ARCHITECTURAL OBJECT-STATES:
THE CRISIS OF THE ARCHITECTURAL OBJECT

by

Kyrylo Lobach
Bachelor of Architectural Science, Ryerson University, 2009

A Thesis
presented to Ryerson University

in partial fulfillment of the
requirements of the degree of
Master of Architecture
in the Program of Architecture

Toronto, Ontario, Canada, 2013
© Kyrylo Lobach, 2013

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Kyrylo Lobach

Architectural Object-States: The Crisis of The Architectural Object
Master of Architecture, 2013
Kyrylo Lobach
Program of Architecture, Ryerson University

Design computing is the architectural object's first step towards acquiring an architectural objectile.

Abstract

Design computing is concerned with general conditions deriving a specific result. This is done through an explicit design of process. This results in a shift from design process to process design. Process design describes design through series of algorithms together forming a notion of an objectile — an incomplete general yet specific notation capable of generating a family of objects or object-states. In a practice that embraces process design, the creation of custom algorithmic tools (both virtual and actual) and the design of architectural objects, will be augmented with the design of architectural objectiles and object-states.

Acknowledgements

This thesis would have not been possible without the continuous support and encouragement of John Cirka, Marco Polo, Vincent Hui, Emma Lee, Ryerson International Fund and my family.

John Cirka has been as engaged in my topic as I have been. I would like to thank him for finding time to turn one hour meetings into two hours and sharing his vast and varied knowledge with me. Ultimately this thesis is far richer because of that. Our discussions ranging from architectural history, design computing, theoretical physics, creativity and its computability will be missed. I hope we will both find time to continue our conversations. I would like to thank Marco Polo for providing me with a broad perspective and connections in my thesis topic to other architectural concerns. Thank you for questioning and reminding me of important questions. I would like to thank Vincent Hui for explaining and guiding me to clarity in my thought and a clear thesis defence. Thank you – Emma Lee for calling me out on my sometimes ridiculous ideas and your unconditional support and encouragement. I definitely know this thesis would not have been possible without you. I would like to thank Ryerson International Fund for generous financial support that made attending the ACADIA 2011 conference possible; it kick-started my research in the right direction. Last but not least I would like to thank my family for unconditional support and understanding in anything I mind to tackle. Thank you.

Table of Contents

1	Object and Objectile	1
1.1	Computational design	1
1.2	Variation	2
1.3	Building the actual	4
	Interlude I: Defining an Objectile	8
2	Computers and tool making	14
2.1	Before computers	14
2.2	Modern computers	19
2.3	User Interfaces and Software	20
2.4	Tool Making	24
2.5	The three categories	31
	Interlude II: PixelFacade – Objectile of a small building and its variation	38
3	Boxes, blobs and curves; computational context	42
3.1	The extent of things	42
3.2	Contemporary scene	48
3.3	Next?	52
	Interlude III: City Blocks as Generic Conditions?	58
4	Programming and algorithms	62
4.1	Natural and artificial languages	62
4.2	Computability	63
4.3	Scripting and programming languages	68
4.4	Algorithms	69
	Interlude IV: myCabin – sketch, draw, model & compose	76
5	Auto/Novel	85
6	Virtual/Actual and Static/Dynamic	92

7	Appendix A – Additional Project Images	144
8	Appendix B — Geometry generation process	150
9	Appendix C — Source Codes	170
10	Bibliography	183

List of Figures

Figure 1: NOX 'myLight' project.

Source: <http://www.nox-art-architecture.com/>

Figure 2: Relations between a proto-objectile, objectile and object-state (body) through the example of a tree.

Source: Kyrylo Lobach

Figure 3: CONTINUUM a customer designed dress from previously pre-designed objectile.

Source: <http://www.continuumfashion.com/D.php>

Figure 4: SIM Residence by Sean Lally

Source: <http://www.weathers.cc/projects.html>

Figure 5: Overlay of human portraits

Source: Kyrylo Lobach

Figure 6: Example of LTYL population of possibilities. Illustrated in plan view.

Source: Kyrylo Lobach

Figure 7: Example of LTYL population of possibilities. Illustrated in perspective view.

Source: Kyrylo Lobach

Figure 8: First step in LTYL generation; program asked for initial parameters

Source: Kyrylo Lobach

Figure 9: Second step in LTYL generation; program asked for secondary parameters

Source: Kyrylo Lobach

Figure 10: Resulting geometry of LTYL processes.

Source: Kyrylo Lobach

Figure 11: Possible outcome of the Garden Wall process

Source: Kyrylo Lobach

Figure 12: Snippets of the three GH codes

Source: Kyrylo Lobach

Figure 13: Detail of Charles Babbage 'Difference Engine' replica.

Source: http://mallorea.student.utwente.nl/~jorg/Photos/Japan/Post31/London/IMG_0077crop%20Part%20of%20Difference%20engine%20

No2%20as%20designed%20by%20Babbage.JPG

Figure 14: Jacques de Vaucanson duck automaton.

Source: <http://howstuffworks.files.wordpress.com/2011/06/duck1.jpg>

Figure 15: Punch cards of Jacquard Loom

Source: <http://www.flickr.com/photos/kradeki/6131571845/>

Photo by Agnes Chang, flickr user name: kradeki

Figure 16: Light pen input device

Source: <http://images.yourdictionary.com/light-pen>

Figure 17: Mac and PC Television Advertisement

Source: <http://4.bp.blogspot.com>

Figure 18: Raspberry Pi Open-source Computer

Source: <http://awards.t3.com/categories/innovation-of-the-year/raspberry-pi>

Figure 19: 'Descriptio Urbis Romae' Replica

Source: Carpo, M. (2011). The alphabet and the algorithm. Cambridge, MA: MIT Press.

Figure 20: Physical Tool. Left to right order – Phantom Geometry, super-KUKA, 5-Axis Motion and VPRP

Sources:

(1) <http://www.archdaily.com/284752/sci-arc-gehry-prize-awarded-to-phantom-geometry/>

(2) <http://cargocollective.com/fabroboticsnet/SuperKUKATools>

(3) <http://www.liftarchitects.com/>

(4) Oxman, N. (2010). Material-based Design Computation (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Figure 21: Research projects by Gramazio & Kohler. From left to right FOAM, Clay Molding and Procedural Landscapes 2

Source: <http://www.dfab.arch.ethz.ch/web/e/lehre/211.html>

Figure 22: Variable Property Rapid Prototyping (VPRP) by Neri Oxman

Source: Oxman, N. (2010). Material-based Design Computation (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Figure 23: Timeline of notable digital tool makers

Source: Kyrylo Lobach

Figure 24: Ivan Sutherland using 'Sketchpad'

Source: <http://design.osu.edu/carlson/history/images/ivan-sutherland.jpg>

Figure 25: Left: Induction Cities design processes. Right: On-Demand City digital tool (1995) by Makoto Sei Watanabe

Source: Watanabe, M. S. (2002). Induction design: A method for evolutionary design. Basel: Birkhäuser.

Figure 26: mTable object-states

Source: <http://www.gramaziokohler.com/web/e/projekte/17.html>

Figure 27: Top row left to right: Crumple, Drift and Grid. Bottom row left to right: MITList Without Out, Net and Onthevergeofcollapse

Source: <http://www.mos-office.net/>

Figure 28: Turing Pavilion printed model and digital tool by Biothing

Source: <http://www.biothing.org/?cat=24>

Figure 29: Prizma by Biothing, an objectile of housing complex

Source: <http://www.biothing.org/?cat=26>

Figure 30: PixelFacade by author. Showcasing different objectile parameters and GUI

Source: Kyrlo Lobach

Figure 31: John Hancock Tower by Henry Cobb

Source: <http://openbuildings.com/buildings/john-hancock-tower-project-file-6686#>

Figure 32: Peter Eisenman House Transformations

Source: http://4.bp.blogspot.com/-let-UH1Ze_g/TVZNmEQr62I/AAAAAAAAABH0/AiPt0tC3mVw/s1600/eisenman+house+IVc.jpg

Figure 33: NOX Water Pavilion

Source: http://www.nox-art-architecture.com/NOXARCH/Projects/Project%20Images/High%20Res/01_h2o_1.jpg

Figure 34: Design Computing Multi-Disciplinary Areas of Concern

Source: Kyrlo Lobach

Figure 35: Puppet Theater by MOS Office

Source: <http://www.mos-office.net/>

Figure 36: Voussoir Cloud by Iwamoto Scott

Source: <http://www.iwamotoscott.com/>

Figure 37: TMV by Theverymany

Source: <http://theverymany.com/>

Figure 38: Shenzhen Airport by Massimiliano Fuksas

Source: <http://www.fuksas.it/#/progetti/1405/>

Figure 39: 0-14 by Rieser Umemoto

Source: <http://i46.tinypic.com/2iu5ac0.jpg>

Figure 40: Hilton Pattaya by Department of Architecture

Source: <http://www.archdaily.com/119316/hilton-pattaya-department-of-architecture/>

Figure 41: ICD + ITKE 2010 Research Pavilion lead by Achim Menges

Source: <http://icd.uni-stuttgart.de/?p=4458>

Figure 42: Embryological Houses by Greg Lynn

Source: <http://www.iaacblog.com/digitalfabrication/files/2010/10/E2.jpg>

Figure 43: From top left to right: Java programming, RhinoSCRIPT scripting, Grasshopper graph

Source: Kyrylo Lobach

Figure 44: Toronto Urban Fabric

Source: Kyrylo Lobach

Figure 45: Montreal Urban Fabric

Source: Kyrylo Lobach

Figure 46: San Francisco Urban Fabric

Source: Kyrylo Lobach

Figure 47: Houston Urban Fabric

Source: Kyrylo Lobach

Figure 48: Washington D.C. Urban Fabric

Source: Kyrylo Lobach

Figure 49: New York City Urban Fabric

Source: Kyrylo Lobach

Figure 50: London Urban Fabric

Source: Kyrylo Lobach

Figure 51: Paris Urban Fabric

Source: Kyrylo Lobach

Figure 52: Barcelona Urban Fabric

Source: Kyrylo Lobach

Figure 53: Python IDE and API (left column) inside Rhinoceros Modeling Software

Source: Kyrylo Lobach

Figure 54: Grasshopper interface inside Rhinoceros modeling software

Source: Kyrylo Lobach

Figure 55: Depiction of algorithms in cinema- a scene from the movie The Social Network

Source: Social Network. Dir. David Fincher. Perf. Jesse Eisenberg, Andrew Garfield, Justin Timberlake. Columbia Pictures, 2010. DVD.

Figure 56: Overlay of Western Dwellings- A Housing Objectile?

Source: Kyrylo Lobach

Figure 57: ICD 2012 Research Pavilion

Source: Correa, D. (2012, September 2). Google + profile video [ICD 2012 Pavilion Construction video]. Stuttgart.

Figure 58: The Truffle by Ensamble Studio. Hay Assembly and Concrete Casting

Source: <http://www.ensamble.info/actualizacion/projects/truffle#>

Figure 59: The Truffle by Ensamble Studio. Cutting and Excavating

Source: <http://www.ensamble.info/actualizacion/projects/truffle#>

Figure 60: The Truffle by Ensamble Studio. Finished dwelling

Source: <http://www.ensamble.info/actualizacion/projects/truffle#>

Figure 61: Lakeshore cabin (left); Woods cabin (right)

Source: Kyrylo Lobach

Figure 62: Study of Lakeshore Cabin. Preliminary Plan and Sectional Sketches

Source: Kyrylo Lobach

Figure 63: Studies of Lakeshore Cabin.

Source: Kyrylo Lobach

Figure 64: Lakeshore Cabin Material Scheme and Setting.

Source: Kyrylo Lobach

Figure 65: Lakeshore cabin material scheme and setting

Source: Kyrylo Lobach

Figure 66: Possible myCabin outcomes

Source: Kyrylo Lobach

Figure 67: Algorithmic (Graph) definition of myCabin objectile

Source: Kyrylo Lobach

Figure 68: PixelFacade Perspectives

Source: Kyrylo Lobach

Figure 69: Daylighting Factor App

Source: Kyrylo Lobach

Figure 70: Garden Wall project. Relationship between existing surface mapping and louver mesh

Source: Kyrylo Lobach

Figure 71: Onthevergeofcollapse by MOS

Source: <http://www.mos-office.net/>

Figure 72: Turing Pavilion object-state by Biothing

Source: <http://www.biothing.org/?cat=24>

Figure 73: Composition of an objectile

Source: Kyrylo Lobach

Figure 74: Composition of Rhizome Pavilion objectile

Source: Kyrylo Lobach

Figure 75: Generic Site. Perspective.

Source: Kyrylo Lobach

Figure 76: Generic Site. Plan.

Source: Kyrylo Lobach

Figure 77: Generic Site. Perspectives and plans.

Source: Kyrylo Lobach

Figure 78: Screen-shots of site model with dynamic site mapping algorithm applied

Source: Kyrylo Lobach

Figure 79: Sketches of Orange pavilion belonging to trees area objectile

Source: Kyrylo Lobach

Figure 80: Overall relationship mapping diagram

Source: Kyrylo Lobach

Figure 81: Program Mapping diagram

Source: Kyrylo Lobach

Figure 82: Program mapping process

Source: Kyrylo Lobach

Figure 83: Relaxation Woods static models. Used throughout design

Source: Kyrylo Lobach

Figure 84: Orange Woods. Possible object-state

Source: Kyrylo Lobach

Figure 85: Orange Woods Objectile. Graph algorithm of massing

Source: Kyrylo Lobach

Figure 86: Variation within object-states of same objectile

Source: Kyrylo Lobach

Figure 87: Physical Model of Two Dimensional matrix of possible object-states

Source: Kyrylo Lobach

Figure 88: Rhizome – Wood Pavilion objectile diagram

Source: Kyrylo Lobach

Figure 89: Object-state location based variations

Source: Kyrylo Lobach

Figure 90: Structure studies

Source: Kyrylo Lobach

Figure 91: Initial structure and aperture openings algorithmic order
Source: Kyrylo Lobach

Figure 92: First algorithmic generation physical model
Source: Kyrylo Lobach

Figure 93: Second algorithmic generation. Structure and apertures.
Source: Kyrylo Lobach

Figure 94: Third algorithmic generation. Structure and apertures.
Source: Kyrylo Lobach

Figure 95: Third algorithmic generation. Apertures and Inner/Outer paneling.
Source: Kyrylo Lobach

Figure 96: Third algorithmic generation. Apertures and Inner/Outer paneling.
Source: Kyrylo Lobach

Figure 97: Third generation algorithmic pavilions. Location and designer based variation.
Source: Kyrylo Lobach

Figure 98: Third generation algorithmic pavilions. Sectional perspective. Three variations showing different experiential qualities.
Source: Kyrylo Lobach

Figure 99: Third generation algorithmic pavilions. Sectional perspective. Pavilion with moderate amount of apertures
Source: Kyrylo Lobach

Figure 100: Third generation algorithmic pavilions. Sectional perspective. Pavilion with large amount of apertures
Source: Kyrylo Lobach

Figure 101: Third generation algorithmic pavilions. Sectional perspective. Pavilion with minimum amount of apertures
Source: Kyrylo Lobach

Figure 1: NOX 'myLight' project — although each light fixture is technically different it is still similar to the others.

1 OBJECT AND OBJECTILE

All professions have been and are effected by the arrival of affordable computer hardware. Computer use brought the notion of digital work to light. In most fields, the digital realm is primarily used to store information virtually. Instead of storing information on paper, the digital realm allows for the storing of contents in a binary notation on storage devices on or through the computer. The binary notations can then be translated into visual information through the form of pixel information on digital screens. Alternatively, this binary language can be exported to an external device such as paper through the means of print. In doing so, this paper can also be classified as a storage device. Therefore, the definition of the digital in this sense does not represent anything new for architecture.

1.1 Design computing

Virtually all contemporary architectural offices utilize digital technologies and methods. Recognizing that most firms are operating in a computerization mode – through the emulation of already existing design processes and storage of information in binary notation, digital architecture can be seen as a generic term. However, only some practices operate in said computational mode, which is synonymous with algorithmic and computational design. In the computational method, the designer and computer are perceived as partners through the emulation and extension of the human intellect. It is perhaps fitting that the term computation is derived from the Latin 'computare' which means to 'think together'. (Leach, 2006) The emphasis in computational design shifts from absolute form-making to the addition of form-finding through the design of computing processes. Computation is also about rationalisation, reasoning, logic, algorithm, deduction, induction, extrapolation, exploration and estimation. (Terzidis, 2006) Computerization, on the other hand, is about digitization, automation and mechanization. Designers operating in a computerization mode have already conceptualized an architectural object and the computer is used to enter, manipulate and store that object digitally. Digital in this case, is a method of

storage since the digital language is a way of representing something numerically (in binary notation– which is not necessarily a number). Design computing, on the other hand, materializes architectural form as a result of the designed interactive system– which is stored digitally. As such computerization can be thought of as the utilization of traditional design processes, whereas computation is a novel design paradigm of process design; through the creation of interactive generative systems. Most importantly, the design computing paradigm provides a new addition to an existing architectural design ecosystem of approaches.

Fueled by the interweaving of computer science, biology and mathematics, a computational mode of operation has expanded the everyday vocabulary of an architectural designer to include words such as array list, emergence, formation, polymorphism, data structure, data trees, sliders, for loops, if statements, flow control, recursion, feedback, growth rate, performance, effectiveness and behaviour. Design focus has shifted from the creation of a particular object to an algorithmic definition of a generic object that can generate a variety of actual objects – or to use Bernard Cache’s more precisely defined term of an Objectile. An Objectile is essentially an open-ended algorithm; a generative, incomplete notation, which becomes a specific object only when each parameter is assigned a value. (Cache, 2011) Objects generated through an Objectile become object-states – bodies. This theoretical construct allows for the formalization of any and all possible varieties of a particular design objective which redefines the notion of the actual architectural object (whether it be the component of a building, a building, or a complex of buildings) and pushes it to the state of crisis.

1.2 Variation

Although algorithmic and computational design does not require computer use per se, it is widely accepted as a norm to use programming, scripting or graph languages in order to effectively design objectiles, even with commercial parametric software widely available. A heavy investment in computer programming language is essentially about building a techno-cultural construct that is

AN IDEA OF A TREE

built around predictable outcomes. Thus, the first question is: should the same objectile regenerate designs for different architectural object-states? How is regeneration applicable; is it on the scale of modular components (tiles, millwork, façade, layouts) or an entire building? Ultimately, each object that is derived from an objectile should be considered a variable with similarities of key pre-defined characteristics.

If not thoughtfully applied, the concept of regeneration has the potential to create outcomes that are slightly less unfortunate in comparison to cookie cutter housing. Overt similarities in objects are theoretically possible to overcome by creating a proto-objectile which produces a

Figure 2: Relations between a proto-objectile, objectile and object-state (body) through the example of a tree. series of different objectiles. One example of this would be a particular tree (an object-state) which belongs to a particular species of trees (an objectile) under the common notion of a tree (a proto-objectile). In this case, a tree becomes a term which encompasses a vast array of possible outcomes for a particular tree. Therefore, a proto-objectile is the most generic notion of something that can exist but can never be truly materialized. Once it is assigned one attribute of a purpose, objective or quality, it becomes an Objectile. An object-state arises as an objectile becomes more specific in its attributes.

However, a question still remains regarding whether variety is important and how does it fit with manufacturing technologies which are already in place. Given the creation of objectiles, it is easy to imagine an architectural office becoming a split-agency as defined by Mario Carpo. (Carpo, 2011) Perhaps in the near future, it will be possible to imagine a split-agency who will be selling its clients an objectile with a 'particular objective' embedded into a graphical application. In such a situation, the architects and designers of said split agency, will be the primary authors of a series of algorithms — objectile, while clients will be "secondary authors" generating object-states by directly manipulating the parameters of the embedded logic of an objectile.

1.3 Building the actual

Objectile is a series of algorithms. On the surface, it represents a shift in design methodology. However, will it spill into the consideration of the construction and operation algorithms of buildings to help create structures that understand that they are derived from the body of a particular objectile? Could this be a building that transforms over time based on the design of a particular objectile— whether it be automatically (mechanically and chemically) or with human agents? Mechanically kinetic structures can be problematic in contemporary architecture through poor built quality or significant maintenance requirements. One example of this would be the now cliché, non-moving aperture shading devices at the Arab World Institute in Paris by Atelier Jean Nouvel. This assemblage was an unfortunate failure since the responsibility of its maintenance fell not on an architect, but the building owner. It can therefore be concluded that, kinetic systems of higher energy and maintenance requirements with complex electrical mechanisms and mechanical structures have the potential to be left in disrepair. In recent years, there has been a progressing interest in chemically embedded motions. One example has been demonstrated by the research of Christina Doumptioti. Doumptioti contributed to popularizing chemically embedded motion in architecture by proving that it can maintain kinetic movement with less complex maintenance requirements compared to a traditional (mechanically) kinetic system. (Doumptioti, 2011)

Figure 3: CONTINUUM a customer designed dress from previously pre-designed objectile

Figure 4: SIM Residence by Sean Lally features a mechanically controlled ceiling

It can also be argued that a building is already an objectile with people and organizations creating additions, retrofits and demolitions to vary them. Promising developments which are connected to seamless translations of an objectile to an actual body are demonstrated by ongoing research regarding smart-dust particles, self-assembly and material computation. ("Smart Dust, Sailor research group at UCSD," 2003; Skylar, 2011; Menges, 2011) It is easy to imagine a future with biological construction modules which are pre-programmed with an objectile, creating various formations of different bodies (object-states).

The human body is an object-state generated by the body's objectiles —deoxynucleic acid. The body's objectile (DNA) is embedded within the cellular construction of the body. DNA is composed of various combinations of A-T,C-G pairs; with certain sequences of coding such as A-T,T-A,C-G,G-C,T-A which produce a particular chemical agent which is also known as a genome which initiates growth and change. (Barnes, 2008) DNA is composed of inherited codes from previous generations, with its predispositions towards and resistances to particular diseases. Furthermore, some genomes will not be fully activated if the external or internal conditions are not triggering them to be active.(Ibid) The body can, therefore, be said to have memory- on both material and cellular levels which are derived from a particular objectile.

"... we design the relationships and sequences that inhabit architecture and that emerge as its physical manifestation. But once we begin to invent such material processes, a new way of thinking about architecture reveals itself. It is a conceptual way of designing with architectural parameters, conditions, relationships, and degrees of freedom.

We ask ourselves: which parameters determine the design, and which do not, but still have an effect on its form and function? Using digital logics we define relationships and intentions in the form of rules. We weight the influences that the design-generating factors have on each other. Through the medium of programming we can model complex decision processes, checking and refining them iteratively. Architectural expression thus takes on a different character, because new conventions emerge in the medium of programming.

In this way of conceiving architecture, processes are not mere metaphors for a process-oriented approach to design, but are concrete sequences of operations, procedures that have to be designed. These procedures are determined, they have a beginning and an end. ... When architecture becomes the design of material processes, we no longer have a static plan in front of us, but a dynamic set of rules. We design a behavior." (Gramazio & Kohler, 2008)

Figure 5: Overlay of human portraits. An illustration of a human objectile?

INTERLUDE I — DEFINING AN OBJECTILE

The design research portion of this thesis is composed of a series of interlude sections. Some examples illustrate the concepts discussed which are further supported by a variety of precedents. Other examples explore the potentials and limitations of designing through various algorithmic processes.

Long, Thin, Yellow Legs by the Author

A design experiment titled 'Long Thin Yellow Legs' showcases the design methodology of an objectile. The project was a continuation of a student assignment from the author's graduate summer studio in 2012. The title of 'Long, Thin, Yellow Legs' itself projects the meaning of the experiment; in particular, how to represent each and every adjective of the required construct. The designer through his or her aesthetic sensibilities, knowledge and experience, was to individually determine how to represent these long thin yellow legs. However, such representations were deemed to be classified of an object-state and would not be representative of the full potential that an objectile can produce since an objectile is typically represented with a programming code. As such, numbers gain the ability to represent more than just lines.

The code which was produced for this project was a stochastic algorithm which meant that it would generate different results each time it was run. To create said stochastic behavior, the code asked the user for initial parameters such as the number of leg segments and their heights. These were assigned different values each time the code was run. Additionally, the code generated a random set of co-ordinates out of an acceptable range of values which were predefined in the algorithm by the author. This code represented an objectile of LTYL; an equivalent of human body DNA.

Figure 6: Example of LTYL population of possibilities. Illustrated in plan view.

Figure 7: Example of LTYL population of possibilities. Illustrated in perspective view.

Figure 8: First step in LTYL generation; program asked for initial parameters

Figure 9: Second step in LTYL generation; program asked for secondary parameters

Figure 10: Resulting geometry of LTYL processes. Source code can be found in Appendix I.

Garden Wall by the Author

The Garden Wall project represents a strategy for creating a vegetated louver system for existing built fabric. Its design approach was a systemic procedure. The process itself was divided into three parts, each with its own algorithmic representation in the form of a Grasshopper graph type of language. The three codes together composed an objectile of the Garden Wall. Similarly to LTYL, each part of the code was designed to limit an acceptable range of values that can work in the later stages of the design process.

Division of the code was required due to high computational power which was required to run the entire process. Through a single code, geometry was continuously regenerated making the entire objectile become unwieldy and slow. Dividing the code was to the benefit of the author since the modular nature of the resulting objectile's codes would be used in parts of any future objectile compositions. This approach was also demonstrated by the precedent of Induction Cities by Makoto Sei Watanabe- which will be further discussed later in this document.

Figure 12: Snippets of the three GH codes

Figure 11: Possible outcome of the Garden Wall process

Figure 13: Detail of Charles Babbage 'Difference Engine' replica

2 COMPUTERS AND TOOL MAKING

2.1 Before Computers

Computational design is informed by an array of ideas, from biology to complexity theory. However it would not have been an effective design method if not for the techno-cultural construct called the computer. It is important to point out that it is in fact, a techno-cultural construct since it is easy to define the computer as a technological invention due to its ubiquitous contemporary presence in a variety of personal electronic devices. To define it as only technological would be misleading since it is a cultural invention before a technological one. In western society, the first recorded use of the term dates back to 1646 by Sir Thomas Browne. At that time, it referred to someone who performed calculations that were needed to draw up a calendar. (Wurster, 2001) Therefore, a computer was the job title for a person with mathematical training who performed accurate calculations. This meaning was retained until the 1940's when the notion of a computer as a person was replaced by a computer as an electronic and mechanical device. Fundamental intellectual and technical developments made the computer into the device that we now know: calculation, logical formalization and mechanisation/automation (Wurster, 2001)

The history of calculation can be traced back to the fifth century Before the Common Era (BCE) through the writings of Herodotus. (Wurster, 2001) In his writing, he described how Egyptians used pebbles to add and subtract numbers. In fact, the Latin word for calculus means pebbles. (Ifrah , 2001) Pebble calculations can be thought of as a system for representing figures from other figures by applying particular rules- this is already an abstract machine for a contemporary human. This abstract machine seems unnecessary. At first however, it was the only effective way to calculate since the notational systems at the time were not easy to manipulate for calculations. Manageable calculation notational systems were introduced later in the fifth century Common Era (CE). This system today is called the Arabic notational system and was previously referred to as the Hindu-Arabic notational system. (Smith ,1911) Before its introduction, calculations

of large numbers were rather arduous tasks. As such, the result was a construct of abstract calculating systems. This abstract machine was later replaced by a physical device known as an abacus. Through archeological findings and early writings, the abacus appeared to be a common device throughout different cultures in the world. (Ifrah , 2001) The abacus was a mechanical device which still embedded the same concepts as the pebble manipulations of the Egyptians. It was typically a wooden box with a row of rods, and each rod had a series of wooden beads running through them. These wooden beads were, in essence, the same as pebbles. The difference being that the conceptual knowledge was now embedded in a mechanical tool. This embedding of conceptual knowledge is a reoccurring phenomenon in all technology.(Arthur, 2009) In 1673 Gottfried Wilhelm von Leibniz created a mechanical machine that was able to multiply, divide, add and subtract; in his own words he had created a “living abacus”. (Wurster, 2001) The abacus can be erroneously classified as an early computer; however it is a forerunner of a calculator. For a machine to be classified as a computer and not simply a calculating device, it needs to have an internal logic. An operation such as this can be categorised as logical formalization.

The foundation of logical formalization, which is in essence a formalized logical process, can be attributed to Aristotle. (Wurster, 2001) Aristotelian syllogistic logic is constructed by a major and minor premise followed by a conclusion. The classic example of such is the following: major premise is ‘All men are mortal’, minor premise is ‘All Greeks are men’, followed by a conclusion of ‘Therefore all Greeks are mortal’. A simple architectural example of this is ‘All libraries have bookshelves’, ‘Building A has bookshelves’, ‘Therefore building A is a library’. Obviously such a conclusion will not always be true. Aristotelian syllogism belong to deductive reasoning, which derives specific examples from general propositions.(Copi,2002) In direct contrast to deductive reasoning is inductive reasoning– which derives general examples from specific propositions. In the case of inductive logic, an architectural example will be based on observations and the probability of a building A being a library. Although Aristotelian syllogism has since been replaced by other logical methods and in contemporary education is only studied as historical and introductory examples

Figure 14: Jacques de Vaucanson duck automaton

Figure 15: Punch cards of Jacquard Loom

of logic, it does nevertheless, show that some logical problems can be solved by following if-then statements. What separates a computer from a calculator is the inclusion of the consideration for such statements in a physical device- whether they be mechanical or electronic.

Mechanisation is the third intellectual and technical development that made the computer, as a contemporary device, possible. In general, mechanisation means nothing more than replacing or magnifying muscle power- either human or animal, by a mechanical power of any kind. Classic examples of such include the wheel, the lever and the pulley. These early examples clearly delimited the operational mode of the devices. Later examples of mechanical devices include windmills, water-wheels and simple steam engines. These devices made continuous muscle power unnecessary as the machines derived their own power from natural sources. Such constructs were already automated mechanisms continuously running analogue algorithms impregnated with conceptual knowledge. Although there is a recorded history of mechanical automaton in the form of toys and musical devices, such as Jacques de Vaucanson duck automaton, a major leap in mechanisation did not happen until the 17th century. (Wurster, 2001) A manifestation of this leap was Joseph-Marie Jacquard's loom which was a device that was capable of weaving any pattern through 'programmed' punched cards. (Wurster, 2001) The last intellectual and technical development created fertile ground for the emergence of the computer as we know it today.

The combination of calculation, formalisation and mechanisation/automation as cultural concepts are underplayed in all computational devices. In contemporary times, computers are ubiquitous and employed in a variety of tasks as they are fundamentally responsible for the three aforementioned functions and concepts. The first manifestation of these concepts (in the form of a mechanical computer) was the 'Analytical Engine' designed by Charles Babbage in 1834. (Swade, 2001) Babbage's passion for designing these machines came from his obsession with mathematics. It was initially conceived as a number crunching machine; an advanced mechanical calculator with mechanical memory. Interestingly enough, it was Ada Lovelace who saw a huge potential of

his machines, well beyond number crunching. She has famously stated that just because these machines operated with numbers, does not mean that those numbers cannot represent anything else. (Graham-Cumming, 2012) In a similar manner; just because architects and designers choose to make those numbers represent a particular line in drawings it does not mean those numbers could not represent anything else, such as processes of generating an Objectile.

2.2 Modern Computers

Unfortunately most of Charles Babbage's machines were never fully built. They did, nevertheless, contribute to the inspiration of an engineer named Herman Hollerith to create an electric tabulating machine. This machine was fundamental to gathering, analyzing and storing a massive amount of information for the 1890 census that would not have been possible otherwise. (Picon, 2010). The difference between Herman's invention and today's computers is that his machine was task specific. Contemporary computers, however, are generic machines since they are capable of performing a variety of tasks as long as they can be described in mathematical terms; a universal machine. In fact, this universal machine was conceived as an abstract idea by Alan Turing in 1934. (Turing, 2012) In his paper "On Computable Numbers with an Application to the Entscheidungsproblem" Alan created an idea of an abstract machine to carry out series of algorithms to prove the main argument in his paper. (Turing, 2012) In his paper, he attempted to justify his mechanism — The Turing Machine, by appealing to the intuitive idea of computing the number by referring to an individual — the computer. (Chabert, 1999) This additionally highlights that the computer was a social idea before it was technological. As the computer became a machine, it started being referred to as technology. Technology, especially for a device like a computer, is always embedded with conceptual and cultural knowledge.

Although The Turing Machine was just a by-product of Alan's paper, it popularized an understanding of the differentiation between software and hardware. (Wurster, 2001) In fact, John von Neumann's paper 'First Draft of a Report on the EDVAC' outlined the schematic design for the first

American built computer through borrowing many of Turing's concepts and ideas.(Turing, 2012) It can be said that virtually nothing has changed from the 1946 EDVAC built by von Neumann and his team to contemporary computers. The devices are smaller, faster and are connected in a network for the most part; however they are still composed of arithmetic units, control units, storage (memory), and input/output units— von Neumann's architecture. (Wurster, 2001)These actions resulted in significant changes between the interaction of humans and computer devices. This had a profound effect not only within the architectural realm, but also virtually all professional fields.

2.3 User Interfaces and Software

The way we interact with contemporary computers is through hardware input devices such as a mouse, keyboard and touch screens. These input hardware devices control computer software interfaces: Graphical User Interface or GUI. Computer operating systems such as Windows, Mac OS, or Linux can be considered the main user interface, which allows for the running and manipulating of other software's GUIs on the computer; such as an AutoCAD program. GUI can be thought of as a graphical shell that encompasses all the algorithms included in a given program. These algorithms can be manipulated with the click of an icon rather than through the direct engagement with the programming code. This interactive development greatly expands the computers user base from highly-specialized individuals to the general public.

Although initial developments in graphical user interfaces were seen as early as 1951 (such as the Light Pen for the Whirlwind computer), it was not until the 1980's, when computers were finally accessible to non-specialists. (Wurster, 2001) As a result, the graphical user interface is the only form of the computer that some people of today's generation know. Graphical user interfaces brought direct manipulations of geometry to architecture in the form of intuitive three-dimensional modeling software. (Silver, 2006) GUIs made it easy to design through the discovery of manipulation rather than through mathematical formulation and descriptive geometry. In many ways, they were emulating the process of making a physical model through the act of form mak-

Figure 16: Light pen input device

Figure 17: Mac and PC Television Advertisement

ing. The explosion of graphical user interfaces during the 1980's and early 1990's also increased the significance of software development and made it just as (if not more) important than the hardware that was being used. This transition was clearly expressed in the popular Apple Inc. commercials titled 'I am a Mac and I am a PC'. The graphical user interface differences between the MacOS and Windows operating systems was the emphasis of this television advertisement. The hardware used by both companies, however, now use the same Intel processor chips. Contemporary competition in the computer industry—which includes cellular telephones and tablets, is increasingly focused around creating a greater ease of use through various advancements in their software offerings, application ecosystems, aesthetics and graphical user interfaces.

The increasing popularity of computer use in architecture is directly related to intuitive graphical user interfaces. The intuition of GUIs primarily rely on the emulation of existing analogue working methods— such as drafting with a T-square, physical and clay modeling. Sophistication and closer emulation in GUIs progressed further from the early 1980's due to the increased computational power of computer devices. (Ince, 2011) The capacity of GUIs to emulate existing processes has become increasingly accurate. An example of such is the creation of a digital model through various hand gestures via input devices such as Microsoft's Kinect and Leap Motion. ("Kinect for windows," n.d.; "Leap Motion," n.d.) However, these developments rely on a continuous computational power increase. Graphical interfaces are created primarily through computer graphic algorithms which are continuously running in the background which results in taking up a significant amount of computational power. The founder of the hardware company Intel — Gordon Moore made a prediction in 1965 which is now famously known as Moore's Law. Moore stated that the density of silicon circuits used in the computer hardware will double every two years and that its density will have a direct relationship to the computational power of the computer.(Ince, 2011) Over a period of roughly 70 years, the technology for computers has seen quite the transformation— from 5 million dollars for the colossal EDVAC to the contemporary 22 British Pound cellular phone-sized Raspberry Pi (and the slightly more expensive BeagleBoard).(Wurster, 2001; "Rasp-

commercial scale. There is also additional risk of it never even making it to the general consumer market since it is an experimental technology. s While GUIs bring sophisticated and intuitive ways of working in architecture, it may plateau at a certain point. Extreme computerization which creates fully immersive virtual reality “caves”– where the architect molds building by hand, might never happen.

The trend of bringing the fluidity and control of programming codes through GUIs is already in place– such as the example of a Grasshopper plug-in. It is generative modeling software that approaches the creation of algorithms through visual (graph) means and operates inside a three dimensional modeling software– Rhinoceros. The limitations of computing power might never allow for a complete border dissolvent between ease-of-use of GUIs and the fluidity and control of programming. Another issue regarding this topic which is not concerned with the computational power availability but described by Achim Menges and Sean Ahlquist in their seminal work: the shift of thinking processes required to design through computational means.(Menges,2011) The graphical user interfaces emulating existing processes might not be what early pioneers imagined when creating the first generation of computational devices. Nevertheless, it follows closely with Turing’s idea which in Alan Turing’s exact words, “it is a machine capable of simulating any other machine”– meaning anything which can be described in mathematical terms. (Wurster, 2001)

2.4 Tool Making

If a computer is a machine capable of simulating all other machines, is there not merit in developing a new set of architectural machines (tools) for existing or new paradigms? Or in other words, tools and processes that either do not exist in analogue forms or are not as productive without the automation of logical processes. The creation of tools by an architect for the purposes of creating architectural design and constructs is not a novel idea. In fact, an example of custom made architectural tools can be dated back to the 1440’s in the work of Leon Battista Alberti; a project titled ‘Descriptio Urbis Romae’. (Carpo, 2011)

Figure 20: Physical Tool. Left to right order
– Phantom Geometry, superKUKA, 5-Axis
Motion and VPRP

This Albertian machine (tool) consisted of a polar co-ordinate drawing system and a list of co-ordinates to be drawn that was compiled by Alberti. Through the transferring of the co-ordinates to the appropriate locations on the device, the user would get a map of Rome as surveyed by him. Alberti created this device as a response to an inaccurate transfer of images during the printing process— for the purposes of creating accurate representations of images that he intended to be in the book. At the time, printing presses of words was accurate through the use of block printing devices. However, the majority of images were drawn by artists and were at times altered by them without notifications to the original author. (Carpo, 2011) Due to these inaccuracies, Alberti advocated the use of words to describe a particular architectural feature. In the ‘*Descriptio Urbis Romae*’ project, where the transferring of images was fundamental to the project, Alberti was set to overcome the printing press accuracy limitation by inventing his own device as pictured in the above Figure 19.

The Albertian analogue machine (tool) was a physical manifestation of his frustration with the limits of the printing technology of his time. The lack of appropriate tools for Alberti’s intent, creation of identical copies; in this case, geographical maps, made him design his own tools. Similarly within our contemporary world, architects and designers have the capacity to create their own tool if their intents are not met with readily available devices. The major difference between now and the 1440’s is the migration of tool making from atoms to bits (physical to digital); architects and designers can be digital toolmakers. Although the majority of architects and designers do not engage with the creation of their own tools, there are a handful of disciplinary borders defining architects and designers that do. The majority of tools are digitally based creations.

Even the majority of contemporary analogue tools have a deep connection with the digital realm. Such analogue tools can be exemplified by the creation of a custom robotic arm, such as the KUKA— a multi-purpose robotic armature with attachments allowing the robot to carry out tasks not initially envisioned by its manufactures. Such tool making was exemplified by the work of ‘Ro-

bot Fabrication Unit' by Gramazio and Kohler, 'superKUKA Tools' by Wes McGee and Dave Pilgram at Michigan University, '2012 Annual Pavilion' by Achim Menges and his students at Stuttgart University, 'Five-Axis Robotic Motion Controller' by Lift Architects, 'Phantom Geometry' by Liz and Kyle von Hasseln, and 'Variable Property Rapid Prototyping' by Neri Oxman.

All of these physical tools were created in order to translate a particular project, paradigm or method from bits to atoms. In the case of Gramazio and Kohler's work (in projects such as '17 degrees of Deviation'), they used the KUKA robot — KR150 L110 arm, to recreate a computational design with the help of custom written software.(Gramazio, 2008) This digital construct was created for the purposes of contributing to their exploration of novel methods of construction techniques and changed the culture of design by reevaluating the modular construction of bricks. (Reuters, 2011) However, it can be argued that the attachment for holding the brick by five-axis robot could have been reincorporated from the robot's already envisioned functional attachments such as sorting at the factory. Other G&K projects such as 'FOAM' incorporated means that are not envisioned by the robot arm manufacturers by creating attachments for liquid additive manufacturing. In other research projects, attachments have been made to mold clay and create procedural landscapes from sand as per Figure 21.

5-6 Axis CNC Robotic arms are computer numerically controlled and can easily be categorised as the same as 2.5-3 Axis CNC Routing machines. However, there is a significant difference between the two. A robotic arm is a multi-functional device whereas routing machines are task specific. Consequently, programming requirements for robotic arms are much higher than a routing machine since routing machines can utilize off-the-shelf software such as Master-CAM. (Gramazio, 2008) These programming requirements led Fabrication Robotics Network's Wes McGee and David Pilgram to create 'SuperKuka Tools' for the 7-Axis KUKA Industrial Robot, which is a custom written software package for executing a variety of typical fabrication processes such as: 2-dimensional knife-cutting, large-scale additive fabrication (foam deposition), robot-mounted hot-wire

Figure 21: Research projects by Gramazio & Kohler.
From left to right FOAM, Clay Molding and Procedural
Landscapes 2

Figure 22: Variable Property Rapid Prototyping (VPRP)
by Neri Oxman

1962 – SKETCHPAD BY IVAN SUTHERLAND
1969 – JOHN FRAZER'S AA THESIS
1970 – ARCHITECTURAL MACHINE BY NICOLAS NEGROPONTE
1989 – TOPDOWN PROGRAMMING LANGUAGE BY ROBIN LIGGETT AND WILLIAM MITCHEL AT UCLA
1990 – INDUCTION CITIES BY MAKATO SEI WATANABE
2002 – MTABLE BY GRAMAZIO & KOHLER
2002 – GEHRY TECHNOLOGIES BY FRANK GEHRY ARCHITECTURAL PRACTICE
2005-2012 – MAJORITY OF BIOthing PROJECTS
2007-2012 – TOOLS BY MOS OFFICE
2001-2012 – VARIOUS TOOLS FOR PROJECTS BY AA EMTECH AND ADRL

Figure 23: Timeline of notable digital tool makers

cutting, and robot-tended rod-bending. (McGee,2011)

Neri Oxman's Variable Property Rapid Prototyping (VPRP) development was one of the most significant tool making processes that was driven by her intent to shift the construction of an architectural object from material assemblies to material distribution. Oxman created a three-dimensional printing technology that could easily print variable property objects in one continuous three-dimensional take. (Oxman, 2011) The design of the objects that Oxman printed with said VPRP patented technology were created with the help of computational means. Therefore, she was not only a physical tool maker but also digital.

Figure 24: Ivan Sutherland using 'Sketchpad'

2.5 The Three Categories

The progression of digital tool making can be divided into three categories; theoretical approaches, general project tools such as Digital Projects by Gehry Technology, and lastly of specific project tools such as R.V. project by MOS Office.

Essentially, all of these tools are examples of computer software which are composed of a series of algorithms that both generate geometry and create GUIs. The third category is software that could be used to generate a particular project, making this software a generative tool. Therefore, the algorithms and workflow designed represent a generic object or an objectile. Once variable

parameters within this objectile (software/tool) are initialized with a particular value, an object-state is created.

The first category of projects date back to late 1960's. These include the Sketchpad by Ivan Sutherland in 1962, John Frazer's Architectural Association Thesis project of Generative Space Frames in 1969, Architectural Machines by Nicolas Negroponte in 1970 and Top-Down project by Liggett and Mitchell in 1989. The common theme in all of these projects was that they did not necessarily create a particular architectural object and were not necessarily explicitly connected to an architectural design. However, all of them play a part in furthering architectural thinking of tool creation for the purposes of generating architectural objects or object-states. Sketchpad by Ivan Sutherland was the first GUI that was related to production of design. (Picon, 2011) It was intended to create an effective means to communicate between man and machine. Through the manipulation of drawing objects with the help of a light pen, the designer could draw anything on a monochromatic CRT screen. Furthermore, the object could be adjusted to have straight corners with the push of a physical button- thus showing logic embedded in the program in addition to the computational logic needed to display the drawing. The importance of the project was demonstrating that design can in fact, be created on a computer, through the combination of emulating existing analogue processes be it through drawing and creating new computer driven processes including said automatic adjustments of geometrical shapes.

John Frazer's thesis in 1969 was an architectural exploration of generative design through the engagement with computational devices. (Frazer, 1995) Through the design of algorithms, Frazer was able to create a variety of different space frame structures all generating from an initial seed.(Burry,2011) This essentially was the creation of the first objectile.

The second category was the creation of a digital tool with the most generic or multipurpose approach to an architectural project. A clear example of such was the Digital Project software which

Figure 25: Left: Induction Cities design processes. Right: On-Demand City digital tool (1995) by Makoto Sei Watanabe

was developed and offered by Gehry Technologies- an AEC technology development and consulting company founded by Frank Gehry's architectural practice. The difference between Digital Project and AutoCAD software was that the former was conceived as a Building Information and Associative Modeling software to deal with the complexity of the Frank Gehry architectural office designs. The latter was the reflection of analogue techniques of drawing which were made by an established company — Autodesk, wanting to gain the widest market possible. The second category also included general tools which could then be combined to create a specific project such as the Induction Cities project by Makoto Sei Watanabe. This project, which started in 1990, was composed of a series of digital tools which were capable of working independently of each other. Together these tools were envisioned to help design a city by engaging with known working city practices. (Watanabe, 2002) The six digital tools were: Sun-God City (1994), Generated City Block (1995), City of Hills (1995), Sun-God City-2 (1995), Wind-God City (1995), On-Demand City (1995). Each had a specific objective. For example, Wind-God City generated urban fabric

with pleasant breezes while avoiding winter winds.(Ibid) Each program could be thought of as an objectile generating a specific object-state with limited objectives. The Induction Cities, as per Watanabe's book, refers to the combination of these digital tools with each one producing the best possible result, through an evaluation matrix, to be passed on to the next tool. The result was a tabula rasa city demonstrating all of the objectives of each one of the objectiles.

Each one of the Watanabe's digital tools however can be applied individually to either re-evaluate existing conditions or work with data generated with other means than his processes. An example of the latter is the application of On-Demand City, which generates functional zones of the city based on proximities, – to pre-defined road and lot divisions.

The third category is digital tools created for the purposes of generating a specific project. Prominent works of such an approach are exemplified by Gramazio & Kohler's mTable (2002), a collection of projects by the MOS office of Michael Meredith and Hilary Sample, and the majority of projects designed by Biothing of Alisa Andrasek and Jose Sanchez as a frequent collaborator.

mTable by Gramazio & Kohler differs from the other examples since it was also exploring the designer to client relationship. In essence, they created an objectile of a table design. They also incorporated a graphical user interface into their digital tool and distributed it over the mobile cellular device. As a result their client/customer could directly engage with the parameters that drove the table design.(Gramazio, 2008) mTable was of course an objectile as per Gramazio & Koehler quote from 'Digital Materiality'.

"... Customers chose the table's dimensions, material and color on their mobile phone display. They can then place deformation points on the underside of the table and apply pressure on them. This turns the underside into a landscape. ... The program on the mobile phone regularly verifies whether the table is still structurally feasible despite the holes in it. ... After placing an order, the table is cut according to the mobile phone data by a computer-controlled milling machine. ... (E)very table unique — admittedly only superficially, as they all share a common origin of form and concept ... yet together they form a single entity: the mTable design family." (Gramazio, 2008)

Figure 26: mTable object-states

Projects by MOS Office, such as Crumple, Drift, Grid, MITList Without Out, Net, Onthevergeofcollapse, Temporary Cinema and FORTRESS, and the majority of projects by Biothing differ from mTable as they were created for the internal use of generating designs.

From the list of MOS digital tools, some were used for generating installations and pavilions and others, not unlike in the approach of Makoto Sei Watanabe, were used in combinations. In the case of Biothing, what makes their work unique was the explicit computational tool creation for every project. Each digital tool represented the projects' complete generative capabilities. These digital tools were explicitly said to be created by the programming language Processing. (Andrasek, n.d.)

Therefore, it can be concluded that digital tool making represents the design of algorithms for the purposes of creating generative design under defined constraints and objectives — in other words an objectile and its body.

Figure 27: Top row left to right: Crumple, Drift and Grid. Bottom row left to right: MITList Without Out, Net and Onthevergeofcollapse

Figure 28: Turing Pavilion printed model and digital tool by Biothing

Figure 29: Prizma by Biothing, an objectile of housing complex

This project, created by the author, can be thought of from two different perspectives. Firstly as a generative device, an objectile, for particular object-states (bodies) of a balcony addition to an existing facade. Another way to look at it is as a creation of a digital tool with embedded objectiles generating different object-states depending on user input from a range of predefined values on the provided sliders of the GUI. Both perspectives of the project are correct and it essentially depends on who is operating the program once it is running, the author or a third party — the user. In the case of the author, the sliders provide a way for rapid regeneration of the object-state since each value on the slider feeds back as an external parameter to the objectile making it regenerate a new object-state. Alternatively the author could directly manipulate the code in order to regenerate object-states. In the case of a third party — the user, the slider GUI provides a way to manipulate the objectile without knowing the inner workings of the algorithms and the programming code that it is composed from. This objectile was created through Java and Processing programming languages. There are two other approaches: scripting and graph. Each will be discussed in detail in the next chapter 4 — Programming and algorithms.

Figure 30: PixelFacade by author. Showcasing different objectile parameters and GUI

>

3.1 The Extent of Things

In addition to the examples from the previous chapter, it is important to examine the ecosystem of design computing projects in varying scales and levels of implementation. Before that, the circumstances leading up to the current computational scene should be examined closely.

As previously discussed, digital design can be divided into two categories. Each category however, does not have embedded discriminations as buildings produced by digital means can be a box or a blob and anything in between. Banal and repetitive structures can be produced and built with the help of building information modeling software such as Autodesk Revit and there is of course, a tendency to categorize these buildings as something else but digitally produced. (Carpo, 2011) At the same time, digital and computational design's association with curves and complexity is not surprising as curves are marginally more difficult to digitally model than a straight line, be it in the case of direct digital manipulation, or mathematical definitions such as through an algorithmic definition. Yet the origin of curves in the architectural scene of the early 90's was not founded by computer use but rather, through formal tactics of continuity, as seen in the example of a smoothing of folds that was created in order to differentiate the massive homogeneous volumes in the work of Henry Cobb's John Hancock Tower in Boston amongst many. (Lynn, 1993) The fold following this example in Greg Lynn's article from 'Folding in Architecture' issue of Architectural Design hinted on the approach towards dealing with a particular design problem.

The curving of the fold was simply augmented through computational means and at the same time, positively fed curve explorations in the digital realm. This feedback was so prominent that the curve became associated with the digital realm instead of a formal strategy to deal with heterogeneous and conflicting forces of the site and program conditions. Complexity and curvilinearity can be easily explored by the designer with computationally defined rules. As such,

Figure 31: John Hancock Tower by Henry Cobb

architects and designers exploring the design computing approach often resort to the exploration of complex, descriptive geometry as a status-quo, since such constructs are difficult– if not impossible, to create through analogue or analogue emulation means. Design computing also has manageable control of the complexity created through generative geometry than anything that is digitally designed, through direct manipulation of the geometry. This complexity and the status-quo associated with it is what seems to define the contemporary design computing practices. However, there are two ways of approaching design computing– as design tools and as design-in-itself. The former represents the work by Makoto Sei Watanabe such as Induction City which was composed of a collection of design tools. The latter was exemplified by work of MOS Office and Biothing among many.

These two approaches cater to a variety of theoretical concerns. Over the last century, they have been broadly shifting in an unprecedented fashion in architecture. (Leach, 2006) This represents both the variety and the lack of a single theoretical discourse during the last decade as pointed out by the editor of 'Constructing a New Agenda' Krista Sykes. These rapid shifts or the lack of concentration nevertheless have created a rich culture of different methodologies that are available to architects and designers to use without triggering a ban on the previously existing methods. This variety is suggested to be stemming out of the critique of a utopian modernist quest to rescue society for its ills– which architects recognise as an unattainable task which is positioned for a failure. (Sykes, 2010) As a result, the architects' approaches became more pluralist in nature. (Nesbitt, 1996) In essence, the last century saw an increase in a variety of approaches, theoretical concerns and methodologies in the architectural ecosystem.

It can be thought that these varieties of approaches were born out from the criticism of the previously used methods and theoretical concerns– such as the move to post-modernism from the critique of modernism. Post-Modernism argued against Modern standardisation and instead advocated for differentiation, variation and choice (Carpo, 2012) – as was the case in Robert

Venturi's seminal essay "Complexity and Contradictions in Architecture".(Picon, 2010) Post-modernism was followed by the deconstructionist phase which could be considered to be inspired by the surface reading of Jacques Derrida.(Ibid) Deconstruction in architecture meant to formally express the conflicting logics and processes of the surrounding context and culture, which often led to literally fragmented forms of an architectural object.(Ibid)

The early work of Peter Eisenman is an example of such transformations in architecture, as shown in Figure 32. Eisenman carried out a series of geometrical transformations during the design stage, with the final result being the literal representation of the built work. The contradictions and opposing forces of the found conditions- whether they were concerning site, culture or economic, were simply augmented and fractured.(Carpo, 2012) Therefore, deconstructivist theoretical concerns were formalized as fragmentation strategies.

Figure 32: Peter Eisenman House Transformations

Architectural forms that are typically referred to as digital architecture are indebted to the reaction in the early 1990's concerning the fragmented formalisation of deconstruction architecture. (Picon, 2010) Folding, as defined in a seminal essay by Greg Lynn titled "Architecture Curvilinear-

earity”, is the continuation of deconstructivism’s concern of heterogeneous conditions through formal strategies of continuity instead of fragmentation. It is important to note that the reaction was regarding the formal strategies of fragmentation and not of the initial concerns. This continuity was inspired by the term Fold, triggered in architecture by the translation of “Le Pli – Leibniz et le baroque” by the French philosopher Gilles Deleuze. In the Fold, Deleuze insisted on the its possibility to deal with complexity through different means rather than through discontinuity and frontal collision.(Ibid) Therefore, it is easy to see how counter arguments of deconstructivism’s formal agenda is formerly through surface reading of the Fold– ultimately, applying a theoretical concept to literal form. It was Peter Eisenman, one of the most prominent figures in the deconstructivism movement, who discovered and adapted the Deleuzian ‘Le Pli’ to post-deconstructivist architectural theory. (Carpo, 2011) In both “Unfolding Events” and “Folding in Time”, Eisenman introduces terms of an objectile and in the latter work, the term of object–event. (Eisenman, 1992; Fischer, Aicher, Eisenman, Speer, & Olin, 1992) In “Folding in Time”, Eisenman

Figure 33: NOX Water Pavilion

argues that for Deleuze the notion of the object is change and it is concerned with the temporal modulation that implies a continual variation of matter.(Fischer, Aicher, Eisenman, Speer, & Olin, 1992) Folding for Eisenman has always been about the process rather than a literal translation of folding in architecture. (Carpo, 2011) “Le Pli” offered Peter Eisenman a new perspective on complexity and an alternative to a fractured deconstructivism formalist movement. These perspectives were theorized in a collection of essays edited by Greg Lynn, Eisenman’s former student, for the Architectural Design issue in 1993 titled “Folding in Architecture”. The special issue of Architectural Design included texts by prominent figures such as Gilles Deleuze, Jeffrey Kipnis, and John Rajchman and shared projects by prominent architects: Peter Eisenman, Frank Gehry and Philip Johnson. This list lent weight to the publication and the concept of the fold and generated an intense interest in the phenomenon for the remainder of the decade.(Sykes, 2010) Folding, although envisioned by Lynn as an approach to deal more effectively with increasing complexity of conflicted urban and cultural contexts, mostly charted a new formal approach which was marked by continuity and smooth transitions.(Picon, 2010) Culminating in the visual/formal style coined by Greg Lynn as a “blob” — Binary Large Object. (Lynn, 1998)

The blobs, having curvilinear forms, were easier to design through computer software means – more precisely, with computerization. A positive feedback loop was then established with computer software methodologies informing form making techniques of blob architecture and computerized architectural practice in general– through the association of digital architecture with digital form making or as defined previously classified as computerization. It would be naive to say that computation does not lead to formalization since all architectural projects materialize themselves into a form. This form is viewed and defined as a dynamic relationship which is informed by a variety of forces and multi-disciplinary concerns. Computation which is critiqued as only formal explorations is best addressed by Sanford Kwinter in his seminal essay titled “Who’s afraid of formalism?”.

... Form[...] is ordering action, a logic deployed, while the object is merely the latter's distant theme. ... The great formalists [...] have always been able to peer into the object toward its rules of formation and to see these two strata together as a mobile, open and oscillating system subject to a greater or lesser number of external pressures. The manifest form — that which appears — is the result of a computational interaction between internal rules and external (morphogenetic) pressures that, themselves, originate in other adjacent forms. The (pre-concrete) internal rules comprise, in their activity, an embedded form, what is today clearly understood and described by the term algorithm. (Kubo & Ferré, eds., 2003)

DESIGN COMPUTING			
FORMALIZATION	MATERIAL COMPUTING	MASS CUSTOMIZATION	BOTTOM UP/EMERGENT
evolution	evolution	evolution	evolution
formations/morphogenesis	formations/morphogenesis	formations/morphogenesis	formations/morphogenesis
theory of transformation	theory of transformation	theory of transformation	theory of transformation
general system theory	general system theory	general system theory	general system theory
metabolism	metabolism	metabolism	metabolism
material gestalt	material gestalt	material gestalt	material gestalt
objectile	objectile	objectile	objectile

Figure 34: Design Computing Multi-Disciplinary Areas of Concern

3.2 Contemporary Scene of Design Computing

Following Kwinter’s quote, it is easy to see how design computing refers to generative form through algorithmic means. This method of approach does not necessarily need computer use per se– as demonstrated by the material experiments of Gaudi and Frei Otto. However, it would be difficult and limiting without it, if not impossible, to effectively generate form through analogue computing. Design computing, as exploration and the utilization of algorithmic processes is connected to deconstructivism processes as exemplified by the work of Peter Eisenman. These concerns however, were based on the notion of conflicting forces and heterogeneous conditions, similar to a Post-Modernism approach. Some authors, as in the case of Mario Carpo, would argue for Design Computing vindicating, decades later, the arguments of Post-Modernism’s call for differentiation, variation and choice. (Carpo, 2012) Besides Design Computing’s apparent heredity in process concerns of deconstructivism, it also brings together a deeper multi-disciplinary

discourse. The multi-disciplinary discourse can be further divided into areas of interest with prominence given to different subjects.

At least four areas can be identified, as per Figure 34, each with a variety of built scales. The overall theories informing Design Computing are (1) Evolution, (2) Formations, Morphogenesis and Theory of Transformation, (3) General Systems Theory, Metabolism, (4) Material Gestalt and (5) Objectile. These theories are derived from and represent a combination of previous ideas.

(1) Evolution as an idea dates back to the time of ancient Greeks. However, it was only during nineteenth century with the publication of *On the Origin of Species* (1859) by Charles Darwin, that the idea of organisms evolving gained ground. (Ruse & Travis, 2009) Evolution as an idea stems from the realization that all organisms, living and dead, including humans, are the product of a long natural process of change through which each species is descended from other, different ones. (Ibid) Furthermore, Darwin gave this idea a driving device of natural selection. The way it applies in architecture is through thinking of a particular design or typology as being in evolution. On one hand, design evolution can be thought of as a fancy word for design iterations- as in the case of architectural approach of Bjarke Ingels Group (BIG). (Ingels, 2010) On another hand, evolution as it applies to architectural design can be approached through evolutionary computing, which uses genetic algorithms to create iterations of the design. (Man, Tang, & Kwong, 1999) Each iteration is thought to be evolving closer to a given objective of the design. Essentially this means Design Computing is using objectiles to generate forms with each body being tested and evaluated through a matrix of fitness, as defined by the designer. (Watanabe, 2002; Shiffman, 2012; Ibid) After the completion of the process, an objectile will generate the best fit for the given task. The second way of thinking about evolution in architecture is through a continuous reworking of a given architectural typology.

(2) Evolution occurs because there are variances between different individuals within a species. While natural selection is an idea of what happens due to these differences, Morphology, as described by Johann Wolfgang von Goethe in 'Ideen uber organische Bildung' in 1806, is a study of form and why these differences happen. (Menges & Ahlquist, 2011) Furthermore, Morphology, as described by Goethe, links geometric behaviour with functional logic. Goethe links this to individual form being a continuous formation that transforms over time. (Ibid) Formations are therefore, elements of a larger system that undergoes metamorphosis. Morphology effectively describes why variations in a species occur, which allows for natural selection. Seminal work by D'Arcy Thompson, On Growth and Form (1917), is connected to Morphogenesis through describing a device — Theory Transformation that explains how variances in species happen. (Thompson & Whyte, 1942) Thompson sought to define form as a mathematical system through understanding how physical forces produce structure and pattern. (Menges & Ahlquist, 2011) These mathematical systems created forms through organising itself in the presence of both internal and external forces. (Thompson & Whyte, 1942) Consequently, different internal and external forces produce different forms. In architecture, Morphogenesis and Theory Transformation are an inspiration behind parametric and associative logics of form. (Menges & Ahlquist, 2011) Parametric design can be understood as establishing methods of interconnected behaviour of forms and forces through mathematical and geometric rules. (Woodbury, 2010)

(3) General Systems Theory is a critique and an alternative to classical physics' deductive methods. (Menges & Ahlquist, 2011) It was formulated by Ludwig von Bertalanffy in 1950 and 1969 and proposes that nothing in nature exists in isolation, but rather, needs to be understood as complex systems of interactions and reciprocities. (Ibid) Furthermore, each system is said to have equifinality and feedback. (Ibid) The former refers to the notion of a systems' state being achieved from various initial starting conditions. The latter refers to a mechanism by which information is reinvested into the system to provide a constant rebal-

ancing and recalibration of its functioning state. General Systems Theory was incorporated into architecture by Christopher Alexander in his Architectural Design article titled 'Systems Generating Systems' in 1969. Alexander focused on architectural design through systems whose behaviour was derived through interaction among its parts. (Ibid) This further confirms that architecture is composed of formations that undergo morphogenesis. Furthermore, systems thinking outlines how such interactive and behaviour based systems can be achieved, which directly connects to parametric and associative logics of form. In addition to Christopher Alexander's research, John Frazer has also applied biological principles of morphogenesis and evolution in architecture, as described by his book titled 'An Evolutionary Architecture'. Frazer's approach was similar to Alexander's in creating systems that closely resembled the robustness, variability and complexity of natural systems. (Frazer, 1995) Furthermore, Frazer proposed that architecture is a participant of a natural system while also exhibiting metabolism; exchanging with the environment and responding to feedback.(Ibid)

(4) Material Computing as exemplified by the work of Achim Menges (amongst others) is a relatively recent addition to design computing. (Menges & Ahlquist, 2011; Menges, 2012) Material computing was a significant part of an underlying theoretical foundation of design computing prior the work and writing of Achim Menges which put it in the light of a digital medium. Even Antoni Gaudi and Frei Otto have both experimented with analogue material computing systems for the purposes of form finding. (Ibid) The difference however, was the continuous effort to incorporate material systems in the production of the design systems (objectiles). Where previous theories have outlined how systems can be built for the purposes of design, material computing suggests that the systems' concerns should be based on material behaviour, methods of manipulation and assembly.(Ibid)

(5) Objectile, as discussed previously, refers to a generative, incomplete notation, which becomes a specific object only when each parameter is assigned a value. (Cache, 2011) The

incomplete notation is a series of algorithms which represent a generative system. Objectile is an all-encompassing term, which refers to a system and implies its inner workings, without describing how that system should be constructed.

Design computing is a construction of interactive systems, originating in emulations of natural systems and phenomenon. These interactive systems calculate their behaviour and make simple logical decisions based on previous calculations and automatically adjust according to the system's design. Essentially using three cultural concepts which are calculation, logical formalization and automation, all embedded in computational devices, these concepts concentrate on the 'how-question' with regards to creating these interacting systems with inspirations that are derived from other disciplines observing natural phenomena and constructs. The 'why-question' is not sufficiently articulated aside from the ambition of bringing the design methodologies closer to one of a natural system. Some authors, such as Rudolf Finsterwalder in 'Form Follows Nature', argue that humankind has always been striving to emulate natural processes in their own constructs. (Finsterwalder, 2011) Yet other researchers and practitioners, such as Achim Menges and Neri Oxman, address the 'why-question' with an increasing emphasis on material computation. (Menges, 2012; Oxman, n.d.) Therefore, the simplified argument states: design computing should be utilized because it can account for material computing in its systems. Nevertheless, the material computing reasons for utilization refers to the natural processes, with examples similar to material depositions and reconfiguration of calcium in the human bones. (Ibid) Design computing methodology operates under the pretext of superior intelligence of natural systems and while this intelligence might be correct, it is not exactly the point. The dominance of 'how-answers' in relation to 'why-answers' have created an environment of architectural projects with easily recognizable design computing heredities which materialize in the same formal strategies.

3.3 Next?

These common strategies as discussed and illustrated above are represented in the form of vari-

Figure 35: Puppet Theater by MOS Office

Figure 36: Voussoir Cloud by Iwamoto Scott

Figure 37: TMV by Theverymany

able element repetition; whether it be through the design of a small scale building or detailed component. Each of the elements behaves as part of a system. In many built examples, design computing is predominantly associated with various exterior or interior building components. The exterior components are typically exemplified by the façade treatments; the façade is subdivided in various manners with computationally designed components nesting within each subdivision. Each component has pre-programmed (or embedded) behaviour and as a result, create variety in each of its components, depending on its internal and external conditions. One built example of this methodology is the Shenzhen International Airport by Massimiliano Fuksas and O-14 Office Building by Rieser & Umemoto. Interior design is typically exemplified through the repetition of variable components- not the same but similar in the treatment strategies of wall and ceiling construction, as seen in Hilton's Pattaya Hotel which was designed by Department of Architecture office.

In the current computational scene, the design of components is predominant; however some practitioners are creating small structures through computational means. Projects of these small structures are also including material computation as one of their exploratory aspects. As mentioned previously, this trend is best exemplified by practitioners such as Achim Menges and Neri Oxman. Upon further investigation of their projects it can be argued that they are extremely concerned with the design of generic components with embedded behaviour that will take various forms depending on the internal/external conditions. As such, it makes these projects similar in approach to the former examples.

There are also other examples of design computing on the scale of a small building, such as the Embryological Houses by Greg Lynn. The emphasis of this project was to develop an object of the house in order to generate a family of houses as a marketable product. In Greg Lynn's words from the Architectural Design issue dating to the year 2000: "The technique engages the need for any globally marketed product to have brand identity and variation within the same graphic and

Figure 38: Shenzhen Airport by Massimiliano Fuksas

Figure 39: O-14 by Rieser Umemoto

Figure 40: Hilton Pattaya by Department of Architecture

Figure 41: ICD + ITKE 2010 Research Pavilion lead by Achim Menges

special system, allowing both the possibility for recognition and novelty.”(Carpo, 2012) The generative system of the house was composed of a series of connected sub-systems which therefore allowed for associative changes throughout. It might appear that Embryological Houses did not follow the same formal strategies of previous examples, however, the sub-systems of variable louvers clearly confirms the opposite.

Figure 42: Embryological Houses by Greg Lynn

This predominance of a variable element repetition is not surprising as computer algorithms are always created as repeatable routines. This repeatable routine is best demonstrated through a simple example of generating two points and drawing a line between those two points. Figure 43 showcases three different approaches to creating such an algorithm. The first approach is programming, second is scripting and third is through graphs. Each of these methods has its benefits and drawbacks and will be discussed in detail during the next chapter. As evident from Figure 43, it makes little sense to create such algorithms for such minor results unless this routine will be used more than once. This routine also becomes useful and appropriate if thousands of such points need to be generated with lines drawn between them.

Additional generative behaviour can be programmed in the algorithms such as line avoidance of selected points. The lines that fall into the avoidance area will, with their new functionality,

evade the area through regenerating themselves into curve, while still connecting to the same points. This simple example shows how repeatability is explicitly connected to the creation of algorithms- where the continuous re-use of the algorithm is useful for huge but simple and highly repeatable tasks.

This aspect of computer algorithms technically explains the 'how' centric approach in design computing. However, the 'why' might not be as closely connected to the multi-disciplinary theories and natural processes- which ultimately leads back to another 'how' in relation to design computing's heredity in the Post-Modernist manifesto. Design computing is composed of algorithms that together create systems which are capable of producing differentiation, variation and choice.

Figure 43: From top left to right: Java programming, RhinoSCRIPT scripting, Grasshopper graph

INTERLUDE II — CITY BLOCKS

Although every design computing system can generate a variety of object-states each time it is run, the initial list of parameters and constraints will stay the same. Arguably, some parameters and/or constraints can be assigned values of zero which in effect, make them obsolete. These initial conditions of the series of algorithms composing an objectile suggest that every external environment, for example a building site, must have same list of conditions. These conditions do not have to be the same but must still be present in some form. Following the building site example, this suggests that every site, although different, must have some sort of commonality between each other. The figures below explore whether commonalities can be found in different building sites. The urban fabric of eight different North American and European cities are compared figure 44 – 52.

North American cities such as Toronto, Montreal, San Francisco, Houston and New York City, although different, all follow regular rectangular grids. They can be classified to be topologically similar. One North American city in the study, Washington D.C., and all European cities follow a different logic of irregular polygonal divisions. Therefore, while most North American city fabrics can be defined as a rectangle with four vertices, European city fabrics are defined as polygons with n-number of vertices. Importantly polygons with four vertices create a rectangle– thus, extending the commonalities between North American and European city fabrics. Extending this logic allows a theoretical building massing objectile to be defined, with one of its initial parameters being the city block size and shape which can work in both North American and European conditions.

Figure 44: Toronto Urban Fabric

Figure 45: Montreal Urban Fabric

Figure 46: San Francisco Urban Fabric

Figure 47: Houston Urban Fabric

Figure 48: Washington D.C. Urban Fabric

Figure 49: New York City Urban Fabric

Figure 50: London Urban Fabric

Figure 51: Paris Urban Fabric

Figure 52: Barcelona Urban Fabric

4 PROGRAMMING AND ALGORITHMS

... Tools embody conceptual knowledge. Harnessing tools may relieve the designer of some physical and mental effort, but may also allow or suggest the acquisition of new conceptual knowledge. Therefore, never be limited by the available tools. Think beyond the tool. Tools should challenge the designer. The designer should challenge the tools. Become your own tool builder. Challenge yourself. (Aish, 2011)

4.1 Natural and Artificial Languages

Generative objectile tools are composed of a series of algorithms written in the form of programming languages. These include: object-oriented programming languages (OOP) such as Java and Processing; scripting languages such as RhinoSCRIPT, Maya Embedded Language (MEL) and Python and graph languages such as Grasshopper for Rhinoceros three-dimensional modeling software. They are called languages since like English or French; they follow certain accepted syntax and semantics. Languages such as English and French are called Natural Languages. (Coates, 2011) Languages that are used to program are called Artificial Languages regardless of whether it is for a machine, assembly or high-level languages. (Coates, 2011) Ultimately both languages have similar functions. They are meant to communicate: natural between humans and artificial between humans and computing devices. The knowledge of syntax and semantics is important in natural languages but does not guarantee that a person is going to be a great creative writer. Similarly in the artificial language, knowledge of syntax is important but it does not guarantee communication of valuable ideas/processes. A series of routines — algorithms, is what is being communicated to the computer through artificial language. The design of algorithms — design computing, process design, objectile — is essentially the creative writing equivalent.

Both natural and artificial languages have evolutionary history. While the former enjoyed 100 000 years of development, the latter has been combined throughout an evolutionary period of about

65 years.(Coates, 2011) As mentioned earlier, programming language is a broad term that refers to three different kind of language types: Machine Languages, Assembly Languages and High-Level Languages.(Deitel, 2012) They descend in the order of accessibility. Most of contemporary programming is done through high-level languages. The high-level languages can then be divided into scripting based languages, programming and graph categories. Additionally, high-level languages are extensible, which means previously written code can be used, without re-writing the code, in the new algorithms as libraries (which is a combination of different codes). The basic structural principle of high-level language remains unchanged since 1966, and prior to that popular languages – BASIC and FORTRAN. These were structured around ‘goto’ statements which meant that after the sequential execution of the code the statement ‘goto line 52’ indicated the program will jump to line 52 of the code to continue executing. This structure had proven to be confusing and error prone if the code needed to be modified later on.(Deitel, 2012) In 1966, Corrado Bohm and Giuseppe Jacopini published a paper titled ‘Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules’ which demonstrated that programs could be written without any ‘goto’ statements.(Deitel, 2012) Their work established that all programs could be written with only three control structures: the sequence structure (consecutive line-by-line), the selection structure (if, if...else, switch) and the repetition structure (for loop, while, do while). This is important since programming languages communicate algorithms. (Chabert, 1999) This means that every algorithm can be built through three control structures. Algorithms can represent anything that is computable, or in other words, anything that can be mathematically represented.

4.2 Computability

Computability is important in the light of design computing, since generative systems (objectiles) are created through a series of algorithms. What follows, if everything is in fact computable, is that design computing systems can be created to represent everything the designer chooses to create, even design intelligence of him or herself.

Very simple examples of mathematical exponents can be easily codified and executed as algorithms. (Chabert, 1999) Codification and the creation of algorithms for other processes and human activities, such as creative thought, have been investigated prior to the use of computers. Among many, Gottfried Leibniz in the seventeenth century was the first to theorize that human thought could be broken down and represented through a series of binary decisions. (Steiner, 2012) He concluded, although without any adequate proof, that every decision is influenced by other external and internal parameters and previous experiences. Following that, Leibniz devised that human thought can be mathematically described and as a result codified as an algorithm. (Berlinski, 2000) In fact many human processes, previously non-mathematical in nature, were not showing to be an algorithmic reality. (Steiner, 2012) Theoretically anything could be mathematically described, as in computable, with the result represented by an algorithm. Algorithms, fueled by ubiquitous computing have essentially become part of our everyday life. A contemporary example of an everyday application is algorithmic trading on Wall Street pioneered by Thomas Peterffy in the 1970's. (Ibid) His algorithmic creation started at commodity trading, such as gold, which has few unknown parameters. Over the decade his algorithmic creativity moved to the stock options market where there were many unknown factors and parameters. Peterffy brought quantifiable systematic evaluations and speedy action to the market previously defined as a 'gut feeling' field. (Ibid) Speed was so crucial that in 2010, Spread Network (<http://www.spreadnetworks.com/>) built a fiber optic cable through a geographically straight line from New York to Chicago, just so algorithmic trading between the two cities could happen four milliseconds faster than before. (Slavin, 2011)

Another example of human activities which were previously thought of as unquantifiable include poker bots. (Steiner, 2012) There were inherent problems associated with creating algorithms for the purposes of playing poker. These difficulties could be thought as illogical decisions made by poker players with a bad hand, most commonly referred to as bluffing. Yet in 2005 Tuomas Sandholm, a computer science professor at Carnegie Mellon, and other CMU professors started

work on a poker playing algorithm. In 2012, his algorithm beat every human it faced when battling head-to-head in limit poker. (Ibid) However, Sandholm's algorithm lagged behind the best human player when it came to no-limit poker and a big number of people in a hand. In the industry, these varieties of algorithms are called game theory and they are closely modeled on varying methods of decision making processes of a human being. In essence, these algorithms, which follow game theory, incorporate many unknown variables.

Other examples of algorithms include music, which is closer to the creative field like architectural design. David Cope designed music-related algorithms. Cope used algorithms to create symphonies, operas and oratorios. (Steiner, 2012) Thus far, he has created three generative algorithms: Emmy (Experiments in Music Intelligence) in 1987 which created music in the style of Bach; Emily Howell in 2004 and lastly, Annie which David Cope is still developing. (Ibid) The importance of Cope's accomplishment is showing how music follows patterns which can be algorithmically recreated. Cope argues that artistic creativity can be defined by rule making and controlled rule breaking. Such creativity can be reduced to a long, efficient equation. (Ibid) So if music creation can be broken down into recognizable patterns that can be codified, could architectural design be enhanced with similar algorithmic creations?

Although it was written with many unknown variables, Tuomas Sandholm's poker playing algorithm performed in a game that had static rules. If the rules of the poker game were to change significantly, the algorithm would cease to be useful. Similarly in the case of David Cope, every algorithm is based on analysed music patterns of famous composers; however they will always be composed of music at a particular time he/she existed. If Peter Zumthor was to build an algorithm, it would only capture design practice patterns of the architect in this era. Future conceptual knowledge of Zumthor would not be embedded in the algorithm. Granted, the increase in quality of work in human beings is not guaranteed. Ultimately what is great architecture? Given the contemporary architecture pluralist theoretical approach, 'great' can mean anything depend-

ing on how the architect/designer positions his argument.

Such an all-around 'great' architectural construct algorithm is theoretically possible to create, however it might be utterly useless. It could suffer the same fate as dating websites, such as eHarmony and Perfectmatch.com, which utilize matching people into couples' algorithms. These algorithms work perfectly as they are told to, however research done by Eli Finkel, a professor of social psychology at Northwestern University, points to this matching algorithm being flawed through its underlying logic. (Steiner, 2012) In Finkel's words: "The reason for algorithms' failures is that most relationships are subverted by issues that emerge only after the couple meets and begins spending time together".(Ibid)

So can architecture be pristinely 'great' in all its aspects? Should architects and designers concentrate on the one argument they believe to be the most important and let objectiles, which are composed of algorithms, incorporate the 'great' of other arguments? Furthermore can this 'greatness' be defined in a project's early design stages and extend even further after ten or twenty years of being built? This would suggest some sort of feedback mechanism of the design computing system which is embedded in the actual construct. These questions lead to further investigation of the programming languages used in design computing systems, since algorithms are written in artificial languages.

4.3 Scripting, Programming and Graphs

There are literally hundreds of programming languages, ranging from data based communications to engineering purpose operations. Fortunately, there are currently only a few languages which are useful for architectural applications. This does not mean that certain programming language syntaxes are more applicable to design computing, but more so concerns itself with the ability of the language to handle computational geometry. The availability of extension libraries—a collection of predefined algorithms for use in any new code and an engaged community of users is also

Figure 53: Python IDE and API (left column) inside Rhinoceros Modeling Software

Figure 54: Grasshopper interface inside Rhinoceros modeling software

significant. The popular languages that fit this category are: Python, Java, Processing, C++ and Grasshopper. These languages are also in different high-level language categories of scripting, programming and making graphs.

Scripting languages are embedded in a particular software package; such as Python. Although it is a full programming language, it is also considered a scripting language. The majority of current 3D modeling software have an interface or IDE (Integrated Development Environment) that can be used to write code which can be run within the program.

The Application Programming Interface (API) includes documentation on how to create a particular geometry, routine and data manipulation code for the scripts. Although it is based on a full programming language, scripting language is essentially about running algorithms within a 3D modeling software package. The output of a particular code is created as geometry within a particular program. As a result, any further changes in the input parameters to create new variations and iterations of the design will require the code to be run again within the program. The benefits of scripting include geometry generation through existing manually modelled digital geometry as the input. A series of scripts can be used to compose algorithms which can then generate entire components, building massing and eventually, finished building designs. Recent years have seen the adaptation of the Python programming language as a scripting language for a variety of 3D modeling software packages, including Autodesk Maya, Autodesk Softimage, Cinema 4D, McNeel Rhinoceros and Modo.

Graph-type languages operate in a similar manner to scripting since they are based on a specific program. The major difference is that the algorithms are represented through visual means as predefined generic codes which are represented by an icon. The icons are then joined together with visual wires, which then compose the entire algorithmic routine. The codes created, just as in scripting languages, can be reused in other projects assuming that the same three dimensional

modeling software is used. Since the graph language is in itself a piece of software, such as the case of the Grasshopper plug-in for Rhinoceros, the code cannot be translated to the other 3D modeling packages as in cases which use Python scripting language.

A full programming language, or main programming language, is independent of any software package. The drawback is the required investment of learning how to compose graphical user interface algorithms and other manipulation devices such as sliders for the input of data manipulation. Object-state geometry that is created can be imported into any modeling software for further manipulation or traditional documentation. Furthermore, as in the case with Java and Processing programming languages, the defined algorithms can be distributed over the internet as standalone generative objectiles. Programming languages can include infinite drawing loop algorithms which essentially means that geometry is regenerated every millisecond. Through this method, different input parameters are produced with variability and iterations can be explored while running the code without creating a particular object-state (geometry) or termination of the code.

Any program or tool can be created by following syntax rules of a particular programming, scripting or graph language- which is not unlike following the syntax of English, French or any other natural language. People communicate by following accepted natural language syntax; we additionally acquire generic phrases and figures of speech which can be considered generic algorithms. Utilizing algorithms in artificial language requires a different method of thinking. Since the algorithm represents an objectile- which is an elastic notation of designed possibilities, the designer also needs to operate in an elastic, relational manner.

4.4 Algorithms

A simple definition of an algorithm is creating a series of steps in order to arrive at a particular result. This can be applied to almost any human endeavor, such as drinking a cup of water. An

algorithmic definition in mathematics and computer science constitutes an effective method or procedure which is expressed as a finite list of logically defined instructions for calculating a function. (Poletto & Pasquero, 2012) Algorithms begin with various input parameters and after a series of transformations, deliver an output. Algorithms can also be stochastic which means the same input parameters deliver different outputs. Such algorithms include chance and random functions as part of their overall composition. (Terzidis, 2006) The origin for the term algorithm was derived directly from al-Kwarizmi, the author of the oldest known book on algebra during the ninth century of the Common Era. (Chabert, 1999) However, the algorithm as a methodology existed well before the time of al-Kwarizmi, as basic arithmetic operations used by Sumerians in 2500 BC. (Ibid) Very simple examples of mathematical exponents can be easily codified and executed as contemporary computer algorithms. The repertoire of algorithms can be extended further from simple arithmetic to creative processes such as architectural design.

Figure 55: Depiction of algorithms in cinema– a scene from the movie The Social Network.

Algorithms as discussed previously can be applied to a variety of problems which were previously thought to be incompatible to such a solution. More importantly, algorithms are becoming an important cultural construct. These include Amazon book suggestion engines to Netflix new movie suggestions. Services that run algorithms to generate information and sales have become extremely valuable and relevant. (Slavin, 2011)

Algorithms nevertheless remain seen as a subject primarily of the computer science and mathematics fields. This can be confirmed by the majority of the algorithm books which address the design of a variety of algorithms for their further use in programming languages. Many algorithm design manuals are irrelevant to the architectural field at large—except for the algorithms which concern themselves with computational geometry. Understanding geometrical algorithms is important when considering complex formal expressions and variable elements of a design system. This significance is exemplified by newly formed consulting practices and in-house departments that deal with complex, parametric geometries. A non-inclusive list of such consulting offices includes Design-to-Production, Case Inc., Aedas R&D and Norman Foster's Specialist Modeling Unit. Computational geometry algorithms, such as the definition of a vector, can be imported into a design code of a library from previously defined codes. This allows an engagement with computational geometry as a tool in generating objectiles and not a topic of intense research in-itself. Algorithms which are more significant to architecture have existed prior to the creation of computational devices. In architecture, these algorithms have often been referred to as typology. In 1978, Rafael Moneo in his seminal essay 'On Typology' discussed different architectural ideas regarding typology. Most importantly, he refers to type as something that is transformable and composed of elements. Furthermore he argues against the view that typology is a "frozen mechanism", adding that "... the very concept of type, as it has been proposed here, implies the idea of change, or of transformation. The architect identifies the type on or with which he is working, but that does not necessarily imply mechanical reproduction. ... The type can thus be thought of as the frame within which change operates." What Moneo is describing in his essay as type can be closely compared to an objectile—a design computing system composed of a series of algorithms that generate object-states. Typology can be thought of as being composed of a series of algorithms. An objectile following that definition is created as a design-in-itself construct, seen in the example of the Turing Pavilion by Biothing in the previous chapter.

It can also be argued that architecture has always materialized itself through algorithms. Building

is a material representation of the algorithm which is run during its construction. One example, as discussed previously, are Peter Eisenman's Houses which represent algorithms of geometrical transformations from a basic volume. Working drawings can be equated to material assembly algorithms, represented both in drawings and written notational forms. Furthermore, an architect's stamp on a set of working drawings signifies that "this material assembly algorithm will run properly" – of course in most cases this algorithm requires debugging. In fact debugging, which refers to finding and fixing coding and logical errors of a programming code that failed to compile and run on the computer, is exactly what some architects or project managers do during the construction phase of a typical architectural project. This potential debugging drives the architectural industry to using known and tested designs and in particular, material assembly algorithms that are in essence composed from the same objectile. The major difference in the above mentioned algorithm from a series of algorithms which define design computing objectiles is the explicit nature of the latter.

Some architectural projects try to incorporate the explicit nature of algorithms in both their design and construction. This is exemplified in the Institute of Computational Design (ICD) 2012 Pavilion and The Truffle project by Ensamble Studio. Both redefine conventional algorithms of material assembly (construction) for their object-states. The former is both computationally designed and manufactured while the latter uses explicit algorithms in an analogue approach.

The ICD Pavilion is an annual project fueled by material computing research lead by Professor Achim Menges at Stuttgart University. This year's research explores material techniques in designing and fabricating a pavilion from woven strands of glass fibre. From the videos shown by one of the participants in the research – David Correa, it is evident that the fabrication of the pavilion is done through the choreography of a 5-axis robotic controlled arm and circling platform where the pavilion's scaffolding is placed. (Correa, September 2, 2012) This harmonious movement is an explicit computer algorithm of an objectile materializing the final constructed pavilion. The

Figure 56: Overlay of Western Dwellings– A Housing Objectile?

design of an objectile in the ICD 2012 Pavilion is considered the design of a generating system and the design/execution of the manufacturing system of a specific object-state.

In the Truffle project, it is not clear if the architects have used design computing in their work. Nevertheless, the construction sequence is an explicit analogue algorithm– most likely not previously used to create architectural projects. First, hay bales were placed according to the design. Concrete was then poured around the bales with rammed earth acting as formwork. Once the concrete was set the result was excavated and cut in a specific location on both sides, revealing the blocks of hay inside. Next, the hay was used as food for a local cow, which inhabited the space and day by day revealed the negative volume which was once created by the hay blocks. The final result was a concrete shell, cast as a result of the negative hay bale space.

The construction of architecture in the projects above is inherently composed of explicit algorithms. There is a twofold importance to this. First is that the construction process can be thought of as a novel algorithmic process– hence the design of choreography for material assembly. This is arguably how every architectural project should be materialized. Secondly, design computing can be further extended from the design of generative design systems, objectiles, to design construction algorithms.

Figure 57: ICD 2012 Research Pavilion

Figure 58: The Truffle by Ensamble Studio. Hay Assembly and Concrete Casting.

Figure 59: The Truffle by Ensamble Studio. Cutting and Excavating.

Figure 60: The Truffle by Ensamble Studio. Finished dwelling

myCabin by Author

This project explores explicit analogue algorithms through mixed analogue and digital methods of production. The objective of the architectural construct was defined to be a vacation cabin with minimal functional requirements. These included a multi-purpose space, washroom/shower, kitchenette and sleeping area. The author proceeded with thinking of the site conditions that the finished architectural construct might find itself in. Two conditions were imagined: (1) next to the lake cabin and (2) in the woods cabin. The author defined the objectives of each cabin, with the first one being a gateway between land and water and the second being concerned with unobstructed views.

The two cabin conditions were preliminarily sketched, as formal and functional materializations. The first result was a painfully literal gateway to the water and the second sketch pointed toward a watchtower through its form. The first concept was further developed from sketches to a digital model. This digital model incorporated a window component as its objectile. Smaller windows were scattered on the surface of the cabin, with a higher concentration around the kitchenette and sleeping area on the interior.

Findings: an Objectile is formed when one or two attributes are attached to a generic notion of something. Once the objectile gathers enough attributes, it actualizes itself in an object-state. The initial attribute can be thought of as a design objective. This reinforces an idea of an objectile being an algorithmic definition of typology. The myCabin defining design objective was a formal one of a gateway cabin. This attribute is enough to define an objectile that can be algorithmically described, as a result, creating a variety of object-states.

Arguably each object-state is different from one another, as per figure 66. At the same time each

Figure 61: Lakeshore cabin (left); Woods cabin (right)

Figure 62: Study of Lakeshore Cabin. Preliminary Plan and Sectional Sketches.

Figure 63: Studies of Lakeshore Cabin.

Figure 64: Lakeshore Cabin Material Scheme and Setting.

Figure 65: Lakeshore cabin material scheme and setting.

one has a visually recognizable connection to an idea of a gateway cabin as an initially defined design objective. The algorithmic definition of this object created here only allows for the creation of gateway cabins and not Wood Cabin as per figure 61. It is possible to combine each definition together allowing for initial parameters to be shared. For example both cabin definition might start as an n -sided polygon shape in the algorithm but further into the bifurcation of the code the cabin takes the form of gateway or woods cabin.

Furthermore a decision of whether the Wood Cabin or Lakeshore Cabin object-state is created can be connected in the code to external parameters, such as geographical location, that are continuously monitored. This way the user input is minimized and decisions are automated. Such a scenario of automation might not be desirable or needed for each and every project. Additionally the time invested into creation of such code might not be beneficial to the overall architectural project. If the design decision of geographical location to overall building form must be carried out over several project iterations or different but similar projects then computer algorithm development might be a sound plan of action. Thus appropriate use and creation of digital tools must be evaluated per-project.

This need for evaluation is ultimately connected to the nature of algorithmic thinking described in the scenario above. Additionally it points to the apparent simplicity of the process where human cognitive action can create valid results just as quickly as a computer can, although arguably in some cases even quicker. The only time a computer algorithm is beneficial, as stated above, is the case of a repeating volume of similar process. This kind of process design is categorized by the author as Automation, in contrast to the Automation as Novel category.

Figure 66: Possible myCabin outcomes

Figure 67: Algorithmic (Graph) definition of myCabin objectile

Figure 68: PixelFacade Perspectives

5 AUTO/NOVEL

Although the author categorized process design as either Automation or Novel both categories should be thought of as an extreme end of a gradation of process design approaches to a given project. Ideally in any given project, that is benefiting from design computing, there will be different gradients of approaches – some siding closer to automation other to novel. Any given extreme of the gradient is not necessarily better than the other, but rather should be applied when appropriate.

Previous chapter – Interlude IV, included project titles myCabin. This project can be categorized as a automation approach example. Similar results to the variable outcomes of the algorithm can be achieved through manual computer modeling. Additionally the design itself does not require any digital tools and computer use at all to be carried out. Another example of an automation approach was introduced in Interlude II titled PixelFacade. This project is further away in the spectrum from extreme automation, in comparison to the previous example of myCabin. This is due to the volume of the panels involved in the making of the façade. Time consuming manual modeling techniques will limit the amount of possible iterations that could be produced on such a project. Algorithmic description of the PixelFacade allows one to create virtually unlimited number of iterations, given upfront creation of a computer algorithm.

Daylighting Factor app by the author represents a similar example to the PixelFacade project. The difference being while the latter algorithm creates representational geometry as a final output, the former is created as a daylighting design helper tool. The algorithm calculates daylighting factors for a given room and window opening dimensions specified by the user. This results in a real-time design criteria evaluation offered to a designer who might not be knowledgeable

Figure 69: Daylighting Factor App

Figure 70: Garden Wall project. Relationship between existing surface mapping and louver mesh

enough in green design strategies, specifically passive lighting.

Garden Wall — a project created by author that was introduced in previous chapter, Interlude I, steps further away from the automation end of spectrum and much closer to novel. There are a variety of factors that contribute to the novel dimension. First is the amount of variables that need to be considered for the vegetated louver façade system. These include: proximity of existing window openings, orientation, plant life appropriate to given shading level, weaving pattern density and structural anchoring. The variety of parameters to consider leads to the overall project being composed of a series of algorithms. These processes together create an interactive system capable of producing iterative results for a given specific condition. Furthermore the project being a strategy for retrofitting existing facades with a vertical green wall and louver system, it should be a repeatable process for similar conditions (vertical building wall with window openings and solid walls).

While the automation approach was described in the previous chapter, it is important to point out what a novel approach in design computing actually embodies. Novel approach refers to the process design that makes a particular architectural project possible which with conventional design means would be prohibitive to carry out. This can be due to taxing iterations, complex interactions and complex geometry. Some examples among many projects utilizing design computing in novel approach are: 2012 Pavilion by ICD, Onthevergeofcollapse by MOS, Turing Pavilion by Biothing.

ICD Pavilion was discussed in the previous chapter for its role in redefining conventional algorithms of material assembly (construction) for its object-states. The construction of the pavilion did not have to be carried out with the use of mechanical agents (robotics) to be completed. Nevertheless even if only human labour was used the project would still be an explicit algorithmic process that redefines current construction methods. Arguable translation of the designed object-state to materialized artifact would have been an arduous task to carry out without the involvement of robotics.

Figure 71: Onthevergeofcollapse by MOS

Figure 72: Turing Pavilion object-state by Biothing

On the verge of collapse by MOS can be considered as a series of projects that utilize the same algorithm. Essentially it is a physics based engine custom written computer software, that lets the user stack blocks in a designed or random order without the use of adhesive. Through this custom made design tool, project iterations as a result of object-states creation can be achieved fluidly. The results of the outcomes, as per Figure 71, show the complexity of the parts involved in a seemingly simple project.

Turing Pavilion by Biothing is a good example of a project on the extreme end of the novel spectrum. Creation of an object-state of the pavilion is generated through the simulation of Reaction-Diffusion chemical reactions on a XY plane progressing in Z-space with each new movement of the particles. Upon completion an irregular mesh is created, which cannot be constructed with any conventional construction techniques. Thus in order for the object-state to be materialized a novel process of construction should be developed. Instead of developing such a process the Biothing team used 3d printing to create a scaled model. One can see such geometry needs to be printed (materially distributed instead of materially assembled) with the emergence and future development of additive manufacturing on a construction scale.

The common theme of the projects above, and as pointed out in previous chapters, is the apparent need for the algorithm to be repeated multiple times over the course of the specific project. If the interactive system, algorithm, was created specifically for the purposes of a specific project, usefulness of such algorithm becomes limited. On the other hand algorithms that operate on very generic and omnipresent input parameters, such as façade surfaces, lighting conditions, vegetation categorization, can be used on variety of projects that share similar parameters. Furthermore there is a link between projects leaning toward each end of the spectrum, from automation to novel. Automation projects tend to be implementing typical construction methods that have known, tried and tested, explicit processes of material assembly. While other projects leaning towards the novel extreme of the spectrum implement more unique construction methods, re-

quiring research and development, methods of material assembly and distribution. Hence the connection between design and materialization algorithms is evident. Furthermore it can be said that architecture is a manipulation of space through the materialization of ideas. Materialization is constrained by means of construction. Construction is an event creating material formations. It materializes through interaction of materials and agents (both human and mechanical). These interactions are a series of algorithms/processes resulting in digital and physical outcomes. Inversely design of algorithms/processes (both digital and physical) results in the interaction of material and agents that manipulate space — creating built form.

In other words although we are capable of creating a variety of novel forms, through design computing means, we are always constrained by our means of production. As a result, the benefits of the ease of creating complex forms through design computing means are not of a real benefit at the current time.

Design computing contributions to the nature of the design process discussed thus far include: notions of objectile and object-states, variable element repetitions and spectrum of automation to novel approaches in physical manifestation. Additionally, as discussed in Interlude III — City Blocks, we saw that in the algorithmic approach it is important to know which general conditions exist and which do not, rather than the specificity of each individual condition. The specificity is addressed through interactive system design (objectile creation).

Following all of these concepts it is important to portray design computing's potential roles in an architectural design project. The authors project titled Rhizome serves as an example pointing towards the potential benefits and shortcomings of design computing approaches in an architectural project setting.

The Rhizome projects incorporates process design — creation of interactive systems embedding notions of an objectile with each iteration being an object-state. Rhizome is composed of a variety of algorithms as opposed to one all-encompassing algorithm. Each algorithmic process is designed to deal with a specific condition, whether massing, structure or aperture (window openings) placement, while delivering a family of potential object-states.

The breakdown of the design project into a series of algorithms is informed by the experiments from the Interlude sections. In particular what became apparent is the difficulties of code revisions, thus different design outcomes, when a design project is created through one mega-algorithm. The initial design outcomes of both one mega-algorithm and multiple algorithms can be identical, however the logistical benefits of the latter become apparent when design changes need to be introduced into the project. These changes might only apply to one particular algorithm in the series and not the entire process. An example of this will be further illustrated in the Rhizome project.

An idea of a tree

Figure 73: Composition of an objectile

Site

The site is a generic beach shore, while the program is defined as a gradient of activities. The generic beach site is composed of three geologically different areas of water, sand and forest. Additionally the site is located in climate with four different full seasons, with typical weather and diurnal patterns. Although seemingly a simple site it allows for a variety of heterogeneous environments. These environments also have further variety on a finer scale, such as specific locations of an individual tree.

Program

In a traditional design sense the program of Rhizome project includes various pavilions that support a variety of beach activities throughout the site. Through the design process it became evident that program does not specifically require the use of design computing techniques at this scale of project. The use of design computing is rather informed by design choices involving variable element repetitions and project specific routines requiring design iteration within established design limitations. The former can be best described as geometry based algorithms, whereas an example of the latter is site placement explorations.

Figure 74: Composition of Rhizome Pavilion objectile

Figure 75: Generic Site. Perspective.

Figure 76: Generic Site. Plan.

Figure 77: Generic Site. Perspectives and plans showing heterogeneous conditions at different pavilion locations (red squares)

In order to describe the whole depth of methodology used in the Rhizome project the following text is divided into five sections. Each section describes ideas and design iterations undertaken. The overall Rhizome workflow consists of the following five steps:

1. EXISTING CONDITIONS: Site mapping (includes symbolic notations for use in further algorithms)
2. DIAGRAMMING: Relation establishment (internal/external/sub parameters)
3. DESIGN AIDS: Program site mapping (includes symbolic notation of program gradient)
4. DESIGN ITERATIONS: Static (objects/object-states) and dynamic models (objectile)
5. (v)Architecture (from specific to hyper-specific, virtual and actual)

(1) Site mapping in a traditional architectural project is a survey and documentation of existing conditions in a static drawing form, along with the site being analysed by an architect and external consultants (for example, to determine soil conditions). This static mode of representation includes any traditional form of representation: including drawings, diagrams, etc. In contrast to the static is a dynamic representation, which in a design computing project means algorithmic representation or creation of a dynamic interactive system. It is appropriately dynamic due to an algorithm's ability to capture variable outcomes using the same routines with different input variables.

Site mapping in the Rhizome project follows a hybrid (static/dynamic) model. Static representation is a three dimensional topological computer model with additional points and polylines representing site trees and different areas of the site, respectively. The dynamic part of the model is an algorithm that turns points and polyline areas into symbolical notations. These notations are simple geometries (lines) at a specific length, each having a specific meaning in further algorithms.

Figure 78: Screenshots of site model with dynamic site mapping algorithm applied

This routine is captured in three screenshots of figure 78. The first screenshot (upper) is showing a generic model, which is typically used in any current digital architectural workflow. This model incorporates site contours and areas of water and land. The second screenshot (middle) shows the dynamic mapping achieved through a graph algorithm inside a three dimensional modeling program. This algorithm creates three different areas of water, sand and soil/trees. In the above screenshots these areas are represented as blue, black and green respectively. The last screenshot (bottom) shows simple line geometries generated with each line having a length of either 1000, 2000 or 3000 (although any three different number of lengths can be used). The length corresponds to the line belonging to either water, sand or tree areas, as a result acting as a symbolic notation for further algorithms, such as program site mapping.

(2) In order to design dynamic models, parameters driving the potential design have to be explicitly established as well as any possible relationships between these parameters. This parameter/relationship establishment can be achieved through static modes of representation, such as sketches and diagrams. Hand sketches shown below can be thought as initial parameter sketches for a specific objectile in the Rhizome project.

These parameter sketches are further detailed through diagrammatic means. The diagrams establish relationships of different input parameters that can be used as a guiding principle of a dynamic system's design (algorithm and hence an objectile). Two such diagrams of 'Overall Relationship Mapping' and 'Program Mapping' are shown below.

The diagram's purpose is to visualize all possible relationships between different aspects of the input parameters. These connections are used as a guiding principle for the inclusion of generic conditions in the form/geometry generating algorithms. Generally these conditions can be divided up into two categories of external and internal constraints. The former includes possible objects and phenomena that can exist on a generic beach shore site. The latter includes spe-

Figure 79: Sketches of Orange pavilion belonging to trees area objectile.

1.washrooms 2. suntanning 3. change rooms (clothing/skates) 4. lockers 5. shade 6. day-dreaming
7. star-gazing 8. reading 9. relaxation/chill-box 10. chatter-box 11. kayaking 12. water jumping 13.
ice fishing 14. lighthouse 15. rain/fog/snow pavilion 16. bonfire 17. blackout-box

— complementary
- - - conflicting

Figure 80: Overall relationship mapping diagram

cific parameters that can be used as an objective in the objectile. External condition can also be thought of as parameters that are typically present, such as weather and diurnal cycles. Internal condition are less rigid and can include subjective parameters established by a designer. It is important to note that it is logistically impossible to account for every possible input parameter that can be affecting an architectural objectile. Additionally some parameters can be considered as intangible/subjective such as designer/client/user individual preferences. Overall the Relationship Mapping diagram also clearly divides up the program into three gradients. The three colours, of pink, orange and blue, represent the three pavilion types corresponding to different envisioned program varieties.

These program variations are grouped together by the key words of support (pink), relaxing (orange) and active (blue) as per Figure 80. Rather than over-specifying what activities can happen at a particular pavilion, the objective is to create places of possibilities while hinting at a potential program gradient. Thus the three areas of program gradient represent overall objectiles. Each overall objectile is divided up into three or two main geographical areas as per above figure. All other parameters are integrated in the appropriate geographical objectile as per previous figure.

(3) While the majority of digital tools created in the Rhizome project either generate form or a notational system for further algorithms, other tools were created as design aids. 'Program Mapping' is one such algorithm with the purpose of situating potential object-states on the site. This tool consists of visual placement aids that allow the designer to see the main generic parameters considered in the given objectile as well as maximum generative boundaries of the object-state. The Program Mapping algorithm uses symbolic notation generated by the Site Mapping code to recognize where the current area of a particular pavilion is located (whether water, sand or woods).

Once pavilion locations are established the Program Mapping algorithm generates a land swatch of the given location for the form/geometry generating algorithm.

(4) While all of these Rhizome pavilions are dynamically represented, some designs began as traditional static representations in the form of sketches and models, while others were established and designed throughout the processes of algorithmic definition writing. Furthermore traditional three dimensional modeling was used throughout the process, both at the beginning of the design and in different iterations. Thus it is important to note that a design computing workflow does not simply replace the established methodologies in architecture but rather augments them.

Two pavilions of pink (support) and orange (relaxation) gradient were further developed, within these are five objectiles with three belonging to the orange variety (water, sand and trees) and the two to the pink (sand, trees). It is important to note that there are technically five objectiles and not two. Each design was rooted in the geographical location as a base attribute of an objectile. Theoretically it is possible to combine all the location based objectiles into one overarching algorithm, however such a process does not contribute to the creation of an architectural object-state but rather becomes a computer science project. Additionally such an overarching algorithm would be difficult to work with due to its complexity.

Relaxation and support pavilions mentioned above are documented/illustrated in Appendix I. Orange (relaxation) pavilion in the woods objectile was developed in finer detail and is extensively documented/illustrated below.

Parameters of the relaxation pavilion in the woods include tree proximity, site contours and sun direction. Its primary construction material is waffle wood framing. Additionally a variety of small to big sized apertures are scattered around the surface of the exterior wall. These apertures are placed through algorithmic means in accordance to whether to avoid a sun path or allow sun into the pavilion at a particular time of the day.

Figure 83: Relaxation Woods static models. Used throughout design

Figure 84: Orange Woods. Possible object-state.

Figure 86: Variation within object-states of same objectile

Figure 87: Physical Model of Two Dimensional matrix of possible object-states

The Relaxation pavilion is composed of three different algorithms. In the initial iteration of the routine the first algorithm creates overall massing, while the second and third algorithms create surface apertures and structure respectively. Results of the first process are illustrated from figure 85 to 87. This dynamic model incorporates tree proximities, site contours and primary sun direction as main driving parameters. Additionally the massing form can be further adjusted with internal parameters such as angle toward/away from the sun, main structural rib depths and heights.

Furthermore effects of the massing models deformation according to tree proximity can be decreased or amplified beyond initially set values. Additionally a change in the physical location of a pavilion changes input variables and as a result produces different outcomes. (Illustrations showing complete generative process of the massing algorithm can be found in the Appendix I.)

Figure 88: Rhizome - Wood Pavilion objectile diagram

The two following algorithms, in particular the structural system, required initial explorations through static models as portrayed in Figure 89. This static exploration, although it was seemingly working in a particular case, showed itself to be incapable of being applied to a variety of different massing results from the first algorithm. Following the structural static model results the algorithmic model was based on a simplified waffle approach.

Figure 89: Object-state location based variations

Figure 90: Structure studies

The first routine iteration placed the structure algorithm prior to the aperture algorithm, as per above figure, as a result making it difficult to create a cohesive outcome. Consequently the generating algorithm of the apertures and structure of the relaxation pavilion went through a series of iterations. As mentioned before the initial routine created a conflicting layering effect, limiting sun light penetration. Although that might be desirable in certain cases it also limited the potential lighting effects. As a result further iterations were designed to work with the aperture algorithm running prior to the structural waffle algorithm. The latter algorithm then takes symbolical notations from the aperture algorithm to avoid the conflicting layering effect. In essence iterations of the relaxation pavilion can be thought of as a refinement of the process. Further refinement of the process allows for cohesive results with further aperture variations as shown on 'gen-3' figures.

Figure 91: Initial structure and aperture openings algorithmic order

Figure 92: First algorithmic generation physical model

Figure 93: Second algorithmic generation. Structure and apertures.

Figure 94: Third algorithmic generation. Structure and apertures.

Figure 95: Third algorithmic generation. Apertures and Inner/Outer paneling.

Figure 96: Third algorithmic generation. Apertures and Inner/Outer paneling.

Figure 97: Third generation algorithmic pavilions. Location and designer based variation.

(5) The Relaxation Pavilion in the woods is created through digital interactive tools described above. The design of it is represented through a dynamic model based on specific, yet general, conditions. They are specific due to the incorporation of specific parameters, such as tree proximity. General in a way that the precise position of trees does not matter at the time of the algorithmic definition. This precise position becomes specific once the algorithm is run and the dynamic model transforms to the final object-state.

Dynamic representation of the pavilion allows for object-state results to vary with different external input, such as location, and/or new internal value such as heights and amplification values. This greatly expands creative possibilities, deriving process design outcomes that might have not been considered through the workflow of a traditional design process.

The workflow used in the Rhizome project is not fully applicable to every single traditional architectural project. The extent of the application of the design computing in some projects will be to the condition of an architectural façade. A façade lends itself to being a highly generic site since most façades are generally composed of continuous geometrical planes that are subdivided. Thus projects incorporating variable element repetition, whether at an aperture scale or building scale can benefit from algorithmic workflow. Figures above show the variety of building options which it is possible to achieve through algorithmic routines in a short period of time.

In Closing

Objectiles of design computing have embedded social constructs of computing devices (calculation, logical formalization and automation). They are systems that have been informed by many 'how-to' centric approaches from inter-disciplinary knowledge. While the 'why' question can be connected to the legacy of the post-modern concerns, this question should not overshadow what design computing can do. In other words it might not matter why anymore, but rather what it can allow.

Computers are thought of as universal machines capable of creating any other machine. In architecture it holds the promise of creating universal architectural machines capable, in turn, of creating architecture. Such promise however is unfulfilled and highly unlikely to ever be completely fulfilled. Computers can be used to create digital tools that either aid architectural design and thought or automate form generation, instead of creating architectural machines.

These tools are created from algorithms and as the author has demonstrated a series of a few algorithms are needed in order to arrive at a complete architectural project. Additionally these algorithms are designed through an iterative process. Iteration of these processes is simpler to maintain through smaller chunks rather than one mega-algorithm. Collection of algorithms in essence are explicit design intelligence routines documented through a particular time. What this allows is a library of algorithmic routines not unlike a library of CAD-blocks or more closely Revit-families (since Revit families can include a variety of objects in one file), which in theory can increase the potential efficiency of an architectural office. Unlike libraries of CAD-blocks and Revit-families, however, algorithms can benefit from continuous feedback loops improving their material assembly routines or any other processes that would be later tested in the field conditions.

Algorithms are written down in order to be repeated. As a result in an architectural context design

computing is worthwhile if repetition is present. This repeating can be interpreted as family of object-states generated through a design strategy rather a singular building project. Additionally repeating can be interpreted as elements of an architectural project such as façade panels. Furthermore since algorithmic tools are capable of creating family of object-states, design computing is effectively used when variable elements repetition is needed in an architectural project.

It is evident there are pragmatic benefits in using design computing. However, the true shift is in design thought. A shift from design process to process design allows for an elastic approach to project changes, within a designed algorithmic scope. Architectural designers, as a result, have a tool which provides design variations for evaluation and flexible updates to design parameter changes. The designer, however has to think of built form as an organic entity where relationships matter and the specifics, such as dimensions, adapt. The change from an architectural object to an architectural object-state is less about tool use. This shift is a change in the designers frame of mind which embeds an algorithmic process design in the architectural design process.

Figure 98: Third generation algorithmic pavilions. Sectional perspective. Three variations showing different experiential qualities.

Figure 99: Third generation algorithmic pavilions. Sectional perspective. Pavilion with moderate amount of apertures

Figure 100: Third generation algorithmic pavilions. Sectional perspective. Pavilion with large amount of apertures

Figure 101: Third generation algorithmic pavilions. Sectional perspective. Pavilion with minimum amount of apertures

APPENDICES

Appendix A:
Additional Project Images

Appendix B :
Geometry generation process

Appendix C:
Source Codes

APPENDIX A

ADDITIONAL PROJECT IMAGES

Wood pavilion exterior perspective.
Algorithm generation one. Possible outcome

Wood pavilion – interior looking up.
Algorithm generation one. Possible outcome

Beach activity pavilion.
Algorithm generation one. Possible outcome

Beach support pavilion.
Algorithm generation one. Possible outcome

APPENDIX B

GEOMETRY GENERATION PROCESS

Beach activity pavilion
Generation Process

Water pavilion
Generation Process

Wood pavilion
Generation Process

$\vec{e}_x$

$\vec{e}_x$

$\vec{e}_x$

Beach support pavilion
Generation Process

Wood support pavilion
Generation Process

APPENDIX C

SOURCE CODES


```
1 //Demonstration of code representation
2 //By:Kyrylo Lobach
3 //Date: 120919
4 package longThinYellow;
5 import processing.core.*;
6 import igeo.*;
7 import java.util.*;
8
9 @SuppressWarnings("serial")
10 public class LongThinYellow extends PApplet {
11
12     Scanner input = new Scanner(System.in);
13     private IVec initPt1, initPt2, endPt1, endPt2;
14     private ArrayList <IVec> legOneVec = new ArrayList <IVec>();
15     private IVec[][] legOneArray;
16     private ArrayList <IVec> legTwoVec = new ArrayList <IVec>();
17     private IVec[][] legTwoArray;
18     private int numOfGen;
19     private double width;
20     private ArrayList <Double> widthDec = new ArrayList <Double>();
21     private double genHeight;
22     private double overallHeight;
23
24
25     public void setup(){
26         size(800,800,IG.P3D);
27         IG.background(200);
28         IG.pers();
29         existing();
30         createLegs();
31     }//end of setup method
32
33
34     private void createLegs(){
35         setInitVec();
36         setVar();
37         setEndPt();
38         createSumVec();
39         setWidthList();
40         createLegsArray();
41         createLegsSurface();
42     }//end of create method
43
```

```

44
45 private void createLegsSurface(){
46     new ISurface(legOneArray,1,2,true,true).clr(IRandom.geti(230, 255),
47     IRandom.geti(230, 255),IRandom.geti(0, 255));
48     new ISurface(legTwoArray,1,2,true,true).clr(IRandom.geti(230, 255),
49     IRandom.geti(230, 255),IRandom.geti(0, 255));
50 }//end of create legs surface method
51
52
53 private void createLegsArray(){
54     legOneArray = new IVec[widthDec.size()][4];
55     for(int i=0;i<legOneArray.length;i++){
56         legOneArray[i][0] = legOneVec.get(i).dup();
57         legOneArray[i][0].mv(0, widthDec.get(i), 0);
58         legOneArray[i][1] = legOneVec.get(i).dup();
59         legOneArray[i][1].mv( 1*widthDec.get(i), 0, 0);
60         legOneArray[i][2] = legOneVec.get(i).dup();
61         legOneArray[i][2].mv(0, 1*widthDec.get(i), 0);
62         legOneArray[i][3] = legOneVec.get(i).dup();
63         legOneArray[i][3].mv(widthDec.get(i), 0, 0);
64     }//end of i for loop
65     legTwoArray = new IVec[widthDec.size()][4];
66     for(int i=0;i<legTwoArray.length;i++){
67         legTwoArray[i][0] = legTwoVec.get(i).dup();
68         legTwoArray[i][0].mv(0, widthDec.get(i), 0);
69         legTwoArray[i][1] = legTwoVec.get(i).dup();
70         legTwoArray[i][1].mv( 1*widthDec.get(i), 0, 0);
71         legTwoArray[i][2] = legTwoVec.get(i).dup();
72         legTwoArray[i][2].mv(0, 1*widthDec.get(i), 0);
73         legTwoArray[i][3] = legTwoVec.get(i).dup();
74         legTwoArray[i][3].mv(widthDec.get(i), 0, 0);
75     }//end of i for loop
76 }//end of create legs array method
77
78
79 private void setWidthList(){
80     double widthDecNum = width/numOfGen;
81     for(int i=0;i<numOfGen;i++){
82         widthDec.add(widthDecNum*i+0.5);
83     }//end of i for loop
84 }//end of set width list method
85
86
87 private void createSumVec(){
88     for(int i=0;i<=numOfGen;i++){

```

```

89     legOneVec.add(initPt1.sum(endPt1,(double)i/numOfGen));
90 }//end of i for loop
91 for(int j=0;j<=numOfGen;j++){
92     legTwoVec.add(initPt2.sum(endPt2,(double)j/numOfGen));
93 }//end of j for loop
94 }//end of create summation vectors method
95
96
97     private void setEndPt(){
98         IRandom.initByTime();
99         endPt1 = initPt1.dup();
100        endPt1.x(endPt1.x+IRandom.getDouble(0,3));
101        endPt1.y(endPt1.y+IRandom.getDouble(0,3));
102        endPt1.z(overallHeight);
103        endPt2 = initPt2.dup();
104        endPt2.x(endPt2.x+IRandom.getDouble(0,3));
105        endPt2.y(endPt2.y+IRandom.getDouble(0,3));
106        endPt2.z(overallHeight);
107    }//end of set end point method
108
109
110    private void setVar(){
111        System.out.print("Please input number of generations " +
112        "(integer between 5 and 10): ");
113        numOfGen = input.nextInt();
114        System.out.print("Please input height between generations " +
115        "(double between 2 and 7): ");
116        genHeight = input.nextDouble();
117        overallHeight = numOfGen*genHeight;
118        width = IRandom.getDouble(2,7);
119    }//end of set variables method
120
121
122    private void setInitVec(){
123        IRandom.initByTime();
124        initPt1 = new IVec(IRandom.getDouble(0, 25),IRandom.getDouble(0, 25),0);
125        initPt2 = new IVec(IRandom.getDouble(0, 25),IRandom.getDouble(0, 25),0);
126    }//end of set initial vectors method
127
128
129    private void existing(){
130        new IPoint(0,0,0).clr(255,122,5);
131        IVec[] recVec = new IVec [4];
132        recVec[0] = new IVec(0,0,0);
133        recVec[1] = new IVec(0,25,0);

```

```
134     recVec[2] = new IVec(25,25,0);
135     recVec[3] = new IVec(25,0,0);
136     new ICurve(recVec,true).clr(80);
137     }//end of existing method
138
139
140 }//end of long thin yellow legs class
```


vacONE Source Code by Author; Java + Processing Libraries

```
1 //Vacation Cabin Example (balcony type)
2 //Final version date: November 7, 2012; 11:51PM
3 //Kyrylo Lobach
4 package vacationOne;
5
6 import processing.core.*;
7 import igeo.*;
8 import khelpers.*;
9 import controlP5.*;
10
11 @SuppressWarnings("serial")
12 public class _1210Version_VacOneTest extends PApplet {
13
14
15     //CONTROLLER TEXT VARIABLES  for credits
16     private Controller textCredits, textCredits2;
17     //MASSING VARIABLES  box massing
18     private Controller boxPtsON, boxLinesON, boxW, boxL, boxH, textMassing;
19     private IVec outlineCorner = new IVec(0, 3, 30);
20     private BoundBoxCorner boxMassing;
21     //WALL SUBDIVISIONS
22     //controller variable
23     private Controller textBeginSubdiv, switchBeginSubdiv,
24     uNumbers, vNumber, panelPercent, seedNum;
25     //variable to be assigned massing box point array
26     private IVec[] boxMassPtsArr;
27     //Wall surfaces;
28     private ISurface leftSrf, frontSrf, rightSrf;
29     //WALL PANELS
30     //left side panels
31     private IVec[][] leftPointGrid;
32     private IVec[][] leftCells;
33     //front panels
34     private IVec[][] frontPointGrid;
35     private IVec[][] frontCells;
36     //right side panels
37     private IVec[][] rightPointGrid;
38     private IVec[][] rightCells;
39     //FLOOR AND CEILING
40     private IVec[] floorSlab = new IVec[4];
41     private IVec[] ceilingSlab = new IVec[4];
42     //END OF VARIABLES
43
44
```



```

45     public void setup() {
46         size(1000, 800, IG.GL);
47         IG.background(150, 110, 80, 60);
48         IG.pers();
49         IG.fillWire();
50         controlVOne();
51     } // end of setup method
52
53
54     @SuppressWarnings("deprecation")
55     private void controlVOne() {
56         //Object of control p5 class
57         ControlP5 p5 = new ControlP5(this);
58         //CONTROL WINDOW
59         ControlWindow cw = p5.addControlWindow("V ONE GENERATOR", 1050, 35,
60         300, 800);
61         cw.hideCoordinates();
62         /*
63         //MASSING CONTROLLERS
64         //text label
65         textMassing = p5.addTextlabel("MASSING", " //MASSING", 10, 10);
66         textMassing.setWindow(cw);
67         //massing box points on/off toggle
68         boxPtsON = p5.addToggle("PTS ON/OFF", false, 10, 30, 20, 20);
69         boxPtsON.setWindow(cw);
70         //massing box line outline on and off toggle
71         boxLinesON = p5.addToggle("LINES ON/OFF", false, 100, 30, 20, 20);
72         boxLinesON.setWindow(cw);
73         //massing box dimensions
74         boxW = p5.addSlider("WIDTH (m)", 10, 35, 20, 10, 70, 200, 20);
75         boxW.setWindow(cw);
76         boxL = p5.addSlider("LENGTH (m)", 5, 20, 12, 10, 100, 200, 20);
77         boxL.setWindow(cw);
78         boxH = p5.addSlider("HEIGHT (m)", 8, 15, 10, 10, 130, 200, 20);
79         boxH.setWindow(cw);
80         /*
81         //FORM GENERATION CONTROLLERS
82         //subdivisions text
83         textBeginSubdiv = p5.addTextlabel("FORM GEN", " //FORM GENERATOR      ",
10, 180);
84         textBeginSubdiv.setWindow(cw);
85         //generate form on/off toggle
86         switchBeginSubdiv      =      p5.addToggle("GENERATE      FORM      ON/
OFF", false, 10, 200, 20, 20);
87         switchBeginSubdiv.setWindow(cw);

```

```

88 //walls subdivision
89 uNumbers = p5.addSlider("CELL WIGHT",2,20,8,10,240,200,20);
90 uNumbers.setWindow(cw);
91 vNumber = p5.addSlider("CELL HEIGHT",1,20,8,10,270,200,20);
92 vNumber.setWindow(cw);
93 //panel type percentage
94 panelPercent = p5.addSlider("SOLID PANELS
%",5,100,50,10,300,200,20);
95 panelPercent.setWindow(cw);
96 //panel random seed
97 seedNum = p5.addSlider("SEED NUM",0,500,50,10,330,200,20);
98 seedNum.setWindow(cw);
99 //*
100 //CREDITS
101 textCredits = p5.addTextlabel("CREDITS", "//CREATED BY KYRYLO
LOBACH ", 10, 770);
102 textCredits.setWindow(cw);
103 textCredits2 = p5.addTextlabel("CREDITS2", "//OCTOBER 2012
", 10, 780);
104 textCredits2.setWindow(cw);
105 }// end of control vacation one setup method
106
107
108 public void draw() {
109 Refreshlgeo.refresh();
110 existing();
111 boxMassing();
112 if((int)switchBeginSubdiv.value() != 0){
113 createSrfWalls();
114 subdivideSrfWalls();
115 linearToCellArr();
116 createPanels();
117 floorCeiling();
118 }//end of if; generate form toggle
119 }// end of continuous draw method (main method)
120
121
122 private void floorCeiling(){
123 //create vector array for floor
124 for(int i=0;i<floorSlab.length;i++){
125 floorSlab[i] = boxMassPtsArr[i];
126 //new IPoint(floorSlab[i]);
127 }//end of i loop for floor array
128 //create vector array for slab
129 for(int i=0;i<ceilingSlab.length;i++){

```

```

130     ceilingSlab[i] = boxMassPtsArr[i+4];
131     //new IPoint(ceilingSlab[i]);
132 } //end of i loop for ceiling array
133 //Create surface for floors
134 new ISurface(floorSlab).clr(0);
135 //Create surface for ceiling
136 new ISurface(ceilingSlab).clr(0);
137 } //end of floor and ceiling method
138
139
140 private void createPanels(){
141     //test cells colors
142     IRandom.init((int)seedNum.value());
143     for(int i=0;i<frontCells.length;i++){
144         if(IRandom.percent(panelPercent.value()))
145             new ISurface(frontCells[i][0],frontCells[i][1],frontCells[i]
146 [2],frontCells[i][3]).clr(i*1,30,150);
147         else if(IRandom.percent(100))
148             new ICurve(frontCells[i],true);
149     } //end of loop
150     for(int i=0;i<leftCells.length;i++){
151         if(IRandom.percent(panelPercent.value()))
152             new ISurface(leftCells[i][0],leftCells[i][1],leftCells[i]
153 [2],leftCells[i][3]).clr(1,30,150);
154         else if(IRandom.percent(100))
155             new ICurve(leftCells[i],true);
156     } //end of loop
157     for(int i=0;i<rightCells.length;i++){
158         if(IRandom.percent(panelPercent.value()))
159             new ISurface(rightCells[i][0],rightCells[i][1],rightCells[i]
160 [2],rightCells[i][3]).clr(220,30,150);
161         else if(IRandom.percent(100))
162             new ICurve(rightCells[i],true);
163     } //end of loop
164 } //end of create panels method
165
166 private void linearToCellArr(){
167     //Initialize k helper class to create cell array from linear arrays
168     frontCells = LinearToCellArr.getArray(frontPointGrid);
169     leftCells = LinearToCellArr.getArray(leftPointGrid);
170     rightCells = LinearToCellArr.getArray(rightPointGrid);
171 } //end of converting linear to cell array method
172
173 private void subdivideSrfWalls(){
174     //create curves at the top of left, front and right surfaces for later mea-
175 surement

```

```

173     ICurve leftCrv = new ICurve(boxMassPtsArr[4],boxMassPtsArr[7]);
174     leftCrv.hide();
175     ICurve frontCrv = new ICurve(boxMassPtsArr[7],boxMassPtsA
rr[6]);
176     frontCrv.hide();
177     //get length of the top sides of the surfaces
178     double lenLeftCrv = leftCrv.len()*10;
179     double lenFrontCrv = frontCrv.len()*10;
180     //U & V subdivisions, U is based on length of the one side of the
surface
181     int unumLeft = (int)lenLeftCrv/(int)uNumbers.value();
182     int unumFront = (int)lenFrontCrv/(int)uNumbers.value();
183     int vnum = (int)vNumber.value();
184     //U & V divided by 1, since points on surface are only between 0
and 1
185     double uincLeft = 1.0/unumLeft;
186     double uincFront = 1.0/unumFront;
187     double vinc = 1.0/vnum;
188     //Initialize k helper class for surface UV subdivision
189     leftPointGrid = UVSrfPointGrid.getPointGrid(leftSrf, unumLeft,
vnum, uincLeft, vinc);
190     frontPointGrid = UVSrfPointGrid.getPointGrid(frontSrf,
unumFront, vnum, uincFront, vinc);
191     rightPointGrid = UVSrfPointGrid.getPointGrid(rightSrf,
unumLeft, vnum, uincLeft, vinc);
192     }//end of subdivide surface walls
193
194
195     private void createSrfWalls(){
196     //get point array from the bounding box
197     boxMassPtsArr = boxMassing.getPtArray();
198     //create left, front and right surfaces based on bounding box ar-
ray
199     leftSrf = new
ISurface(boxMassPtsArr[4],boxMassPtsArr[7],boxMassPtsArr[3],boxMassPt
sArr[0]);
200     leftSrf.hide();
201     frontSrf = new
ISurface(boxMassPtsArr[7],boxMassPtsArr[6],boxMassPtsArr[2],boxMassPt
sArr[3]);
202     frontSrf.hide();
203     rightSrf = new
ISurface(boxMassPtsArr[5],boxMassPtsArr[6],boxMassPtsArr[2],boxMassPt
sArr[1]);
204     rightSrf.hide();

```

```

205     }//end of create surface walls method
206
207
208     private void boxMassing() {
209         boxMassing = new BoundBoxCorner(outlineCorner, boxW.value(),
210         boxL.value(), boxH.value(), (int)boxPtsON.value(), (int)boxLine-
211         sON.value());
212         boxMassing.buildBBC();
213     }//end of new massing method, create massing box for further
214     use
215
216     private void existing() {
217         Grid grid = new Grid(10);
218         grid.drawGrid();
219         //box representing existing massing
220         new IBox( 0.2,0,0, 30,40,50).clr(255);
221     }//end of all existing and grid elements method, creates all exist-
222     ing geometry and grids
223 }// end of vacation one class

```


ALGORITHMS

Carpo, M. (2011). The alphabet and the algorithm. Cambridge, MA: MIT Press.

Chabert, J., & Barbin, E. (1999). A history of algorithms: From the pebble to the microchip. Berlin: Springer.

Man, K. F., Tang, K. S., & Kwong, S. . (1999). Genetic algorithms: Concepts and designs. London: Springer.

Meredith, M., & Sasaki, M. (2008). From control to design: Parametric /algorithmic architecture. Barcelona ; New York: Actar-D.

Salingaros, N. A. (2010). Twelve Lectures on Architecture Algorithmic Sustainable Design. Isi Books.

Skiena, S. S. (2008). The algorithm design manual. London: Springer.

Slavin, K. (Writer). (2011, July). How algorithms shape our world [Video]. Retrieved September 20, 2012, from http://www.ted.com/talks/lang/en/kevin_slavin_how_algorithms_shape_our_world.html

Steiner, C. (2012). Automate this: How algorithms came to rule our world. New York: Portfolio/Penguin.

Terzidis, K. (2006). Algorithmic architecture. Oxford: Architectural Press.

Terzidis, K. (2009). Algorithms for visual design using the Processing language. Indianapolis, IN: Wiley Pub.

Watanabe, M. S. (2002). Induction design: A method for evolutionary design. Basel: Birkhäuser.

Watanabe, M. S. (2011). Natural Logic (B. E. Brownell, Ed.). In Matter in the floating world: Conversations with leading Japanese architects and designers (pp. 217–227). New York: Princeton Architectural Press.

BIOLOGICAL PARADIGM

Ball, P. (2011). Shapes. Oxford [u.a.: Oxford Univ. Press.

Bonnemaison, S., & Beesley, P. (2008). On growth and form: Organic architecture and beyond. Halifax: TUNS Press.

DeLanda, M. (2011). *Philosophy and simulation: The emergence of synthetic reason*. London: Continuum.

Finsterwalder, R. (2011). *Form follows nature: Eine Geschichte der Natur als Modell für Formfindung in Ingenieurbau, Architektur und Kunst = A history of nature as model for design in engineering, architecture and art*. Wien: Springer.

Frazer, J. (1995). *An evolutionary architecture*. London: Architectural Association.

Hensel, M., Menges, A., & Weinstock, M. (2004). *Emergence: Morphogenetic design strategies*. Chichester: Wiley-Academy.

Hensel, M., Menges, A., & Weinstock, M. (2006). *Techniques and technologies in morphogenetic design*. London: Wiley-Academy.

Hensel, M., Menges, A., & Weinstock, M. (2010). *Emergent technologies and design: Towards a biological paradigm for architecture*. Oxon: Routledge.

Johnson, S. (2001). *Emergence: The connected lives of ants, brains, cities, and software*. New York: Scribner.

Oosterhuis, K. (2003). *Hyperbodies: Toward an e-motive architecture*. Basel: Birkhäuser.

Ruse, M., & Travis, J. (2009). *Evolution: The first four billion years*. Cambridge, MA: Belknap Press of Harvard University Press.

Thompson, D. W., & Whyte, L. L. (1942). *On growth and form*. Cambridge [Eng.: The University Press.

Weinstock, M. (2010). *The architecture of emergence: The evolution of form in nature and civilisation*. Chichester, U.K.: Wiley.

COMPUTERS, PROGRAMMING AND PROGRAM USE

Aish, R. (2011). *DesignScript: Origins, Explanation, Illustration*. In *Computational Design Modeling 2011* (pp. 1–8). Berlin: Springer.

Aranda, B., & Lasch, C. (2006). *Tooling*. New York: Princeton Architectural Press.

Burry, M. (2011). *Scripting cultures: Architectural design and programming*.

Chichester, UK: Wiley.

Coates, P. (2010). *Programming architecture*. London: Routledge.

Copi, I. M., & Cohen, C. (2002). *Introduction to logic*. New Jersey: Upper Saddle River.

Deitel, P. J., & Deitel, H. M. (2010). *Java: How to program*. Upper Saddle River, NJ: Pearson Prentice Hall.

Dyson, G. (2012). *Turing's cathedral: The origins of the digital universe*. New York: Pantheon Books.

Graham-Cumming, J. (Writer). (2012, July). The greatest machine that never was[Video]. Retrieved September, 2012, from http://www.ted.com/talks/lang/en/john_graham_cumming_the_greatest_machine_that_never_was.html

Greenberg, I. (2007). *Processing: Creative coding and computational art*. Berkeley, CA: Friends of Ed, an Apress.

Ifrah, G. (2001). *The universal history of computing: From the abacus to the quantum computer*. New York: John Wiley.

Ince, D. (2011). *The computer: A very short introduction*. Oxford: Oxford University Press.

Kang, M. (2011). *Sublime dreams of living machines: The automaton in the European imagination*. Cambridge, MA: Harvard University Press.

Native Client. (2012, July). *Edge: The Future of Interactive Entertainment*, 12–15.

Pearson, M. (2010). *Generative art*. Greenwich, CT: Manning.

Reas, C., & Fry, B. (2010). *Getting started with Processing*. Beijing: O'Reilly.

Shiffman, D. (2012). *The Nature of Code*.

Smith, D. E., & Karpinski, L. C. (1911). *The Hindu-Arabic numerals*. Boston: Ginn and.

Swade, D., & Babbage, C. (2001). *The difference engine: Charles Babbage and the quest to build the first computer*. New York: Viking.

Tedeschi, A. (2011). Parametric architecture with Grasshopper®: Primer. Brienza, Italy: Le Penseur.

Turing, S. (2012). Alan M. Turing. Cambridge: Cambridge University Press.

Watson, P. (2005). Ideas: A history of thought and invention, from fire to Freud. New York: HarperCollins.

Woodbury, R. (2010). Elements of parametric design. London: Routledge.

Wurster, C. (2001). Computers: An illustrated history. Köln: Taschen.

DESIGN COMPUTING

Aish, R. (2011). Designing at t+n. In Experimental Green Strategies (6th ed., Vol. 81, Architectural Design, pp. 22–29). London: Wiley-Academy.

Andrasek, A. (n.d.). Biothing. Biothing. Retrieved December 14, 2012, from <http://www.biothing.org/>

Berg, M. D. (2008). Computational geometry: Algorithms and applications. Berlin: Springer.

Burry, J., & Burry, M. (2010). The new mathematics of architecture. London: Thames & Hudson.

Carpo, M. (2012). The Digital Turn in Architecture 1992–2010 (AD Reader). Wiley.

Colletti, M. (2010). Exuberance: New virtuosity in contemporary architecture. Hoboken, NJ: Wiley.

F., Kubo, M., & Ferré, A. (Eds.). (2003). Phylogenesis: Fao's ark foreign office architects. Barcelona: Actar.

Kolatan, F., & Sabin, J. E. (2010). Meander: Variegating architecture. Exton, PA: Bentley Institute Press.

Krauel, J., Noden, J., & George, W. (2010). Contemporary digital architecture: Design & techniques. Barcelona: Links.

Lally, S., & Young, J. (2007). Softspace: From a representation of form to a simulation of space. London: Routledge.

Leach, N. (2006). Digital Morphogenesis. In *Architectural Design* (1st ed., Vol. 79, pp. 33–37). London: Wiley-Academy.

Liaropoulos-Legendre, G. (2011). *Mathematics of space*. London: Wiley.

Lynn, G. (1993). *Architectural Curvilinearity: The Folded, the Pliant, and the Supple*. In *Folding in architecture*. Chichester, West Sussex: Wiley-Academy.

Menges, A., & Ahlquist, S. (2011). *Computational design thinking*. Chichester, UK: John Wiley & Sons.

Menges, A. (2012). *Material computation: Higher integration in morphogenetic design*. Hoboken, NJ: Wiley.

Meredith, M., & Sample, H. (2010). R.V. In *ACADIA CATALOG 2010* (pp. 66–71). WI: Printinghouse.

Picon, A. (2010). *Digital culture in architecture: An introduction for the design professions*. Basel: Birkhäuser.

Poletto, M., & Pasquero, C. (2012). *Systemic architecture: Operating manual for the self organizing city*. Abingdon, Oxon [England: Routledge.

Rahim, A., & Jamelle, H. (2007). *Elegance*. Chichester, England: Wiley-Academy.

Rahim, A. (2002). *Contemporary techniques in architecture*. London: Wiley-Academy.

Reas, C., McWilliams, C., & Barendse, J. (2010). *Form+code in design, art, and architecture*. New York: Princeton Architectural Press.

Silver, M. (2006). *Programming cultures: Art and architecture in the age of software*. London: Wiley-Academy.

Yoon, J. M., & Höweler, E. (2009). *Expanded practice: Höweler + Yoon Architecture / My Studio*. New York, NY: Princeton Architectural Press.

DIGITAL MANUFACTURING

Borden, G. P., & Meredith, M. (2012). *Matter: Material processes in architectural production*. New York: Routledge.

Correa, D. (2012, September 2). Google + profile video [ICD 2012 Pavilion

Construction video]. Stuttgart.

Corser, R. (2010). *Fabricating architecture: Selected readings in digital design and manufacturing*. New York: Princeton Architectural Press.

Deamer, P., & Bernstein, P. (2010). *Building (in) the future: Recasting labor in architecture*. New Haven: Yale School of Architecture.

Fox, M., & Kemp, M. (2009). *Interactive architecture*. New York: Princeton Architectural Press.

Gibson, I., Rosen, D. W., & Stucker, B. (2010). *Additive manufacturing technologies: Rapid prototyping to direct digital manufacturing*. New York: Springer.

Gramazio, F., & Kohler, M. (2008). *Digital materiality in architecture*. Baden: Müller.

Iwamoto, L. (2009). *Digital fabrications: Architectural and material techniques*. New York: Princeton Architectural Press.

Lynn, G., & Gage, M. (2010). *Composites, surfaces, and software: High performance architecture*. New Haven, CT: Yale School of Architecture.

McGee, W., & Pigram, D. (2011, March). Super Kuka Tools. Fabrication Robotics Network. Retrieved November 13, 2012, from <http://cargocollective.com/fabroboticsnet/SuperKUKATools>

Oxman, N. (2010). *Material-based Design Computation* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Oxman, R., & Oxman, R. (2010). *The new structuralism: Design, engineering and architectural technologies*. Hoboken, NJ: Wiley.

Reuters. (2011, December 2). Flying robots, the builders of tomorrow. Reuters Video. Retrieved November 24, 2012, from <http://www.youtube.com/watch?v=xvN9Ri1GmuY&feature=related>

Sheil, B. (2012). *Manufacturing the bespoke: Making and prototyping architecture*. Chichester, U.K.: John Wiley and Sons.

OTHER & MISC.

Anderson, C. (2012). *Makers: The new industrial revolution*. New York: Crown

Business.

Arthur, W. B. (2011). *The Second Economy*. McKinsey Quarterly, (10).

Arthur, W. B. (2009). *The nature of technology: What it is and how it evolves*. New York: Free Press.

Barnes, B., & Dupré, J. (2008). *Genomes and what to make of them*. Chicago: University of Chicago Press.

BeagleBone Product Details. (2012, November 2). Beagle Board. Retrieved December 14, 2012, from <http://beagleboard.org/bone>

Berke, D., & Harris, S. (Eds.). (1997). *Architecture of the Everyday*. Princeton Architectural Press.

Ingels, B. (2010). *Yes is more: An archicomic on architectural evolution*. Köln: Evergreen.

Kinect for windows. (n.d.). Kinect for Windows. Retrieved December 14, 2012, from <http://www.microsoft.com/en-us/kinectforwindows/>

Leap Motion. (n.d.). Leap Motion. Retrieved December 14, 2012, from <https://leapmotion.com/>

Lee, C. C., & Jacoby, S. (2011). *Typological urbanism: Projective cities*. Chichester: Wiley.

Lynn, G. (1998). *Folds, bodies & blobs: Collected essays*. [Bruxelles]: La Lettre volée.

Mallgrave, H. F., & Goodman, D. (2011). *An Introduction to Architectural Theory: 1968 to the Present*. Malden, MA: Wiley-Blackwell.

Nesbitt, K. (1996). *Theorizing a new agenda for architecture: An anthology of architectural theory, 1965–1995*. New York: Princeton Architectural Press.

Raspberry Pi About. (n.d.). Raspberry Pi. Retrieved December 14, 2012, from <http://www.raspberrypi.org/about>

Smart Dust, Sailor research group at UCSD. (2003, February 17). Smart Dust, Sailor Research Group at UCSD. Retrieved December 14, 2012, from <http://sailorgroup.ucsd.edu/research/smartdust.html>

Sykes, K. (2010). *Constructing a new agenda: Architectural theory 1993–2009*. New York: Princeton Architectural Press.

PATTERNS

Agkathidis, A. (2009). *Modular structures in design and architecture*. Amsterdam: Bis.

Andersen, P., & Salomon, D. L. (2010). *The architecture of patterns*. New York: W.W. Norton &.

Ball, P. (1999). *The self-made tapestry: Pattern formation in nature*. Oxford [England: Oxford University Press.

Garcia, M. (2009). *Patterns of architecture*. London: John Wiley.

Glasner, B., Schmidt, P., & Schöndeling, U. (2008). *Patterns 2: Design, art and architecture*. Basel: Birkhäuser.

Jackson, P. (2011). *Folding techniques for designers: From sheet to form*. London: Laurence King Pub.

Pottmann, H., & Bentley, D. (2007). *Architectural geometry*. Exton, PA: Bentley Institute Press.

Schmidt, P., Tietenberg, A., & Wollheim, R. (2005). *Patterns in design, art and architecture*. Basel [Switzerland: Birkhäuser.

Urbach, H., & Steingraber, C. I. (2009). *J. Mayer H. Ostfildern: Hatje Cantz*.

Weil, C., & Weil, T. (2009). *Geometric ornament in architecture, art & design*. Atglen, PA: Schiffer Pub.

VARIATION, OBJECT AND OBJECTILE

Cache, B., & Speaks, M. (1995). *Earth moves: The furnishing of territories*. Cambridge, MA: MIT Press.

Cache, B. (2011). *Projectiles* (Vol. 6, *Architecture Words*). London: Architectural Association.

Doumptioti, C. (2011). *Responsive and Autonomous Material Interfaces*. In *ACADIA: Integration Through Computation* (pp. 318–325). Stoughton, WI:

The Printing House

The Eclipse of Beauty: Parametric Beauty [Motion picture on YouTube]. (2011). Harvard GSD.

Eisenman, P. (1992). Unfolding Events. *Incorporations*, zone, 422–427.

Fischer, V., Aicher, K., Eisenman, P., Speer, A., & Olin, L. (1992). Frankfurt Rebstockpark: Folding in time. München: Prestel.

Moussavi, F., & Lopez, D. (2009). The function of form. Barcelona: Actar.

Spuybroek, L. (2008). The architecture of continuity: Essays and conversations. Rotterdam: V2 Pub.

Spuybroek, L. (2009). The architecture of variation. London: Thames & Hudson.

Spuybroek, L. (2011). Sympathy of things: Ruskin and the ecology of design. [Rotterdam]: V2 Publishing.

Spuybroek, L. (2011). Textile tectonics. Rotterdam: NAI.