Logic Matter

Digital logic as heuristics for physical self-guided-assembly

by Skylar J.E. Tibbits
B.Arch - Philadelphia University [2007]

Submitted to the Department of Architecture and the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Masters of Science in Architecture Studies
and
Master of Science in Electrical Engineering and Computer Science

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY.
June 2010

Signature of Author - Skylar Tibbits ....
Department of Architecture
Department of Electrical Engineering and Computer Science
May 21 2010

Certified by - Terry Knight ....
Professor of Design and Computation, Department of Architecture
Thesis Advisor

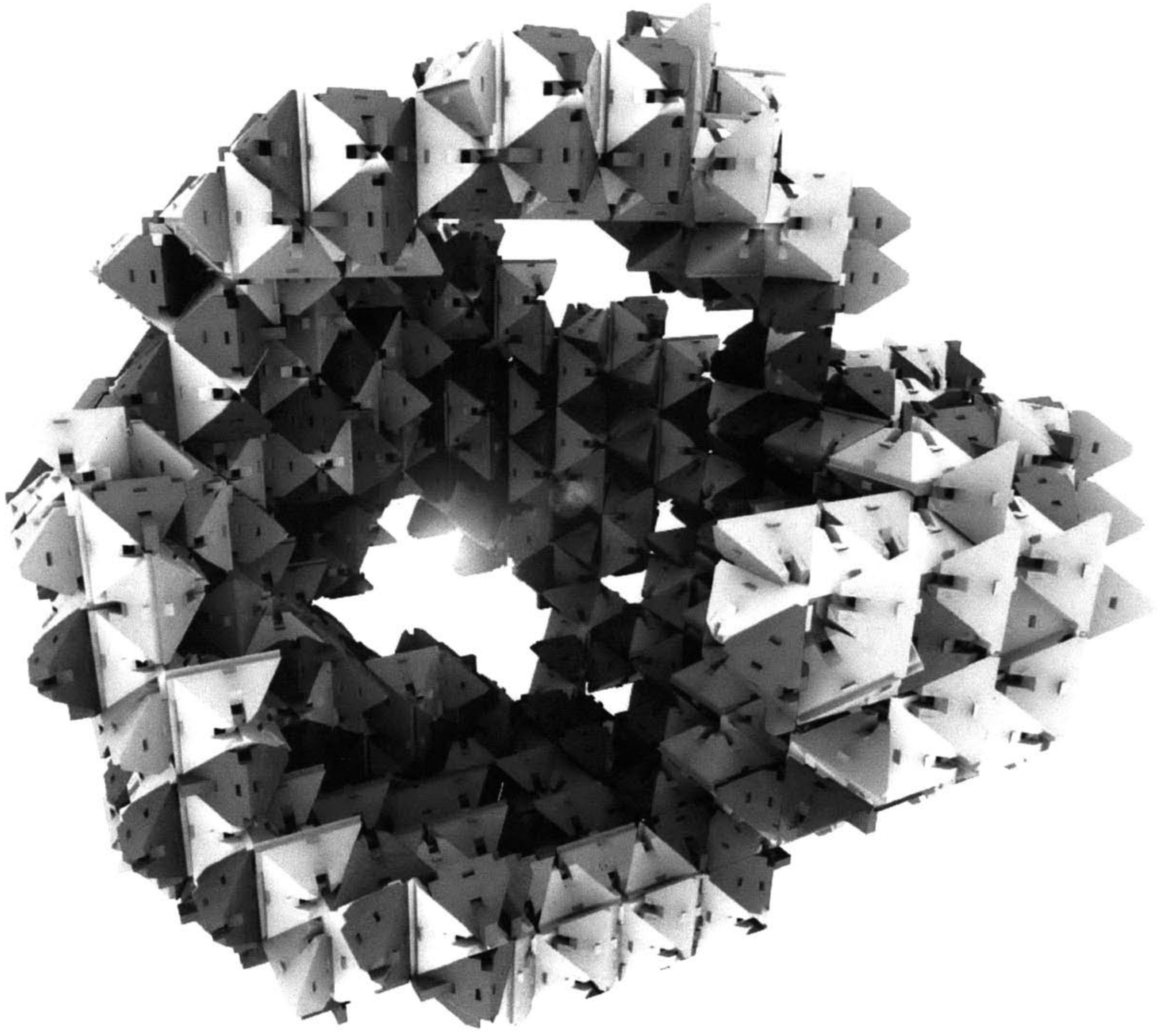
Certified by - Patrick Winston. ....
Ford Professor of Artificial Intelligence and Computer Science, EECS
Thesis Advisor

Accepted by - Julian Beinart.......... ....
Professor of Architecture, Department of Architecture
Chair, Committee on Graduate Students

Accepted by -Terry P. Orlando......... ....
Professor of Electrical Engineering, EECS
Chair, Committee on Graduate Students

Logic Matter

Digital logic as heuristics for physical self-guided-assembly



Thesis Advisor
Terry Knight
Professor of Design and Computation, Department of Architecture, MIT

Thesis Advisor
Patrick Winston
Ford Professor of Artificial Intelligence and Computer Science, EECS, MIT

Thesis Reader
Erik Demaine
Associate Professor, EECS, MIT

Logic Matter

Digital logic as heuristics for physical self-guided-assembly

by Skylar J.E. Tibbits

Submitted to the Department of Architecture and the
Department of Electrical Engineering and Computer Science on May 20, 2010 in partial
fulfillment of the requirements for the degrees of

Masters of Science in Architecture Studies
and
Master of Science in Electrical Engineering and Computer Science

Thesis Advisor - Terry Knight
Professor of Design and Computation, Department of Architecture, MIT

Thesis Advisor - Patrick Winston
Ford Professor of Artificial Intelligence and Computer Science, EECS, MIT

Thesis Reader - Erik Demaine
Associate Professor, EECS, MIT

[] Abstract

Given the increasing complexity of the physical structures surrounding our everyday environment -- buildings, machines, computers and almost every other physical object that humans interact with -- the processes of assembling these complex structures are inevitably caught in a battle of time, complexity and human/machine processing power. If we are to keep up with this exponential growth in construction complexity we need to develop automated assembly logic embedded within our material parts to aid in construction. In this thesis I introduce *Logic Matter* as a system of passive mechanical digital logic modules for self-guided-assembly of large-scale structures. As opposed to current systems in self-reconfigurable robotics, *Logic Matter* introduces scalability, robustness, redundancy and local heuristics to achieve passive assembly. I propose a mechanical module that implements digital NAND logic as an effective tool for encoding local and global assembly sequences. I then show a physical prototype that successfully demonstrates the described mechanics, encoded information and passive self-guided-assembly. Finally, I show exciting potentials of *Logic Matter* as a new system of computing with applications in space/volume filling, surface construction, and 3D circuit assembly.

Table of Contents

Acknowledgements

Thank you Terry, Patrick and Erik for being vastly unique in your thoughts, comments, suggestions and personalities. Thank you for trusting me and being extremely supportive. I am fortunate that I was surrounded by such brilliant and inspiring people. Without your help this thesis would not have been possible.

I would like to thank Neil Gershenfeld and The Center for Bits and Atoms for being extremely supportive during my time at MIT. Thank you for the man/brain power to build crazy robots. Thank you for the resources, endless avenues of inspiration and daunting knowledge. Thank you to the rest of the group for great laughs, unbelievable support, instruction, criticality and plenty of beers and pool. Kenny, Max, Jonathan, Ara, Asa, Peter and Nadia.

Thank you Steffen, German, Varvara and Ari for the laughs, late nights, critical discussions and just about everything that made the two years at MIT possible. Thank you to the rest of the group for the support, criticality and camaraderie: Carnaven, Mark, Adela, Shani, Joseph, Murat. Thank you DesComp PHDs for showing us the way and having a relaxed attitude because you've already been through it!

Thank you MIT, the Media Lab and the Department of Architecture for the amazing facilities, financial support and brilliant people!

Thank you Tom Knight, Jonathan Bachrach and Mark Feldmeier for the extremely inspiring conversations.

I must thank Mr. Coffee, illy, Snapple and most importantly Daft Punk for getting me through two thesis projects and three degrees in seven years.

Finally, I would like to thank my parents: Dina and Jim, my girlfriend, Veronica and my grandmother, Nan, for sticking by me through seven years full of stress, crazy hours and no contact with the outside world (including sometimes you). Thank you for being supportive, understanding and loving.

-Sky

[01] Introduction

Introduction

This thesis explores the nature of assembly, specifically in the context of complex structures, i.e. assemblies with extremely large numbers of parts, assemblies with extremely small parts in large numbers or any variety of possibilities in between. The problem that arises from geometric complexities and difficulties in assembly techniques include material tolerances, error propagation, difficult construction sequences and increasing complexity of the information required to build complicated structures. Many of these assembly problems relate to the complexity of information processing and information transfer from material-to-material and from assembler-(human or machine)-to-material. A number of techniques have been developed to fight the associated problems with assembly including; precise Computed Numerically Controlled (CNC) Machines, robotic arms, large-scale 3D printing and many others, however these machines can be seen to only avoid many of the issues of assembly rather than offering resolutions. In this thesis I will argue that if we want to build more complex structures than humanly possible today, then we need to embed discrete assembly information directly into our materials to self-guide the successful assembly of complex structures. Initially, I will explain the notion of digital information, as compared to analog or continuous information, relating to the problems of assembly and systems of fabrication today. Then I will outline a number of case studies that attempt to infuse digital logic as a system for information transfer and assembly. I will introduce a system called, *Logic Matter*, that directly embeds digital information into material parts to provide self-guided assembly of complex structures. *Logic Matter* offers a new paradigm to resolve many of the associated problems with assembly systems today. I will demonstrate that we can actually describe useful geometries (lines, surfaces and volumes) through single sequences of binary inputs and geometric transformations. Further, I will demonstrate that *Logic Matter* provides powerful computing possibilities for complex assemblies. Finally, a working prototype will be shown that emphasizes the benefits of *Logic Matter* as a system of material parts for self-guided assembly, utilizing digital information for computing through construction.

Digital Materials

It is important to identify the term *digital material* and differentiate it from an analog material and the systems of fabrication/assembly that surround us today. The term digital fabrication is common terminology in most schools and facilities around the world, referring to cutting edge technology that utilizes Computed Numerically Controlled (CNC) machines to add, subtract or manipulate/fabricate materials. As Neil Gershenfeld at MIT's Center for Bits and Atoms has argued, this cutting edge technology is inherently analog, not digital![01] Many digital fabrication machines are as analog as the first analog computers that represented information as continuous physical properties. These machines; CNC routers, waterjet machines and laser cutters, are as imperfect as the parts they utilize and actually become increasingly worse as the scale and rate of production increases. This is a fundamental property of analog systems, as seen in primitive analog telephone communication and many other noisy analog systems.[02] Likewise, today's CNC fabrication machines rely on continuous (external) information for tool paths and allow errors to propagate as the system scales and parts increase in size.[03]

Alternatively a digital system -- as introduced by Shannon, after encountering severe limitations while working on the Differential Analyser -- is a system that transfers discrete information (0 and 1), can produce reliable systems from unreliable components and utilizes redundancy to prohibit errors from accumulating.[04] This type of system actually increases its rate of perfection as the scale increases! Shannon demonstrated this by introducing the idea of a threshold in relation to the amount of noise errors in a system, explaining that "below a certain amount of noise, the error rate is effectively zero."[05]

If we now apply this idea of a digital system back to our materials and fabrication machines we can imagine a system that does not rely on external intelligence, does not rely on continuous

01 Neil Gershenfeld. Fab: The Coming Revolution on Your Desktop--from Personal Computers to Personal Fabrication.
02 Neil Gershenfeld. Fab.
03 Neil Gershenfeld. Fab.
04 Neil Gershenfeld. Fab.
05 Neil Gershenfeld. Fab.

information and does not allow errors to propagate with an increase in scale. Based on digital logic and Von Neumann's work on self-replicating systems, Gershenfeld explains how a digital system could actually carry its own assembly instructions, saying "this medium is quite literally its message, internally carrying instructions on its own assembly. Such programmable materials are remote from modern manufacturing practice, but they are all around us." He goes on to describe the ribosome and its sequence of self-programmed folding of proteins as an example of a self-assembling digital process within our human bodies.[06,07]

> There's a pattern here. Shannon showed that digital coding can allow an imperfect communications system to send a message perfectly. Von Neumann and colleagues showed that digital coding can allow imperfect circuits to calculate perfect answers. And the ribosome demonstrates that digital coding allows imperfect molecules to build perfect proteins. This is how the living things around you, including you, form from atoms on up. It's necessary to precisely place 10^25 or so atoms to make a person, an ongoing miracle that is renewed in every on every day. The role of error correction in fabrication is as close as anything I know to the secret of life.[08]
>
> The discovery of building with logic is actually a few billion years old; it's fundamental to the emergence of life. Current research is now seeking to do the same with functional materials, creating a fundamentally digital fabrication process based on programming the assembly of microscopic building blocks.[09]

We can now see how digital materials could be utilized and how they differ from analog counterparts. However, it is probably necessary to further describe why digital materials would be inherently useful as building blocks for our physical world. To do this we can imagine an example of using bricks to build a wall. If the wall is sufficiently small we should be able to successfully build a straight wall. However, if the wall increases size dramatically, such that we have 10^23 parts, or the desired output of the wall is incredibly complex, we won't be able to simply rely on the information-less and error prone material of a brick. We could then utilize a new type of brick, one that contained information within its material. Information that would instruct us where to place the brick, how it relates to the previous bricks, check for errors in our placement and guide us in the right direction for achieving our extreme complexity. This type of material would inherently require some form of digital logic embedded within its material

06 John Von Neuman. Theory of Self-Reproducing Automata. 1966.
07 Neil Gershenfeld. Fab.
08 Neil Gershenfeld. Fab.
09 Neil Gershenfeld. Fab.

parts to benefit from the previously outlined advantages of digital systems (discrete information transfer (0 and 1) , increasing perfection of the system as it scales and minimizing the required information/accuracy to achieve immense complexity), thus offering self-guided-assembly for large, complex, physical structures.

George Popescu and Neil Gershenfeld have defined a number of key aspects for what makes a digital material. The essential properties are listed below:

* the set of all the components used in a digital material is finite (i.e. discrete parts).
* the set of the all joints the components of a digital material can form is finite
* the assembly process has complete control over the placement of each component[10]

Gershenfeld goes further to explain that we can easily send 10^23 bits of information, then asks, can we successfully build structures with 10^23 number of parts?[11]

This thesis will attempt to infuse the outlined beneficial characteristics of digital information and materials/fabrication for self-guided-assembly of complex structures. In the next section I will outline the current problems associated with our construction/assembly systems, then look at a number of relevant case studies that have attempted to infuse digital information for assembly.

"Bits to Atoms and Atoms to Bits" - Neil Gershenfeld

10 George A. Popescu. Digital Materials for Digital Fabrication. (Masters of Science Thesis. MIT, 2007)
11 Neil Gershenfeld. Fab.

[02] Problem

Problem

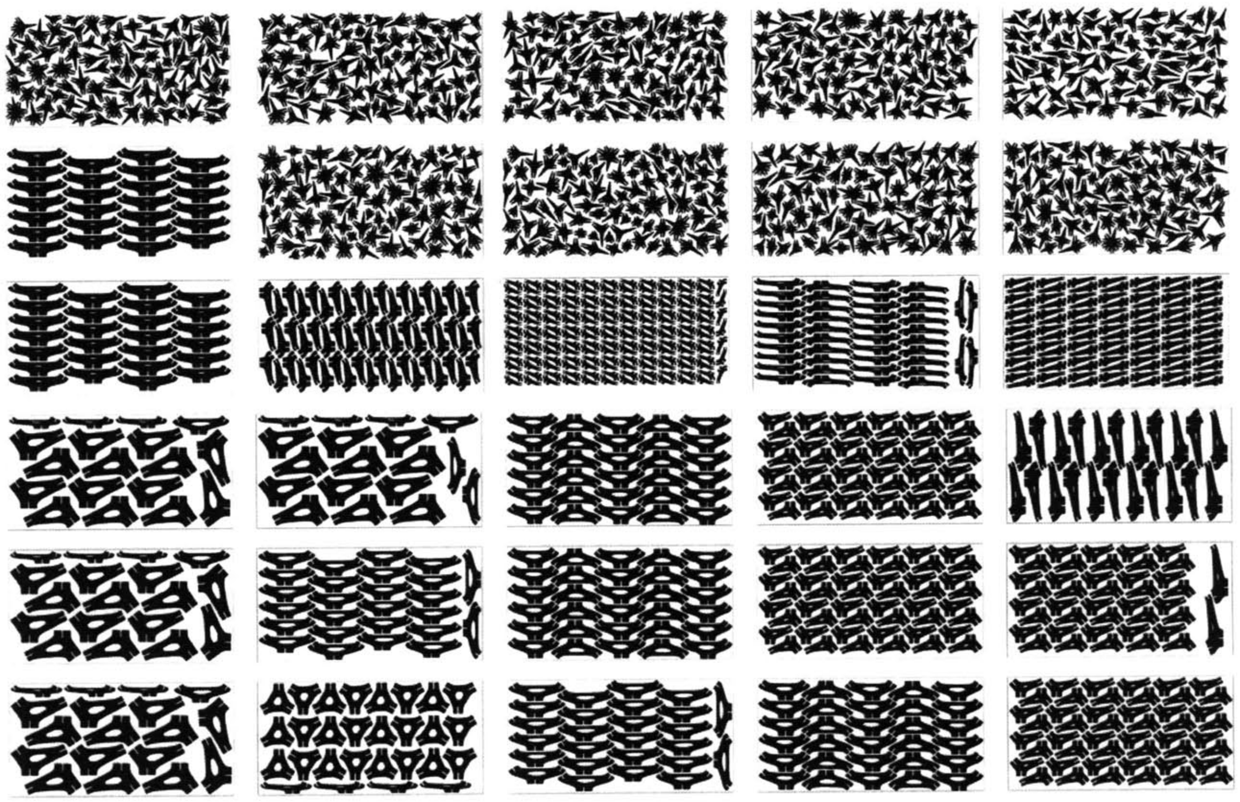


Figure 01. (30) 4' x 8' Polyethylene sheets to be CNC milled and assembled by hand.

Given the increasing complexity of the physical structures surrounding our everyday environment -- buildings, machines, computers and almost every other physical object that humans interact with -- the processes of assembling these complex structures are inevitably caught in a battle of time, complexity and human/machine processing power. If we are to keep up with this exponential growth in construction complexity we need to develop automated assembly logic embedded within our material parts to aid in construction. In order to embed our parts with assembly intelligence we must first understand processes of assembly, the types of intelligence that are necessary and the mechanisms required to build such systems.

Expanding our notion of digital materials, as discussed previously, we can see that there is an opportunity to embed information into our material parts in order to accomplish useful and complex overall configurations from simpler unintelligent and inaccurate parts. This is the fundamental backbone of this thesis, through which I will demonstrate a response to increasing difficulties in assembling increasingly complex structures. I will argue it is not that we should reduce the complexity of our built structures or increase the complexity of our material parts. Rather, it is the opposite, we should reduce the complexity of our parts to only include essential information; information that will guide, direct and be discretely transferred with redundancy and resultant accuracy for successful self-guided assembly.

The cutting edge technology surrounding the construction industry increasingly strives to make larger and more complex machinery to build smaller-than-the-machine parts with imperfect strategies. They fight precision issues with motor/sensor feedback, material tolerances and inevitable machinery fatigue. Before even taking the parts off the assembly line, the machines are fighting accumulating error from imperfect tools, measurements and continuous information.[12] The number of unique elements increases everyday as we read yet another argument for mass-customized building components.(See Figure 02) Likewise, assembly teams fight man or machine hours to physically assemble these complex systems with accruing construction tolerances, seemingly approaching intractable problems. All of these systems are fighting an uphill battle, one that we can compare to the analog telecommunications industry's attempt to fight error by improving long-distance telephone lines, trying to reduce noise and "ever-more clever ways to send their signals."[13] As previously emphasized, we can learn a tremendous amount from the paradigm shift of digital information, therefore, we must embed digital information into our material parts to similarly take advantage of discrete information transfer, resolving error propagation and simplifying assembly sequences.

12 Neil Gershenfeld. Fab.
13 Neil Gershenfeld. Fab.

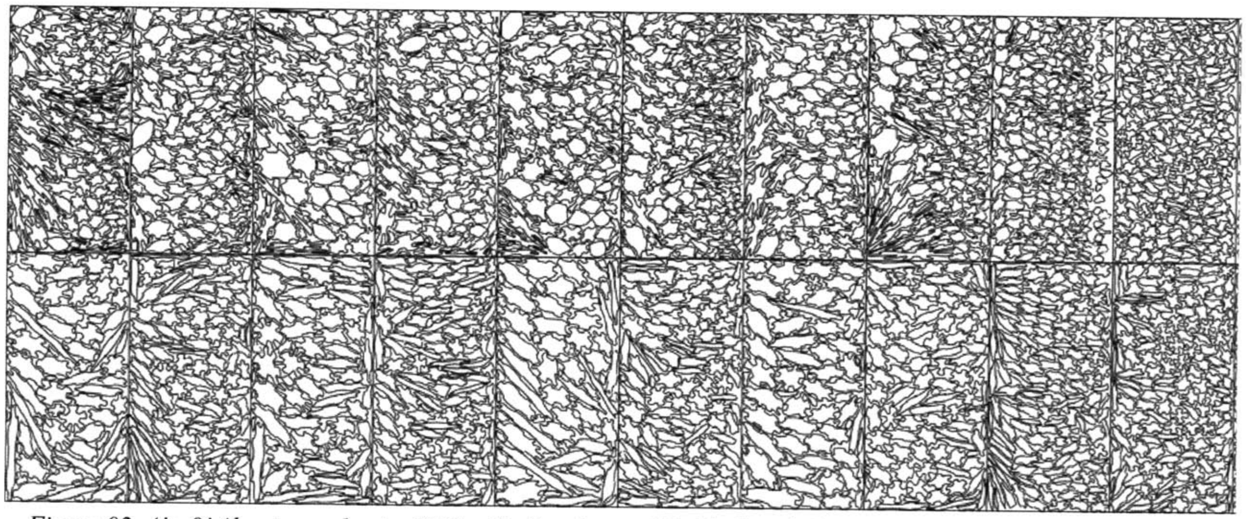


Figure 02. 4' x 8' Aluminum sheets, CNC milled and assembled by hand. 2796 individual parts with 5375 rivets.

The information required to build the complex geometries that flood our built environment embody local, geometric, decision making. Each piece, at the very least, must be numbered and named with the appropriate nomenclature. This number should either aid the assembler in describing the proper 3D orientation/position or should indicate the associated drawing/3D model that will further instruct assembly. Aside from the nomenclature and drawing information the user needs to be able to orient themselves in reference to the drawing and 3D model such that they can accurately understand the local orientation of the material part. For example, if the user has correctly identified the piece and its local adjacent pieces but has incorrectly located them with respect to the global geometry they may incorrectly place the parts, force the parts into place or consequently destroy the overall configuration. At each step the assembler needs to be able to know which piece is next, be able to find that piece in a soup of thousands or hundreds of thousands of other unique pieces and properly orient themselves as well as the specific piece. All of this information is directly stored in the user and at best can be partially stored in a numbering system to be accompanied by a set of drawings or 3D models. Hence, we are losing a battle against complex physical geometries and we must find ways to let the materials speak to us and speak to themselves in guiding their assembly, successfully and effectively.

The systems of assembly that are prevalent today include the previously mentioned mass-customized parts, electromechanical building materials and large machines to build small parts. Mass-customized parts can be beneficial because they can allow the parts to be assembled in only one specific way. However, this technique leads to inevitable failure when you are required to spend time finding each piece in a soup of parts. Steps are being taken to better encode the information for finding parts in systems such as Radio Frequency Identification (RFID) and barcode technologies.[14] However, these fall short of the information required on site to position each piece in the appropriate place. The second assembly system steps in to fill this gap with machines that assemble parts or deposit material. For example, Contour Crafting, robotic arms or any number of future robotic assembly systems demonstrates the vision to build complex machines that will save humans from excessive labor and mental anguish with our complex structures.[15] Immediate limitations like the scale of the machine versus the size of the part, cost, torque and errors/tolerance directly question the feasibility for extremely large-scale projects.

There is a plethora of contemporary research on self-assembling robotics (that will be covered in the next section) from 1D chain robots to lattice robots, even 3D printing conductive and non-conductive materials that are addressing the idea of robotic material parts to solve complex assembly issues. Reconfigurable robotics has extremely useful advantages; programmability, functionality and geometric versatility, however, many of these systems fail in scalability due to the expensive start-up costs, expert development and repair teams required and insistent failure with each additional device.(See Figure 03) Consequently, robotic building parts seem idealistic and inevitably prone to failure either from cost, technological limitations or inevitable failure with thousands of electromechanical devices. The question then becomes, how can we embed the advantages of reconfigurable robotics and *Programmable Matter* to large-scale physical structures without the reliance on electromechanical devices? This thesis will attempt to answer this question by embedding digital information into the geometry of passive mechanical material parts.

This thesis will further outline ways to implement digital information into material parts through

14 Schneider Mike. Radio Frequency Identification (RFID) Technology and its Applications in the Commercial Construction Industry. (University of Kentucky. April 24, 2003).

15 Contour Crafting. http://www.contourcrafting.org/.

a specific mechanical module and the implementation of digital NAND logic for assembly information when building complex systems part-by-part. I will emphasize the benefits of building with digital materials and the potential for a new paradigm of computing through construction - letting our material parts compute the shapes we want to build. This would allows us to eliminate the infrastructure of large-machines, electromechanical parts or even the off-site computing power currently required to generate and build complex physical structures. The material parts actually compute, encoding local decision making and assembly sequences through self-guided assembly of *Logic Matter*.

Figure 03. 8 Person team: Designers, CS, EE, MechE & 16yr old prodigy.

[03] Background & Context

Introduction

The following section elaborates on the definition of digital materials by looking at four examples of related work that emphasize contextual solutions to similar problems of physical assembly, fighting assembly time and error while utilizing the benefits of discrete information transfer.

Mechanical Self-Assembly

(C. Babbage, A. Turing, L.S. Penrose, J. Von Neumann) - 1950's

Von Neumann initiated a quest to duplicate the amazing ability of natural systems -- DNA/RNA -- to self-replicate and self-assemble. Developed after the introduction of digital information, by Shannon and Von Neumann in the 40's, self-replication was examined through a theoretical perspective with complicated automata models.[16] Penrose followed suit by simplifying the elements for self-assembly and articulating the essential principles:

1. Each unit must have at least 2 states
2. The activated structure must have defined boundaries (beginning & end) - Preventing them from attaching to the wrong units - non-aperiodic
3. Kinetic energy must be captured and transferred to potential energy
4. Each activated unit must be capable of communicating its state to another unit with which it is in close contact
5. Must ensure eventual unit contact & latching.

+ Release Mechanisms[17]

16 John Von Neuman. Theory of Self-Reproducing Automata.
17 L. S Penrose. Self Reproducing Machines. (Scientific American vol.200: pp 105-114, June 1959).

Penrose's principles lead to the development of his physical implementation of a mechanical latching system for self-assembly.[18] The mechanical latches had two initial units that could be set to an arbitrary string. A series of additional latches would be loaded into a track and finally agitated to provide the energy to actuate the latches. As the latches came into contact with one another, a duplicate of the initial string would be generated.[19] Penrose and Von Neumann demonstrated that the essential qualities for natural reproduction and cellular self-assembly were possible in mechanical systems with real world applications.[20]

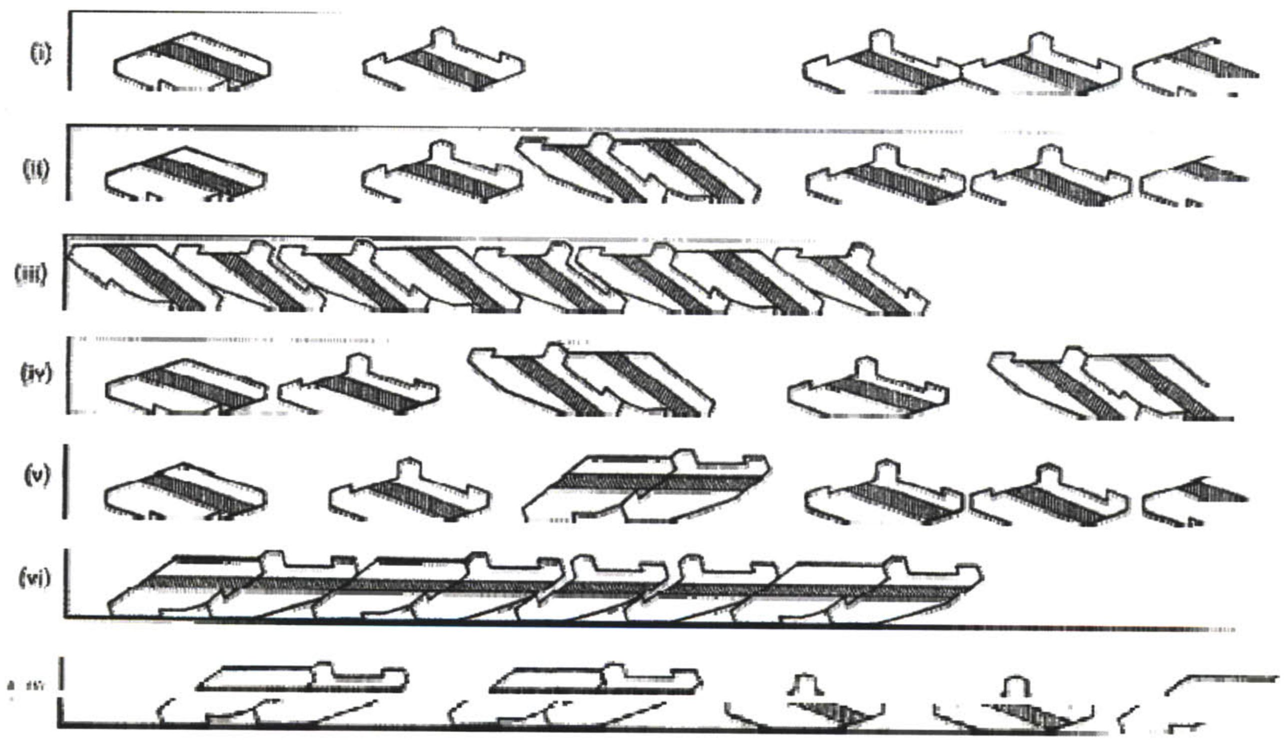


Figure 04. Mechanical Self-Assembly - L.S. Penrose

18 Penrose, L. S. Self Reproducing Machines.
19 Penrose, L. S. Self Reproducing Machines.
20 Saul Thomas Griffith. Growing Machines. (PHD diss. MIT, 2004).

Billiard Logic

(E. Fredkin, T. Toffoli) - 1982

Von Neumann and Penrose's work on self-replication introduced the potential for physical/ mechanical elements to transfer information or arbitrary strings. Fredkin and Toffoli in 1982 introduced the possibility of actually computing digital information transfers through physical materials. In their work, billiard balls were demonstrated to have the capability to function as digital logic gates. The hypothetical billiard balls took two inputs(or physical transformations representing input) computed and realized a final physical transformation in correlation to the output. This demonstrated that we can actually discretize the information transfer between a seemingly continuous environment.[21] Further, physical environments could now actually function as computing devices, opening a world of possibilities for digital logic. This thesis was heavily inspired by the notion that we could infuse physical materials with digital logic and attempts to go further by implementing the benefits of discrete information for physical assembly.

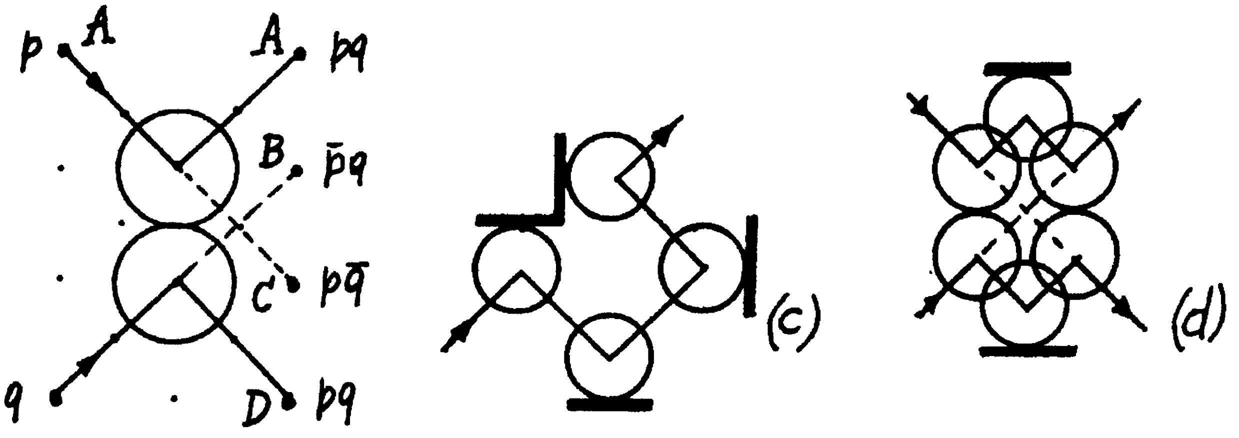


Figure 05. Billiard Logic - Interaction gate - (c) delay (d) nontrivial crossover

21 E. Fredkin and Toffoli, T. 2002. Conservative logic. In Collision-Based Computing, (A. Adamatzky, Ed. Springer-Verlag, London, 47-81).

Programmable Matter & Self-Reconfigurable Robotics

(N. Gershenfeld, J. Jacobson, S. Griffith, H. Lipson, D.Raus, K. Stoy etc.) - Current

Stemming from the idea of mechanical self-assembly, a number of influential people and projects have developed attempting to program materials/robots to reconfigure, self-assemble and replicate. Neil Gershenfeld explains, the goal, "is to re-implement the functionality of molecular biology in engineered materials, in order to enlarge the material set, expand operational scales, ease design, and improve reusability and reconfigurability. Viewed from the bottom up, we want to build programs out of, rather then into components."[22] This explanation emphasizes the goals of a field of research called, *Programmable Matter*. Properly defined, the intention of the Defense Advanced Research Projects Agency's (DARPA) *Programmable Matter* program, "is to demonstrate a new functional form of matter, based on mesoscale particles, which can reversibly assemble into complex 3D objects upon external command."[23] Put more simply, *Programmable Matter* is currently realized through a variety of techniques; reconfigurable robotics, 3D printing technologies, pick and place assembly machines any many others. The goals of self-assembling modules are most notably achieved through reconfigurable robotics, or programming robotic modules that move, connect/disconnect and function in a variety of ways.(See Figure 06)

The field of reconfigurable robotics is flooded with techniques for achieving actuation, flexibility, high torque strengths to carry neighboring modules and number of other technical necessities. The modules are usually packed with electronics and motors that allow them to hopefully function for the lifespan of the live demo. These systems, although impressive, exciting and approaching functionality, offer little hope in terms of scalability to large applications or complex structures. Robotics is plagued with high costs, excessive failures in electronics or mechanical devices and communication issues, making it less than stellar for wide-spread applications. While the community is steadily overcoming these issues, it is important to look back at the fundamental principles of self-assembly and self-replication as introduced by Von Neumann

22 Millibiology Project. http://milli.cba.mit.edu/.
23 Programmable Matter. http://www.darpa.mil/dso/thrusts/physci/newphys/program_matter/index.htm.

and Penrose as well as the simplicity of computing through physical interaction seen in the Billiard Logic example. These examples demonstrate that it is possible to embed the same programmability and functionality as seen in cutting edge reconfigurable robotics projects today, without the reliance on heavy electromechanical devices. In this thesis I describe a system called *Logic Matter*, as a programmable system that embodies many of the self-assembly/replication possibilities infused in our own biological systems (DNA/RNA) while similarly, not becoming reliant on the technologies of today, thus affording scalability and robustness.

"[I]n self-reconfigurable robots, even small problems may take a long time because solving them involves the physical movement of modules and not just the flipping of bits."[24]

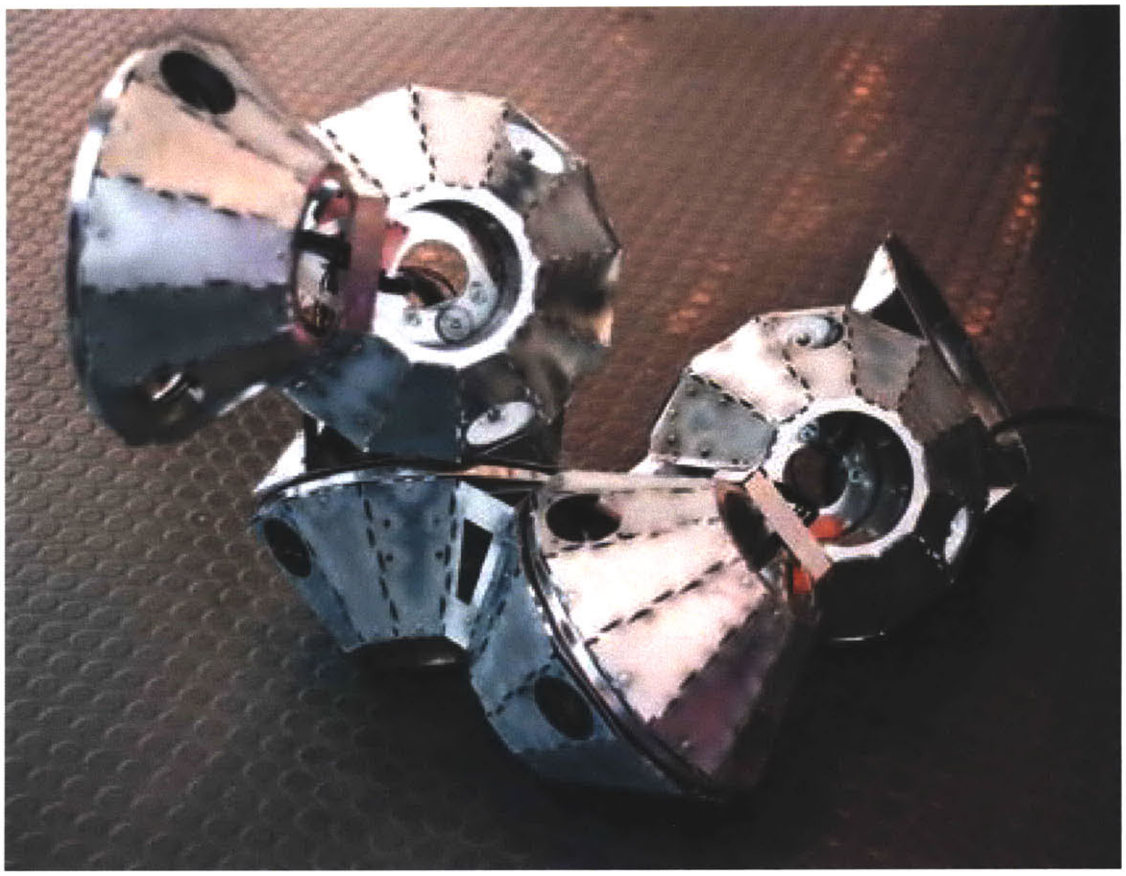


Figure 06. MacroBot - 1D Robotic folding chain system with electromechanical actuation.

24 Kasper Story, David Brandt and David J. Christensen. Self-Reconfigurable Robots.)Cambridge, Massachusetts : The MIT Press, 2010).

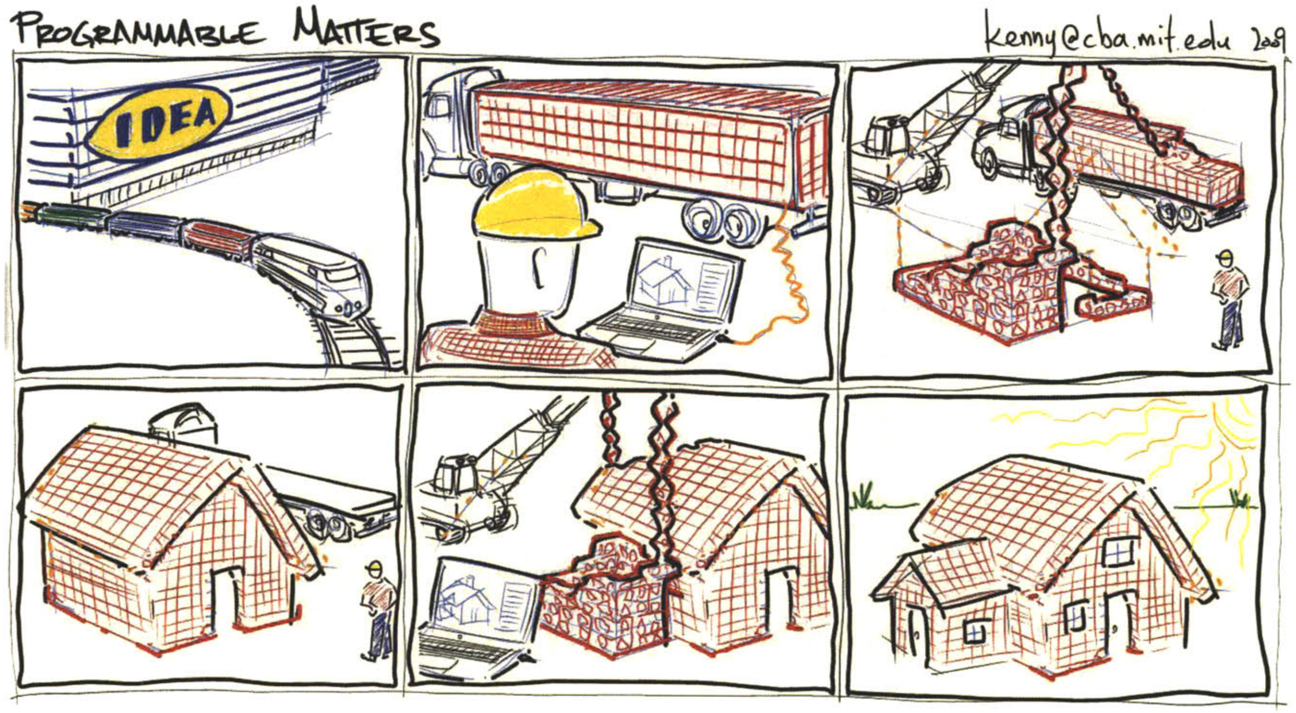


Figure 07. Programmable Matter Cartoon - Kenny Cheung - Center for Bits and Atoms, MIT 2009

Spatial Computing & RALA-

(N. Gershenfeld, J. Bachrach, E. Demaine, D. Darlymple) - 2007

The idea of programmability in the physical world lead to a new model of computing called Spatial Computing and more specifically the implementation of a programming language called Reconfigurable Asynchronous Logic Automata (RALA). Darlymple explains, "RALA can be seen as a generalization of traditional integrated circuits (the current most realistic model of computing) to where circuits can locally reprogram themselves (making them universal) and gates synchronize locally (removing the need for a global clock). We argue that a model with all of these properties is necessary and sufficient for computing to scale optimally according to the laws of physics."[25] RALA and other Spatial Computing models combine the worlds of physical and computer science, relating to the examples of physical logic described previously. Specifically, RALA is built upon a grid of logic operations (AND, OR, XOR, NAND, Copy, Delete) that pass, asynchronously, information in multiple directions.[26] This allows for a

25 David Dalrymple, Erik Demaine, Neil Gershenfeld. Reconfigurable Asynchronous Logic Automata. (MIT, 2009).
26 David Dalrymple, Erik Demaine, Neil Gershenfeld. Reconfigurable Asynchronous Logic Automata.

distributed model of computation where the distance between modules actually equals the time of computation, differing greatly from contemporary computing architectures.[27] This thesis emerged directly from the idea of RALA, imagining that we could embody the same type of distributed communication and information transfer in a, real world, physical system. This distributed physical logic system, *Logic Matter*, would then be powerful for passing computed information between modules and thus, an exciting opportunity for physical self-assembly.

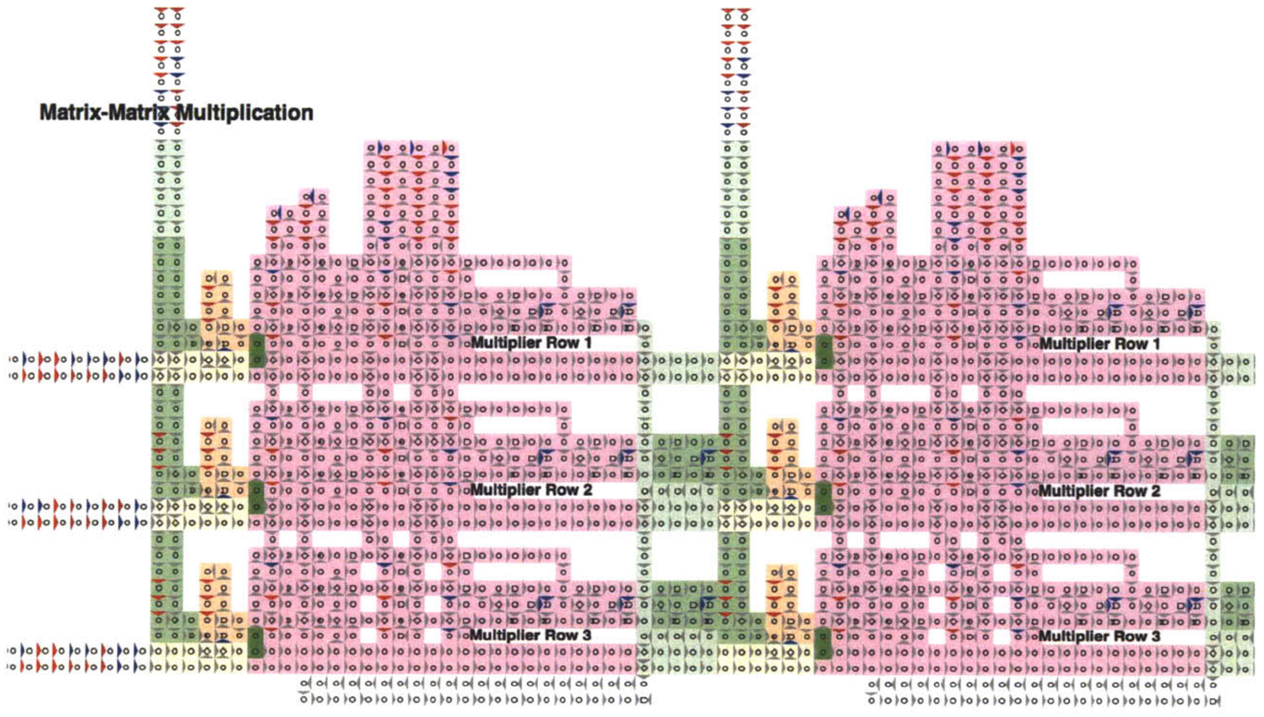


Figure 08. RALA program - Matrix-Matrix Multiplication with Logic Gates as distributed computing

27 David Dalrymple, Erik Demaine, Neil Gershenfeld. Reconfigurable Asynchronous Logic Automata.

[04] Method

Introduction

In the previous chapters I discussed a context of work that inspires and questions whether we can embed self-assembly logic directly into our material parts without the reliance on electromechanical devices and processing power. In this chapter I will demonstrate that it is indeed possible and will identify a number of key elements that enable design possibilities, local design making and mechanical computing. I will also address the main drawback to previous self-assembly systems -- node failure -- and how we can utilize a system called *Hairy Chains* as a resolution. Finally, I will question where the intelligence lies in this system and how we can maximize the material intelligence by introducing digital information within our material parts.

Imagine doing a jigsaw puzzle with a large number of pieces. You look at the picture on the box and try to find pieces that match the colors in a particular area to give you clues as to the pieces' orientation. Standard jigsaw puzzles have all unique pieces that should each fit correctly in one position. However, now imagine that we remove the image from the puzzle (or similarly, throw away the box). This eliminates all heuristics for searching/deciding which piece to place next. You are now left with only a brute force, trial-and-error, method, thus the puzzle has essentially become intractable. But, what if the pieces told you the next moves to make? When you place a piece, it would check the previous pieces and then instruct you which piece to place next. This wouldn't be a very fun puzzle; however it would be an extremely efficient way to successfully assemble a complex puzzle. If only our pieces could tell us the next steps!...

Logic for Assembly

With the goal of designing a system that can assemble itself, or instruct another to assemble itself, the obvious initial questions are; how to store information, how to translate information, what type of information is needed and how to make local and global decisions. We need a system that can be informed and inform neighbors, one that can translate input to output as a physical transformation and one that has a limited number of states. Boolean logic perfectly embodies these characteristics, specifically digital logic gates, with the unique function of passing information between gates while combining gates to achieve greater global functionality. Boolean logic gates take two elements of input, or two voltages (0V & +5V), and performs one of seven potential logic operations. The possible Boolean logic operations include: NOT, AND, NAND, OR, NOR, XOR and XNOR.[28] Each of these operations takes two streams of input (voltages or bits) and provide a single output corresponding to the logic operational result of the two inputs. For example if we are using an AND gate then it requires that both inputs are [1] in order to return a result of [1]. (This is commonly expressed as saying, "input 1 AND input 2 need to be [1] in order to return a [1]").

The question then is, *why is logic useful for assembly*? We can answer this by looking at the characteristics of Boolean logic's input and output information and how this can correspond to assembly. Logic gates are unique because they provide output not solely based on input, rather it is based on the combination of the two inputs as well as the description, or decision, of its own type. This means that the type of gate essentially dictates how it should react given any one of four input possibilities. Each gate will react differently. For assembly this means that the placement of a brick in a wall would not be placed at random or based directly on the last two placements of a brick, rather it would be placed based on the combination of previous placements as well as an internal/local deciding factor. This decision is physically stored when the brick is placed, it then acts to potentially inform the next bricks. This in effect means that

28 Paul Scherz. Practical Electronics for Inventors. (New York: McGrawHill, 2000).

the materials we use to build with are actually informing the next assembly sequences. No interpretation or error from the user, only pieces informing pieces. Four input possibilities and only 2 output possibilities, or four brick configurations to dictate 2 possible placements. Thus, logic provides the local decision making for our system of self-assembly.

Utilizing the correct type of logic gate at the correct moment will enable us to turn right, left, up or down based on any number of previous configurations. If this is applied to a physical system (without a processor) one can see how utilizing a geometric/physical form of a logic gate would enable the system to instruct the next steps for assembly. It is then essential that we can change the type of gate throughout the system, or possibly utilize the effects of any of the gates while still only needing one module. This can be done with *combinational logic*, where gates are combined with themselves or others to create different gates and higher functionality.[29] For example, if you take the output of a NAND gate, split the result, and use it as two inputs for another NAND gate you can create an AND gate.[30] In this way you can begin to see how a single gate can be utilized to have any number of local decision making possibilities based on combinations of previous inputs.(See Figure 09)

The NAND mechanism is the perfect choice for a gate that can utilize combinational logic because it is considered a "universal" gate, or a gate that has the ability to create any other gate (NOT, AND, OR, NOR, XOR, XNOR) by combining a series of NAND gates in different configurations..[31](See Figure 11) The universality also means that through repeated inputs we can get all possible outputs (geometric transformation) and build upon computing mechanisms in sequence. The next question is how to utilize the functionality of the NAND mechanism within an individual component, without the reliance of electronic components or motors.

29 Paul Scherz. Practical Electronics for Inventors.
30 Paul Scherz. Practical Electronics for Inventors.
31 Paul Scherz. Practical Electronics for Inventors.

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

Figure 09. NAND Truth Table with A & B inputs.[32]

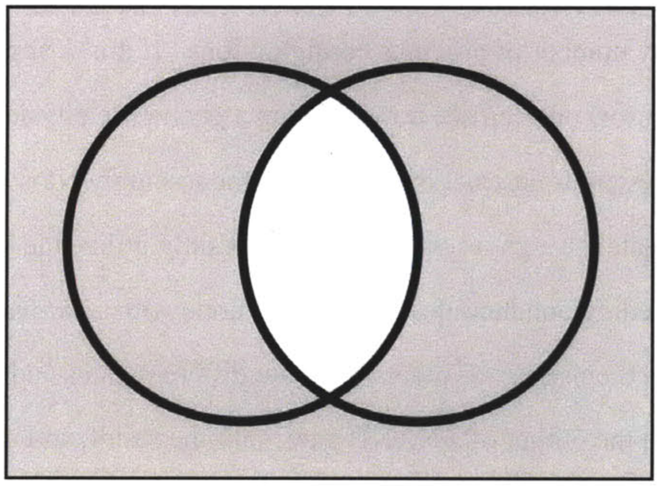


Figure 10. NAND Venn diagram[33]

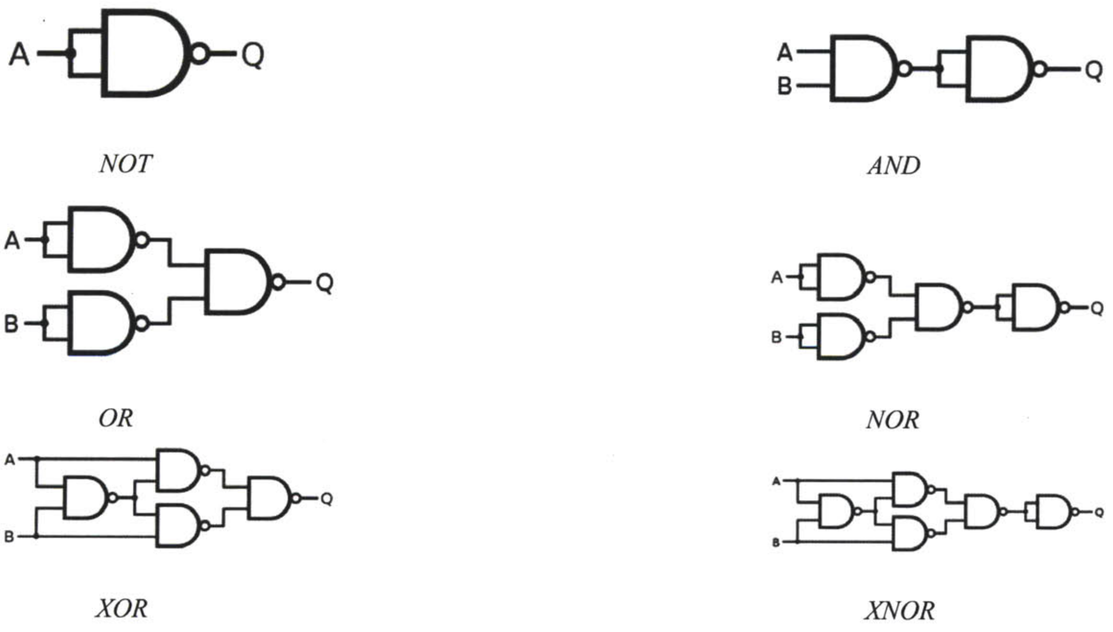


Figure 11. NAND Equivalents of AND, OR, NOR, XOR, XNOR.[34]

32 Paul Scherz. Practical Electronics for Inventors.
33 NAND Venn Diagram. http://en.wikipedia.org/wiki/File:VennFTTT.svg.
34 Paul Scherz. Practical Electronics for Inventors.

As the previous example demonstrated, NAND gates are *universal*, that is they are able to be combined to form any other gate.[35] This is a useful feature in that it allows a user to have only one type of device and still maintain all possible combinations of input/output. Next I need to infuse NAND functionality into a geometric unit that allows all possible input/output configurations and maintain scalability. In order to allow scalability in either direction, large or small, the unit should not rely on devices, electronics or costly additive parts; rather the unit should deploy geometric principles that can scale to any size and maintain the mechanical functionality. This section will explain the geometric principles that will allow the NAND mechanism to be utilized in a scalable entity.

As discussed in Section 03 on Billiard Logic and Bubble Logic, it has already been demonstrated that digital logic can be comprised of a physical geometric entity (or relationships between them) without relying on the traditional transistor/electronics component construction.[36] However, if we want to be able to describe useful geometries and construct a variety of global structures then we need to have a geometry that at least packs space perfectly. The geometry must also allow for the appropriate number of input/output faces that results in the overall *fanout* of the gate, or the amount of influence one gate has on the following gates.[37] Finally the geometry of the unit must embody a geometric transformation in direct relation to the different outputs of the NAND gate. The perfect realization of these principles is the right-angle tetrahedron.(See Figure 12) First, the right-angle tetrahedron has the wonderful quality of packing space perfectly and being able to assemble in a variety of ways to form everything from chains to volumes. The right-angle tetrahedron also has four identical faces with two opposing axis. The four faces are split into two groups of two faces straddling the axis that are rotated 90 degrees about one another. Let two of the faces represent inputs and two of the faces represent outputs. This affords two inputs and two possible outputs, although at any given time only one of the output faces will be occupied, depending on the series of inputs received. The occupation of one of the output faces creates the local geometric transformation (turn left, right, up or down) that is needed to be able to

35 Paul Scherz. Practical Electronics for Inventors.
36 Paul Scherz. Practical Electronics for Inventors.
37 Paul Scherz. Practical Electronics for Inventors.

describe useful geometries. This transformation can be performed completely passively through geometries that either restrict or permit access to the face, and thus only allows the user to place the tetrahedron on the appropriate face, according to the inputs and the NAND functionality. Finally the right-angle tetrahedron geometry allows a single chain of NAND mechanisms to built in sequence, or cascaded thus allowing combinational and sequential logic.[38]

Figure 12 shows the proposed NAND geometry, the right-angle tetrahedron. This geometry has the ability to compose all of the functionality of a digital NAND gate directly within the single unit: input, output and digital gate. Depending on the placement of the unit, the geometry can act in one of the three circumstances, providing information to a neighbor or receiving information, storing the state of the input and deciding the transformation of the next step. In order to fully operate the NAND geometry there needs to be two input faces attached to a Gate unit, thus, three units need to be present to force the transformation of a fourth unit. The specifics of the mechanism will be described further in Section 05, however, it is important to emphasize the elements of the base geometry that provides for the possibility of having input, output and decision making in a single unit.

38 Manu Prakash and Neil Gershenfeld. Microfluidic Bubble Logic. (Science 315 (5813), 832. 9 February 2007).

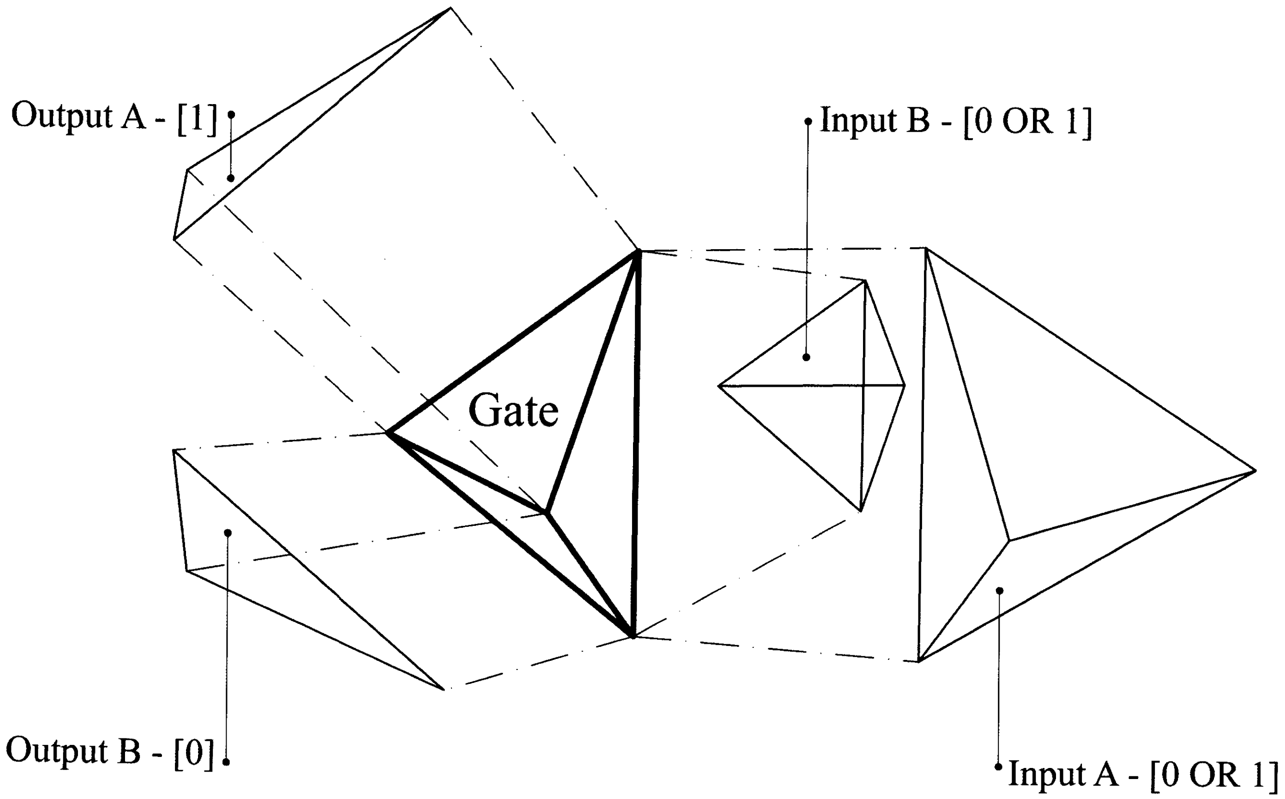


Figure 12. Right-angle tetrahedron demonstrating the functionality of a NAND gate through input, output and gate decisions. Output A and Output B are the two output faces, either [0] or [1]. Input A and Input B are the two input faces. The input faces can both receive either [0] or [1] at any time. The input faces will dictate the decision in the Gate unit as to which face (Output A [1] or Output B [0]) will be utilized, direclty based on the NAND truth table.

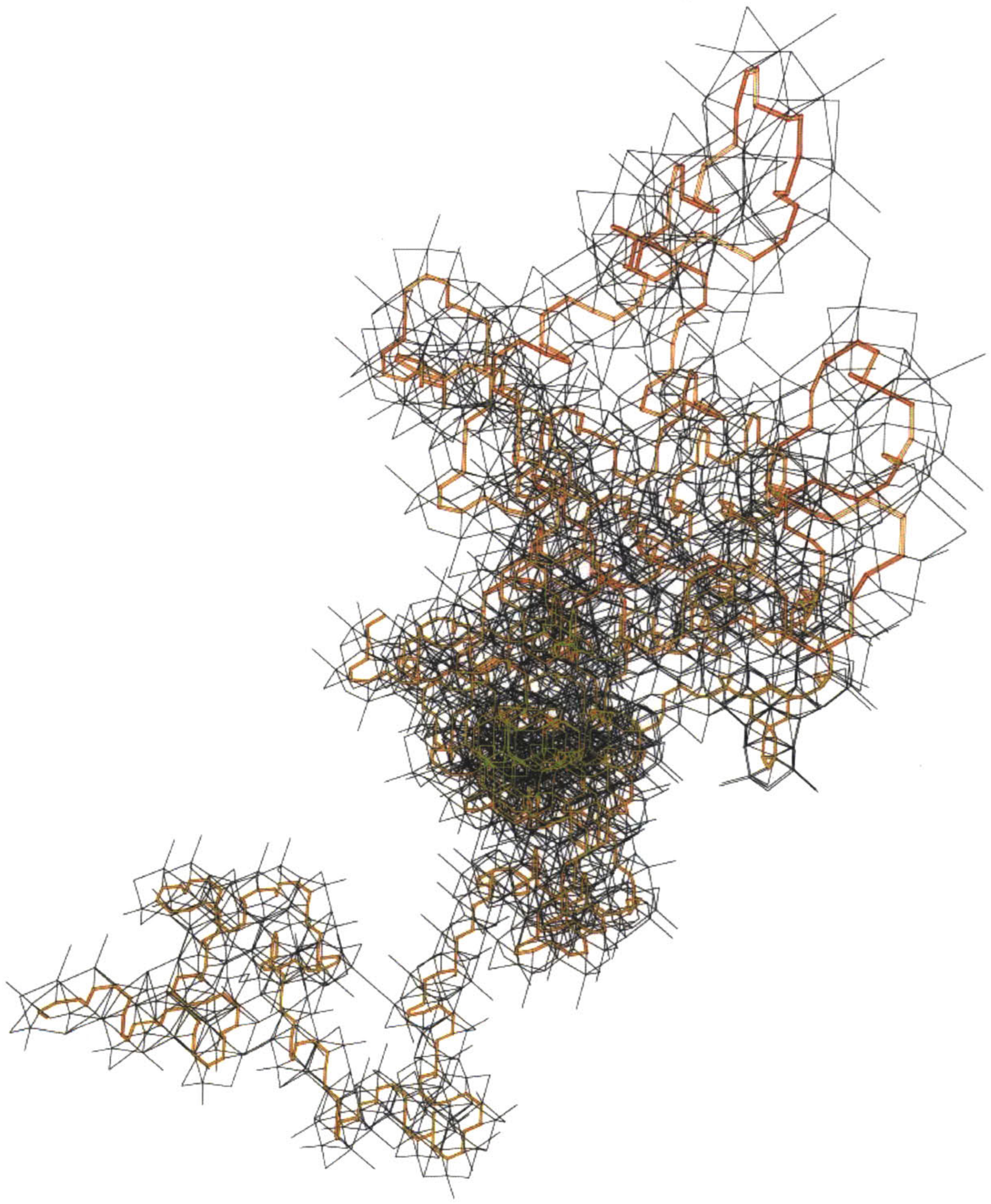


Figure 13. Single path assembly of a random growth showing redundant inputs. Single path shown in orange/green and redundant nodes shown in black.

Single Path & Redundancy

The assembly of NAND geometries follows in the sequence of two inputs, one output with an open face for a second input leading to another output etc. This perpetual cycle results in a chain-like configuration of input and gate units. The input units are completely redundant and serve multiple purposes. If the redundant units were removed we would still be left with a single path of gate units folding in all three dimensions. However, the system would lose a number of important characteristics that are afforded by the redundant input elements. This section will look at the qualities of the single path (or chain) configuration as well as the importance of the redundant tetrahedrons.

In Section 03, I discussed the domains of self-reconfigurable robotics and *Programmable Matter* and looked a number of promising opportunities emerging. A number of common robotic configurations were also discussed (chain, lattice and hybrids), each with positive and negative attributes.[39] It was emphasized that many of these systems rely heavily on electromechanical devices to solve a myriad of tasks, however, especially in chain configurations; they are limited in their capabilities to survive mechanical or electrical failure. Many of the chain robotic systems are limited to the capacity of strength and robustness within each module because if any of the modules fail then the entire system fails.[40] This is commonly found in any chain or rope systems. For example, if any point in a rope fails, then the entire rope is broken. Alternatively, the chain configuration, or single path strategy, has a number of potential benefits due to its simplicity and versatile geometry.

Geometrically the single path technique allows a one dimensional line to fold into any 1D, 2D or 3D configuration simply by folding any node in the chain by a specified angle. This allows the 1D path to describe any number of 3D geometries fairly easily. This also means that the entire description of complex geometry can be compressed into a single string of information (or joint

39 Kasper Stoy, David Brandt and David J. Christensen. Self-Reconfigurable Robots.
40 Kasper Stoy, David Brandt and David J. Christensen. Self-Reconfigurable Robots.

angles). The single path technique also means that the physical construction of the system is far simpler than a lattice type structure where nodes need to connect to multiple neighbors and loads are distributed in multiple directions. The single path provides an exciting platform for quick and easy development of a physical and programmable device as well as simplified geometric descriptions (programmable sequences).

The previously described NAND geometry, the right-angle tetrahedron, follows suit with the single path typology by continually adding to the last open face and growing in a linear motion. The linear growth allows for two possible turns at every step; right, left or up and down depending on the orientation in 3D space. Similarly, this single path of NAND geometries benefits from the qualities of a single path programmable chain robot, however it attempts to go a few steps further by introducing redundancy to fight the problem of single node failure. The input tetrahedrons serve a number of roles through redundancy; structural robustness, work-arounds for failure and branching opportunities.

The main benefit of the redundant input tetrahedrons is robustness for node failure. As the chain example illustrates, any node on a chain that fails can lead to entire system failure. However, if we have redundant tetrahedrons clumping around the outside of the single chain then we can work-around node failures. I have called this system of redundancy the *Hairy Chain* model. Once a unit has failed, either due to mechanical, material or structure failure, the information and structural loads can be rerouted through the redundant units and around the failure. This allows the entire system to become more robust and far more scalable. System redundancy can be seen in a number of biological processes, most notably in DNA sequences with 64 total nucleotide triplets for only twenty amino acids.[41] Likewise, the redundancy of the *Hairy Chain* system allows a single path to be scalable, robust and feasible for a variety of applications, while providing more security that the system will survive several types of system failure.

Similarly, the redundancy of the *Hairy Chain* provides structural robustness through

41 Neil Gershenfeld. Fab.

interconnected redundant tetras. As the single path turns in 3D space, the redundant tetras will start to touch one another (not self-intersect). The interconnection of the tetrahedrons allows the structural load paths to be distributed outside of the single path and actually branch to other parts of the chain. Unlike traditional chain systems, this structural redundancy can actually create networks of structural paths and potential scale to larger loading conditions and aggregation scales. On the same lines as the network of load paths, the redundant units provide a face for potential branching. This goes outside of the traditional chain schemes, however if we are able to branch the single path into a network of paths then it will be possible to describe large geometric configurations faster and more efficient. For example, if we want to build a surface it is far easier and more efficient to construct it with opposing directions (U & V), similar to weaving, than it would be to describing with a single curve (rastering back and forth). The chain typology is simpler in the sequence of moves and description (i.e. left, left, right, up, up, left, left etc) when compared with a branching technique, but may be far less efficient. The redundant tetrahedrons at least provide the opportunity to branch the chain typology into larger aggregations and circuit possibilities.

The proposed *Hairy Chain* technique offers a number of beneficial qualities for system robustness when compared with traditional chain robotic systems. I have outlined the benefits of redundancy for structural robustness, work-arounds for failure and branching opportunities. These techniques are specific to the NAND geometry that was previously described but can be seen as opportunities for pushing many traditional programmable chain systems toward larger applications and real-world implementation. After demonstrating the valuable qualities of digital materials and analyzing a number of background examples I have set forth the groundwork for utilizing the NAND geometry, a right-angle tetrahedron mechanism, through a single-path redundant framework. In the next chapter will explain the specific implementation of this system, called *Logic Matter* and will go more in depth with programmability, geometric descriptions and computing assemblies.

[05] Logic Matter

User Programmability

Logic Matter is a system of physical modules that can be combined with one another to locally compute the location of a next move based on the previous moves. This system of building blocks contains both local and global information. The local information is built into the mechanics and geometry of the module. Each module is a functioning NAND gate and therefore takes in two elements of input (two previous modules plug-into a single module) which then dictates the possible placement of the next module. For example, if two modules are placed in a given orientation that represents [0,0], when you plug-in the 3rd module it would read this [0,0] of the previous modules and dictate the orientation of the next module to be in the [1] position. This is taken from a NAND truth table and thus demonstrates the functioning NAND mechanism.(See Figure 09)

If the local information is the geometry and mechanism of the NAND gate then the global information is the sequence of input values that add up to any large structure of these modules. For example if we are attempting to describe a sphere with these modules we would need to locally know how to place the modules in relation to one another, making sure that each module is in the accurate position, orientation and has been placed without error. We would also need to know the global, step-by-step, inputs to repeat hundreds of times in order to accurately describe the surface of a sphere. The global information can come from a number of sources such as the physical environment or a pre-determined pattern of 0 and 1 inputs etc.. This section will attempt to describe a few of the possible sources for global information.

It is important to question how a user could actually program, or construct, these large, complex, structures. To do this we should look at the information, or construction sequence, that is required to build such structures and ask where that information might be stored. To make the point slightly more clear, lets quickly look at the example of building a brick wall. In traditional brick construction, the global information is solely stored in the human, the brick layer. He must know where to place each brick, how many bricks to place in total, how to check for errors and what local and global configurations are desired. At the opposite end of the spectrum is a fully automated robotic system that expedites this process by taking a desired goal and interpolates each move required to achieve this configuration. However, as I will describe, the robotic system is fundamentally no different than the traditional brick layer in terms of the local and global information required. I should also note that this robotic system remains an analog system and thus does not gain any of the important characteristics of a digital system (discrete information transfer (0 and 1) , increasing perfection of the system as it scales and minimizing the required information & accuracy) as was discussed in Section 01. Let us expand this idea and dive deeper into understanding the importance of local and global information for constructing any complex system.

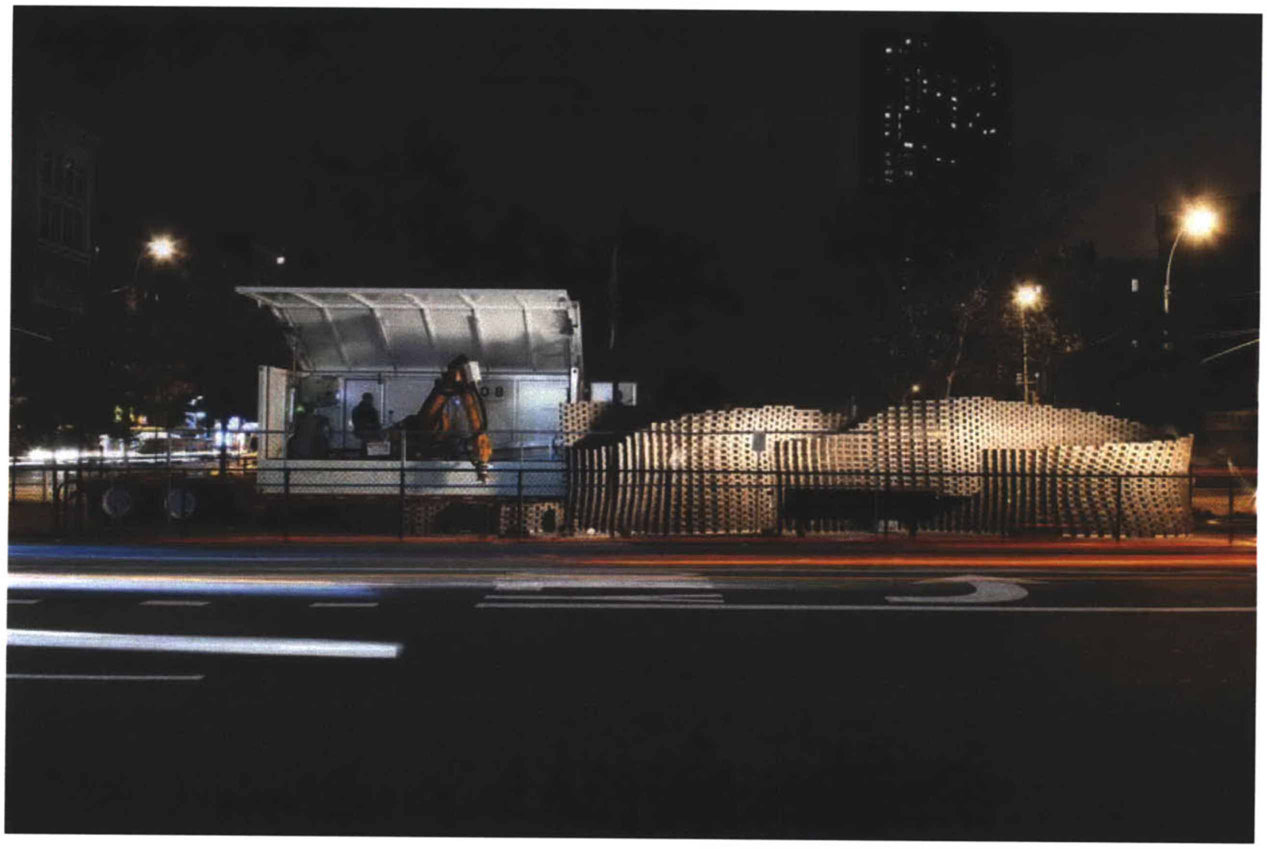


Figure 14. (c) Gramazio & Kohler, ETH Zurich[42]

42 Gramazio and Kohler. http://www.gramaziokohler.com/.

User as Information

As seen in the brick example, a pattern, or algorithm, may be defined that is followed step-by-step to add up to complex global solutions. This pattern may take the form of a sequence of 0's and 1's, or other numerically controlled sequence in the case of a robotic constructor, or simply a set of building instructions for a human. This approach relies solely on the person/ machine constructing the system and puts all information/responsibility in their hands. As Alan Turing explained, a man doing mathematical calculations has only two pieces of information, mathematical facts or truths, and instructions for performing the proper steps in the proper sequence.[43] We can easily see how this method of utilizing local truths and a sequence of steps could lead to global solutions. This directly relates to the two previous examples of the human brick layer and the robotic brick layer. Both of these examples take a pattern of moves and execute them repeatedly, hopefully achieving the goal configuration (as long as errors don't accumulate without the human or robot noticing). The robotic system only speeds up the human process, not adding any of the benefits of a truly digital system. We could slightly improve this system if we utilize a "smarter" robotic constructor that utilizes closed-loop feedback to reduce and adjust to errors. Likewise, the "smarter" robotic system might also compute online and be able to react on-demand to obstacles or other live failures.

This direction quickly seems like an uphill battle, fighting the amount of motors and electronics one can fit on the robotic system, fighting the overall cost of the system and the number of experts needed to build such a system as well as the scale because your constructor seemingly needs to be larger than the structure your are building. All of these factors lead to the conclusion that these systems do not scale well (extremely big or extremely small in terms of the number of units or the size of each units), although they can demonstrate simple procedures that lead to globally complex structures. If we remember that goal is to embed this information into our physical materials as to minimize the information required for the human/constructor, then we should try to further reduce the information required by the user.

43 H. Wang. 1965. Games, logic, and computers. (Scientific American, 98–106, November).

Environment as Information

The second obvious source for the required global information comes from external conditions. This means that some physical factors of the environment are influencing the input at each step. For example if we look at gravity to directly influence the placement of each module we might first place a module and look to see if it is falling based on gravity and its current orientation. If the module is falling place a 1, otherwise place a 0. (i.e. A module placed in space may fall or not fall depending on its 3D orientation as well as the other supporting modules around it). Another well-known example is iRobot's Roomba® robotic vacuum cleaner.(See Figure 15) The goal configuration is to maximize the coverage of any given room therefore cleaning the most possible square footage. It might not seem impossible since the robot doesn't know what the room looks like or what obstacles lie its path. However, we can understand fairly well how the Roomba® works simply by looking at the information required and where it is stored. In this case, the robot may only know to move forward until it hits something, then turn a given angle and move forward again. Alternatively, the robot could be programmed to spiral or follow walls until collides with an object, then change angles and continue. If the robot repeats these context-specific procedures for an extended period of time the amount of square footage covered (cleaned) should approach 100%. This example well highlights the point that information for complex global configuration may be stored in the external environment and can be utilized to inform local moves.

Computing Through Construction : *Logic Matter*

We have now seen how local and global information are required to build any complex system and a number of ways to implement both types of information. I have shown how we can embed the local assembly information into a physical NAND mechanism through a system called *Logic Matter*. From the previous examples we saw ways to implement global information from the environment to simple pattern algorithms, external computing/robotic devices and globally specific units that constrain the possible configurations. Now I should expand our understanding of digital information and demonstrate how to utilize the NAND functionality as means for embedding the global information for desired goal configurations.

The tools to measure each of the previously described systems should be based on (1) the amount and type of information the user/constructor needs in order to build any given configuration, (2) how it restricts or allows error propagation and (3) how the system signifies progress or if the goal state has been reached. As seen in Figure 17, each system has different types of information required. I will now demonstrate the programmability of *Logic Matter* and demonstrate a simple execution that encompasses the benefits of such a system.

	User as Info.	**Environment as Info.**	**Global Specific Unit**	**Compute as Building**
Example	Robot Brick Layer	Roomba®	Curved Bricks, Alum. Installation	Logic Matter
Amount/Type of Info.	Location, Orientation, Global Pattern	Local Depends on Unit No Global	Location, Orientation Which type of Brick?, Global Pattern	No Local - NAND Mech. Global as Computing
Error Propagation?	Errors Propagate, Potential Closed Loop	Depends on Unit, No Inherent Error Mechanism	Error Trapping by Closing Loops, Unit Stops Error	Redundancy Stops Errors Discrete Info. Passing - Stops Errors
Signifies Progress?	None - Must Know Global at All Times	None	When Pieces are Gone Goal is Complete	Redundancy Stops Errors Discrete Info. Passing - Stops Errors

Figure 17. Chart Comparing Types of Physical Programmability & Information Required.

SAVE MOM FROM THE MOP!
iRobot Scooba®
Floor Washing Robot
Unlike mopping, Scooba doesn't just spread dirt and germs around, it removes them—up to 98% of common household bacteria!
Remember, Mother's Day is May 9, 2010
Buy a Scooba, get up to 2 FREE Bottles of Clorox® Cleaning Solution
BUY NOW ▸

Figure 15. iRobot Corporation ©2010. All Rights Reserved.[44]

Globally Specific Unit

The next logical step would be to implant all information into the material such that the material specifies exactly the desired output. This may be done with a very specific type of brick that is designed specifically with a desired goal in mind. Alternatively this could be done with a minimum number of bricks that each respond to different conditions (i.e. corner conditions, straight portions, curvature etc.). Each brick would then need to be utilized in a very specific sequence. This type of system has the possibility of trapping errors from propagating simply by closing loops. For example, if you are constructing a cylinder out of metal strips that overlap slightly, you can guarantee that the cylinder will be constructed accurately as long as you force the last two pieces to touch.(See Figure 16) The cylinder may have local errors within it, but once the last two pieces connect then you know that you have at least contained the errors within the cylinder, i.e. closing the loop. Regardless of the error trapping, this type of system is contrary to our goals because it actually adds more information required at each step. The constructor would now need to know exactly which brick to use, where to place it, how to check for errors and where they are in the overall configuration.

44 iRobot Corporation. http://www.irobot.com/.

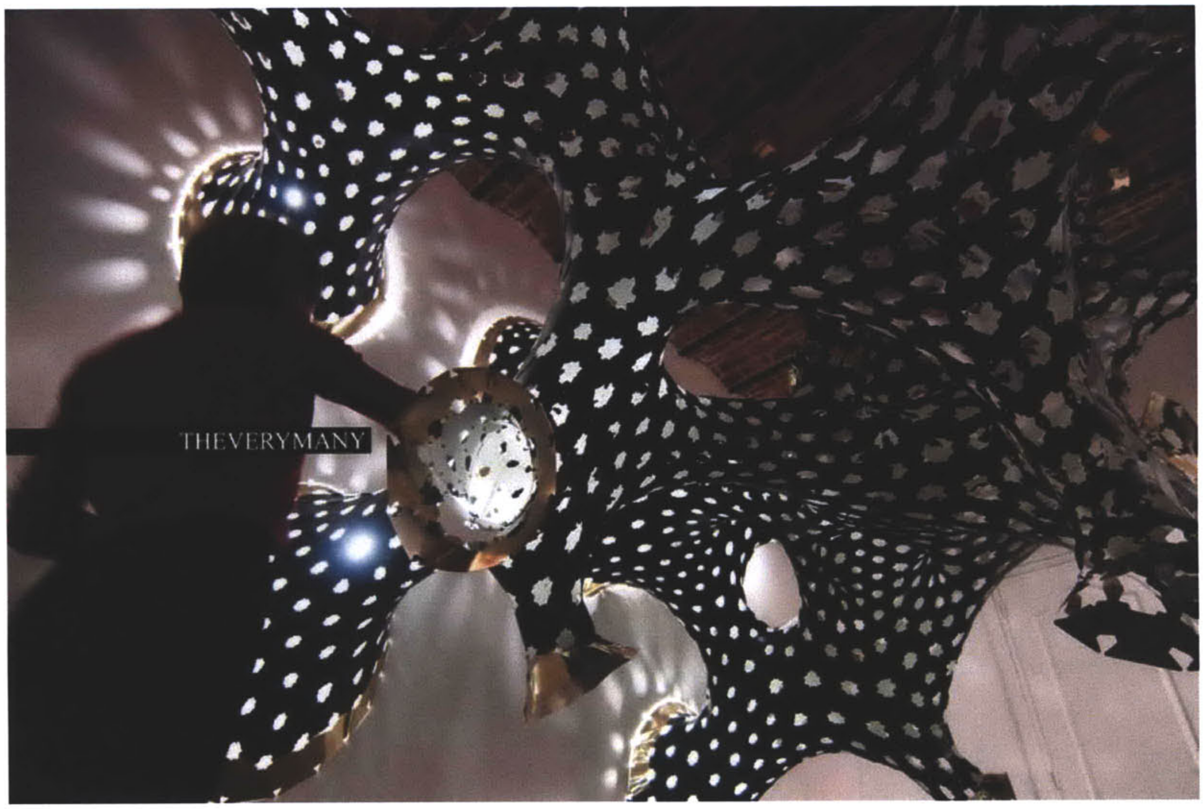


Figure 16. Skylar Tibbits, Marc Fornes - Installation in Paris, FR. 500 Aluminum strips w/ aluminum rivets.

If we look at the first suggestion where we only have one brick but it is designed for a very specific application then we can again remove the decision of which brick to pick-up. For example, lets say we had a curved brick that could go in two orientations; concave and convex. Designed for the construction of one dimensionally curved surfaces or walls, this brick would allow us to decide which orientation to place the brick. This removes one piece of information from the user, which brick to pick up, because we only have one brick. The main drawback to this system is that the brick is extremely specific for the type of overall outcome desired. We can easily see how this could translate to make a sphere or a curved wall, but what if we want to make a straight wall? We would not be able to use the concave and convex brick for every application; rather we would need to design specific bricks for each structure. Thus we can see that we cannot entirely embed the local and global information within our bricks without having some type of computation embedded in our system that defines the next moves and necessary pattern. We saw algorithms in the case of the human/robotic brick layers, but could we actually embed these algorithms directly into our system rather than in an external process?

Previously I described *Logic Matter* as a right-angle tetrahedron with opposing input and output faces that represent 0's and 1's. I also explained that one needs to place two inputs (either [0,0], [0,1], [1,0], or [1,1]) into the tetrahedron, then constraining the possible output as the resultant 0 or 1 based on the NAND truth table. This is all constructed through a physical geometry/ mechanism that represents the system's local information. The local information includes, where and what orientation to place the units as well as the resultant input/output. The unknown variable is the global information. To incorporate the computability of the system for the global information we can look to an example where we utilize a binary increment as the global pattern.

The binary increment could simply start at a given number, lets take 128 which in binary is 10000000. Then step-by-step the user would decrement the input values simply following the binary numbers. If familiar with binary numbers this is as simple as counting ordinary numbers, and if not, this amounts to following a very simple 0 and 1 pattern that gradually shifts places (carry bits), much like an abacus or carrying numbers when adding/subtracting. (See Figure 18) This sequence of decremented binary numbers becomes the global pattern of inputs. At the moment this seems to be very similar to the original example of the robotic brick layer following a simple pattern/algorithm. However, the binary decrement provides a few key opportunities specifically when implemented in *Logic Matter* - NAND geometries. First, the binary decrement system tells us exactly when the goal configuration has been reached. This is signified when the user reaches a point where it can only place all 0's and there is no carry bit. In the case of 128 it would be 00000000. This would only happen after 128 steps. Thus, we do not need to know anything about the global configuration or measure our progress, we are directly signaled when we have reached the goal. Next, this system reduces error propagation by using the binary counting and signifying exactly the next moves rather than remember an arbitrary/ random pattern. As noted in the *Hairy Chain* and NAND geometry descriptions, this system forces redundancy with the input units, ultimately reducing the possibility for errors and allowing the system to work around failure. Further, the NAND mechanism reduces errors through its specific geometry which will be discussed in the follow sections. Finally, this system produces global configurations through the specific input/output transformations of the geometry. The

NAND output dictates either Right, Left, Up or Down moves, depending on the input at each step, thus resulting in a three dimensional configuration based directly on the global binary sequence. This binary counting technique could be expanded to any number of patterns as long as they satisfy these criteria: (1) reduce the amount of information to remember i.e. be repeatable or incremental, (2) should signify the termination/progress of the configuration, (3) should dictate the next input steps without ambiguity, (4) should minimize/eliminate error. (Further discussion will elaborate on potential global patterns that relate to a Turing machine approach, level-by-level updating output based on memory and input combinations.) Through this example I have outlined the programmability of *Logic Matter* in terms of the actual storing of information and direct computation in the units, not externally driven. I have also demonstrated the reduction in user information required to build any arbitrary complex configuration while reducing error propagation and signifying goal termination. This system can be compared to the previously outlined types of information-construction systems, weighing favorably in terms of the user programmability and the functionality of the units. Next we will look at the potential global configurations that can be utilized by such a system.

```
1 1111111111111110000000000000000
1 1111111000000001111111100000000
1 1110000111100001111000011110000
1 1001100110011001100110011001100
1 0101010101010101010101010101010
```

Figure 18. Binary Counting as Input Sequences.

Describing Geometry

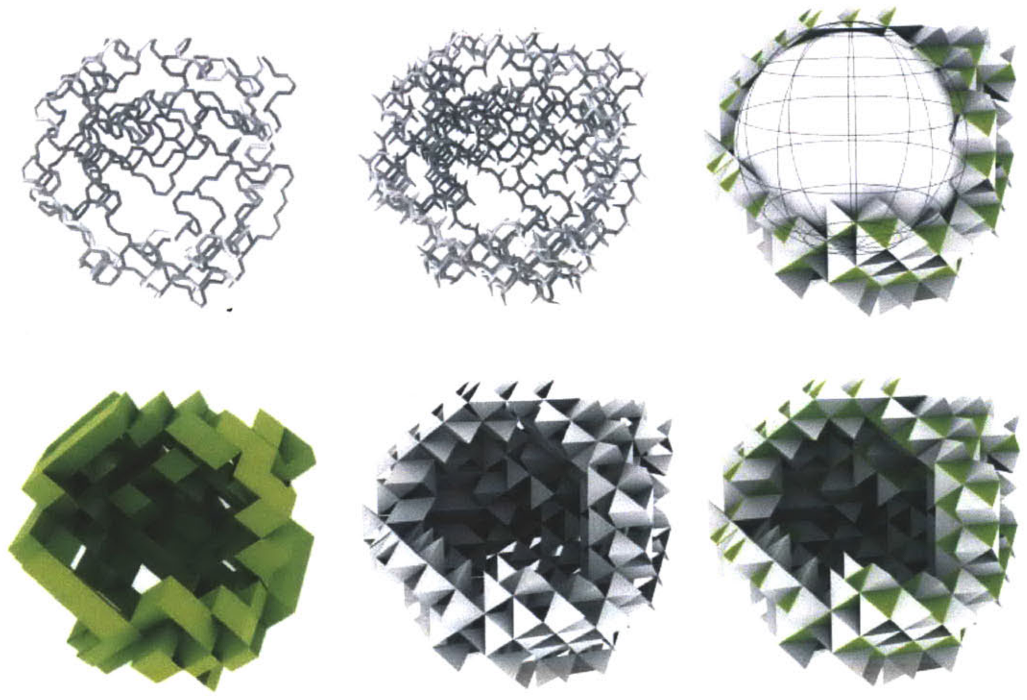


Figure 19. Random walk search on Sphere - Single path, green tetrahedrons are gates, white tetrahedrons are input

After incorporating local and global information, the NAND modules should be able to define useful physical structures. In order to accomplish this goal we need to develop possible approaches for implementation. There are three main types of geometries that can be described with *Logic Matter*; 3D curves, surfaces, and volumes. The first is utilized as a technique to describe the remaining two typologies. The second is aimed at providing a complete description of any given input surface. This may be implemented through a variety of techniques including random walks with greedy heuristics, branching strategies, quad-tree subdivision and many others. The goal is to provide a simple approach to walking along a surface (with a given module), avoiding obstacles, avoiding self-intersection while maintaining complete surface coverage. In this case, the criteria for a successful search strategy would be to maintain a single path of continuity or at least the minimum amount of single paths. The single path strategy relates to a technique within reconfigurable robotics and the physical properties of the tetrahedron modules utilized in this project.

The next implementation focuses on the interior of a closed volume rather than the surface treatment. The goal is to fill any arbitrary closed volume perfectly with a given module. There are also a number of well-known approaches to fill arbitrary volumes; however, the difficulty lies in the interest to maintain the continuity of a single path. How can we fill an arbitrary volume, packing perfectly with only a single path? This path should be directly based on a given physical module or brick. (i.e. a cube would tile different than a tetrahedron) Again, this single path description of the interior of a closed volume would be then translated as the sequence of moves for physically constructing a chain of modules (up,left,up,right,down,up,down,up or 011000100).

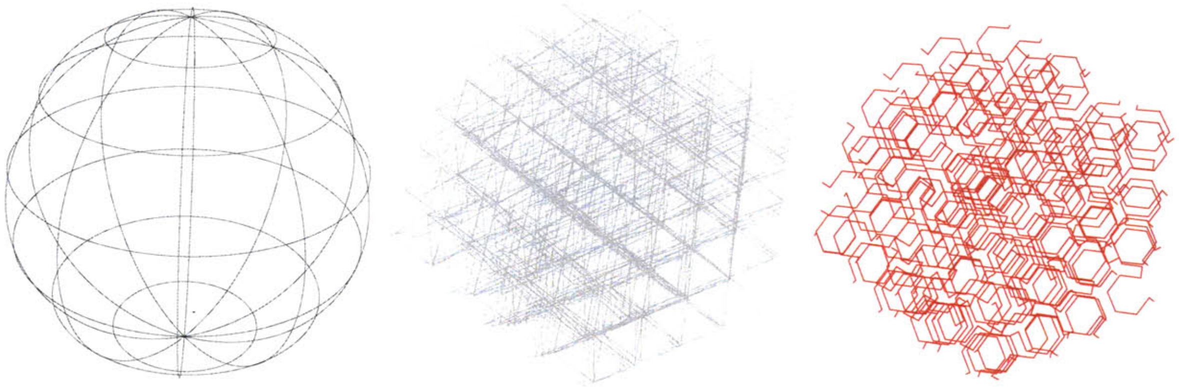


Figure 20. Volume packing with single path description.

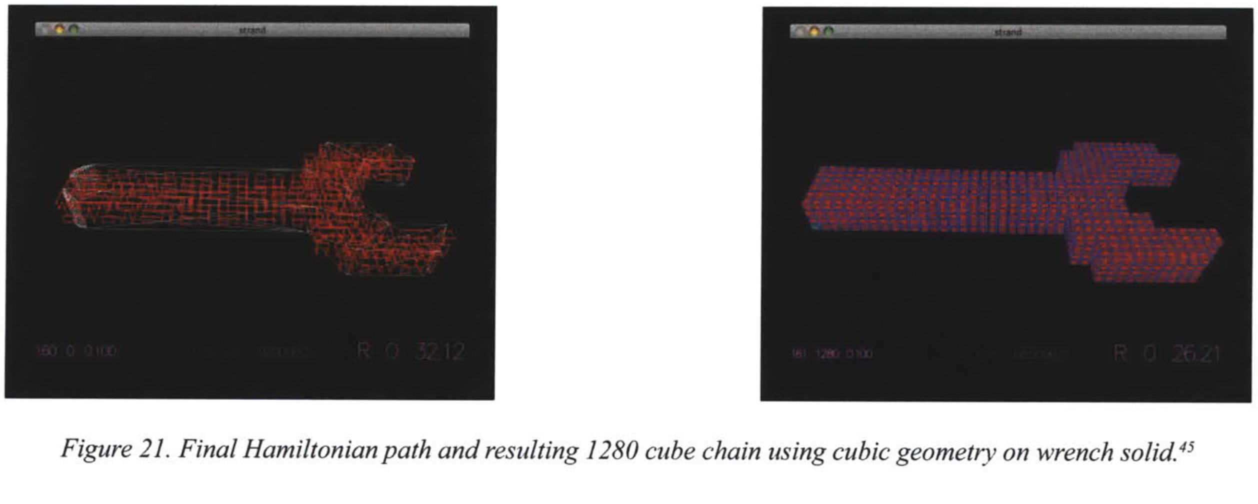


Figure 21. Final Hamiltonian path and resulting 1280 cube chain using cubic geometry on wrench solid.[45]

45 Jonathan Bachrach., V. Zykov, S. Griffith. Folding Arbitrary 3D Shapes with Space-Filling Chain Robots: Folded Configuration Design as 3D Hamiltonian Path through Target Solid.

Figure 21 shows two images from a 3D Hamiltonian Path algorithm by Bachrach et. al.. This algorithm is applied to chain robotics with cube geometries and finding a single path description that will search through the packed cubes, resulting in a 1D description of the interior of any solid 3D geometry. The algorithm works by first packing cubic voxels on the interior of any closed solid. A similar algorithm will be covered in more depth in the next section, with a slight modification for surface geometries rather than the interior of closed volumes. After having packed voxels the algorithm attempts to find a single path that will occupy every possibly position. The voxels are packed in a meta-module approach where large chunks are first placed then subdivided evenly to ensure that each meta-module can contain a single path description. Next, Bachrach et. al. demonstrate a method of incrementally merging, level-by-level, each of the single paths meta-modules to form the overall single path.[46] Bachrach et. al explain, "Paths can be merged if they share a parallel neighboring line segment. In this case, surgery can be performed on the two paths A and B at the parallel line segments to route the flow between the two paths."[47] The algorithm finally outputs a single path description of any closed volume that can be fed into a chain robot for folding sequences, allowing a chain robot to interpolate between geometric configurations.

This algorithm has been examined and redeveloped for surface applications with a focus on a simple search technique through the packed geometries rather than the incremental merging. The following sections will explain further details of surface descriptions and a number of techniques for single path configurations. Before proceeding I should discuss a final typology of *Logic Matter*, that of random growth. The previous implementations assumed that we are attempting to describe a given geometry: line, surface or volume. However, there may be instances that we do not have a given geometry and thus should grow from local rules. This relates to a larger field of self-organizing structures, where global patterns can emerge from only local interactions between individual elements, or particles.[48] This may also allow useful global configurations

46 Jonathan Bachrach., V. Zykov, S. Griffith. Folding Arbitrary 3D Shapes with Space-Filling Chain Robots: Folded Configuration Design as 3D Hamiltonian Path through Target Solid.
47 Jonathan Bachrach., V. Zykov, S. Griffith. Folding Arbitrary 3D Shapes with Space-Filling Chain Robots: Folded Configuration Design as 3D Hamiltonian Path through Target Solid.
48 Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous Computing. MIT. 2000.

to emerge, however the goal of a growing system would be to design local interactions or decisions for directional growth. At any given point we could impose a heuristic to persuade the growth to move in a more/less favorable direction, thus imposing an external factor onto the growth. Growth as a geometric typology directly relates to a number of biological systems and can offer obvious connections to evolutionary processes to achieve higher functionality or "fit" global configurations. The main focus of this thesis, however, will be describe known or desired geometries rather than arbitrary growths, however, it should be noted that linear or branching growth is certainly a potential area of exploration, one that links to many adjacent disciplines and may offer exciting opportunities when combined with digital information and computing possibilities.

In the following section I will explain a more focused look at surface descriptions using a number of single path techniques. I have elected to emphasize surface construction due to its versatility for a wide variety of architectural applications and close link to other geometries. If we are able to sufficiently describe a surface with a high percentage of coverage one can easily see how this can be translated to closed surfaces, thus describing hollow volumetric geometries. I will also introduce a specific algorithm that was developed called *Pack & Inverse Spiral* that appears to successfully describe any single surface geometry with a high percentage of coverage and search characteristics with few backtracking steps.

Surface & Random Walk

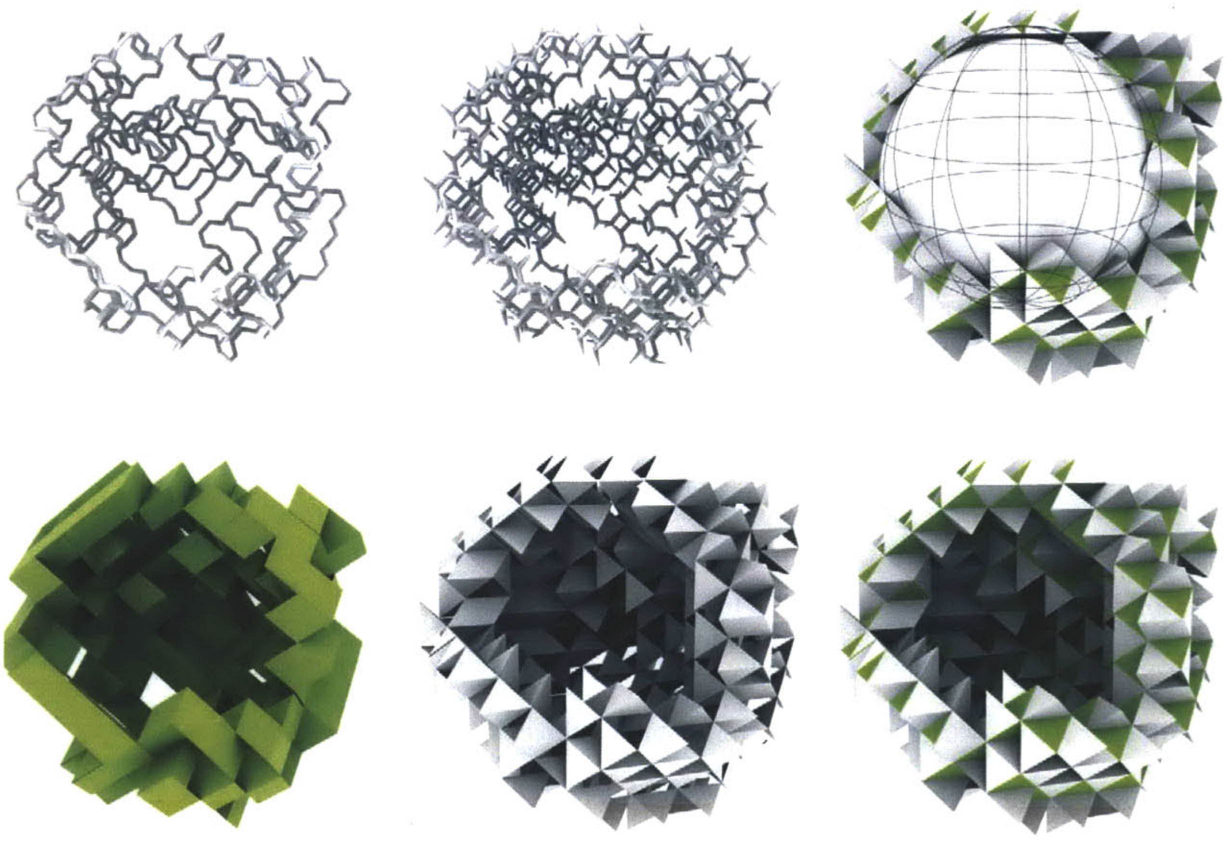


Figure 22. Random walk on sphere Surface. Single path, input & gate tetrahedrons.

```
-Get Surface
-Do:
        -Place 2 Output Gates
        -For Each Gate:
                -Check Dist to Closest Pt on Srf
                -Check for Self-Intersection
        -Select Min. Cost Tetra
        -Place Unit
        -Record Corresponding Input
```

Figure 23. Pseudo Code for surface descriptions through a random walk.

The first algorithm that was tested for surface description utilizes a random walk and local heuristics to simply traverse a given surface. An initial surface is given as the input for the tetrahedron description. Starting tetrahedrons are also input with specific attention to their 3D coordinates as they will dictate the starting position of the random walk. The first step of the algorithm places two possible output tetrahedrons.(See Figure 22) For each of the tetrahedrons the algorithm checks the centroid and resultant distance to the closest point on the surface. The closest point will be the straight line distance from the centroid of the tetrahedron to a point on the surface. This distance is compared for both of the tetrahedron centroid points. The closest distance is selected and the tetrahedron is stored. A last test is performed which will iterate through all of the stored tetrahedrons to check if the latest tetrahedron has ever been occupied previously (meaning the 1D chain has intersected itself). If the tetrahedron is found in the stored list then the alternative output option will be selected. Otherwise, the closest unit will be selected and the alternate tetrahedron option is removed. This process repeats until a exit condition is met (percentage of coverage, number of iterations or until the computer crashes!).

This type of algorithm is hardly successful at covering the surface evenly or thoroughly, it is also computationally expensive and relatively unintelligent as a means for searching the possible placements to cover a surface. However the code is sufficiently easily to write and the output can be more dynamic and spatially interesting when compared with the more successful surface covering techniques.

The random walk algorithm was quickly transformed into a slightly better performing technique - surface rastering. The surface rastering technique is actually a simple extension to the random walk, only differing in the heuristics for deciding the next output tetrahedron. I start this algorithm start by rastering over the entire surface and generating a single string of points that walks back and forth across the U & V directions of the surface. These raster points are stored in an array and then fed through the random walk algorithm to decide the local best move. At each move, both possible tetrahedrons are again placed and the centroids are obtained. The distances are now checked, not from the closest point on a surface, rather, I compare the distance from the centroid to the next point in the raster point array. Whichever tetrahedron has a smaller distance to the current raster point is placed and stored. This technique would seem to work quite well given that it makes a local decision that has already been pre-computed to determine a global description of the surface. However, this places a heavy burden on the spacing of the raster points in both U and V directions. If the spacing of the raster points is too small in either direction then the tetrahedrons will start to clump up and veer off course. They may also start to self-intersect one another or actually block the 1D chain from moving forward and alternatively have to go in another direction. This will perpetually get worse as the 1D chain moves farther and farther from the desired raster point because the raster points are iterating at a constant speed regardless of the speed and success of the tetrahedron placements. Conversely if the tetrahedron spacing is too great then the tetrahedrons may also start to waver from the raster path because they cannot actually move in a straight line and will thus move in larger diagonals between points.(See Figure 24)

The first fix would seem to be linking the speed and spacing of the raster path with the speed and size of the tetrahedrons. However, the speed of placing the tetrahedrons actually depends on a number of factors and could be hard to calibrate. Likewise the spacing of the raster points should not solely depend on the placement of the tetrahedrons because it will most likely stay in one place because it will be the easiest move to minimize the distance between itself and the tetrahedron. This technique was similarly abandoned to approach something without a predetermined "path" and something that is less likely to clump or waver from the surface. The following diagrams show a few successful attempts at describing surfaces with a raster path algorithm. I will next introduce two algorithms, *Pack & Spiral/Inverse Spiral*, that proved more successful and universal for describing complex surface.

Surface Raster Algorithm

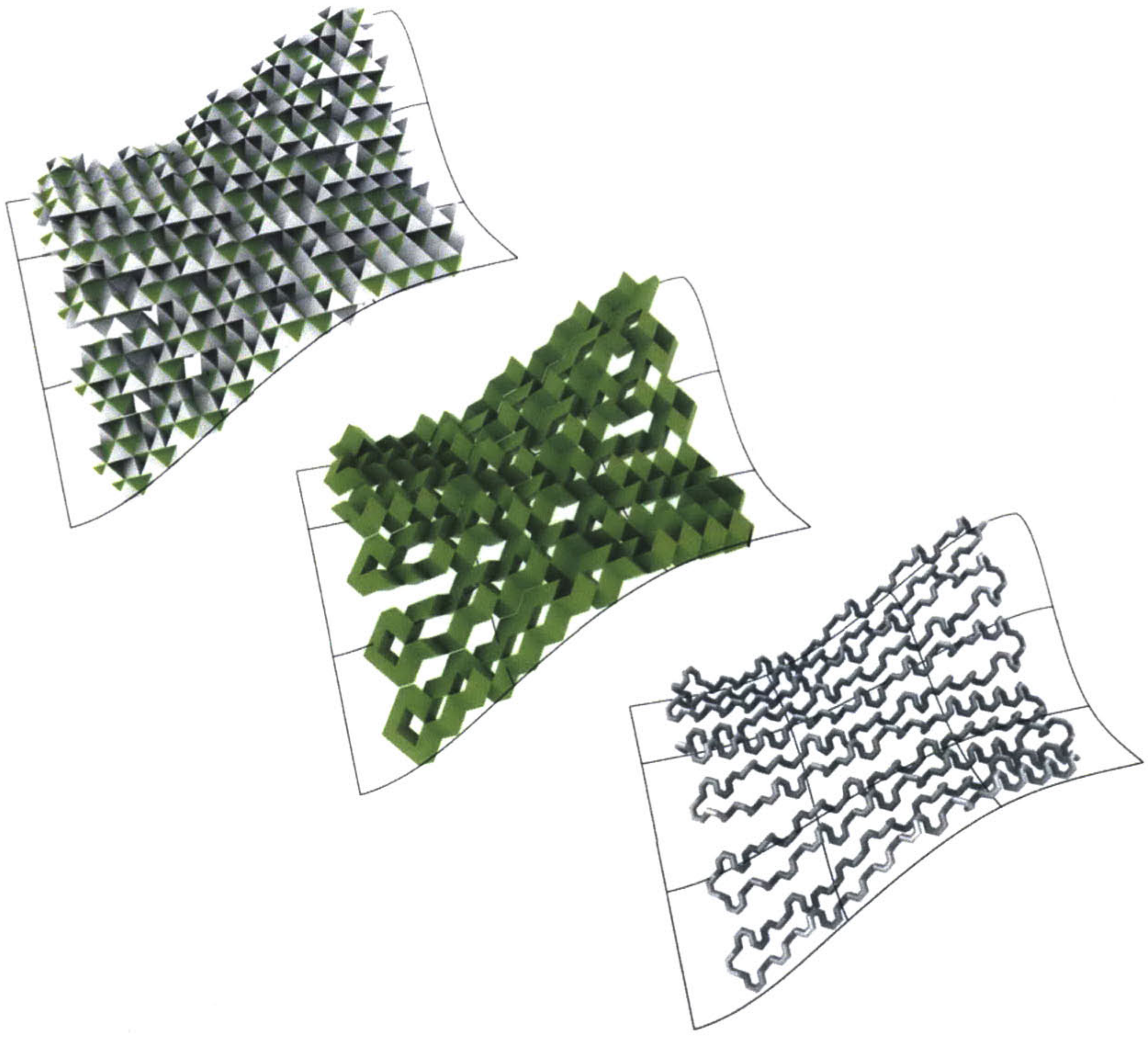


Figure 24. Raster search on a surface - Single path, green tetrahedrons are gates, white tetrahedrons are input

```
-Get Surface
-Extract U & V Coordinates in Raster
-Do:
        -Place 2 Output Gates
        -For Each Gate:
                -Check Dist to Raster Path
                -Check for Self-Intersection
        -Select Min. Cost Tetra
        -Place Unit
        -Record Corresponding Input
```

Figure 25. Pseudo Code for surface descriptions through a raster path.

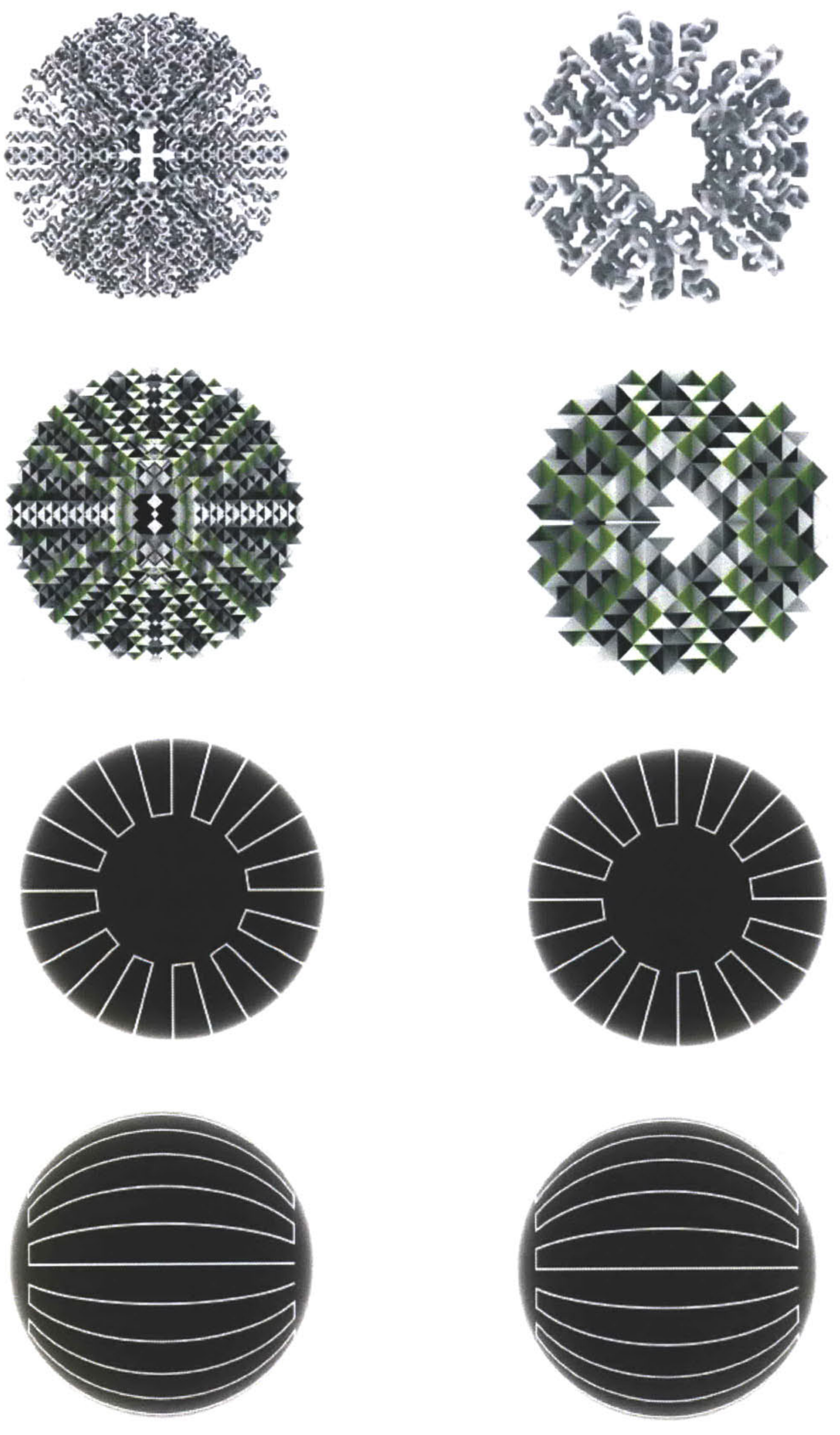


Figure 26. Raster search on Sphere - Single path, green tetrahedrons are gates, white tetrahedrons are input

The pack algorithm was designed with the idea that I should be separating the problems of searching along the surface from perfectly packing the surface. These problems seem to be connected as the search path should lead to the perfect pack, however through experience these problems tend to impact and conflict one another. For example, if we look at the random walk, this is simply a search across the surface that may or may not result in a perfect pack of tetrahedron units. The random walk may lead to self-intersection or miss regions, causing holes or clumping to occur. The raster path attempts to blend the two problems; however as was explained in the previous section, the raster path technique leads to overshooting and veering off course, resulting in inefficient descriptions of the surface. If I am able to separate the two problems then we should first look at the issue of packing tetrahedrons onto the surface evenly and perfectly. Only then can I search through them knowing that the given pack is efficient with maximum coverage.

The pack algorithm was actually trivial to code as it simply arrays a given set of tetrahedrons within the bounding box of a given surface. The user initially inputs the surface and an eight point bounding box is returned. This bounding box takes the largest dimension in all three axis and returns a cube encompassing these dimensions. Next, a set of tetrahedrons are selected and the resulting bounding box is returned. The surface bounding box is then divided in all three dimensions with the tetrahedron bounding box to compute the number of possible tetrahedrons that can be packed. These three division factors become the number of iterations for copying the tetrahedrons in the X, Y and Z dimensions. The final sequence of code simply takes each set of tetrahedrons, checks the centroid distance to the given surface and keeps or deletes them based on a given threshold dimension. If the centroid distance is farther than the threshold, lets say 12" (or the actual width of the tetrahedrons depending on the user's criteria), then the tetrahedron is deleted. This will result in a perfect packing of only the tetrahedrons that are actually touching the given surface. This lays the ground work for the next step, searching for a perfect, non-self-intersecting 1D path through each tetrahedron.

Pack & Spiral/Inverse Spiral Algorithm

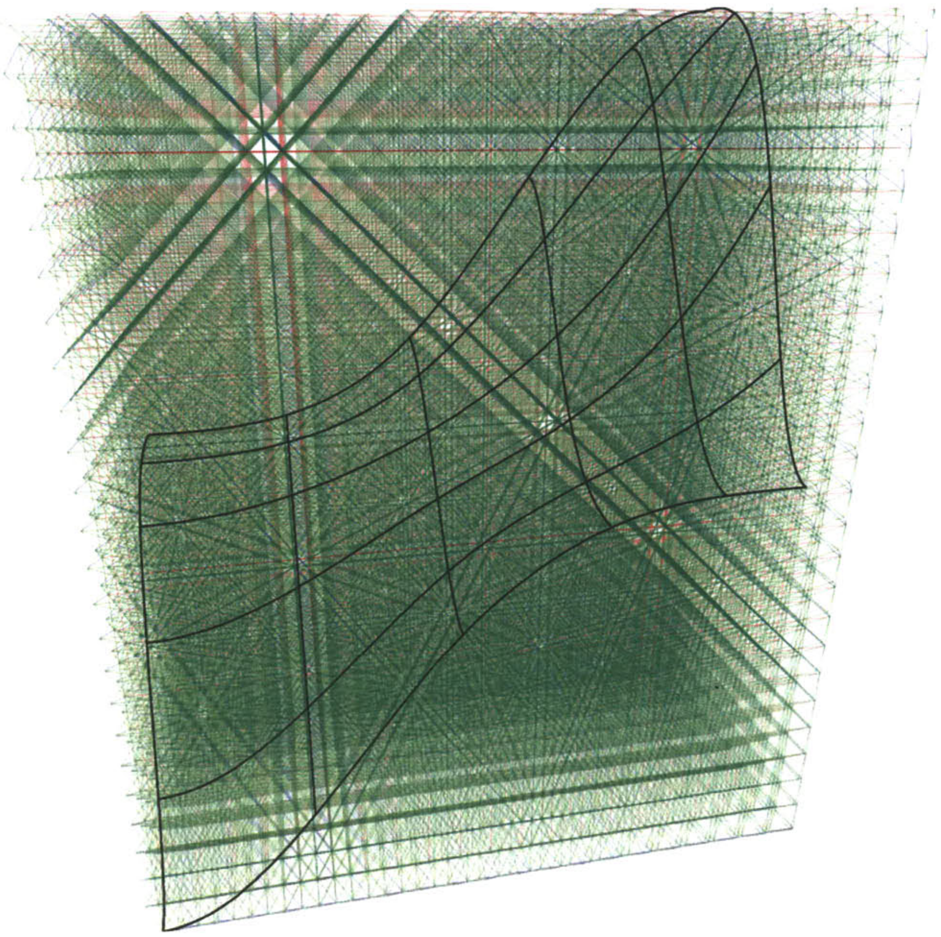


Figure 27. Packed tetrahedrons in 3D bounding box of a given surface.

```
-Select Surface
-Get Bounding Box
-Select Tetras
-Get Bounding Box
-Divide Srf Bbox dimensions by Tetra Bbox dimensions
-For x dimension:
        -For y dimension:
                -For z dimension:
                        CopyTetras
```

Figure 28. Pseudo Code for packing tetrahedrons inside surface bounding box.

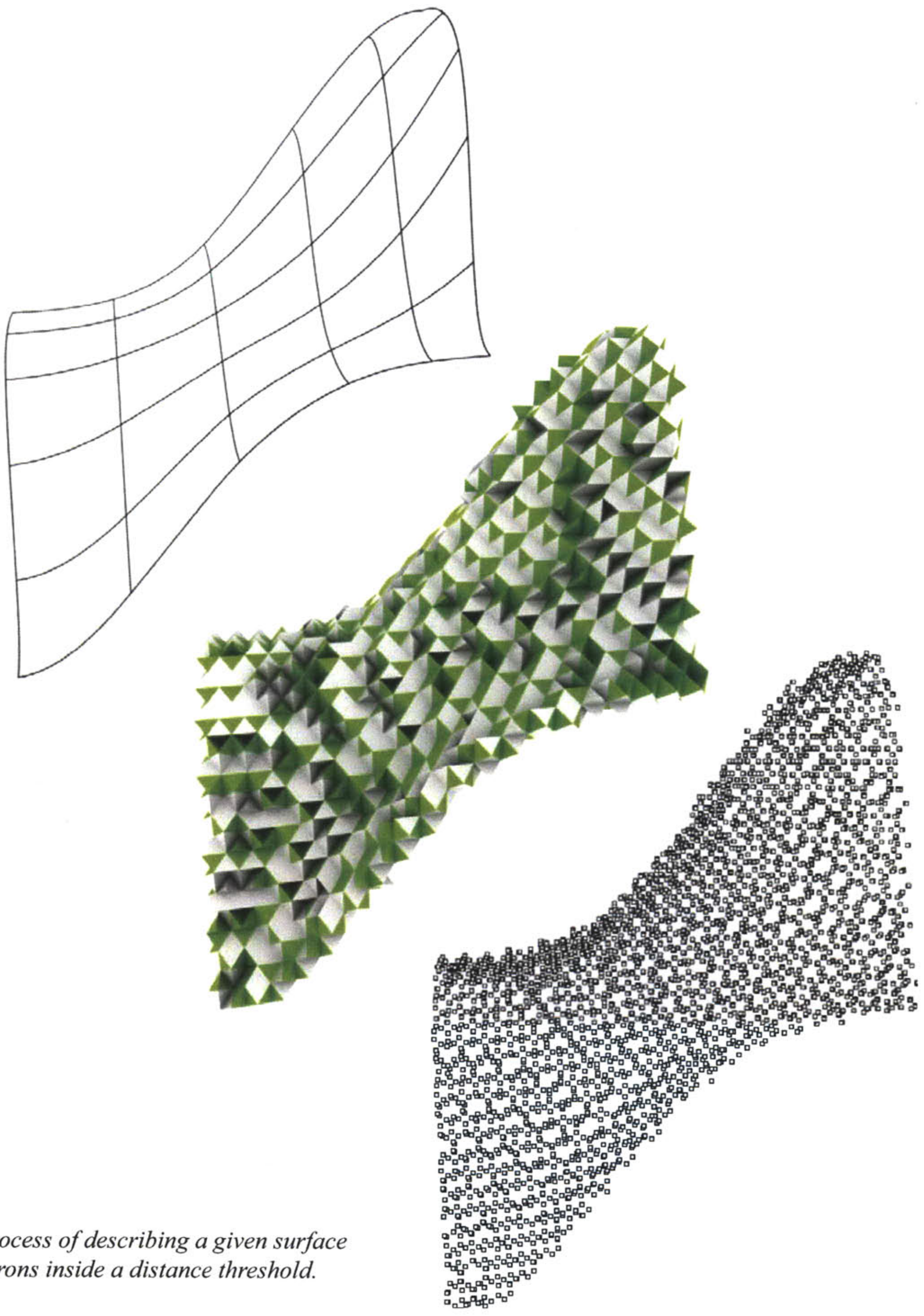


Figure 29. Process of describing a given surface with tetrahedrons inside a distance threshold.

```
-Select Grid of Tetras
-Select Surface
-For each tetra:
        -Get Centroid
        -Check distance to closest point on Surface
        -If Surface is outside range:
                -Delete Tetra
```

Figure 30. Pseudo Code for removing tetrahedrons outside a distance threshold.

The first implementation of search through this newly packed surface lead to the spiral algorithm. The spiral algorithm actually attempts to generate a spiraling path through all of the tetrahedrons starting at a given center point. This algorithm takes all of the tetrahedrons, the initial surface and a starting point as input. For a given number of iterations or until the path has occupied every tetrahedron the code runs and checks possible next moves. Each iteration looks for the nearest points to the previous tetrahedron. It then checks to see if the potential points are actually direct neighbors of the starting by checking centroid distances to the current tetrahedron and making sure the potential unit is within a distance threshold. If the tetrahedron is proved to be a direct neighbor it double checks that this tetrahedron has never been occupied previously. This makes sure that the 1D path is not self-intersecting. Finally the algorithm checks to see the distance between the potential neighbors and the centroid of the surface. The algorithm prefers to minimize the distance to the centroid in effect creating a spiraling motion outwards. If it cannot find a neighboring tetrahedron that satisfies each of the conditions then it will backtrack one step repeatedly until it finds a successful point.

This algorithm initially appeared to be very successful in covering complex surfaces as it slowing described larger and larger portions of the surface. However, with further iterations it was realized that the algorithm tends to get stuck in the corners or in one half of a surface. This is a direct result of the outward spiral. Presumably the algorithm could eventually describe the entire surface if backtracking was repeated enough times to get out of the corners and difficult zones. However, the next implementation, the inverse spiral, proved to be a stronger approach for describing more complex surfaces.

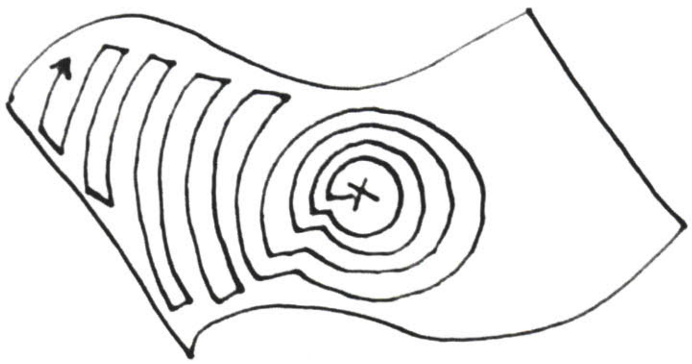


Figure 31. Pack and Spiral progression sketch.

```
-Select Tetra Centroid Points
-Select Surface
-Get Surface Centroid Pt
-Get Starting Point
-Do Until Surface is Covered:
        -For each Tetra Pt: #FIND POTENTIAL POINT
                -If (Pt not Used):
                        -If (Neighboring PrevPt):
                                -If (MIN Dist to Centroid): #SPIRAL
                                        -Make Potential Pt
        -If NO Pt Found: #BACKTRACK
                -Step Back
        -Else:
                -Take first & add to list
                -Draw Temporary Plyline
```

Figure 32. Pack and Spiral pseudo code.

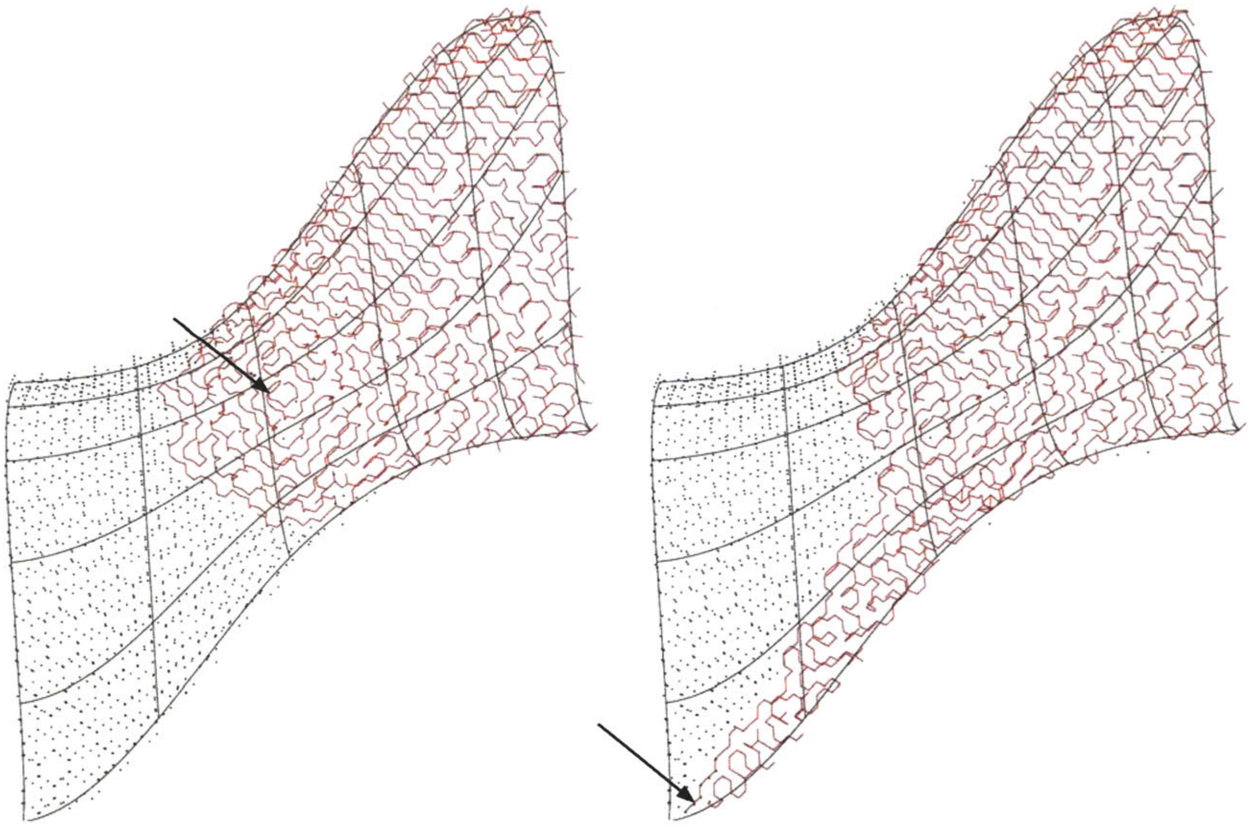


Figure 33. Local Min/Max problems as the single path gets trapped in corner conditions.

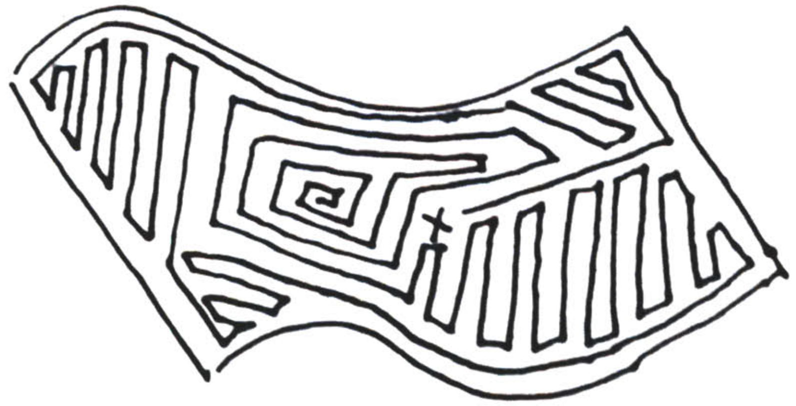


Figure 34. Pack and Inverse Spiral progression sketch.

The final implementation took the spiral approach and flipped it to create an inverse spiral. For the most part the inverse spiral actually has same input and conditional checks as was described in the spiral algorithm. However, the inverse spiral checks each potential next move for the farthest distance from the initial starting point (or the centroid of the given surface), rather than the closest point as was previously described. For example, if we look at each tetrahedron and ask 1. is it actually neighboring the current tetrahedron that we occupy 2. has it not previously been occupied by our path and 3. is it farther than any other previously checked tetrahedron, then we should consider any tetrahedron that satisfies these constraints as a potential next move. For any given position that we can occupy there can only ever be three possible tetrahedrons that would satisfy these three constraints; the three touching tetrahedrons on the opposite faces from where we entered. We decide between these three tetrahedrons in exactly the same manner, by taking the tetrahedrons with the farthest distance from the given surface centroid. This effectively creates an inverse spiral. The path initially shoots from the center out to one of the edges. It then starts congregating in one of the corners until it reaches a point where it can climb along the edge to the other corner. At this moment the path has generally covered the entire first corner and the alternative corner is farther from the centroid. It repeats these steps until all of the corners are complete and it has created a pseudo empty circle around the centroid. It then spirals inward from the empty circle until it reaches the original centroid and cannot find a free tetrahedron. At this point the single path has successfully described the given complex surface with a 1D chain.

```
-Select Tetra Centroid Points
-Select Surface
-Get Surface Centroid Pt
-Get Starting Point
-Do Until Surface is Covered:
        -For each Tetra Pt: #FIND POTENTIAL POINT
                -If (Pt not Used):
                        -If (Neighboring PrevPt):
                                -If (MAX Dist to Centroid): #INV. SPIRAL
                                        -Make Potential Pt
        -If NO Pt Found: #BACKTRACK
                -Step Back
        -Else:
                -Take first & add to list
                -Draw Temporary Plyline
```

Figure 35. Pack and Inverse Spiral pseudo code.

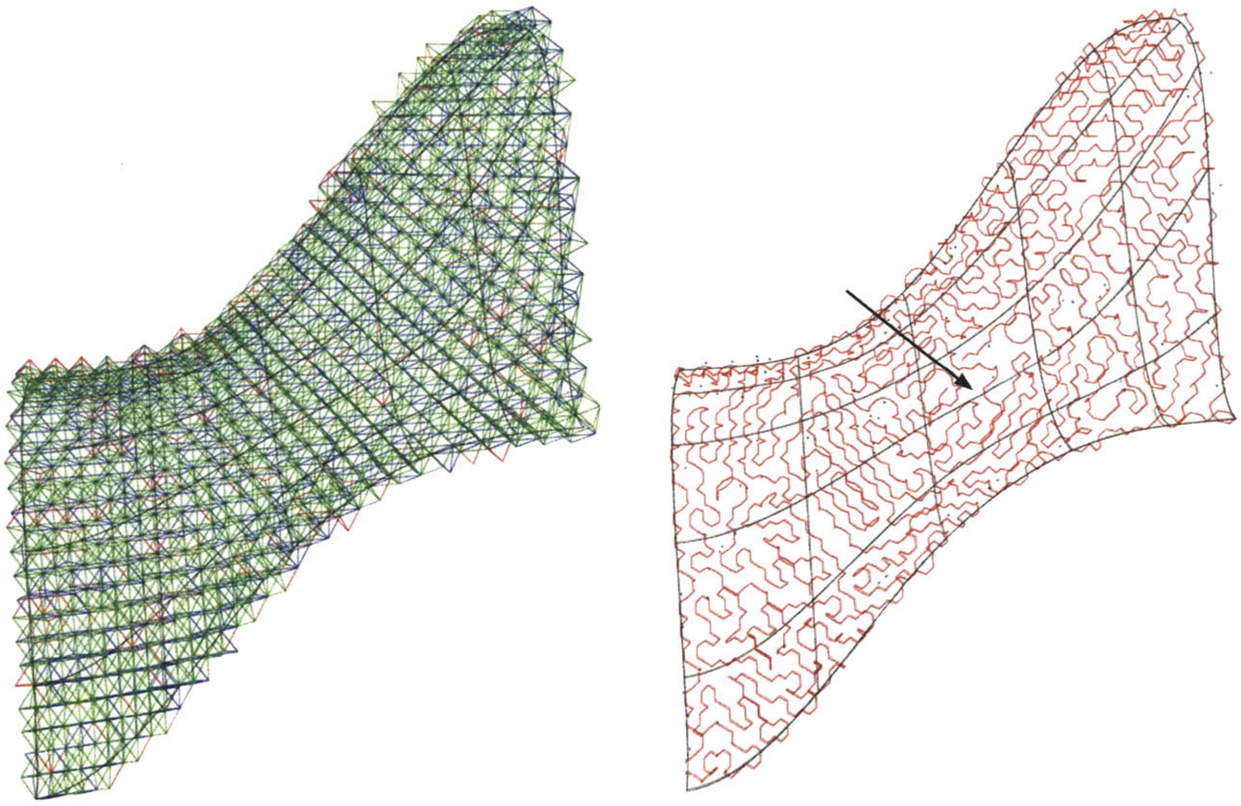


Figure 36. Pack and Inverse Spiral description of a complex surface.

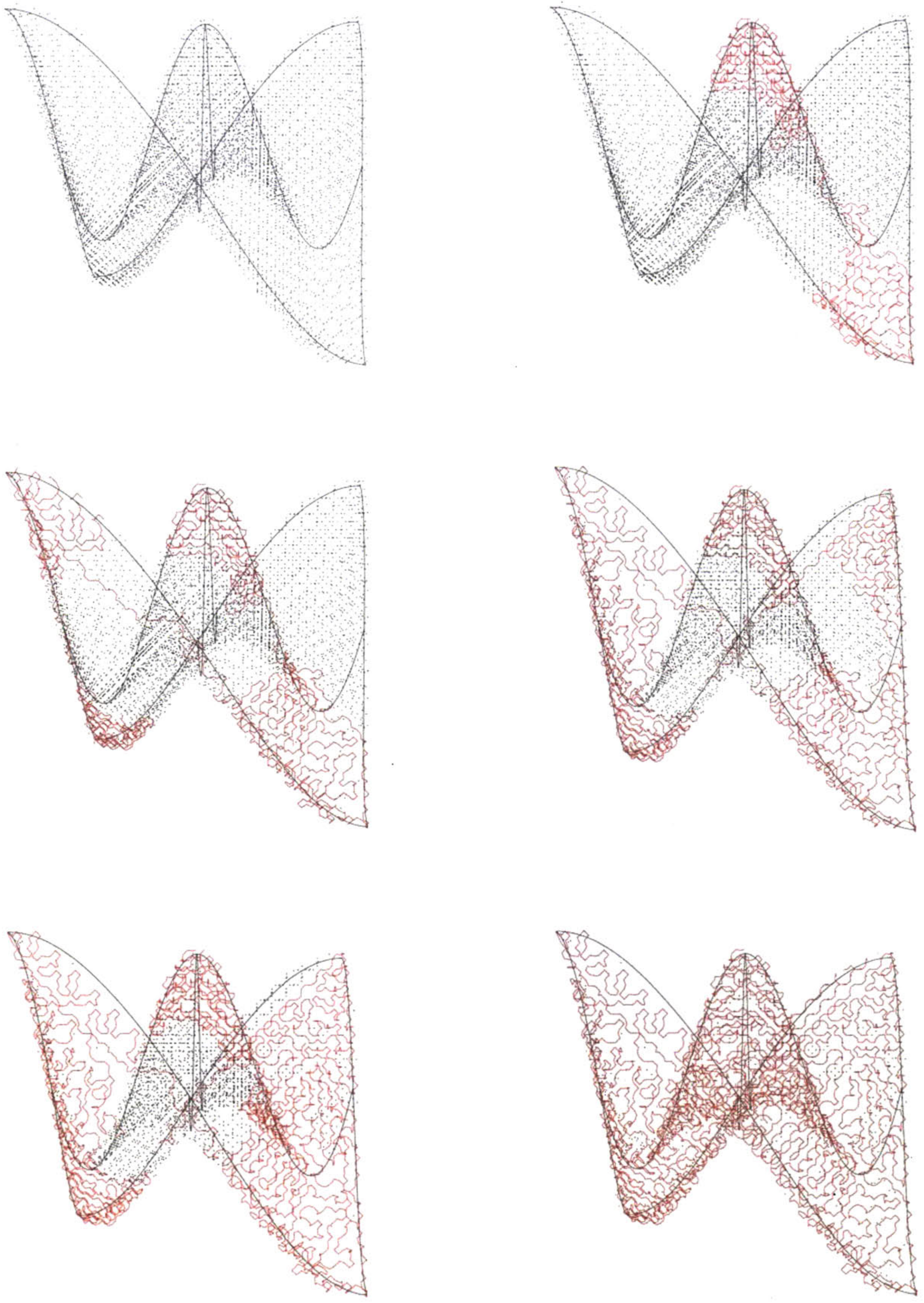


Figure 37. Pack and Inverse Spiral growth sequence, describing a complex surface.

Pack & Inverse Spiral Analysis

To compare the search algorithms we can evaluate the number of points occupied versus the total number of points available to deduce the percentage of surface coverage. We can also take the length of the curve as the efficiency of coverage, with preference towards a shorter length curve because it implies that the path has not wondered or backtracked an exorbitant amount. The pack and spiral algorithms result in a less than successful approach to the task of full surface coverage. The first attempt, starting from the centroid of the surface, gives a total surface coverage of 61% with a total number of occupied points at 1,791. The total length of the curve is 19,079 inches. Conversely, the same algorithm when started from the corner of the surface, rather than the centroid, resulted in 29,134 total inches. The percentage of coverage only increased to 68% with 1,996 total points occupied out 2,951 total points. This implies that the corner starting point resulted in an excessive amount of backtracking and inefficient paths because the length of the line is drastically higher while the percentage of coverage only increased by 8%.

The inverse spiral algorithm compares favorably to the standard spiral algorithm in both the percentage of surface coverage and the total length of the line. The first inverse spiral attempt resulted in a 97% surface coverage with a total number of occupied points at 2,854 out of 2,951 total points. The length of the curve was 30,867 inches which is less than double the standard spiral attempt that only covered half of the surface before getting stuck in the corners. Similarly, the second inverse spiral attempt ranked well in the surface coverage with 92% and 4,575 points occupied out of a total, 4,981. These numbers show the relative success of the inverse spiral technique compared with the spiral and raster approach with respect to the percentage of surface coverage, efficiency of the single path description and number of backtracks. This technique should be tested on further surfaces and extreme curvature areas to test its robustness to failure and getting stuck in local min/max conditions.

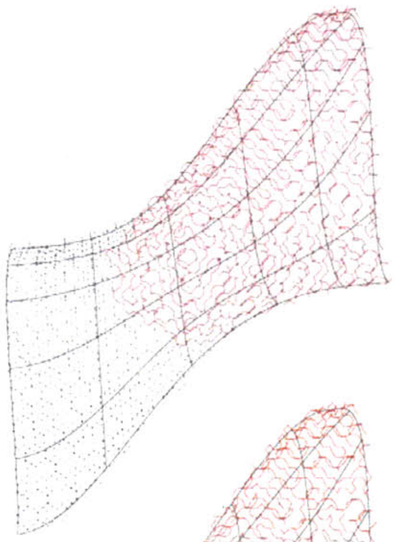
Center Spiral

Number of Points Total: 2951

Number of Points Occupied: 1791

Percentage of Coverage: 61%

Length of Curve: 19079"



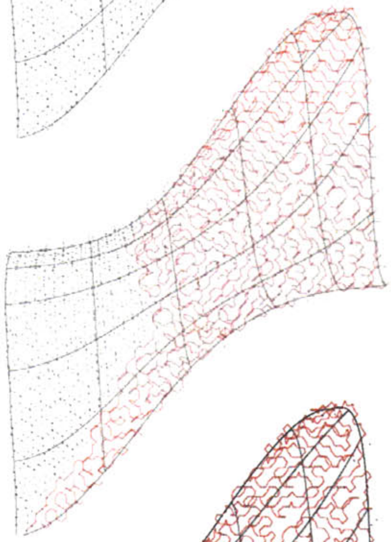
Corner Spiral

Number of Points Total: 2951

Number of Points Occupied: 1996

Percentage of Coverage: 68%

Length of Curve: 29134"



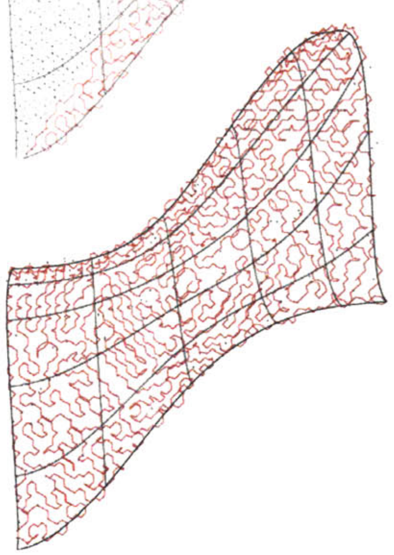
Inverse Spiral 01

Number of Points Total: 2951

Number of Points Occupied: 2854

Percentage of Coverage: 97%

Length of Curve: 30867"



Inverse Spiral 02

Number of Points Total: 4981

Number of Points Occupied: 4575

Percentage of Coverage: 92%

Length of Curve: 81241"

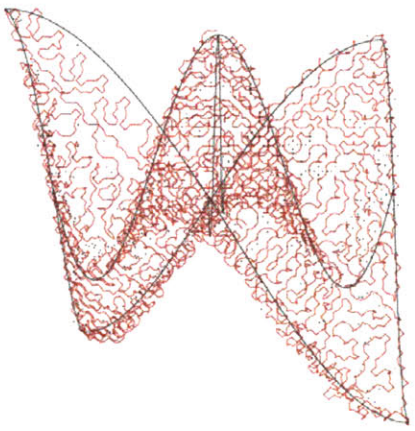


Figure 38. Pack and Spiral/Inverse Spiral Analysis.

Computing Through Construction

I have demonstrated, through a number of geometric examples, that we can do useful things with *Logic Matter*, such as describing the interior and exterior of closed volumes/spaces, describing complex surfaces as well as any 1D, 2D or 3D curves in space. Architecturally and spatially speaking, the ability to construct and describe a plethora of geometric typologies is useful and an essential element of any building component. If I am to argue the benefits of *Logic Matter* as an aid for assembling these geometries, it is imperative that I show specifically how they can advance the process of assembly by utilizing the computational abilities and stored digital information within the NAND mechanism. We have already seen how the entire descriptions of complex geometry can be translated into a single binary sequence of physical moves (up, down, left, right). I will now try to go a few steps further and explain the powerful opportunities of computation that lie in *Logic Matter* and how these translate to the process of assembly.

In this section I will emphasize four beneficial qualities of computing offered by the NAND mechanism for physical assembly:

1. Only local knowledge is necessary
2. Material as a storage device
3. Stopping mechanism
4. Single stream processor and hard drive

I will demonstrate these capabilities of computing through five primary examples:

1. Building an infinite line
2. Building a square
3. Failure and disassembly
4. Building infinitely small structures
5. Self-Guided-Replication

Building an Infinite Line

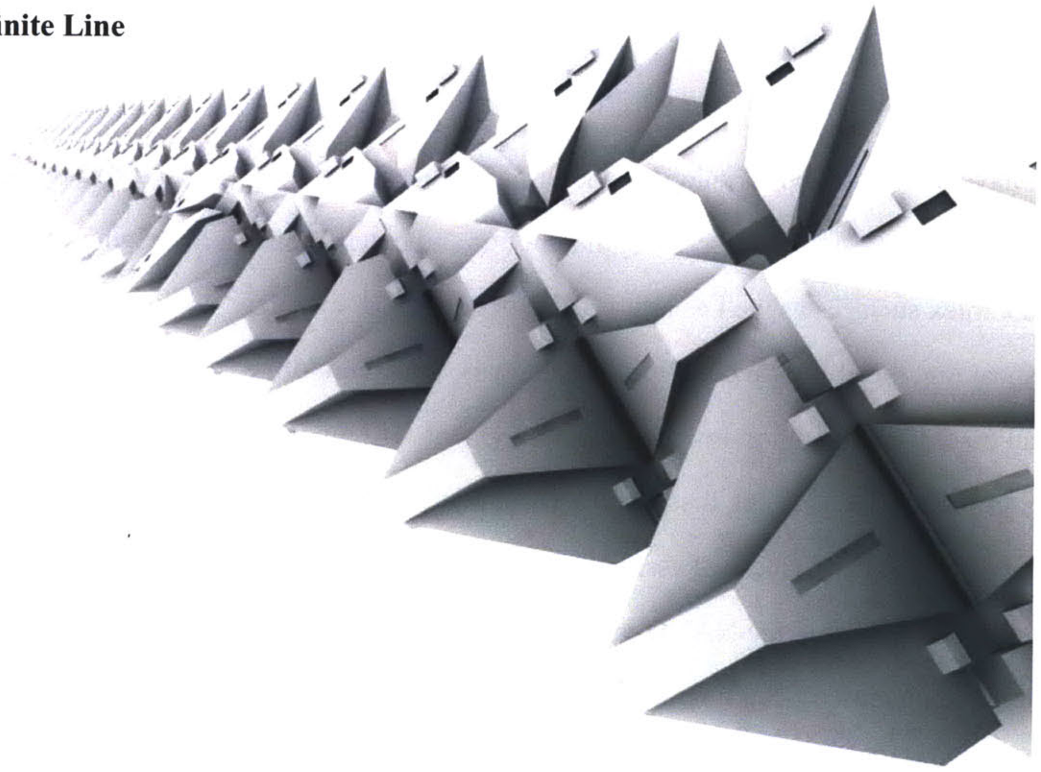


Figure 39. Infinite Line Example with NAND mechanisms.

As the structures in our environment scale in extreme directions, up or down, with the number of parts and overall size, it is important that we start thinking about the local information needed, step-by-step, to build complex global geometries. Let us imagine a scenario where we need to build an infinitely long, perfectly straight, line (or wall) from an infinite number of small elements. Through this example we can quickly see a number of problems with our current construction techniques and why we need to focus on local information. As the length of the line increases, the difficulty of understanding the global geometry decreases. That is, the farther along in the process of construction we are, the less we know about what we have already done. If we suppose that the line has been constructed for 20 miles, it has already become fairly difficult to ensure that we have proceeded in a perfectly straight manor. (We might use a series of checkpoints to ensure we have gone straight. However, that only ensures that we can fix some of our error, not actually dictating that the line has been perfectly straight. Similarly, we might not even be able to produce the checkpoints in a guaranteed straight line.) It is also difficult to know exactly how much further we need to go, if in the beginning, we were only told the total length (precise to some small range, for this example we can imagine that we need to be precise to a 1/32" and our units are roughly 8" in length). We cannot be sure that we have proceeded in a straight line because at each step we are only making an estimate of being straight from the previous pieces that were placed. We also cannot know precisely how much farther we need to proceed unless we regularly remeasure what we have already constructed. This process of remeasuring, or knowing the global structure, will often become laborious and/or impossible. In our infinite line example, the process of remeasuring is nearly impossible because eventually we do not have tools long enough to measure and if we use a small measuring tool repeatedly, end-to-end, we will inevitably accrue error. Likewise, we cannot continually trace our steps to figure out if we have proceeded in a perfectly straight path and most likely we will have strayed from the path with some high amount of tolerance.

In the future, if we are to build structures larger than humanly possible today, we need to only rely on local information and be guaranteed of the global consistency. This local information, in the case of *Logic Matter* is actually computed through the NAND mechanism. With each placement of input there is only one possible output. This output dictates the next direction to proceed with accuracy, consistency and redundancy. The user, constructing the infinite line, only needs to know the local sequence of inputs because they contain the information needed for the next series of inputs as well as the guarantee that the global structure has been constructed accurately. There are two inputs values at each step and an orientation required. The first input value is directly computed from the last output (therefore this is eliminated from the user), the orientation is also directly computed from the last output and saves the user from having to know exactly where to place the next module. If the global sequence of input values is designed in a specific way, the user also does not need to know the 2nd input value at each step.

```
1 | 1111111111111110000000000000000
1 | 1111111000000001111111100000000
1 | 1110000111100001111000011110000
1 | 1001100110011001100110011001100
1 | 0101010101010101010101010101010
```

Figure 40. Binary Counting as Input Sequences for Infinite Line

Binary counting can be seen as important example for the infinite line problem that allows the 2nd input value to be removed at each step. Let's say that our infinite line is constructed from a number of NAND mechanism directly adjacent to one another and that each NAND mechanism contains one binary digit at each step. We can then imagine that it would be possible to initially set our NAND bits to an infinitely large binary number simply by creating that binary number as inputs to the NAND mechanism. For example, if we want to build a line with 32,767 steps then we can initially set fifteen NAND mechanisms to the sequence: 111111111111111. The next sequence of inputs would be one less than the previous, 32,766 or 111111111111110 as inputs to the NAND mechanism. We can imagine that we would do this repeatedly until we do not have any further carry digits and we land at the state 000000000000000 as inputs. This binary counting example provides two major advantages when utilized as input. First, this allows the user to only know the local sequence, thus, the system of units actually stores the current state and tells the user the next state through a pattern of decreasing 1's in standard binary counting. In this way the user does not need to know anything about the global configuration to be confident with how to proceed at the next step and guarantee that the global structure has been constructed in the proper geometric path. Finally, this example of binary counting as physical input informs the user how much further to proceed and more importantly, when to stop building! When the user reaches the point when they have all 0's as input they know they have reached the end of the line and they know that the line has been constructed geometrically precise according to the initially designed length. This demonstrates the possibility and exciting benefits of computed assembly only through local information.

Building a Square

To demonstrate the benefits of local and stored information, the complexity of the structure need not be infinitely long and immeasurable. To take this idea slightly further, we can imagine constructing a square (or any closed object) from a sequence of inputs. If it is known that the sequence of inputs to construct a square is alternating seven digits of 0's then seven digits of 1's four times, then we can most likely picture the overall input/output relationship. (To scale the square we could simply increase the number of repeated digits in a series. For example, nine digits would increase the length of each side of the square etc.) However, if this sequence got slightly more complicated or slightly longer, it would be much more difficult to remember exactly where you are in the overall sequence and what input to place next. If we are constructing the square with the NAND mechanism, we are in luck because we can rely on the storage capacity of the materials to tell us what we have previously input, where we are and what we should place next. Further, this can be entirely done within the local series we are constructing. The NAND mechanism will eternally store the overall sequence of inputs, simply by looking at the faces of the input mechanisms utilized. We could imagine that building any large structure will take a series of days/weeks/months/years and that we will need to stop and start construction numerous times. If we were required to remeasure and recalculate the place where we left and decide what the next input should be, we would lose tremendous time and accuracy due to human error. The NAND mechanism allows us to directly know the output of the last move and the previous seven moves (or however many local moves are in the repeating sequence) in order to dictate the only possible next move. At any time we can look back at the sequence get the stored information and know exactly what to place and how much further we need to build in order to complete the global configuration.

The example of the square contains one final important characteristic, error trapping. The closed nature of the square ensures that any errors that were built within the process of assembly will be contained within the square. Errors may accumulate based on material tolerances, material stresses or any number of real world symptoms and our NAND mechanism can either attempt to deal with these errors directly or worse, the mechanism my add to the errors.(See Section 01 on Digital Information) Any closed geometry provides another layer of error trapping, aside from the features of the mechanism.(See Section 05 on Globally Specific Units) As the square is closed and the last unit is forced to connect with the first, we are guaranteed that the errors, however large they may have grown, will only be contained within the square (as long as we can force it closed). Likewise, if we connect another square to the first square, we can be sure that the errors within the first square will not affect the next square or the global structure. This example demonstrates the benefit of having locally closed aggregations that add up to larger structures, the potentials for error trapping, local information and the ability to store information within the NAND mechanism.

Failure and Disassembly

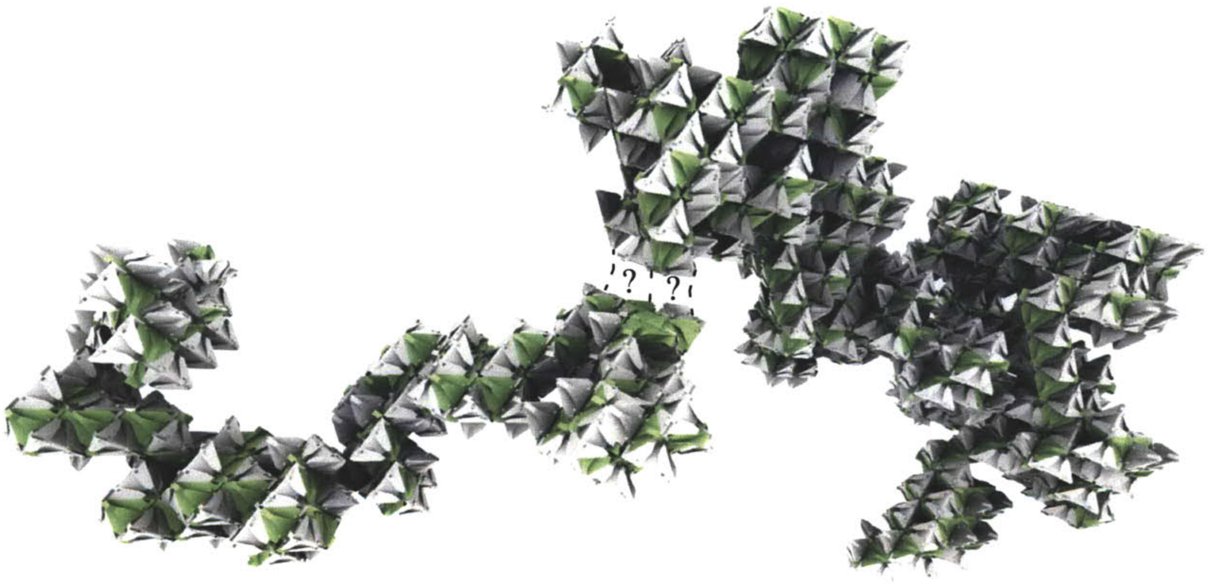


Figure 41. Failure and Disassembly Example. A complex structure with a single point of failure that can be reassembled accurately by reading the adjacent unit's stored information.

The ability to stop and restart an assembly process seamlessly, without recalculating or remeasuring is an extremely beneficial characteristic of computing and storing information while building. This can even be utilized after the global configuration has been complete! In this example we will imagine that a large, extremely complex, structure has been built, containing an infinitely long and arduous sequence of inputs. If this structure were to eventually fail or collapse at any point along its length we might imagine that we would find ourselves in an extremely difficult situation trying to reconstruct the failed portion. However, as we have already seen, the NAND mechanism computes and stores its state as the materials are assembled, thus, we can utilize this principle for reconstructing failed portions. We know that the structure will eternally contain the assembly blueprints because the encoded information (inputs/outputs) used to construct the assembly will always remain as part of the structure, not degrading over time or distance. From the stored sequence, we can deduce that there can only be one possible local move to replace a failed unit that will complete the circuit. Similarly, there is only one possible face and orientation to place a unit, based on the surrounding inputs/outputs that could be utilized to re-link the chain. The units that surround the failure will explicitly dictate the orientation, type of input and eventual output that is needed to fix the failure. We could actually replace the parts without the blueprints! This is an incredible property of the stored information within a *Logic Matter* assembly. We can relate this to the telecommunications industry and the benefits of digital information for reconstructing broken or lost signal sequences.[49]

Stored assembly information can be utilized in a variety of ways from providing an external assembly mechanism the exact instructions for replication to part encoding within disassembly procedures. We can take the example of the relocation of the London Bridge, where every part was identified, disassembled and completely relocated to Lake Havasu, Arizona.[50] *Logic Matter* already contains all of the information for constructing, disassembling and reassembling the structure, directly in the material parts. The material parts indicate the process of assembly by computing through construction! We can imagine a world of applications for structures that can dictate their instructions for replication and encoded information for disassembly, guarantees on global configurations and many others. Let's now look at a world of infinitely small structures and the types of information that might be required.

49 Neil Gershenfeld. Fab.
50 David Crouch and Nina Lubbren.Visual Culture and Tourism. (Oxford, UK: Berg Publishers, 2003).

Building Infinitely Small Structures

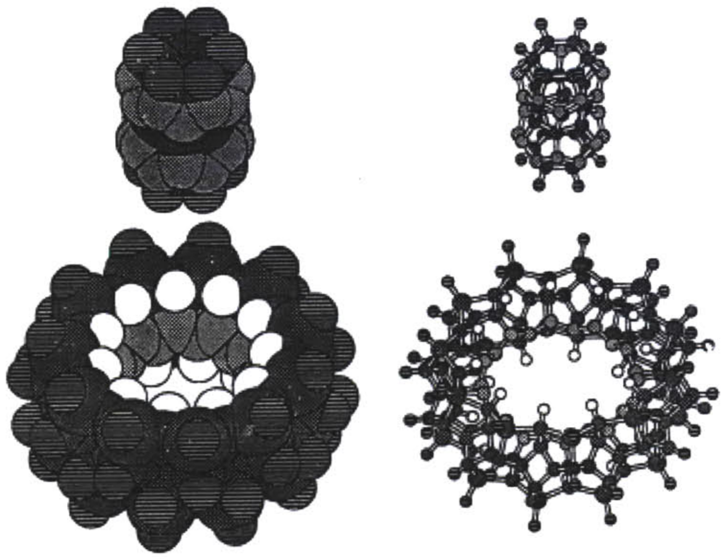


Figure 42. Diagram for Molecular Machinery from K. Eric Drexler's PHD dissertation: Molecular Machinery and Manufacturing with Applications to Computing. [51]

By attempting to build structures that defy the known limits of scale, either large or small, we will face new assembly challenges, specifically with the amount of information required, the type of information (discrete or continuous), accruing errors and a plethora of other issues of assembly. If we imagine building structures at minute scales there emerges an interesting paradox between the possible types of computation and the devices which we can build. The first problem lies in the assembler or machine that will need to construct these structures since it is most likely not possible for a human to manipulate the elements directly. At extremely large scales we may see the same problem because eventually we cannot build machines that are larger than the parts we want to produce. In both cases we will need some type of device that can locally climb, deposit and assemble material without knowledge of the global. The small scale example better emphasizes the problem that we will eventually need to build functional assembly "machines" at extremely small scale lengths (possibly biological machines) with extremely limited computing and storage capacity. Without computing or storage capacity like human assembly or robotic assembly, how can we code the necessary assembly information to ensure that we make the correct decisions at each step? Can we use the material as a hard drive to store

51 K. Eric Drexler. Molecular Machinery and Manufacturing with Applications to Computing. (PHD diss. MIT, 1991).

the information for assembly, identify where we have been and how far we need to go? The material would literally tell the machine what to do and the machine would simply be used as a force. *Logic Matter* offers just that. The information for assembly is embedded directly within the assembled material, we simply need a machine to be able to read the string and translate the local sequence into the physical output of the next move. The material dictates the next move and with certain input sequences the step-by-step inputs are determined simply by reading the material already placed. This relieves the necessity of having universal computation in the assembler (human or machine) and distributes information throughout the structure, bit-by-bit storing and building its own computation.

As a supplemental example of small scale assembly let us look at an opportunity afforded by the *Hairy Chain* redundancy, creating scaffoldings for growth. Scaffolding can be seen in a number of construction applications from the building industry to biological materials.[52] Scaffolds afford a temporary structure to allow for the construction, assembly or growth of a secondary system. The redundancy of the input tetrahedrons implies that only the redundant units would need to be pre-assembled, then the gate units could actually grow or fold within the scaffolding, assuring that the final configuration was constructed correctly and within the correct local moves (left, right, up, down). The scaffold material could be made to dissolve or could be less expensive, while the gate units could be made more permanent, therefore emphasized by dissolving the redundancy and only leaving the single chain gate geometry. The scaffolding also could allow the system to be scaled either extremely large, to have a cheap and quick substructure for an extremely large-scale chain to be folded on the interior. The scaffold actually contains all of the information required to build the structure, thus eliminating the information required in the actual gate units as to which direction to fold. On the opposite end, the scaffold could be approaching the biological scale and allow a protein-like structure to fold or grow within the information-infused scaffolding. This scaffolding example demonstrates another possible feature for embedding localized information directly into the assembled materials, thus easing assembly and guaranteeing accurate, complex structures.

52 Martin P. Vacant. 2008. Biological Scaffolding Material. (U.S. Patent 7,319,035,B2 filed, October 17, 2003 and issued January 15, 2008).

To expand the idea of the single strand building its own computing device, it is important to first compare a number of existing computational architectures with respect to the storage of processing information versus data information. Implementations such as the Harvard Architecture (construction information and data are stored separate) and Von Neumann Architecture (construction information and data are stored in the same place - RAM), common in almost all CPU's today, are opposite in their approach to information and data storage with advantages and disadvantage in speed, memory usage and robustness.[53] Conversely, the important aspects of *Logic Matter* are; (1) stored information in physical materials, or materials as a hard drive. (2) The information to do the computing, i.e. the digital gates (AND, OR, XOR, NAND etc.) are built directly into the mechanics of the parts and are utilized by varying the sequences of inputs and thus the NAND mechanism. (3) The computing is built with the exact same string of information as the data itself, that is, the string of inputs actually builds the architecture to compute the same string of inputs. This is significantly different than most of the current computing models because of the lack of distinction between program information and data information (or construction/computing information and input sequences). *Logic Matter* contains no distinction between the information to build a series of AND, OR, XOR gates that arise from sequences of NAND mechanism and the same exact string of inputs that the gates will compute as a result. (4) *Logic Matter* respects the laws of physics in terms of locality and time, unlike the common computing models where the laws of physics are suppressed, i.e. similar information is not stored with proximity to one another and the amount of time to use that information does not depend on distance.[54] David Darlymple and Neil Gershenfeld from MIT's Center for Bits and Atoms explain, "Physics inherently allows only local information transfer, and computation, like every other process, relies on physics."[55] Darlymple and Gershenfeld emphasized the importance of designing a system of computing that relies on the laws of physics and locality, offering increased speed and parallelization.

53 David Dalrymple. Asynchronous Logic Automata.
54 David Dalrymple. Asynchronous Logic Automata.
55 David Dalrymple. Asynchronous Logic Automata.

Let us look at a concrete example to help clarify the computational benefits of *Logic Matter* with its simultaneous computing and constructing, requiring only a single input string. Field-programmable gate arrays (FPGAs) are a integrated circuits which consist of a two dimensional array (or grid) of logic gates.(See Figure 43) As Marchal explains, the FPGAs "can be programmed at three distinct levels: (1) the function of the logic cells, (2) the interconnections between cells, and (3) the inputs and outputs. All three levels are configured via a string of bits that is loaded from an external source, either once or several times." [56] After the FPGAs have been "programmed," or wired, to connect the various types of logic gates, a second string of inputs is fed through the circuit and determine the computed output.(See Figure 44) This has distinct advantages in terms of reconfiguration and reprogrammability of the circuit, distinct from the actual computation of the input bits. However, in the case of *Logic Matter*, we actually reduce the amount of information to perform the computation to a single string of inputs, rather than two separate strings (one to program the configuration of the computing device and one to send the bits to be computed). *Logic Matter* actually builds the computing device as the bits are streamed. For example, two bits, [0] and [0], are placed into a Gate unit and dictate the result [1] which forces the placement of the next Gate to be on the [1] face of the previous Gate. (In the UP direction) (Essentially this passed an input, [1], into the new Gate. This Gate is now waiting for a second input.) The first Gate has just acted as a functioning NAND mechanism (0 & 0 = 1). If we then place a new input, [1], into the latest Gate we will get an output of [0]. (1 & 1 = 0). That means that the two NAND mechanism in serial have acted directly as an AND Gate or an OR Gate because the initial inputs of [0] and [0] have ultimately resulted in the output of a [0] after two Gates of computation. In this example we can see how the sequence of inputs ([0],[0],[1],[1]) as well as the functionality of the NAND mechanism actually constructed the logic gates and architecture for computation simultaneously while computing the exact same sequence of inputs!

56 Pierre Marchal. Field-Programmable Gate Arrays. (Communications of the ACM; Apr99, Vol. 42 Issue 4).

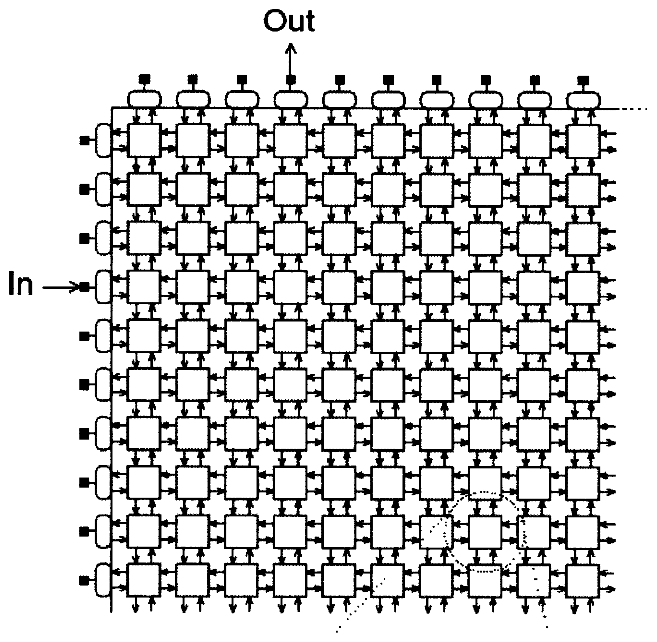


Figure 43. Field-programmable Gate Array. Two dimensional array of digital logic gates. [57]

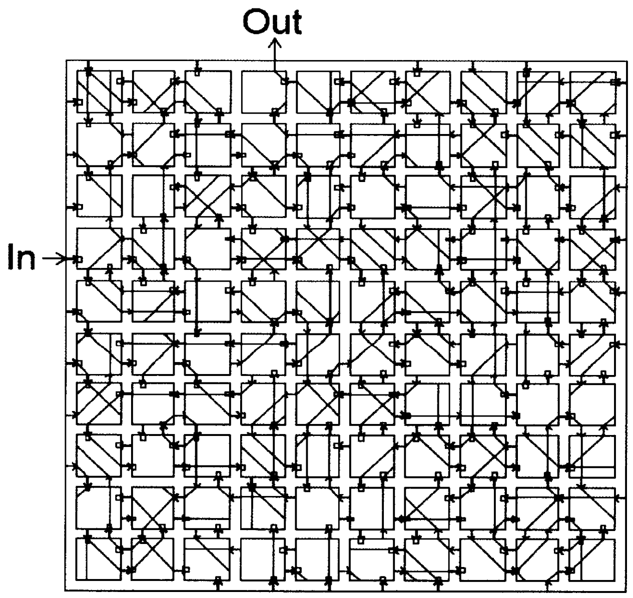


Figure 44. Routed Field-programmable Gate Array. First input sequence has dictated the gate connections. A second input sequence will base the bits to be computed. [58]

57 Adrian Thompson. An evolved circuit intrinsic in silicon entwined with physics. (COGS University of Sussex Brighton, UK).

58 Adrian Thompson. An evolved circuit intrinsic in silicon entwined with physics.

Self-Guided-Replication

As previously described in Section 03, there have been a number of attempts at synthetic (non-biological) self-replication systems, not to mention the countless examples of biological self-replication processes (i.e. DNA replication).[59] Most notably, Von Neumann's work and L.S. Penrose's Mechanical Self-replication system where mechanical latches were demonstrated to be able to duplicate an arbitrary input string.[60][61](See Figure 04) Penrose further described the necessary requirements for any self-replicating system:

> First, the replicating structure must be built by assembling simpler units present in the environment. Secondly, more than one design can be built from the same set of units though the only replicating structure that can be automatically assembled will be one exactly copying a previously existing structure.[62]

The proposed NAND mechanism of *Logic Matter* provides the unique property of material storage, containing the sequence of information used to construct any arbitrary configurations. This offers the key ingredient to accomplish self-replication because the actual structures contain the blueprints for an identical copy. The assembler (human or machine) can throw away their initial knowledge, blueprints or input sequence that was used to construct the first structure, because the second, third and infinite number of further structures can be built by directly reading from the first assembly. The redundant input tetrahedrons store the sequence of inputs that could be used as a direct read-write procedure to produce the inputs for duplicate structures. We could imagine that a machine could be built (on any number of scales) that would only read the local information from the materials, step-by-step working its way along the single path, producing a simultaneous action/procedure on a duplicate structure. This would eventually create the perfect duplicate configuration that could be then used for another duplicate and so on.

Logic Matter may not be able to claim full self-replication, seeing that it does not have self-locomotion or electronic programmability. However, the example of using the material as a

59 Saul Thomas Griffith. Growing Machines.
60 L. S. Penrose. Self Reproducing Machines.
61 John Von Neuman. Theory of Self-Reproducing Automata.
62 L. S. Penrose. Self Reproducing Machines.

storage device clearly demonstrates the possibility of self-guided-replication, where the system informs another assembler (either human or machine) the explicit sequence of instructions to build a duplicate structure. Self-guided-replication might have applications spanning from the automation of manufacturing complex structures to biological realms and replicating natural scaffolding construction, most of which will not be addressed in this thesis.[63] However, it is important to emphasize self-guided-replication as an exciting potential of *Logic Matter* and computing through construction. Through examples like building an infinite line and self-replication we have seen that *Logic Matter*, a system of programmable physical logic gates, can be utilized for useful geometric descriptions and applied to a plethora of applications by utilizing local computing for assembling complex structures.

The following Figures; Figure 46, Figure 47, Figure 48 and Figure 49 show attempts at computing larger assemblies through varying techniques of binary inputs. Figure 46 and Figure 47 show the sixteen possible combinations of repeating binary inputs with four initial placeholders. All combination between 0000 to 1111 were tested for their resulting configurations. All of the repeating patterns result in straight lines or repeating structures, a somewhat obvious output of the binary inputs, however if they are utilized in combination they can be made to turn in any direction and to describe straight line segments of any geometry.(See Figure 48) Figure 48 and Figure 49 show gradient inputs of the number of 0's and 1's in an input sequence. These results are far more interesting spatially and far more complex, making it harder to guess the relationship between input to output. As seen in the simple repeating patterns, most of the configurations are not necessarily useful, however if they are utilized in sequence we could potentially realize exciting potentials for describing complex geometries with simple input sequences. These input sequences should hopefully contain the beneficial qualities as seen in the infinite line example where we can not only guarantee the success of the structures we are creating, but the input can actually inform the assembler of the next moves, store the information of the past and announce the completion of the assembly.(See Figure 39)

63 Saul Thomas Griffith. Growing Machines.

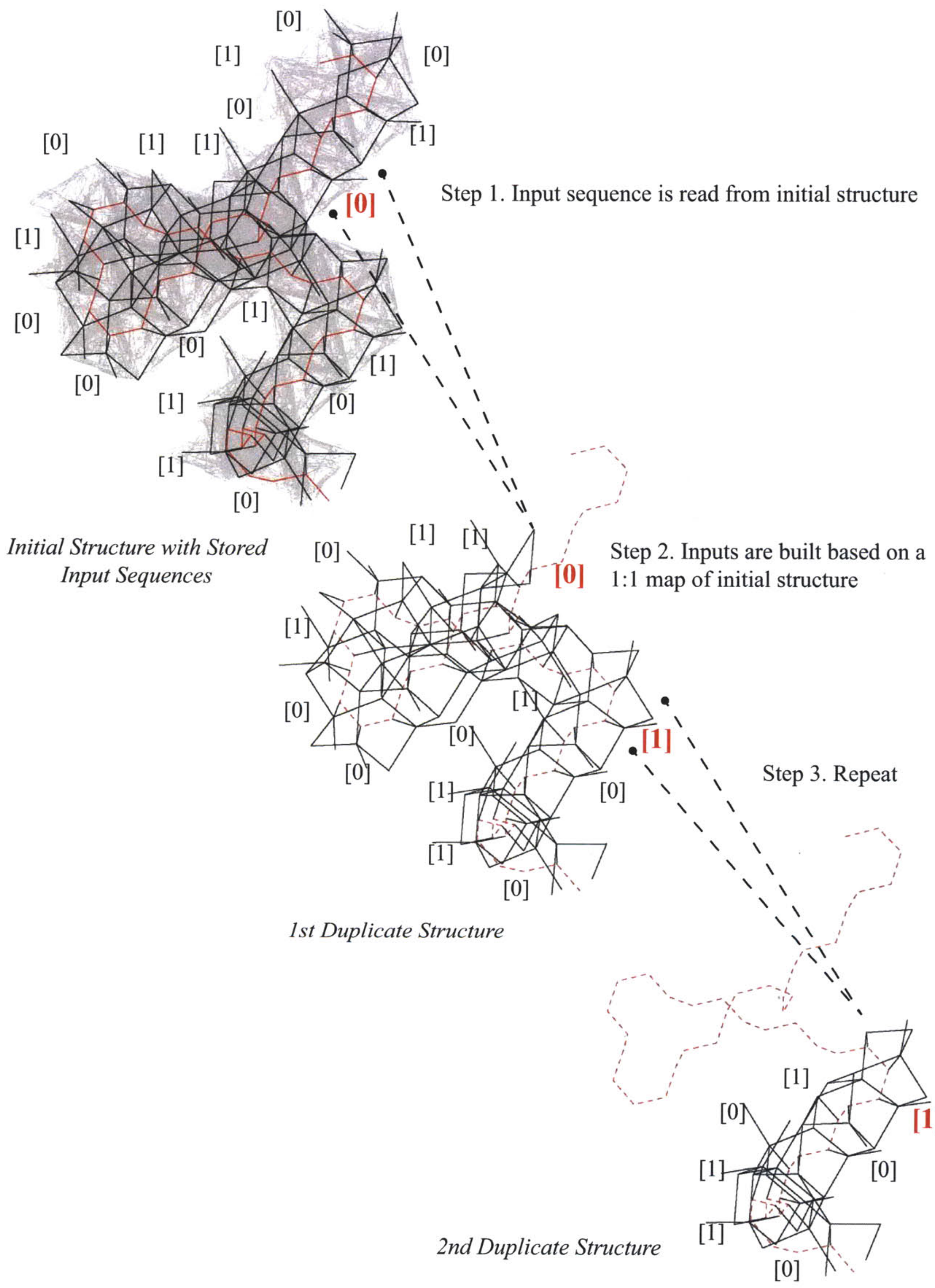


Figure 45. Self-Guided-Replication. A single sequence is read and written to generate duplicate structures.

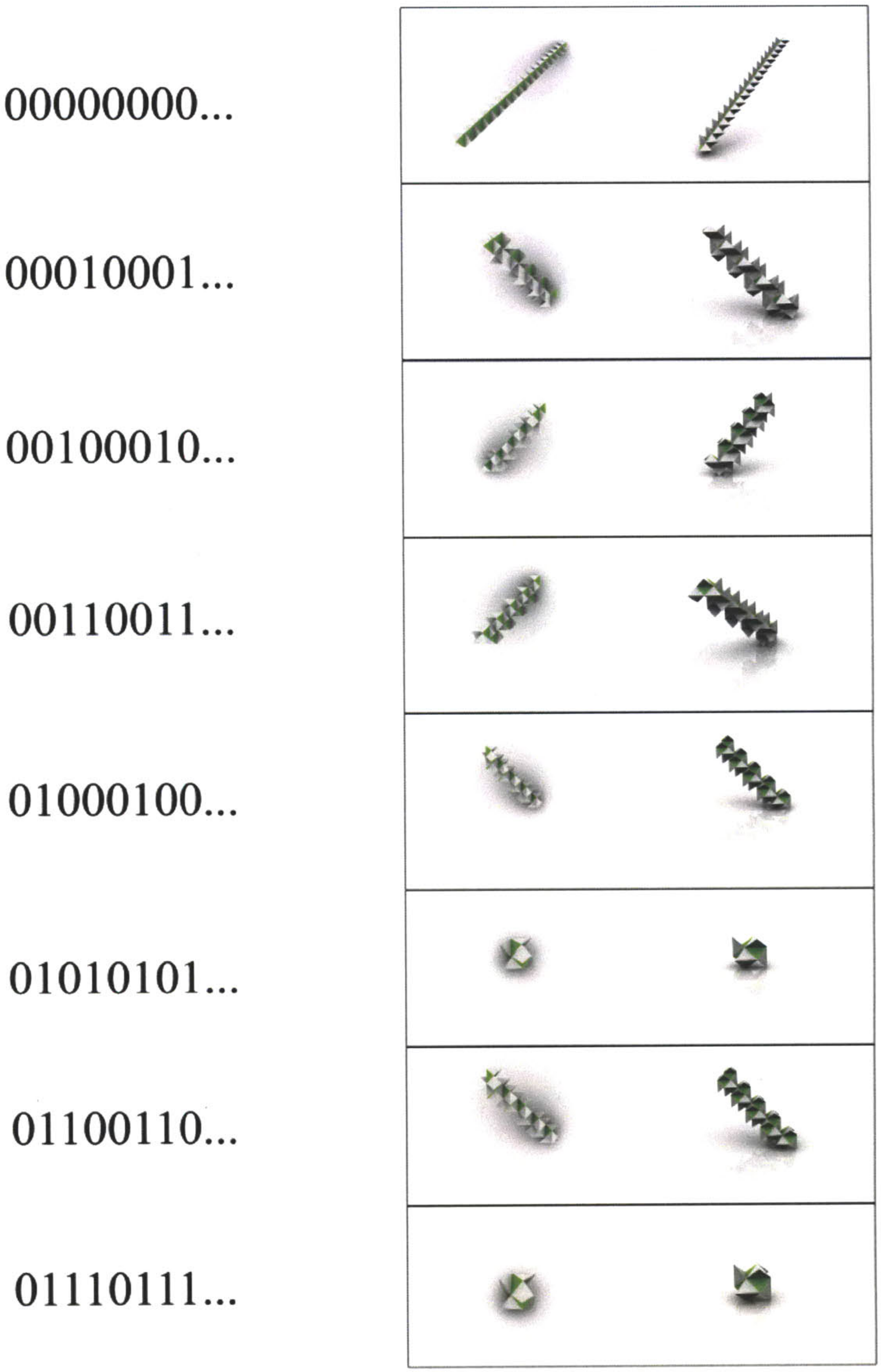


Figure 46. 0-7 binary numbers as a repeating input pattern.

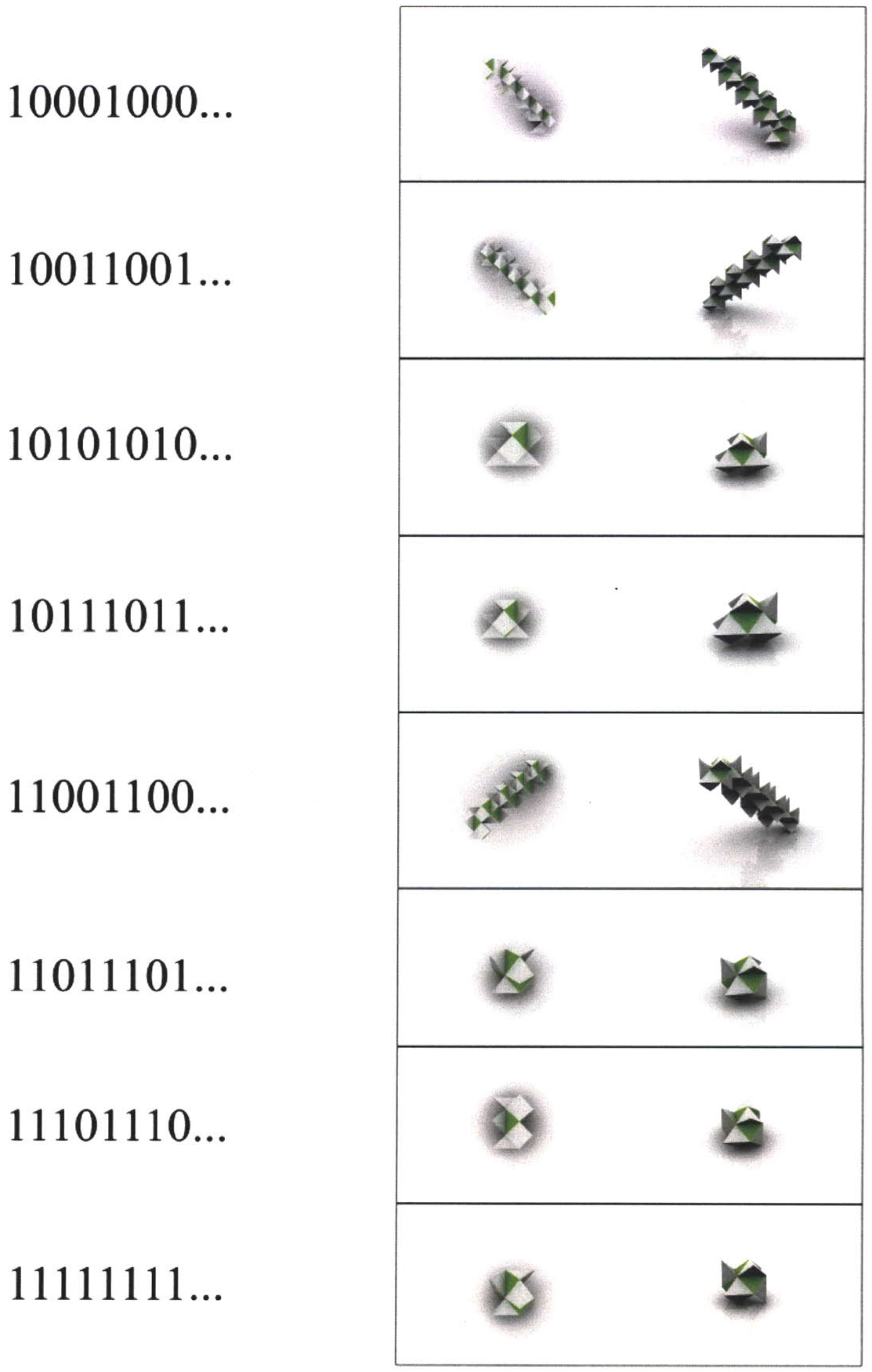


Figure 47. 8-15 numbers as a repeating input pattern.

Pattern: 10100100010000100000100000001...

Spacing: 1 2 3 4 5 6

Linear Increase

Pattern: 1010100100100010001000010001...

Spacing: 1 1 2 2 3 3 4 4

As x increases....x/2 = #0's

Pattern: 1010101001001001000100010001...

Spacing: 1 1 1 2 2 2 3 3 3

As x increases....x/3 = #0's

Pattern: 10101010100100100100...

Spacing: 1 1 1 1 2 2 2 2

As x increases....x/4 = #0's

Pattern: 10101010101001001001001001...

Spacing: 1 1 1 1 1 2 2 2 2 2

As x increases....x/5 = #0's

Pattern: 1010101010101001001001001001001...

Spacing: 1 1 1 1 1 1 2 2 2 2 2 2

As x increases....x/6 = #0's

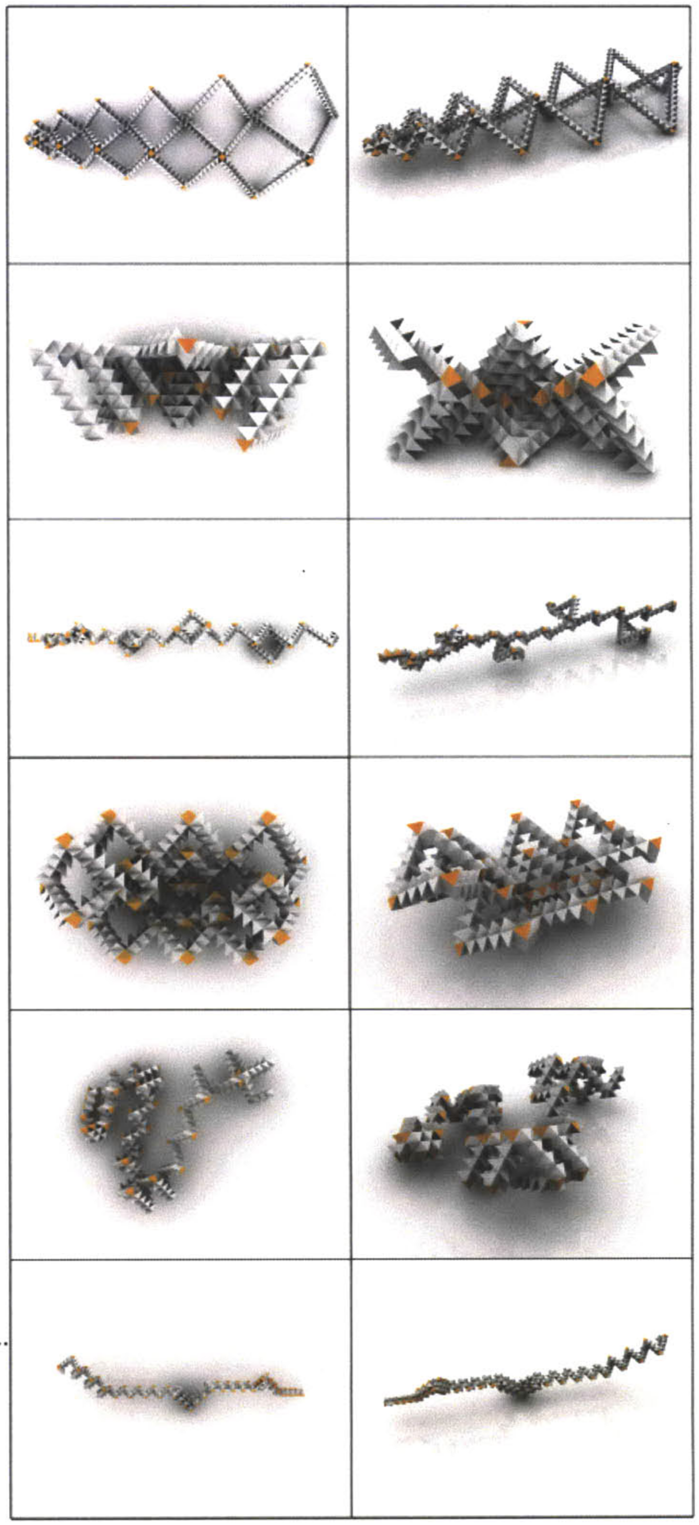


Figure 48. Binary gradient sequences & resultant spatial output. Orange units are inputs with value 1.

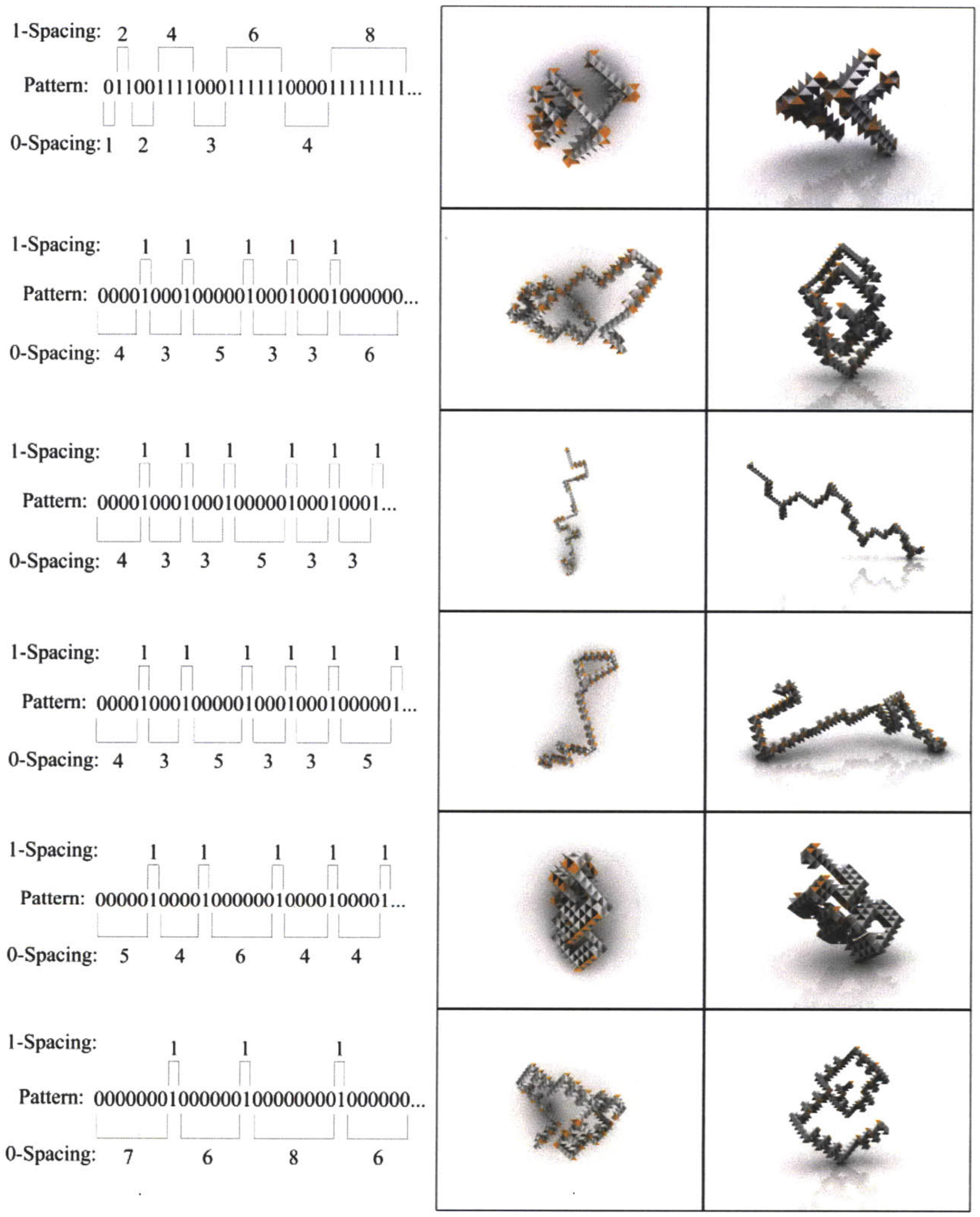

1-Spacing: 2 4 6 8
Pattern: 01100111100011111100000111111111...
0-Spacing: 1 2 3 4
1-Spacing: 1 1 1 1 1
Pattern: 00001000100000010001000100000000...
0-Spacing: 4 3 5 3 3 6
1-Spacing: 1 1 1 1 1 1
Pattern: 0000100010001000001000100001...
0-Spacing: 4 3 3 5 3 3
1-Spacing: 1 1 1 1 1 1
Pattern: 00001000100000010001000100000001...
0-Spacing: 4 3 5 3 3 5
1-Spacing: 1 1 1 1 1
Pattern: 000001000010000000100001000001...
0-Spacing: 5 4 6 4 4
1-Spacing: 1 1 1
Pattern: 00000001000000100000000010000000...
0-Spacing: 7 6 8 6

Figure 49. Binary gradient sequences & resultant spatial output. Orange units are inputs with value 1.

Mechanics

Physical Mechanism - Blocking Pin Technique

I have outlined the importance of digital logic and why to use the NAND gate as well as the geometry that can be used to represent the NAND functionality. I then described where the information is stored, how to describe surfaces, volumes and circuits as well as the computation provided by such a system. It is now important to outline the specificity of the designed mechanism that allows for the computational functionality and spatial configurations.

As previously described, the optimal geometry for representing NAND functionality is the right-angle tetrahedron due to its four faces and symmetries. This offers two faces for inputs and two possible faces for output. The mechanism was generated by adding and subtracting elements to the primitive tetrahedron as to allow or restrict rotational transformation. The goal of the NAND mechanism is to take two incoming tetrahedron faces as input (either [0] or [1]) and make the appropriate digital logic output (according to the NAND truth table) by allowing or blocking the input of a new tetrahedron upon one of the output faces. This results in the placement of a new tetrahedron upon a completed tetrahedron (meaning two input tetrahedrons have been inserted and the digital gate is able to compute an output) in either an up [1] or down [0] position (relative to the orientation in 3D space). Throughout all of the attempted mechanisms a strategy was used to simplify the NAND truth table to two states: (A) anything that can be input to result in a [1] output and (B) anything that can be input to result in a [0] output. This means that if a face [0] is ever plugged-in, it can only result in one possible output, a [1]. This led directly to the design of the blocking pin technique.

The proposed mechanical design highlights each of the details that allow for the NAND functionality.(See Figure 51) I will now outline each of the elements and their respective roles, either acting as input, output or as a gate. First, I will describe the input/output faces because they perform the majority of the work. The input faces are recessed to receive the [0] or [1] faces of another tetrahedron. The recess signifies its role as input while the output faces extrudes outwards to emphasize the output action. The recess and extrusion also act as a channel to direct and force the incoming/outgoing tetrahedron in the appropriate position. The inner faces of the recess and extrusion are slanted to auto align the unit as it slides in place. These faces also focus the insertion of the input tetrahedrons to perform the blocking action and allow the appropriate output transformation.

The blocking pin is located on the tip of the [0] output face. The pin is an extruded rectangle that intersects the triangular output face. When the [0] face is inserted into an input face of an adjacent tetrahedron (acting as a Gate) it slides through a female slot. These female slots are located on both tips of the input faces and were created with a Boolean operation from an extruded rectangle that intersected the input face (the exact dimensions of the male blocking pin). The height of the male blocking pin is designed specifically to slide through the female slot of the adjacent tetrahedron (Gate) and terminates approximately 0.25" past the output face on the other side. This extended portion acts to restrict the placement of the next tetrahedron and acts as the essential element for the NAND functionality.

The extension of the male blocking pin forms a right-angle extrusion on the output face. This extrusion restricts the placement of the newly placed tetrahedron upon the output face of the gate unit. The right angle piece corresponds to a right angle Boolean on one of the input faces. This Booleaned portion fits snuggly onto the right angle of the male blocking pin. This allows the placement of the new unit in an upward position (or left/right position depending on the orientation of the unit in 3D space). If the new unit was attempted to be placed in the downward position it would be blocked by the right angle piece because it does not have a corresponding Boolean slot. This effectively means that any time a [0] output face is inserted into an adjacent tetrahedron that is acting as a gate it will only allow an upward placement on the [1] output face. Likewise if there are no [0] output faces (meaning there are two [1] faces) inserted into a gate tetrahedron then it must go the opposite direction and result in a [0] placement. This corresponds to the NAND truth table and demonstrates its complete functionality.

The next details of the designed NAND mechanism are the graphical annotations to identify input/output faces as well as the orientation required. There are two graphical elements utilized for easing the user's operation. First, to identify the output faces, each is labeled with a "0" and "1" number, respectively. This easily allows the user to decide which output face should be inserted into an adjacent tetrahedron. Any two sequence combination of [0] and [1] can be input simultaneously to produce one of two output configurations/transformations. The second graphic label is a triangle placed on one of the input faces to identify the orientation of an incoming tetrahedron after a NAND operation has been completed. For example, when two faces ([0] and [0]) are input into a tetrahedron acting as a gate it will result in a [1] output. This forces the next tetrahedron to be placed in the [1] position through the male blocking pin. However, if the required orientation was not identified then the newly placed tetrahedron could go in 1 of 2 orientations and thus would not be consistent throughout the assembly. This triangle graphic instructs the user to always place the new tetrahedron with its orientation such that the triangle points towards the male blocking pin of previous gate unit.

The final elements of the NAND mechanism are the positive and negative detents that allow for a snug, snap-in-place, fit between tetrahedrons. There are two male detents located on the output faces ([0] and [1]) on each unit. There are corresponding female detents located on the input faces of each unit. Similarly there are two larger male detents located on these input faces that will lock into place on the female Booleaned slots that sometime receive the male blocking pin. This is to ensure a tight fight when no blocking pin (no [0] faces) have been inserted. All of the detents and tapered faces help to auto-align the pieces, minimizing error propagation and accumulated tolerances. The detents also embody the fasteners that connect units and allow a strong global configuration.

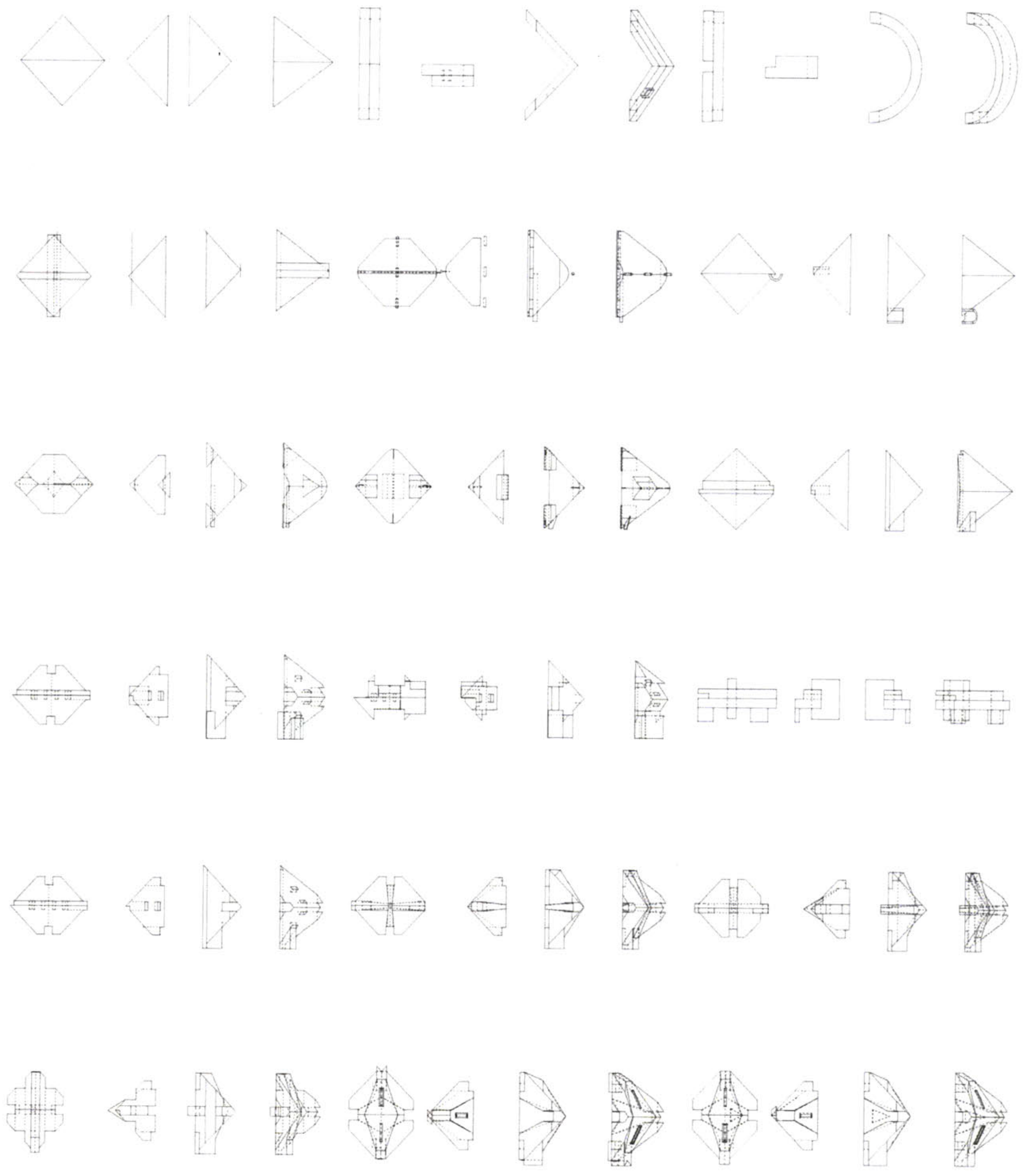


Figure 50. NAND mechanism design iterations.

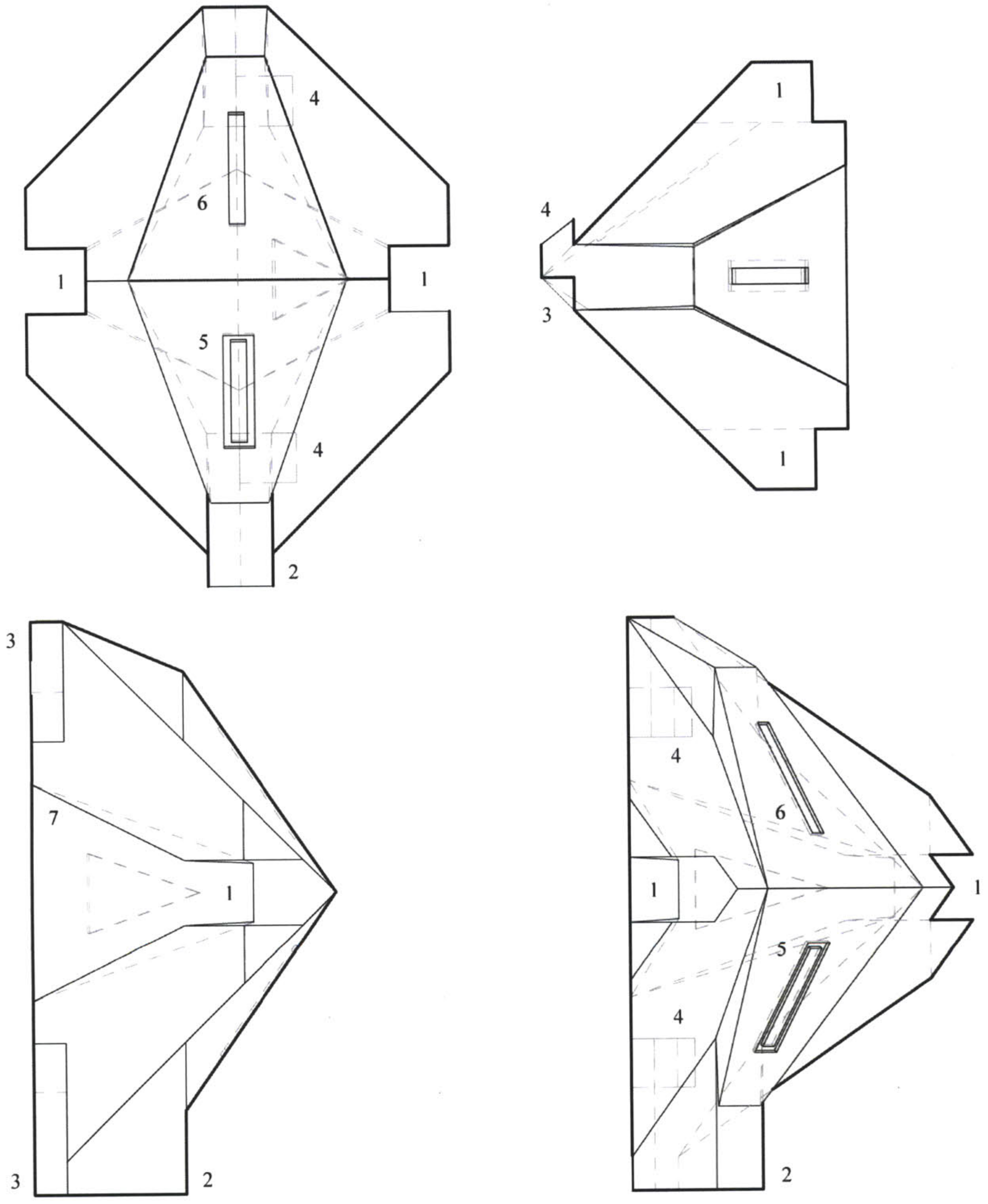


Figure 51. 1. Female slots to receive (2) male peg 2. Male peg on negative face (5) 3. Half notch to allow positive output and receive male peg (2) 4. Half clip to allow negative output - only when no male pegs (2) exist 5. Negative Face (Output) 6. Positive Face (Output) 7. Input face

Prototype

Fabricating the Prototype

In previous section I described the physical qualities of the NAND mechanism and how they enable the functionality of digital logic to be discretely passed between elements. I will elaborate on this functionality by showing a physical prototype, the fabrication process for creating the prototype as well as the aggregated potentials of the overall assembly. First, I should explain the details for fabricating physical NAND mechanisms.

While developing the numerous iterations of the NAND mechanism units were tested through a variety of 3D printing technologies, making it possible to iteratively improve and test new latches, detents and other mechanical elements for the NAND functionality.(See Figure 51) The current version was eventually reached through tolerance improvements and better latching mechanisms, leading to the final prints on an InVision™ 3-D printing machine. This machine outputs plastic parts with a wax based support material. This machine has a high level of precision and gives sharp edges and a hard plastic solid object. This printer was used specifically for the precision, strength and waxy finish, all beneficial properties for the next step in fabrication, molding. After cleaning the printed part, it was suspended inside of a plastic box that was roughly 1/2" larger in all dimensions. (See Figure 53) This plastic box was used as the shell that houses a rubber mold. Once the printed piece was secured with a 3/8" dowel, suspending it inside the middle of the plastic box, liquid rubber was poured into the box, surrounding the printed mechanism. The rubber was poured to the exact mid-way line of the printed part and left to dry overnight. Another layer of liquid rubber was then poured on top of the dried first half. Once fully dried this allowed for the two rubber parts to be separated from one another easily. A single line was cut along the vertical seam of the rubber to easily release the print from the two part rubber mold. After removing the print, the two part rubber mold contained the exact impression of the print with a high degree of precision. The 3/8" dowel was also removed to reveal a 3/8" hole in the rubber that was later used as the pour spout for liquid plastic.

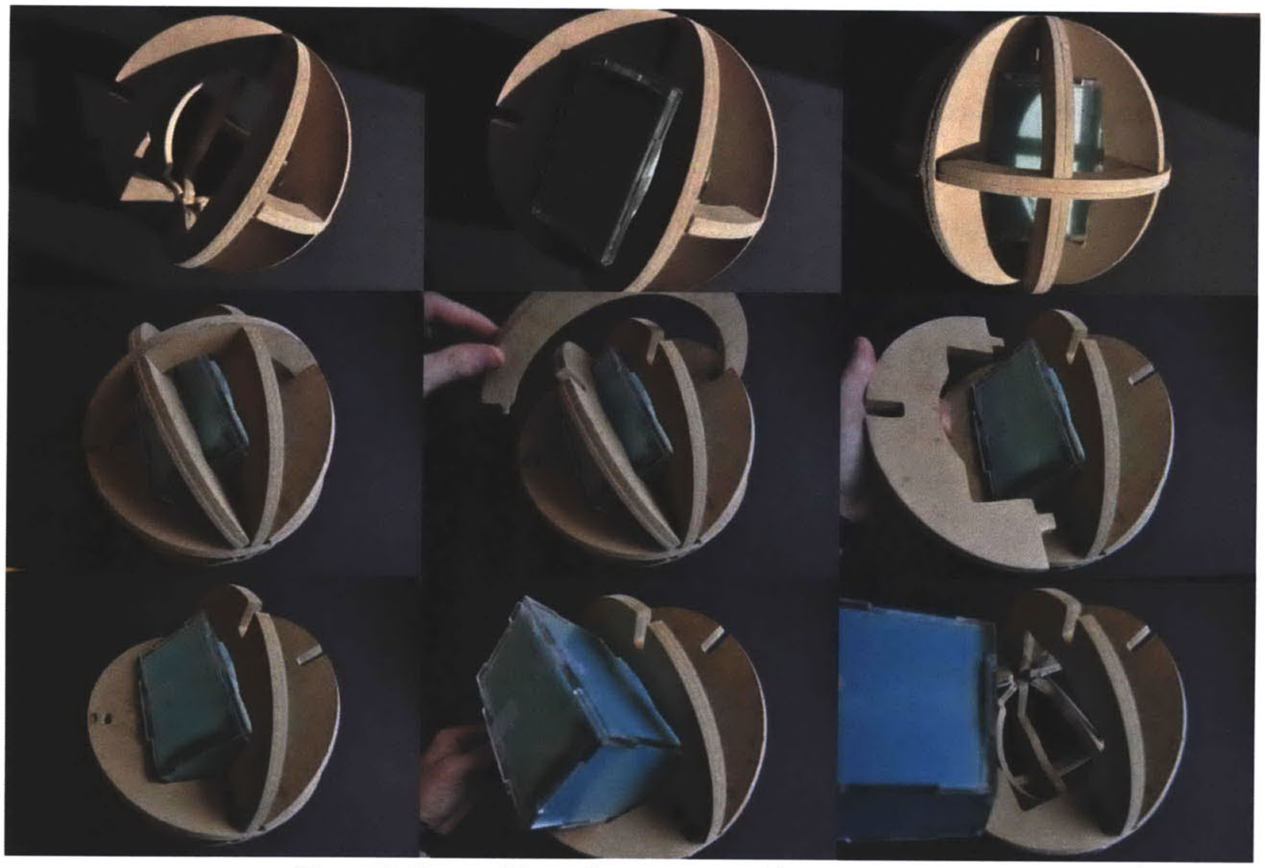


Figure 52. Roto molding sphere constructed from MDF wood.

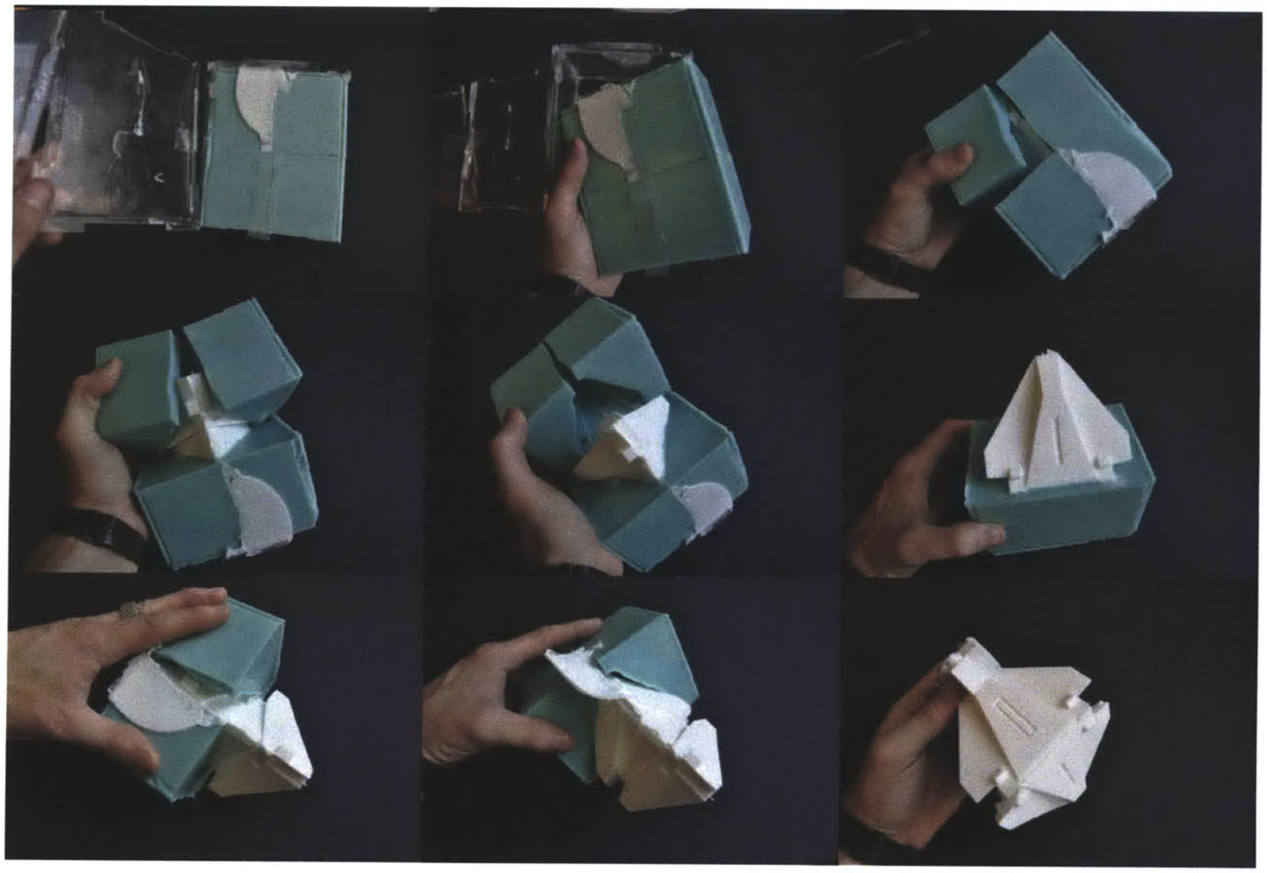


Figure 53. Releasing the two part rubber mold after spinning it in the roto molding sphere.

The two part rubber mold was then placed back together and inserted into the plastic bounding box. 80 grams of liquid plastic was poured into the rubber mold through the 3/8" spout. A rubber cap was placed into the spout and the mold and surrounding plastic box were rotated repeatedly in two directions. The rotation allows the liquid plastic to coat all of the mold's negative spaces. This process is repeated continually for 10 minutes then opened to reveal a hardened plastic positive of the NAND mechanism.

In order to relieve the rotating process, a sphere was constructed to be rolled on the floor and similarly spin the mold in two directions.(See Figure 52) The sphere was constructed from 1/2" MDF wood circular sections that are notched and snap-together. Two of the half-circular sections are not permanently connected to the rest of the sphere and allow access to insert the mold. After plugging the rubber spout, the plastic box was quickly inserted into the wooden sphere, the two half-circular sections snapped in place and the ball was rolled randomly on the ground for 10 minutes. The rolling process greatly reduced the amount of energy required to rotate the somewhat heavy mold in multiple directions for any length of time.

After 10 minutes passed the sphere was opened and the mold removed. The positive plastic part was removed, still warm from the chemical reaction of the plastic, and let cooled for a few minutes. A quick cleaning was performed with a knife to remove any plastic burrs. The final piece contains every element of the printed unit, with high precision and crisp edges.(See Figure 57) A colored die was used to darken half of the units to a charcoal grey color, signifying the functionality of the units as they are assembled. The grey color signifies the mechanisms that are acting as Gates and the white identifies the Input mechanisms. Next I will explain the overall assembly and demonstrate the functionality of the NAND mechanism and physical prototype.

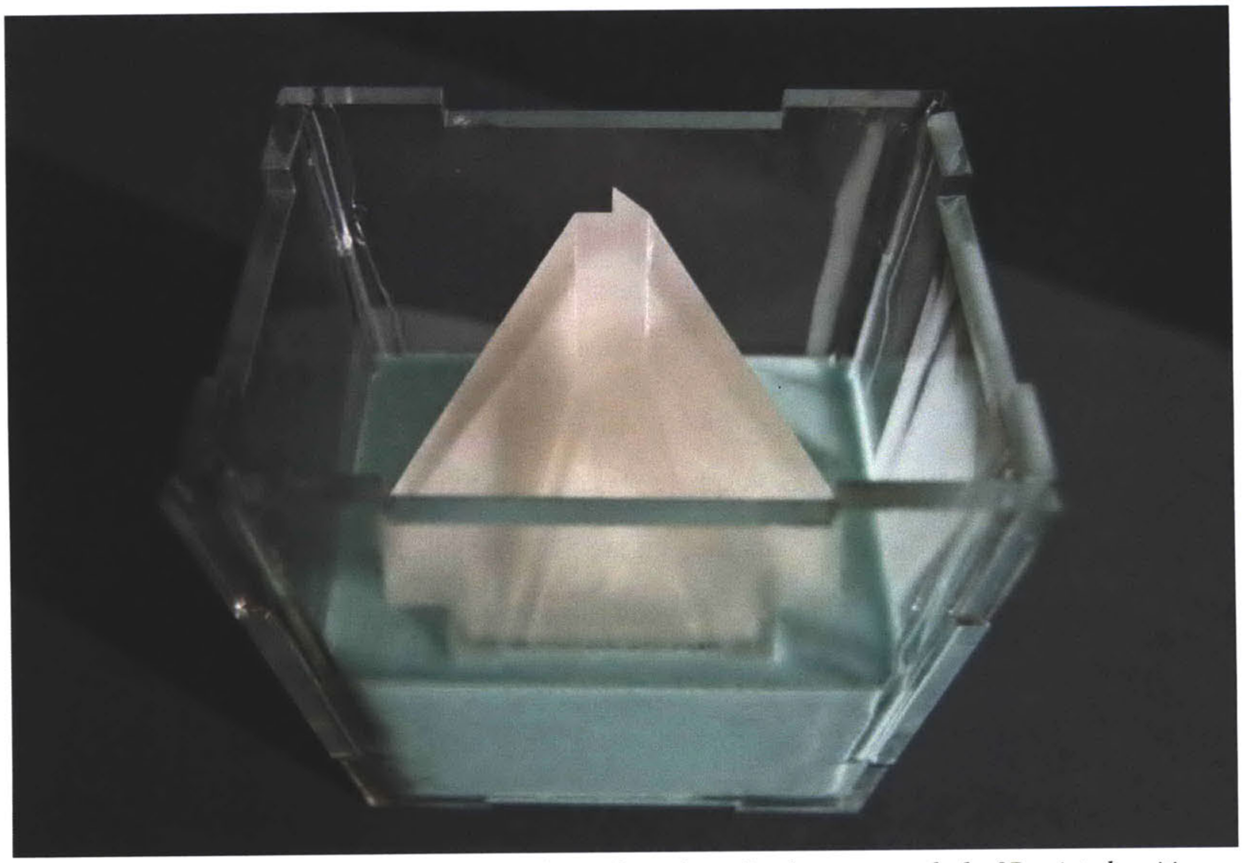


Figure 54. First half of rubber mold, poured into plastic bounding box, surrounds the 3D printed positive.

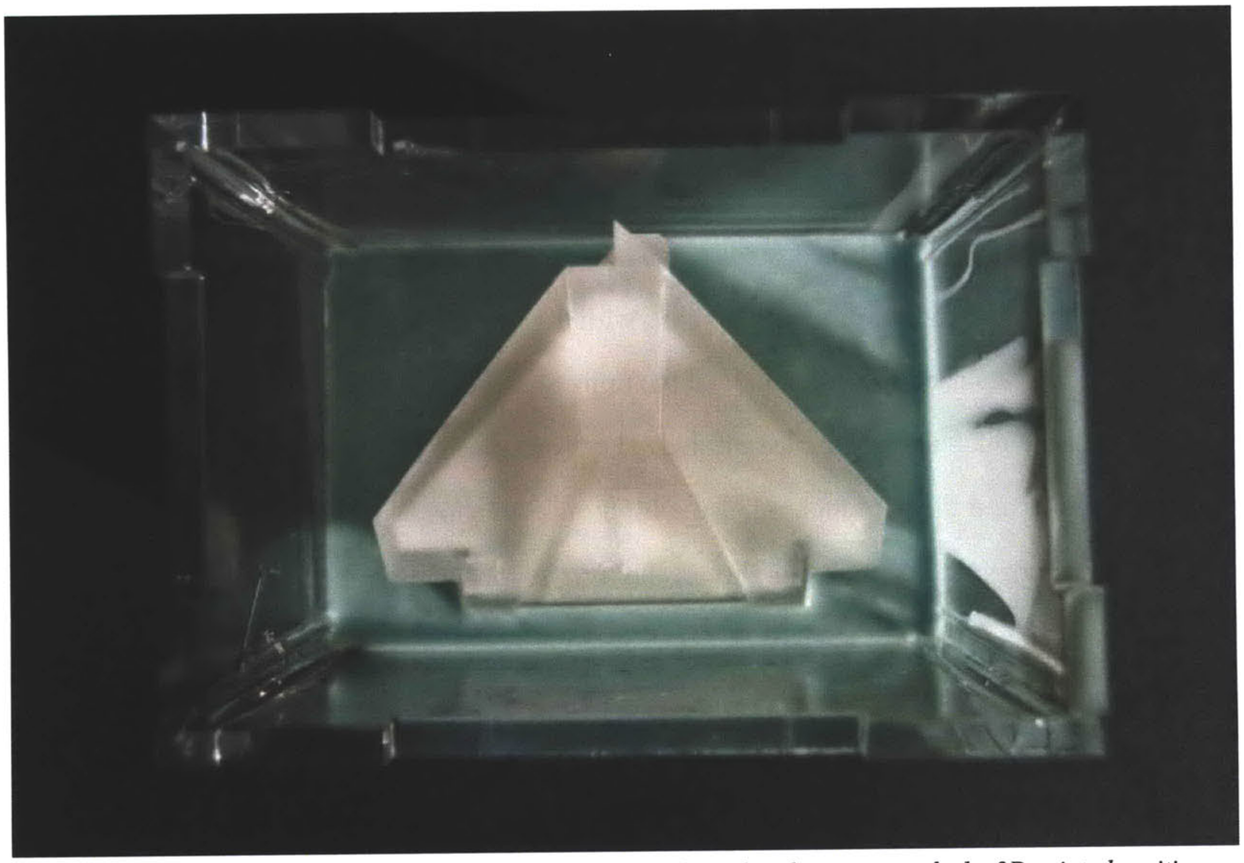


Figure 55. First half of rubber mold, poured into plastic bounding box, surrounds the 3D printed positive.

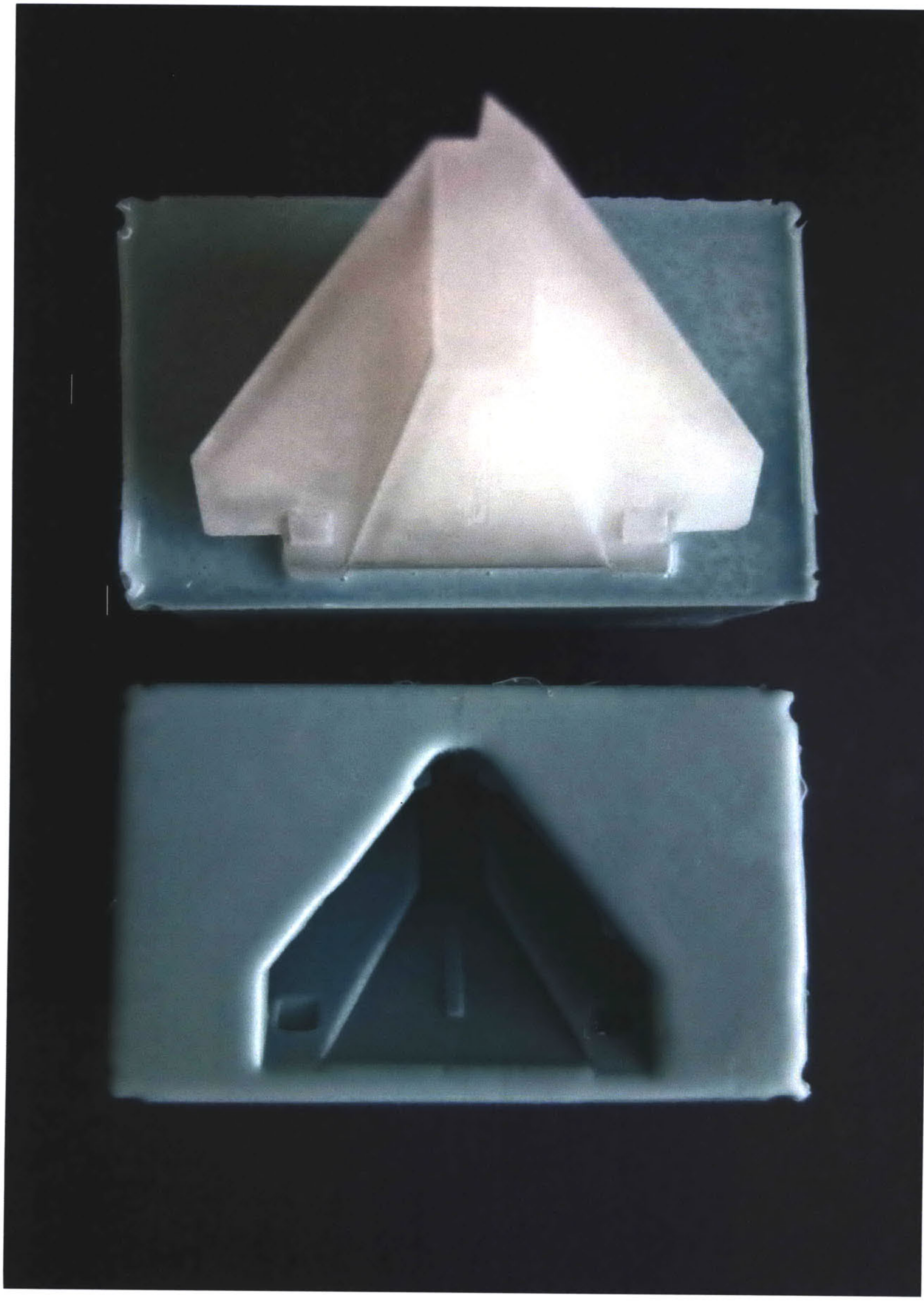


Figure 56. Two part rubber mold and 3D printed positive.

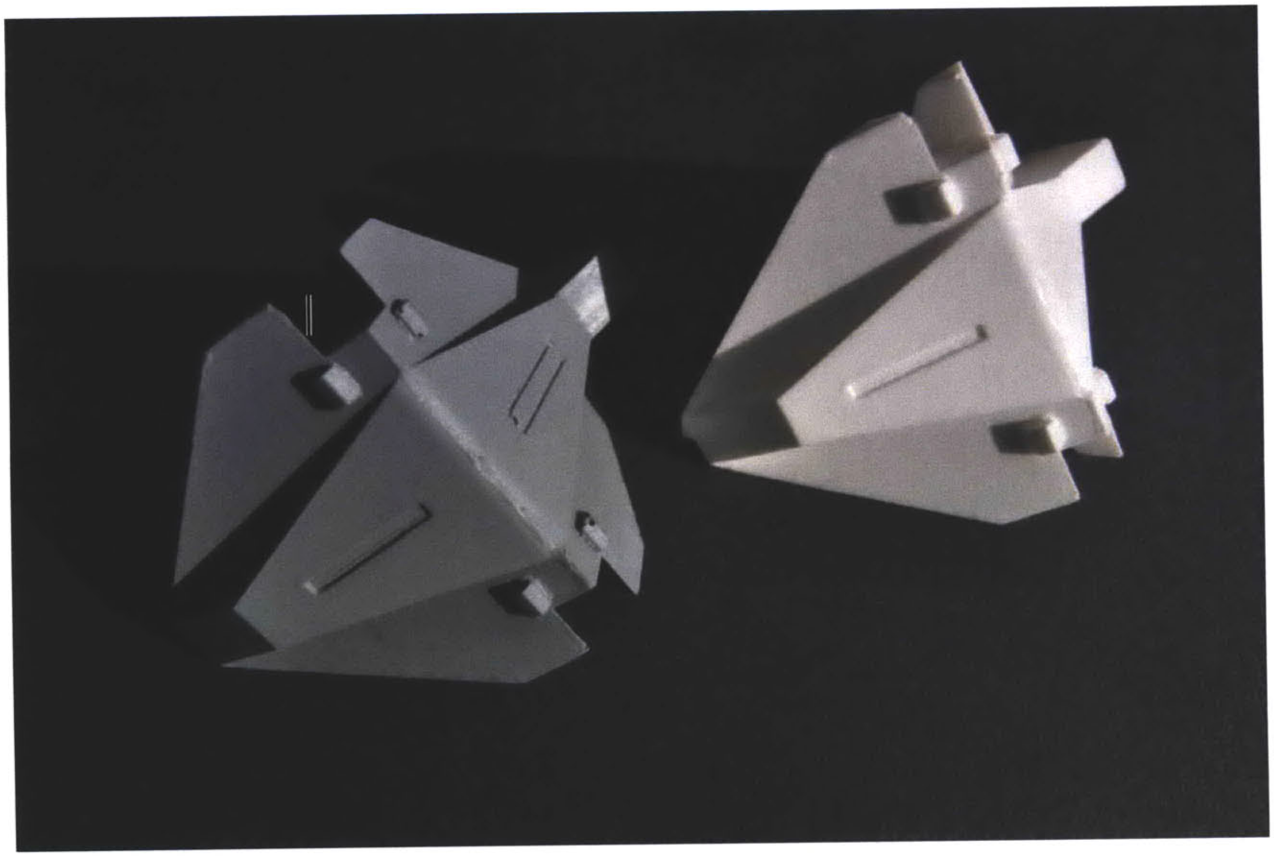


Figure 57. Grey and white plastic positives after roto molding.

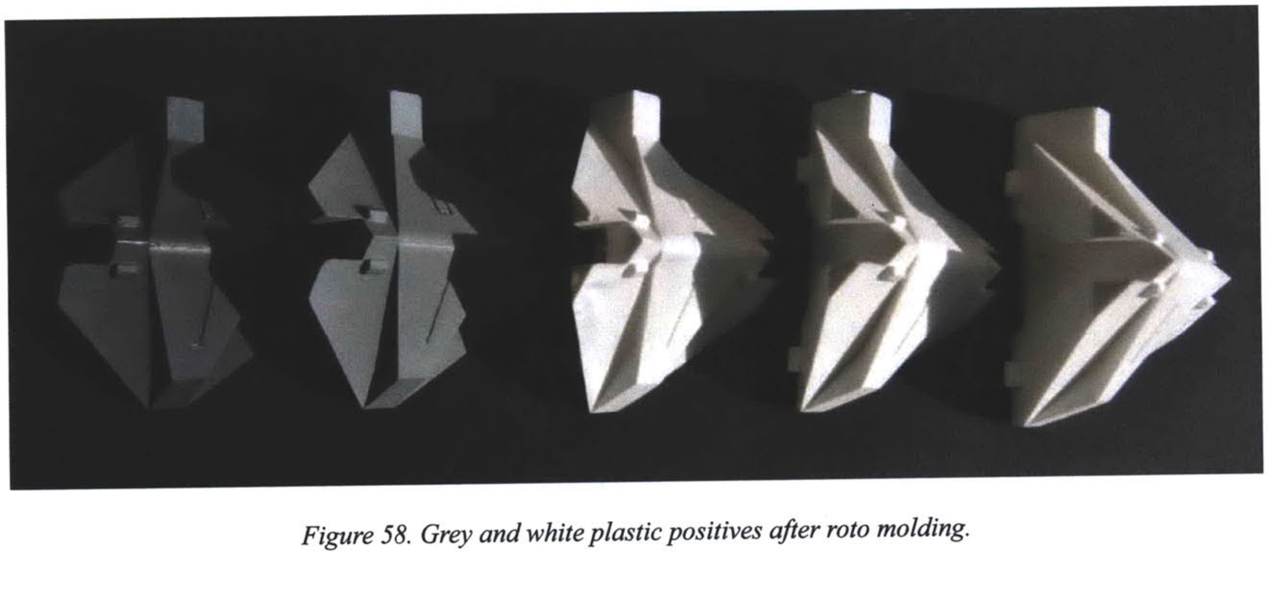


Figure 58. Grey and white plastic positives after roto molding.

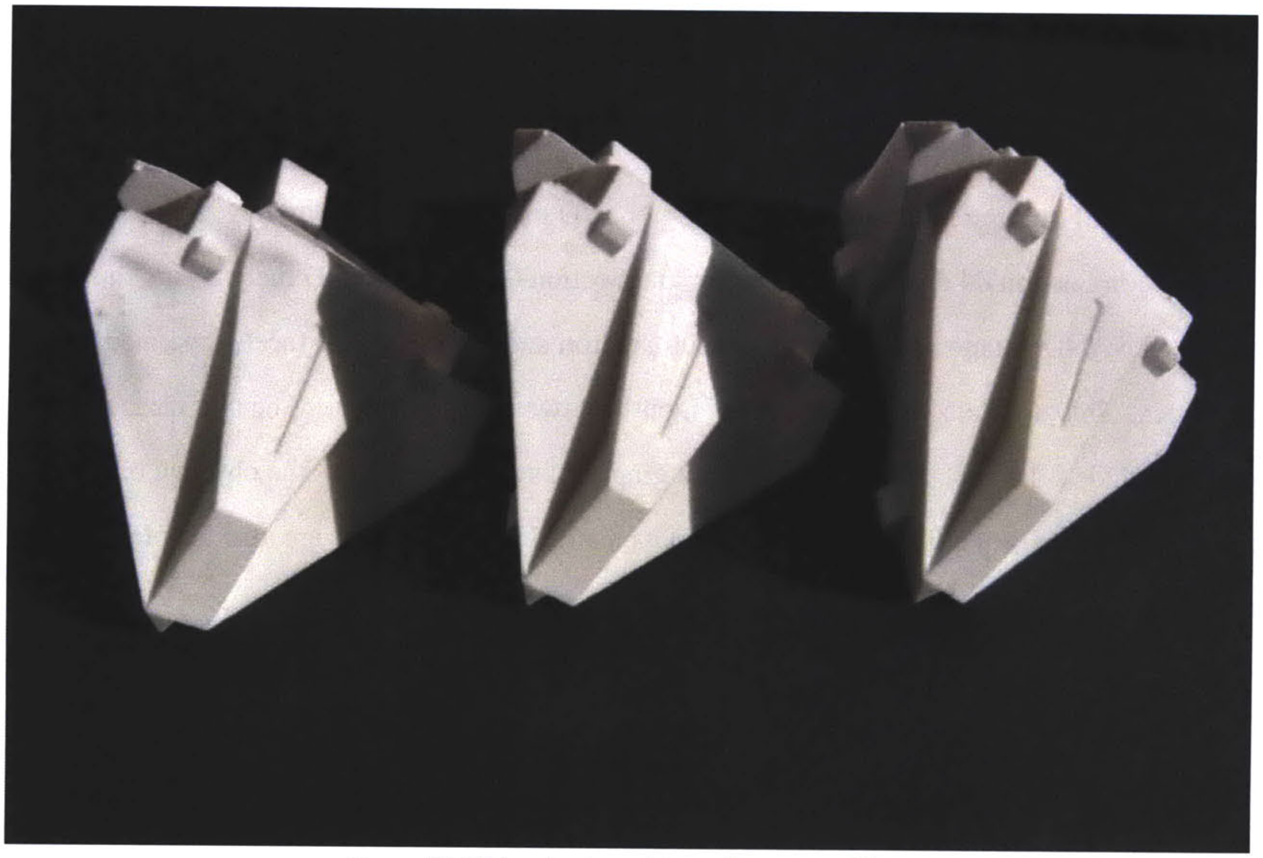


Figure 59. White plastic positives after roto molding.

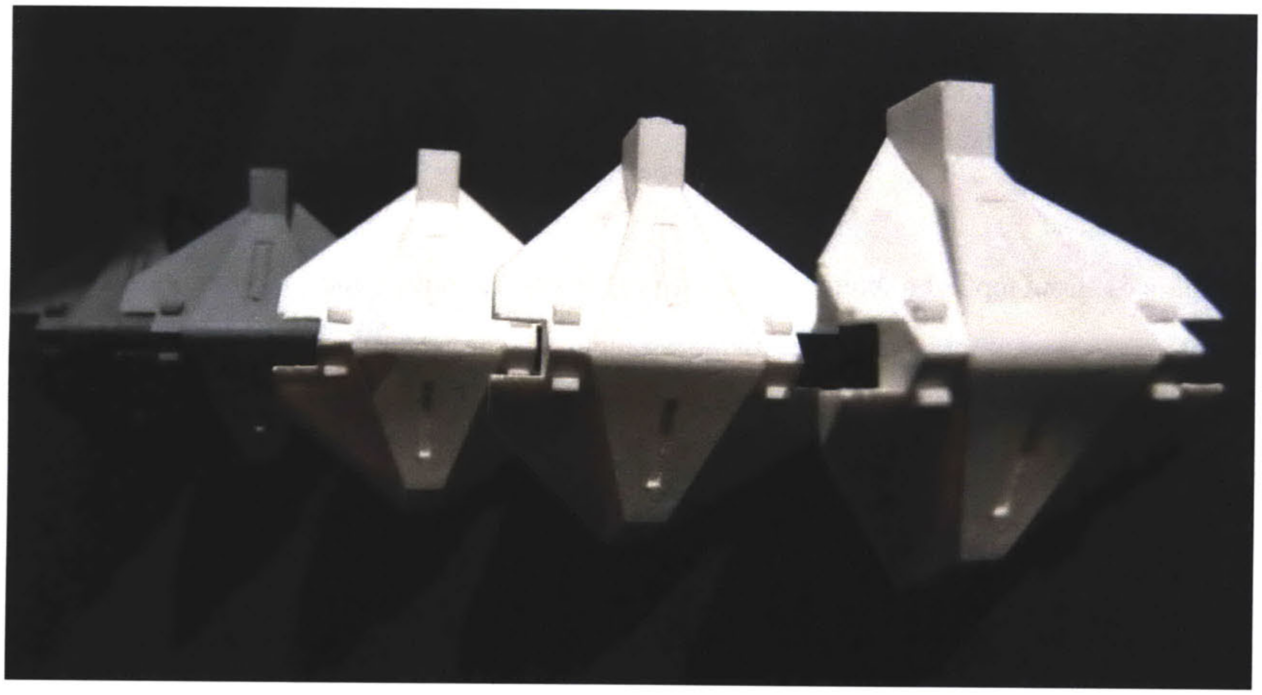


Figure 60. Grey and white plastic positives after roto molding.

Programming the Mechanism

The process of programming the *Logic Matter*, NAND mechanisms, is relatively simple. As explained in Section 04 on NAND geometry and Section 05 on the NAND mechanism, the proposed NAND geometry is a right-angle tetrahedron and has four equal faces, separated into two sides. This translates into two faces of input and two faces of output. The two faces of output include [0] and [1] indicated both by a graphical symbol as well as the physical blocking pin feature.(See Section 05 on the NAND mechanism) The output faces are also denoted by extruding outward from the unit, creating a male triangle.(See Figure 51) Conversely, the female faces are denoted with a negative impression the unit, or a female triangle. The mechanisms are "programmed" by the user placing an output face of one unit into the input face of another unit. This directly indicates that the receiving tetrahedron is acting as a Gate and has just received a [0] or [1] as input. The user then places a second tetrahedron output face [0] or [1] into the same Gate, on the open input slot. The Gate has now been fully programmed and will output the resulting NAND logic.(See Figure 09) The following diagrams; Figure 61 and Figure 62,show the only two possible output configurations from two different input configurations (out of 4 possible configurations).

In Figure 61, from top left to bottom right, the progression from two inputs [0] and [0] to the output [1] is shown through a physical demonstration. The process stars by placing a single [0] face into the Gate mechanism. The [0] face contains the blocking pin mechanism that has been inserted into the female slot of the Gate unit. A second tetrahedron's [0] face is inserted into the Gate, similarly with the blocking pin mechanism sliding into the female slot. The Gate now contains two NAND units with two blocking pins. The two blocking pins dictate the only possible output configuration to be towards the [1] face (in Figure 61 this is denoted in the last image where the tetrahedron is place in the upward position). The [1] output is expressed when the next tetrahedron is attempted to be placed and will only fit in one position, the [1] position. The next example will demonstrate inputs [1] and [1] with an output of [0].

In Figure 62, from top left to bottom right, the progression from two inputs [1] and [1] to the output [0] is shown through a physical demonstration. The user places a single [1] output face into a Gate unit. Likewise, a second tetrahedron and [1] output face are inserted into the same Gate unit. The [1] output faces do not contain the blocking pin mechanism.(See Figure 51) After the Gate has received both inputs, [1] and [1], it has now been programmed and is ready to execute the output. The output is executed by the user placing the next tetrahedron in the only possible position [0] (in Figure 62 this is denoted in the last image where the tetrahedron is placed in the downward position). This successfully demonstrates two of the possible four NAND input/output configurations. The remaining two inputs can be [0],[1] and [1],[0] which will act in the exact same manor as the [0],[0] sequence that was previously demonstrated. The blocking pin mechanism of the [0] face essentially ensure that any instance that contains at least one [0] face will force the output configuration to turn toward the [1] face. This is useful because it means that the NAND mechanism will perform exactly as the truth table dictates; (NOT AND) anything other than two [1] inputs will result in a [1] output, or anything with at least one [0] input will result in a [1] output.(See Figure 09)

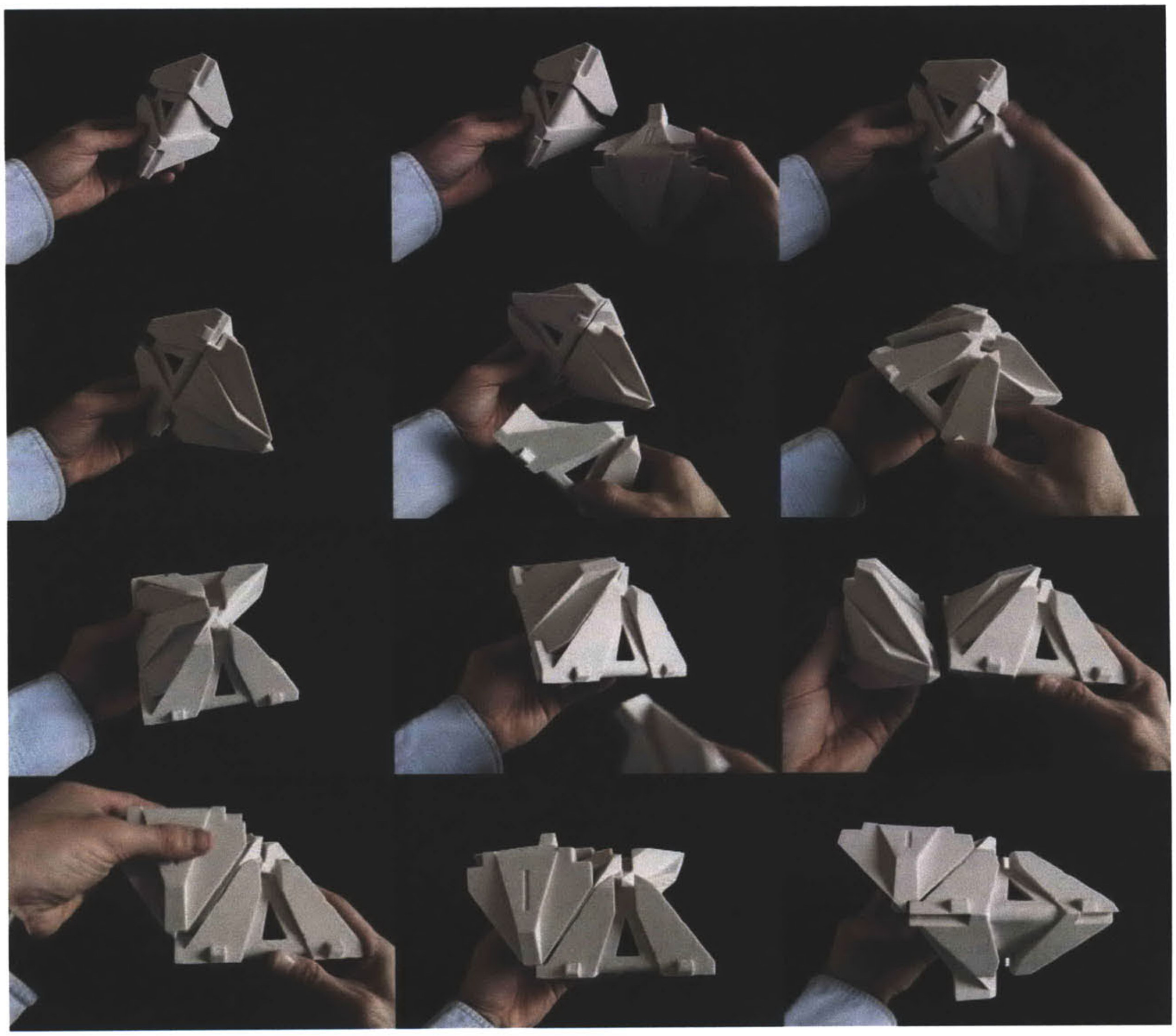


Figure 61. 4 Logic Matter modules demonstrating programmability of base configuration [0,0] = 1. The final configuration demonstrates a 180 degree vertical rotational difference from output = 0.

$$[0,0] = 1$$

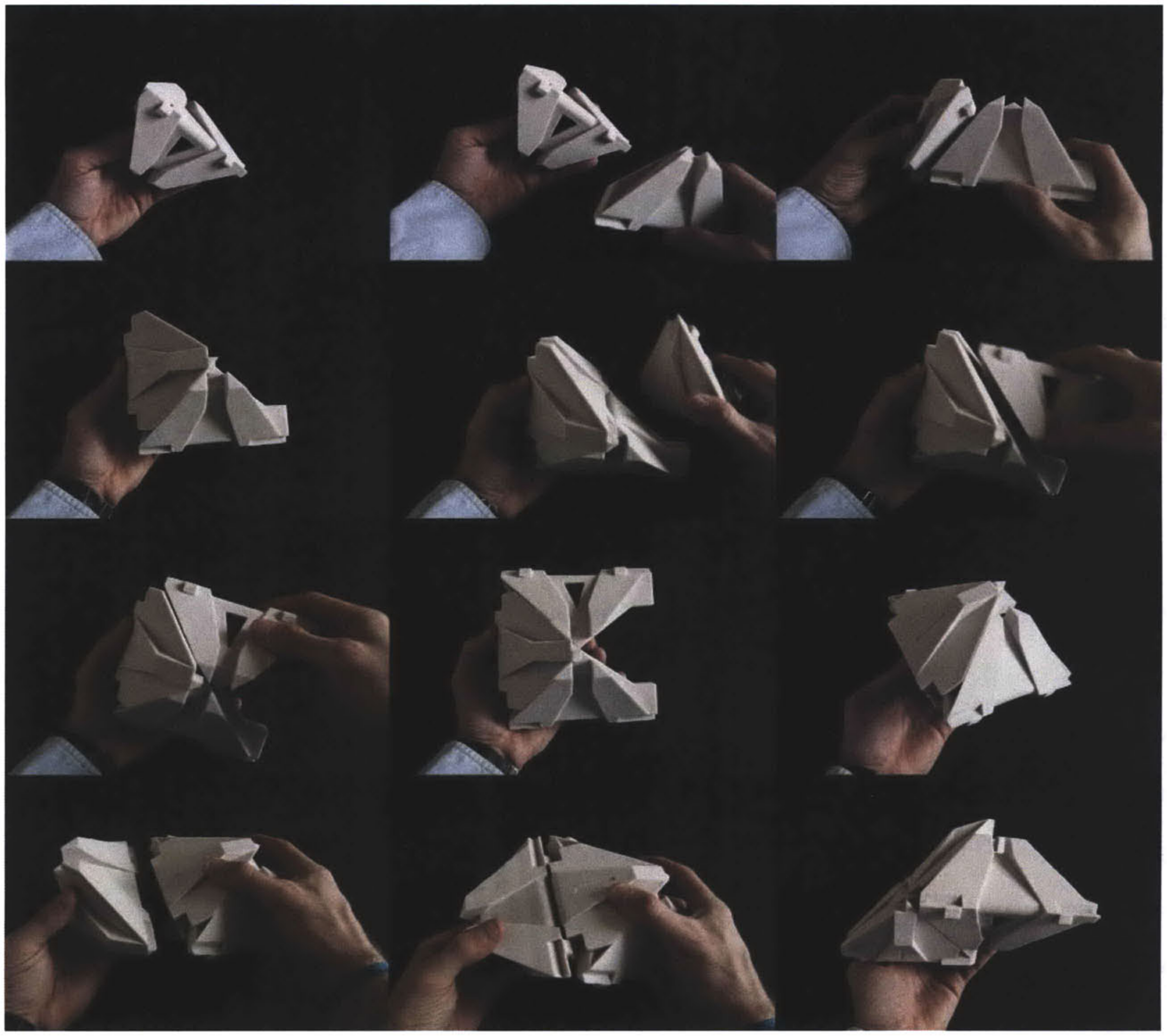


Figure 62. 4 Logic Matter modules demonstrating programmability of base configuration [1,1] = 0. The final configuration demonstrates a 180 degree vertical rotational difference from output = 1.

$$[1,1] = 0$$

Overall Assembly

After being able to successfully program a NAND mechanism, it is important to show a larger assembly of NAND mechanisms, demonstrating a number of useful qualities of *Logic Matter* in terms of geometric descriptions and computation. In this section I will describe a working prototype made from sixty NAND mechanisms, explain the process of assembly and finally, show important characteristics of computing through assemblies. First, let us look at the process of assembly and the individual inputs to create the overall configuration.

The prototype assembly followed a repeating pattern of input rather than the gradient of inputs (0 or 1) as explained in previous sections.(See Figure 48) (See Figure 49) The repeating pattern was decided upon because of its simplicity and the desired final configuration. The overall configuration is a single path that turns in all three dimensions while spanning +/- 18" in an arch-like path. The repeating pattern of inputs consisted of seven 0's followed by seven 1's repeated twice. There are four sections of the prototype each made through on of the seven bit input strings (either seven 0's or seven 1's). Each one of the sections is a straight line, however when they connect they create a three dimensional single line. The key aspect of the input sequence is the repeated seven bit pattern. This seven bit pattern emphasizes the importance of local knowledge, as explained in the previous sections.(See Section 04 on Computing Through Construction) At any given point in the construction the user only needs to know seven bits of information. Depending on their place within those seven bits they are informed of their next move. If the user is at the start of the entire sequence then they know that they have seven more 0's to place. At the next step the user knows that they have placed one 0 input and need to continue for six more 0's. At the end of the seven bit sequence they know that they need to flip the bit and start inputting seven 1's. If this pattern continued indefinitely the user would only be required to know the seven bits of information and could eliminate any global knowledge. This would ensure that the previous sections had been made correctly and that the material informs them of the next steps. Unfortunately, the repeating seven bit sequence does not demonstrate the stopping capabilities, as described in Section 05 on Computing Through Construction, however we can assume the user was told the number of steps to repeat. Ultimately, the input sequence should inform the user of the stopping point, thus reducing a further element of information that the user is required to retain. For this prototype this was not shown for reasons of simplicity.

Another important aspect of the working prototype is the material's ability to store the overall input sequence, or material as a storage device. The user repeatedly places an input unit (redundant white tetrahedrons) to complete the NAND mechanism at each step, thus dictating the next orientation according to the NAND functionality and the recorded inputs. The redundant inputs remain in place, acting as redundant structure, mechanism to allow failed units to be bypassed as well as a recording of the construction sequence. The information storage can be extremely useful in terms of unit failure because each failed unit can easily be replaced without the original construction information (blueprints), knowing that there is only one possible face and orientation that will fit in-place.(See Section 05 on Computing Through Construction) In the event of node failure, the surrounding units dictate the input/output relationships that define the only possible replacement unit. The storage of assembly information is also vital for complex construction projects that take several working periods. The assemblers can stop and start at will without losing their place or having to remeasure the overall. The previous units and inputs dictate the local positions and the next steps. Finally, the material storage capacity could allow an exact copy of the working prototype to be built without the initial sequence. A human or machine could directly read the first assembly and make a perfect copy, simply by reading and writing step-by-step from each of the white input mechanism. This shows promising avenues for mass-produced assembly by simplifying the construction process and the information that is required to be stored in the machines or humans that assemble the structures.

Finally, the working prototype demonstrates the single stream construction of instructions and data for computing. The single stream of input takes the form of the repeating seven 0's and seven 1's. This in turn, because of the properties of the NAND mechanism, allow computing to be performed on the string, while accruing multiple levels of digital gates based on the pattern of inputs. If we look at a few steps of input it should help to emphasize the point. If the user places two 1 inputs into a grey Gate unit, the output results in a 0 passed into a new grey Gate unit. This successfully demonstrates the NAND input/output functionality. Next, the user passes another 1 into the open slot of the latest grey Gate. This results in a 1 output, similarly functioning as a NAND mechanism. Both steps resulted in a functioning NAND mechanism, however the global configuration acted as either an AND or and OR Gate because the initial inputs were [1,1] and the final output, after two steps, was a [1]. This shows that a number of NAND mechanism can act as different gates and that the computing devices are actually configured directly as the input string is passed, unlike the two strings, instruction and data, that we saw in the Field Programmable Gate Arrays and common to most computer architectures today.(See Figure 43)[64]

64 Adrian Thompson. An evolved circuit intrinsic in silicon entwined with physics.

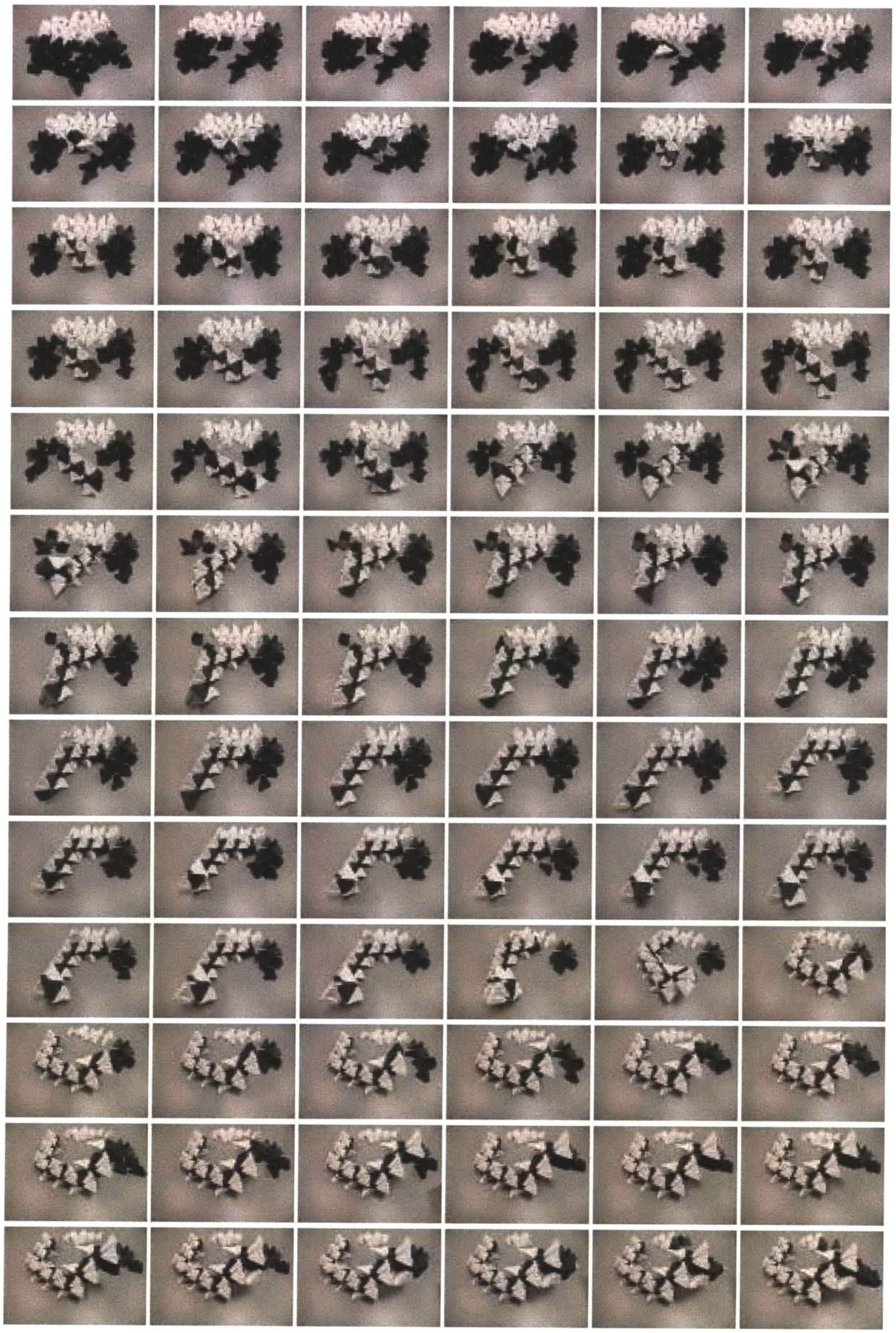


Figure 63. Assembly animation from 60 individual units to final global configuration.

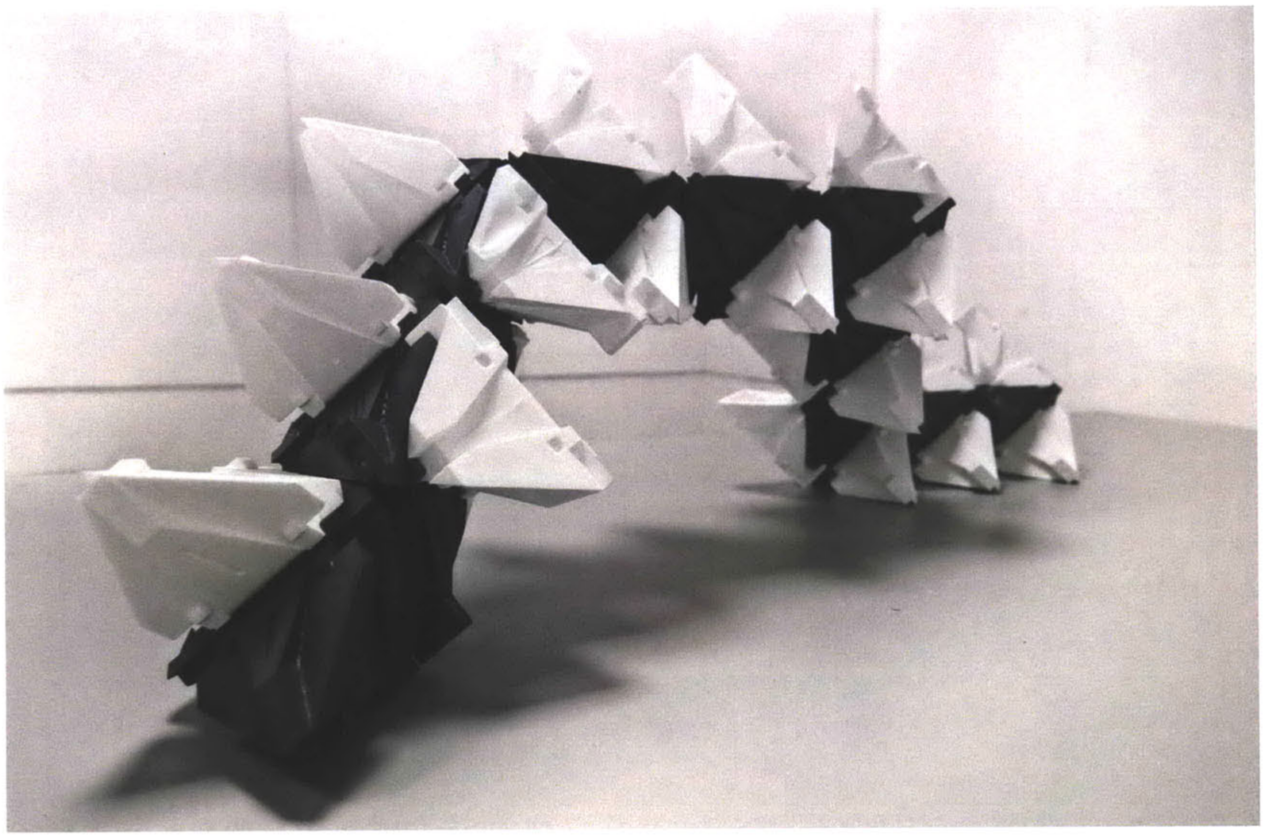


Figure 64. 60 unit working prototype demonstrating three dimensional single path.

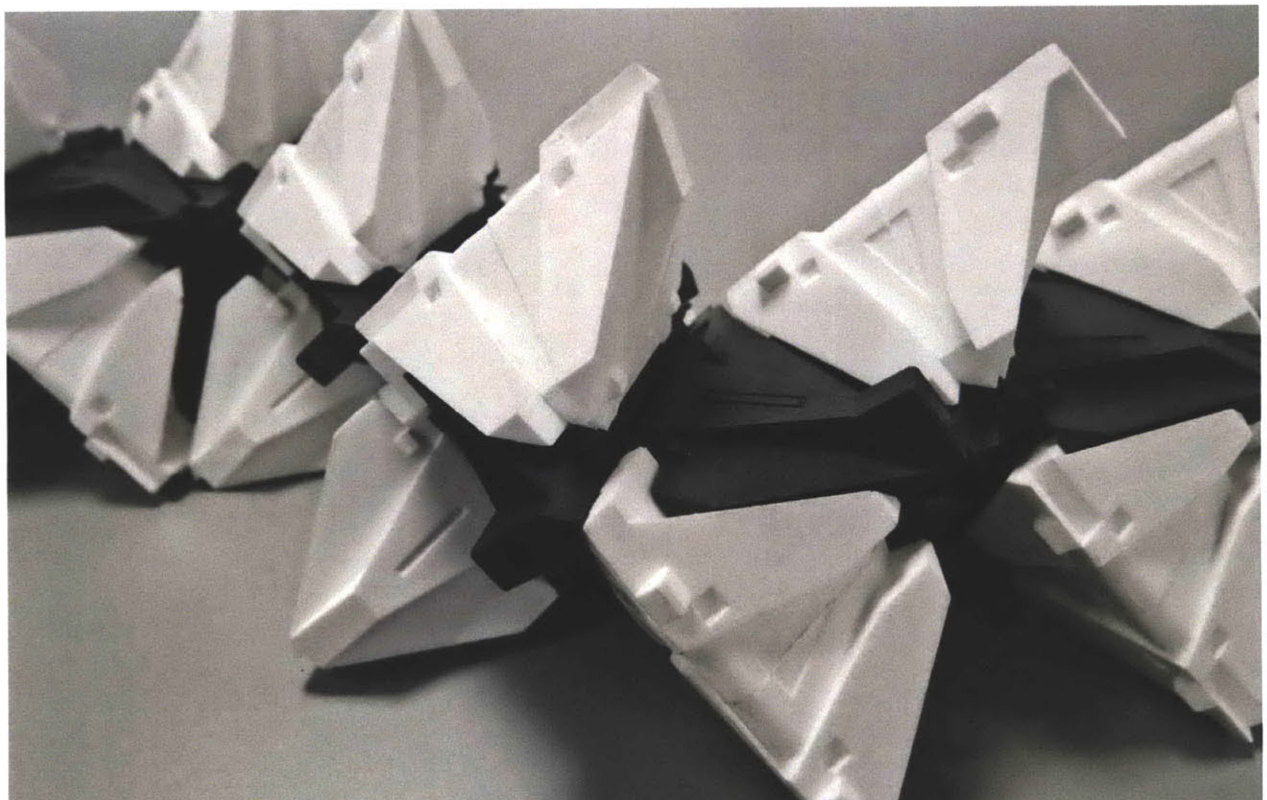


Figure 65. Prototype detail with white redundant tetrahedrons as input and grey gate tetrahedrons.

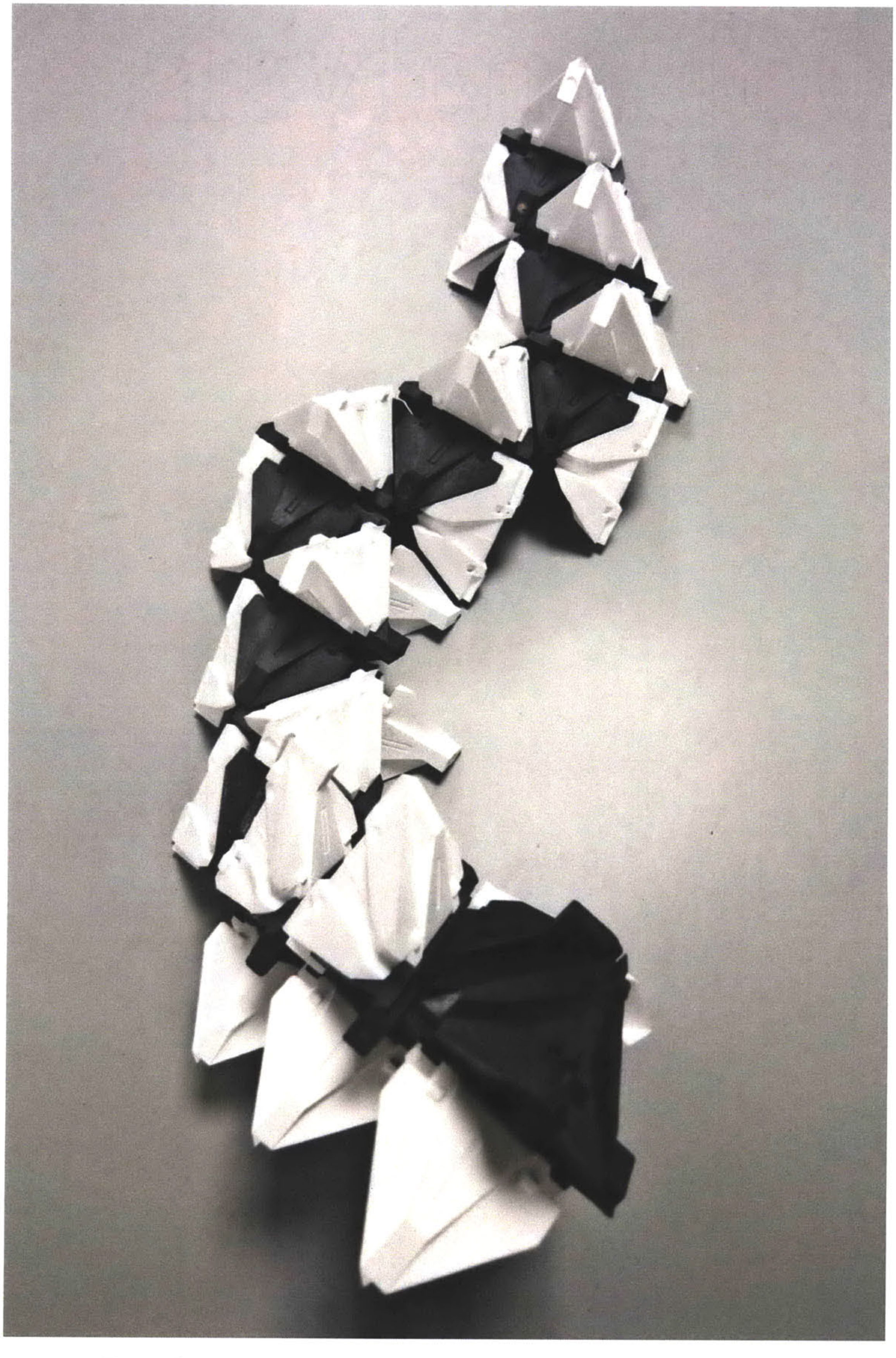


Figure 66. 60 unit working prototype demonstrating three dimensional single path.

[06] Analysis

Computation

Logic Matter can be analyzed in terms of its potential for computation as well as it ability to describe geometry and function as a physical assembly system. In this section I will attempt to decompose the potentials of *Logic Matter* as a system of computation by relating it to a number of important examples including; the abacus, Wang Tiles, jigsaw puzzles and DNA computing.

The abacus is one of the earliest examples of a physical aid used for making calculations.[65] The human actually performs calculations in their head while sliding the different beads of the abacus to remember the state, sum and any carried numbers.[66] The abacus can be used to calculate addition, subtraction, multiplication, division, square roots and cube roots of any number.[67] This device attempts to minimize the amount of information the user is required to remember while maximizing the amount to be physically computed. The human remembers a simple algorithm but can achieve far quicker and greater calculations than if they were left alone. *Logic Matter* relates to this example as a device to store information, process the input/output of binary information through digital logic gates and interact with the user, informing them of the resultant next moves. *Logic Matter* goes further than simple calculations as it can actually compute logic operations based on binary input/output and can be used as combinational and sequential logic to form complex logical circuits. Similar to the abacus, *Logic Matter* could be used a simple device to work hand-in-hand with the user, computing simultaneously as elements are placed, restricting moves and allowing only the proper computed output, then waiting again for the user's input. The benefits of the device lie in the sequence of inputs that are fed into the system, and the operations that are performed on the inputs, dictating the next inputs, storing the state of the system and processing future decisions. Similar to modern computing devices, *Logic Matter* utilizes a series of instructions and data sequences, however, uniquely, *Logic Matter* only requires a single strand of inputs to build the instruction hardware (sequence of NOT, AND,

65 A Brief History of the Abacus. http://www.ee.ryerson.ca/~elf/abacus/history.html.
66 A Brief History of the Abacus. http://www.ee.ryerson.ca/~elf/abacus/history.html.
67 Kei-Chen, Lee. HOW TO LEARN LEE'S ABACUS. Lee's Abacus Correspondence School. (Taiwan, China, 1958).

OR, NAND, XOR, XNOR gates) and process the data (sequences of 0's and 1's). This could potentially lead to faster computing and fewer errors while fetching data and instructions.

The next model of computing that can be compared with *Logic Matter* is Wang Tiles. Wang Tiles were introduced by Wang in 1965 as a series of colored edge tiles that can be tiled only in accordance to their neighbor's colored edges.[68] The question was asked, how many tiles and of what color combinations are required to perfectly tile an infinite plane.[69] Wang tiles were initially demonstrated to be solvable through only periodic tiling, and then later were extended to include certain aperiodic tiling, or non-repeating patterns.[70] Aperiodic tiling has sparked interested in a number of people due to their inherent link to applications in DNA computing.[71] Wang tiles were further demonstrated to be Turing complete, or simulate a single tape Turing machine.[72] Being Turing complete ensures that rules followed in sequence from an arbitrary sequence of inputs can produce the output of any calculation.[73]

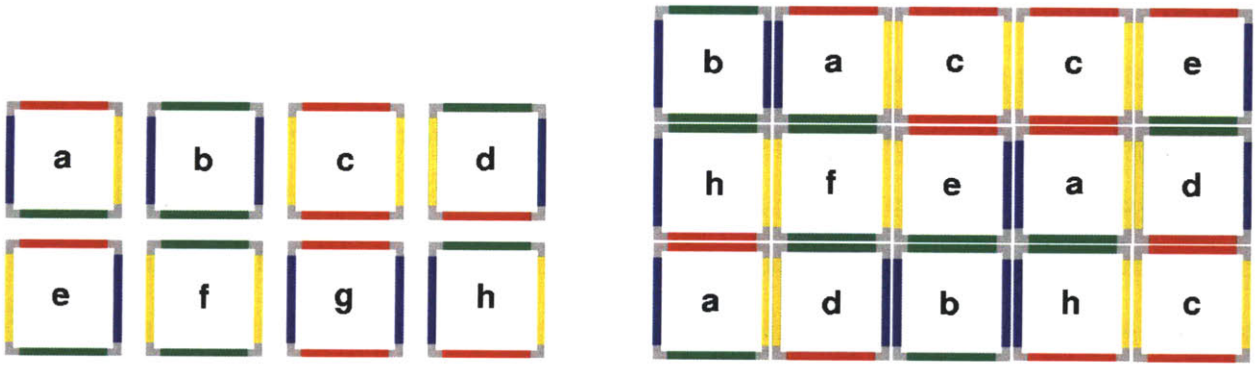


Figure 67. Figure 1: a) Eight Wang Tiles that can stochastically tile the plane; b) A small portion of the plane with a valid tiling.[74]

68 H. Wang. 1965. Games, logic, and computers. (Scientific American. 98–106, November).
69 H. Wang. 1965. Games, logic, and computers.
70 Michael F. Cohen, Jonathan Shade, Stefan Hiller, Oliver Deussen. Wang Tiles for Image and Texture Generation. (Microsoft Research, Wild Tangent, University of Washington and Dresden University of Technology).
71 Michael F. Cohen. Wang Tiles for Image and Texture Generation.
72 Michael F. Cohen. Wang Tiles for Image and Texture Generation.
73 Turing Complete. http://c2.com/cgi/wiki?TuringComplete.
74 Michael F. Cohen. Wang Tiles for Image and Texture Generation.

Wang Tiles are extremely similar to *Logic Matter* in that they define moves and provide constraints for neighboring moves based on unit interactions and orientation of the placed unit. Wang Tiles are said to have an infinite number and type of tiles whereas *Logic Matter* utilizes only one type of tile to describe any arbitrary geometry. A single tile eases the coloring constraints associated with Wang Tiles; however the input/output NAND mechanism adds an additional layer of logic to the possible global configurations. After each placement of a Wang Tile, the space becomes increasing constrained based on neighboring colors. With *Logic Matter*, the space is not additively constraining (except based on self-intersection) rather, each step only constrains the next move then provides one output and waits for a second. The second unit then constrains the possible placement and repeats. The global geometry attempting to be described through *Logic Matter* -- resulting in a series of binary input/output strings or instructions -- is actually the most globally constraining principle. The geometry constrains the possible moves and produces a series of inputs. Conversely, a gradient input string constraints the direction of growth as well as the possible moves at each step. These methods of constraining input can relate to a jigsaw puzzle, as described in Section 04, where each piece constrains the next moves and there is only one possible configuration of the pieces. Jigsaw puzzles have been shown to be NP-Complete and directly equivalent to other types of edge-matching puzzles. It can easily be imagined that jigsaw puzzles or any complex assembly of units would be far simpler and faster to solve if each move not only constrained the next moves, but informed the user of the move to make. This means that costly search and backtracking can be eliminated when decided the next moves. *Logic Matter* attempts to infuse this local decision making and information storage directly into our puzzle pieces, or material parts to aid in the assembly of puzzle-like three dimensional structures.

We saw that *Logic Matter* can be seen as comparatively close to Wang Tiles in terms of complexity and embedded computing, while offering a third dimension of geometric adjacency. We can thus deduce that with reasonable extensions *Logic Matter* should prove to be similarly complete as a model of computation. We will look at a final example of DNA computation that will hopefully further articulate the implied universality.

As noted in previous sections (Section 03, Section 04), *Logic Matter* can be well associated with micro scale structures, more specifically the encoding of DNA/RNA into single strand sequences. DNA computation has been introduced by a number of individuals including: Abelson et. al's Amorphous Computing, Knight and Sussman's Cellular Logic Gates, Rothemund's DNA origami and work on DNA Turing Machines and many others.[75][76][77] Amorphous computing aimed at building arbitrary computational models from less-than perfect individual components. Knight, Sussman and Rothemund have demonstrated that we can do useful computing through logic in cellular structures, even demonstrating DNA as a complete Turing Machine.[78][79] Rothemund also showed that we can actually describe useful geometries and force DNA to fold to form arbitrary single path strands. There are obvious correlations with *Logic Matter* in terms of the single strand typology and the embedded instructions of DNA sequencing, however, I should go slightly further to demonstrate the elements of a Turing machine within *Logic Matter*.

The essential elements of a Turing machine include; a tape, head, table and state register. The tape obviously compares to the single strand of the NAND mechanism, possibly extending infinitely in either direction. Each NAND mechanism stores the state of its position through its interaction with adjacent elements. The occupied face registers the current state. The head is not directly implemented (other than the user); however a simple extension could be imagined that would travel along the tape and read, element-by-element, the instructions for the next steps. Similarly, the head could also be defined as the last unit placed at any moment, because it actually reads the state of the previous and decides the fate of the next. The table can relate to the possible input sequences at any given move and the relationship with the previous moves. There are four possible input combinations and two possible output combinations. The current implementation of *Logic Matter* obviously has restrictions that differ from the DNA and Wang Tile examples but it should be emphasized that both strongly point to the possible extension of *Logic Matter* as a powerful model of computation, potentially capable of being Turing Complete.

75 Harold Abelson. Amorphous Computing.
76 Thomas K. Knight. Cellular Gate Technology. (MIT Artificial Intelligence Laboratory. July 1997).
77 Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines. (CalTech).
78 Thomas K. Knight. Cellular Gate Technology.
79 Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines.

Physical Properties

The specifics of the NAND mechanism and any aggregation with a large number of parts can be analyzed in terms of the efficiency to describe geometries, as well as its performance for programmability. The proposed mechanism, right-angle tetrahedron geometry and linearity of the assembly does not offer excessive efficiency, rather it imposes an extremely redundant overall configuration. For any global configuration there will be twice as many modules used than were actually necessary to describe the single path. That is a direct result of the redundant input tetrahedrons, allowing the geometric and digital logic transformation at each step. The powerful opportunities of redundancy have already been mentioned, however it is important to realize the limitations of such a redundant system.(See Section 03) In terms of cost per module, the units could be manufactured fairly efficiently and inexpensively (depending on the fabrication technique), however it is inevitable that the global configuration will cost twice as much as necessary. This leads to the possibility that the redundant tetrahedrons could be utilized on-demand at each step and removed after activating the gate. They could be only left in moments of extreme loading or points that required higher redundancy.

In the construction of larger NAND circuits, the different configurations of the universal NAND to construct the other gates mandates that inputs and outputs be split and distributed between connected gates.(See Figure 09) This imposes a direct problem with the proposed mechanism because we are limited to linear growth, rather than offering points for branching at each step. Previously, I discussed that the redundant tetrahedrons could offer the moments of branching, but it is unclear at this point of the functionality of the universal NAND could be utilized simply through the redundant inputs. Alternatively, the splitting of output traces could be thought of as a complete copy of the path so far, where the entire string would be duplicated and used as the input to the last module. This is even more redundant than the original string but allows the fanout of more complex circuits to be created with the NAND mechanism.

Another limitation of the proposed mechanism is the limitation for loops, or flow control. The programmability of the assembly sequence is somewhat limited by the on-the-fly computing nature of the designed NAND mechanism. This does not allow for a programmed sequence to actually loop back to itself and re-influence the entire string. After each node is computed (two inputs leading to one output) that gate is considered fixed or complete. That means that no other inputs/outputs can have an effect on that unit. This ultimately means that the global configurations are permanent and cannot be reconfigured. This is known to be a major drawback to the current implementation and would hopefully be addressed in the very first revision of the system. However, one exciting work-around that could lead to even more exciting results is the idea of the "snake eating its own tail", where the units at the beginning of a long sequence are actually removed and utilized at the end. This would allow the structure to be reprogrammed and actually loop through itself for recursive influence. This idea has not been fully tested but potentially offers an exciting way for reprogrammability and broader functionality as a device for computing.

Specifically as a physical, user-interactive, device for computing, *Logic Matter* will most likely not become influential due to the immense number of modules needed for powerful computation and fairly slow speeds due to users actually assembling the structure. However, in principle, if it is deployed as a system of assembly that utilizes the infused nature of computing, it should prove to be far faster and more accurate than traditional means of construction when attempting to build large or complex structures. The time saved from look-up tables, reading blueprints or 3D models, excessive scouring for parts in piles of thousands of pieces, the eminent remeasuring and time lost to errors argue the benefits of *Logic Matter* for efficiency while building complex assemblies. The emphasis has been placed on complex assemblies due to the inherent difficulty in assembling these structures. Complex assemblies better demonstrate the benefits of *Logic Matter* as opposed to simple structures that are produced fairly easily and successfully with traditional assembly systems.

Correlations could be drawn from the work on Amorphous Computing and their initial thoughts on the usefulness as a tool for cellular computing:

> "Thus, we do not anticipate that cellular computing in itself will be a good way to solve computationally difficult problems. On the other hand, the ability to organize cells into precise patterns and to cause cells to secrete chemical components could be the foundation for the engineering construction of complex extracellular structures and precise control of fabrication at the subnanometer level."[80]

Finally, opportunities may be afforded through *Logic Matter* as a low energy, low set-up and minimal information storage system for assembly. To build a global configuration, *Logic Matter*, requires little to no energy, other than the physical labor of placing parts because electricity and motors have been eliminated. This allows infrastructure to be removed and enables opportunities for communities without established technologies to build complex, functional structures while benefiting from the same technological advantages that we receive through our digital machines. Finally, the information required to build such structures can literally be removed from the user, minimizing the number of assemblers (human or machine) required and the information, confusion and foreseen errors common in today's assembly systems. Further improvements to reduce the complexity of the physical NAND mechanism could help to eliminate even more information/confusion from the user as well as ease the process of fabricating excessive numbers of modules required to build substantial structures.

80 Harold Abelson. Amorphous Computing.

[07] Conclusions

Contributions

In this thesis I have attempted to illustrate opportunities in the assembly of complex structures, arguing the inevitable flaws of our current assembly and construction systems while leveraging the well-known benefits of digital information and self-assembly. I argued that if we want to build more complex structures than humanly possible today, then we need to embed discrete assembly information (and computation) directly into our material parts. I outlined a context of work, the demands of today's complex physical environment and tomorrows desired structures, necessitating infused digital logic within our material parts. I then introduced the foundation of this thesis, a system called *Logic Matter* that offers possibilities for the assembly of large-scale complex structures by embedding discrete assembly information into physical NAND mechanisms. I described the designed NAND mechanism and demonstrated its functionality and programmability. Useful applications were outlined by showing the decoded descriptions of any arbitrary geometry (lines, surfaces and volumes) through a sequence of NAND mechanism. A method of computing through construction was developed that enables a number of exciting potentials in physical assembly of complex structures, including: local knowledge, material as a storage device, stopping mechanisms, single stream processor plus hard drive and self-replication. Finally, I showed a working prototype that embodied the geometric descriptions and on-the-fly computing embodied in a physical implementation of digital logic for assembly: *Logic Matter*.

Future Work

Throughout the development of this thesis a number of potential applications and exciting areas of future research have emerged. In this section I would like to touch on a few of the possible area of future work as well as reemphasize applications of *Logic Matter* at extremely small scales, large scales and extremely precise assemblies. First let us look at extremely small scale applications.

The inherent information storage, self-guided-replication possibilities and single stream hard drive/processor combination provide powerful tools for small scale structures due to the power of computing that is embedded directly in the materials. As noted in Section 05 on Self-Guided-Replication, extremely small-scale assembly machines (actual mechanism that can occupy our blood stream and perform a myriad of tasks) may be built in the future and most likely won't pack universal computation into each device. *Logic Matter* provides a powerful possibility of packing computation directly into the deposited materials rather than the device. Further, if the machines are actually depositing natural materials rather than synthetic geometries, we might be able to utilize the scaffolding principles, discussed in Section 05, to build temporary structures that allow proteins to grow/fold within the confines of the informative scaffold. The scaffold could infuse local decision making and dictate folding sequences for naturally grown proteins, much like the impressive power of the Ribosome to decode our bodies RNA into a sequence of folds to create complex proteins.[81] This argues for the ultimate application of constructing complex, highly functional, biological structures from unreliable and inconsistent elements!

As seen in previous sections (Section 03, Section 04, Section 05) large number of corollary projects have addressed the idea of molecular computing such as Abelson et. al's Amorphous Computing, Knight and Sussman's Cellular Logic Gates and Rothemund's DNA origami and

81 Carl Branden, John Tooze. Introduction to Protein Structure. (New York: Garland Publishing Inc.. 1991).

work on DNA Turing Machines.[82][83][84] These projects offer an interesting correlation to the proposed system of *Logic Matter* in that they are aimed at functional geometric descriptions by embedding information computing into materials, thus allowing imperfect parts to interact and add to larger, perfect, structures. When combined with ideas in synthetic biology and unresolved problems in protein folding, the informed folding of arbitrary single strand sequences in *Logic Matter* becomes increasing exciting and plausible!

> We can envision applying this technology to the construction of molecular-scale electronic structures. One plausible way to construct complex, information rich electronic systems is to first fabricate a largely passive but information-rich molecular-scale "scaffold" consisting of selectively self-assembling engineered molecules. This scaffolding would be used to support fabrication of molecular conductive and amplification devices interconnected as the engineer requires. Proteins represent good candidates for scaffolding components; they are chemically and thermally stable and have exquisitely selective binding domains.[85]

A second exciting avenue for *Logic Matter* is the construction of three dimensional, functioning, circuits. Due to the nature of the working NAND mechanism, *Logic Matter* offers the materials necessary to build arbitrary circuits. The main emphasis of this thesis has been on computing directly for assembly -- not necessarily focusing on the output sequence after being computed. However, functioning NAND circuits could potentially offer useful computation and circuit schematics. This may require that the configurations branch, loop or be reconfigurable, all of which are not directly embedded in the current generation of the designed NAND mechanism, although, they could be easily be implemented in future. (See Section 03 on branching). The current NAND mechanism still can perform circuit assembly, simply by copying large chunks of the sequence and using it for input, representing branching or fanout in the circuit diagram. I have tested the implementation of a half-adder circuit that successfully adds any two digit number (0+0=0, 0+1=1, 1+0=1, 1+1 = 2).(See Figure 68)

82 Harold Abelson. Amorphous Computing.
83 Thomas K. Knight. Cellular Gate Technology.
84 Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines.
85 Harold Abelson. Amorphous Computing.

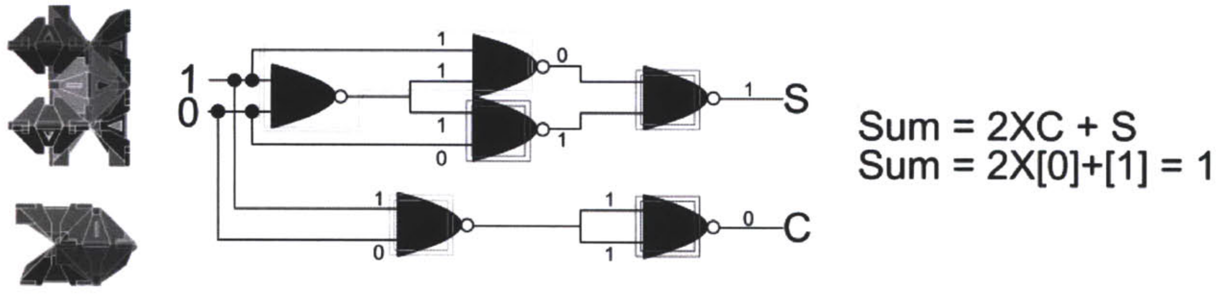


Figure 68. Successful falf-adder circuit construction from NAND mechanisms.

The circuit description might be given as a graphic representation of the circuit or a translated syntax, then the search strategy could be to arrange and grow the NAND mechanism such that the circuit traces are maintained. The information would be passed through the appropriate gates and a successful output would be realized. For example, the adder might be desired and the modules would be constructed in a way as to realize the adder circuit in 3D space while concurrently computing the addition of the input binary numbers. The resultant output is the combination of 3D configuration and final carry bits, dictating the result of the computation. Three dimensional circuit construction might have potential applications for schematic testing of circuits before prototyping. *Logic Matter* would thus become a tool to aid users in circuit design. The circuits might also be functioning while embedded directly into our physical environment of walls, building and infrastructure. Computing materials could allow our walls to be passing information, deciding on environmental conditions and dictating adaptive spatial transformations as output. A major area for further development is in the reconfiguration of the overall assembly. The current design only allows computing to be performed directly as the units are placed. That implies that the final configuration can no longer perform computation. If we want to perform larger computations or have larger inputs (from sensors or environmental influences) then we need to have reconfigurable systems that can feedback to themselves, and adjust the processed information. This would require functional loops in the system, possibilities for disassembly and a redesigned mechanism that could store energy for the bit switching. This has been largely unexplored in this thesis but could inspire exciting topics for future work.

On the large-scale, *Logic Matter* has obvious applications in terms of precision construction and localized building, rather than constantly requiring global information (how far have we gone, have we veered off course etc). (See Section 05 on Building an Infinite Line). Structures are becoming increasingly larger, digging deeper into the earth and higher into space.(See Figure 69) Consequently, these structures can no longer rely on the error prone construction techniques of our current industries. *Logic Matter*, offers a system of assembly that empowers the construction team to have greater guarantees in the accuracy of their work, requires less skilled labor and knowledge of the global configuration while leading to few mistakes and communication issues. The inherent nature of *Logic Matter* is scaleless, the functioning of the mechanism depends solely on geometry that can scale in any direction, large or small. The units could be directly scaled in size or in number of parts, while still maintaining the computing functionality and transfer of information. However, if we want to utilize a variety of scales in our system -- units functioning as larger structural elements with hierarchy of scales and functions -- the interface between differently scaled units would need to be designed and tested. This transition between differently scaled units still remains unexplored and offers promising avenues for future research.

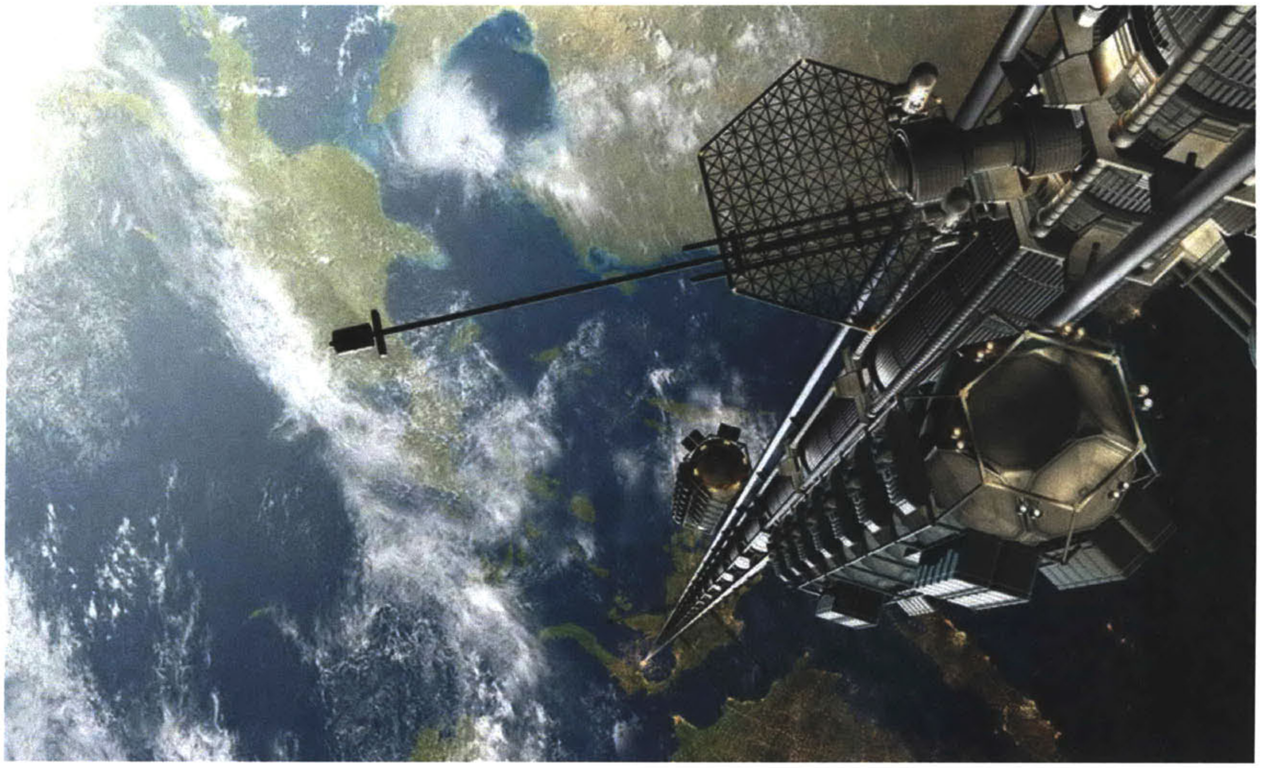


Figure 69. Space Elevator.[86]

86 Space Elevator Reference. http://blog.lib.umn.edu/whee0113/architecture/images/06_SpaceElevator.jpeg.

Our material parts of the future should talk to one another. They should talk about the accuracy of the past and the decisions of the future.

Q: Can we build arbitrary structures from buildings to bio-machines with unbiased parts, while knowing nothing about the assembly sequence or global configuration?

[08] Bibliography

[00] Abelson, Harold, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous Computing. MIT. 2000.

[01] Amend, J. R., Jr., H. Lipson. Shape-Shifting Materials for Programmable Structures. Cornell University.

[02] Bachrach, J..DARPA Programmable Matter Report: Programming Chained Robotics in the Gas Programming Language. Makani Power. 2009.

[03] Bachrach, J., V. Zykov, S. Griffith. Folding Arbitrary 3D Shapes with Space-Filling Chain Robots: Folded Configuration Design as 3D Hamiltonian Path through Target Solid. Makani Power.

[04] Bachrach, J., V. Zykov, S. Griffith. Folding Arbitrary 3D Shapes with Space Filling Chain Robots: Reverse Explosion Approach to Folding Sequence Design. Makani Power.

[05] Branden, Carl, John Tooze. Introduction to Protein Structure. New York: Garland Publishing Inc.. 1991.

[06] Cohen, Michael F., Jonathan Shade, Stefan Hiller, Oliver Deussen. Wang Tiles for Image and Texture Generation. Microsoft Research, Wild Tangent, University of Washington and Dresden University of Technology.

[07] Crouch, David and Nina Lubbren. Visual Culture and Tourism. Oxford, UK: Berg Publishers, 2003.

[08] Dalrymple, David. Asynchronous Logic Automata. Masters of Science Thesis. MIT, 2008.

[09] Dalrymple, David, Erik Demaine, Neil Gershenfeld. Reconfigurable Asynchronous Logic Automata. MIT, 2009.

[10] Drexler, K. Eric. Molecular Machinery and Manufacturing with Applications to Computing. PHD diss. MIT, 1991.

[11] Fitch, Robert and Zack Butler. Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots. The International Journal of Robotics Research 2008; 27; 331

[12] Frazer, John. An Evolutionary Architecture. 1995

[13] Fredkin, E. and Toffoli, T. 2002. Conservative logic. In Collision-Based Computing, A. Adamatzky, Ed. Springer-Verlag, London, 47-81.

[14] Gershenfeld, Neil. Fab: The Coming Revolution on Your Desktop--from PersonalComputers to Personal Fabrication. New York: Basic Books, Inc.. 2007.

[15] Griffith, Saul Thomas. Growing Machines. PHD diss. MIT, 2004.

[16] Kei-Chen, Lee. How To Learn Lee's Abacus. Lee's Abacus Correspondence School. Taiwan, China, 1958.

[17] Knaian, Ara. Design of Programmable Matter. Masters of Science Thesis. MIT, 2008.

[18] Knight, Thomas K.. Cellular Gate Technology. MIT Artificial Intelligence Laboratory. July 1997.

[19] Kriesel, David, E. Cheung, M. Sitti, and H. Lipson. Beanbag Robotics: Robotic Swarms with 1-DoF Units. Cornell University & Carnegie Mellon University.

[20] Lobo Daniel, D. A. Hjelle 1, H. Lipson. Reconfiguration Algorithms for Robotically Manipulatable Structures. Cornell University & Universidad de Málaga.

[21] Marchal, Pierre. Field-Programmable Gate Arrays. Communications of the ACM; Apr99, Vol. 42 Issue 4, p57-59, 3p.

[22] Minsky, Marvin. The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind. Simon and Schuster, 2007.

[23] Penrose, L. S.. Self Reproducing Machines. Scientific American vol.200. June 1959.

[24] Popescu, George A.. Digital Materials for Digital Fabrication. Masters of Science Thesis. MIT, 2007.

[25] Prakash, Manu and Neil Gershenfeld. Microfluidic Bubble Logic. Science 315 (5813), 832. February 2007.

[26] Rothemund, Paul W. K.. A DNA and restriction enzyme implementation of Turing Machines. CalTech.

[27] Rothemund, Paul W. K.. Folding DNA to create nanoscale shapes and patterns. Nature, Vol 440. March 2006.

[28] Scherz, Paul. Practical Electronics for Inventors. New York: McGrawHill, 2000.

[29] Schneider, Mike. Radio Frequency Identification (RFID) Technology and its Applications in the Commercial Construction Industry. University of Kentucky. April 24, 2003

[30] Stoy, Kasper, David Brandt and David J. Christensen. Self-Reconfi gurable Robots. Cambridge, Massachusetts : The MIT Press, 2010.

[31] Thompson, Adrian. An evolved circuit intrinsic in silicon entwined with physics .COGS University of Sussex Brighton, UK.

[32] Tolley T. Michael, M. Krishnan, H. Lipson, D. Erickson. Advances Towards Programmable Matter. Cornell University.

[33] Wang, H. Games, logic, and computers. Scientific American. 98–106. November 1965

[34] Winston, Patrick. Learning Structural Descriptions from Examples. PHD diss. MIT, 1970.

[35] Vacant, Martin P.. 2008. Biological Scaffolding Material. U.S. Patent 7,319,035,B2 fi led, October 17, 2003 and issued January 15, 2008.

[36] Von Neuman, John. Theory of Self-Reproducing Automata. University of Illinois Press, Urbana and London. 1966.

[37] Zakin, Mitchell. "Programmable Matter." Defense Science Office. DARPA. May 2009.

[09] Table of Figures