SCRIPTED TOWER

[CREATIVE CODING FOR RESPONSIVE ENVIRONMENTS]

CLAUDIA VAN VELTHOOVEN
SCRIPTED TOWER

[CREATIVE CODING FOR RESPONSIVE ENVIRONMENTS]

BY

CLAUDIA VAN VELTHOOVEN

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

Victoria University of Wellington
2017
ACKNOWLEDGMENTS

It is impossible to thank all the academic staff, colleagues, professionals, friends and family whose guidance and support, throughout my 5 years of study, have made this thesis achievable.

I would like to say a massive thank you to my incredible supervisor; Marc Aurel Schnabel. Your endless enthusiasm, valuable knowledge and experience, time and support have been truly revered. I would also like to thank Kevin Sweet and Serdar Aydin whose time, knowledge and support were invaluable throughout the year.

A special thanks to all of my friends, especially Anna and my flatmates. Thank you for being there for me despite my minimal presence in normal life this year. Thanks to all of the Archi Fam - it has been an epic journey together and I will miss you all. And thank you to my desk neighbour, Shayna, for keeping me sane.

There are not enough words to thank my Mum & Dad and siblings adequately. Your support has made it all possible.

A final thanks to my bro Isaac; thanks for surviving, and thanks for all of your caring.
Coding
Type 1
- L-Systems
Type 2
- Recursive Structures
Critical Reflection

Responsive Systems
Environmental input
- Wind Data
- Physical Modelling
Developed creative code
- Code 1
- Code 2
- Code 3
Discrete System
Critical Reflection

Code systems research
Developed creative coding
Computational Implications

Open Source
Authorship
Conclusion
# CONTENTS

ACKNOWLEDGMENTS ........................................................................................................... iii

Programming Architecture

**Abstract** .......................................................................................................................... 04
**Introduction & Terminology** .......................................................................................... 06

Research Proposal

**Emergence & Responsive Environments** ........................................................................ 14
**Methodology Review** ...................................................................................................... 22
**Position & Motivation** ...................................................................................................... 32
**Research Question** ......................................................................................................... 36

Computational Systems

**Coding** .............................................................................................................................. 39
**L-systems** .......................................................................................................................... 42
**Recursive Structures** ....................................................................................................... 54
**Critical Reflection** ........................................................................................................... 60

Responsive Environments

**Wind Modelling** .............................................................................................................. 63
**Physical Modelling** .......................................................................................................... 74
**Code 1** ................................................................................................................................ 82
**Code 2** ................................................................................................................................ 88
**Code 3** .................................................................................................................................. 94
**The Creative Code & Responsive Design System** ............................................................. 104
**Critical Reflection** .......................................................................................................... 112

Digital Property

**Open Source** .................................................................................................................... 120
**Authorship in the Digital Realm** .................................................................................... 122
**Critical Reflection & Conclusion** ................................................................................... 123

LIST OF FIGURES .................................................................................................................. 125
**Works Cited** ...................................................................................................................... 131
APPENDIX ............................................................................................................................... 135
PROGRAMMING
CHAPTER 1 | DIGITAL PROPERTY

1.0 ABSTRACT/INTRODUCTION
"A MEANINGFUL BUILDING OF THE DIGITAL AGE IS NOT ONE THAT IS BUILT USING DIGITAL TOOL; IT IS ONE THAT COULD NOT HAVE BEEN BUILT WITHOUT THEM."

MARIO CARPO. ‘THE DIGITAL TURN IN ARCHITECTURE’, 2012
(Carpo, 2012)
ABSTRACT

This thesis is based on investigating computational design methodologies for the generation of responsive environments, exploring the opportunities that creative coding and parametric based design approaches can offer architects of the 21st century. It is highly research focused with a test-case design formulated as an outcome of this research.

Architecture has always evolved with history to remain a manifestation and reflection of society and the materials/tools available to the architects at the time. Today, technological advancements and computational techniques have redefined the agency of design methodologies and manifested a paradigmatic shift to the context and practice of architecture. Computationally mediated form finding techniques have rapidly evolved over the past twenty years and continue to advance the possibilities to generate new building forms and complex, responsive, and adaptive geometries. It is undeniable that within the last decade, digital technologies have begun to initiate a paramount ensemble of affordance that are reconfiguring design and design thinking. This research invites the possibilities of creative-code-based design systems, as this approach becomes more widely taken up and more confidently embedded into the design process by architects and designers across the globe. Scripting/coding is typically seen primarily as a productivity tool compared to its potential to assist design exploration however, the power of computationally mediated design does not lie in computer aided design (CAD) based work flows, but in the possibility to engage with algorithmic design processes that unite design with the computer; the difference between computerisation (CAD) and computation (scripting).

This research investigates the potentials of a computational design methodology that integrates creative coding techniques for early-stage design approach. There are end-
less ways to utilise code-based processes for design; to begin, this research focuses on exploring a few common techniques, leading into a novel, innovative code that is context specific and locally and globally responsive. The nature of code is inherently open-source with digital files easily transferable and shared online. This poses prevalent current concerns surrounding topics of authorship, and generates speculations of how this may affect/contribute to the wider profession and the future of the architectural profession - another discourse addressed in this research.

The outcome of this research process is tested through a system for a responsive tower design. The tower is highly speculative and is primarily concerned/focused on the innovative design process and, accordingly, the potentials for creative-code-based generative design in architecture. The tower is context specific to Wellington, embedding local wind data and immediate environmental conditions into the code, employing an emergent/generative design that could not have been prior conceived-of with the human mind alone.

Developing on Carpo’s quote (page 10) further, this thesis builds upon this concept that there is meaning in employing digital design methodologies in such a way that allow for design developments which are unique, original, meaningful, and only possible because of the digital tools currently available to architects of the 21st century.
This thesis will cover techniques and methods that associate computation and design research in novel ways. It will discuss the effects of new design paradigms, brought forward by the integration of computation, by incorporating the interdisciplinary domain of climatology and local weather data sets, enabled through creative coding. Moreover, this research will debate how these different domains integrate with architectural design by setting up innovative methodologies which employ computationally mediated form generation. To conclude, this thesis will discuss the effects of computation and digital property in design and how the growing open-source nature of design may impact our architectural profession.

The critical understanding of computational design comes through its definitive distinction from computerisation (CAD). To clarify the terminology of specific language that will subsist throughout this thesis, the following definitions are given:

Scripting/coding
In this research, scripting/coding will be used interchangeably and both can be taken to mean computer programming through written text-based code. Coding, at its essence, is the “the means by which the user gives highly specific instructions to the computer with which they are interacting”. (Bury, 2011)

Creative Coding
The purpose of integrating code into the design process for this research is to use it as a means to create something novel and expressive, rather than its more common use for functional means. Creative coding is becoming increasingly popular within the architecture and design discourse and offers endless possibilities to designers with the relevant skills and knowledge. Creative coding may be original code written by the programmer/designer, it may be modified code from open-source platforms, it may be a collaboration
between multiple coders, or it may be pre-written code that is used in novel ways.

Algorithm
An algorithm, in the context of this thesis, is a “computational procedure for addressing a problem in a finite number of steps … it is the systematic extraction of logical principles and the developments of a generic solution plan” (Terzidis, 2006).

Programming
In regards to digital technology, programming is to write code for a computer program application. ‘Python’ programming language and ‘Maya Embedded Language’ (MEL) is used for this research.

Computation
Is about extending the human intellect through the power of computational processes. It is the procedure of determining something by algorithmic methods through bottom-up processes, without preconceived formal outcomes decided. (Terzidis, 2006).

Computerisation
Mouse based manipulations of 3D computer modelling where the outcomes or processes are preconceived in the designer’s mind before being digitized through manually entering, manipulating and storing it in the computer (Terzidis, 2006). (nb. this research does not use mouse-based design methodologies and hence this definition adopted from Terzidis, is not the conventional/typical definition but rather a programming based definition.)

Parametric design
Parametric design is where values of parameters within a schema of relationships can be manipulated for the generation of procedural instances. Parametric design focuses on “a logic of associative and dependency relationships between objects and their part-to-whole relationships” (Oxman & Oxman, 2014).

Bottom-up
“As opposed to top-down, the idea that you can investigate phenomena by starting from its simplest components and simulating their relationships to generate the overall structure. Part of the notion of emergence” (Coates, 2010)

Complexity
“To be distinguished from complicatedness. Complexity is the idea that a quite small
collection of linked components can generate a very large number of usually different outcomes. In particular, that the outcomes are not specified somewhere in the system, but emerge once you start the system up.”” (Coates, 2010)

Coding and computation is not new to architectural design but was originally considered the task of a specialist (Bury, 2011). It is only in recent years, through the increased integration of technological advancements and appreciation of digital design potential, that code based design approaches have promoted change to conventional methods and initiated a new trajectory for digital design in architecture.

Coding is a higher level of computer programming than typical for architectural design processes where, in the existing “open-source” environment, ‘libraries’ of functions can be combined with preconfigured routines (algorithms) as a means to produce … a significantly deeper engagement between the computer and the designer by automating routine aspects and repetitive activities, thus facilitating a far greater range of potential outcomes for the same investment in time” (Bury, 2011). In short, scripting, as an approach to computational design, offers access to whole new ways of exploring design. However, computational techniques and technological advancements of the 21st century comprise both an immense opportunity and a considerable challenge to architecture and related design disciplines, as discovered and discussed throughout the course of this thesis.

The generated forms from the scripts written by the designer become a result of forces applied within a generative system, concepts that draw parallels with evolutionary systems observed in nature. (Tedeschi, Wirz et. al, 2014). The complexities present in these naturally occurring systems can be observed and applied as generative morphogenetic processes to model building forms through computation. This is an enticing concept within the architectural discourse as designers inherently seek unique forms albeit restricted by construction limitations. By looking to nature for inspiration, one can understand how complexity and beauty can be created through simple recursive component aggregation and rule-based behaviours not unfathomable to the construction industry.
Research Proposal
2.0
EMERGENCE & RESPONSIVE
ENVIRONMENTS / METHODOLOGY
REVIEW / POSITION & MOTIVATION /
RESEARCH QUESTION
Figure 2.0: Example of design through Emergence by Kokkugia
Design approach can be divided into two opposing categories; “top-down” and “bottom-up”. Typical design and problem-solving is predominantly top-down, in contrast with more explorative, and evolutionary, approaches which are predominantly bottom-up. A top-down strategy assumes that the author knows what the outcome may/can be as a whole, and just needs to work out what are the components to make it up, whilst a bottom-up approach assumes that the author can define the underlying components and rules, but does not have any preconceived idea of how they can be combined to form the outcome/solution. (Coates, 2010). “Emergence” or “generative” design fits into the “bottom-up” category.

The bottom-up theory of emergence is adopted to allow for the research in this thesis. As described by Hensel, Menges and Weinstock, “Emergence’ is the scientific mode in which natural systems can be explored and explained in a contemporary context. It provides ‘models and processes for the creation of artificial systems that are designed to produce forms, and complex behaviour, and perhaps even real intelligence” (Hensel, Menges, et. al, 2010). Emergence can also be understood as “generative design systems”. A well known definition of emergence is given by John Holland as “much coming from little” (Holland, 1998), or by Steven Johnson in his popular book of the topic as “movement from low-level rules to higher-level sophistication” (Johnson, 2001).

This bottom-up approach can be understood as algorithmic code-based systems that refer to the aforementioned “emergence’ of forms and behaviour from complex systems of the natural world” (Hensel, Menges, et. al, 2010) that are able to “manifest themselves in a dynamic and adaptive way based on the basic codes and parameters that define them” (Lombardi, 2014). Creative generative algorithms allow designers creativity to describe
forms, shapes, organisations systems and configurations as relational, dependent of the systems legible internal forces that determine them. The generated forms become a result of forces applied within a generative system, concepts that draw parallels with evolutionary systems observed in nature. The complexities present in these naturally occurring systems can be observed and applied as generative morphogenetic processes to model building forms, as exemplified in the following investigations.

The methodology of this thesis is very much research based and an exploration of various computational methods to understand how a designer may integrate creative coding into early stage design approaches, hence the thesis follows a trial-and-reflection design structure, before arriving at a design methodology generated as an outcome of the research. The focus of the design approach which integrates scripting techniques is that of form finding, as opposed to form making. Form finding is in relation to a bottom-up approach, whereas form making is top-down. For the scope of this thesis, I am only interested in bottom-up strategies that are capacitated through computational systems/techniques.

Emergent and generative design is achieved through programming architecture. Programming in architecture has continued to gain intrigue through its capacity to manifest phenomenal modes of expression. Stephen Wolfram, a mathematician of the 1980’s, was the first to suggest that computation, rather than mathematics, was the correct method to simulate complex natural and artificial behaviours. Wolfram published that simple rules can produce vast complexities - a concept that has since been readily adopted by designers worldwide. Simultaneously, DeLuzian theories of difference and repetition, inherent to bottom-up processes, became intensified for architectural programming environments (McCullough, 2006). Significant precursors of this arena can be identified in the work of Greg Lynn, Karl Chu, John Fraser, to name a few, along with earlier work of Frei Otto.

In the text ‘The Autopoiesis of Architecture’, Patrik Schumacher denotes architecture as an autonomous network of communications. Autopoiesis, meaning self-production, refers to a concept within biology which describes characteristics of “circular organization that reproduces all its specific components out of its own life processes” (Schumacher, 2011). The theory that Schumacher inculcates in his book is that architecture is, and should remain, a closed and self-referential system that reacts to outside sources as one. This is a concept which aligns with that of emergence and of particular interest is the idea that a system is conditional to its environment and visa versa.
Figure 2.1: Xi An - Competition proposal project by Richard Goodwin

Figure 2.2: Xi An - Competition proposal project by Richard Goodwin
Figure 2.3: khalęesi tower
(Tucker, 2017)
Leading today’s architecture profession are those designers who pioneer with creativity and experimentation through programming, reducing the exhausted priorities of disciplinary continuity and function. In order to move forward, we must discontinue to repeat what is already known. Self-developed creative code and computational strategies fuel intuition and motivate artistic production in architecture and design, while stimulating the spread of individual creativity - an example of current projects exemplifying this movement can be seen through Richard Goodwin’s work in images 2.1 and 2.2. (Goodwin, 2017). Goodwin is notably interested in site and contextualising public art and architecture in novel ways. He observes that “site specific artworks, within the realms of the city, are in crisis.” and that “architecture is the new site for public art in the realm of the city”. (Goodwin, 2006). Of particular appreciation is Goodwin’s artistic development and design philosophy and approach. He pushes boundaries and tests ideas with seemingly no reservations for conventional beliefs. He strives to “test the functional and aesthetic boundaries of public space in the city” (Goodwin, 2006) through opening up the city and breaking away from solid repetition, a concept which is also appreciated when carrying out the research and tests of this thesis, along with his belief that “buildings are machines that can be discarded like old clothing when technologies improve” (Goodwin, 2011).

The edition of the AD journal ‘Evoking through design: contemporary Moods in Architecture’ (Campo, McMorrough et al., 2016) provides an extremely relevant quote in relation to the aim of this thesis;

“While forgotten by many, architecture is, in fact, a creative act. And yet the power of architecture and its imagery to prompt change exists only as architectural myth and ancient lore - a power associated only with the Miesian gods of yesteryear, a magic no longer known or practised. Instead architects today focus on simpler problems, more local problems of context (it looks like the mountains in the distance), of programme (put a starbucks in the bathroom), of marketing-friendly animal metaphors (it’s a bird), or how to consume a bit less energy (mum look - LEED copper). Like a decrepit sorceress robbed of her powers, architecture sits mute in a world of technologically enabled explosive possibilities, vaguely remembering having had, in the past, perhaps ore respect, more prestige and certainly a lot more power.” - (Campo, McMorrough et al., 2016).

To summarise, computational tools give architects the power to advance the creative process, as exemplified through the work of the previously named designers.
Along with the focus of this research being on emergent computational processes, the designed outcome is directed towards exploring the tower typology as a scenario to test the methodology. Dominant public building designs should not be in isolation from their existing environment but rather should be derived from it, and stand inherently related to the context in which they root. Tower architecture devotes a public meaning/impact and this public meaning should respond to, and interact with their context, taking the focus away from their height and single verticality, and becoming in dialogue with the public realms of the city. In a paper published by the Council on Tall Buildings and Urban Habitat, the author states that skyscrapers are in need of liberation as “ultimate height had become the only achievement the skyscraper aspires to” and that the “one dimensional competition skyscrapers engage in has exhausted its typology, making it a typology of mediocrity” (Gianotten, Koolhaas, et. al, 2014). Furthermore, New Zealand will inevitably need to start going up, rather than continually out, which is already beginning to happen with new tower proposals for Auckland in motion. New Zealand does not have many tall buildings (see comparisons in images 2.5, 2.6 & 2.7). One could even argue we have none when compared with on a global scale (see image 2.6 with the tallest NZ building, the Vero Center, being 172m (the skytower is the tallest structure at 328m) compared to image 2.7 with the tallest building in Canada being 355m). However tall buildings will become a more prolific typology. As declared by Bates Smart, “they will increasingly play a role in meeting the needs of society through the creation of public space and streets, improving amenity and sustainability as well as contributing iconic forms to the skyline” (Bates Smart Architects, 2016).

To analyse the value of this project and the final design outcome, it is pertinent to discuss another designer; Mark Foster Gage Architects tower design proposal ‘The Khaleesi’ (Figure 2.3) in New York City. The tower is described as follows, “the project is not only differentiated from the local context by height, but also by high resolution detail that invites curiosity rather than providing for an immediately understandable architectural one-liner” (Campo, McMorrough et al., 2016). The objective of the Khaleesi Tower was to provide a truly inspirational design that is capitalizing on our current advancing technologies and progressive computational design approaches. Mark Foster Gage worries that the architectural profession has fallen too far down the path of pragmatism and practicality and argue that designers are too focussed and constricted by aspects such as program, tradition and sustainability. Current technology is fast expanding the possibilities for unprecedented and wildly intriguing additions to our built environment, yet the abiding focus on pragmatism leaves little space for this truly inspirational creativity.
Another project that is aligned with the analysis of this work is that of Toyo Ito’s wind Tower (see Figure 2.4). This project “not only embraces technology and involves it in a dialogue with the city, but also establishes a direct symbolic relationship between nature and the installation” (“AD Classics: Tower of Winds / Toyo Ito”, 2017). The same symbolic and semiotic relationship with the unique climatic environment of Wellington, can be understood through this research project, due to the novel design approach and ‘hyperlocal’ outcome.

The design of which this research has concluded at, pictured in image 4.27, is continually reiterated, focussed predominantly on the design process and approach, however, it is this computational approach that is capable of producing the uniquely creative and awe inspiring designs of which Mark Foster Gage alludes to. Novelty must be the result of discovery which is possible through the codes and systems described in this research. Designers must exploit the potentials that computational design systems can offer the discourse of architecture and laterally take advantage of advancing building technologies in order to keep pushing the boundaries of what architecture can offer society, and reinstate the relevance of an architect’s role in the 21st century.
Figures 2.5: Heights of the top 20 tallest buildings in the world

Figures 2.6: Heights of the top 20 tallest buildings in New Zealand

Figures 2.7: Heights of the top 20 tallest buildings in Canada
METHODOLOGY REVIEW

design approach precedents

Four key precedents are introduced in this section whose design focus is based on the principal points discussed in the preceding literature research of emergence. These practices/projects are chosen for their design approach and design process with their outcomes not being of so much interest, as the objectives of this thesis is similarly focussed on the exploring and testing similar computational processes. The ideas include themes of emergence, computational form finding, and creative code-based integrated design systems. The precedents on the following pages exemplify one or more of these themes through the chosen project examples.
‘Biothing’, founded by architect Alisa Andrasek in 2001 seeks to mediate the intersection between matter and information. Biothing is a cross-disciplinary laboratory that “focuses on the generative potential of computational systems for design” (Andrasek, 2017). Biothing progressed into ‘CONTINUUM’ in 2005 as a research collective with a stronger focus on advanced computational geometry and software development.

The project pictured in Figure 2.8 is a design system for a furniture display series that draws on built ecologies found in nature - it is titled ‘a_maze’, 2009. A recursive algorithm that employs logics of fractal Koch curve mathematics was scripted for the design and folding of strips of material that make up the growing furniture display. The algorithm is programmed to ‘adapt’ in certain ways as it grows, meaning that the frequency and sizes of the folds vary throughout the process. The system is also parametrically programmed so that it can recursively subdivide through any number of prescribed generations (Andrasek & Brayer, 2009).

The research and processes Andrasek employs of a computational approach that engages directly with the script in a open source manner, is of great interest for the research within this thesis. The open source nature and growing culture of shared knowledge in this area of architecture and design is fuelling an accumulative library of codes and methods for computational strategies. It is also opening up discussions centred around topics of authorship and cross-disciplinary collaboration. These are key topics for this thesis and ‘a_maze’ is a specific example for script-based design that draws on algorithms found in nature - a primary concept explored for the system and process that has been designed as an outcome of the following research.
Joze Sanchez is an architect, game designer and founder of his practise and research titled ‘The plenty project’. His library of coding tutorials have been of major contribution towards the learning and research carried out in this thesis. Along with gaming, his practise is interested in bridging the relationship between architecture and generative design. One such generative project is ‘Bloom’, a joint project with the aforementioned Alisa Andresek (Figure 2.9).

The objective of Bloom was to engender a social/ crowd-sourced ‘game’, whose formations would emerge through public participatory design. This approach allowed for the system to remain versatile, with infinite configurational variations to unfold. The generative open-ended design of the system meant that the resulting designs grew in response to the social input and parameters immediate to it. The ‘game’ is a geometric system based on the aggregation of the same component. Sanchez and Andresek wrote a custom code to simulate the optimal initial geometric component that was able to facilitate the most variable and compelling formations. They developed simulations for the modular construction system that was capable of generating complex cellular configurations from the standardized unit. The initial unit prescribes certain growth rules from its shape, and through different combination sequences, endless designs will emerge in a ‘bottom-up manner’. This is a process that can be observed in nature - compelling patterns developed from the same, simple repeating component. This research takes lead from the success and precedent of Bloom and looks to also examine how complexity can emerge through simple component aggregation - an enticing concept for the search for unique yet constructible forms that is ever present within the architectural profession.
The core concept for Theodore and Stephen Spyropoulos, founders of ‘Minimaforms’, is that of architectural/spatial form finding; finding being the key word, as opposed to form making. They understand architecture as a field of performance and knowledge that is continually undergoing formation to reflect and evolve in line with new material and social realities.

A growing interest of theirs, and of particular relevance to this thesis, is their investigations towards the making of behaviours themselves. Minimaforms have developed many responsive environments/designs and describe, in one instance, this concept as
reconceiving buildings “in ways that emphasise less their physical features, and more the relationships that describe these entities to their surrounding worlds (whether artificial, like other buildings or infrastructures, or natural, in the sense of physical, environmental qualities).” (Spyropolous, Steele et al, 2013).

Theodore Spyropoulos also is director of a studio course at the AA DRL school in London. The diagrams in Figure 2.10 are an example of one of the projects carried out in this studio and is relevant to this research.
'Kokkugia’, a design research lab based in Melbourne, founded by Roland Snooks (Snooks, 2017), with a current agenda which is focussed on iterative algorithms to produce an augmentation of intricate mass, rather than predefined form. Snooks design approach employs multi-agent systems for the exploration of ‘behavioural formation’ that emerge from computational processes (Snooks, 2016). The project depicted in Figure 2.11 is defined by complex material assemblage rather than standard wall, surface, or volume attributes, and thus offers “weird and wonderful” atmospheric effects (Snooks, 2016).

A central focus of these projects is the affectual capacity of the object that is generated and intrinsically tied to the behaviour scripted into the design process. The emergent ‘objects’ are perceived as ‘strange’ and ‘weird’ however their characteristics are explicit to the parameters that define them and offer a complexity that stimulates user/viewer imagination. It is this affectual potential which Kokkugia seeks to capitalize and explore. This concept is relevant to the research of this thesis in terms of how the process can affect the outcome and obtain alternative spatial effects to conventional and well-understood architectural conventions. Snooks employs computational investigations with a focus on the process and the open-endedness of the generative spatial organization that emerge.
Figures 2.12, 2.13, 2.14, 2.15, 2.16, 2.17: Wellington wind making the news
POSITION & MOTIVATION
the aim of this research

The motivation for this research thesis is to position myself within the digital movement that has enriched our profession’s approach to design. I want to equip myself with the skills (ie coding) and understanding/knowledge to create generative, emergent and responsive forms and environments, through manipulation of programming environments and explore the potentials for integrating scripted processes into the creative design process.

Hence the focus of this research is to form a computational design methodology that recognizes the design of an architectural form as an opportunity to design a unique, coherent system of signification, explicit to its contexts climatic conditions through utilizing climatic data as the driver in the design process. In the process of realising this stage of the research, many tests and research-focussed experiments were carried out and documented along the way. Processes that draw on nature are a dominant focus for many of these tests. Specifically, the research looks to explore how we can interpret, represent, or express our surroundings taken as natural, through the design process that generates artifices (Hensel, Menges, et. al, 2010). As such, the final stage of this research decides to work with one local condition of site specific data sets: Wellington’s wind. It will focus on how these rules derived from the windy conditions of Wellington, can be drivers in computational techniques to generate an architecture of specific geometry that is unique to its context by extracting live local data for the embedment of site specific data into the design process. This research suggests that by integrating site specific wind patterns into a generative design process, the designer can generate forms that respond to the surrounding context, and can then analyse the system to create local responses within the systems, forming one discrete system, open to morphologic variations.

Through investigating the role of computation in the design process and its ability to
empower designers to breach into other fields including various scientific domains i.e. climatology, the design become capable of infusing innovative ideas into design.

The final investigation focuses on the climatic factors of Wellington’s wind and how the extraction and application of live weather data can relate to architecture and spawns a hyperlocal and hence socially significant building form. The investigation offers a methodology for a preliminary design approach that encompasses climatic considerations for the emergence of unique forms, which can only be derived from its specific environment. Hence it empowers the form with the ability to convey a novel design significance for the local context and is therefore argued as hyperlocal as it is using immediate environmental data for generation/emergence of data embedded formations.

The outcome aims to form an emergent design created through a responsive system recognizing some of the existing phenomena of both global (in this case environmental data) and local influence (variables within the system and between the various coded constructs).

Two different aspects of the research carried out during this thesis were published and presented at conferences in London1 and Japan2. The topic of the London conference was focussed around ‘Parallelisms in architectural computing techniques’. The paper I presented generated discussion amongst the participants who included Joze Sanchez, Gilles Retsin, Mario Carpo and Patrik Schumacher, and they provided essential feedback/forward for the direction and learning of the research explained within this thesis. Patrik Schumacher and Joze Sanchez discussed my research with me and offered questions and postulations around the creative coding aspect and relevance of the work. These conferences proved that the digital movement within architecture is pivotal for the development and innovation of the profession and placing myself within the scope of this digital realm was the central motivation for this thesis.

The other conference in Sendai, Japan provided stimulating discussions from renowned architects such as Toyo Ito. This conference was focussed on climate and cultural resilience and my paper that I presented, based on interdisciplinary design approach through architecture and climatology in Maya, was received with great interest and resulted in winning the ‘Best in Session’ Award. The research presented at both of these conferences is elaborated on throughout this thesis.


Academic Award
Figures 2.18: Patrik Schumacher and I at the PACT conference in London, September 2016

Figures 2.19: Toyo Ito and I at the ISAIA conference in Sendai, September 2016
RESEARCH QUESTION

To summarise the intent of this thesis, the research question is as follows;

*Can creative coding generate emergent forms that lead to meaningful and novel architectural designs?*

The main points to note are categorised as;

• Creative algorithmic coding strategies
• Responsive environments for emergent form *finding*
• Coding in design
computational
3.0 CODING/ L-SYSTEMS/ RECURSIVE STRUCTURES/ CRITICAL REFLECTION
CODING
in relation to architecture and design

Procedural/algorithmic processes facilitated by code is the foundation of form generation in digital design (Oxman & Oxman, 2014). As noted above, one of the methodologies behind computationally mediated design is the use of generative/emergent algorithmic systems. The integration of code into the design process allows designers creativity to describe forms, shapes, organizations systems and configurations as relational, dependent determined rules within the code. This concept is used for all the following code experiments and developments.

There are numerous programming languages in existence and their value lies in the task of which they are need for. For this research, Python programming language and MEL script (Maya Embedded Language), along with the visual programming of Grasshopper3D, will be used. Code written in Processing will also be learnt and employed for brief experiments in Processing software sketchbook. Python was chosen as it is a very powerful general-purpose language with endless resources due to its popularity of being ‘beginner-friendly’. It is also very widely used across multiple softwares and disciplines. Coding requires diagrammatic thinking, conducted through notating out the relationship rules of the system (Coates, 2010).

The intrigue of code-based design approach is in its power to conceive the inconceivable. The human mind is unable to comprehend levels of complexity that are able to manifest when the designer is capable of working with the computer. This concept is clearly described by Kostas Terzidis in his book, Algorithmic Architecture (2006);

“The power of computation, which involves vast quantities of calculations, combinatory analysis, randomness, or recursion, to name but a few, point out to new
“thought” processes that may have now ever occurred to the human mind. These “idea generators” which are based on computational schemes have a profound ability not only to expand the limits of the human imagination but also to point out the potential limitations of the human mind. What was inconceivable once may have been so mainly because it may have escaped the possibilities of existence.”

This gives rise to shared authorship with the computer, the software, programme codes, and designer, and is further discussed in chapter five.

It is key to note that although the term ‘algorithmic’, and in turn the outcome, are often synonymous with complexity, the process itself does not necessarily follow the same complexity (Terzidis, 2006). This is part of the beauty of code-based algorithmic design as intricate and interesting designs may emerge that are not impossible to comprehend. Precedent to this concept can be understood through the work of Jeremy Ham (Figures 3.0 & 3.1), notably from his research of musical improvisation and parametric spatial improvisation (Ham, Schnabel, et. al, 2016). Ham uses parametric design tools to explore musical improvisation integration for spatial design. His work is centrally focussed on generating a novel creative process through computational techniques that involve scripting strategies and data (in his case, musical) integration. Within his research, Ham states that “the core intention of many parametric design processes in to push boundaries”; a concept sought for this thesis also.

Scripts/codes are researched, learned and developed as the central part of this thesis with the aim of generating a novel creative process for emergent design outcomes. The scripts are used to define a set of rules with specific data embedment that perform on a variety of set parameters and hence argue as a ‘hyperlocalized’ design outcome. The process of this research, code development, and reflection is documented in the following chapters.
Early research of coding strategies led towards scripting strategies that drew on the algorithmic beauty of nature. One well-known technique found in natural processes is that of branching, conceived as a mathematical theory of plant development which was introduced as ‘Lindenmayer-systems’ (or ‘L-systems’) by Aristid Lindenmayer in 1968. The core idea of L-systems is that of recursion, or rewriting. Basically, rewriting is a method for “defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or productions” (Prusinkiewicz & Lindenmayer, 1990).

Simply put, L-systems are substitution systems; they begin with an initial axiom/seed and follow prescribed rules through the generations of the code. Figure 3.4 demonstrates an exemplar of a rewriting/substitution system that is represented on strings of letters. Figure 3.3 demonstrates further simple rewriting/substitution systems written by Stephen Wolfram. His examples depict the system as coloured boxes in which the rules for the substitution system specify how each box should be subdivided for the subsequent generation (Wolfram, 2002). As evident, these simple set of rules can quite quickly generate a lot of complexity with easy variations available by simply altering the rule(s). The system itself is not complicated but by calling on itself (recursion), complex behaviour develops increasingly through the generational output. These are all adaptations of L-systems and are using the mathematical phenomenon found in nature to model emergent/generative plant/cell growth.
RULES: A = BC; B = AA; C = D; D = CB
AXIOM/SEED: ABCD

Figures 3.4: Rewriting string system

Figures 3.5: Recursive system
Figures 3.6: L-system project by Michael Hansmeyer

Figures 3.7: L-system project by Michael Hansmeyer
The text example in Figure 3.4 can be translated into geometrical conditions from using the same rewriting structure but additionally prescribing drawing rules to the letters (this is also known as ‘turtle graphics’). Through the use of processing language in processor variations of L-systems were written, as seen on page 54 in Figure 3.10. The variations were achieved by simply changing the base rules in one line of the code. Depicted on this page are 4 different scripts formulated as L-systems in Processor. Each have varying rules and axioms and demonstrate their growth through increasing generations.

Shifting into 3D, initially the Grasshopper-Plugin RABBIT was used. RABBIT provides a very basic way to explore natural emergent systems such as L-systems and Cellular Automata in a 3D environment. Figure ( ) illustrates an outcome of this adaptation.

L-systems in architecture have been adopted to model organisational logics, space segmentation and structural systems albeit in a very speculative manner. Most well known is the work of Michael Hansmeyer (Figures 3.6 & 3.7 (Hansmeyer, 2017)). The exploration of L-systems for this research has led to an understanding of emergent behaviours in a computational environment and the development of complexity from simple components. This understanding, although derived from L-systems, extended towards the potentials of recursive structures. L-systems, however interesting, were found to be restrictive in their ability to adapt as an architectural tool, beyond speculative modelling or pattern forming. Learnings from this were carried onto exploring recursive structures beyond just L-systems.

1. turtle graphics are used to assign drawing rules to the system ie forward, left, right, back, up etc (Coates, 2010)

2. Cellular Automata are another type of emergent behaviour found in nature and can be described as a "collection of cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighbouring cells", (Wolfram, 2002).
Figures 3.9: L-system programmed in RABBIT
Figures 3.10: L-system written in Processor. Simple changes to the initial axioms can generate a large range of variation.
Figures 3.11: Animation of a scripted L-system with changing angle parameters within the code every second time step.
Figures 3.12: emergent outcome of another L-system structure inside processor
Figures 3.13: Toyo Ito serpentine pavilion
Algorithmic techniques that involve recursion is an extremely efficacious tool for form generation. A built example involving recursive design strategies can be seen in Toyo Ito’s serpentine pavilion (images 3.13). The rhythmic shapes and structural system were formed from a recursive system of rotated concentric squares.

Recursion, as a broad definition, is a “process of repeating objects in a self similar way” (Reas, McWilliams, et. al, 2010). For example a snowflake is a recursive form; each flake is composed of a series of smaller flakes, which are in fact developed through an L-system structure (see image _) using simple individual components. The principal concept of recursion, and a more technical definition within the context of code, is defined as a function that is able to reference itself as part of the function. To describe this we could say;

Recursion

See “recursion”

This is a dynamic and powerful technique. Designs can quickly develop immense complexity, but it can be difficult to control if the designer does not understand the concept adeptly. As highlighted in the text example above, recursive structures can cause an infinite loop, unless there is a condition to exit the sequence. This is a compelling and powerful process which clearly emulates nature and the concept is applied and developed further on in this research.

Many investigative simulations involving recursive structures were written to explore the possible emergent formations (see images 3.14 - 3.19). This was a pivotal exercise for this research and recursive logics were adopted for the final responsive scripts as documented in the following chapter. L-systems and recursive coding has allowed for the understanding of prescribing rules and parameters into the code which became central for the next stages of this thesis.
Figure 3.14: Recursive structure based off L-system logics
Figure 3.15: Recursive structure emerging through multiple generations, built in Grasshopper through python code.

Figure 3.16: Recursive structure emerged through python code built in Maya.
Figure 3.17 (left column): Recursive structure built in Grasshopper
Figure 3.18 (right column) Emergence built in Maya with python code
Figure 3.19: visualisation of recursive system coded in Grasshopper Python
L-systems and recursive algorithmic structures were explored here to see whether these logics can open up possibilities in the field of architecture. There are strengths and weaknesses to L-systems and recursive logics. There are few studies that have intricately documented the architectural interpretation of L-system based results. While L-system structures could have been examined further for their adaptation as an architectural design tool, it was progressed forward before this stage. Instead, this research proved its value on how characteristics of L-systems translate into constraints and opportunities for architectural design. The constraints proved to be in the systems’ ability to respond to external influences. Although different parameters could be set in regards to generations, angles, axiom, rewriting rules - these are all internal parameters and an overriding script would need to be incorporated after the L-system simulation in order for the emergent form to adapt to external parameters. The opportunities for L-system structures in regards to architecture were found to lie in its ability to serve as an organisational logic system, for the segmentation of space, or the augmentation of a structural system.

Recursive structures, developing on from L-systems rewriting logics, proved to have greater architectural potential. Within the scope of architecture, and as demonstrated through Ito, recursion is an “ontological process that involves the existential formation of infinitely nested structures through self-replication.” (Terzidis, 2006). Such strategies can have a unique structural and aesthetic architectural value. The logics of recursive codes were continued and developed into the final responsive design codes, documented in chapter 4.
ENVIRONMENTS

CHAPTER 4

RESPONSIVE
ENVIRONMENTS

WIND MODELLING / PHYSICAL MODELLING /
CODE 1 / CODE 2 / CODE 3 / FINAL CODE
& RESPONSIVE DESIGN SYSTEM / CRITICAL
REFLECTION
The research in this section first begins by understanding the challenges and opportunities that coding can offer to design, progressing onwards from the research in chapter 3. A few more different scripting processes are explored. The design then looks to act as a hyperlocal generative system that is in response to its immediate environment and is formulated from the locations dominant climatic features. The systems that are coded are tested through the design of a speculative tower within different wind zones of Wellington CBD. The aim of the code is to generate a system that acts responsively to the environmental data and, furthermore, responds locally within the system to all of the programmed variables, generating a discrete system, open to morphological variations.

It is important to note that the following scripts are integrating site specific climatic data (wind), not for the environmental analysis and building performance effects from wind, but simply as a local data set that is specific to a particular site and provides notable variation for improvisation within a novel creative process. The idea is to use a specific data set to investigate the potential of scripting in the design process for generative design methodologies, not for environmental building performance analysis.

The wind climate in Wellington plays a major role in framing social interactions and behaviours throughout the city. It is arguably the windiest city in the world (Windfinder.com, 2016). The wind in Wellington can be considered as an extreme weather condition that is specific to the Wellington context. Although other factors such as air temperature and solar radiation influence human comfort, wind speed is the dominant factor of Wellington's local climate that contributes to human perception of overall comfort. Due to its dominant presence, Wellington's wind is used to embed locally found data into the design process.

The process of using place as catalyst takes the vagaries of concept and plants them into immediate and highly communicative forms that excel in exploring the possibilities
of a hyperlocalized architecture based on its immediate environment. ‘Place’ becomes a powerful driver for understanding the dynamic characteristics of how fully realized environmental architecture manifests in response to existing conditions. The challenges of this environmental shaping can generate the building forms that become an emergent semantic commentary of its place.

To clarify the research intention of this section of the research, it is important to highlight that this work is not concerned in the optimization of form performance in a wind environment nor its technical effect within urban environments - this research has been thoroughly investigated already and can be found elsewhere.

The methodology for simulating the behaviour of the wind and the resulting response to this phenomenon involves two dominant elements; the behaviour of Wellington’s wind and using this wind behaviour as a driver in the design research. The two are intricately linked and by encoding the relationship between the two we can begin to design spaces as hyperlocalized formations that articulate the urban environment within which they belong.

Within the software platform of Maya, there are many dynamic tools for simulating wind. A few of these methodologies are demonstrated below in Figures 4.1, 4.2 and 4.3. Of particular interest for this research, and demonstrated in Figure 4.2, are the nDynamic functions within Maya including nCloth, and nParticles. nCloth has been used to simulate natural forces, in this case wind, using rules of physics that can be applied through the nCloth nucleus. As can be seen on the next page in Figure 4.1, the nCloth is used to generate forms that are driven by the external wind forces and becomes a dynamic object that is used to simulate this naturally occurring climatic phenomenon. In some instances exemplified in Figures 4.1 and 4.3, animation snapshot has been applied to the geometries transformation in order to gain an insight into an extension of this simulation. The nCloth simulations are have been visualised with a simple rectangular plane, initially in a static state. The attributes of the nCloths nucleus are then edited to mimic the state of Wellington’s wind. The wind speed and direction as depicted by current wind readings at windfinder.com/report/wellington, were inserted into these attributes and the nCloth responded accordingly.

The behaviour and subsequently generated forms of wind have an intricate relationship. The generated form affects the behaviour of further wind dynamics in the environments generating the dynamic simulations. The behaviour can be understood as non-linear and context specific. The simulations in Figures 4.0, 4.1, 4.2, 4.3 and 4.5 demonstrate a process of emergence to visualise wind behaviour for further analysis of this novel design language and form generation process.
nCloth uses rules of physics to simulate natural forces. In this instance, setting were programmed to best match Wellingtons wind climate. The ncloth becomes a dynamic object that can be used to simulate this physical phenomena. The nucleus engine allows for the manipulation of some specific material parameters such as stretch resistance, rigidity and mass, enabling the object to mimic real-life material behaviours. These are shown in the adjacent models.

Maya blend shape can animate a smooth transition between any ‘parent’ geometries to create numerous iterations. The iterations become generations that carry different levels of traits from the originating forms. The genotype is the transformation values of each point of the object surface, and the phenotype is the overall object typology. This concept/ evolutionary behaviour was studied briefly and could have been integrated further if the scope had allowed.

Figures 4.1: nCloth simulations
nParticle in Autodesk Maya is a unique tool that can simulate a large number of points that interact with fields and dynamic objects. nParticles become a central tool for the codes that are developed in the following chapter. The images below show early experiments that were run with nParticle simulations.

Maya physics engine can simulate real-world physics for form finding methodologies. The dynamic simulation provides instantaneous geometric form output in response to predefined forces, in this case, wind. The wind in the 3 images below were used as external parameters that transformed the initial geometry into a performance driven form.

Figures 4.2: nparticle and physics simulations
These are further images of experiments that were run whilst beginning to understand the tools necessary for the development of the code integration.

Figures 4.3: Maya simulations
Figure 4.4: experiments of 3D printing
some of these forms from fig 4.0
Figure 4.5: Further trials to experiment and understand particle attributes and effects inside Maya for further use and development
Figure 4.6: Possible initial component designs

Figure 4.7: Evolutionary morphological variations of component design

Figure 4.8: All of the possible secondary generation connection types
PHYSICAL MODELLING
understanding rule-based systems through physical modelling

The evolution and development of biological self-organisation of systems proceeds from small simple components that are assembled together to form larger structures that have emergent properties and behaviour, which in turn self-assemble into more complex structures. This concept was highlighted through the methodology precedents mentioned earlier. To explore this concept for better understanding, physical modelling was exercised. The modelling seeks to understand and demonstrate relationships and behaviours of rule-based form generation. As photographed in images on the following pages, the model is composed of replicable components that morph and adapt as it grows. Through increasing generations, it also gains structural stability. The system is based off simple rules that are defined by the design of the original component; the system can only grow through connecting each component into one of the prescribed connection points. This designed component issues the invariable rules for the system but other variable rules can be defined by the user where patterns can be formed and recursive structures can emerge; this is exemplified through the same model in images in Figures 4.11. The initial component was designed through morphological adaptations of a simple geometry, influenced at first by process discussed in Figures 4.1 and 4.2. This process is depicted Figures 4.6, 4.7 and 4.8.
Figure 4.9: Photographs of the physical model depicting the behaviour of the system as the model grows.
Figure 4.10: Photographs of the physical model
Figure 4.11: Photographs of the physical model
import maya.cmds as cmds
import json
import urllib2

WEATHERURL = 'http://api.openweathermap.org/data/2.5/weather?id=LOCATIONID&APPID=WEATHERAPPID&units=metric'

WEATHERAPPID = 'f27e2e3f7af6d410dbbb22dffc512c5a'

def get_weather_data(location='2179538'):
    try:
        url = WEATHERURL.replace('LOCATIONID', location)
        url = url.replace('WEATHERAPPID', WEATHERAPPID)
        content = urllib2.urlopen(url).read()
        weather_data = json.loads(content)
        print('Weather data loaded for ' + weather_data['name'])
        return weather_data
    except Exception as e:
        print('Could not load weather data for [' + location + '] ' + str(e))

w = get_weather_data()
print(w)

# Isolate wind pressure data
mainWeather = w['main']
print(mainWeather)

airPressure = mainWeather['pressure']
print(airPressure)

airTemp = mainWeather['temp_max']
print(airTemp)

airDensity = (airPressure/287.05 * airTemp)/10
print(airDensity)

windSpeed = w['wind']['speed']
print(windSpeed)

windDegree = w['wind']['deg']
print(windDegree)

windDegreeX = math.cos(windDegree)
windDegreeY = math.sin(windDegree)
print(windDegreeX)
print(windDegreeY)

cmds.nParticle(p=[(0,0,0), (1,1,1), (2,2,2), (3,3,3)])

cmds.setAttr(“nucleus1.windSpeed”, windSpeed)
cmds.setAttr(“nucleus1.windDirectionX”, windDegreeX)
cmds.setAttr(“nucleus1.windDirectionY”, windDegreeY)

if cmds.int(“windSpeed”) >= 8:
    cmds.setAttr(“nucleus1.windNoise”, 1)
else:
    cmds.setAttr(“nucleus1.windNoise”, 0.5)

// SETTINGS
string $pName = “nParticle1”;

// Start and End Times
int $start = 1;
int $end = 100;

// Go to First Frame
currentTime -e $start;

for($j=($start - 1);$j<$end;$j++)
{
    // Store Particle Count
    int $pCount = `getAttr ($pName+”.count”)`;
    
    // If there are particles, continue
    if($pCount > 0)
    {
        // Go through every particle
        for($i=0;$i<$pCount;$i++)
        {
            // Store particles world position
            float $pPosition[] = `getParticleAttr -at worldPosition ($pName + “.pt[” + $i + “]”)`;
            
            // See if there is a curve for the particle
            int $sourceExists = `objExists(“curve_pt_” + $i)”; 
            
            // If no curve exists, make one
            if($sourceExists == 0)
            {
                // Make a curve
                
                // Create the curve
                createInitialCurve(“curve_pt_” + $i);
            }
            
            // If there is a curve, continue
            if($sourceExists == 1)
            {
                // Add a curve point at current particle position
            }
        }
    }

playButtonStepForward;
Creative coding of emergent design is the central process for this research. These next codes have been developed from all of the previous research and from self-taught coding skills. Although not at the level of a skilled programmer, the codes allow for the research to be carried out. Creative coding may be original code written by the programmer/designer, it may be modified code from open-source platforms, it may be a collaboration between multiple coders, or it may be pre-written code that is used in novel ways, all of which were experimented with for this research and documented as follows.

**CODE 1**
see appendix for code

To accurately test and visualise the changing wind behaviour of Wellington, an nParticle system was scripted within the python editor of Maya that embeds live wind data of Wellington City directly into the simulation parameters. Initially, this embedded data was visualized in 2D, as depicted in Figure 4.13. The live data was extracted and applied to the nParticle system within Maya, for the same time frame, as follows; air density affects the number of particles emitted, wind direction dictates the x, y, z directions of these particles, and wind speed determines the particle speed. This simulation was run across a few typical days to demonstrate the different effects of this applied process.

Next, the data was animated within a 3D environment. Using MEL script, the particle motion path can be converted into a NURBS curve. Applications of the processes displayed in Figures 4.12 can be applied to the curves that emerge, and forms that are in response to this live weather data are originated. An example of this process simulated on site is depicted in Figure 4.13. This process allows the data to behave as parameters that can be analysed within the script for more freedom and improvisation in the production of multiple localized design outcomes. This parametric methodology enables the designer to delve into other fields, such as climatology, allowing for a unique interpretation of localized data that generates hyperlocalized resulting forms. As a further study, the generated forms from the local wind behaviours can be analysed in terms of their performance within the wind environment and undertake a simulated wind tunnel analysis to understand how this process of form generation will affect the said built environment that derived their form.

The above outlined design process fuses for the domains of climatology and architecture, not in a scientific manner, but in a creative and form-synthesizing process that is derived from data extracted locally. The resulting artifice is the product of its environment and explicitly unique to its context from the applied parameters, thus arguing as a hyperlocalized typology.
Figure 4.13: 2D visualization of wind data extraction and application as parameters to Maya nParticle system.

Figure 4.14: 3D wind data extraction and embedment as controlling parameters to generate responsive architectural reactions to site specific conditions over one week (12–17 May 2016 at around 18:00 to 21:00).
Figure 4.15: Output from running the script within the boundary constraints of the site. The output can produce infinite variations depending on the wind data at the time of simulation. The above output is the chosen simulation for the development of the system.
Figure 4.16: Various curve connection combinations are possible.
import rhinoscriptsyntax as rs

def allPts(srf):
    border = rs.DuplicateSurfaceBorder (srf)
    lines = rs.ExplodeCurves (border)
    center = rs.SurfaceAreaCentroid ( srf )
    allPts = []
    allPts.append (center[0])
    for line in lines:
        pt = rs.CurveEndPoint (line)
        allPts.append(pt)
    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]
    if (count %3 == 0):
        newObj = rs.OrientObject (obj, source, target2)
    else:
        newObj = rs.OrientObject (obj, source, target1)
    return newObj

def recursiveAggregation (obj, gens, objList, count):
    allSrf = rs.ExplodePolysurfaces (obj)
    pointSet1 = allPts(allSrf[0])
    pointSet2 = allPts(allSrf[21])
    pointSet3 = allPts(allSrf[22])
    pointSet4 = allPts(allSrf[27])
    pointSet5 = allPts(allSrf[32])
    pointSet6 = allPts(allSrf[37])
    pointSet7 = allPts(allSrf[2])
    if (count % 2 == 0):
        newObj = aggregate (obj, pointSet5, count)
        copy = rs.CopyObject (newObj)
        objList.append(copy)
    else:
        newObj = aggregate (obj, pointSet6, count)
        copy = rs.CopyObject (newObj)
        objList.append(copy)
    if (gens > 0):
        recursiveAggregation (newObj, gens-1, objList, count + 1)
        recursiveAggregation (newObj2, gens-1, objList, count + 1)
    else:
        newObj = aggregate (obj, pointSet5, count)
        copy = rs.CopyObject (newObj)
        objList.append(copy)
    if (gens > 0):
        recursiveAggregation (newObj, gens-1, objList, count + 1)
    return objList

allNewObjs = []
count = 0
a = recursiveAggregation (brep, generations,allNewObjs, count)
b = generations

Figure 4.17: Code 2
The next step in this line of inquiry is another code which is a developed progression from code 1 and uses the curves that emerge from that outcome as the basis of the code 2 initiation. This design proposal has now extended to involve multiple towers over the city. This introduces more site variables and better demonstrates the variable outcomes that are possible due to the dependence of these site variables. The chosen sites throughout Wellington city remain speculative but are notably located in different wind zones. This decision means that the differing environmental conditions of each site will directly influence the design which will emerge in response to these immediate conditions. The morphological variations produced from this script can be seen in Figure 4.19 through the location of the tower in differing wind zones.

The algorithmic logic that produces the complex pattern depicted in Figure 4.18 is quite simple: the curves from code 1 are all connected via a series of new curves with their frequency dependent on the wind zone that they are in. Multiple variations are available for the order of connection, as described by solid plane connections in Figure 4.16. This order can be a design decision by the author or, alternatively, another code could be written to connect the curves in a particular order that is dependant on a variable. In this instance, after wide testing, the connection order is decided based on aesthetic reasons and to also minimise curve intersections.

As evident in 4.19a lot of variety is possible. Complexity is also achieved without having to involve complicated geometry or processes. The system remains parametric so that all elements of the design become easily variable and mutually adaptive.
Figure 4.18: Output example from code 2
Figure 4.19: code responds to the site's wind zone location, generating morphogenic changes of the tower forming.
import rhinoscriptsyntax as rs

def allPts(srf):
    border = rs.DuplicateSurfaceBorder(srf)
    lines = rs.ExplodeCurves(border)
    center = rs.SurfaceAreaCentroid(srf)

    allPts = []
    allPts.append(center[0])

    for line in lines:
        pt = rs.CurveEndPoint(line)
        allPts.append(pt)

    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]

    if (count % 3 == 0):
        newObject = rs.OrientObject(obj, source, target2)
    else:
        newObject = rs.OrientObject(obj, source, target1)

    return newObject

def recursiveAggregation(obj, gens, objList, count):
    allSrf = rs.ExplodePolysurfaces(obj)
    pointSet1 = allPts(allSrf[0])
    pointSet2 = allPts(allSrf[21])
    pointSet3 = allPts(allSrf[22])
    pointSet4 = allPts(allSrf[27])
    pointSet5 = allPts(allSrf[32])
    pointSet6 = allPts(allSrf[37])
    pointSet7 = allPts(allSrf[2])

    if (count % 2 == 0):
        newObject = aggregate(obj, pointSet5, count)
        newObject2 = aggregate(obj, pointSet1, count)
        copy = rs.CopyObject(newObject)
        copy2 = rs.CopyObject(newObject2)
        objList.append(copy)
        objList.append(copy2)
        if (gens > 0):
            recursiveAggregation(newObject, gens-1, objList, count + 1)
            recursiveAggregation(newObject2, gens-1, objList, count + 1)
    else:
        newObject = aggregate(obj, pointSet6, count)
        copy = rs.CopyObject(newObject)
        objList.append(copy)
        if (gens > 0):
            recursiveAggregation(newObject, gens-1, objList, count + 1)

    return objList

allNewObjs = []
count = 0
a = recursiveAggregation(brep, generations, allNewObjs, count)
b = generations

Figure 4.20: Code 3
The third code in this line of inquiry is initially developed independent of the previous two codes. This was intentional as the aim at this stage was to develop a separate system that could later be modified with the other system and have one responding to the other, and visa versa, to form an internally responsive environment and inherently discrete system.

This code is a direct progression on from the research completed in the previous chapter. It adopts recursive structures but for the organisational logics of spatial formation. The initial component to adopt for the system was tried and tested with many different designs and eventuated with the adoption of a truncated octahedron. This shape provides 14 faces, each face with point symmetry and can tessellate 3-dimensional space compactly. This means that recursive rules can be written for the connection of the faces while avoiding awkward collisions that appeared through other, less symmetrical components. The truncated octahedron also provides a sizable volume that could be interpreted further on as walls, floors, rooms, furniture etc. Here, the component was chosen to represent different spaces as explained in Figure 4.23 and in more detail in Figure 4.24.

The rules in the code are written to define the connection logics of the system. This means that the designer could, for example, prescribe that every third component was connected to an alternative face and hence branched off in a different direction. Layers of similar rules can be prescribed simultaneously and very quickly a neighbourhood of various sized spaces can emerge. This code is visually explained in Figure 4.22 and the outcome in Figure 4.21. As the generations increase, complexity increases. The system also remains parametric and through simple alterations to the rules in one or two lines of the code, endless variations are possible. Thus the code here makes it inherently easy for the designer to test multiple outcomes and analysis the design through a less labour intensive manner that would have be required if the computational approach had not been used.
Figure 4.21: Output from multiple generations of code 3

```python
import rhinoscriptsyntax as rs

def allPts(srf):
    border = rs.DuplicateSurfaceBorder(srf)
    lines = rs.ExplodeCurves(border)
    center = rs.SurfaceAreaCentroid(srf)
    allPts = []
    allPts.append(center[0])

    for line in lines:
        pt = rs.CurveEndPoint(line)
        allPts.append(pt)

    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]
    if (count % 3 == 0):
        newObject = rs.OrientObject(obj, source, target2)
    else:
        newObject = rs.OrientObject(obj, source, target1)
    return newObject

def recursiveAggregation(obj, gens, obList, count):
    allSrf = rs.ExplodePolysurfaces(obj)
    pointset1 = allPts(allSrf[0])
    pointset2 = allPts(allSrf[21])
    pointset3 = allPts(allSrf[22])
    pointset4 = allPts(allSrf[27])
    pointset5 = allPts(allSrf[32])
    pointset6 = allPts(allSrf[37])
    pointset7 = allPts(allSrf[2])

    if (count % 2 == 0):
        newObject = aggregate(obj, pointset1, count)
        newObject2 = aggregate(obj, pointset2, count)
        copy = rs.CopyObject(newObject)
        copy = rs.CopyObject(newObject2)
    else:
        newObject = aggregate(obj, pointset3, count)
        newObject2 = aggregate(obj, pointset4, count)
        copy = rs.CopyObject(newObject)
        copy = rs.CopyObject(newObject2)

    if (gens > 0):
        recursiveAggregation(newObject, gens - 1, obList, count + 1)
    else:
        return allNewObjs

allNewObjs = []
count = 0
a = recursiveAggregation(brep, generations, out, a)
```
Figure 4.22: Step-by-step visual description of how code 3 works
import rhinoscriptsyntax as rs

def allPts(srf):
    border = rs.DuplicateSurfaceBorder (srf)
    lines = rs.ExplodeCurves (border)
    center = rs.SurfaceAreaCentroid (srf)

    allPts = []
    allPts.append (center[0])

    for line in lines:
        pt = rs.CurveEndPoint (line)
        allPts.append (pt)

    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]

    if (count % 3 == 0):
        newObject = rs.OrientObject (obj, source, target2)
    else:
        newObject = rs.OrientObject (obj, source, target1)

    return newObject

def recursiveAggregation (obj, gens, objList, count):
    allSrf = rs.ExplodePoly SURFfaces (obj)

    pointSET1 = allPts(allSrf[0])
    pointSET2 = allPts(allSrf[2])
    pointSET3 = allPts(allSrf[2])
    pointSET4 = allPts(allSrf[2])
    pointSET5 = allPts(allSrf[2])
    pointSET6 = allPts(allSrf[2])
    pointSET7 = allPts(allSrf[2])

    if (count % 2 == 0):
        newObject = aggregate (obj, pointSET5, count)
        newObject2 = aggregate (obj, pointSET1, count)
        copy = rs.CopyObject (newObject)
        copy2 = rs.CopyObject (newObject2)
        objList.append (copy)
        objList.append (copy2)
    else:
        recursiveAggregation (newObject, gens-1, objList, count + 1)
        recursiveAggregation (newObject2, gens-1, objList, count + 1)

allNewobjs = []
count = 0

a = recursiveAggregation (brep, generations, allNewobjs, count)

return objList
Figure 4.23: Combinations can be programmed to describe various studio compositions and sizes
import maya.cmds as cmds
import json
import urllib2

WEATHERURL = 'http://api.openweathermap.org/data/2.5/weather?appid='
WEATHERAPPID = 'f2762e3f7af6d410dbb2b44d4f2513123a

def get_weather_data(location='2179586',):
    try:
        url = WEATHERURL.replace('LOCATIONID', location)
        url = url.replace('WEATHERAPPID', WEATHERAPPID)
        print(url)
        content = urllib2.urlopen(url).read()
        weather_data = json.loads(content)
        print('weather data loaded for: ', weather_data)
        return weather_data
    except Exception as e:
        print('Could not load weather client data for', e)

w = get_weather_data()
print(w)

# get the wind data only
wind = w['wind']
print wind

# isolate wind pressure data
mainweather = w['main']
print mainweather
airPressure = mainweather['pressure']
print airPressure
airTemp = mainweather['temp_max']
print airTemp

# air density = pressure/287.05^T
import math
airDensity = (airPressure/287.05 * airTemp)**0.5
print airDensity

# create a name for the wind speed and wind degree
windSpeed = wind['speed']
windDegree = wind['deg']
print windSpeed
print windDegree

# convert the wind degree into x and y vectors
import math
windDegreeX = math.cos(windDegree)
windDegreeY = math.sin(windDegree)
print windDegreeX
print windDegreeY
THE CREATIVE CODE & RESPONSIVE DESIGN SYSTEM

Using the skyscraper typology for conceptual research of computational techniques, the resulting form in Figure 4.27 is generated as a combination of two autonomous systems developed from code 1, 2 & 3 above. Exploring the intersection between responding locally to each of the scripts, numerous scenarios emerge that are parametrically controlled and dependant on location and data input. The two separate systems are combined through an additional grasshopper script that directs parameters of one code to respond to parameters of the other code, this is seen in Figure 4.26. The generation of the curves in code one which is relying on the wind zone that the site is located, is now causing the generation of the truncated octahedrons in the other system to respond to the frequency of these curves. The increase of the generation within the systems subsequently increases the complexity whilst always following the same simple rules prescribed in the script. The design has become a discrete system of which its parts cannot be differentiated from the whole as every part relies on all other parts in order to manifest in a certain way, an inherently responsive environment.

This emergent structure is unconceivable without the use of the computational algorithmic design approach. Through use of scripting techniques and relating to the parameters of the site and local data sets, two programmatically independent segments constitute an algorithmic structure capable of growing a recursive system that could be perceived, if desired, as modular housing (see Figure 4.30). The system manifests in response to its environment and as such, systematic, adaptive variation, and dynamic, parametric figuration all contribute towards finding (not making) the tower design outcome.

Figure 4.25: Code output
Figure 4.26: Growth of discrete responsive system
Figure 4.24: System on site

- MARITIME TOWER
  70M

- SCRIPTED WIND TOWER
  200M

- NZ POST OFFICE BUILDING
  ~50M

Figure 4.27: Final code in context of a particular windy site in Wellington CBD
NZ POST OFFICE BUILDING
~50M
Figure 4.28: Height in context with tall Wellington buildings
Figure 4.29: Section of combined systems

Figure 4.30: Depiction of system as housing system
As with most design, it is hard to decide on the end point of a piece of code. Code can continually develop with layers added for complexity and variation. Code, like architecture, is a medium that evolves over time but revisits itself to contextualise and interpret/reflect the current society. It can be original, taken and adapted, or combined; this is the nature of open source and is discussed in the following chapter. The three codes used for this experiment trialled all three of the aforementioned approaches.

Code one is an original script. The result of code one is the starting point of code two. Code three is an adapted code that was originally sourced from an online library (www.theplethoraproject.com). Until the scripts were combined to form a fully responsive and discrete system, each system could be likened to water and oil; when forced to coexist they are unable to bind. However it was not until I had all of the ‘ingredients’ of the individual codes, that I was able to manifest the larger responsive system for the final emergent tower. Similar to Goodwins methodology (chapter 2), I first had to deconstruct all of the parts before being able to construct the whole. Part-to-whole relationships was a prominent topic of the conferences which I attended with the overall consensus that it is the sum of the individual parts that host the exciting moment for designers. This held true for my investigations as once I had merged all of my codes and discovered the behaviour of these systems, I was able to develop an overall environment for all systems to evolve within and produce a novel architectural design.

The outcome is evidently novel yet this system could evolve further to exact
more meaning. The system produces multiple morphological variations and the
decision of which output to use is where this research could potentially go next.
The process is lacking a ‘decision maker’. This could simply be a conscious deci-
sion by the designer based entirely on aesthetics and subjective opinion (as was
true in this case for the final chosen image displayed in Figure 4.27). Ideally, an
optimization process would be developed within the software so that the final
design was appropriated due to various requirements and determining factors.
This would be in keeping with the computational process where the code and
the computer becomes an integral partner in the creative design process, and the
outcome would hold a lot more exact meaning.

The novel-ness of the final design is achieved through the integration and
imbedding of local wind data sets (along with the novel makeup/structure of
the code itself). It is important to note that this could in fact have been any
other type of local data set. The data was used as a driver of the script that the
code could react to. The data was specific to Wellingtons wind and would have
produced a different outcome elsewhere, deeming it a ‘hyperlocal’ outcome of
Wellington. Hyperlocal has become a buzzword in the architectural world (Bul-
livant, 2017) due to the availability of shared data and open-sourced platforms.
The combination of computational strategies, open-source data embedment, and
local information, culminated through code, allowed for the outcome of this
‘hyperlocal’ speculative emergent tower. This wind data could now be used as
the driving factor for the optimization of the ‘decision making’ process; a path
that I would have liked to have included and hope to continue to develop.

As a diversion of the research focus, in an attempt towards this widely sought
focus of pragmatism, the design outcome was pushed into revit and considered
as a conventional design (see Figure 4.31 and 4.32). The generated pod for-
mations were converted into housing designs with the modelled curves being
understood as the structural system to hold up the tower (see Figures 4.29 and
4.30). This step towards pragmatism was viewed to be unsuccessful as it drove
the discussion away from the research intent of this thesis and devalued the
emergent behaviour of the form finding process through creative coding. It sup-
pressed the explorative nature of the system into traditional understandings of
which it cannot compete because, evidently, it is not a practical building (and it
is not intended to be analysed in this way).
The design of which this research has concluded at, pictured in image_, is, as continually reiterated, focussed predominantly on the design process and creative coding approach, however, it is this computational approach that is capable of producing uniquely creative and novel designs. Novelty must be the result of discovery which is possible through the codes and systems described in this research. Designers must exploit the potentials that computational design systems can offer the discourse of architecture and laterally take advantage of advancing building technologies in order to keep pushing the boundaries of what architecture can offer society, and reinstate the relevance of an architect’s role in the 21st century.
Figure 4.31: Elevations produced within Revit
Figure 4.32: System modelled inside Revit
5.0
OPEN SOURCE / AUTHORSHIP IN THE DIGITAL REALM / CRITICAL REFLECTION & CONCLUSION
Figure 5.0: Output from explorations of L-systems inside Grasshopper
The nature of computational architecture is feeding a rapidly growing online databases of code libraries. Codes written by computer scientists, gamers or any type of coder have always been open-source and readily accessible for use and customization. The last AD publication addresses this issue with stimulating discussions (Fok, Picon et al., 2016).

The status of architectural design is changing/profoundly due to the current digital movement. The design process, as documented in earlier chapters, has vastly advanced and designs are increasingly opening up to endless possibilities. The result of these designs and their corresponding scripts are, naturally, in digital format, and hence can be indefinitely circulated. This prompts popular discussions around authorship, plagiarism and digital circulation. Designs are travelling at instantaneous speeds and are constantly shared and customised at will to adapt to specific project requirements and author desires - so if a designer uses someone else’s script, who is the author? Does one need to reference them if they alter a line in the code? Even the tiniest of changes in a large script can cause drastic changes to the output and become completely unique to the way the script was originally intended for use. BIM (building information modelling) is a great example of open-source architectural systems and management.

Through the ever-increasing appreciation and contribution towards open-source architecture, digital architecture is gaining momentum as effective, efficient and endlessly exciting for the architectural profession. A truly open-source architecture means the voluntary sharing of ideas, software, scripts, and models. The possibilities that this would open-up to designers, and the advancement of the architectural discourse, is immense but the nature of intellectual property that exists in practice today bodes an obstacle as to whether this possibility will ever become a reality. (Fok, Picon et al., 2016).
The research carried out in this thesis has relied on the status of open-source platforms which are readily accessible to anyone with access to the internet. It is common practice amongst coders to share scripts online that one writes, with the hope that others will use them, share them, and make use of them. Furthermore, there are constantly users online who are ready and willing to help with any bugs in one’s script, answer any questions about how to make codes more efficient, achieve something specific, or offer alternative solutions. The world becomes connected with the common goal of the advancement of knowledge and design.

Scripts that I have written as part of this research have been shared into this open-source database. My scripts are open to now being used and altered by others, without my knowledge, and advanced on towards further discoveries - this is the beauty of open-source architecture.
Over the course of one project, or even just concept design phase, there could potentially be a multiplication of contributors, depending on the origins of the code. This halts the momentum in validating open-source architecture as fair retribution to these contributors becomes an issue. It is an issue because it is often impossible to find out who made changes to which part of which script and what actually was the original script and author. Sharing data files is so easy and accelerated that this information is lost by the time one goes to modify it themselves.

As recorded for the codes used for the system created in this research, parts of the codes were obtained, and then altered, from open-source libraries, whilst parts of the code were original. It is common practise to publish ones codes online (or otherwise) so that their codes may be further developed by other designers, nurturing the growth and possibilities of creative coding in design. As such, my codes are available for anyone to use (as a whole, as parts, or for development) so that I too am contributing to the digital realm. The codes can be found in the appendix and at www.vanvelthooven.org.

Another issue regarding authorship is that between the architect and the computer. When the outcome is not conceived in the designer’s mind, are they still the creator of the design produced? This is a common concern amongst those engaged with the digital realm of design as the role of the designer is somewhat altered, and has become popular criticism to computational based design. However, as noted by architectural programmer Dr. Milos Dimicic, “the creative processes is enriched and informed by computational tools, but those do not substitute the role of the creative mind”. (ProgramArchitecture, 2013). The designers creative input is now focussed on the relationships and behaviours that make up the script, rather than the final product. This is, as mentioned right at the beginning of this work, the difference between a top-down (conventional) and bottom-up (scripted) method of design. The later providing greater affordance to novel and meaningful creative solutions and designed complexity.
CRITICAL REFLECTION & CONCLUSION

of entire research and works

The design through research presented here has manifested a responsive environment through creative coded systems to generate a hyperlocal tower of evolved complexity. The design has been reflected on in terms of its novelty and meaning. Vast research, and various stages of learning and exploration, was necessary to arrive at the outcome of this research.

Processes that draw on nature, such as L-systems and recursive structures, were a critical explorative research stage of this thesis. They were both important steps for discovering emergent systems and logics. Although L-systems were proven to be too restrictive in their behaviour, the evolving nature of the system and the structure of the code setup were platforms for the progression of the research. Recursive systems were developed further as these simple logics are able to generate vast complexity and also can be integrated to respond with other parameters for morphological variation. These researched processes for a responsive emergent system were explored through the tower typology and through using a local climatic dataset (wind). However this typology and dataset could easily have been something different - it is just a notion created to test the system. This means that the system’s ability to express this notion is the focus of the research, rather than the notion itself.

The creative codes written for this tower design, some original and some developments from open-source libraries, were the fundamental tool for the emergent design; ie the outcome could not have been conceived of without the scripted system in place. The drivers that allowed for configurable differentiation within the system were based on site specific climatic data sets. Wellington’s dominant climatic element, the wind, was extracted from online and directly embedded into the system. The system was run across
multiple sites in Wellington in order to test and analyse the variegation.

The system not only performed in response to the site specific wind data, but also to a second script that was integrated as a sub-system. This sub-system grew as the generations increased, and the generational increments were dependant on the scale of the wind on site. Linking all of the parameters of the system to rely on one another, generates a discrete system - capacitated entirely through computation.

Thinking about architectural design in a computational manner, means that one has to think about every element or part as a piece of computable data within the system, commonly referred to as part to whole relationships. This is what it means to be a ‘discrete’ system and is what is sought for the codes that generate the final tower designs from the research carried out. Many implications in regards to fabrication, authorship, and process are generated through computational design thinking however, progressive creativity and the desire for unique aesthetics drive a fundamental change to how architecture is traditionally produced and procured. There is vast opportunity for the continuation of the research documented within this thesis to explore these implications and develop the system further, especially in terms of optimisation.

Computational design through the writing and rewriting code plays a fast increasing role in architecture. It is central to innovative design, allowing for previously unconceivable complexities to emerge. A lot of what is presented in this research is not new to the discourse of architecture, but for me it was a very new learning process and, in the context of digital literacy, I realised that codes and computational algorithms can be continually revisited and developed for novel and meaningful designs. Furthermore, to move forward and to participate in the digital shift that has taken ahold of the architectural profession, one has to first understand what digital approach means and the processes possible, in order to apply it to a new context. With differing contexts and quickly developing softwares and resources, endless different possibilities emerge. This means that digital design and creative coding in design will remain relevant, and likely grow in relevance, and thus places value on the undertaking of this research for future growth as a designer. This research provides a valuable depiction of a novel design approach and meaningful outcome that aligns with the current status of progressive architectural intrigue, accordanant of the digital age that we are currently living.
LIST OF FIGURES

All figures are created by the author unless otherwise stated

Cover Image: Own digital drawing created in grasshopper script

Figure 1.0: L-system simulation, digital drawing.

Figure 2.0: Emergence by Kokkugia

Figure 2.1: Xi An - Competition proposal project by Richard Goodwin

Figure 2.2: Xi An - Competition proposal project by Richard Goodwin

Figure 2.3: Khaleesi tower

Figure 2.4: Wind Tower by Toyo Ito

Figure 2.5: Heights of the top 20 tallest buildings in the world, digital drawing.
Figure 2.6: Heights of the top 20 tallest buildings in New Zealand, digital drawing.
Figure 2.7: Heights of the top 20 tallest buildings in Canada, digital drawing.

Figure 2.8: _a_maze_ by Alisa Andrasek


Figure 2.9: Project Bloom


Figure 2.10: AADRL studio Spyropoulos project


Figure 2.11: Kokkugia project


Figure 2.16: source: stuff.co.nz,. (2009). Windy Wellington lives up to its reputation. Retrieved from http://www.stuff.co.nz/life-style/blogs/are-we-there-yet/1881632/Windy-Wellington-lives-up-to-its-reputation

Figures 2.18: Me and Patrik Schumacher at the PACT conference in London, September 2016, photograph

Figures 3.0 & 3.1: Work by Jeremy Ham

Figures 3.2: Rewriting system by Stephen Wolfram

Figures 3.3: images of patterns found in nature
Source: all images sourced from google search ‘patterns in nature’.

Figures 3.4: Rewriting string system, digital drawing
Figures 3.5: Recursive system

Figures 3.6 & 3.7: L-system project by Michael Hansmeyer

Figures 3.8: diagram of l-system rewriting logic, digital drawing
Figures 3.9: L-system programmed in RABBIT, digital drawing.
Figures 3.10: L-system written in Processor. Simple changes to the initial axioms can generate a large range of variation, digital drawing.
Figures 3.11: Animation of a scripted L-system with changing angle parameters within the code every second time step, digital drawing.
Figures 3.12: Emergent outcome of another L-system structure inside processor, digital drawing.
Figures 3.13: Toyo Ito serpentine pavilion

Figures 3.14: Recursive structure based off L-system logics, digital drawing.
Figures 3.15: Recursive structure emerging through multiple generations, built in Grasshopper through python code, digital drawing.
Figure 3.16: Recursive structure emerged through python code built in Maya, digital drawing.
Figure 3.17: Recursive structure built in Grasshopper, digital drawing.
Figure 3.18: Emergence built in Maya with python code, digital drawing.
Figure 3.19: Visualisation of recursive system coded in Grasshopper Python, digital drawing.

Figure 4.0: Simulations in Maya using CFD and Maya physics engine to simulate wind and the forming of objects due to this wind factor, digital drawing.
Figure 4.1: nCloth simulations, digital drawing.
Figure 4.2: nparticle and physics simulations, digital drawing.
Figure 4.3: Maya simulations, digital drawing.
Figure 4.4: Experiments of 3D printing some of these forms from fig 4.0, digital drawing.
Figure 4.5: Further trials to experiment and understand particle attributes and effects inside Maya for further use and development, digital drawing.
Figure 4.6: Possible initial component designs, digital drawing.
Figure 4.7: Evolutionary morphological variations of component design, digital drawing.
Figure 4.8: All of the possible secondary generation connection types, digital drawing.
Figure 4.9: Photographs of the physical model depicting the behaviour of the system as the model grows, photograph.
Figure 4.10 & 4.11: Photographs of the physical model, photograph.
Figure 4.12: Code 1, digital drawing.
Figure 4.13: 2D visualization of wind data extraction and application as parameters to Maya nParticle system, digital drawing.
Figure 4.14: 3D wind data extraction and embedment as controlling parameters to generate responsive architectural reactions to site specific conditions over one week (12-17 May 2016 at around 18:00 to 21:00), digital drawing.
Figure 4.15: Output from running the script within the boundary constraints of the site. The output can produce infinite variations depending on the wind data at the time of simulation. The above output is the chosen simulation for the development of the system, digital drawing.
Figure 4.16: Various curve connection combinations are possible, digital drawing.
Figure 4.14: Code 2, digital drawing.
Figure 4.15: Output example from code 1, digital drawing.
Figure 4.16: Code responds to the sites wind zone location, generating morphogenic changes of the tower forming, digital drawing.
Figure 4.17: Code 2, digital drawing.
Figure 4.18: Output from multiple generations of code 2, digital drawing.
Figure 4.19: code responds to the sites wind zone location, generating morphogenic changes of the tower forming, digital drawing.
Figure 4.20: Code 3, digital drawing.
Figure 4.21: Output from multiple generations of code 3, digital drawing.
Figure 4.22: Step-by-step visual description of how code 3 works, digital drawing.
Figure 4.23: combinations can be programmed to describe various studio compositions and sizes, digital drawing.
Figure 4.24: Detailed depiction of how these spaces may be designed as housing, digital drawing.
Figure 4.25: Code output, digital drawing.
Figure 4.26: Growth of discrete responsive system, digital drawing.
Figure 4.27: Final code in context of a particular windy site in Wellington CBD, digital drawing.
Figure 4.28: Height in context with tall Wellington buildings, digital drawing.
Figure 4.29: Section of combined systems, digital drawing.
Figure 4.30: Depiction of system as housing system, digital drawing.
Figure 4.31: Elevations produced within Revit, digital drawing.
Figure 4.32: System modelled inside Revit, digital drawing.
Figure 5.0: Output from explorations of L-systems inside Grasshopper, digital drawing.


Campo, M., McMorrough, J., Gage, M., Saunders, A., Young, M., & Roche, F. et al. (2016). *AD Journal: Evoking through design* (1st ed.). Print


APPENDIX

CODE 1

```python
import maya.cmds as cmds
import json
import urllib2

WEATHERURL = 'http://api.openweathermap.org/data/2.5/weather?id=LOCATIONID&APPID=WEATHERAPPID&units=metric'
WEATHERAPPID = 'f27e2e3f7af6d410dabb22dffe512c5a'

def get_weather_data(location='2179538'):
    try:
        url = WEATHERURL.replace('LOCATIONID', location)
        url = url.replace('WEATHERAPPID', WEATHERAPPID)
        print(str(url))
        content = urllib2.urlopen(url).read()
        weather_data = json.loads(content)
        print('Weather data loaded for ' + weather_data['name'])
        return weather_data
    except Exception as e:
        print('Could not load weather client data for [' + location + ']' + str(e))

w = get_weather_data()
print(w)

# gets the wind data only
wind = w['wind']
print(wind)

# isolate wind pressure data
mainWeather = w['main']
print(mainWeather)

airPressure = mainWeather['pressure']
print(airPressure)

airTemp = mainWeather['temp_max']
print(airTemp)

# air density = pressure/287.05*T
import math
airDensity = (airPressure/287.05 * airTemp)/10
```

135
print airDensity

windSpeed = wind['speed']
windDegree = wind['deg']

print windSpeed
print windDegree

# convert the wind degree into x and y vectors
import math
windDegreeX = math.cos(windDegree)
windDegreeY = math.sin(windDegree)

print windDegreeX
print windDegreeY

cmds.nParticle(p= [(0,0,0), (1,1,1), (2,2,2), (3,3,3)] )

cmds.setAttr ("nucleus1.windSpeed", windSpeed)
cmds.setAttr ("nucleus1.windDirectionX", windDegreeX)
cmds.setAttr ("nucleus1.windDirectionY", windDegreeY)
cmds.setAttr ("nucleus1.gravity", 0)
cmds.setAttr ("nucleus1.airDensity", airDensity)

if cmds.int('windSpeed') >= 8:
    cmds.setAttr ("nucleus1.windNoise", 1)
else:
    cmds.setAttr ("nucleus1.windNoise", 0.5)

}
// Make a curve
rename $initialCurve ("curve_pt_" + $i);
}

// If there is a curve, continue
if($curveExists == 1)
{
// Add a curve point at current particle position
}
}
}
playButtonStepForward;

import rhinoscriptsyntax as rs

def allPts(srf):
    border = rs.DuplicateSurfaceBorder (srf)
    lines = rs.ExplodeCurves (border)
    center = rs.SurfaceAreaCentroid ( srf )
    allPts = []
    allPts.append (center[0])
    for line in lines:
        pt = rs.CurveEndPoint (line)
        allPts.append(pt)

    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]
    if (count % 3 == 0):
        newObject = rs.OrientObject (obj, source, target2)
    else:
        newObject = rs.OrientObject (obj, source, target1)
    return newObject

def recursiveAggregation (obj, gens, objList, count):
    allSrf = rs.ExplodePolysurfaces (obj)
    pointSet1 = allPts(allSrf[0])
    pointSet2 = allPts(allSrf[21])
    pointSet3 = allPts(allSrf[22])
    pointSet4 = allPts(allSrf[27])
    pointSet5 = allPts(allSrf[32])
    pointSet6 = allPts(allSrf[37])
    pointSet7 = allPts(allSrf[2])
    if (count % 2 == 0):
        newObject = aggregate (obj, pointSet5, count)
        newObject2 = aggregate (obj, pointSet1, count)
        copy = rs.CopyObject (newObject)
        copy2 = rs.CopyObject (newObject2)
        objList.append(copy)
        objList.append(copy2)
    if (gens > 0):
recursiveAggregation (newObject, gens-1, objList, count + 1)
recursiveAggregation (newObject2, gens-1, objList, count + 1)

else:
    newObject = aggregate (obj, pointSet6, count)
    copy = rs.CopyObject (newObject)
    objList.append (copy)
    if (gens > 0):
        recursiveAggregation (newObject, gens-1, objList, count + 1)

return objList

allNewObjs = []
count = 0
a = recursiveAggregation (brep, generations, allNewObjs, count)
b = generations

CODE 3

import rhinoscriptsyntax as rs
def allPts(srf):
    border = rs.DuplicateSurfaceBorder (srf)
    lines = rs.ExplodeCurves (border)
    center = rs.SurfaceAreaCentroid (srf)

    allPts = []
    allPts.append (center[0])
    for line in lines:
        pt = rs.CurveEndPoint (line)
        allPts.append (pt)

    return allPts

def aggregate(obj, pointList, count):
    source = [pointList[0], pointList[1], pointList[2]]
    target1 = [pointList[0], pointList[2], pointList[1]]
    target2 = [pointList[0], pointList[3], pointList[2]]

    if (count % 3 == 0):
        newObject = rs.OrientObject (obj, source, target2)
    else:
        newObject = rs.OrientObject (obj, source, target1)

    return newObject

def recursiveAggregation (obj, gens, objList, count):
    allSrf = rs.ExplodePolysurfaces (obj)
    pointSet1 = allPts(allSrf[0])
    pointSet2 = allPts(allSrf[21])
    pointSet3 = allPts(allSrf[22])
    pointSet4 = allPts(allSrf[27])
    pointSet5 = allPts(allSrf[32])
    pointSet6 = allPts(allSrf[37])
    pointSet7 = allPts(allSrf[2])

    if (count % 2 == 0):
        newObject = aggregate (obj, pointSet5, count)
        newObject2 = aggregate (obj, pointSet1, count)
copy = rs.CopyObject (newObject)
copy2 = rs.CopyObject (newObject2)
objList.append(copy)
objList.append(copy2)
if (gens > 0):
    recursiveAggregation (newObject, gens−1, objList, count + 1)
    recursiveAggregation (newObject2, gens−1, objList, count + 1)
else:
    newObject = aggregate (obj, pointSet6, count)
copy = rs.CopyObject (newObject)
objList.append(copy)
    if (gens > 0):
        recursiveAggregation (newObject, gens−1, objList, count + 1)
return objList

allNewObjs = []
count = 0
a = recursiveAggregation (brep, generations, allNewObjs, count)
b = generations