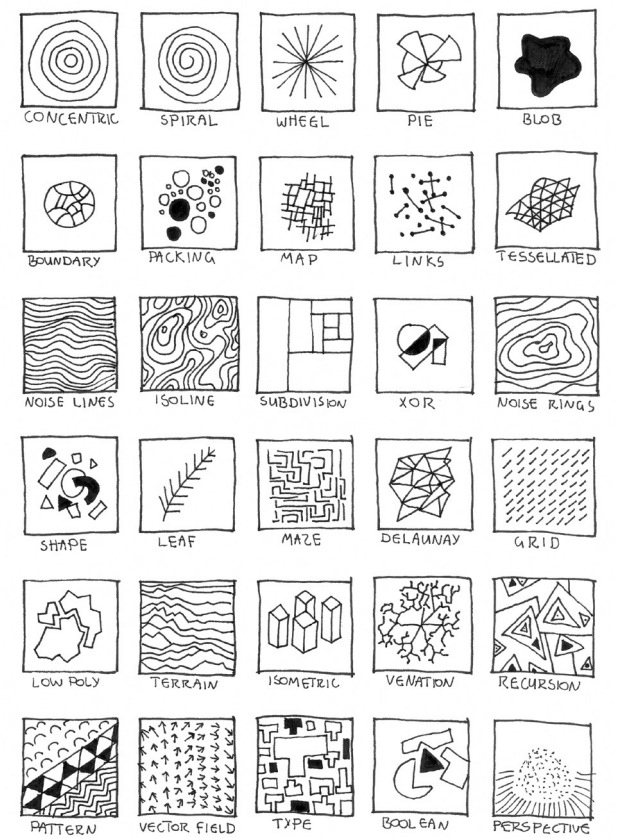


Figure 0.02 Thumbnails of design language used within this thesis. Hand drawing, by Author.

Sec 00

CONTENTS

COLLEGIAL COMPUTATION

This diagram provides the road map illustrating the relationships between project families and the thought / action threads across this thesis journey. The x and y axes connect the trajectory of ideas over time, with the main concepts spanning the full journey. It can be read horizontally as a series of correlations, revealing the evolution of ideas, or vertically as a snapshot in time. Highlighted in yellow are the ideas that are crucial to the end product of this thesis.

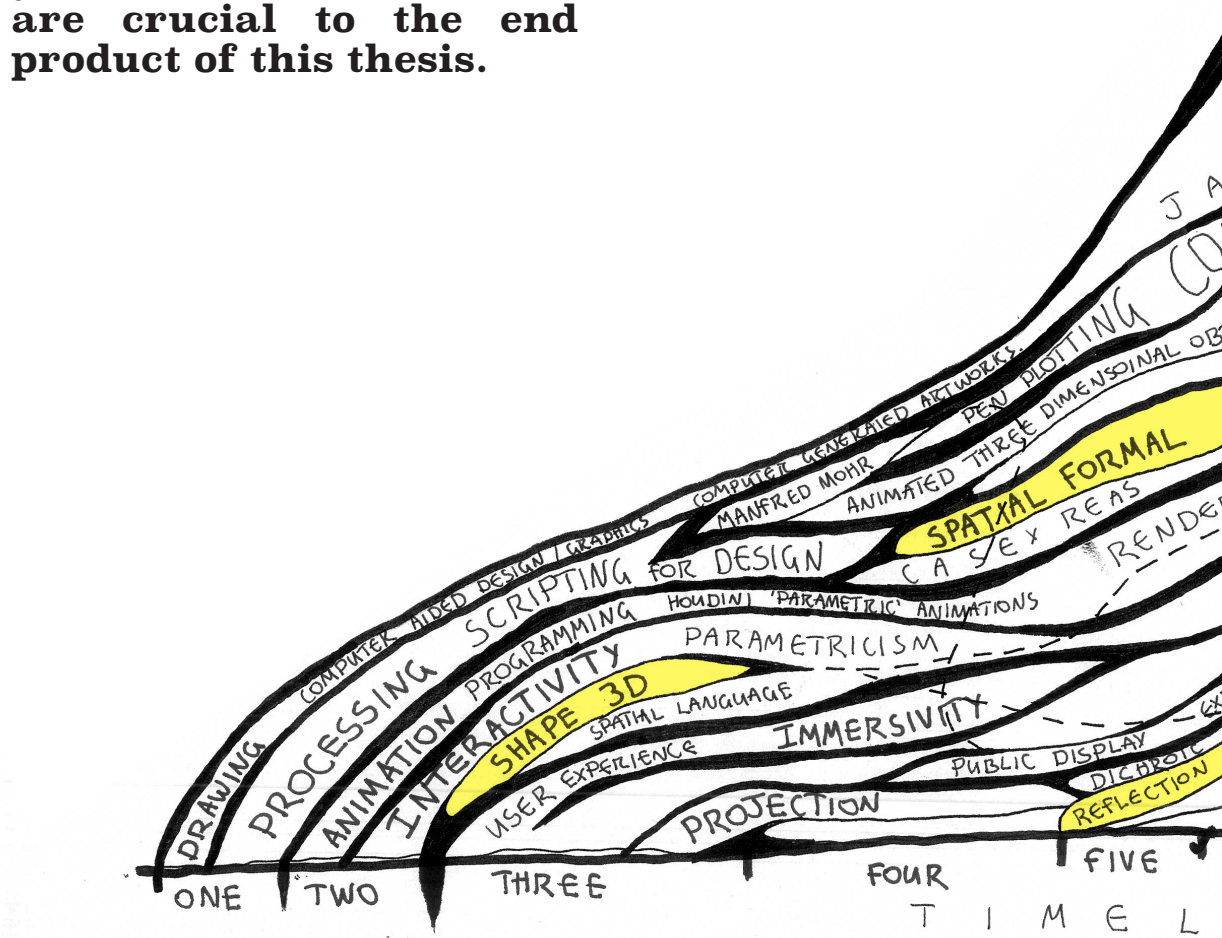


Figure 0.03 Diagram of this Thesis's journey. Hand Drawing, by Author.

CREATIVE
PROGRAMMATIC
USER CENTRIC
FORMAL/SPATIAL
CONCEPTS / IDEAS
DIGITAL DESIGN THEORY
CREATING ARTWORKS
PRINTS
SOFTWARE
SINE
TANGENT
COSINE
FORM
PARAMETERS
PIXELS
MATHEMATICS
CODE + OBJECT = OBJECT BASED PROGRAMMING
COLOUR
RGB vs CMYK
VARIETY OF INPUTS
DESIGN TEACHING TOOL
COLLEGIAL
COMPUTATION
ARCHITECTURE
RULES
SCALE
LIGHTING
SHADING
CONDITIONAL
USER CONTROL
USER ATTENTION SPAN
INPUT
MOUSE
INSTANTANEOUS FEEDBACK
PROCEDURAL
USER INTERFACE
VISUAL
SELF POWERED
MATERIALITY
RELATIONSHIP
PUBLIC ENGAGEMENT
HEADSET
UNITY SPECIAL EFFECTS GRAPHING
C#
VIRTUAL REALITY
HYVE EXPERIENCE XR
RULES CONSTRAINTS
SFX
ATTENTION TO DYNAMIC FORM
LIGHT
VIDEO
VOXELS
VIRTUAL WORLD
VOLUME
PUBLIC SCALE
CONSTRUCTION
GEOMETRY
SIX
SEVEN
EIGHT

Abstract

This research thesis is an architectural inquiry into how scripting techniques can be used within the conceptual stage of architectural design by architects, students and lay-people as a method to generate and create architectural form.

The intention is to create generative procedural programs using Processing, that promote shared agency throughout the design process as a whole. This agency looks at creating an engaging 'conversation' between the user and computer, allowing the computer to have an equal share in the design process.

By programming a range of varying design tests that experiment with conceptual form finding and massing studies, this research aims to experiment with agency through the development of procedural design processes. Together these act

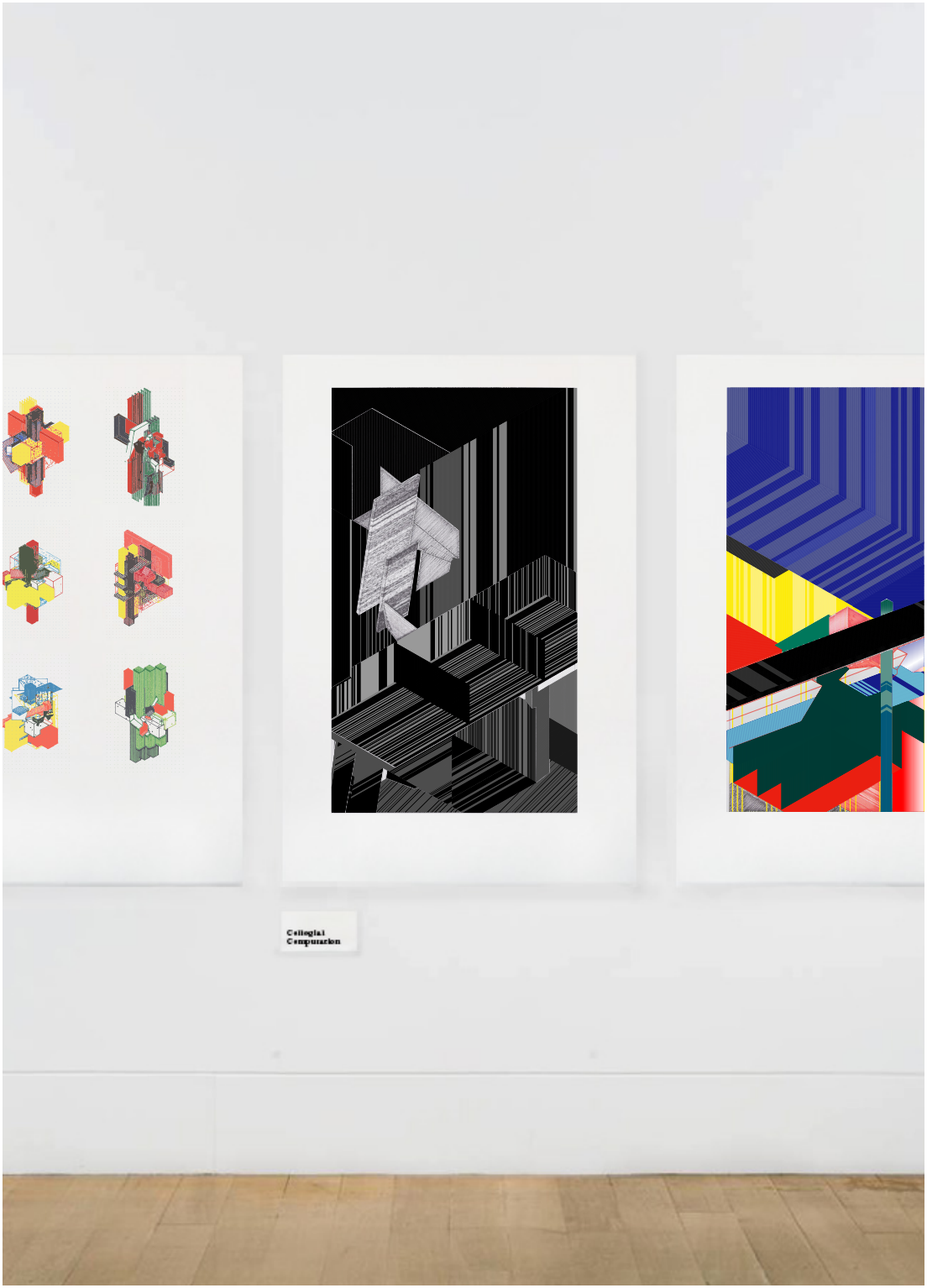


Figure 0.04 A superimposed gallery view of work generated within this thesis. Prints generated with Processing and Illustrator, by Author.

as a vehicle to activate the research question, 'how can we use scripting techniques within the conceptual design stages of architectural design to generate and create architectural form?'.

The results of this research, while provocative, will contribute to understanding how architectural form produced through scripting techniques influences the design outputs, and design process as a whole.

The term 'generative' is used in this context to define the iterative design process; which involves a program that generates outputs, which meet certain constraints, from a series of input variables, ranges and distributions. This process allows the designer to fine tune the desired region by selecting specific outputs or changing the inputs.

Research and discovery through design was used to develop the final generative programs which interpret architectural form. The results show that designing with a digital 'partner', and the sharing of design agency throughout the design process; can generate conceptual architectural form from infrequently investigated computational design methods.

Getting Started 0.0

Packing the Vehicle

What should / would you take on a road trip? I always pack the biggest bag I can find and never wear or use half of it...

In this opening section, Packing the Vehicle, the background of this research thesis is explained in terms of “generative art”.

As we do on a road trip, we prepare ourselves by packing the necessary items we will need while we are away from home. Much like this analogy, the background research situates itself as a learning opportunity as well as a tool kit for moving forward through this research.

This research follows the ‘generative art’ culture, which began in the 1960’s. The development of computers with high processing power. Their increased accessibility and availability has led to an exponential increase of generative art. This has made this field of artistic expression and generation more attainable. In this section, I will provide a guide to defining generative art. Beginning with algorithmic artists in the 1960’s, to myself in 2020, this art form of using computers to create art has expanded across the globe.

His background research should be read as a case study of this niche field. One simple but useful definition is that Generative Art is art produced by programming a computer to intentionally introduce randomness as part of its creation process. The “randomness” commonly leads to two distinct viewpoints that inhibit individuals from appreciating the beauty and variation of generative art.

Myth One: The artist has complete control and the code is always executed exactly as written. Generative art lacks elements of chance, accident, discovery, and spontaneity that often makes art great, if not at least human, and approachable.

Myth Two: The artist has zero control and the autonomous machine is randomly generating the designs. The computer is making the art and the human deserves no credit, as it is not really art.

In reality, generative artists skillfully control both the magnitude and the locations of randomness introduced within the work. Controlled randomness may sound contradictory; however, artists have always sought ways of introducing randomness into their work to stimulate their creativity.

Harold Cohen was one of the first to become involved with computer drawing in 1968. He wrote that the emerging generative artworks have “well defined rules and with the use of random number generators, have guaranteed the creation of never-ending variations of drawings with a very distinctive style (Cohen, 1982). The process of coding generative art is comparable to painting or sketching. In fact, we see that the tool favoured by most generative artists refers to the individual artworks produced as “sketches” (Bailey, 2018).

Those artists that first experimented with coded artistic procedures included Hiroshi Kawano, Herbert Franke, Manfred Mohr, Freider Nake, Georg Nees, Vera Molnar and Edward Zajec (Wrigley, 2015).

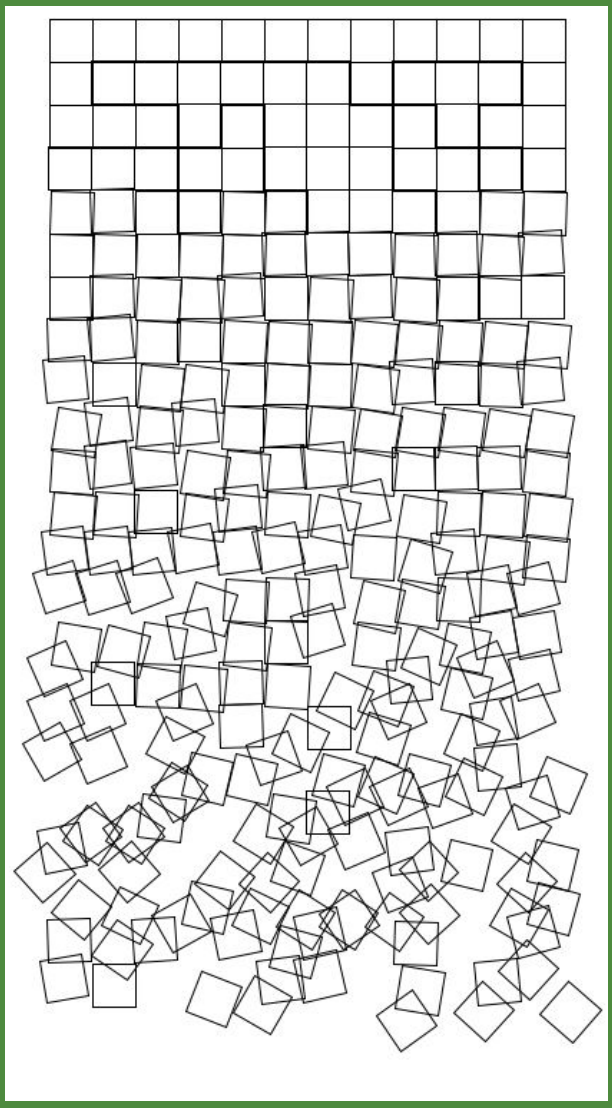


Figure 0.05 Image based on Georg Nees 'Schotter' circa 1965, made in Processing., by Author.

Georg Nees' 1968 work Schotter (Gravel) is one of the earliest and best known pieces of generative art. Schotter starts with a standard row of twelve squares and gradually increases the magnitude of randomness in the rotation and location of the squares as you move down the rows. Imagine you drew the image yourself using a pen and paper. Assuming it took one hour to produce, it would take you another ten hours to add ten times the number of squares. Comparably, Nees could add thousands of additional boxes to the composition through small changes to the code. This is an intelligent and important characteristic of generative art.

Unlike analog art, where complexity and scale require exponentially more effort and time, computers excel at repeating commands and processes without exhaustion. As we will see throughout this thesis, computer aided design testing and calibration create complex images, contributing to the aesthetic of generative art.

As with many innovations, there were several pioneers exploring the potential of generative art in its first few years as an emerging artistic culture. These pioneers, Nake and Noll along with Nees were all using computers which typically had no monitors, and the works were shared by printing the art on plotters (large printers designed for vector graphics).

What will be developed in this thesis is not considered solely artwork, but a systematic manner of thinking to create architectural form through the use of programmatic tools.

Throughout this thesis, there will be terms used to describe actions and processes within the practice of art and design.
I have provided a glossary of terms used by designers in the field. The terms will be defined with a colloquial meaning, and an example for ease of understanding.

Glossary Of Terms

Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

Processing

Processing is a free graphical library and integrated development environment built for the electronic arts, new media art, and visual design communities.

Generative Design

Generative design is an iterative design process that involves a program that will generate a certain number of outputs that meet certain constraints.

Generative

Having the power or function of generating, originating, producing, or reproducing.

Fortran

Fortran is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

Pen Plotter

A plotter produces vector graphics drawings. Plotters draw lines on paper using a pen.

Sketch

In terms of a 'Processing Sketch', a sketch is defined as being an experiment or visual display of the code results.

Code

a system of words, letters, figures, or symbols used to represent others, especially for the purposes of secrecy.

Algorithmic

Expressed as or using an algorithm or computational procedure.

Parametric

Parametricism is a process based on algorithmic thinking that enables the expression of parameters and rules that, together, define, encode and clarify the relationship between design intent and design

Procedural

The use of a procedural or rule based system, provides the user with a toolkit of possible actions to be combined infinitely.

Computational design

The terms design computing and other relevant terms including design and computation and computational design refer to the study and practice of design activities through the application and development of novel ideas and techniques in computing.

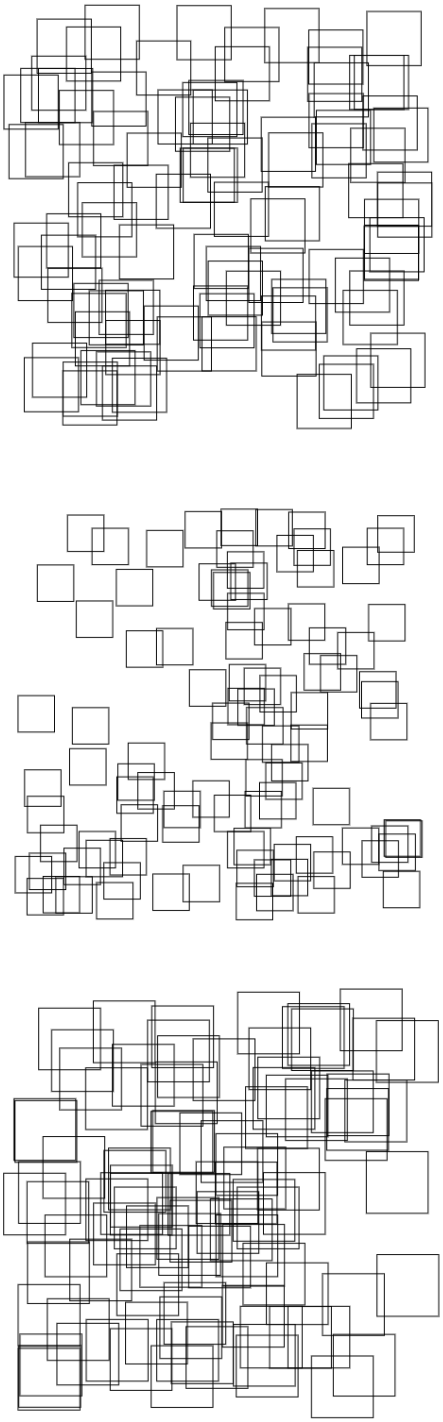


Figure 0.06 A series of ‘random’ rectangles generated in the sketch window. Made in Processing, by Author.

A Guide to Defining Generative Art

From an artists' point of view | Leading to a general overview

Generative art refers to art that in whole or in part has been created with the use of an autonomous system. An autonomous system in this context is generally one that is non-human and can independently determine features of an artwork that would otherwise require decisions made directly by the artist (Wrigley, 2015).

There are clusters of contemporary generative art activity that include; Electronic Music and Algorithmic Composition, Computer Graphics and Animation, The Demo Scene and VJ Culture, and in terms of this thesis, Graphic Design and Architecture (McClintock, 2020).

Design practice has always included an iterative process or creation of a number of samples. The designer then selects among these samples, making incremental changes, additions, and improvements to make hybrid samples, again evaluating the results and so on. This manual and time-consuming process is fairly reminiscent of the evolutionary process of genetic variation and natural selection.

It was seemingly inevitable that after the adoption of the computer by designers as a manual tool for CAD, there would follow the adoption of genetically inspired algorithms for the creation and selection of variants. In fact, generative artist William Latham initially used a system that is based on evolution, that existed on paper, and only later did he move to computerised versions (Kemp, 1998). Latham describes the adoption of computer art as not being a tool that is to perform old artistic tricks, but rather a more profound identification of the new territories computers and art might inhabit in terms of the inherent nature of computational procedures (Kemp, 1998).

Latham explains that standard computer-aided design programs which are aimed specifically at artists are seen to provide ingenious extensions to hand driven techniques. Nonetheless, at this time there were only a handful of others conducting a broader exploration of art forms that can be generated only with computers (Kemp, 1998).

The question "what is art" can be useful. Viable contemporary definitions of art include a notion blurred toward set theory such that some of which may be considered more "art-like" than others.
In an equivalent way we can expect that some works are more generative than others. In addition, current notions surrounding art recognise it as a social and historical activity that evolves over time.

The word 'generative' simply directs attention to a subset of art, system, and process. A subset where potentially multiple results can be produced by using a kind of generating system. These generative methods are characterised by the generation of different solutions, wherein the decision maker has to choose one solution among them.

A useful definition of generative art should one, include known clusters of past and current generative activity, two, exist as a subset of all art whilst allowing for the definition of "art" to be contested, and three, be restrictive enough that not all art is generative art (Galanter, 2003).

"Generative art refers to any art or design practice where the designer uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural intervention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art" (Galanter, 2003).

The use of a complex autonomous system for art making is the defining aspect of generative art. The generative systems can be used to automate or help the designer in their design process, allowing the designer to run through iterations faster and more efficiently and providing the designer with a seemingly infinite number of possible solutions to a design problem.

"Generative art/design can help automate the creation of options, which satisfy a variety of goals that the designer wants to achieve. It can be an exploratory tool to open up a designer's thinking - not necessarily solving the problem or providing the right answer but aiding this exploration'' (Galanter, 2003).

Using algorithms to guide the creation of design options, the computer will provide the 'best' options, given the goals and the size of the exploration that have been specified.

''When there are several inputs to the design, it becomes increasingly hard for the human mind to keep track of all the combinations of input. The computer is not limited and will surprise one with the combination of different inputs and outputs'' (Galanter, 2003).

This is the beauty of a generative system. The algorithm defined by the designer may be simple in concept. In practice however, if the designer 'instructs' the computer through code to compute the position of a sphere on the surface of a pyramid a thousand times, the computer will 'think' of all of the places in which it can place the sphere. Ultimately, it may defy human logic and produce a surprising outcome.

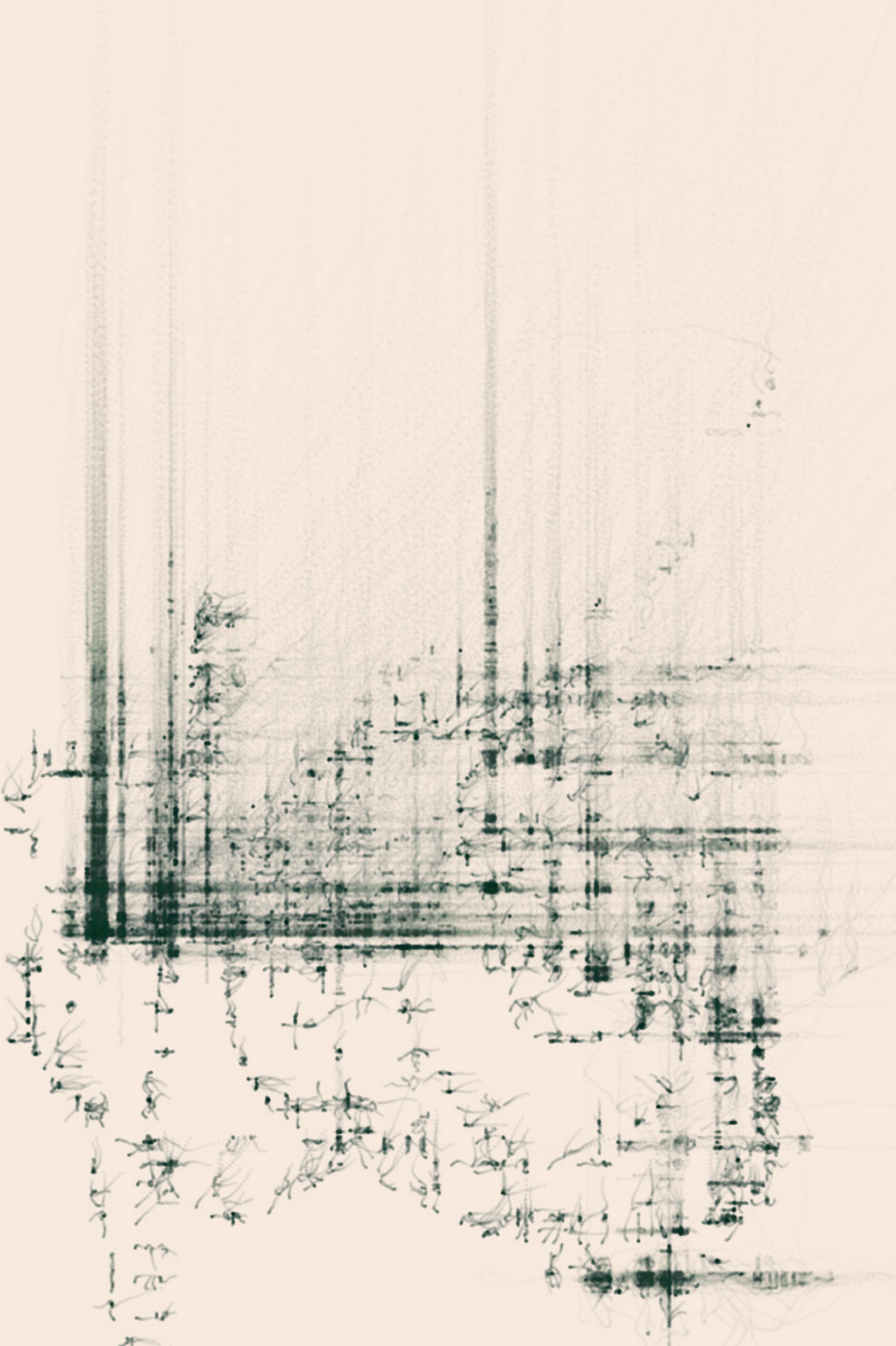
In this research thesis, I am not suggesting the computer is being powered by AI in the general sense of the word. That is, the computer's ability to simulate human intelligence or to think and act like a human. Rather, I am referring to the computer's ability to exhibit traits associated with the human mind such as problem solving and developing an idea.

The computer and processor are reading a set of instructions it can understand: the code. The designer interacts with the computer by writing code in English. This is then interpreted and translated by the computer into rules and instructions which it will follow. In writing this code, the designer provides the computer with a 'choice' or 'chance'. Chance being the ability to choose a number based on another, much like a multiplier. The computer then makes its own decision based on a 'chance' value.

For example, if I were to instruct the computer to choose a number between one and 100, it may choose 59 or 63. These two numbers may be different to those I would have chosen. Given the results, I may choose to repeat this exercise and provide additional parameters or boundaries, such as "choose a number between one and 100, but only if the number is larger than 58 and smaller than 64." This is the way in which the computer and designer interact throughout the programming process.

Terms and applications such as "generative design" are often thought of as forward-thinking and cutting edge. Programmatic artists have used this creative method since the 1960's. Fifty years ago, Vera Molnar, a Hungarian computer artist working on developing the early programming language 'Fontran', used her expertise to generate images examining theme, variation, automated generation, and display of options. Digital artist Manfred Mohr, another pioneer in algorithmic art, generated variations of 3D geometry in the 1960's and 1970's.

Figure 0.07 Beside
Emulation of ink using particles through a Gaussian noise field , made using Processing by Author.



This thesis was a journey filled with options, detours, wrong turns, potholes, and forks in the road. By implementing one thought or idea, reflecting upon it with my supervisors and peers, new very different ideas and routes became apparent. Despite this, the aims and objectives remained constant throughout.

How can we use scripting techniques within the conceptual design stages of architectural design to generate and create architectural form?

Research Aims & Objectives:

1. Use computer code as a procedural method of creation to generate architectural form and spatial qualities.

2. Expand the architectural potential for a generative system to aid conceptual design methodologies in the context of architectural education.

3. Explore how designers can use computer code to enable a generative approach to creation.

Getting Started 1.0

Drawing With Processing

For this initial phase, I want to understand how Processing works. To achieve this, I will use the program to create computational art. I experiment with the program's multitude of possibilities and teach myself how to 'code'. During this opening section, I will be exploring the basics of the program, and seek to uncover a hidden architectural possibility to this visual design program.

Processing is a free graphical library and integrated development environment (IDE) built for the electronic arts, new media art, and visual design communities. The purpose is to teach non-programmers the fundamentals of computer programming in a visual context (What Is Processing?, 2021).

Processing uses the Java programming language, with additional simplifications such as additional classes, mathematical functions and operations. It also provides a graphical user interface (GUI) for simplifying the compilation and execution stage. Users can download Processing and use it for their own projects. Processing uses textual notation that consists of up to 95 percent Java syntax. It is easy to transfer the code and examples to many other textual development environments. Processing is cross-platform, which means the same source code can be used on all operating systems for which a Java platform exists (e.g., Mac OS, Windows, and Linux) and can also be integrated into websites.

An ever-growing, vital, and supportive online Processing community exists that actively exchanges ideas on the Processing website. The website contains online references of all language elements and an index of supplementary libraries.

Processing was founded and initiated in 2001 by Casey Reas and Ben Fry, both formerly of the Aesthetics and Computation Group at the MIT Media Lab. In 2012 they started the Processing Foundation along with Daniel Shiffman. Programming with Processing features an easy to learn Creative Coding methodology, which makes it easier for the non-programmer to use. With visual feedback in the way of a 'Sketch', learning to code/ design in a generative computational sense is made interesting and rewarding from the beginning.

Processing is regarded as being a Creative Coding software, where Creative Coding is a type of computer programming in which the goal is to create expressively, instead of functional outputs.

Initially what interested me about computational drawing was the unlimited variation one would obtain from a ‘simple’ script. By using the inbuilt power of code, variation is accessible to the designer. Building this variation into a script takes time, and experience. Learning to code is made simpler as a lot of computer code is ‘open source’. This means that the code is designed to be publicly accessible; anyone can see, modify, and distribute the code as they see fit. Learning this, as well as reading blog’s and watching Youtube tutorials at the beginning of my journey to code was very helpful. This was a good way to learn the basics of coding. Along with the visual feedback implemented into the program, making changes to the code to obtain a certain ‘look’ the designer is after is made more streamlined.

Processing programs are called “sketches”. Hence, Processing can be understood as an environment for the quick creation of digital artefact’s. The main folder where the user-created programs are stored is called the “sketchbook folder”. This in turn, invokes a creative aspect to a purely text based art form.

My aim is not to blur the lines between drawing with a pen and paper, and with the computer, but to draw a parallel between these two modes of graphical representation and their inherent interpretations.

To reiterate, this thesis aims to explore a generative approach to design within the field of architecture. Specifically, how this approach to design in a field that heavily relies upon the designers own unique robust creative process can be employed alongside to streamline the conceptual design process by augmenting pieces of the conceptual exploration process.

Through this first stage, there is emphasis placed upon architectural design elements such as, form and scale in two dimensions.

Starting at the foundational level, I began with detailed research into Processing’s syntax (the arrangement of words and phrases to create well-formed sentences in a language, or in computer science, the structure of statements in a computer language). While doing this research, I was introduced to “Open Processing”.

Open Processing is a web based open source designing platform built around the focus of teaching coding. With its minimalist design, it removes the complex coding jargon many platforms use. This allows students to focus on the code and visualise their results with a single click. Open Processing uses Java, Javascript, and p5.js. This allows students to access the most important features of the program without leaving their code.

Open Processing allows access to many pre-written code examples which can be used to build your own works. Alongside Open Processing, working through tutorials of Daniel Shifman’s YouTube channel “The Coding Train”, I learned how to understand code and how to write the code myself.

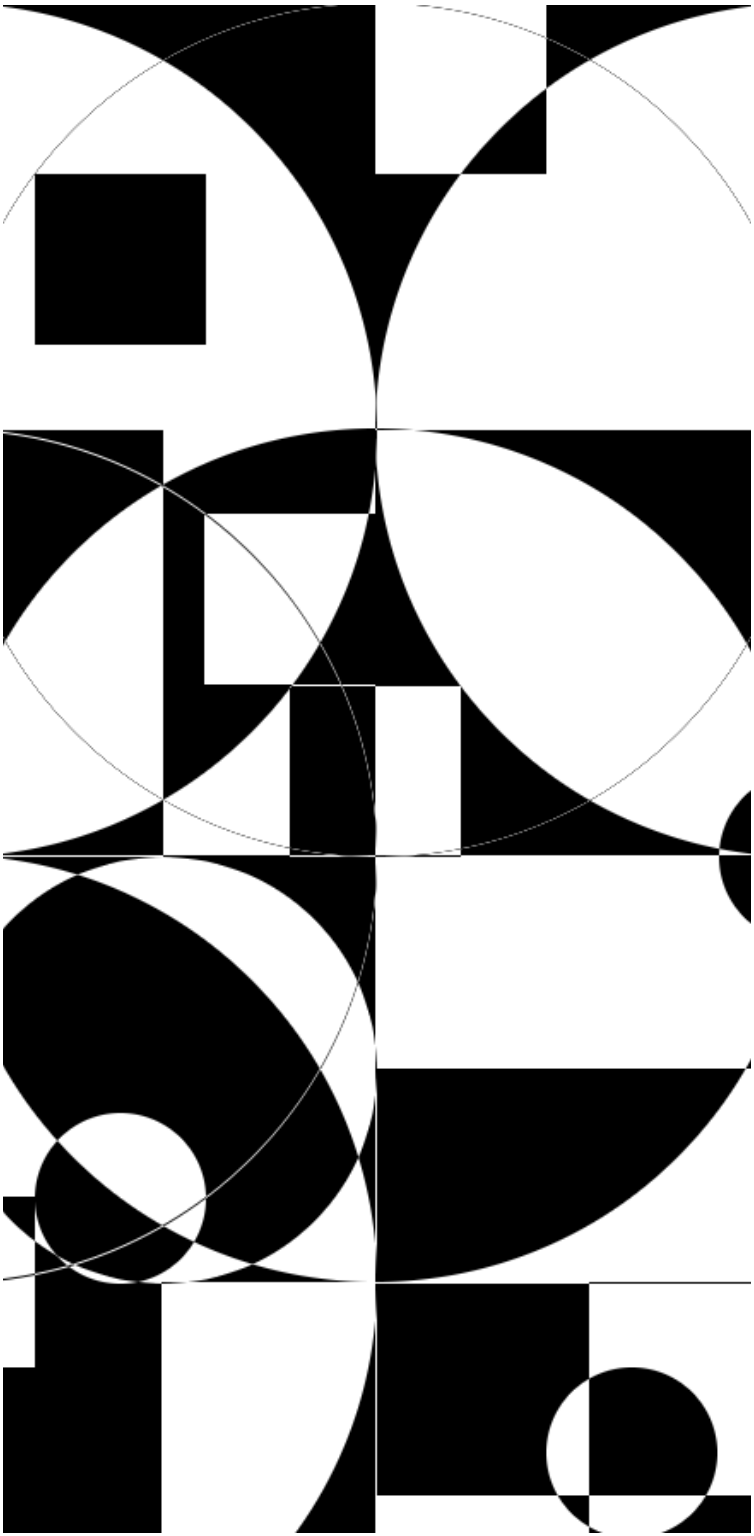


Figure 1.01 Mapping of overlapping squares, circles and rectangles. Made using Processing, by Author.

1.1

Drawing Experiments

To begin this stage of investigation, diving into the code was my first step. Looking at how to use code to create drawings, which are primarily viewed on the computer screen, began with opening Processing and typing in simple commands (such as those I had been introduced to while watching 'how-to' coding videos). This was an interesting exercise as it taught me the ins and outs of Processing's code syntax.

Displayed beside in are some initial drawing experiments. Form, scale, and weight were the three primary drivers to create these compositions.

From this initial investigation, it was apparent that this process or methodology was somewhat similar to drawing or taking notes. The user opens a text editor and begins to write down their thoughts. However instead of the user writing a poem, essay, or fiction, the thoughts are more about organising logic and procedures. In this case, the writing of code produces images. When an iteration of interest to the user is created, the user may develop a piece of the iteration by making incremental changes to the code.

A compilation of simple rules of code can produce creative artefact's that are visually different and go far beyond what was initially expected. This is the interesting part about code. This showed me the power of using a rule based design mechanism. The process of coding can be likened to "music". Every music performance is different. The producer and performer inject their interpretation on top. With respect to "processing," the software has been curated together and is "performed". Each performance (command executed by the computer) is performed differently by the software and computer. The elements unfold differently each time.

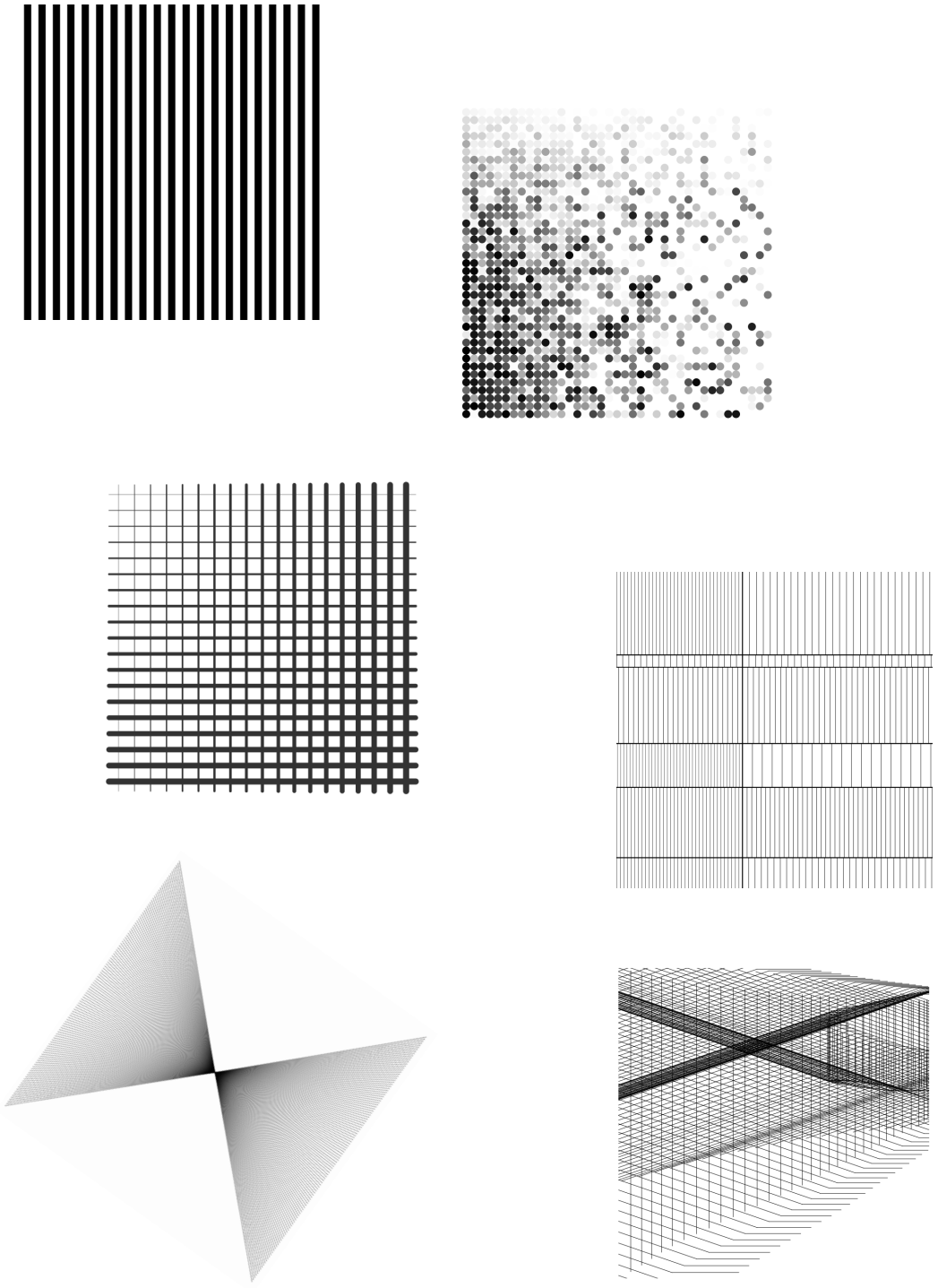
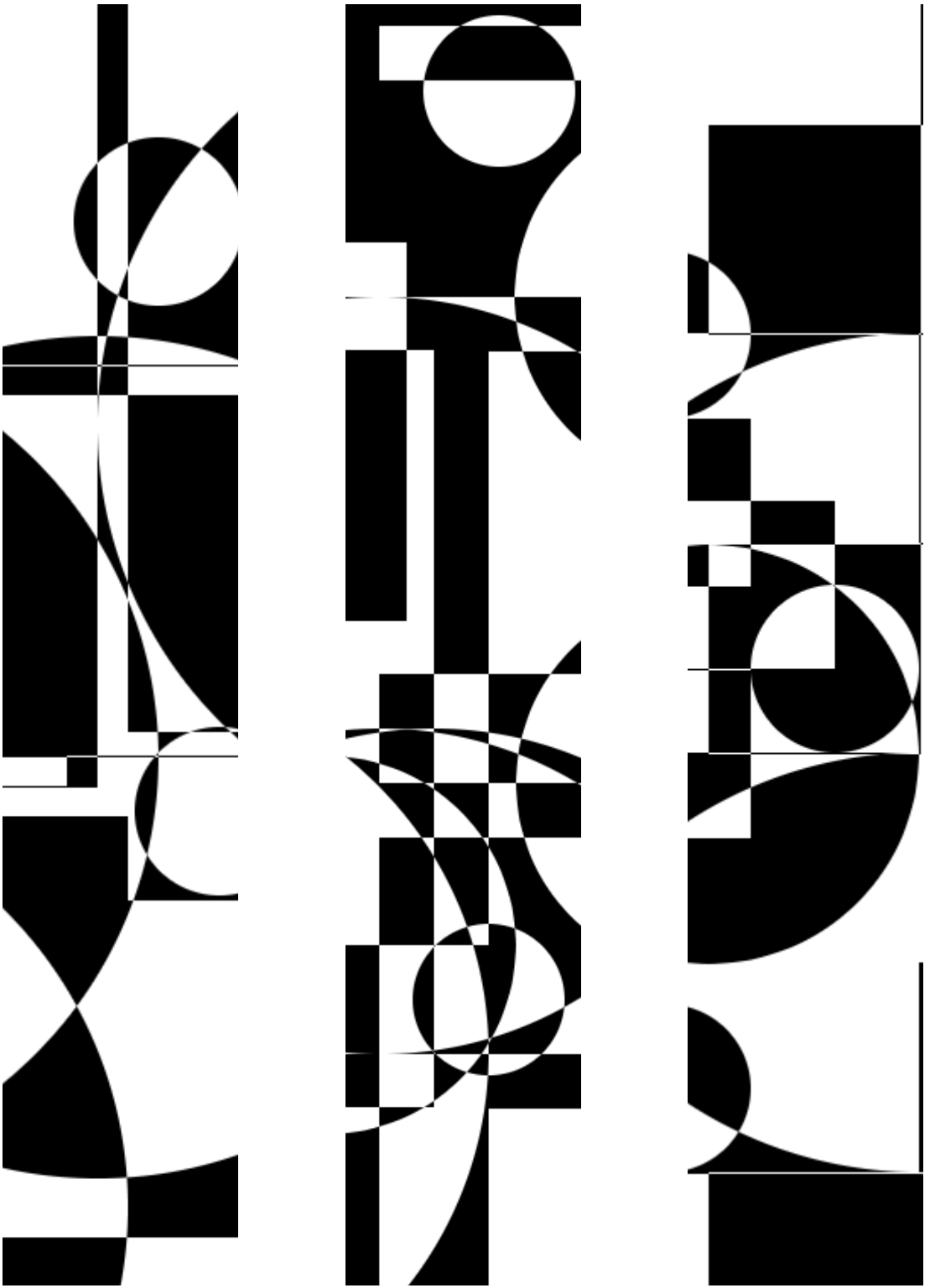
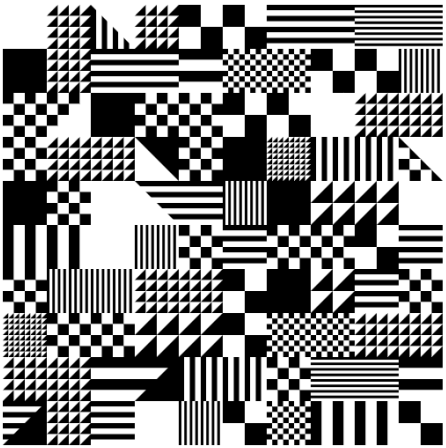
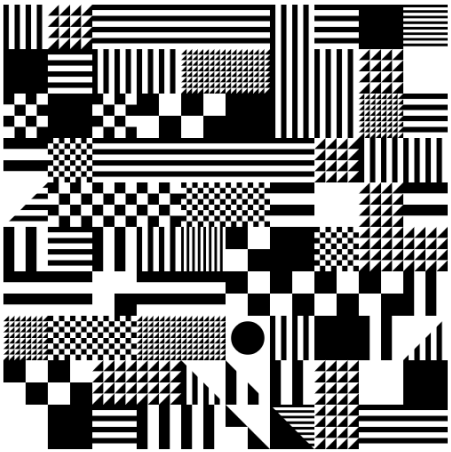


Figure 1.02 Initial drawing experiments examining form. scale, and weight.
Made using Processing, by Author.







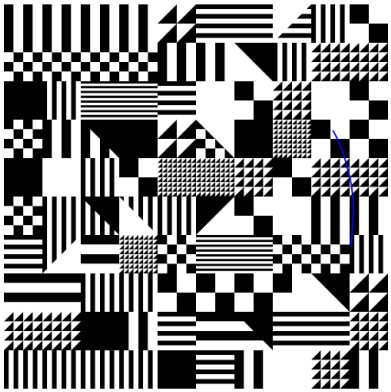


Figure 1.03 Left — Initial drawing experiments, circles , rectangles, and lines. Made using Processing, by Author.

Figure 1.04 Above — Initial drawing experiments, pixel based randomisation. Made using Processing, by Author.

1.2

Hand Drawing to Code

After my initial investigation into drawing with code, I wanted to investigate whether I could sketch (with pen and paper) a desired visual outcome prior to writing the code necessary to create it. I wanted to explore how I could plan the desired drawing to be created rather than sporadically writing code and seeing what comes of it. Particularly, I wanted to develop my own creative method from mind to paper, from paper to text, then from computer to final.

The next visual experiments show an indirect, yet close resemblance to my drawings through the interpretation of the drawing into code. Evidently, some outputs are vastly different to my initial sketches. This links back to my initial investigation of generative systems 'chance' and resulting outputs. When translating "thoughts" into "text," a layer of information can be negated or translated incorrectly.

Unlike drawing, code employs a rule based system of creation which allows the user to interact with various parameters that are set up at the beginning of the code. A rule based system, with minimal input from the user, has the ability to run through tens of iterations before the user could have otherwise created an artefact of their own. This is the beauty of code.

Typically with design, the iterative process can slow down the creative process. Employing a rule-based system into design workflow would enhance the efficiency of the otherwise prolonged process as it enables a greater scope of ideas to be tested. Commonly, this iterative process is a given for many designers, and acts to inspire and test potential ideas. For this reason, I see the rule-based approach to design a useful supplementary tool, not a replacement.

Code, as a rule based system, is not biased toward the want or eye of the designer. There is no emphasis placed upon the look of the artefact's. They are produced one after another, with either the user stopping the process, or the computer running to an iteration count. Another beauty of generative systems for me, is the virtually endless timeframe of creating the coded artefact's. The computer will not tire of producing artefact's, and there will always be something new the computer has not yet produced.

Variables written into the code allow the user to alter the way the computer interprets the code, therefore showing a visual representation of the computer's interpretation. If the user allows the computer to choose from a range of numbers, such as in a 'random', 'smaller than', or 'more than' operation, the computer has the ability to 'randomly' choose a number or numbers to use to create the visual output. This creates a larger scope of opportunity for the user having written code, and allows the power of the computer to choose and run through many different options.

Before I was able to write the code to initiate this design options variable, the same design was repeatedly produced. How could I design the code to allow for the input of a number to control the number of iterations produced? Pictured alongside is a series of iterations where I was able to instruct the computer to make a small change to the design, and produce it twice and with a different combination of the input variables.

This ability to iterate faster and more efficiently led me to consider how this could be used in the creation of architecture. Much of architectural designers' time is spent in the developed design phase. In other words, allowing a conceptual design to be built in real life. Although the developed design phase takes the most time, the conceptual design phase can be a struggle for an architect. Deriving a design from a meeting with the clients, site conditions and site specific restrictions can be a prolonged process, and slow down the completion of a project. Implementing a rule-based "coded" system for creation could overcome the time constraints. In this way, the architect could design the code that an architectural piece is derived from and implement this into a computer such that the computer produces different iterations for them. Thus allowing the computer to iterate through some of the initial design pieces, such as massing options, positioning on site, and form restraints, leaving the architect with this time back in their hands.

Through working in the industry, it has become evident how this methodology could be implemented in practice. Many architectural practices pride themselves on their conceptual design, which is usually done by a principal designer of the firm. Implementing a computer driven workflow into their process, as opposed to a pen and paper conceptual workflow may be challenging. This is because architects often form a relationship with their design, which they have poured their heart into. To have a computer inherit this position instead may be seen as a "red-flag".

A conceptual approach to design is looking at a problem or an opportunity creatively as opposed to pragmatically.. This involves looking at the 'idea' rather than a combination of approaches and processes. The computer looks at a set of problems, and without programming a set of rules, the computer will remain static. Rather than exploring different design options, it will not know what to do. This is the draw back from this conceptual mindset of using a computer to create architecture. A programmer would be needed initially to set up a set of rules for the computer, which the computer could test upon. Although the need to hire a computer programmer for an architectural practice may be perceived as a step in the wrong direction.

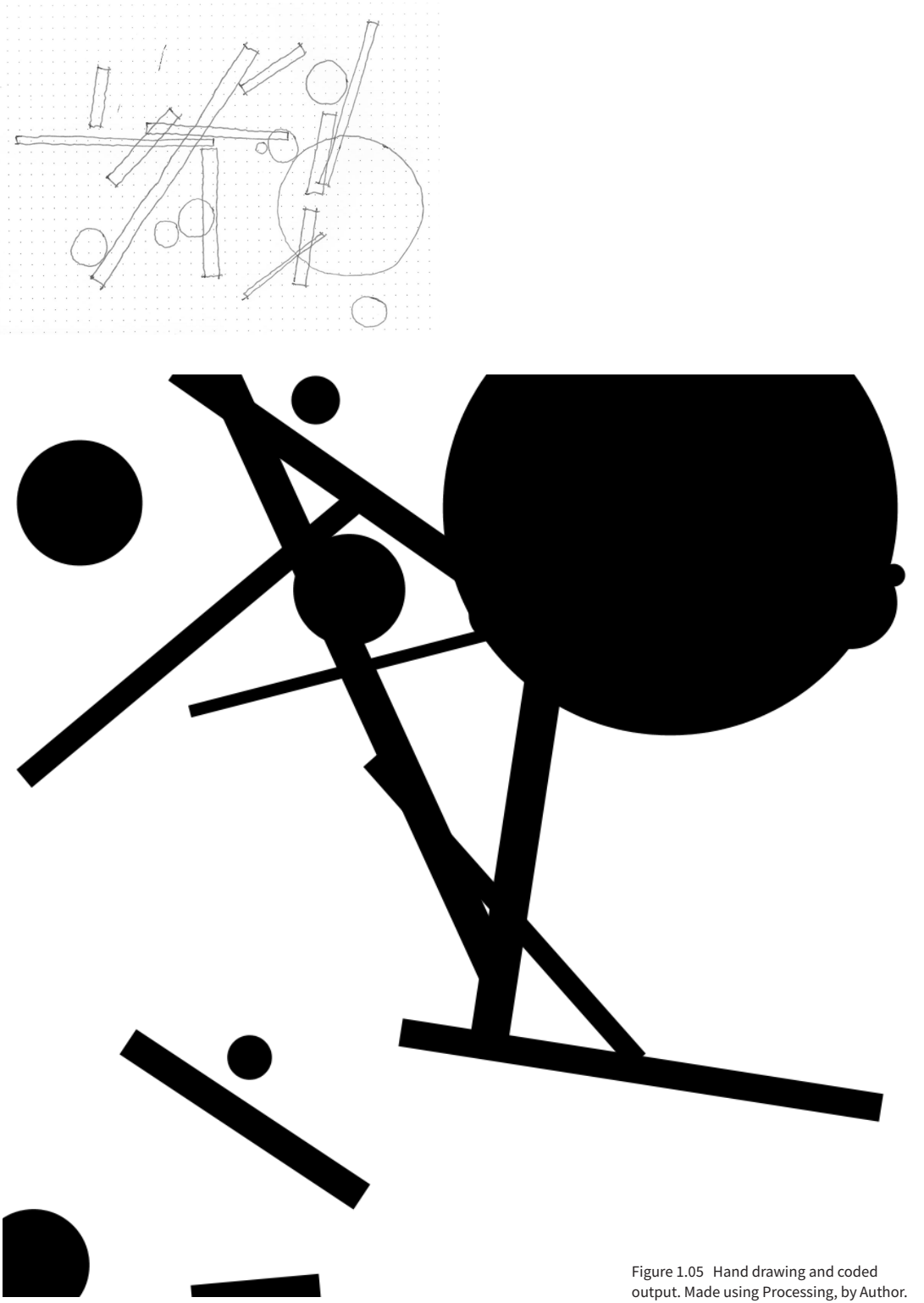
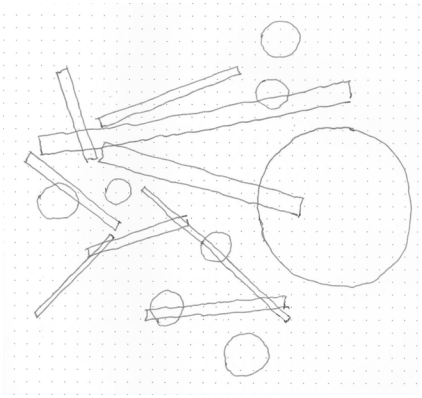


Figure 1.05 Hand drawing and coded output. Made using Processing, by Author.



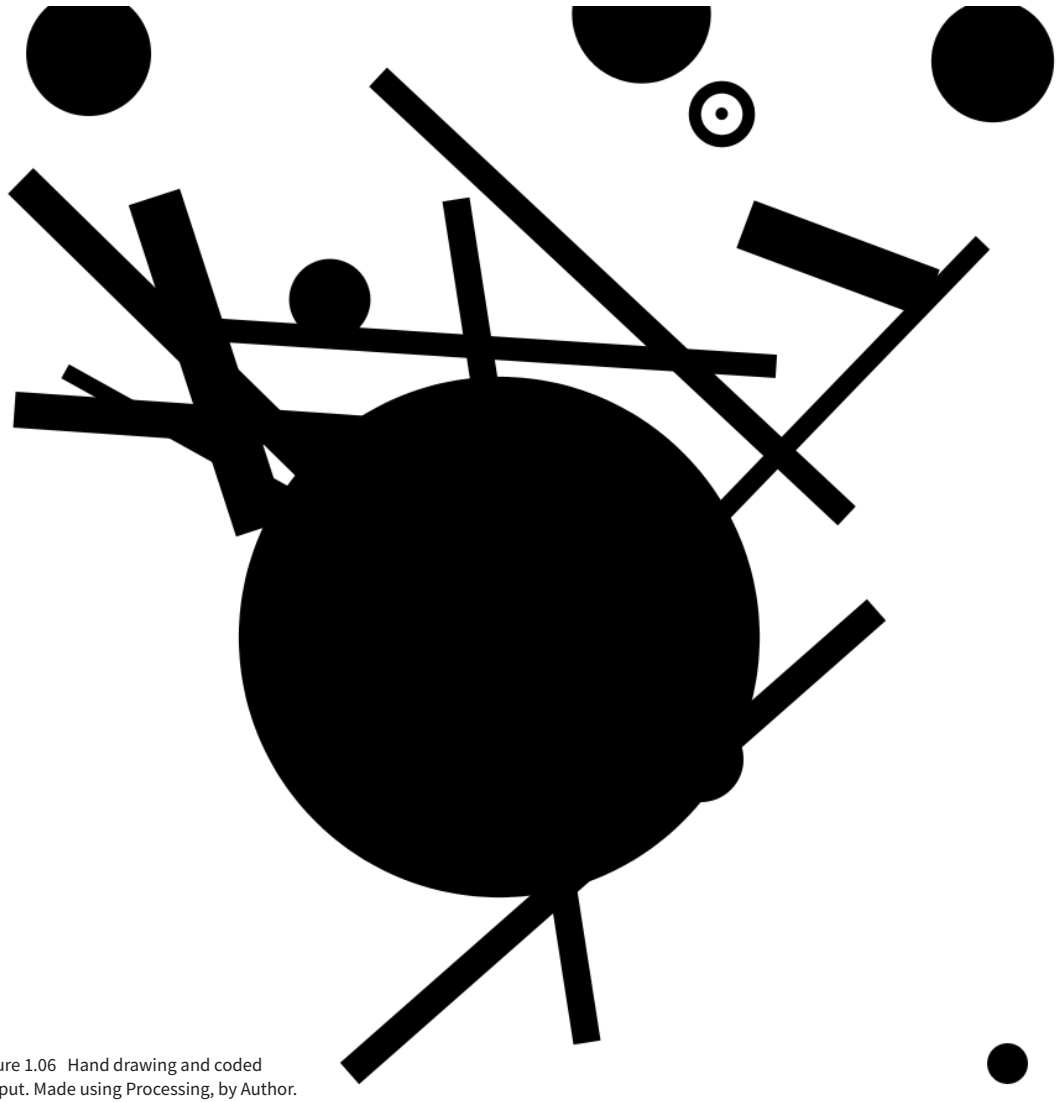


Figure 1.06 Hand drawing and coded output. Made using Processing, by Author.

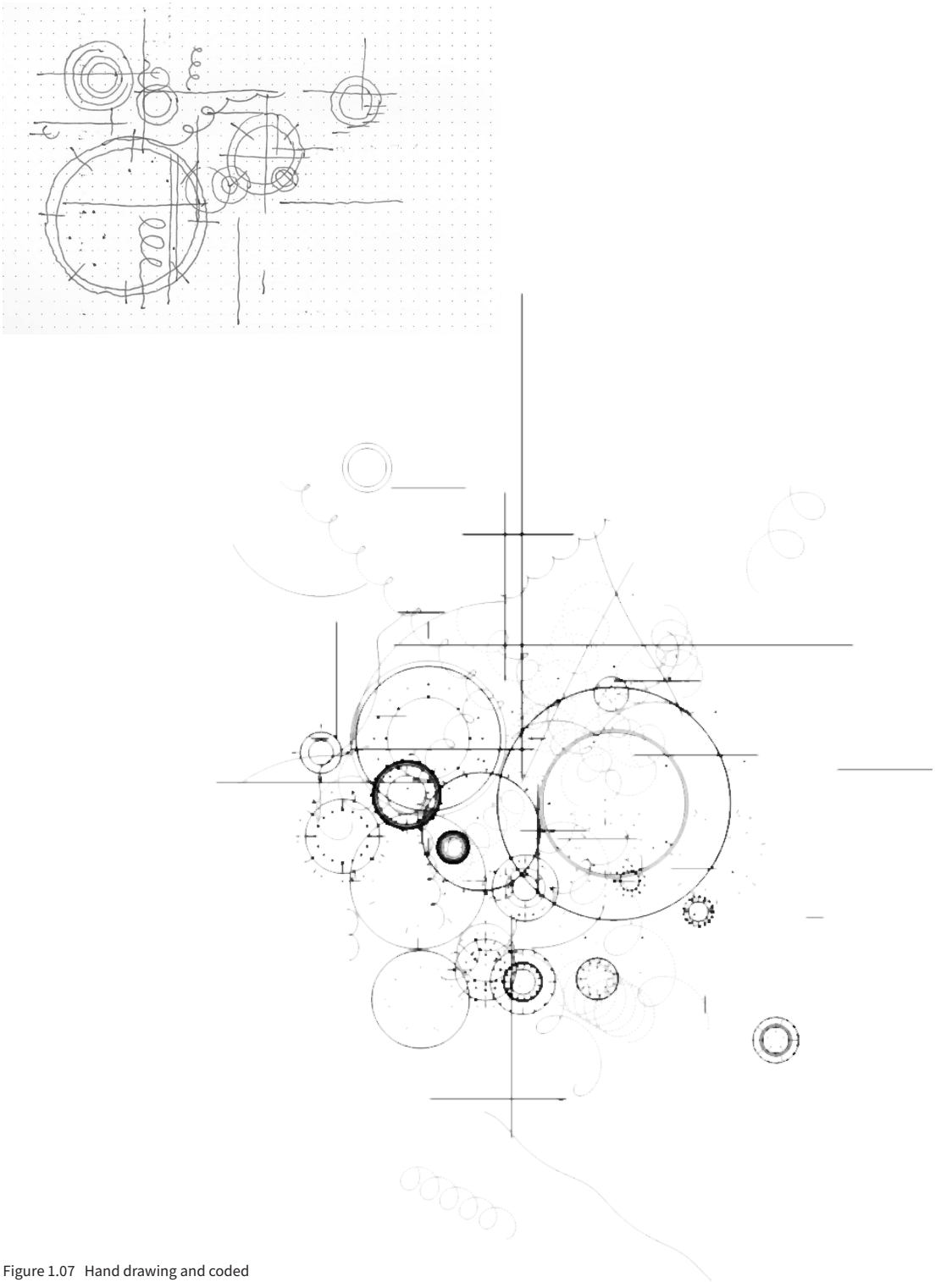


Figure 1.07 Hand drawing and coded output. Made using Processing, by Author.

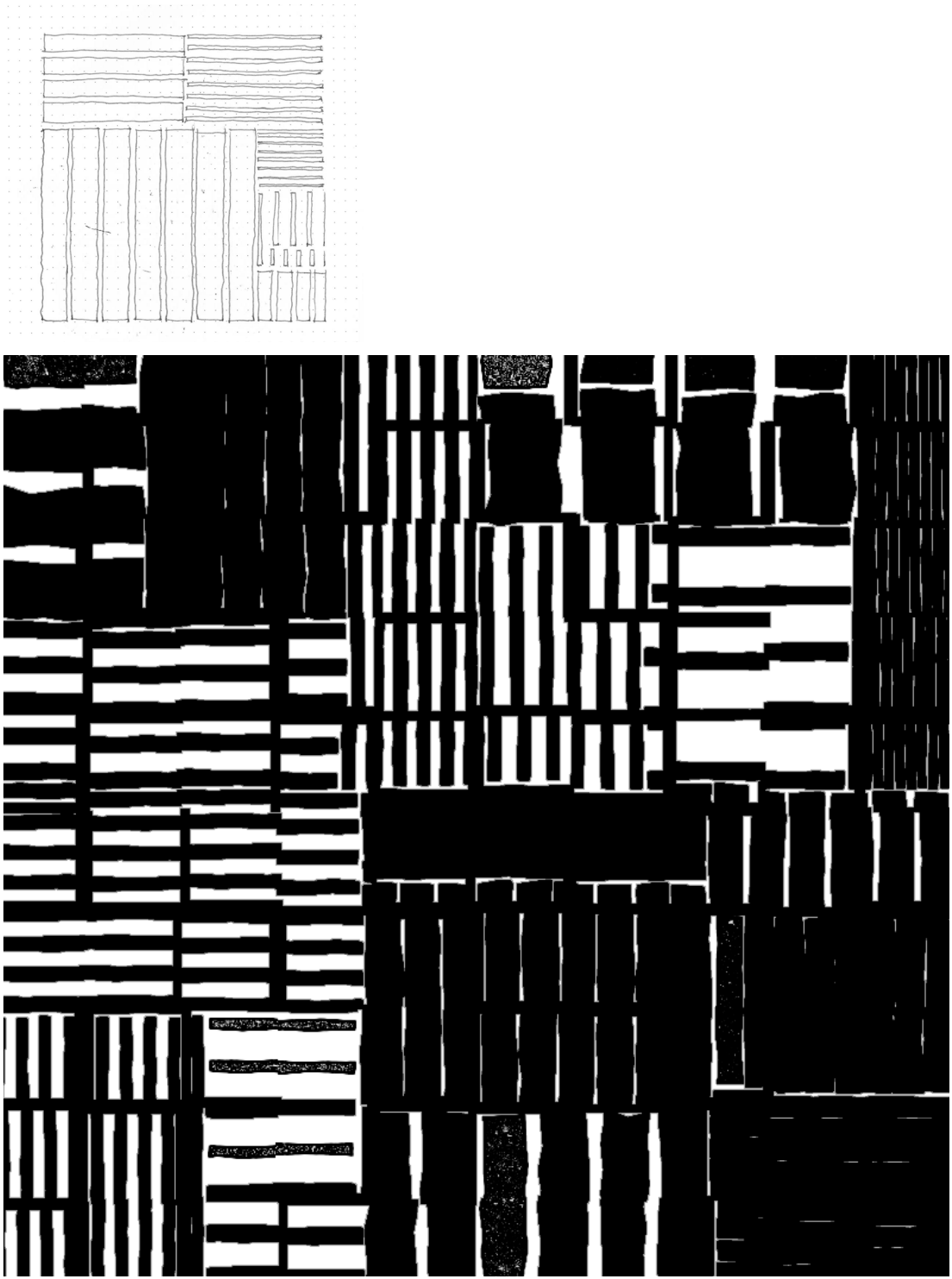


Figure 1.08 Hand drawing and coded output.
Made using Processing, by Author.

The next set of drawings places the mode of creation into the architect or designers' hands. The ability to interact with code from the sketch window without needing to stop the sketch to change variables is built into the sketch. This interaction may be an input from the mouse, keyboard, or even sound.

In the examples pictured beside, I have built in the ability to communicate with the code using the mouse and keyboard. For these explorations, I have built in the ability to pause, play and alter the overall look of the sketch in multiple ways. The ability to rotate around the model in three-dimensional space to see the reverse side of the model can be enough to spark the creativity of the architect. Even one small piece of the produced artefact can be taken away and further developed. Allowing the designer to interact with the program in an intuitive way, much like the software packages we use in practice, allows for the use of this tool by anyone in the industry.

The strength of interactivity comes from humans wanting to influence things around us. In the context of this thesis, this would be considered 'interaction design'. Interaction Design is about making the connection between a device, its interface and the user (Good Interaction Design = Good UX, 2021). It facilitates actions that we might want to make to a given system.

If a designer wants to change the look of a line either by using a thicker pen or changing a line weight variable, interaction design is what responds. It creates a framework which is learnable and intuitive for the user. It delivers that "human element" that makes technology enjoyable and pleasant to interact with (Good Interaction Design = Good UX, 2021).

A conductor in front of their orchestra is leading the tempo, signifying the beginning of a new bar, and prompting an instrument to enter and when to exit the piece; small changes to their movements will alter the way the music sounds to the audience. This musical analogy is productive in relation to this thesis. Creating alongside another distant entity (the computer) symbiotic to the overall piece produced resembles a musical performance. For example, if the trombone in a brass band were played louder or if the musician were to introduce another note into the harmony, the overall musical piece would be altered resulting in a new song. Similarly, the script produces different variations of a drawing with different input variables.

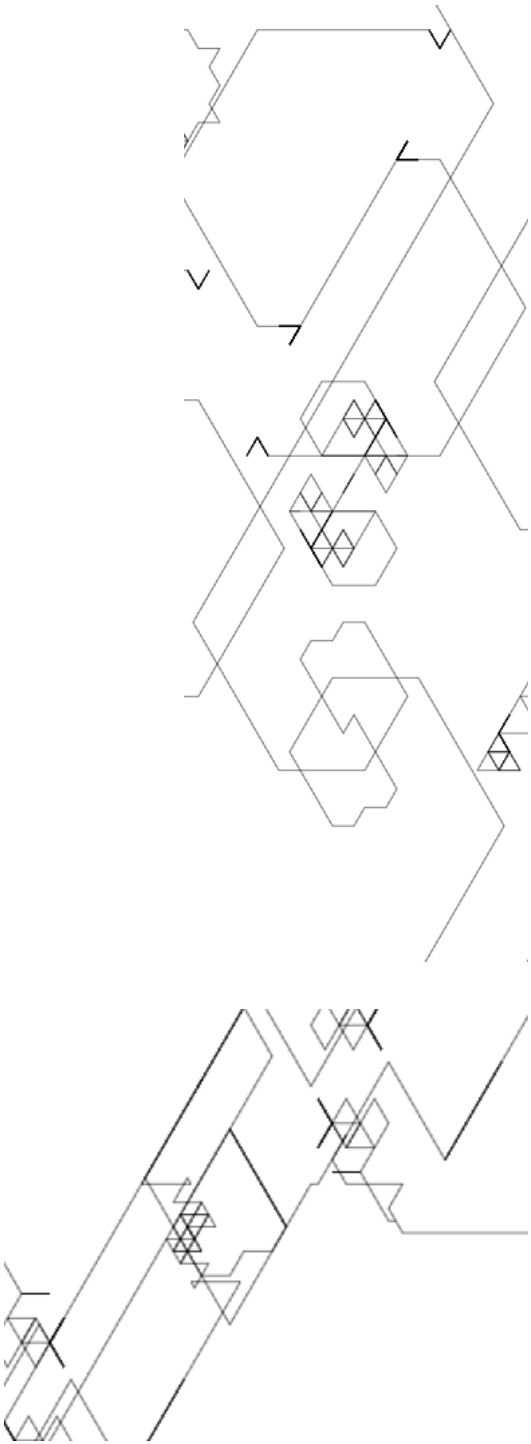


Figure 1.09 Above
Communication with code in three dimensional space.
Made using Processing, by Author.

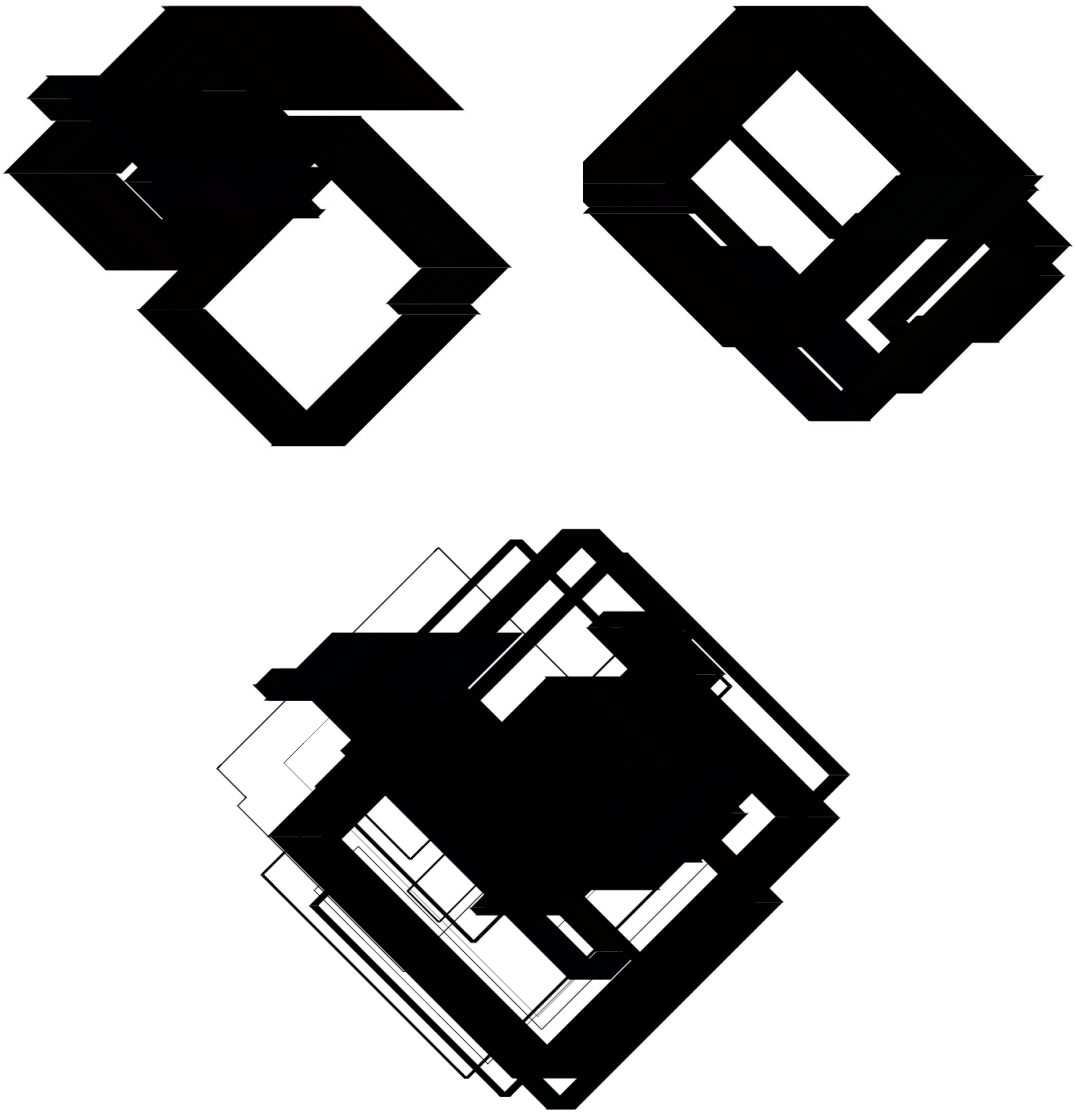


Figure 1.10 Above
Looping lines forming filled squares rotating in three dimsneional space. Made using Processing, by Author.

“The most important part of art is innovation. This means you do something which was not there before.” Manfred Mohr (2016)

1.0
Reflection

Processing will be the program of choice moving forward throughout the design stages in this thesis. This is because of its simple to use interface and developability. That is, the ease in which limitless data types can be entered within the program. The possibilities to create freely are opened by the ability to input video, audio, string data, infrared/ point clouding (Kinect), and mathematical data to be visualised by projection, CSV file format, Arduino, and/or pen plotter. Processing caters to a wide range of possibilities for this thesis, with few bounds for creativity. As mentioned in the last example of work, Interactivity within Processing can be built in, whether it be simply by the use of the mouse and or keyboard.

The strengths of interactivity include engagement of the user, greater diversity of outcomes with minimal input from the user, and increased satisfaction of the user creating. Creating these rules for a designer to use and alter the way the script looked and acted was a satisfying task. The ability to diversify the outcomes by a greater amount each time may prompt use of the program for a greater length of time. In turn, this may leave the user more satisfied with the outcomes created together with the computer.

A major limitation of the exemplified Processing techniques is the inability to record and save conceptual designs that may be opened and edited at a later date. On several occasions, the drawings I produced were not 'recorded' and therefore not saved. This led to the loss of potential design iterations. The inability to rewind meant I was unable to reproduce the lost output.

Drawing with a pencil and paper does not have this same limitation. A drawn line creates a visual marking on the paper. Further, an indent remains even if the pencil marking is erased. In practice, layering design options one-on-top-of-the-other (often with butter paper) allows an architect to draw freely and use inspiration from previous drawings earlier in the layers.

With respect to code and visual outputs on a screen, the longevity of design is limited by the user closing the window or the computer beginning a new set of iterations.

Moving forward, I will explore this in a manner that resembles drawing and layering up in the context of the computer screen.

In the next design stage, Interactivity through processing is further explored in-depth. Particularly, this interactivity makes the design process more engaging for the user, and adds another dimension to the work if the user is able to create for their own within the bounds of the written script.

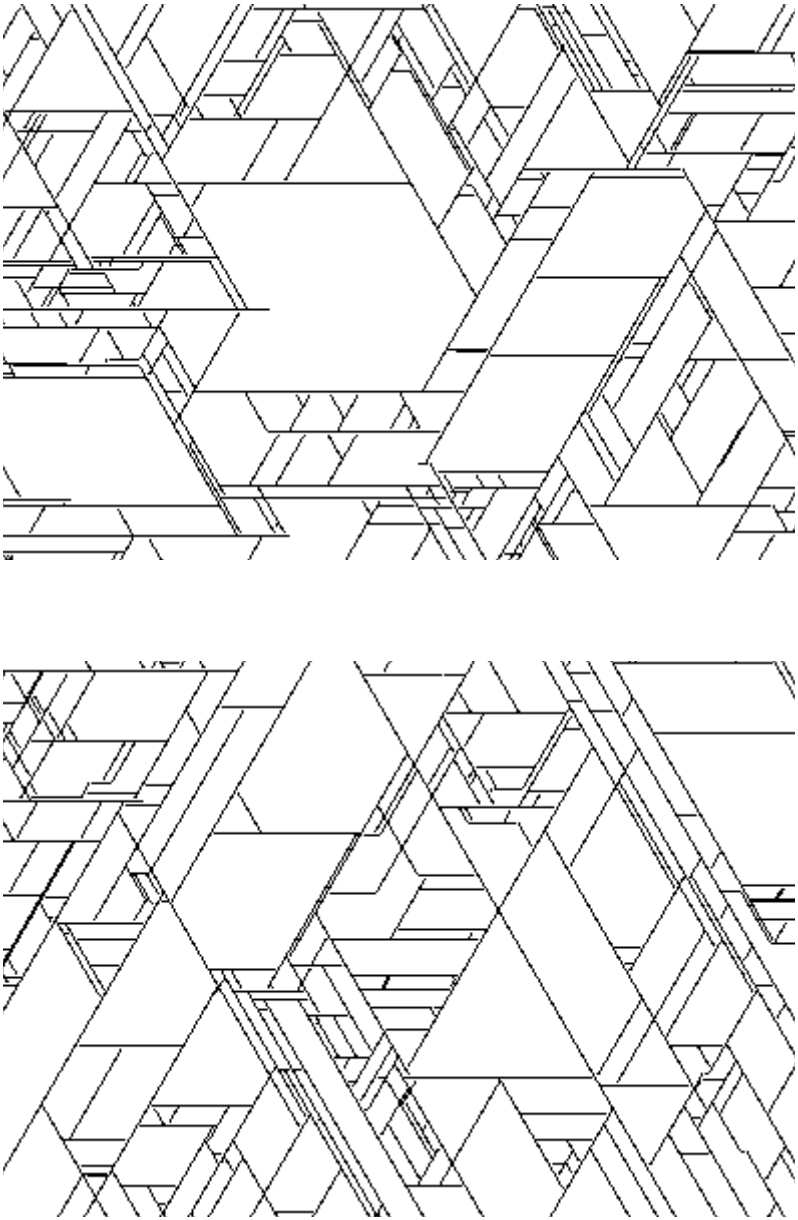


Figure 1.11 Above
Three dimensional planes rotated in sketch window to aquire different views. Made using Processing, by Author.

2.0

Modelling with Processing v1

Following on from drawing in the previous section, I wanted to learn how to create/ generate form and space using Processing. As humans, we view this as three-dimensional objects and space. The drawings in the previous design testing phase were intriguing but lacked three-dimensional form. Until now, I have worked only in two dimensions within the sketch window. Without any research into the modelling side of processing, I was unaware that processing had the ability to manage a three-dimensional object.

Linking back to my initial research question: “how can we use scripting techniques within the conceptual design stages of architectural design to generate and create architectural form”. Creating two-dimensional drawings was a step in the right direction. However, translating these two-dimensional works into three dimensional (3D) works would result in a flat plane. This would hover in the sketch window and would have to be rotated to visualise the reverse face of the drawing, Creating a three-dimensional view of these works requires the use of the ‘Z axis’, in addition to the X and Y axes.

In this section, I will explore further the creation of three-dimensional works. I will begin to explore three-dimensional objects in space, drawing back to spatial relationships that are the basis of all architecture.

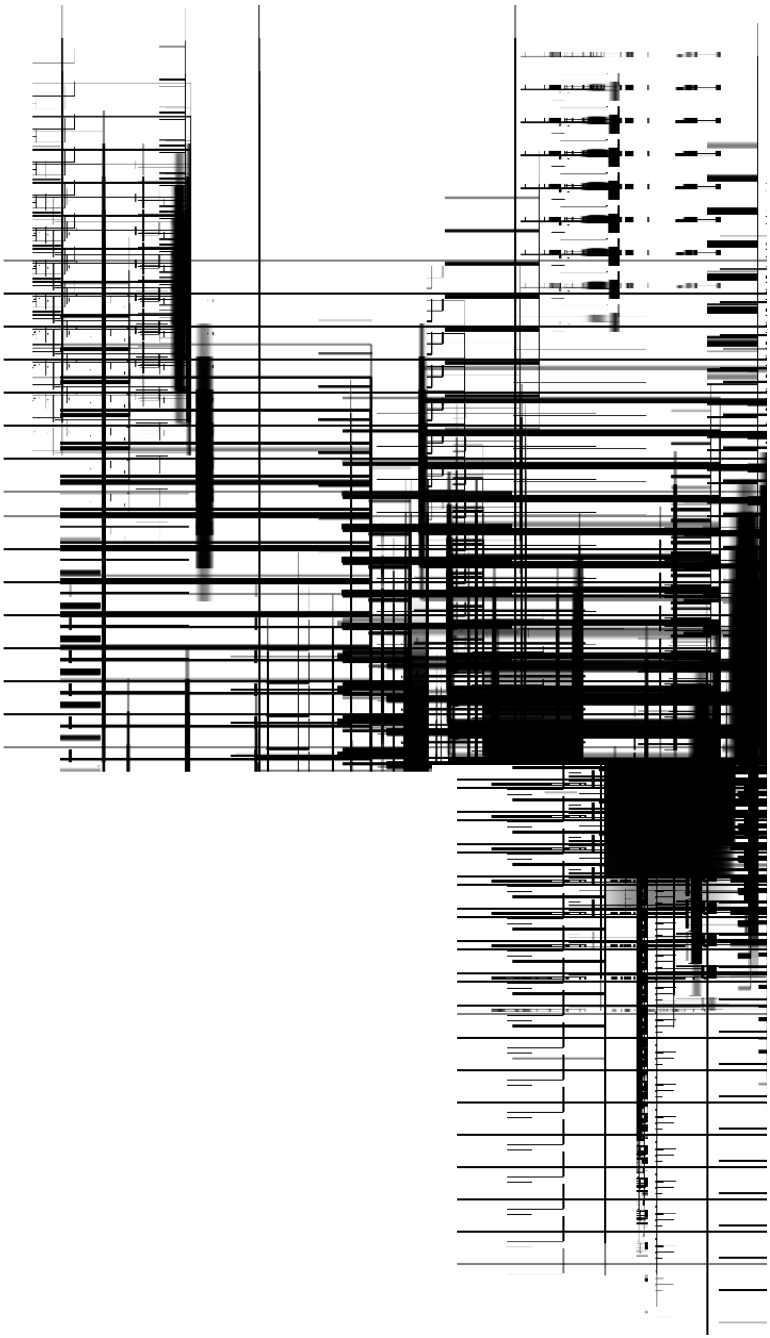


Figure 2.01 Above
Three dimensional planes in space. Made using Processing, by Author.

2.0
Reflection

Building in more interactivity throughout the sketches as discussed resulted in the creation of interesting proactive drawings, and enabled the creative freedom to alter more of the sketches properties within the sketch window. Assigning the number keys to facilitate change during the animation gave freedom to design new outputs. Having built in the ability to produce three-dimensional forms that could be replicated and developed upon an infinite number of times was exciting.

The ability to save the output as a PDF (including its parameter values, hue, saturation and brightness colour values), enabled the user to return to a given iteration for further development. Designing these interactions into the script allowed the program to become more useful as a design visualisation and ideation tool.

Moving forward, I want to investigate how the designer writing and using the code can work collectively with the computer. The idea to work alongside the computer came from watching the computer work through design iterations on its own and myself selecting 'cool' looking options. I thought to myself, I want to be able to watch the computer design and add my own ideas and interpretations into the design. One way of doing this, without altering the look of the overall object, was to "zoom" into the object far enough such that I was able to reside within the artefact, and thus viewing it as a different scale. This scale change is something that architects are playing with all the time. Designing objects to be represented at a larger scale, and some sympathetic objects at another to create design harmony and ultimately beauty.

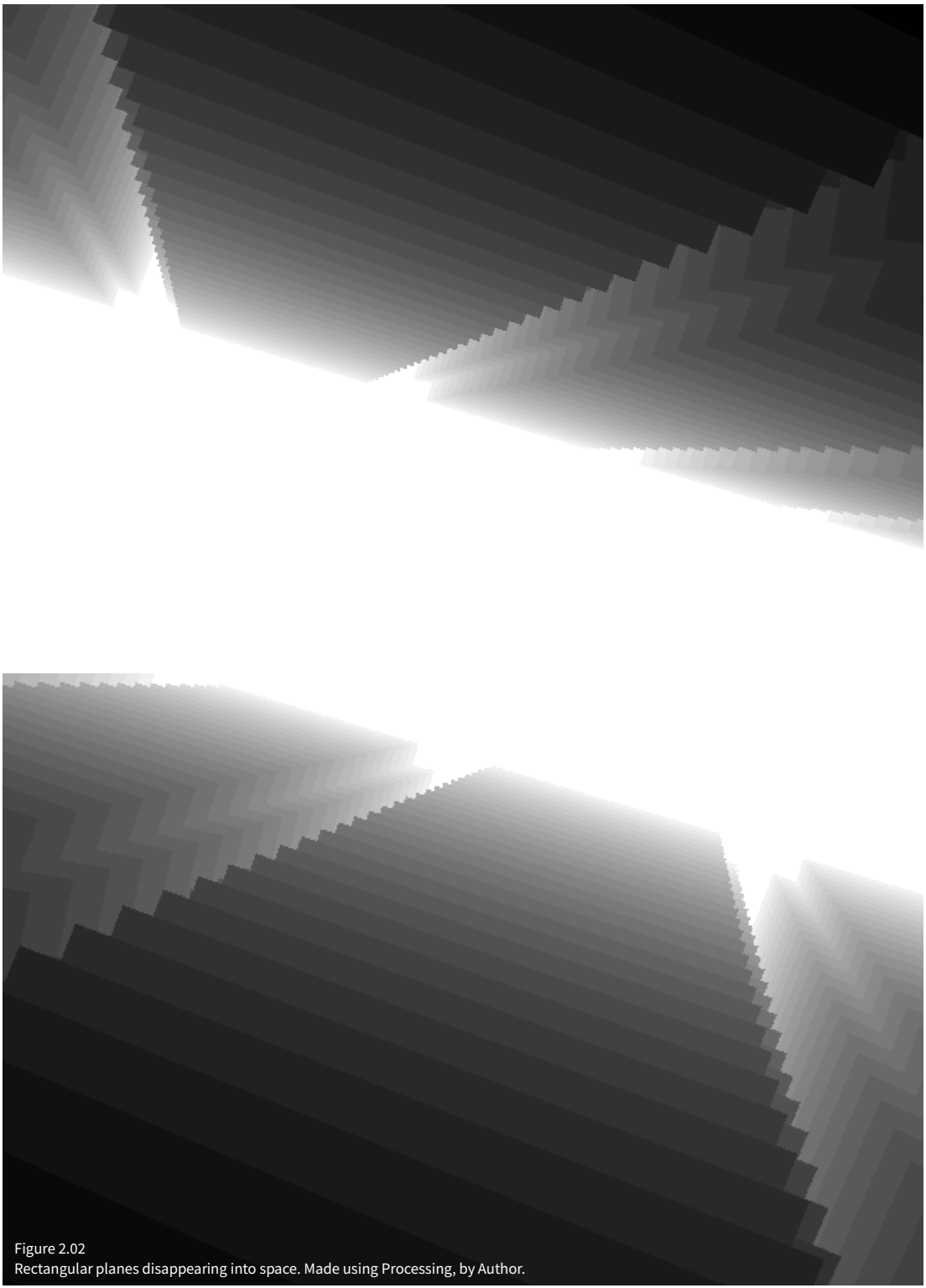


Figure 2.02
Rectangular planes disappearing into space. Made using Processing, by Author.

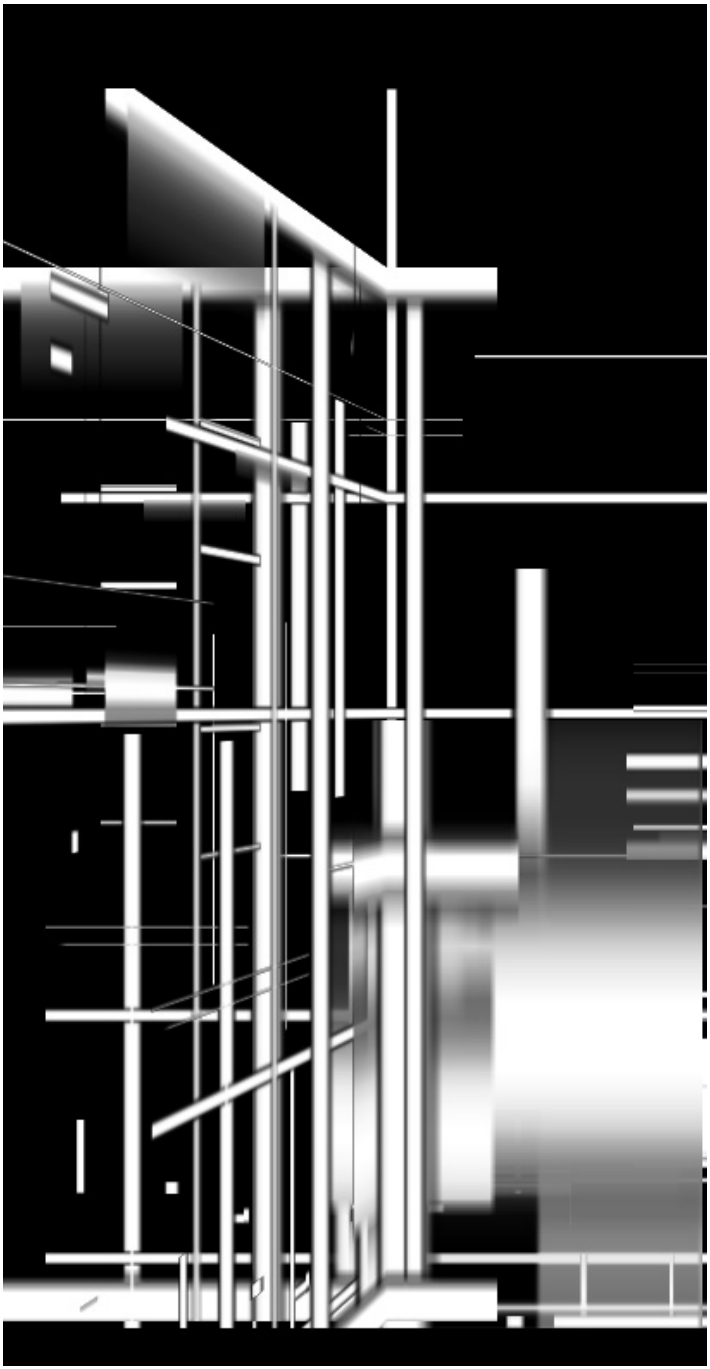


Figure 2.03
Exploring generated three dimensional planes. Made using Processing, by Author.

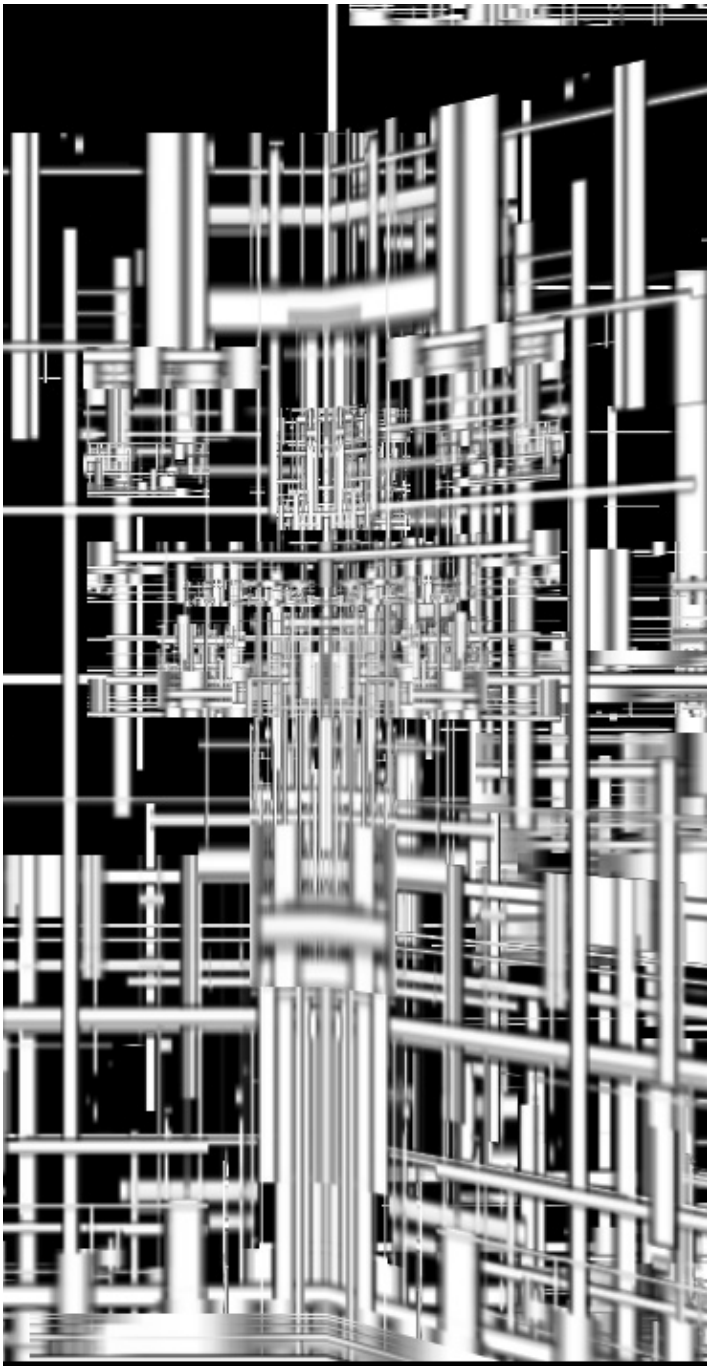


Figure 2.04
Exploring generated three dimensional planes. Made using Processing, by Author.

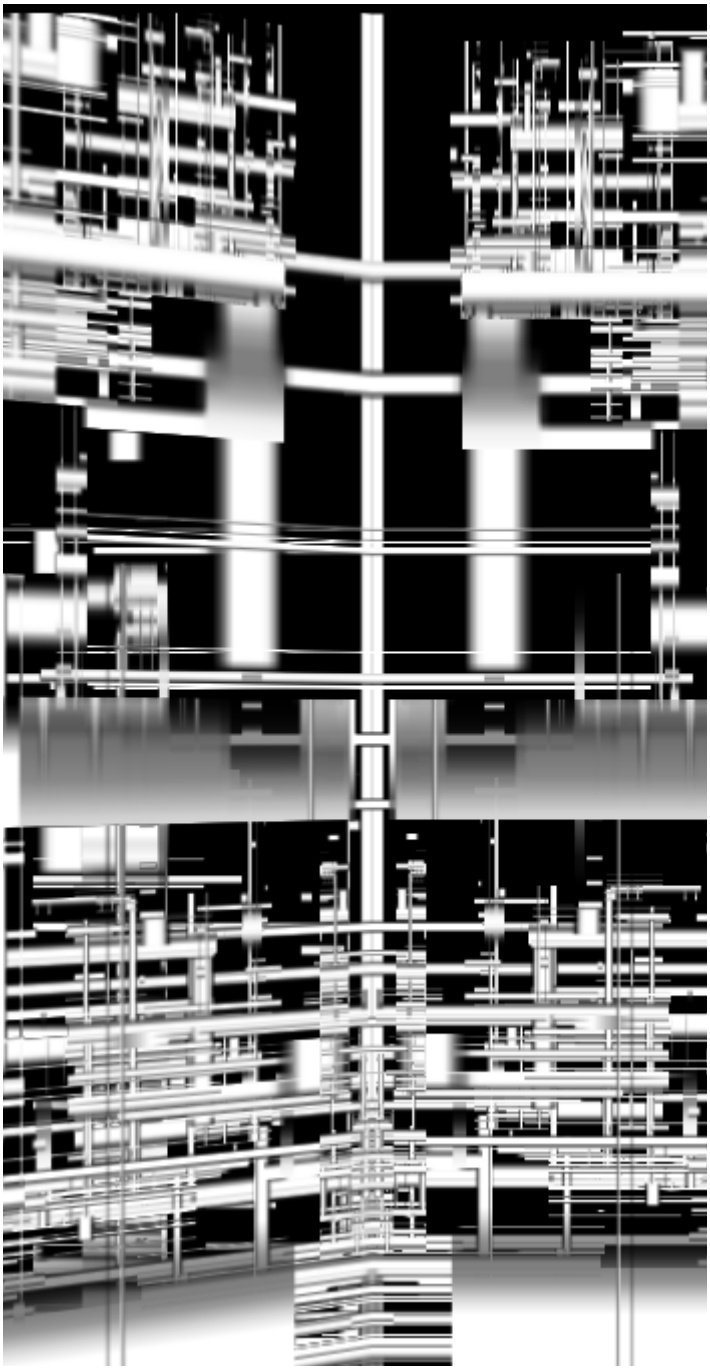


Figure 2.05
Exploring generated three dimensional planes. Made using Processing, by Author.

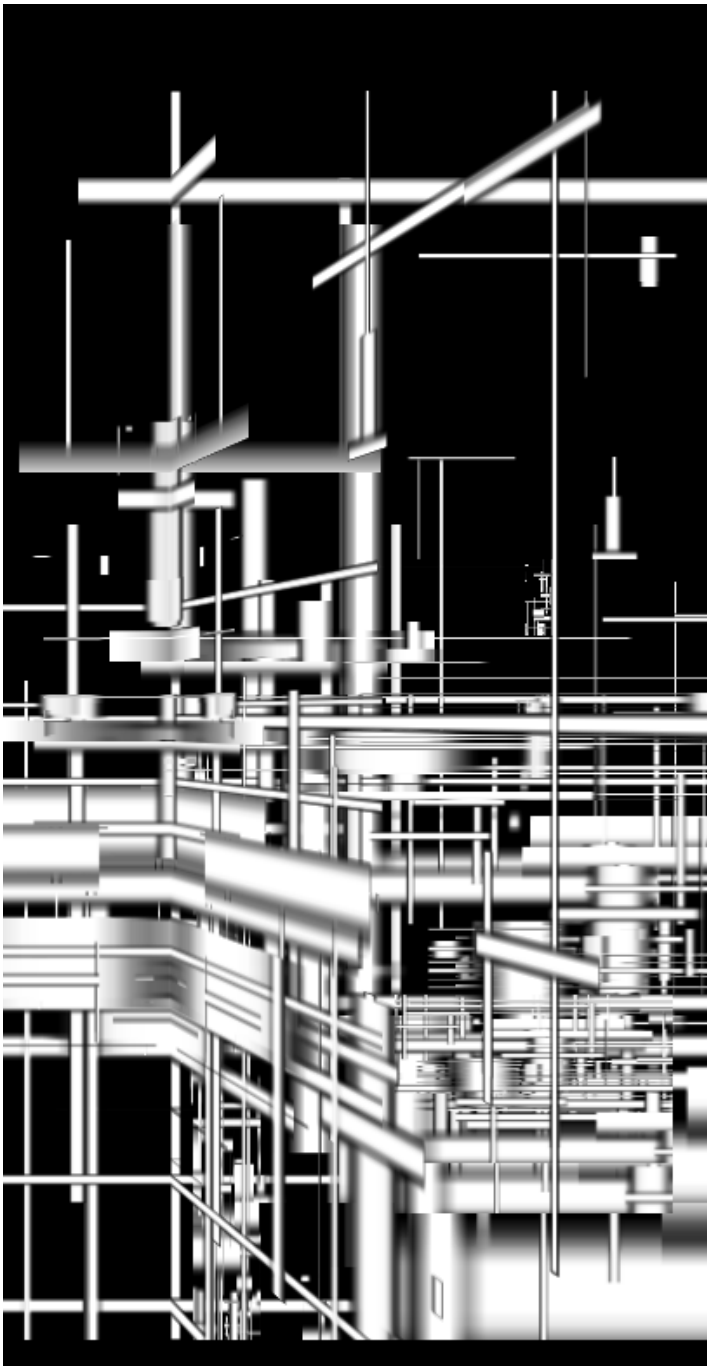


Figure 2.06
Exploring generated three dimensional planes. Made using Processing, by Author.

3.0

Modelling with Processing v2

User interaction within a program, especially involving design, is imperative. When designing, the user ‘wants’ to make it personal, whether that be to change object parameters, or simply change the colour of the paint brush. The user enjoys being in control of the design process.

Experimentation made it evident that even when a drawing or model does not resemble a building, it still may be interpreted as architectural. This was inspiring and drove my thinking away from producing viable ‘architecture’ and to a form that could be read as architectural.

Whilst searching for ways to implement “architectural design-type thinking” into processing, I came across the work of Frederik Vanhoutte, otherwise known in the developing world as W:Blut. Vanhoutte is a physics Ph.D Scholar and works as a medical radiation expert at a university hospital in Belgium. Together with a team of radiation oncologists, physicists, and nurses, Vanhoutte turns medical data into effective treatments for cancer patients. In his spare time, Vanhoutte is a creative coder, whereby he walks the fine line, between art and science, between utility and aesthetics.

I drew inspiration from Vanhoutte (W:Blut) in a number of ways. Firstly, isometric projection is used to explore design possibilities for point, line and plane. Secondly, how animation can be used as a dynamic element within the design process.

After investigating the works of W:Blut, I made contact via email to ask a few questions, and whether he would be happy to share some of his expertise. Specifically, in creating architectural form within processing.

After an initial email conversation, I began to experiment with these concepts. Despite my work diverging from “conventional architecture,” my supervisors and I agreed that the work was interpreted as “architecture” through their form and aesthetic spatial qualities.

For instance, an extruded rectangle to one person might just look like a box on the screen, but to an architect or designer, it could be read as being a volume. This interpretation is interesting because of the subjectivity it presents. To a lay-person, contractor and a designer, these interpretations are all different and subject to personal experience. As mentioned above, an extruded rectangle may look like a box at one scale to one person, but to another it may look like a vestibule or a space to be inhabited at another scale. This scale shift was interesting and something that I will be experimenting with throughout the coming design tests.

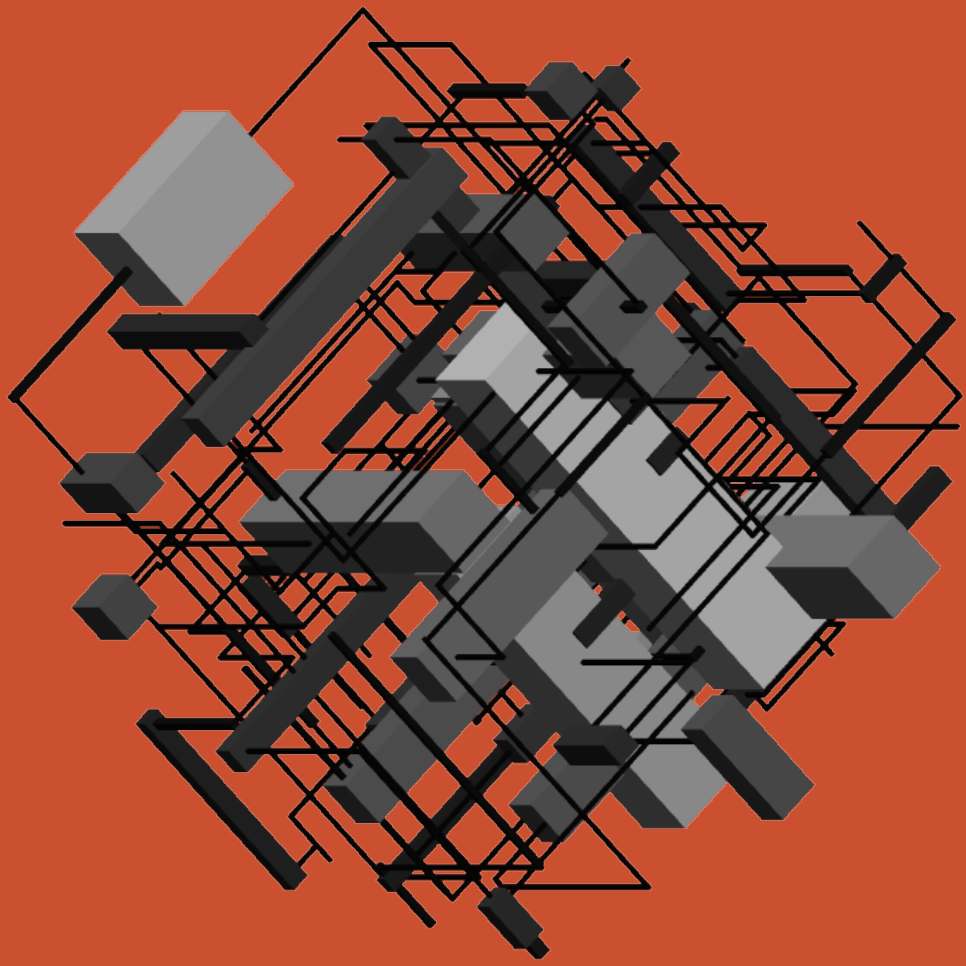


Figure 3.01
Inter-connecting boxes and grid. Made using Processing, by Author.

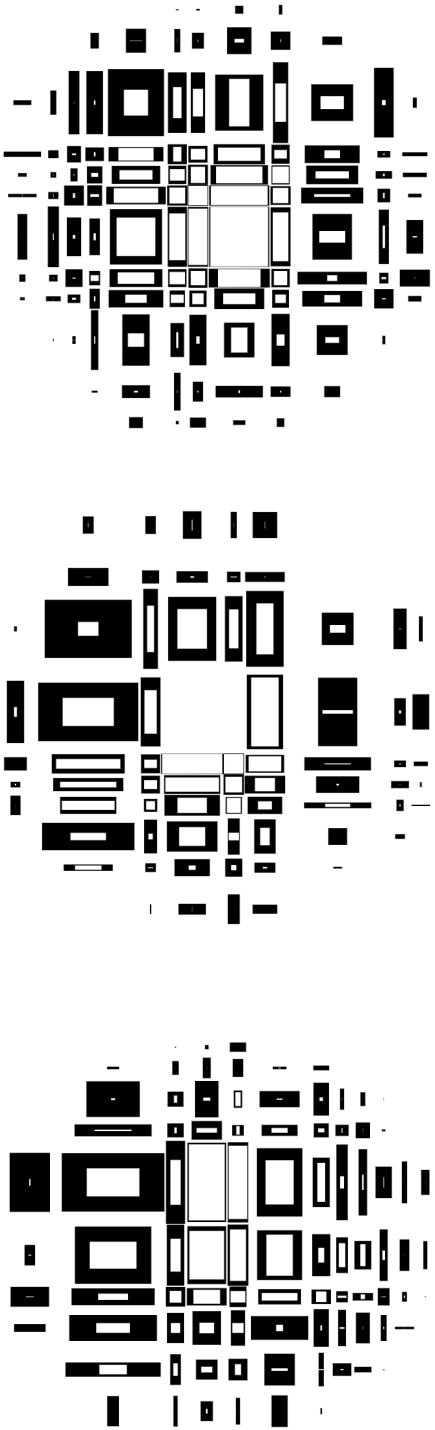


Figure 3.02
Conceptual floor plan iterations. Made using Processing, by Author.

Asking the question, 'What do you think this is?', may have many possible reactions. I think that this makes design and its interpretation exciting and personal. This brings me to the matter of perspective.

Not the perspective of the person, but the position the object is viewed. In most of my design experiments so far, I have opted for the Orthographic projection as a personal preference in the current stage of designing.

Adjusting the zoom and focal length parameters within the sketch window was an excellent way to maximise depth and dimension of drawings in a perspective view.

Using this technique when modelling in three dimensions gives the viewer and modeller a sense of reality, proportion and hierarchy. Perspective is somewhat useless from a construction point of view as the 'proper dimensions' are skewed and not relative to the angle of incidence. When modelling in SketchUp or Rhinoceros for example, the user has the ability to change the view mode to view their design from a different point of view.

Perspective view is excellent for rendering as this gives the designer a sense of the design's reality. As with Orthographic views, these give the designer a sense of scale. Here the scale of surrounding objects is as they were modelled. This is perfect for a cross section or elevational view as it is used to rationalise pieces of a design positioning to each other.

When writing the code for the next design experiments, I will be focussing on experimenting with these view modes. Switching between the two view modes, Perspective and Orthographic, and documenting the differences in interpretation of the object from the computer screen.

Linking back to something that Carl had said to me, is that we "can teach students to see the computer as a colleague" (C. Lostritto, personal communication, 2019). I began to think, how can I make my design process collegial?

This began to be a source of inspiration for this stage of the design.

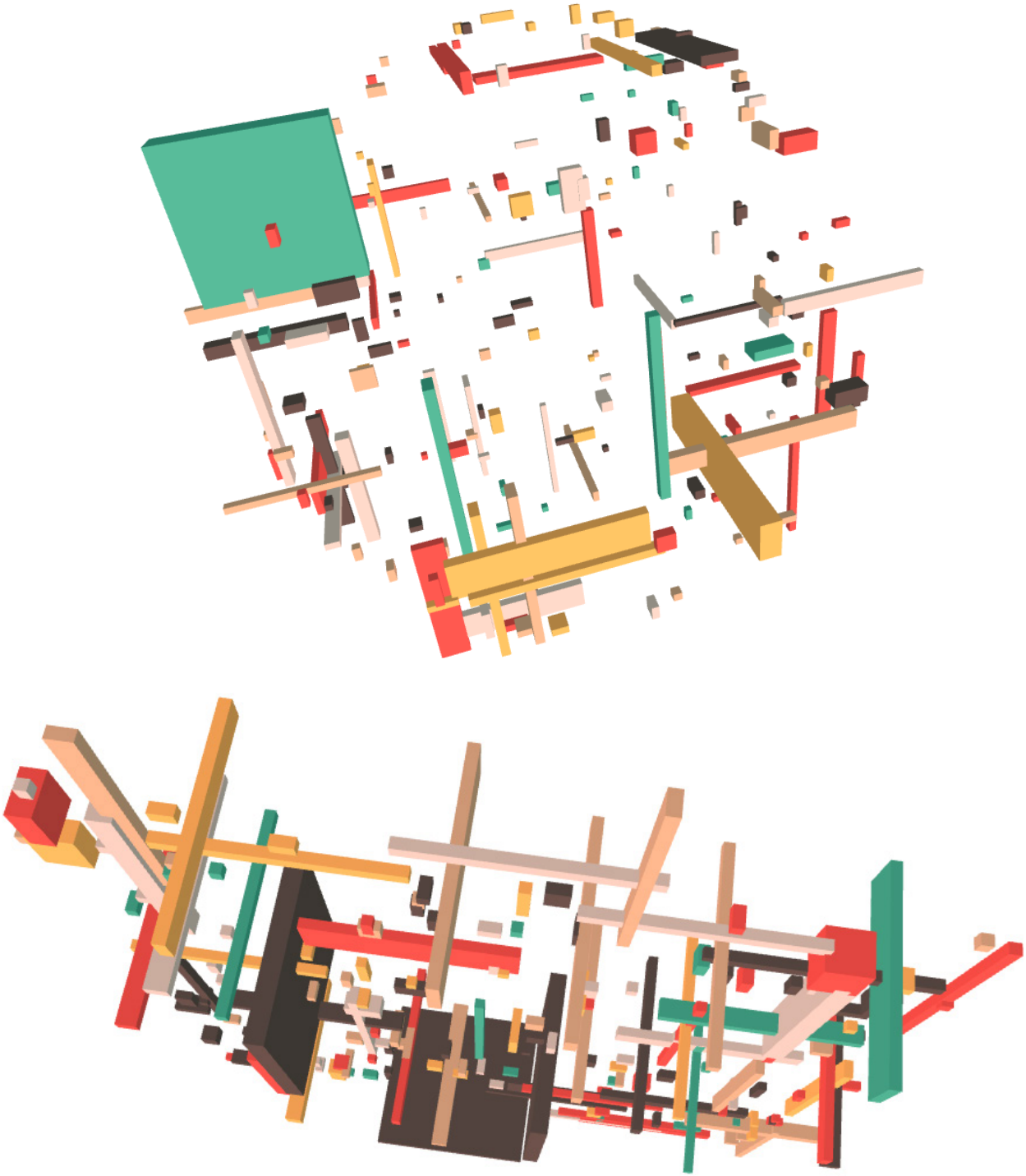


Figure 3.03
Communicating with the computer to alter the arrangement of objects into a plane, sphere, cube and line. Made using Processing, by Author.

4.0
Collegial Design

A colleague is defined as being,

“A person who works with you, a fellow worker”.

(Colleague Definition & Meaning, 2012).

Linking collegiality and how this process is collegial, some further explanation is required.

Within this phase of design, the computer performs a set of instructions outlined by the code. The rules the code has specified are not familiar instructions to a person, but to the computer this is how it understands and executes a set of commands.

Whether it be a simple set of rules to follow, or a complex formula, inherently, there is a conversation occurring. This conversation is not conventional as it is happening behind the scenes.

The conversation between the designer (via the code) and the computer is ongoing. The computer communicates/responds to the designer via a visual representation of the code.

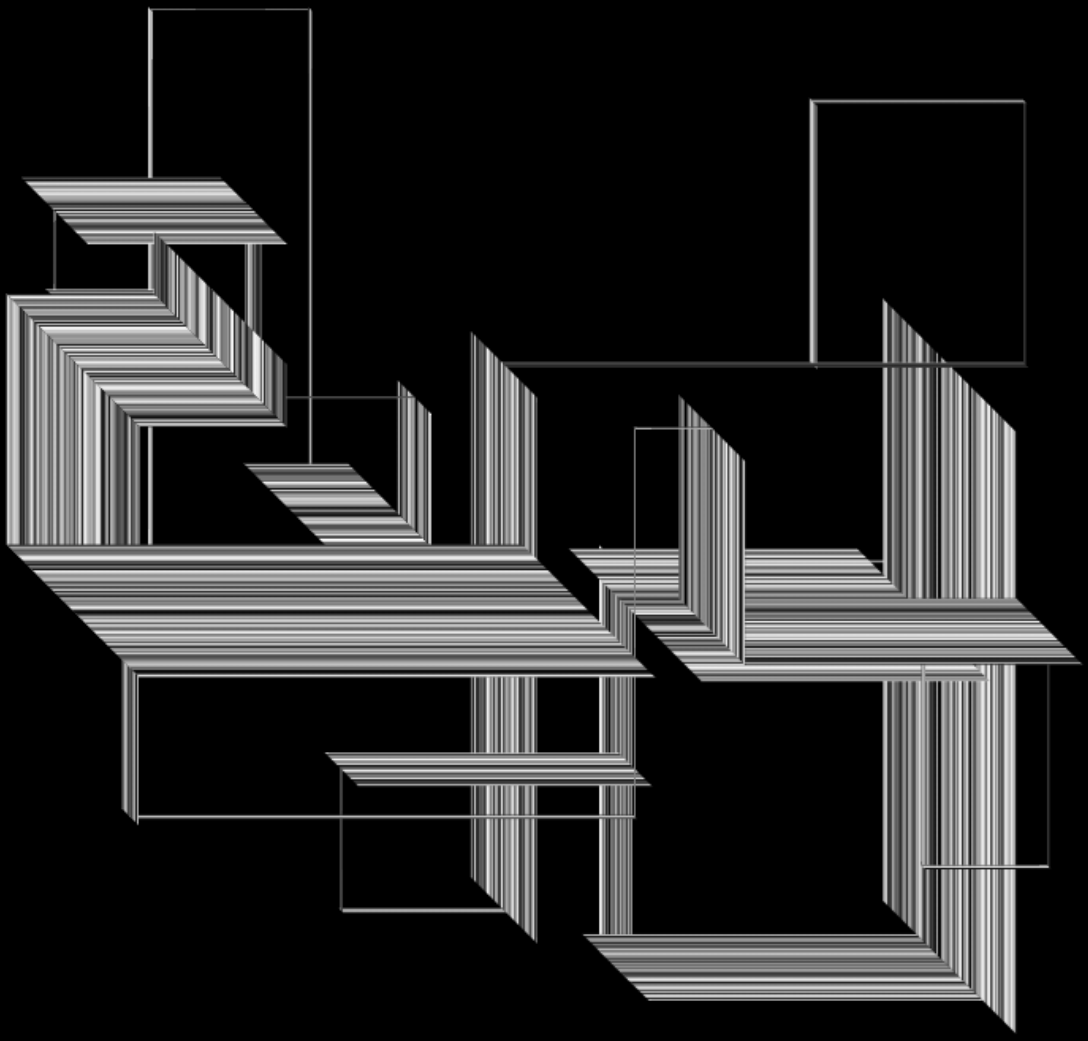


Figure 4.01
Communicating with the computer examining point, line, and plane. Keys on keyboard used to rotate view. Made using Processing, by Author.

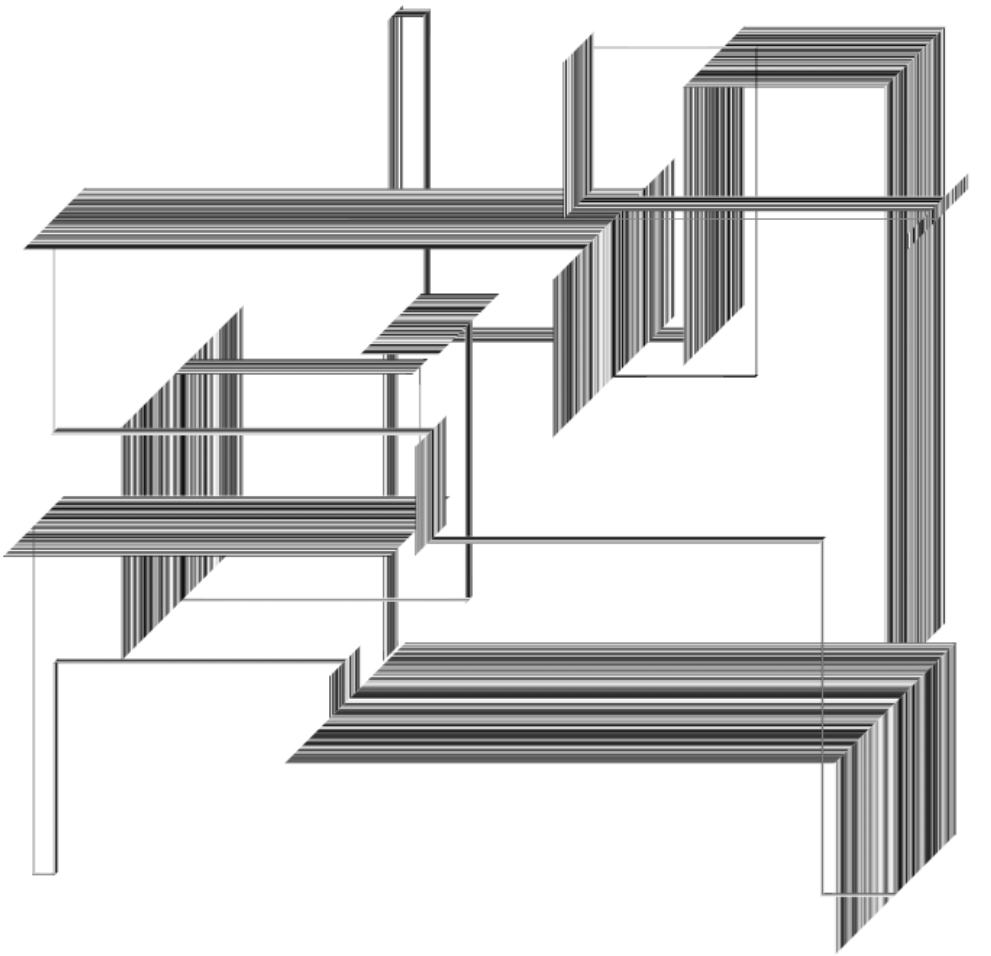


Figure 4.02
Communicating with the computer examining point, line, and plane. Keys on keyboard used to rotate view. Made using Processing, by Author.

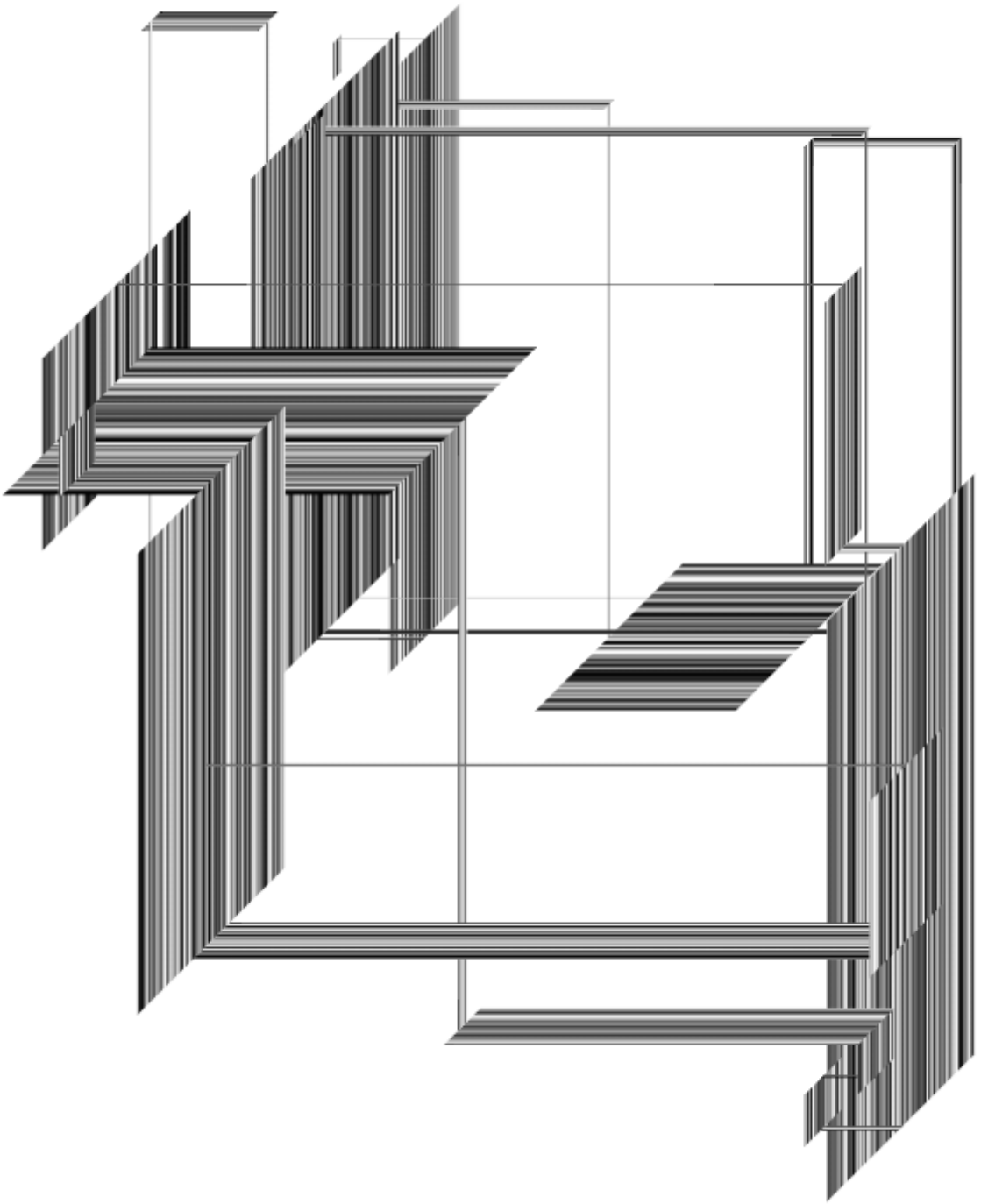


Figure 4.03
Communicating with the computer examining point, line, and plane. Keys on keyboard used to rotate view. Made using Processing, by Author.

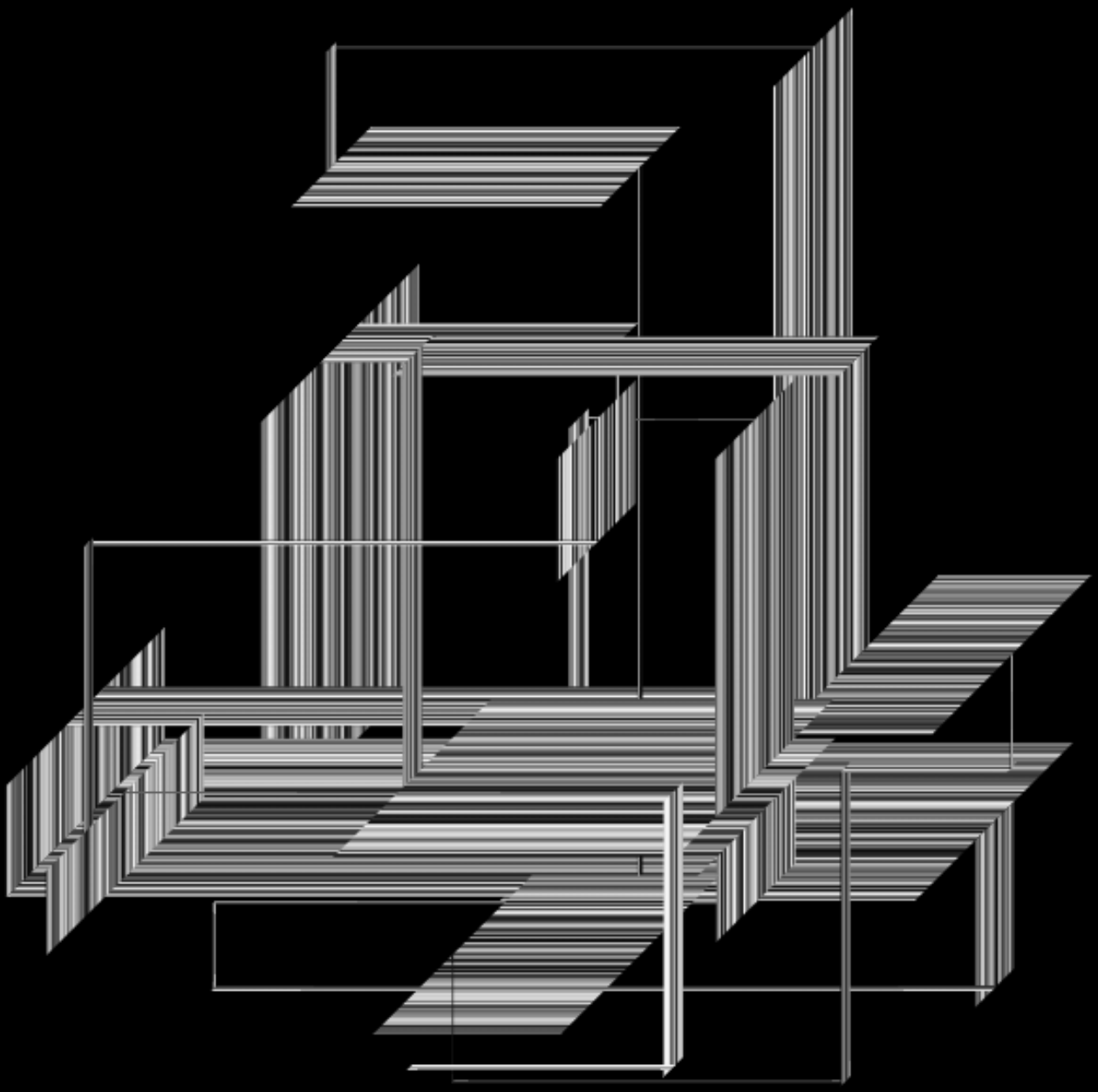


Figure 4.04
Communicating with the computer examining point, line, and plane. Keys on keyboard used to rotate view. Made using Processing, by Author.

Within this phase of design, the computer performs a set of instructions outlined by the code. The rules the code has specified are not familiar instructions to a person, but to the computer this is how it understands and executes a set of commands. Whether it be a simple set of rules to follow, or a complex formula, inherently, there is a conversation occurring. This conversation is not conventional as it is happening behind the scenes. The conversation between the designer (via the code) and the computer is ongoing. The computer communicates/responds to the designer via a visual representation of the code.

Many papers and articles describe the co-operative design process as passing part of the design agency to another entity or group so that they can collectively come to an informed and non biased decision (Bødker et al., 2000). Although this research is productive, my research primarily focuses upon co-operative design with the computer as the entity 'group', not other members of the public concerned with a design outcome. This is where I see this research to be different.

I am seeking to write code from its conception in an iterative and opportunistic way. That is: build a set of instructions that instruct a computer in more of a progressive adaptive manner. For example, instructing the computer in a "do this or this, then this" type way. This gives the computer the opportunity to make its own decisions about what to do next.

Giving the computer agency in the design process and allowing it to make decisions is where this process becomes collegial. The back and forward conversational structure is one that is full of opportunity, interest and provocation. This sits at the core of this design phase and this thesis.

The computer actively produces design options from a combination of variable instructions. While the sketch is running from the Processing window, the animation is essentially building itself, reaching a point and then breaking itself down again. This process of growth and decay is not often seen in other conventional rule based programs.

The initial mass can be altered before the sketch is run by simply changing the lengths of three variables; X, Y, and Z. Working in each of the three dimensions adds complexity to the script, allowing unlimited combinations of input values to form the overall shape.

With an element of interaction built in, the user can inform the computer whether to save this iteration or to restart. This gives both parties shared agency throughout the entirety of the design process. Building this ability to pause an iteration had its positives. Firstly, the ability to save an iteration at a particular point in time during the animation allows the designer to capture a particular moment in time (as discussed in Section 01 Drawing). Secondly, to slow the creative process down in order to reflect and react accordingly. The concept of reflective practice fostered critical thinking, resulting in a different approach to be taken. Schons' model of 'reflection in action' suggests that reflecting on unexpected experiences and conducting experiments serve to generate a new understanding of the experience and change in the situation (Schon, 1983).

A beauty of procedural design is random seed generators. Building this into the programs previously discussed means that every iteration will be different. Each time the program is run, a new 'seed value' is generated, producing a new design iteration. Having the ability to pause and save throughout the animation gives the designer extra control, more particularly, version control. With the code outputting the seed number each time the animation is executed, the designer is able to input the given seed value to achieve a repeatable design. This output variable is a paradox toward random seed generators. This paradox shifts when looking at this design 'helper' from the lens of a design iteration tool. Having the ability to generate the same iteration time and time again removes the 'randomness' from the design process making each iteration special. However, a designer may perceive this as useless as it is not repeatable.

Looking back to Design Phase 1 (“Drawing with Processing”), interactivity built into a program brought active user engagement and a wider diversity of outcomes, with minimal code input for future changes.
In this section, I want to explore the ability for the designer to use their keyboard and mouse to control certain elements within the design process.

- Translation

Translation (where the object is on the screen) of the form - using the mouse position in the sketch window,

- Control

Ability to pause using the spacebar, and rewind and go forward in time using the left and right arrow keys,

- Zoom +/-

Zoom in / out - using the up and down arrow keys and the mouse scroll wheel.

The scripts developed have unlimited variation built into the program with the use of a ‘seed’ value, discussed previously.

A seed is defined as being a number which is used to initialise a pseudorandom number generator (or random number generator). Also to prevent the same number or sequence being generated each time, this seed value must be changed (JavaMathBits, 2021).

For a seed to be used in a pseudorandom number generator, it does not have to be random. For instance, in this program, I have built in a million different seed values, which have an associated 'predetermined' design iteration. The seed is essentially a version control mechanism. It allows the designer to enter a 'seed value' of choice at the beginning of the animation to return a favoured outcome. If this seed value becomes constant within the program, the program will listen and produce the same iteration repeatedly until instructed to choose another random seed from the list via the 'Random' command.

The designer has infinite possibilities for conceptual architectural massing within these programs, making them a powerful tool for design iteration generation.

Using a procedural way of working, the outcome and visualisation of a potential design iteration is also altered with one simple change to the code. By altering the "view mode" within the code from ORTHO to PERSPECTIVE, the spatial relationship and inherent architectural qualities of the output is changed. Moving from the interior of an iteration to the exterior is performed through changing one line of code. For example, changing the zoom parameter value from 0.5 to 3.

While experimenting with these forms, I observed the lack of 3 dimensional spatial qualities. Although the code was generating three-dimensional outputs, the artefact's looked flat. One way I was able to achieve an output that looked more three dimensional was the addition of an atmospheric element. Upon consideration, I realised that this was happening as there were no shadows being cast by the model. For a designer, light and shadow are critical aspects of design, specifically architectural design. To add a sense of reality to the program and thereby replicate the "real world", I needed to add lighting to the scene. Adding light to the scene allowed idiosyncrasies in the design to be highlighted by a bright area of light, followed by a shadow cast on the model. This also added to the spatial qualities of the forms produced.

Pictured beside the same iteration has been produced with the lights turned on versus off. Small details cannot be seen with the lights off. This mode can be used for form finding without the need to dwell on the details. The “lights on” iteration highlights the details and shows the viewer a textural version of the same iteration, giving it depth.

Some may be thinking that for an Architect, the site situates an architecture and that the constraints of the site influence the design. The programs I have created are for conceptual massing studies without the need for a site. This matter was one that I had thought about and attempted to implement into a program previously. In doing so, it took the conceptual idea away from the programs designed thus far.

Many of my peers had concerns that this program and way of working was potentially removing the need for an architect altogether. My answer to this was always, no. The program augments the initial conceptual design phase for an architect. It does not replace a pen and paper. It acts as a guiding tool for ideation and creation. A designer can do whatever they like with the outputs created, whether it be a conceptual massing model for further development, or even just a tool a designer would use in times where there is little to no design inspiration.

Coding can be a creative practice and not merely a tool for graphic representation. It is also a particular design medium with its own affordances and resistances. Using code as a medium for design provides a specific form of feedback that influences the design process and its outcomes as a whole. In other words, code and coding can be attributed agency in architectural design. Students of the Vienna University of Technology (TUW) researched attitudes towards attributed agency and the collaboration of humans and robots. They found an increased level of agency given to the robots throughout their study was associated with negative attitudes towards robots (Zafari & Koeszegi, 2020).

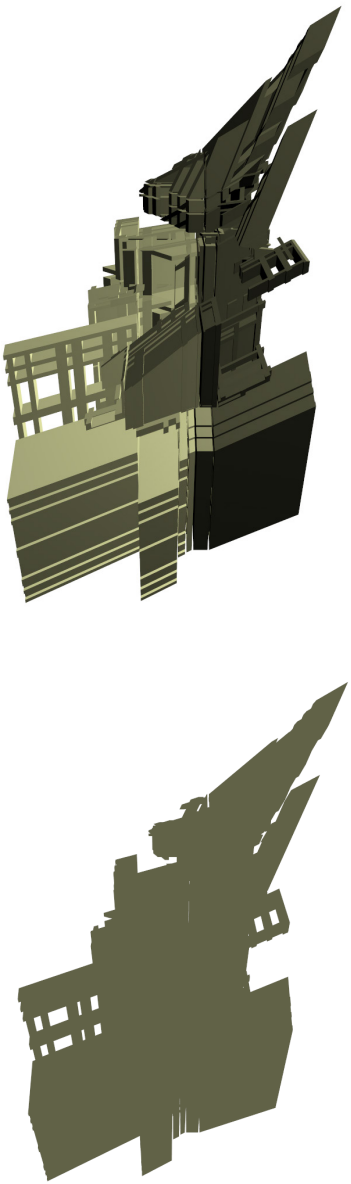
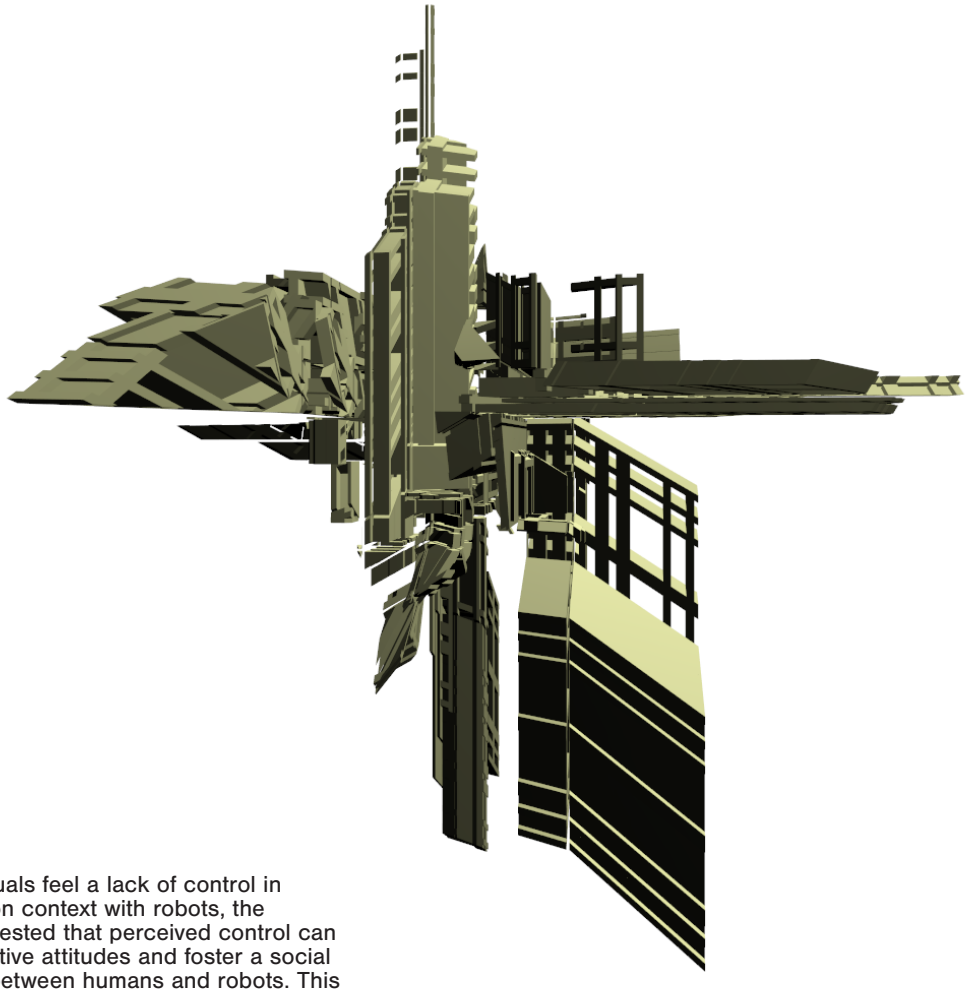


Figure 4.05
The same iteration seed generated with the ‘lights’ on and off.
Made using Processing, by Author.



When individuals feel a lack of control in a collaboration context with robots, the findings suggested that perceived control can mitigate negative attitudes and foster a social relationship between humans and robots. This perceived control of the designer using the script can increase user satisfaction during the creative process by not giving all of the design agency away to the robot, or the computer in this case. The designer is still in complete control of their design helper.

The design process I am proposing is a sketch discussion. A discussion where trial and error is the main driver. During testing when the outputs did not work or look the way I had envisaged, I adjusted the values and tried again. This process of reflection bears resemblance to the "Schon Model of Reflection" (Schon, 1983).

Figure 4.06
A potential design iteration generated by the computer.
Made using Processing, by Author.

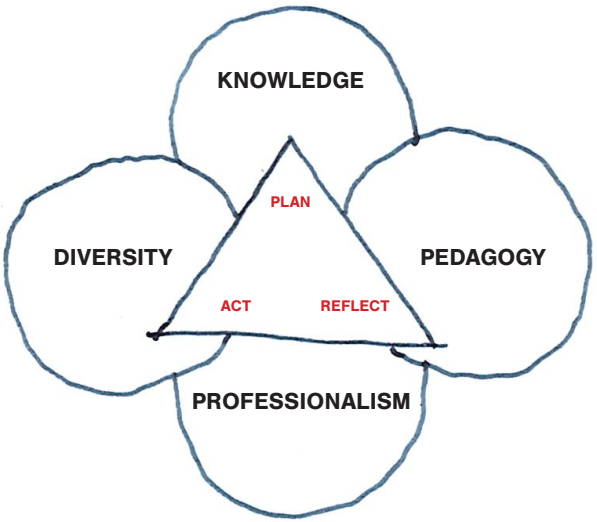


Figure 4.07
Schon's reflective model (Schon, 1983).

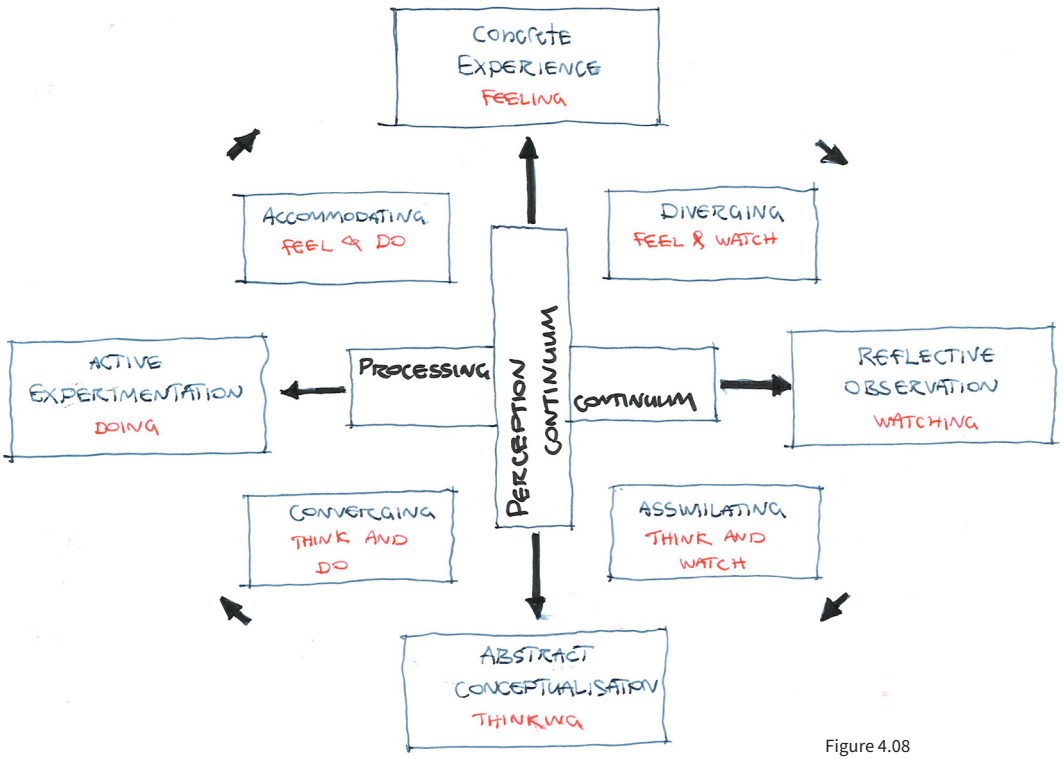


Figure 4.08
A structure of reflection (Schon, 1983), based on Kolb (1984).

Reflection in action, and **reflection on action** were both used throughout the trial and error process of producing viable concept ideas. Experiencing what was occurring on the screen, thinking of what I could do to the code next while it was running and acting upon what needed to be changed immediately.. Thinking about something that has happened, thinking about what I would do differently to the code next time. Taking my time with the code and thinking it through rather than sporadically typing variables. These are all reflections in action (Schon, 1983). Reflecting using these two methods allowed me to progress my codes further and faster, while thinking about what needed to be done to get a particular code working streamlined my design process.

4.0
Reflection

Through the development of the various programs discussed within this section I found that a designer is able to conceptualise and generate architecture using code. Whether individually seen as being an ‘architecture’ or not, an interpretation is needed. Interpretation in design is crucial. An individual is offered a piece of design, and in very few cases, two individuals will experience a piece of design in the same way. I am not going to sit here and say, what I have created and have the ability to generate is ‘architecture’, but what I will say is that what is being generated with each frame that goes by, to me, has architectural interpretation.

This is because I interpret these forms as being architectural, as in designing the code, architectural influence and provocation has been at the forefront. When writing the code, using architectural design language (even though the computer does not know much of the jargon used) for me brings an element of architectural design intent. If I were to use programming jargon throughout the designing of the scripts, inherently, it might remove some of this architectural interpretation.

For someone reading the code, or learning from it, seeing this architectural design terminology may have an effect on their own interpretation of the design outputs. This was one of my goals when writing the code, to inform a layperson, or another designer of the design intent, even before they have viewed what the scripts can create. To me, I think, if I can convey an architectural message within the code, and also within the outputs, I have achieved what I set out to do.

This architectural interpretation also is derived from the different view types used within the output stages of the script. Architects are used to realising their designs in perspective, but often when we draw, we use the orthographic projection to describe and develop scale and proportion. Being an easy shift from perspective to orthographic, it is easy for the designer to alter the way the design is viewed by both designers and also a layperson. A layperson may take a little longer to understand the output if it were only to be viewed in orthographic, as it is somewhat unconventional for someone not in the industry of design to view an object in this way. Personally, I find viewing architecture from an orthographic perspective very informative, in the case of proportion. This view type definitely has a particular look and feel which may not be desirable for a design presentation for instance. An orthographic view is usually situated along each of the object’s axes, or from a set angle above the object. This angle will tend to skew the view of the object and negate a real life view. This is when the perspective view is used, for describing the interaction between different objects at human scale from the perspective of the human.

Taking away from this exploration, the interpretation of an object is defined by the way in which the object is viewed. This understanding comes with personal experience and the interpretation thereof is linked to this experience. Deciding on a particular view type should be dictated by the intended audience and also the intended use.

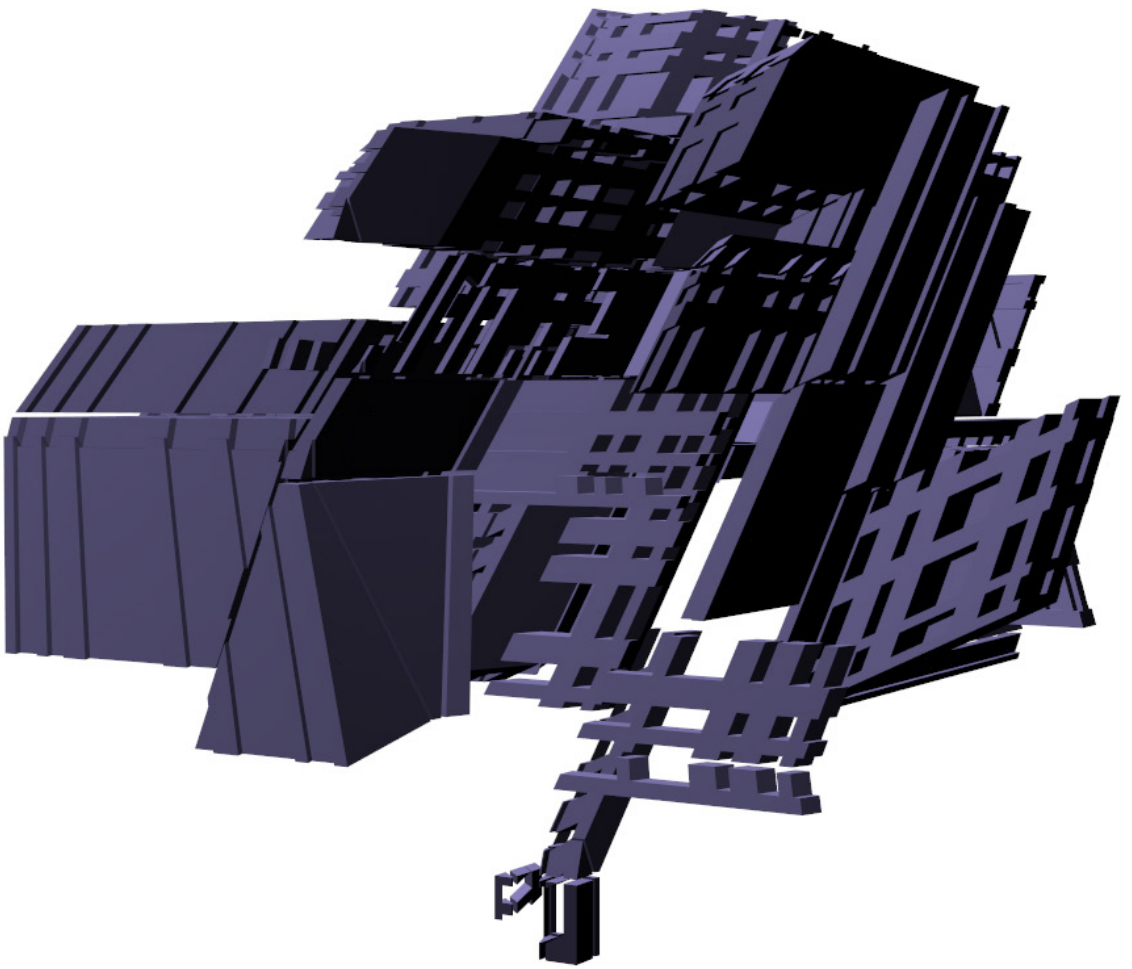


Figure 4.09
A potential design iteration generated by the computer.
Made using Processing, by Author.

5.0
Presentaion

Toward the end of the academic year, and nearing the conclusion of my thesis, I was selected for the New Zealand Institute of Architects Student Design Awards (NZIA Awards). It was an honour to represent Victoria University at the highest level.

After being selected, I had to brainstorm how I wanted to present my work to make it compelling and cohesive to an audience that would be seeing the work for the first time.

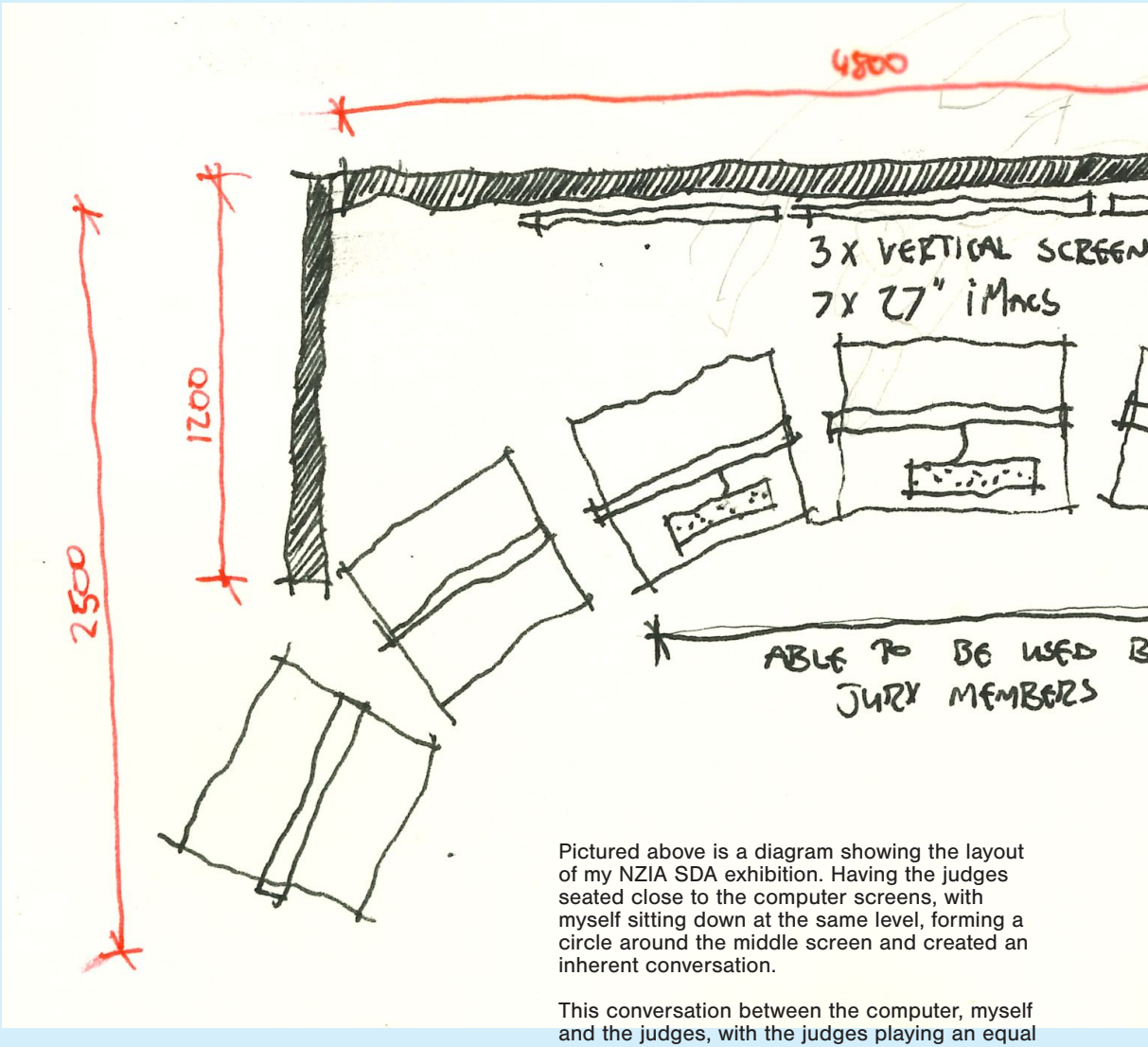
The brainstorm (pictured below) involved mapping out where this journey began right through to where I wanted to end up. This enabled me to piece together the presentation in a sensible fashion, and in a way which is easily understandable from the outset.

Throughout this process I had guidance from my supervisors, and one outside supervisor, Sam Kebbell. Sam was my first point of call for the work I was producing leading up to the awards. Sam gave a unique perspective as (unlike my original supervisors) he had not seen my work develop throughout the year from its conception. This meant he could view my work as would be viewed by the judges and a practicing architect would.

I wanted to design my exhibition presentation as firstly, involving the judges in a specific way. I wanted to make them feel part of the design process. My thesis entailed 'conversing' with the computer as a colleague. So, I wanted the judges to 'sit' around a table as would occur in a design meeting, bounce ideas off one another, and 'talk' to the computer as they would a colleague.

Thinking about how I was going to present my screen based work was an easy one. I did not want to bring my work to the awards not as print outs as this would only capture a moment in their evolution from starting object to final output. Instead, I wanted to bring my work as dynamic elements in their original format. I saw this as an opportunity and the beauty of the work I had created. I knew that if I were to present as purely screenshots, the whole idea of my thesis, creating architectural form together with the computer, would not have been recognised and would have required a far more explanation. The beauty for me was in the movement and the progression of form throughout the animation.

From the raw script presented on the screen at the beginning of the presentation, to the high fidelity and dynamic creation of generating the finished output. I wanted the judges and the entire audience to witness the entire process from beginning to end.

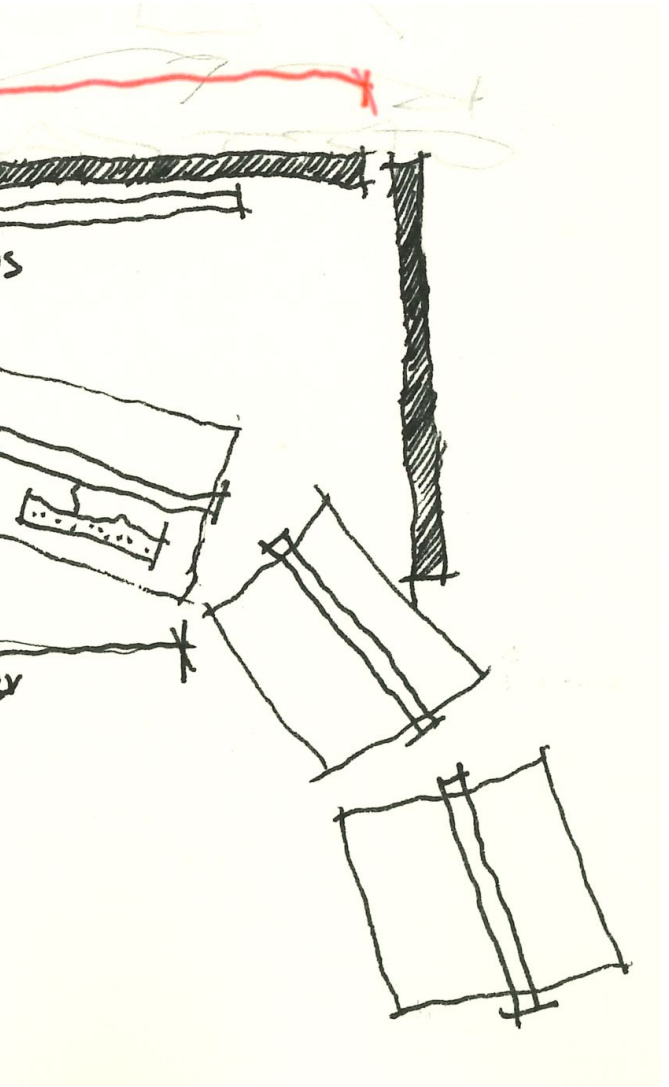


Pictured above is a diagram showing the layout of my NZIA SDA exhibition. Having the judges seated close to the computer screens, with myself sitting down at the same level, forming a circle around the middle screen and created an inherent conversation.

This conversation between the computer, myself and the judges, with the judges playing an equal part throughout the presentation, and an equal part of the exhibition presentation was my key driver when designing my exhibition. The layout of the computer screens also revolved around this mentality. Forming a semicircle of computer screens also allowed this design driver to flow throughout the entire exhibition and surround the judges.

The semicircle of computers was created to follow a similar layout to the judges, and used to focus the attention to the centre of the circle where myself and the judges would all be sitting throughout our conversation. I wanted this circle to feel and act like a round table discussion, with each colleague sitting next to one another around a table, sharing ideas, and designing together.

Figure 5.01
Diagram of my NZIA SDA 2020 exhibition layout.
Hand drawing, by Author.



NZIA Te Kahui Whaihanga
Student Design Awards 2020

Figure 5.02 Photograph of ‘Collegial Computation’ at the NZIA Student Design Awards 2020
David St. George (2020)

Figure 5.03 Photograph of ‘Collegial Computation’ at the NZIA Student Design Awards 2020
David St. George (2020)

NZIA Te Kahui Whaihanga Student Design Awards 2020

The annual Te Kāhui Whaihanga Resene Student Design Awards programme brings together the top four students from each of New Zealand's three schools of architecture: The University of Auckland, Unitec and Victoria University of Wellington.

Each finalist, a fifth-year student, presents their project to a panel of judges. The jury assesses each work and determines the top student of architecture in New Zealand (NZIA, 2020).

Project Description

Collegial Computation, at its core, is a conversation, a Korero. However, this is not a conversation with another human designer, but with a computer as a partner in the conceptual design process. The collegial aspect of this project looks at giving the computer shared agency in its interpretation of the 'unknown' and bringing it into the 'known'.
Coding can be a creative practice within architecture, and this project argues that coding is not a mere tool for designing or graphic representation, but a particular design medium with its own affordances and resistances. Using code as a design medium provides a specific form of feedback which influences the design process and its outcomes as a whole. Code is technological and conceptual support for design thinking and generation. In other words, code and coding can have agency in architectural design and its processes.
This research is based on a number of cases from design practice and theory, from small personal design experiments through to developing a software tool.

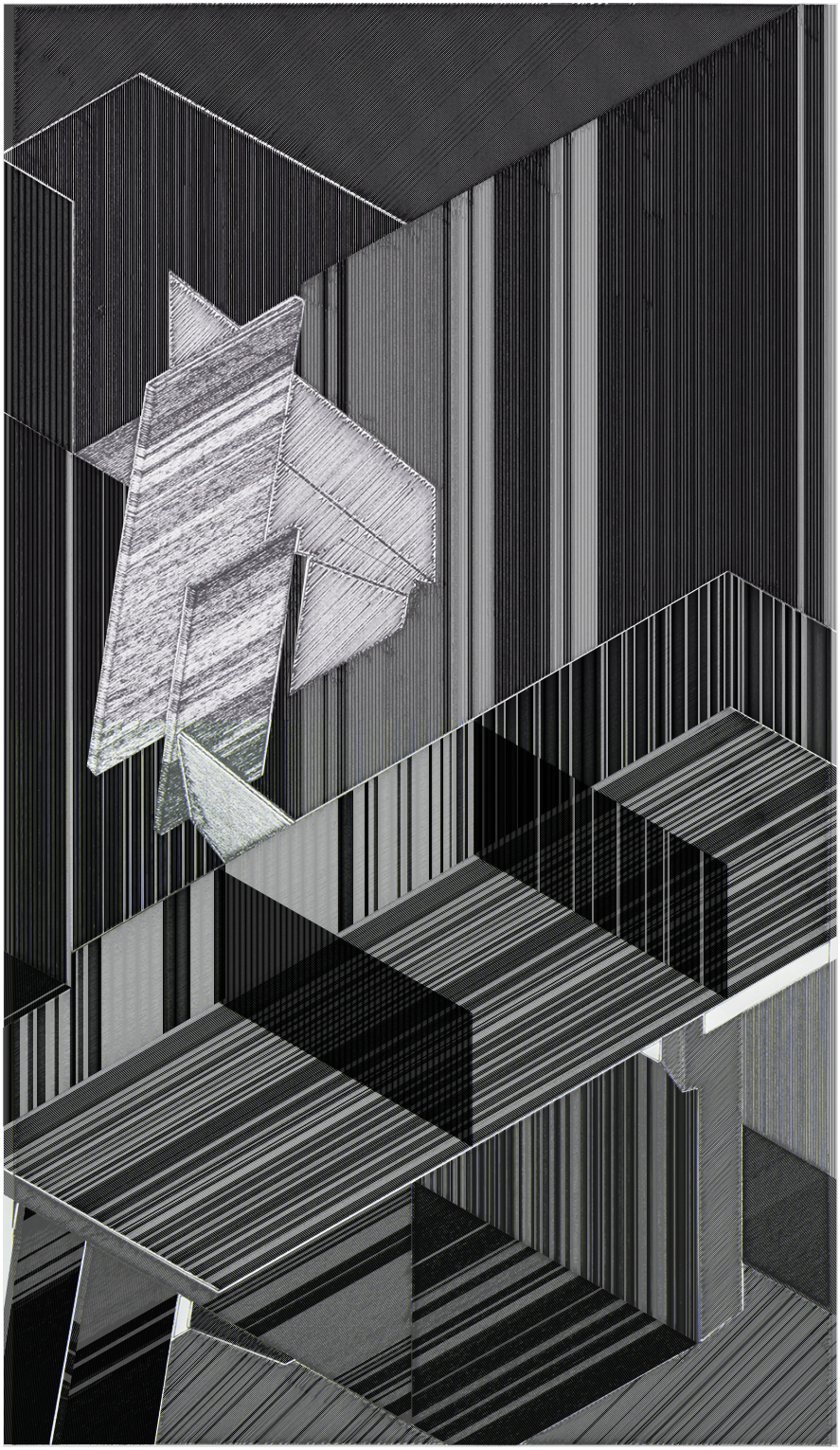


Figure 5.04 Spaces 02. A print generated in Processing, and post edited in Adobe Illustrator, by Author.

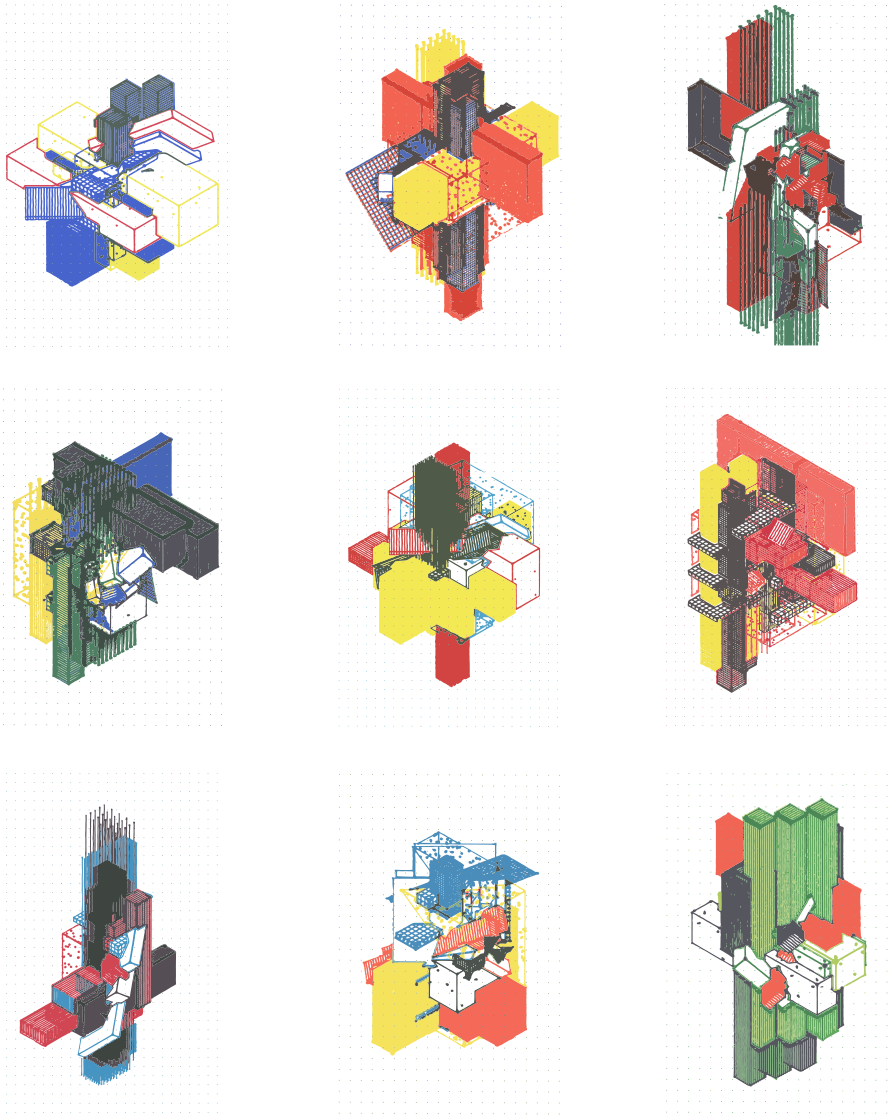


Figure 5.05
Structures 00. A series of design iterations generated in Processing, and post edited in Adobe Illustrator, by Author

Judges Citation

A bold, unconventional vision that reconsiders the computer / human interface in the design process and stretches the idea of collaboration into the digital realm. This is a brave proposal, one that voluntarily relinquishes key parts of the designer's role and instead vests agency into a non-human 'colleague'. Using code, Jordan has succeeded in generating forms which were compositionally and spatially compelling with limited or no active input. Although the designer's role was limited by the nature of the process, his design sensibilities were still readily apparent in the computer-generated forms. The protocol developed has a wide range of applications beyond the field of architecture.

Tanes' Citation

A brave and future-facing body of work that invites us to consider the relationship between designing and computation. The notion posited by Jordan sees 'Colegial-Computation', not as a series of blunt tools, or simple mechanisms to enact orders, but as a rich process of negotiation and discussion. This is a brave proposal, one that rightly acknowledges the role of non-human creativity, and vests agency into how the human colleague can play a larger role in the design. Using code, Jordan maps the new landscape where a more cooperative relationship between creative identities can co-inhabit. Because the designer's role and engagement with digital tools were extended a range of compelling design solutions were arrived at. The protocol developed has a wide range of applications beyond the field of architecture.

Well done, Jordan a genuinely compelling new way of creating.

5.0
Reflection

Upon reflection of the NZIA Student Design awards 2020, and returning to Wellington to complete the remainder of this thesis document got me thinking. This project was very different to the rest of the projects that had been exhibited. It was definitely exciting and motivating. While I was at the awards, many people had approached me and had said, “Wow, I have never seen architecture being portrayed, represented, and created in this way before”. I think this is what made my project stand out at the awards. It was provocative, interesting, and unique.

The judges wrote in their citation that my project “voluntarily relinquishes key parts of the designers role and instead vests agency into a non-human colleague”. In fact I see this being on the contrary. The designer still plays a central role in the design process, even though the computer is generating concepts for the designer. The computer will only give you as much as you give it. Although the designer is able to generate hundreds of designs in a fraction of the time, the design process still remains within the designer’s hands. The computer is only there to listen to the designer and aid them in the creation of a concept. Once a ‘final’ concept has been reached, the designer can do what they like with the output. The output may only serve as a massing exploration, exploring the various ways an architecture can inhabit a specific site, or how the designer could use shadow to invoke a certain feeling within a space.

Sometimes I think that change, especially in the world of architecture, is daunting and slow moving. But with a tool like this, having a digital colleague by your side during the design process would benefit architecture for the better. To use this process within a design and architecture firm would take some convincing and some serious work, but I can see this way of designing becoming an industry standard in the years to come. It may not be using Processing by that time, and it may well be mass produced, but I think using a similar methodology to the one that I have posed throughout this thesis is something to think about.

The forms and spaces created and designed throughout this thesis are subjective to my own design sensibility. I like to call this ‘our’ design sensibility, as it was not only me who created the forms to look the way they do, the code, my colleague also input their own design flair on top. I, like the judges, can see this protocol for designing being used beyond the field of architecture as a learning and teaching tool, even an engineering tool. The possibilities are endless.

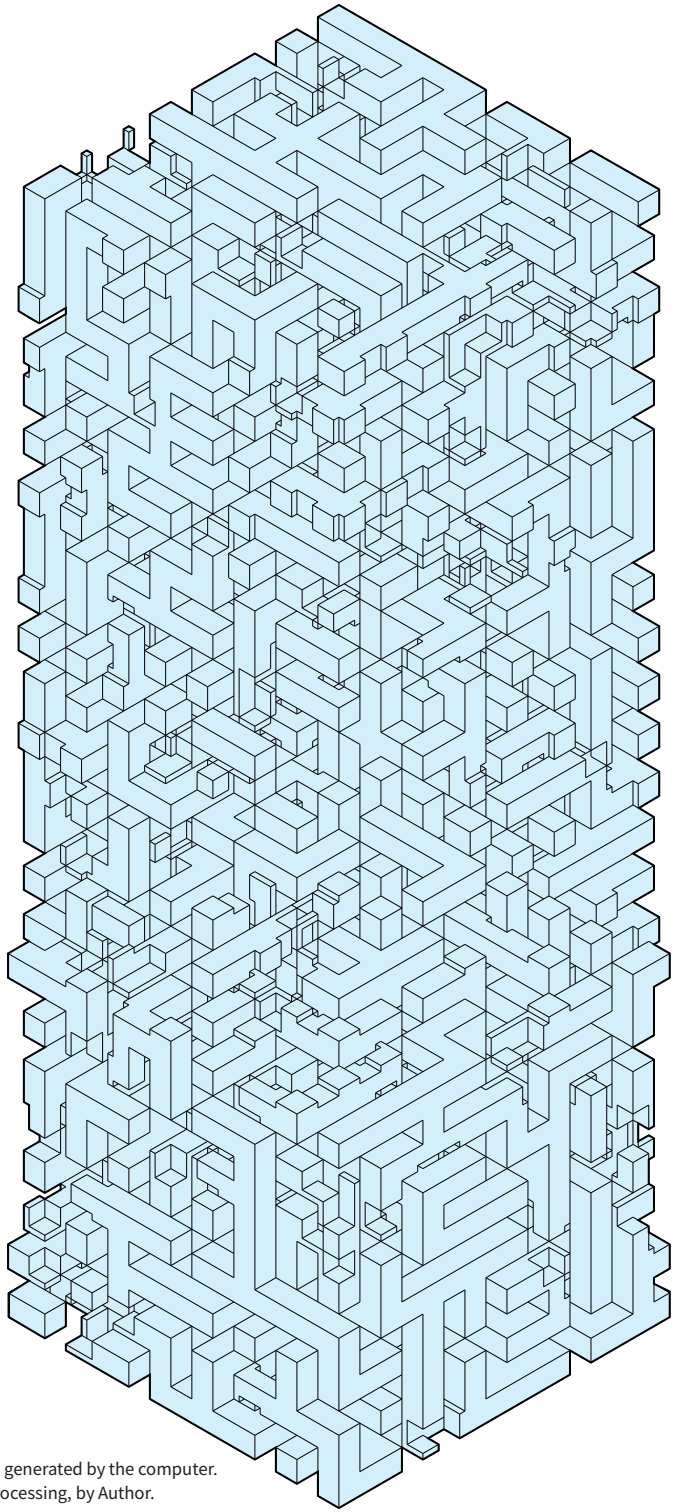
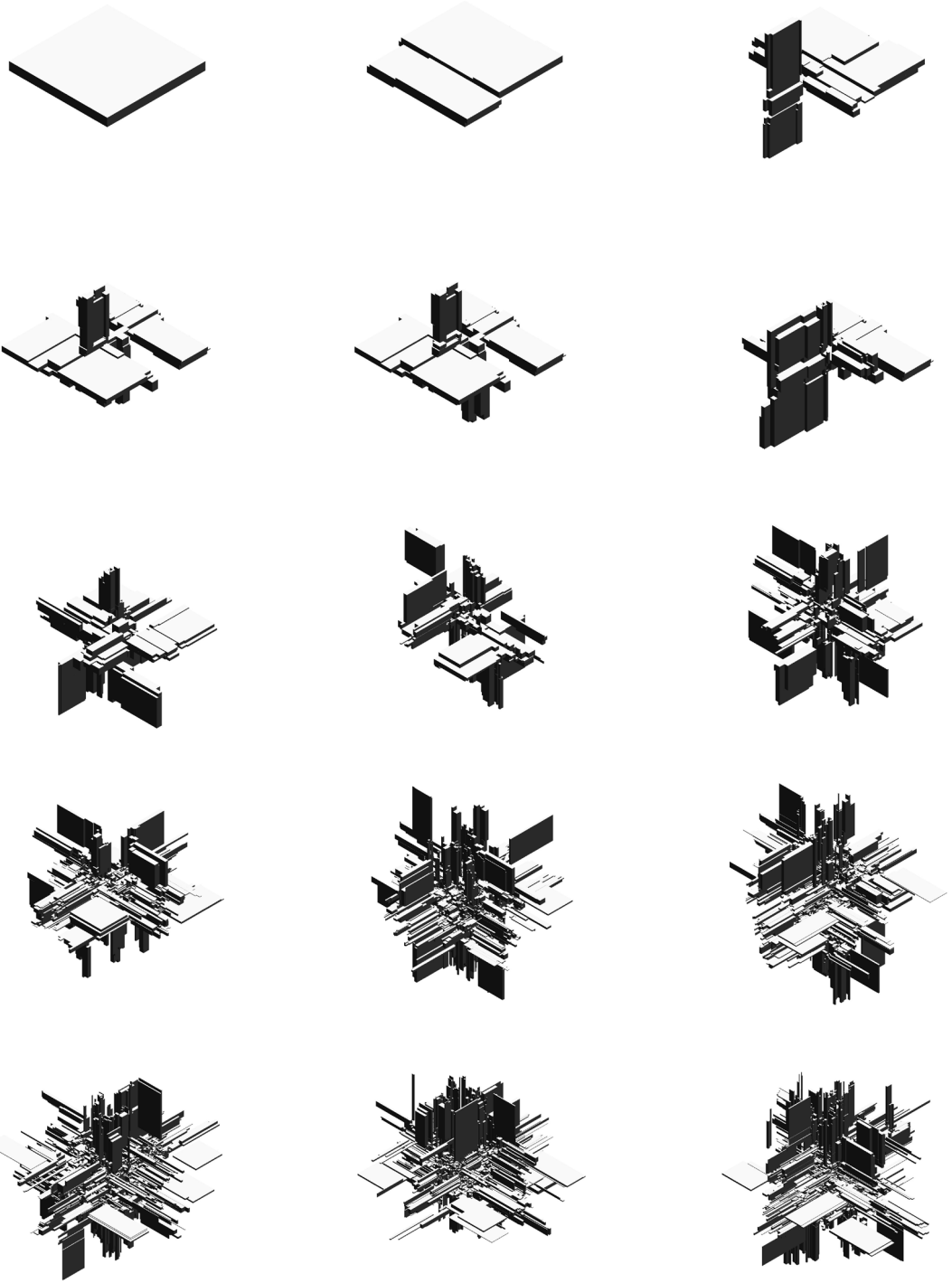


Figure 5.06
A potential design iteration generated by the computer.
Matrix Cube. Made using Processing, by Author.

Final Generations

All of the following outputs have been generated in Processing, and exported as a RAW format High-Definition PDF.



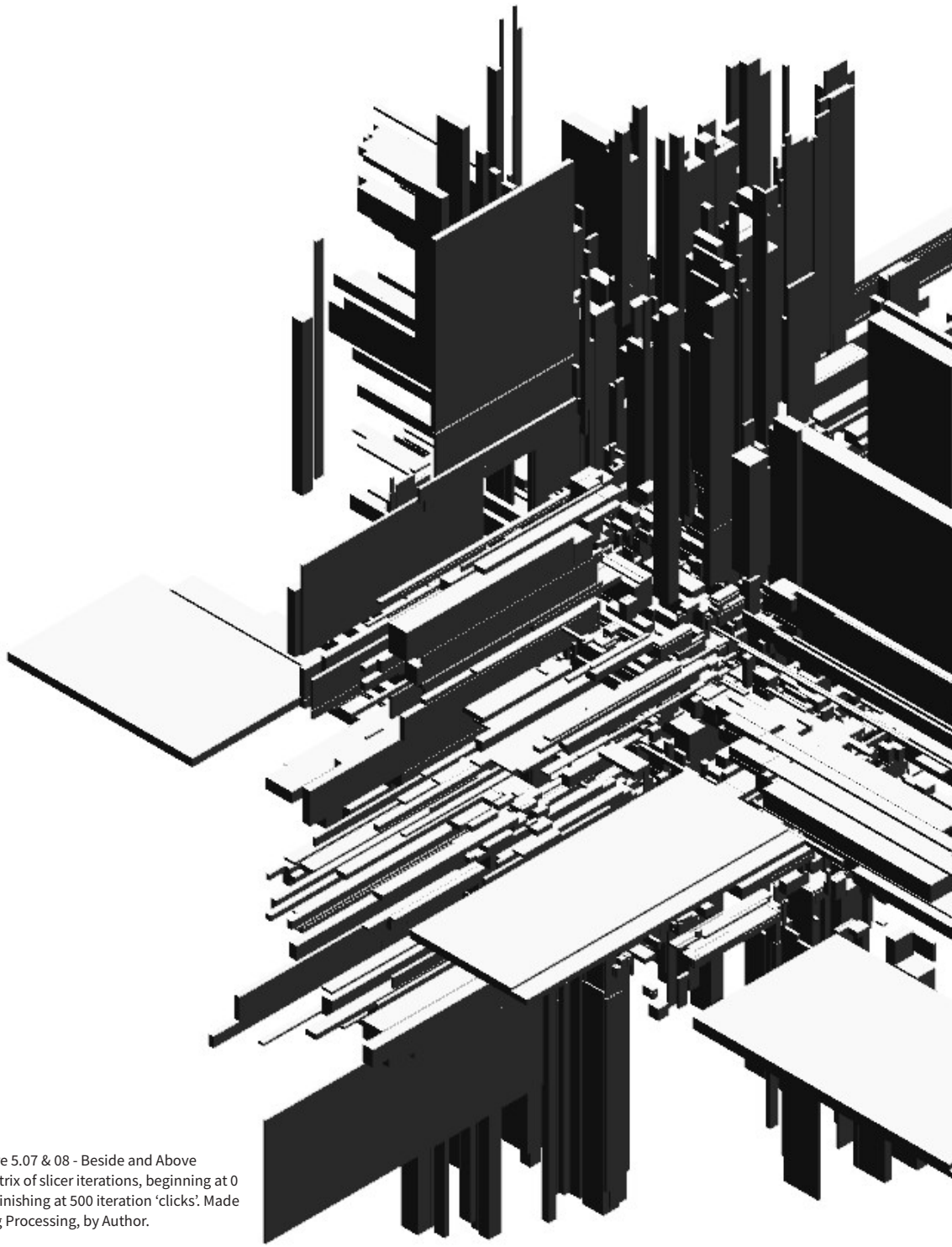


Figure 5.07 & 08 - Beside and Above
A metrix of slicer iterations, beginning at 0 and finishing at 500 iteration ‘clicks’. Made using Processing, by Author.

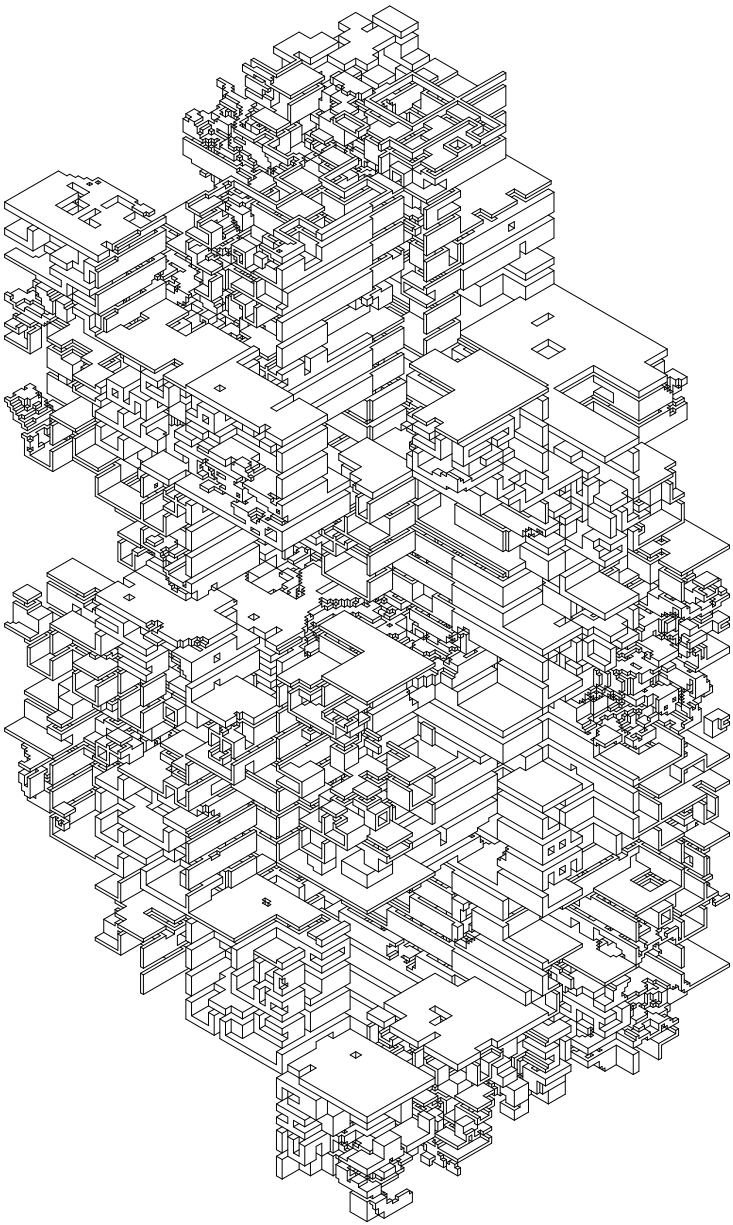
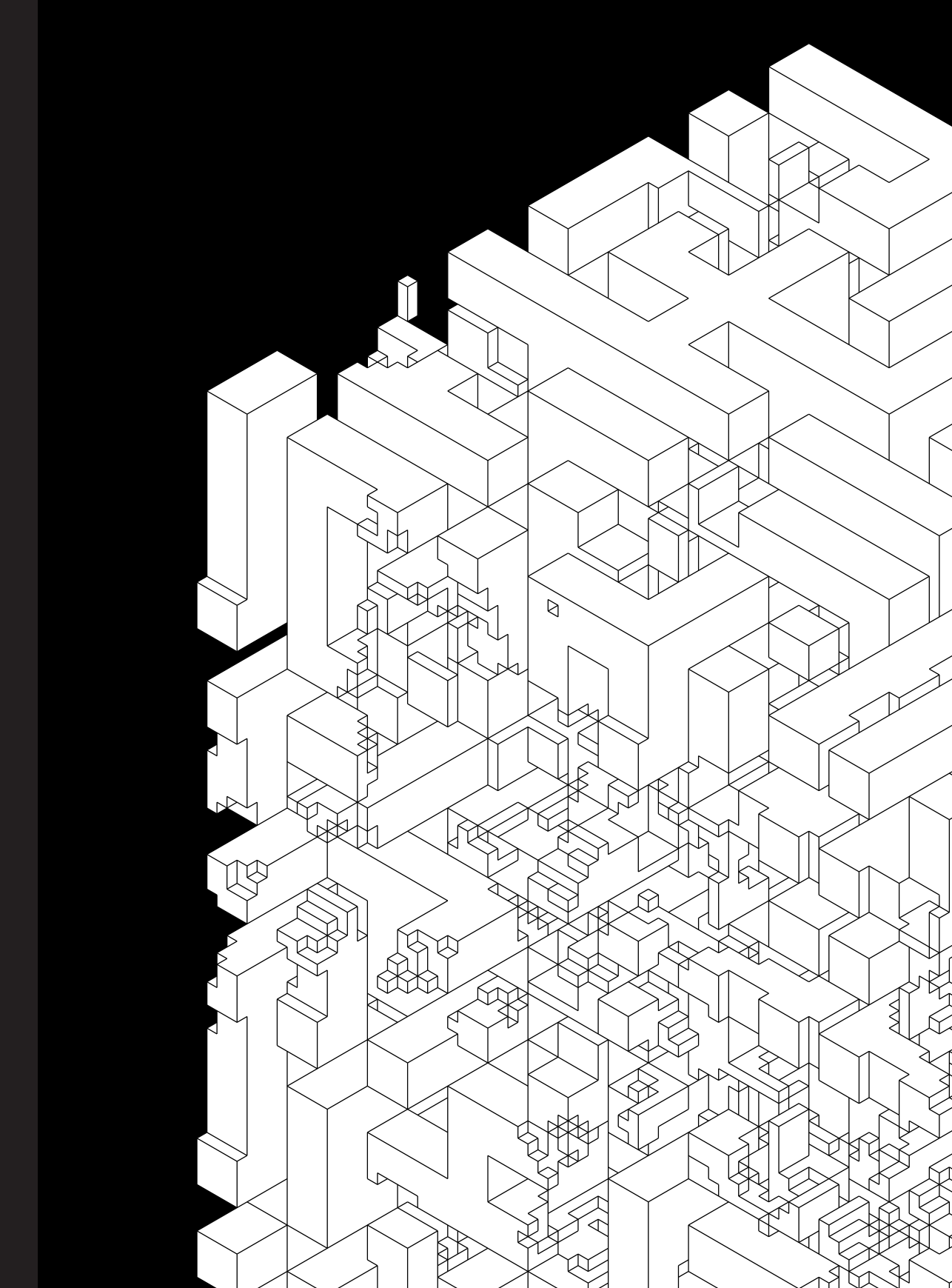


Figure 5.09 & 10 - Beside and Above
Isometric maze generator. Keyboard commands prompted removal of ‘ gaussian noise’ sections. Made using Processing, by Author.



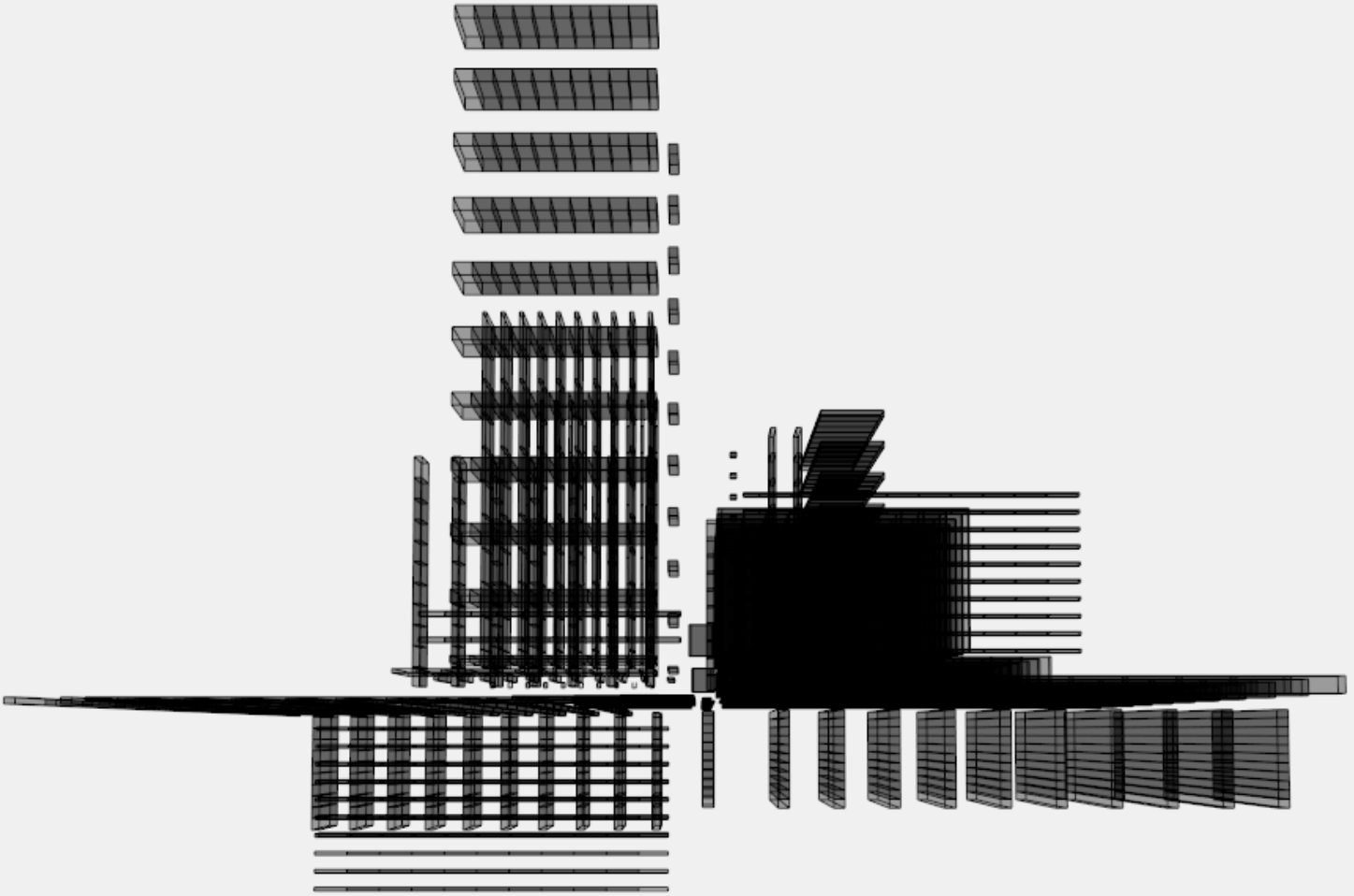


Figure 5.11 & 12 - Beside and Above
A collegial architectural generator. The script begins with a rectangle, and breaks the shape into equal planes, columns, and slabs. Keyboard commands prompt alterations in quantity or elements. Made using Processing, by Author.

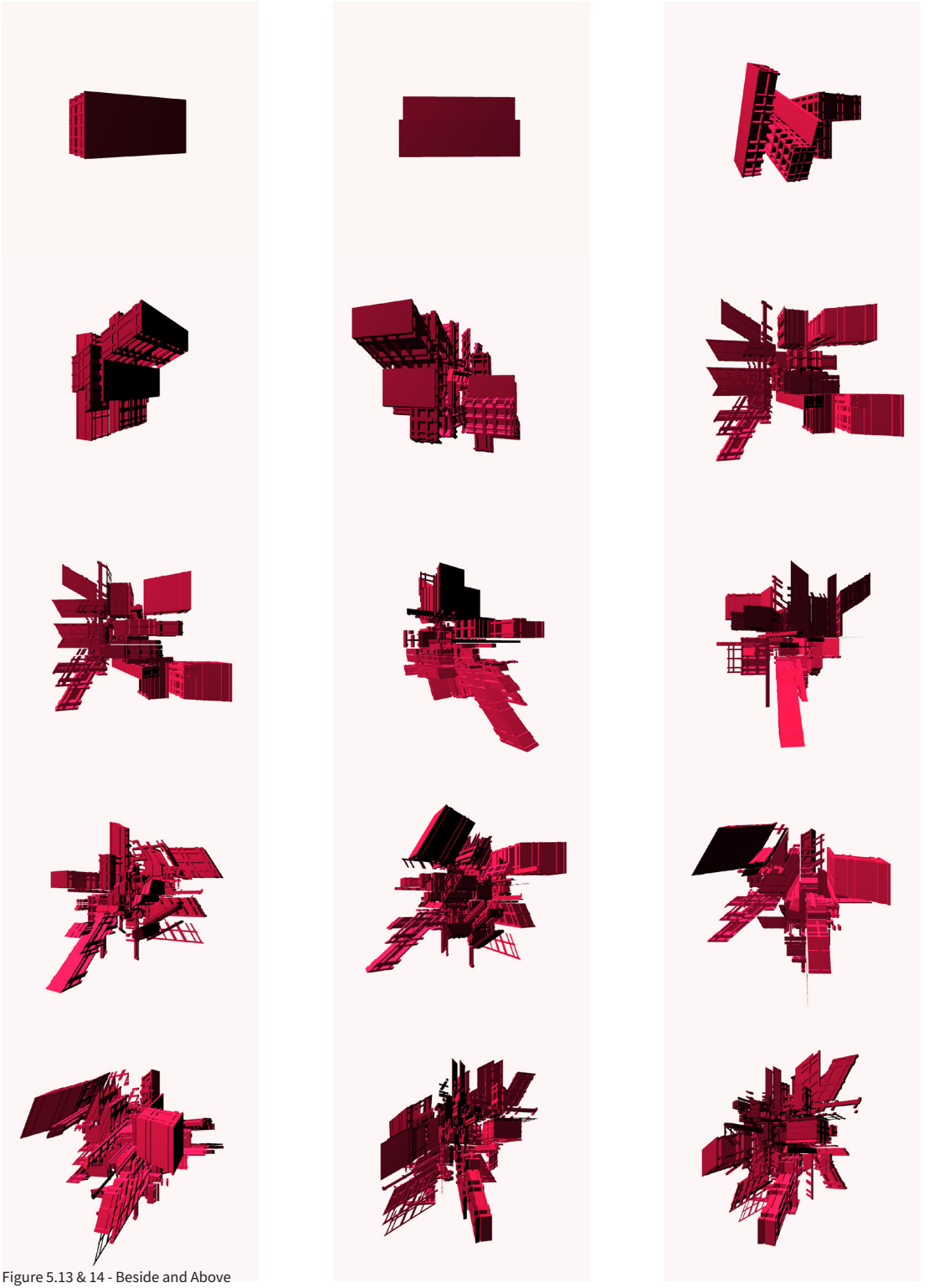
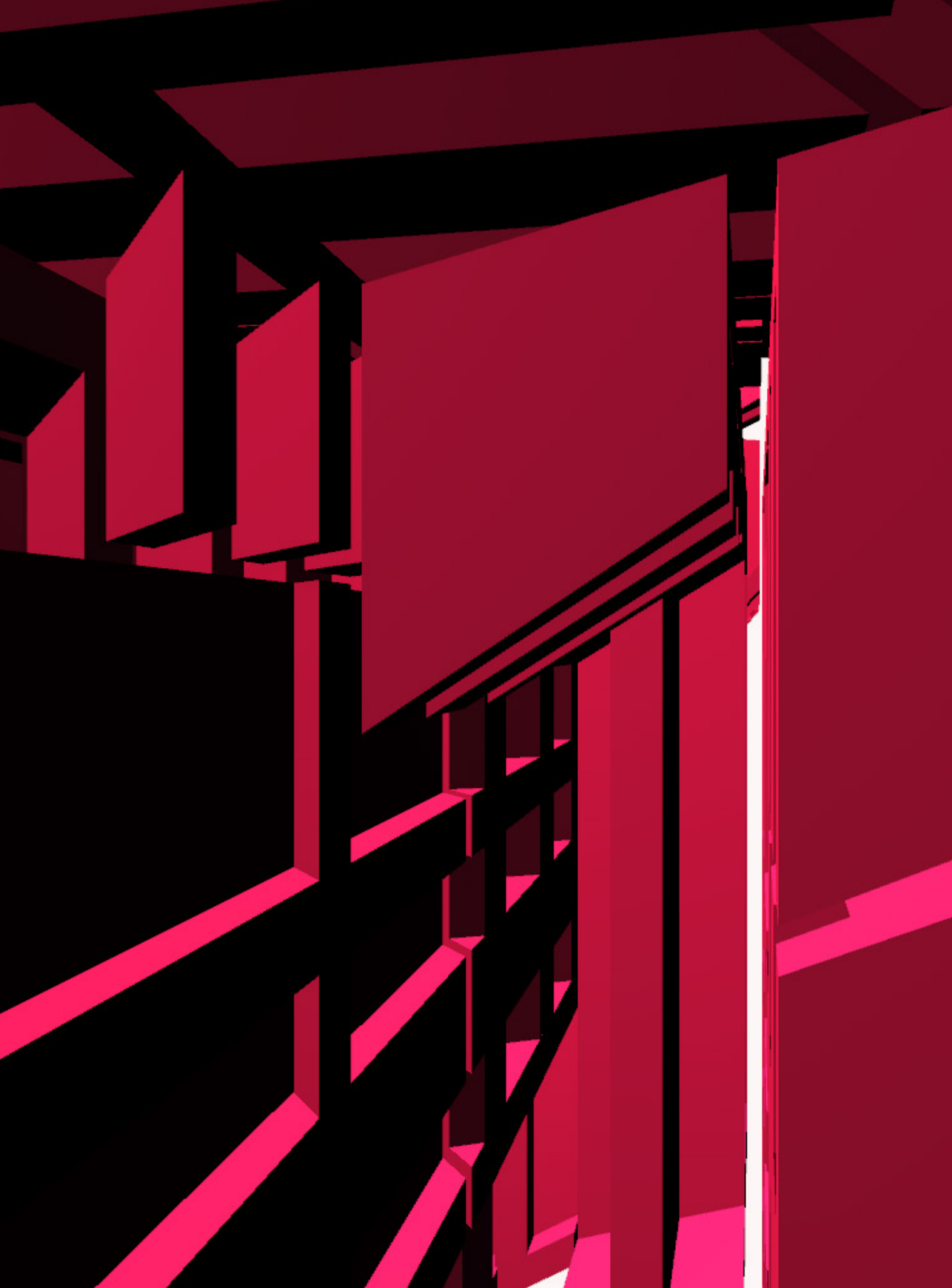


Figure 5.13 & 14 - Beside and Above
Matrix showing evolution of base form pictured first. Inhabition of form is achieved through ‘zooming’into the model. Light conditions provoke an architectural interpretation. Made using Processing, by Author.



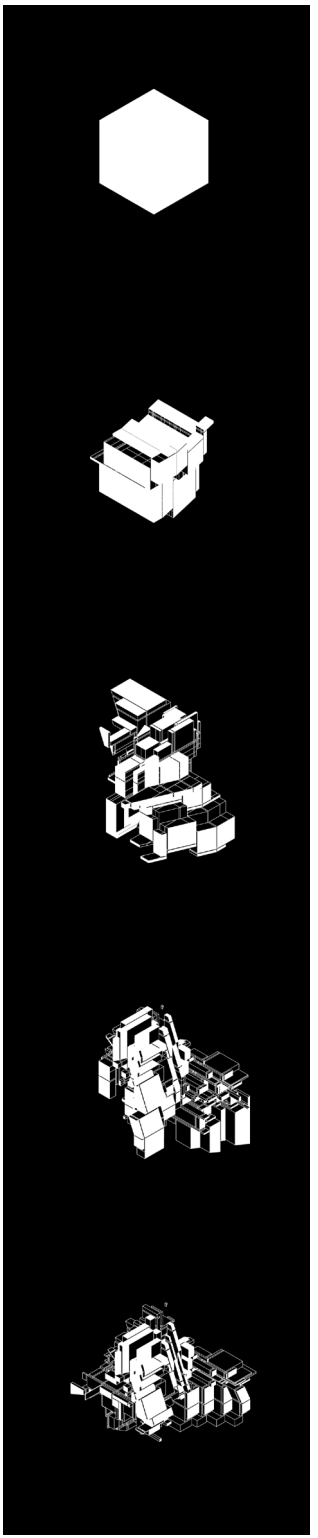


Figure 5.15 & 16 - Beside and Above
A collegial architectural generator. The script begins with a cube, and breaks the shape into shard-like shapes. Keyboard commands prompt alterations in quantity of slices made before returning to a cube. Made using Processing, by Author.

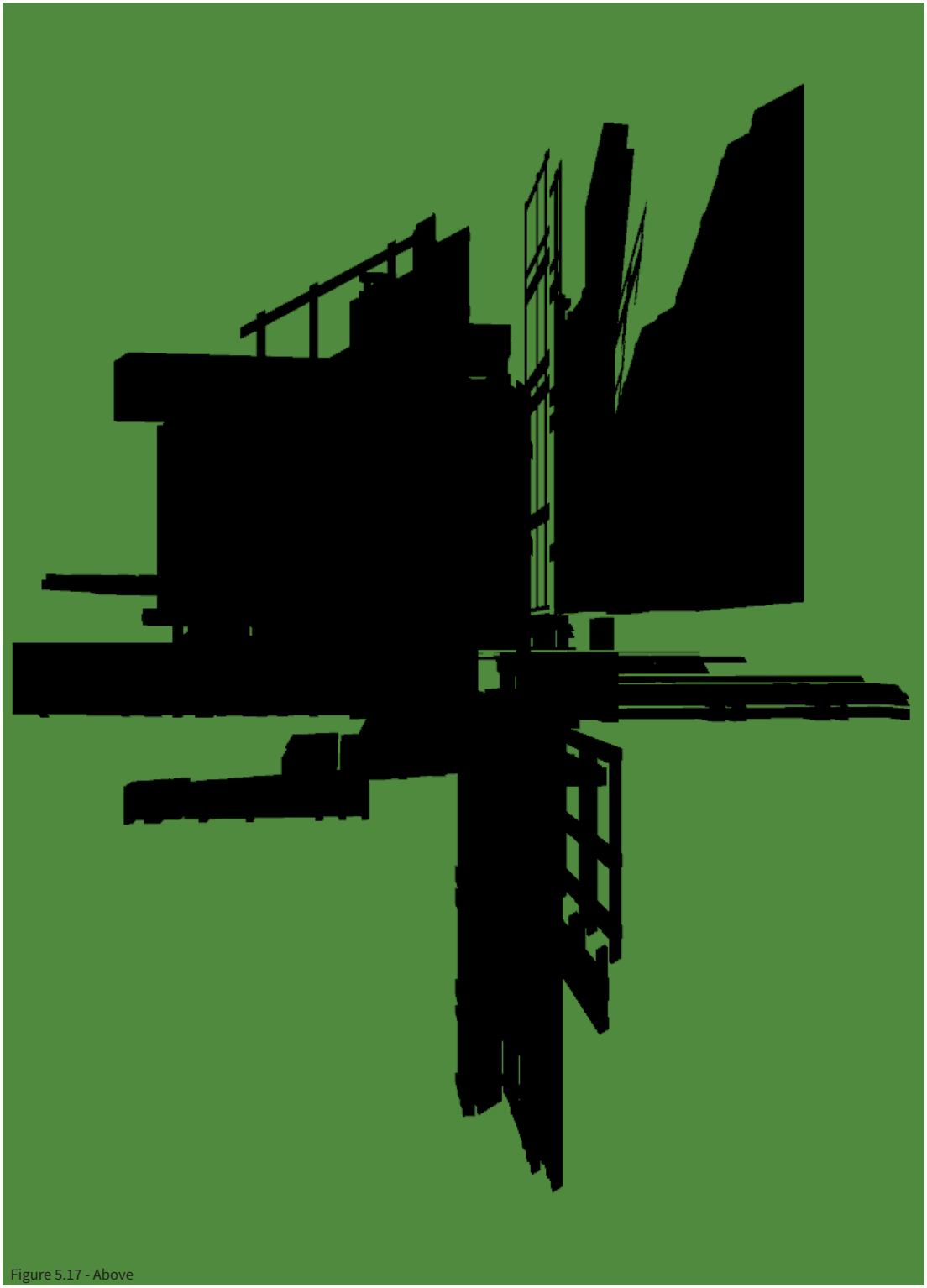


Figure 5.17 - Above
One of my Colleagues multitude of conceptual iterations. Made using Processing, by Author.

Conclusion

In conclusion, this research addresses the question of ‘how we can generate architecture using code and procedural systems’ within the practice of architectural design. This research addresses how architects and designers alike can incorporate a generative procedural tool into their day to day workflow It establishes a conceptual methodology in which I feel can be built upon in the years to come.

The work undertaken in this study informs how powerful a tool it can become and defines how it can be incorporated more frequently into the workflow of the conceptual and development stages of architectural design. Presently, an elementary design tool has been developed, tested, and put into practice with room for development. Enhancing the level of automation and interaction within the program will provide the end user, and their new colleague, extensive conceptual iterations. It will also enhance the user’s attention and satisfaction throughout the design process. Shifting the mode of general architectural conceptualisation and representation, I hope, will expand the field of architectural discourse substantially.

The use of a procedural or rule based system provides the user with a toolkit of possible actions to be combined infinitely, with room for expansion. By studying procedural systems, I have found that they are powerful mechanisms in which architecture revolves around and works with every day. General architectural rules, and codes, govern the field of architecture and make it the field it is today. By adding a new method of designing and iterating, I hope to streamline the conceptual design stage, allowing the Architect to work more efficiently. This will allow them to focus more of their attention on other crucial areas of the architectural design process.

Further development of the idea, ‘can we use code to create architecture’, will be the implementation of ‘site’ into the procedural design process. This one area of expansion would allow the architect or client to discuss with their colleague the site constraints, and to help provide a solution to a possibly tricky translation. Even for an architect drawing with pen and paper, the first step of their process is often to go to the site, to observe these constraints, and to design for and with the site, not against it.
Further development into these procedural systems is one that I hope will expand from what I have created so far, and allow this way of working to proliferate through the field, and not to scare architects away.

Looking at how this tool developed and how it can be used outside of the architectural profession takes me back to a question posed by one of the judges at the NZIA Student Design Awards. They asked; “Do you think that this tool has potential applications outside of the profession? As a design teaching tool perhaps?”. I am glad that someone recognised this potential of the program outside of the profession and for another demographic from the one I had intended the program to be used by. Yes, this tool definitely has applications as a teaching tool, in the way of teaching spatial relationships and massing studies.

With a deep look into this research thesis, a potential disadvantage in what I have created so far is the lack of ability to alter the initial volume of the animation past changing the X, Y, and Z lengths. If an architect or student wanted to undertake a massing study of a different geometric form (other than a rectangle or square), there would be no way for the program to cater to such a need at present. Further development of these programs may bring an ability to input an existing geometry that their 'colleague' would work upon, exploring different potential design iterations.

So, to readdress the initial question:

How can we use scripting techniques within the conceptual design stages of architectural design to generate and create architectural form?

The solution goes deeper than the understanding of code, or the benefits of working in a procedural manner. Rather, it is the appreciation and understanding that we as designers, and people alike, can see and use computers in a different way. Not only for their brute force processing power, but also for their unlimited variability and developability. It is the purpose of the architect, space, and fore mostly the user, that we can understand how a tool like this can be fully utilised in research and in practice. Can we allow the computer shared agency in the design process? Is it right to give up a portion of 'what architects do' to a computer?

This research has provided another step into the 'Architectural Computation' realm, with further design testing and experimentation, with regard to interpretation and representation.

This begs the question: "how would YOU use this design framework in your architecture office?". Only then, can this tool be further developed into one that not only assists creativity, but extends into all areas of architectural design.

List of References

Bødker, S., Ehn, P., Sjögren, D., & Sundblad, Y. (2000). Co-operative Design—Perspectives on 20 years with 'the Scandinavian IT Design Model.' 9.

Cohen, H. (1982). What Is An Image? IJCAI, 6, 1028–1057.

Colleague Definition & Meaning. (2012). Dictionary.Com. https://www.dictionary.com/browse/colleague

Dietrich, F. (1986). Visual Intelligence: The First Decade of Computer Art (1965-1975). Leonardo, 19(2), 159–169.

Fowler, W. (2016). Manfred Mohr – The groovy German who taught computers to make art. The Guardian. https://www.theguardian.com/artanddesign/2016/feb/12/manfred-mohr-the-man-who-taught-computers-to-make-art

Galanter, P. (2003). What is Generative Art? http://philipgalanter.com/downloads/ga2003_what_is_genart.pdf

Good Interaction Design = Good UX. (2021). https://www.getfeedback.com/resources/ux/good-interaction-design-good-ux/

JavaMathBits. (2021). Java Random Generation. MathBits. https://mathbits.com/JavaBitsNotebook/LibraryMethods/RandomGeneration.html

Kemp, M. (1998). Latham's life-forms. Nature, 391(6670), 849–849. https://doi.org/10.1038/36010
Lostritto, C. (2019). [Personal communication].

Kolb, D.A. (1984). Experimental Learning Experience as a Source of Learning and Development. Englewood Cliffs, NJ: Prentice Hall.

McClintock, C. (2020, September 14). A Beginner's Guide to Generative Design. https://www.ptc.com/en/blogs/cad/beginner-guide-generative-design

Mohr, M. (1983). Artists Statement—The Computer and Its Influence on Art and Design. Sheldon Memorial Art Gallery Catalog, University of Nebraska.

Noll, M. (1994). The Beginnings of Computer Art in the United States: A Memoir. Leonardo, 27(1), 39–44.

NZIA, N. I. of A. (2020). Student Design Awards. NZ Institute of Architects. https://www.nzia.co.nz/awards/student-design-awards
Schon, D. A. (1983). The Reflective Practitioner: How Professionals Think In Action. Basic Books.

Vanhoutte (W:Blut), F. (2021). HE_Mesh [Java]. https://github.com/wblut/HE_Mesh (Original work published 2019)

What is Processing? (2021). https://www.computerhope.com/jargon/p/processi.htm

Wrigley, K. (2015). Generative Art – Fascinating Things You Don't Know About. Studio A N F. https://studioanf.com/a-brief-history-of-generative-art/

Zafari, S., & Koeszegi, S. (2020). Attitudes Toward Attributed Agency: Role of Perceived Control. International Journal of Social Robotics. https://doi.org/10.1007/s12369-020-00672-7

List of Figures

Unattributed figures belong to the author

List of Figures