Rapid Design and Simulation of Functional Digital Materials

by

Amanda Paige Ghassaei

B.A., Physics, Pomona College (2011)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Author . . . .
Program in Media Arts and Sciences
August 12, 2016

Certified by . . . .
Prof. Neil Gershenfeld
Director, MIT Center for Bits and Atoms
Thesis Supervisor

Accepted by . . . .
Prof. Pattie Maes
Academic Head, Program in Media Arts and Sciences

Rapid Design and Simulation of Functional Digital Materials

by

Amanda Paige Ghassaei

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on August 12, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Digital fabrication aims to bring the programmability of the digital world into the physical world and has the potential to radically transform the way we make things. We are are developing a novel digital fabrication technique where a small basis set of discrete part types, called "digital materials", are reversibly joined into large assemblies with embedded functionality. Objects constructed this way may be programmed with exotic functional behavior based on the composition of their constituent parts.

In this thesis I build an end-to end computer-aided design (CAD), simulation, and manufacturing (CAM) pipeline for digital materials that respects the discretization of the parts in its underlying software representation. I propagate the same abstract geometric "cell" representation of parts from the design workflow into simulation and path planning. I develop a dynamic model for simulating anisotropic, multimaterial assemblies of cells with embedded mechanical and electronic functionality based on local interactions. I demonstrate the similarities between my mechanical model and the Timoshenko Beam Element. I note an advantage of my model for simulating flexural joints is its non-linear treatment of angular displacements - allowing for large angular deformations to be simulated without costly remeshing. I implement this model in software and demonstrate its potential for parallelization by calculating each cell-cell interaction in a separate core of the GPU. I compare my simulation results with a professional multiphysics software package. I demonstrate that my tool facilitates rapid exploration of the design space around functional digital materials with several examples.

Thesis Supervisor: Prof. Neil Gershenfeld
Title: Director, MIT Center for Bits and Atoms

Rapid Design and Simulation of Functional Digital Materials

by

Amanda Paige Ghassaei

The following people served as readers for this thesis:

Caitlin Mueller . . . .

Thesis Reader

Assistant Professor, Department of Architecture, MIT

Erik Demaine . . . .

Thesis Reader

Professor, Computer Science and Artificial Intelligence Lab, MIT

Acknowledgments

To Neil - thanks for creating a space at MIT for research that doesn't fit wholly within one field. Thanks for setting up probably one of the best places in the world to make things, and thanks for occasionally letting me use the tools to make things that aren't necessarily research.

To my readers Caitlin and Erik - thanks for your time and input and for helping me to ground this work in existing techniques. I hope that we keep working together.

To the CBA students - Sam, Will, Matt, Ben, Eric, Nadya, Prashant, Michael, Noah, Thras, Charles, Lisa, and James - you've all taught me so much. Thanks for the times we hung out and didn't talk about research too! Special thanks to Will for all the long conversations about assemblers and helping me to not lose my mind.

To everybody that keeps CBA going - Jamie, Ryan, Joe, Sherry, Tom, and John - thanks for all the long chats, antics, dog days, digikey orders, for keeping the shop running smoothly, and for taking me all around the world to do cool stuff.

To everybody at E-Line - I've been really enjoying our conversations and the way that our collaboration has been shaping the research. Thanks for helping me to think about this work from a totally different perspective.

To Palash, Jon, Tomer, and Chikara - thanks for the pep talks, dinners, and for help navigating the crit day madness.

To everybody at Instructables - thanks for providing the perfect place to experiment and grow after graduating from college. Special thanks to Dave and Phil for teaching me how to be a real programmer.

To Dwight and Phil and the rest of the Pomona Physics Dept - in a lot of ways I feel like this work is an extension of my undergrad thesis. Thanks for letting me work on projects that didn't seem to fit in a physics department and giving me a safe place to fail.

To my Mom, Dad, and Tracey - thanks for always supporting me even though sometimes it's hard to explain what I'm doing.

To Michael - thanks for putting up with my terrible work/life balance and helping me through the stress and occasional crying. Thanks for staying on the phone with me when I walk home at 3am and for listening when I go on and on about the compass. Even though I'm still not that tough, thanks for making me tougher.

Contents

1 Introduction **8**

1.1 Motivations 9

2 Related Work **11**

2.1 Programmable Materials 11

2.2 Digital Materials and Digital Assembly 11

2.2.1 Macro Assembly 12

2.2.2 Micro Assembly 12

2.2.3 Nano Assembly 12

2.3 Self-Replication 14

2.3.1 Physical Self-Replicating Systems 14

2.3.2 Virtual Self-Replicating Systems 14

2.4 CAD and Simulation Tools for Digital Assembly 16

2.4.1 Cellular Automata-Based Tools 16

2.4.2 DNA Origami-Based Tools 17

2.4.3 Physics-Based Tools 18

3 Software Workflows for Digital Materials **20**

3.1 Design Abstraction 20

3.2 Design Hierarchy 23

3.3 Assembler Abstraction 27

3.4 Off-Ramps . . . 30

4 **Digital Materials and Self-Assembling Assemblers** **33**

4.1 Elements . . . 34

4.2 Functions . . . 38

4.3 Modules . . . 39

4.4 Complexes . . . 39

4.5 Systems of Complexes . . . 40

4.6 Design Hierarchy in Biology . . . 40

4.6.1 Elements, Functional Groups, and Functions . . . 40

4.6.2 Modules and Complexes . . . 43

4.6.3 Insights . . . 43

4.7 Design Hierarchy in Conway's Game of Life . . . 44

4.7.1 Elements, Motifs, and Functions . . . 45

4.7.2 Modules and Complexes . . . 49

4.7.3 Insights . . . 51

5 **Simulation of Functional Digital Materials** **52**

5.1 Solid Mechanics . . . 52

5.2 Modeling Setup . . . 57

5.3 Spring-Damper Characteristics . . . 59

5.4 Translational Forces . . . 64

5.5 Rotational Forces . . . 67

5.6 Actuation . . . 70

5.7 Sources of Error . . . 72

5.7.1 Numerical Time Integration . . . 72

5.7.2 Poisson's Ratio . . . 74

5.7.3 Floating Point Operations . . . 74

5.8 Electronic Simulation . . . 74
5.9 Collision Detection . . . 75

6 Implementation **76**
6.1 Javascript/WebGL . . . 76
6.2 GUI . . . 77
6.3 GPU Programming . . . 78
6.4 Other Performance Speedups . . . 80
6.5 Instability . . . 82
6.6 Examples . . . 83

7 Evaluation **87**
7.1 Comparison with FEA Techniques . . . 87
7.1.1 Conclusions . . . 92
7.2 Comparison with VoxCAD Physics Engine . . . 94
7.2.1 Conclusions . . . 96
7.3 Comparison with COMSOL Simulation . . . 96
7.3.1 Conclusions . . . 99
7.4 Performance Metrics . . . 101
7.4.1 Conclusions . . . 101
7.5 Final Conclusions . . . 105

8 Future Work **106**
8.1 Hierarchical Simulation . . . 106
8.2 Declarative Design . . . 108
8.3 "Fab the Game" . . . 109

Chapter 1

Introduction

An aspirational goal of digital fabrication is the bottom up programmable assembly of meter-scale objects with nanometer-scale precision. With this technology, we could design materials with exotic physical properties and radically transform the way we make almost anything. Current efforts aim to improve the precision and speed of top-down nanofabrication processes to make increasingly more complex nanodevices. We believe a more scalable approach leverages the parallel actions of millions or billions of nanoscale assemblers, rather than a single, monolithic machine. Though it sounds like science fiction, biology has demonstrated that this is possible; data encoded in DNA can be executed like a computer program to construct an immense assortment of molecular-scale machines, which together, coordinate the higher-level structures and functions of an organism.

In our proposed assembly system everything is constructed from a relatively small basis set of discrete feedstock, called "digital materials". Drawing inspiration from the amino acid building blocks of biology, digital materials are reversibly joined in a highly parallel, serial assembly process to produce diverse, functional structures. Since construction takes place one nano-brick at a time, many assemblers work in parallel to build structures of any significant size. Assemblers are designed so that they can be constructed from their own feedstock; assemblers build more assemblers and the rate of assembly scales exponentially.

Parallel nano-assembly will look very different from the way we make things today, and opposes many of the assumptions baked into traditional Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and robotics design. Mechanical systems are built with discrete modules of rigid and flexural components assembled on a regular lattice, and electronics and controls are distributed spatially across a machine. Large structures appear to be "living" in the sense that their surface is teeming with nano-robots, detecting and correcting errors, shuttling material feedstock around, and performing distributed sensing and actuation functions. Machines receive instructions from their environment to coordinate various tasks. The environment is highly structured, allowing locomotion systems to position themselves globally by counting local movements across a lattice.

This thesis outlines work toward fully integrated, end-to-end software workflows for digital materials that will help us to realize the aspirational future I've just described. Current work is targeted at micro, milli, and macro-scale applications, with an eye toward scaling down to nanoscale. The main contributions include:

CAD: a CAD environment that respects the inherent discretization of the parts and expedites the design of large, multimaterial assemblies of digital materials. Hierarchical construction gives the end user parametric controls over the material composition of assemblies. Abstraction of the lattice geometry from its part decomposition simplifies the CAD interface and provides a meaningful underlying representation going into simulation. CAD workflow is detailed in Chapters 3 and 6.

Simulation: a dynamic physics engine that leverages the discrete construction of digital materials to simulate their electronic and mechanical behavior. Because the underlying representation of digital material parts in simulation is equivalent to their CAD representation, my simulation methods result in better performance and less user hassle than is possible using professional multiphysics software packages. My simulation pipeline has been GPU-accelerated so that it performs a dynamic simulation of hundreds of parts in realtime, allowing users to rapidly iterate on their designs. Additionally, the simulation model I've developed in this thesis uses a non-linear treatment of part orientations so that large angular deformations are handled correctly (this is typically problematic using standard FEA techniques, which are full of small angle approximations). Simulation methods are detailed in Chapter 5 and evaluated against existing methods and software in Chapter 7.

CAM: a CAM workflow for the current iteration of macro-scale assemblers in development at CBA. An abstracted machine definition allows a wide diversity of assemblers to be configured through the CAM interface, with hooks for custom g-code generation and machine settings. The discrete representation of the parts in software reduces the complexity of path planning strategies. The CAM workflow is detailed in Chapter 3.

1.1 Motivations

The primary motivation for this work is to provide a platform for rapidly exploring the design space around digital materials in a physically realistic way. This work will inform future research trajectories at CBA and in the broader field of programmable materials, modular robotics, and digital fabrication.

Multiphysics for the Masses

The discrete, regular lattice geometry of digital assemblies reduces the complexity of design and simulation workflows and increases the computational efficiency of simulating hundreds or thousands of parts at once. A side effect of this reduced complexity

is that software tools for digital materials can be designed to have a very low technical barrier to entry. Leveraging this, the tools developed in this thesis aim to be a kind of "Minecraft with physics": a multiphysics sandbox that even a novice user can quickly understand and operate.

Experiments in Self-Replicating Systems

The CAD/simulation tool I've built follows from a lineage of virtual construction environments and simulators where early explorations in distributed machine design and self-replication were first performed. Many of these virtual environments have since been opened up to a larger, global community of researchers and enthusiasts. In the fullness of time, I hope that my software tools will grow into an accessible platform for exploring physical, self-assembling systems based on the foundations of engineering and materials science rather than biology.

Chapter 2

Related Work

The following sections outline current research into discretely-assembled construction platforms across scales and CAD/simulation packages which occupy a similar space to what I have built.

2.1 Programmable Materials

Programmable matter describes materials that can be programmed to change their physical properties based on user input or environmental changes. Often, programmable materials are composed of many smaller constituent elements, whose structural arrangement and local interactions are the source of larger-scale programmable properties. Some forms of programmable material include metamaterials [70] [43], modular robotics [61] [58], and shape-changing materials [86] [33] [78]. Programmable matter may also describe computing systems, for example CAM8 is a computing architecture in which physical phenomena can be efficiently simulated based on local interactions between parallel computing elements [79].

2.2 Digital Materials and Digital Assembly

Current research at CBA revolves around a discretely-assembled form of programmable matter called *digital materials*. Digital assembly is an emerging multimaterial fabrication technique where a finite set digital materials are (often reversibly) joined to form larger structures in a regular lattice. Programmable functional behavior is achieved by patterning parts with different material properties across an assembly. The assembly process is orchestrated by one or many robotic assemblers. The regular spacing of the parts minimizes the complexity of the assembly design, the robotic assemblers that locomote across the lattice, and the "path planning" strategies for assembly. Self-alignment features on the digital material parts maintain global metrology through local interactions; a robotic assembler can construct a structure more precise than itself, within a certain error tolerance. At nano scales, assembly may be guided by programmable self-assembly mechanisms. Chapter 4 describes hierarchical scaling of complexity within digital assemblies.

2.2.1 Macro Assembly

Cheung showed that carbon fiber composite parts can be reversibly assembled to form ultralight materials for aerospace applications [18]. Programmable flexibility is achieved by pattering rigid and flexural parts across an assembly and also by varying the density and connectivity of the parts. Additional research into digital assembly of structural elements is ongoing at CBA, including modeling of the parts and assemblies using finite element analysis [12] and designing new part geometries and robotic assemblers [13].

2.2.2 Micro Assembly

Langford demonstrated how conducting, insulating, and resistive part types ranging in scale from mm-μm could be assembled to form any passive electronic component, including antennas and RF matching networks [47]. Langford designed and built a dual-material assembler (Fig: 2-1) for constructing passive electronic structures from conducting and insulating parts (video) [46]. Langford suggests that by extending the part types to include several types of silicon components, discretely-assembled active components like diodes and transistors would be possible. Work toward the fabrication and realization of these silicon components is ongoing. This work follows from previous work by Popescu et al. on reversibly assembled "GIK" structures spanning many length scales and material types [54] and also work by Ward on discretely assembled electronics [81].

Hiller et al. propose a method of additive manufacturing using multimaterial voxel building blocks [35]. They explore various voxel decompositions of 3D geometry and assess their suitability for rapid, parallel assembly processes (video). Possible applications discussed include desktop manufacturing, especially for electromechanical and microfluidic devices. Hiller et al. also propose reversible processes for disassembling voxel assemblies back into their constituent parts [38].

Though not a reversible process, multimaterial 3D printing (most notably by the Objet printer) deposits material in voxels on the order of $10\mu m^3$ with a total build volume on the order of $1x10^{10}$ voxels [71]. Multimaterial 3d printing has been demonstrated in optical [82], electronic [5], and structural applications [68] [62] [7].

2.2.3 Nano Assembly

"DNA bricks" is a method of discrete nano-assembly based on complementary base pair interactions of short segments of DNA [41]; DNA bricks forms a branch of a field called DNA origami [60] or DNA computing [63] [4]. The brick assemblies have a spatial resolution of 2.5nm; the longest dimension of a DNA brick assembly measures on the order of $1\mu M$ [42]. Unlike the previously discussed processes, assembly of DNA bricks takes place through passive, stochastic interactions between DNA strands in

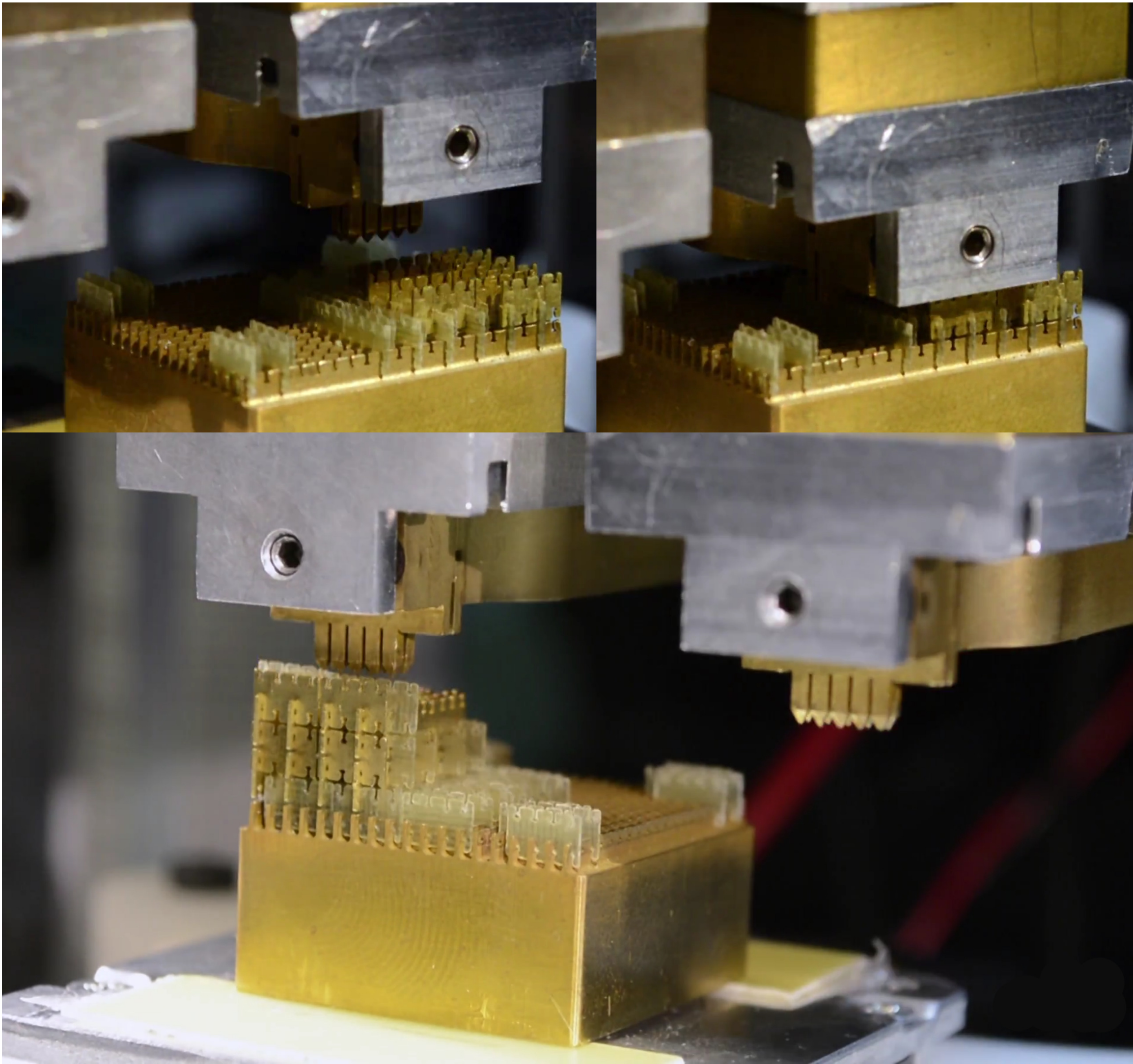


Figure 2-1: “Stapler” assembler designed by Will Langford assembles conductive and insulating discrete part types to form electronic structures with tunable capacitance and inductance. *Image Credit: Will Langford 2015*

solution, rather than guided assembly by robotic assemblers.

CBA is actively involved scaling down the micro-electronic parts designed by Langford [47] into the nanoscale and assembling them using nano-manipulation devices.

2.3 Self-Replication

Some of the first known formal investigations into the nature of self-replicating systems were conducted by John Von Neumann as early as the 1940s. Since then, both physical and virtual self replicating systems have been explored by researchers in robotics, nanotechnology, biology, and computer science, as well as by a community of enthusiasts.

Within the study of self-replication, there is a distinction between so called "trivial" and "non-trivial" self-replication. This distinction is not well formalized; it is loosely based on the difference in complexity between a self replicating system and the parts it's constructed from. For example, crystal growth or polymerization could be considered a type of trivial self-replication because both the assembly and the parts are relatively simple. Penrose Tiles [51] are an example of trivial, artificial self-replication that resembles crystallization. On the other end of the complexity spectrum, self-replicating experiments in the field of modular robotics also fall in the category of trivial self-replication; though the self-replicating robots produced in these experiments are very complex [87] - often with multiple actuated degrees of freedom - the modules they're made from are of essentially equal complexity.

2.3.1 Physical Self-Replicating Systems

Biological self-replication is the only physically-instantiated, non-trivial self-replicating system that is currently known to exist. The field of molecular biology aims to understand structure and functions of this system better, and the field of abiogenesis aims to discover how it first began. An active area of synthetic biology explores the possibility of creating a viable cell from scratch based on knowledge about the core systems required for metabolism, cell maintenance, and self-replication [25]. To date, efforts at creating this "minimal cell" have been successful using a top down approach - starting with an organism with a minimal genome and reducing its genes further [29] [28] [39]. Chapter 4 dives deeper into an explanation of biological self-replication in terms of the work described in this thesis.

2.3.2 Virtual Self-Replicating Systems

This thesis draws inspiration from a lineage of virtual environments wherein the first investigations of self-replication took place. Early experiments were computed by hand on pieces of graph paper or a Go board [26]. With the introduction of personal

computers and graphical user interfaces, it is now possible for anyone to compute large virtual universes and study the behaviors of millions of interacting parts.

Cellular Automata

A cellular automaton (CA) is a discretized model used to describe the dynamics of a system. CAs are discrete in both space in time, meaning space is divided up into many identical *cells* and time moves forward in discrete *steps*. Each cell owns one or many state variables, which may hold discrete or continuous data. The governing equations of a CA system are codified in the "ruleset" that applies universally across all cells in the system; typically a ruleset describes local interactions of a cell with its neighbors. For example, in the popular CA Conway's Game of Life, the state of a cell in the next time step is a function of its current state and the state of its eight neighbors on a 2D grid. Some CAs consider cells to be spatially static with changing state, others employ kinematic rules, where cells can move across space.

CAs have been used to study the requirements of self-assembling systems since Von Neumann's first experiments [50]. To date, CA investigations into self-replication have employed very simple rulesets that violate basic physical principals: matter can be created and destroyed at will, actuators can move an infinite number of cells (cells have no mass), motions are quantized, one material can be converted into another, global synchronicity, etc. This does not imply that CAs are inherently non-physical; in fact, evaluating partial differential equations from physics using finite difference method can be structured of as a type of CA [85].

Von Neumann's Universal Constructor

A universal constructor is a machine that can be programmed to construct nearly any other structure, including a copy of itself. In the 1940's Von Neumann designed a virtual CA world in which he built a universal constructor from 29 distinct parts types with discretized state and behaviors (this work was published posthumously by Arthur Burks in the 1960s [11]). A few notable conclusions fell from this exploratory investigation:

- A distinction between the mechanistic parts of the constructor and the data encoding its construction (the "blueprint") - mirroring the separation of DNA from the proteome.
- Establishing that bimodal operation of the construction blueprint is needed to make self-replication possible. The blueprint must operate in both an interpreted mode - where it is read and executed to build mechanistic components - and a duplication mode - allowing its information to be copied bit by bit. This insight predated the discovery of the structure of DNA and a formal understanding of DNA transcription and replication.

Figure 2-2: Screenshot of a 16 bit computer built in Minecraft by user Ohm. Full video available on YouTube. *Image Credit: Ohm 2011*

2.4 CAD and Simulation Tools for Digital Assembly

At the moment of writing this, there exist several computational tools for designing and simulating static and kinematic, discretely-assembled structures.

2.4.1 Cellular Automata-Based Tools

The CA-based simulation tools discussed here allow for computational efficiency at the cost of accuracy, often using quantized motions, interactions, and states. In general, this classification of tools employs materials and mechanics that are not plausible in the real world, but are still of interest for purely computational investigations.

The most widely adopted example of discrete design is Minecraft, a PC game that gives players the ability to construct their own worlds from over 100 different block types (Fig 2-2). A subset of these block types form the basis of digital logic in the game and another set of part provide a means of simple 1-bit mechanical actuation [48]. Gameplay and available block types are extendable through various mods and user scripts.

Golly is a 2D CA simulator originally intended for Conway's game of life, but is extendable to other rulesets. It implements Gosper's "hashlife" algorithm for optimizing

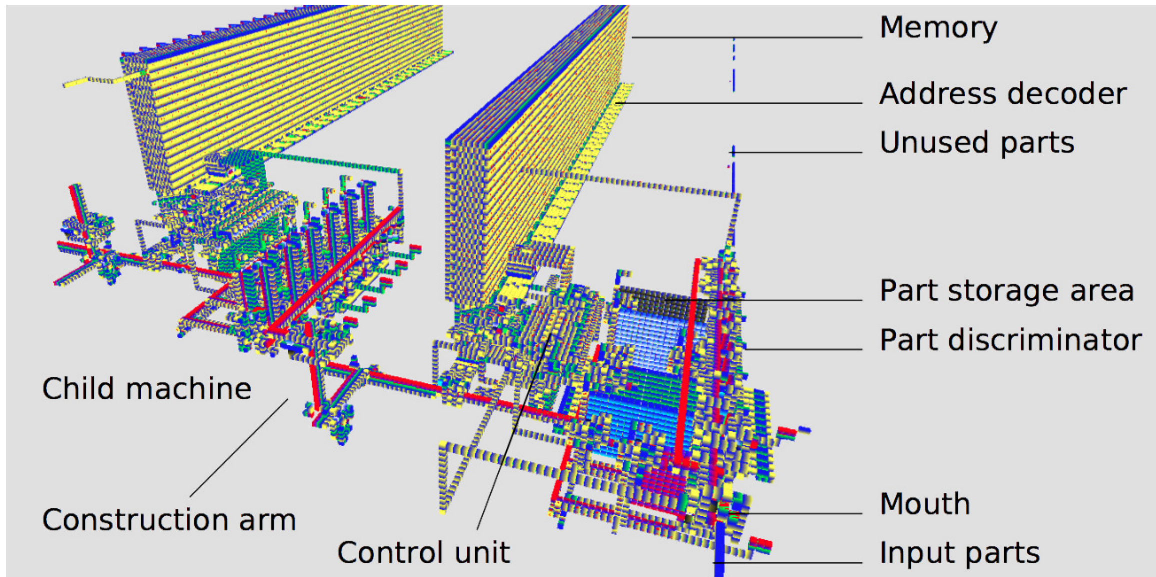


Figure 2-3: A programmable, universal constructor (shown replicating itself) built in CBlock3D by William Stevens from 5040 cells of 6 different types [74]. Full video of assembly process on YouTube. *Image Credit: William Stevens 2010*

the performance of the simulation [30]. Golly's efficient solving and open access has allowed a large online community of CA enthusiasts to collaborate on and experiment with increasingly more complex designs. In 2013 Dave Greene Wade published the "Linear Propagator", a self replicating machine designed in Golly using Conway's ruleset [32]. In 2014, Luke Shaeffer implemented a physically universal CA ruleset in Golly, based on interactions between moving particles [64].

CBlock3D is a 3D cellular automata environment governed by logical and kinematic rules developed by William Stevens [72] [75]. The kinematic simulation in CBlocks3D allows for motions along discrete steps of a regular lattice. Like Golly, it implements a version of hashlife optimization [73] to speed up simulation. Stevens constructed and simulated a self-replicating machine within this design environment from 5040 cells of 6 distinct types (Fig: 2-3) [74].

2.4.2 DNA Origami-Based Tools

CaDNAno is a DNA origami design tool originally written by Douglas et al. [21], and later adopted by Autodesk as a plugin for Maya. It allows users to design 2D and 3D DNA origami structures based on one or several "scaffold" strands folded into a particular shape by many "staple" strands. Structures are designed on a regular square or honeycomb lattice, though, single nucleotide insertions and deletions can be added to create programmable, off-lattice bending [20] [45].

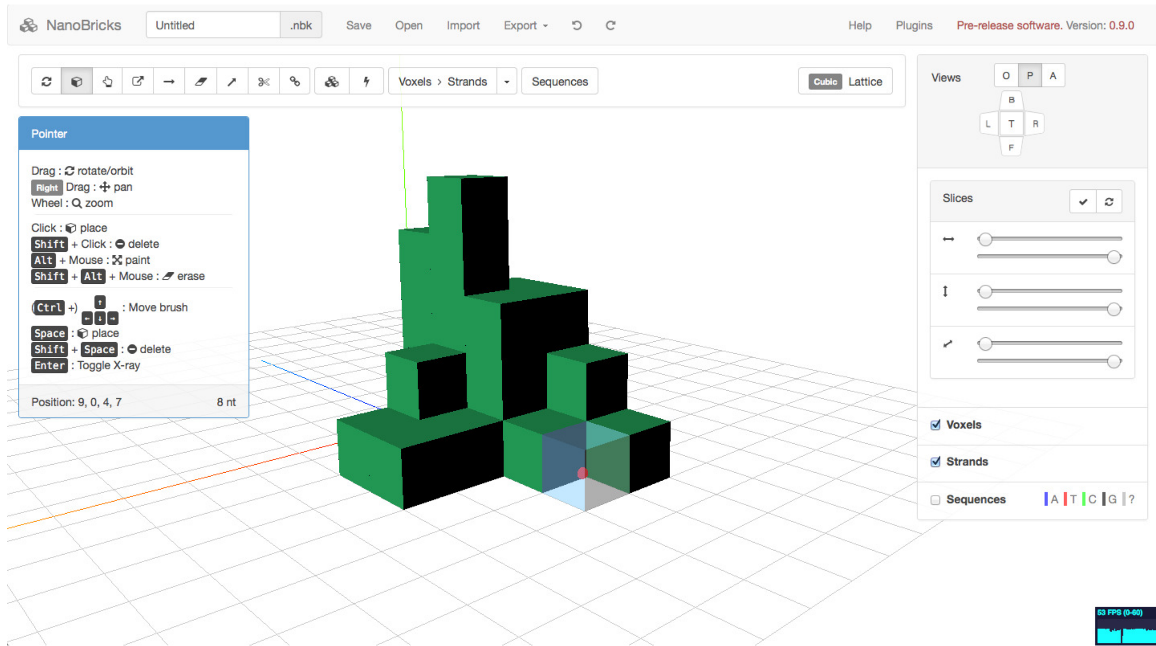


Figure 2-4: Screenshot of NanoBricks, a voxel-based design tool for DNA Bricks by the Peng Yin Lab.

Cando is a DNA origami simulation package developed and maintained by Mark Bathe's group at MIT. It models double stranded DNA as a homogeneous elastic rod with elastic constants and other physical parameters drawn from empirical measurements [52]. Crossovers between double stranded segments are modeled as elastic constraints on the rod elements. Though these crossovers may deform double stranded segments out of a regular lattice configuration (which they are typically designed in) to form complex 3D geometries, simulated CanDo results show good agreement with experimental results [44]. CanDo supports formats from CaDNAno and other DNA design environments.

Nanobricks is a voxel-based design tool for DNA Bricks. Nanobricks allows a user to design nano-scale structures with voxels on a cubic lattice; voxel-based designs are converted to sequences and exported as a text file. Though Nanobricks offers less flexibility than caDNAno (e.g. it does not allow for off-lattice bending designs), it is significantly easier to design valid structures for a novice user. Direct integration with CanDo is forthcoming.

2.4.3 Physics-Based Tools

Voxcad is a physics-based design and dynamic simulation environment by Jonathan Hiller where a user designs virtual soft robots from four block types - two active and two passive [37]. Though the passive block types descended from a simulation of multimaterial 3D printed voxels, the active block types are not easily fabricated and

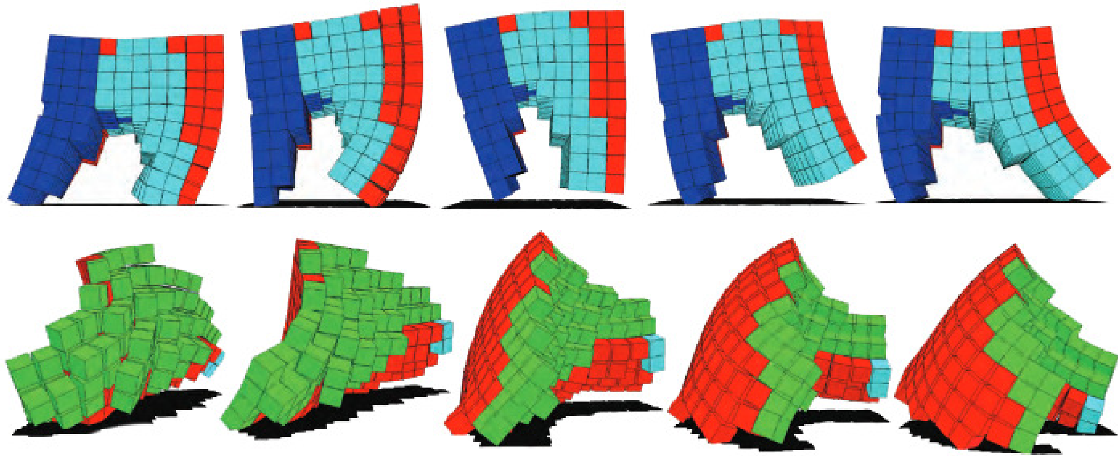


Figure 2-5: Example gaits of two locomotion robots built from four material types in VoxCAD [17]. *Image Credit: Cheney et al. 2013*

actuated [36] and have not been rigorously evaluated. Work into the optimization of theoretical locomotion systems in this virtual design space have been explored (Fig: 2-5) [17] [15] [16].

Chapter 3

Software Workflows for Digital Materials

DMDesign is a CAD/Simulation/CAM environment for digital materials I've built that supports the current research efforts into part and assembler design at CBA. In DMDesign, users construct multimaterial lattice assemblies in a virtual 3D environment, simulate their physical properties, plan out an assembly process, and communicate in realtime with assemblers to physically realize the design [46]. A graphical overview of these capabilities is shown in Figure 3-1.

DMDesign was built to simplify the process of designing and manufacturing digital material assemblies across a range of scales, application spaces, lattice topologies, and assembly processes. This chapter introduces some of the design principles behind DMDesign, namely, how abstraction is used to support a diversity of forms and functions.

3.1 Design Abstraction

As described in Chapter 2, digital materials are discrete parts that interconnect to form a regular, periodic lattice. Accordingly, the DMDesign internals assume an underlying discrete representation of geometry. The core of the CAD interface prompts users to arrange cells into 3D lattice *assemblies*. The particular lattice topology used in the CAD workflow is application specific; several available lattice types are depicted in Figure 3-2. At the time of writing this, a total of 11 different lattice topologies are supported in DMDesign.

The fundamental unit of geometry in DMDesign is a lattice cell, both in its outward facing CAD interface and its internal data representation. This may seem strange since the assembly process happens at the granularity of a part, which may be different than a cell in terms of its geometry and connectivity on the lattice. This abstraction was used to simplify the design interface and allow for a separation of the underlying geometrical representation of an assembly from its decomposition into parts.

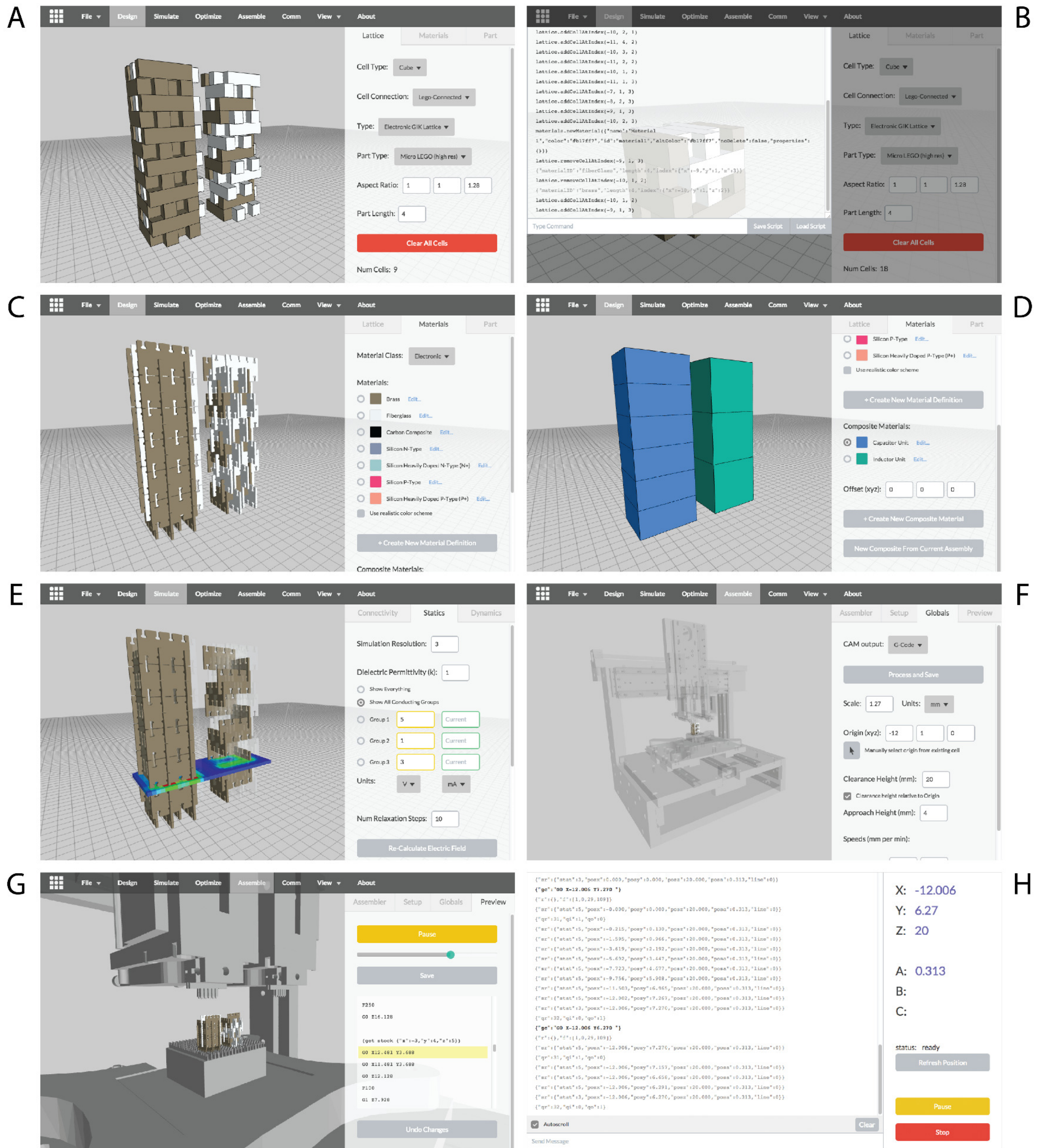


Figure 3-1: Screenshots of DMDesign, a CAD / Simulation / CAM software package for digital materials. Structures are designed from multiple materials in a hierarchical **(D)**, 3D CAD interface or scripting API **(B)** that toggles between a “cell” and “parts” design representation **(A, C)**. Simulation tools include mapping the potential field around arbitrary configurations of conducting and insulating parts **(E)**. CAM interface includes path planning and visualization of the assembly process **(F, G)** and serial interface with machine **(H)**.

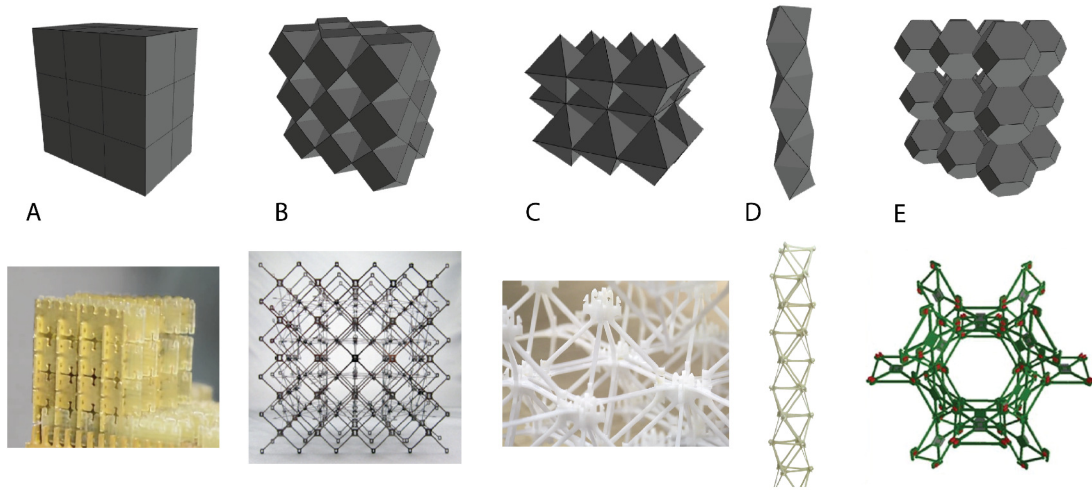


Figure 3-2: A variety of lattice geometries and their corresponding digital material assemblies. From left to right: cubic lattice with electronic GIK assembly **(A)**, cuboctahedron lattice with carbon-fiber composite assembly **(B)**, edge-connected octahedron lattice with laser cut delron voxel assembly **(C)**, face-connected octahedron lattice with injection molded glass-filled nylon truss **(D)**, and Kelvin lattice with electronic "phoxel" assembly **(E)**.

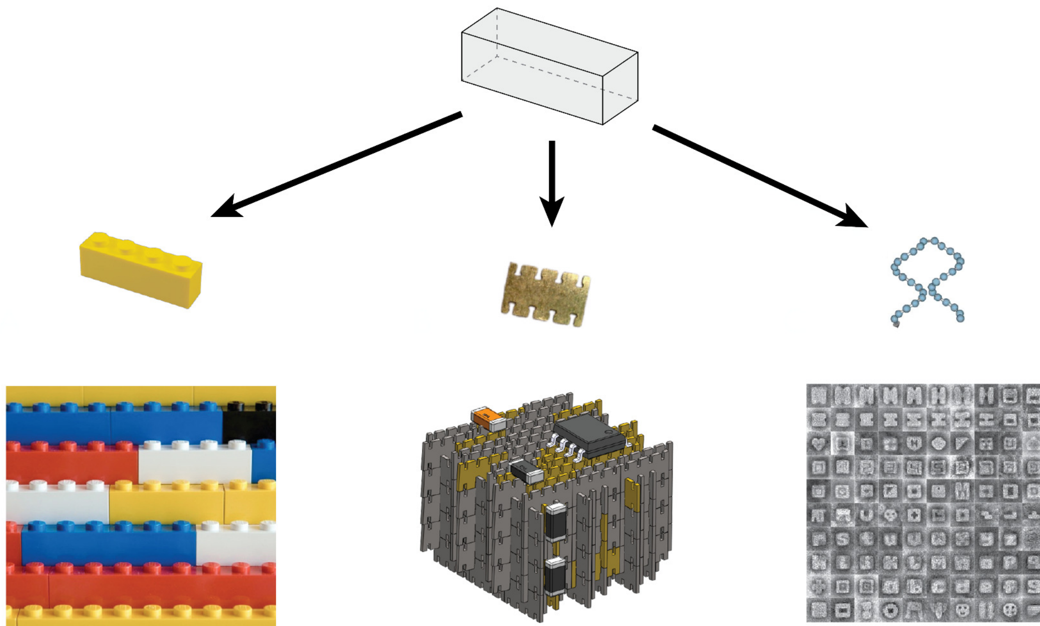


Figure 3-3: Abstraction of cell representation from physical part geometry. In the "lego"-connected cubic lattice, cells are placed in an an assembly with no notion of the physical parts they represent. For example, a cell could represent a piece of LEGO, an electronic GIK part, or even a single strand of DNA in a DNA Bricks assembly. *Image Credit: Will Langford (GIK Images), Ke et al. (DNA Bricks)* [41]

Design abstraction maximizes portability of a single assembly definition across different processes and parts. Figure 3-3 shows how a single lattice cell represents a variety of potential part types. Different joining strategies may require breaking up the same underlying lattice geometry into different repeating units. For example, a vertex-connected octahedron lattice may be broken up into interconnected X-shaped parts, square-shaped parts, or even 3D octahedral voxels.

Additionally, cells are a more useful unit of geometry as we move into simulation. Parts that have been joined together to form a complete, constrained cell add rigidity to the structure. In simulation we are typically concerned only with the behavior of these constrained, rigid elements of the lattice, as is the case for many high-performance aerospace applications.

3.2 Design Hierarchy

Design hierarchy is the establishment of parent/child relationships between elements of a design. In the context of DMDesign, hierarchical data structures allow for a more sparse description of assemblies that contain many similar regions. Hierarchical structures within DMDesign are called "composite cells".

Composite cells are essentially sub-assemblies of cells, illustrated in Figure 3-4. A composite cell definition may contain cells of composite or non-composite types. Each composite cell type is defined once, and all instantiations of a composite cell type within an assembly point to the same composite cell definition.

The hierarchical data structure used in DMDesign is depicted graphically in Figure 3-5. In this structure, cells are stored in a minimally-sized 3D array whose upper and lower bounds change as the bounds of the assembly grow and shrink. Each composite cell type has a definition that includes its own 3D array of cells. This structure allows for a potentially very concise description of assemblies with many regions of self-similarity. In order to render an assembly to the screen, a lattice data structure must be parsed recursively until all composite definitions have been fully evaluated down to their lowest level of description.

Along with a concise design description, hierarchy offers other advantages. Changes to a composite cell definition propagate out to all the instances of that composite cell within an assembly, including instances contained within any parent composite cell definitions. This means users can rapidly design large assemblies of cells and achieve some basic parametric control over their composition. Hierarchical delineations often correspond to behavioral compartmentalization within a design. Though not yet explored in this work, it would be interesting to leverage hierarchical design structure in simulation.

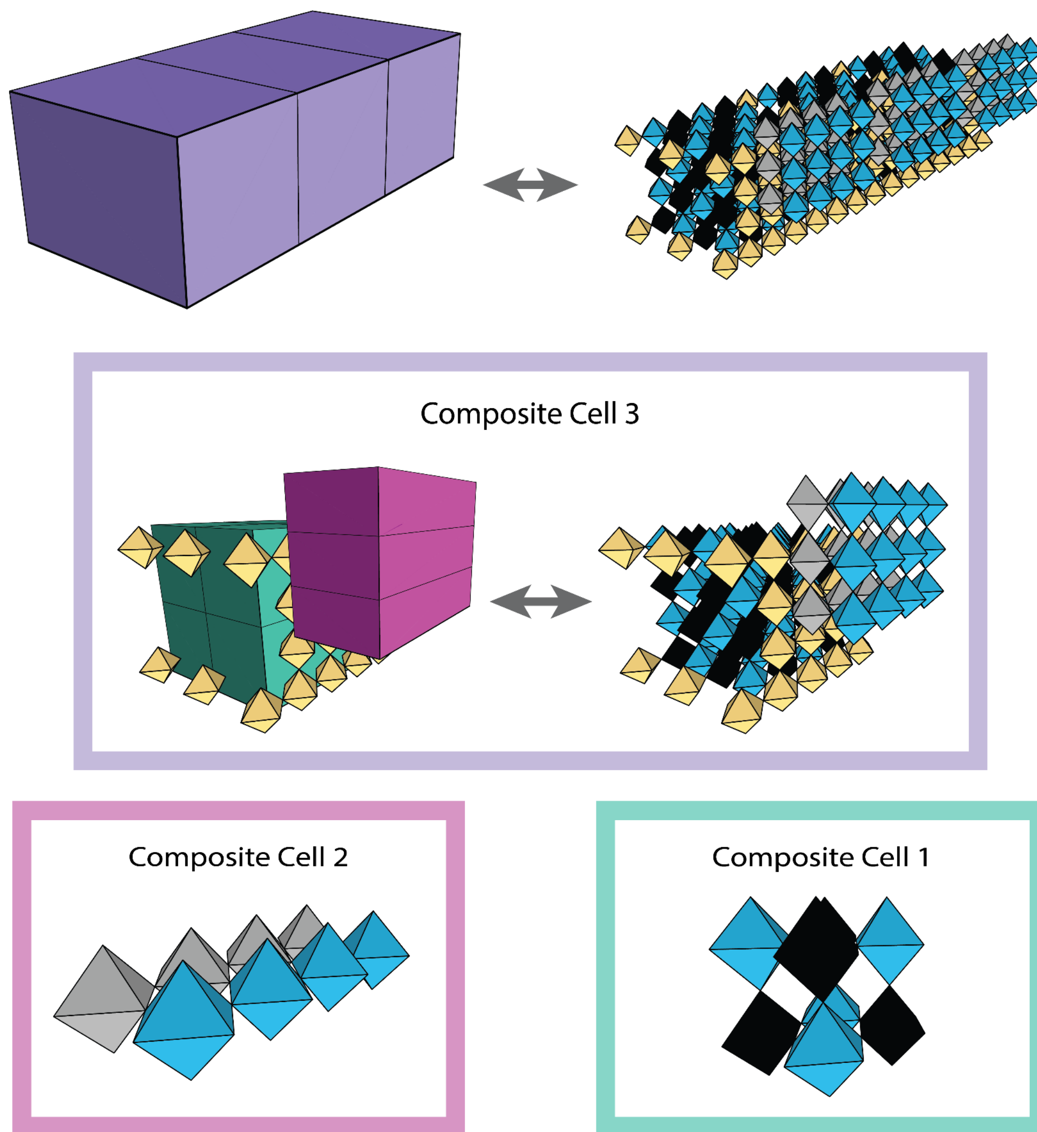


Figure 3-4: Hierarchical assembly of a vertex-connected octahedron lattice. Full assembly depicted in both hierarchical view and cell view (top, left and right). Composite cell definitions 2 and 1 each contain two cell material types (bottom, left and right). Composite cell definition 3 contains a mixture of composite cells and non-composite cells, depicted in both hierarchical view and cell view (middle, left and right).

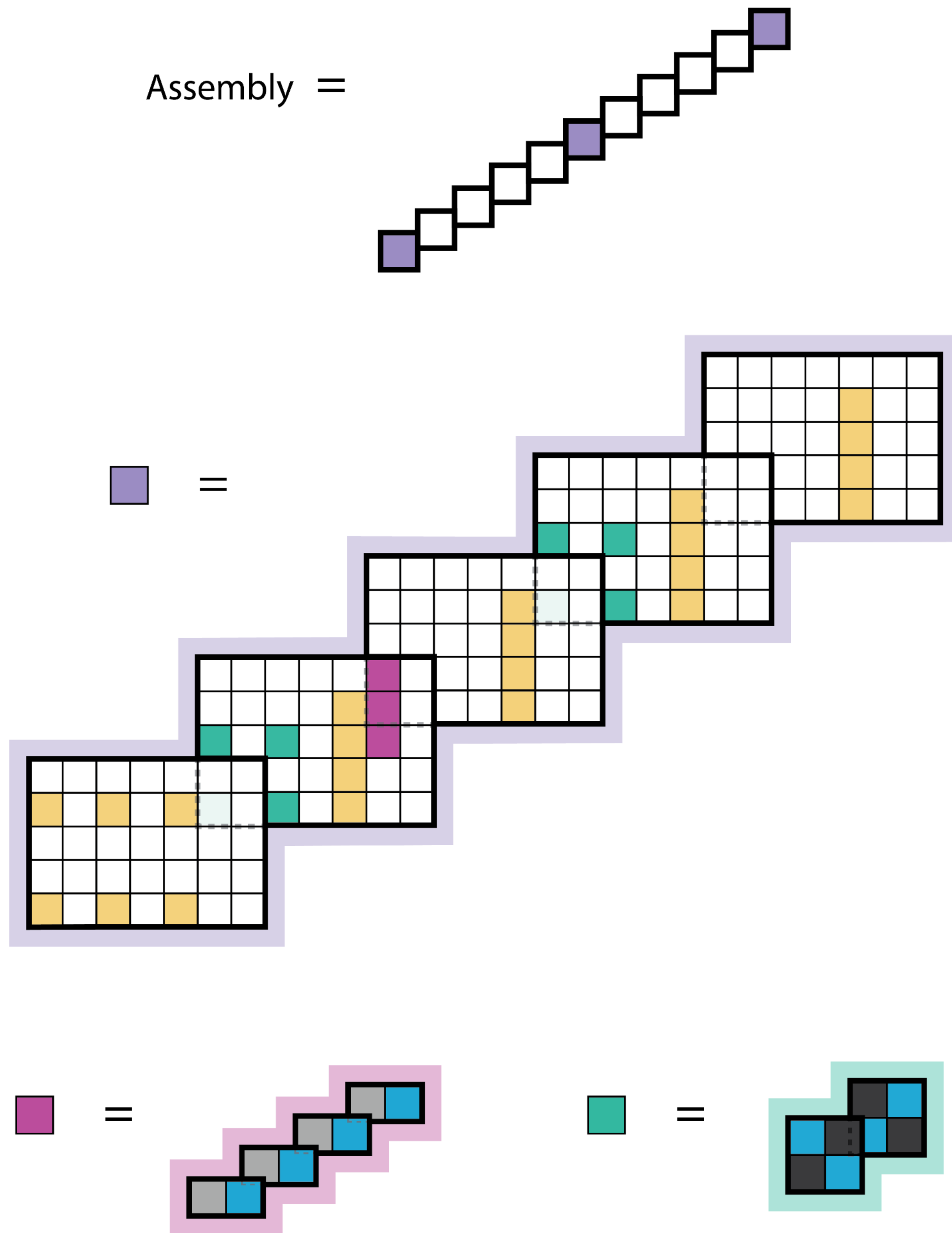


Figure 3-5: Sparse, hierarchical data structure behind the lattice assembly shown in Figure 3-4. White squares indicate null elements. The full lattice assembly is stored in an 11x1x1 array with only three non-null elements, shown at the top. Within each purple cell (composite cell 3) is a 5x7x5 cell definition, containing two more types of composite cells and one non-composite cell type. Composite cell 2 (bottom left) contains a 4x2x1 cell array and composite cell 1 (bottom right) contains a 2x2x2 cell array, both filled with non-composite cell types.

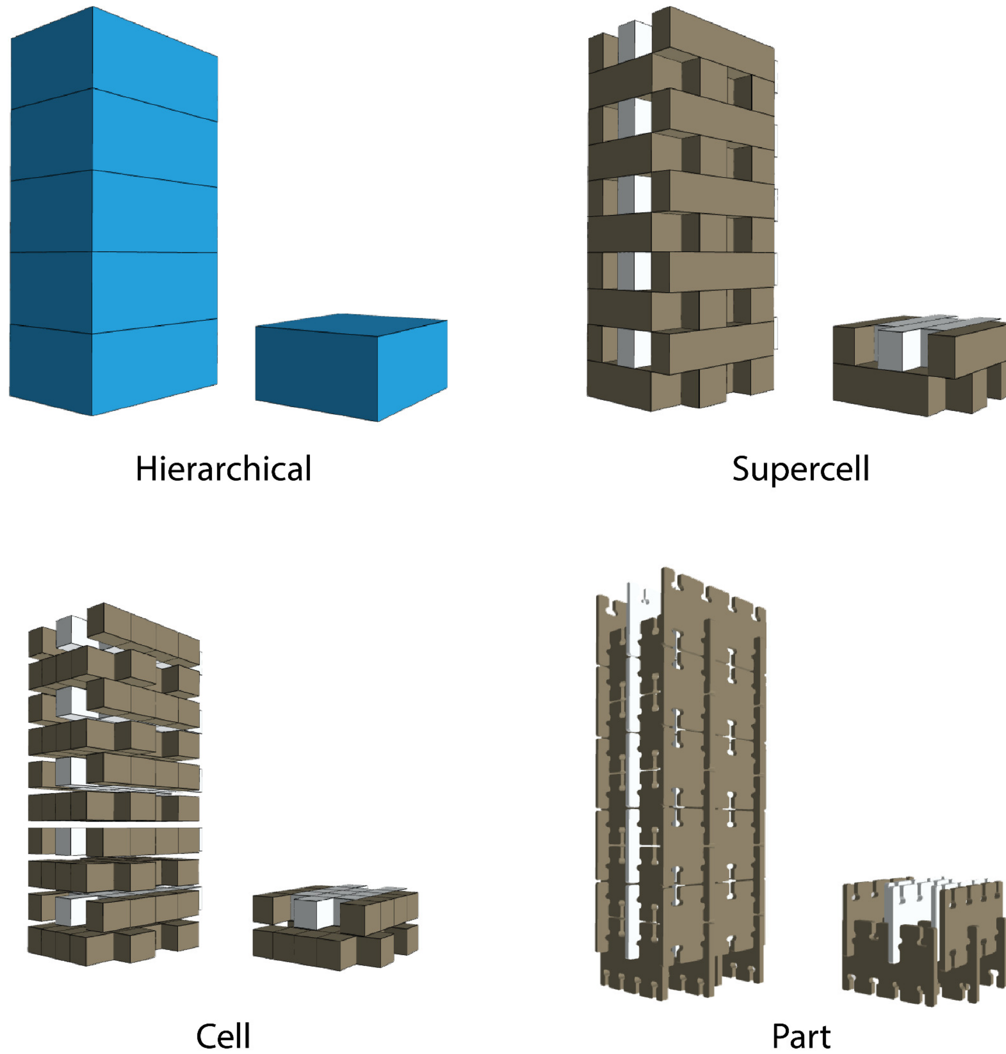


Figure 3-6: Four views of the same capacitive assembly of a GIK Lattice. Nine conducting and insulating GIK parts are arranged to form a capacitive unit structure. A composite cell consisting of this capacitive unit pattern is stacked vertically five times to form a larger capacitive tower. Top left shows the individual composite cells in blue, top right shows the composite cells broken down into their constituent supercells. Each GIK supercell is made from four cells in a cubic lattice, this deconstruction is visualized at the cell level (bottom left). Finally, the underlying physical geometry of the assembly is depicted in the bottom right. Lattices in which the physical part is wholly contained within unit cell do not have the supercell layer of description.

Under certain circumstances, hierarchy manifests itself in one more way. For some applications, the physical part being represented is not contained wholly within a single cell object. For example, in a GIK lattice, one GIK part is made from four cells on a cubic lattice. In order to enforce valid GIK designs and allow the user to control the grouping of cells on the lattice into GIK parts, cells are organized into a mid-level hierarchical structure called a “supercell” (Figure 3-6). These supercells are the lowest level object that can be placed in the GIK design workflow. Applications for which one cell represents one physical part or a collection of parts do not have the supercell delineation.

3.3 Assembler Abstraction

Due to the variety of lattice topologies and assembly strategies, I've organized the CAM workflow of DMDesign so that users can construct a diversity of machine configurations through a simple interface using a few modular, abstract classes. In addition to general machine configuration, global CAM variables - including lattice pitch, machine origin, clearances, and feed rates - are all accessible through DMDesign (Figure 3-8). Once a user has dialed in their settings, a “stock simulation” of the assembly process may be visualized within the software, showing a complete run through of all g-code or other machine commands with a side by side 3D visualization of the assembler. Finally, using a nodeJS interface, machine commands can be sent directly from DMDesign to the assembler.

At the time of writing this, three different machines have been used to assemble parts through DMDesign, depicted in Figure 3-7. The “Batmen” assembler (its name deriving from a portmanteau of its creators - Ben Jenett and Matt Carney) is a 3-axis XYZ gantry with locking feet that locomotes across a lattice while placing octahedral voxels. It is driven by a TinyG controller [77] and consumes standard g-code commands. The Shopbot is a 3-axis XYZ gantry with a custom end effector designed by Ben Jenett to pick and place the same octahedral voxels as Batmen. The Shopbot runs off a simplified g-code-like protocol called shopbot protocol (OpenSBP) [65]. The “Stapler” assembler, designed by Will Langford, is a 4-axis XYZ gantry with a rotary stage that assembles GIK electronic components in an alternating “lego” configuration on a cubic lattice. Like Batmen, the Stapler runs off a TinyG controller. Many more machine configurations are theoretically supported by DMDesign.

Custom machine configurations are stored in a compact JSON format based on the Universal Robot Description Format (URDF) [59]. In this description, “links” and “joints” are connected to each other through parent/child relationships (Figure 3-9). Links are the rigid bodies in a machine; each link contains information about its kinematic properties and any meshes that might be associated with it (Figure 3-10). Joints describe the constraints that govern the motion between two links and the range of motion available. From this information, the inverse kinematics of the machine can be calculated and used to generate a valid path planning strategy.

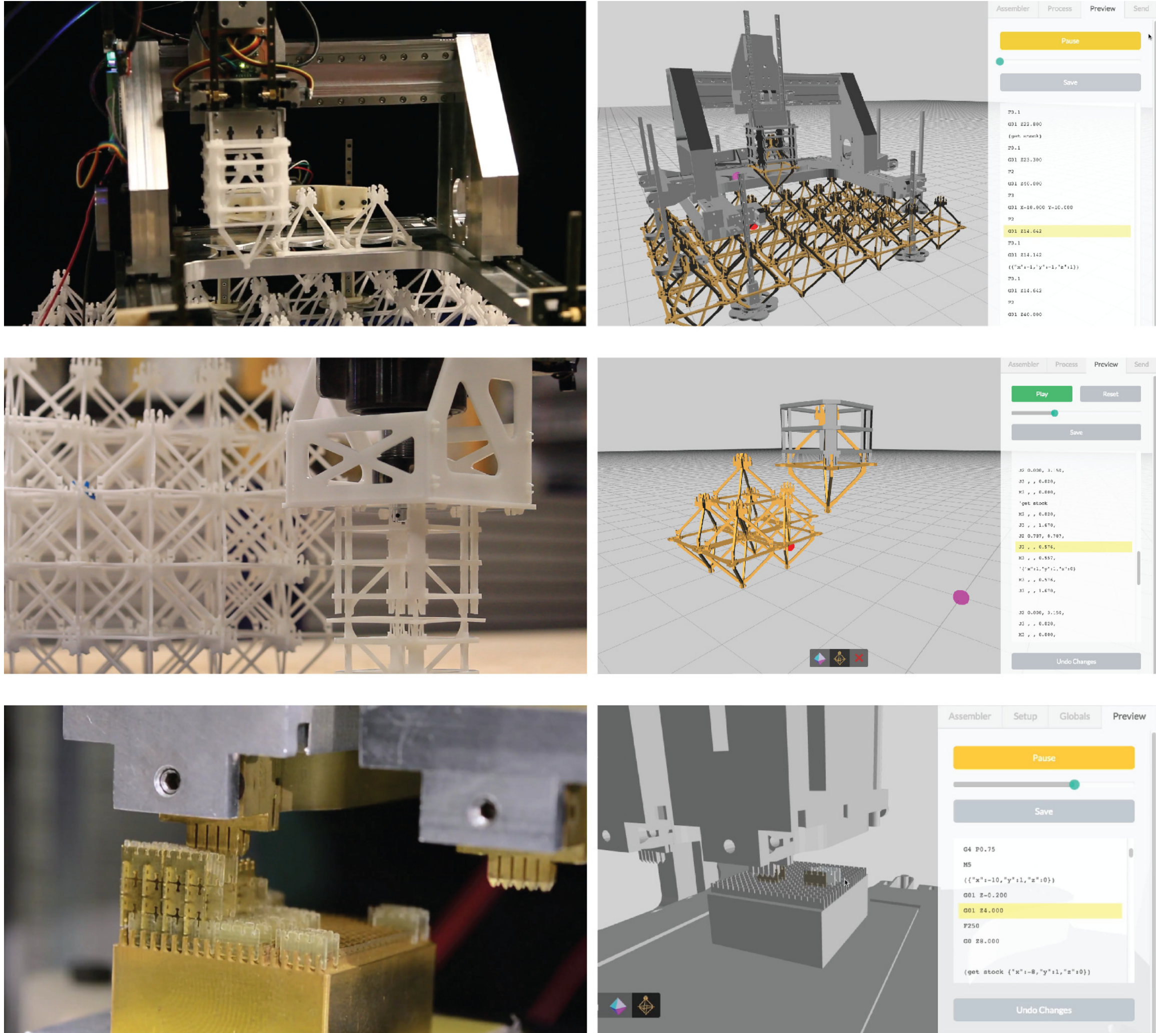


Figure 3-7: Three assembly processes controlled by DMDesign. Top shows assembly of octahedral voxels in an edge-connected octahedron lattice by the "Batmen" assembler by Ben Jenett and Matt Carney. Middle shows assembly of the same octahedral voxels by a 3-axis shopbot with custom end-effector designed by Ben Jenett. Bottom shows assembly of conducting and insulating GIK parts on a cubic lattice by the "Stapler" assembler by Will Langford. *Image Credit: (top left) Matt Carney 2015 and (bottom left) Will Langford 2015*

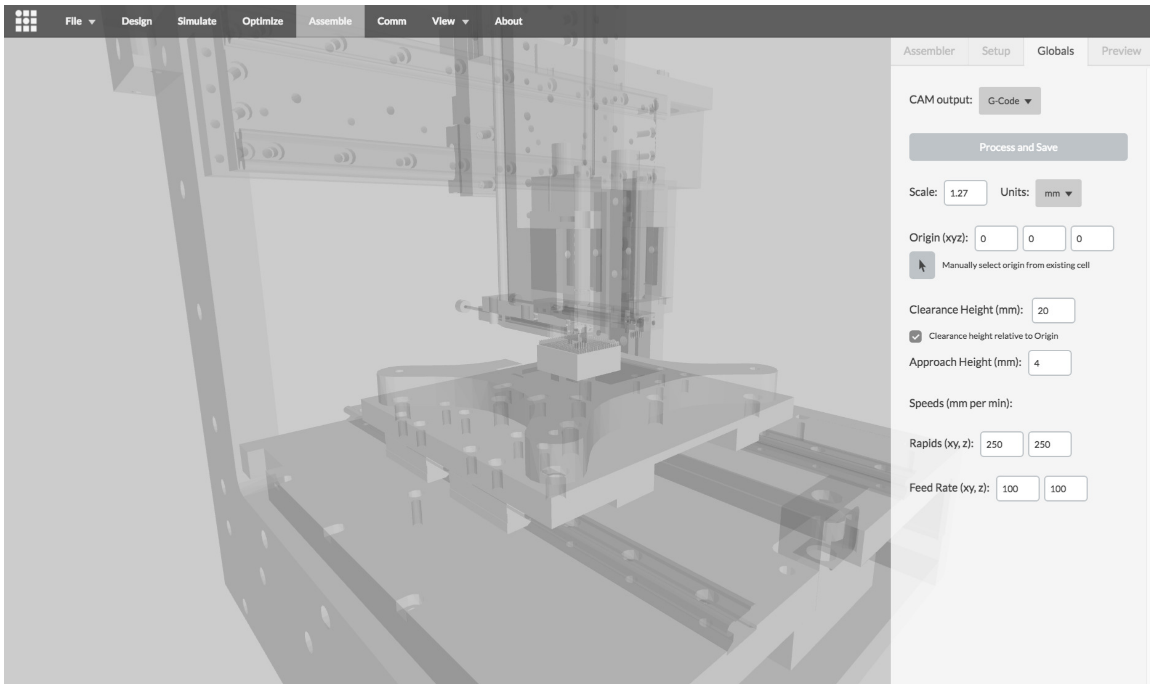


Figure 3-8: Interface for defining global assembly variables such as lattice pitch, machine origin, clearances, and feed rates.

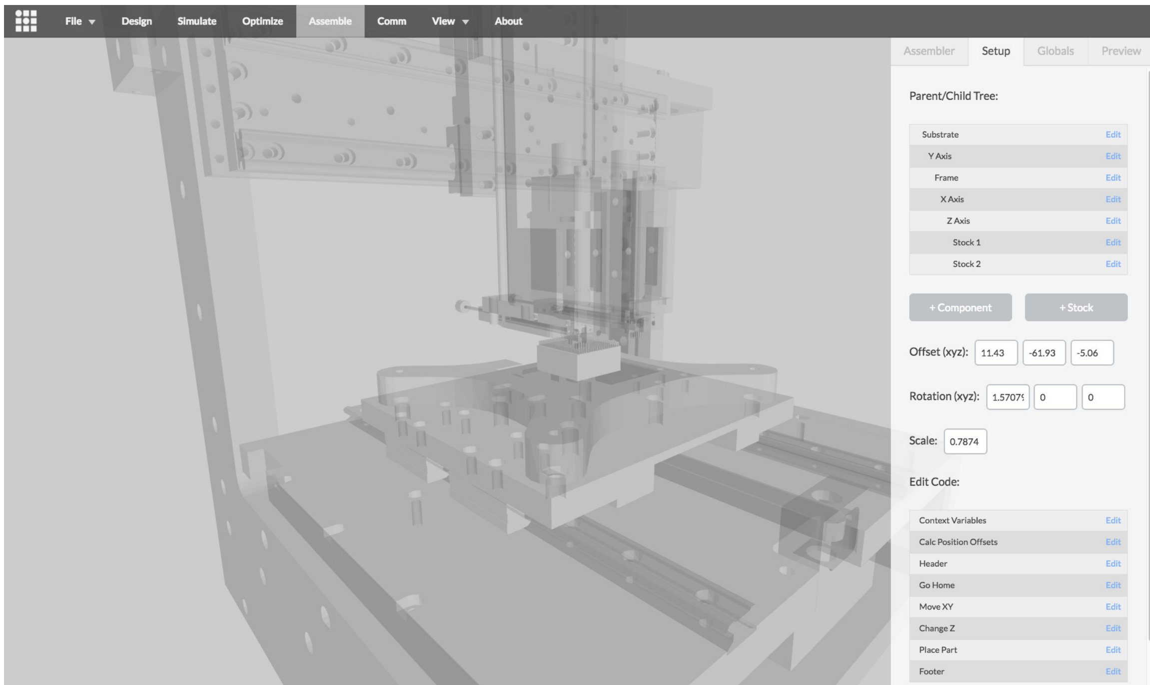


Figure 3-9: Machine configuration interface allows users to edit and save their own custom machine configuration files. Using a format similar to the Universal Robot Description Format, machine kinematics and constraints are described by a series of links and joints connected to each other through parent/child relationships. The parameters of the machine description are used to calculate inverse kinematics.

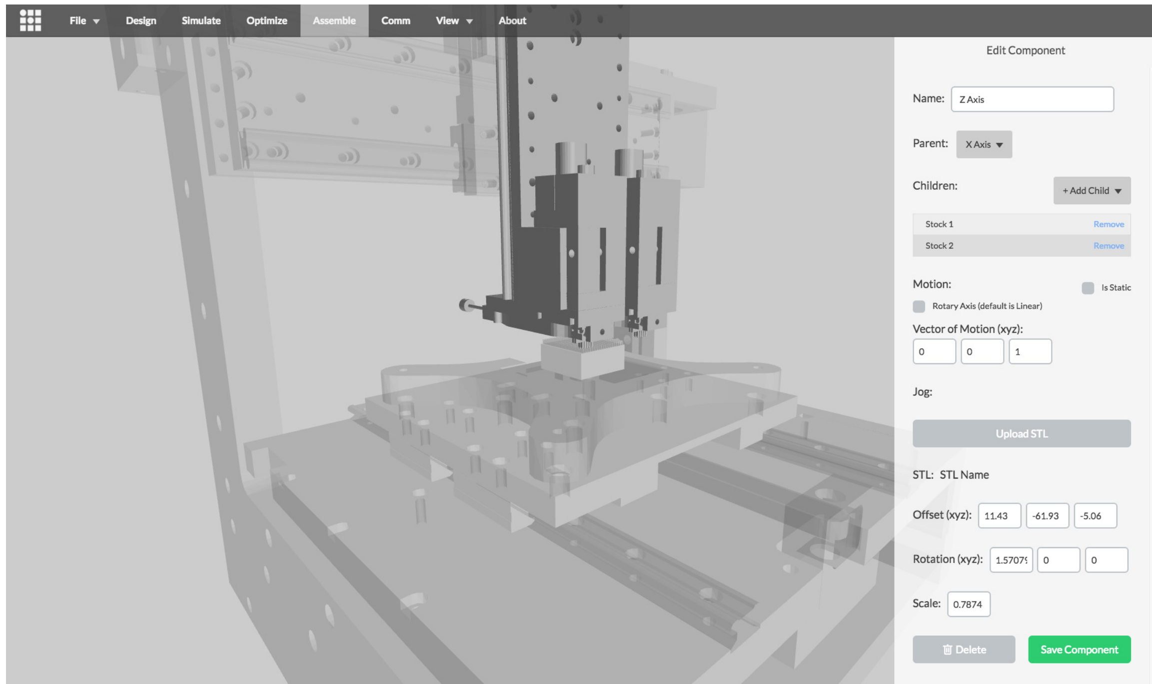


Figure 3-10: Another view of the machine configuration interface shows a detail view of the setup for the "Stapler" assembler z-axis.

Unique machine configurations often require special control logic to address the idiosyncrasies of a particular assembly process. In order to accommodate a wide variety of potential machine configurations (and give more power to the end user) I've created a set of user-definable functions that can be edited and saved within DMDesign. Users can define their own environmental variables and construct custom functions to handle traversals, picking up and placing parts, and headers and footers. Within each of these functions, information such as current part index, position, material type, and any user-defined variables are available. To date, this feature has been used for backlash compensation and multimaterial head placement of the "Stapler" assembler. Eventually, this capability could grow to support closed-loop control between the machine and DMDesign for on-the-fly path planning or error correction.

3.4 Off-Ramps

The intention behind DMDesign is not to replace all existing design and manufacturing workflows, but rather, to create an environment that better accommodates and, at times, leverages the discrete construction of digital materials. In some cases, users will want to interface between DMDesign and other CAD, simulation, and CAM processes.

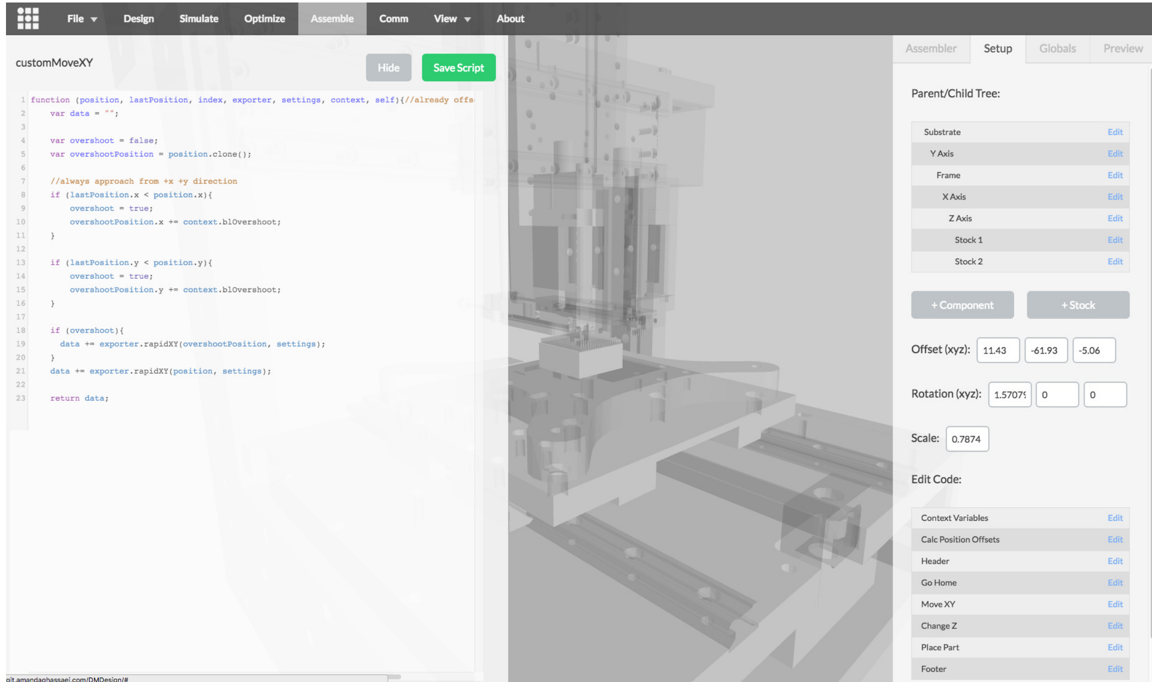


Figure 3-11: Scripting interface for custom machine configuration. This screenshot shows a user-defined function that calculates XY traversals with added backlash compensation specific to the "Stapler" assembler. Other user-defined functions are listed in the bottom right, including: context variables, header, place part, footer, and others.

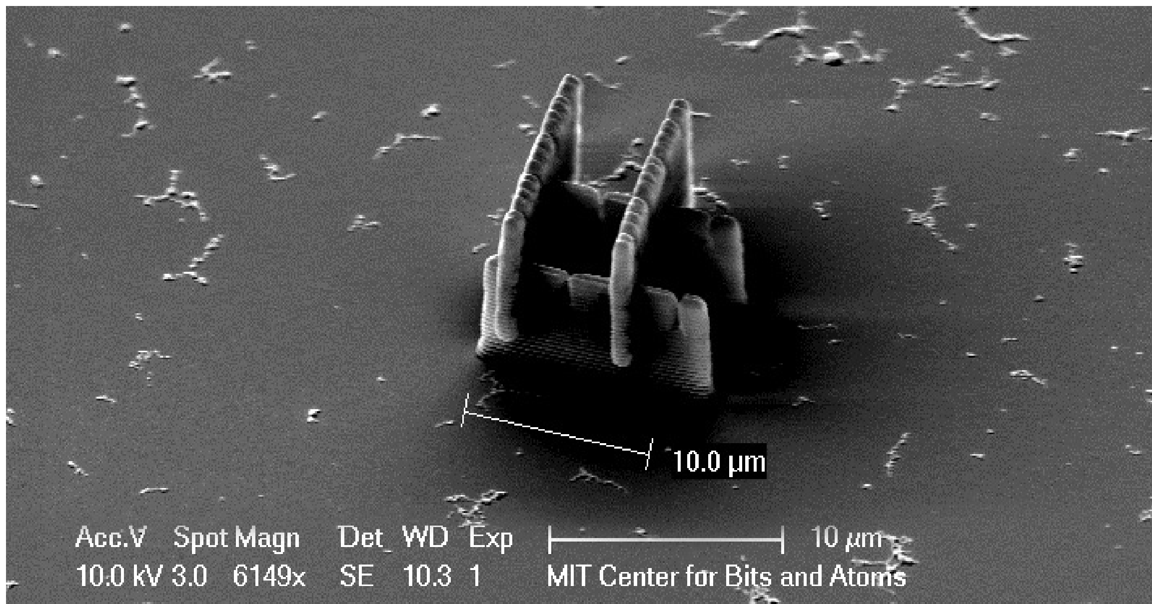


Figure 3-12: SEM image of GIK assembly fabricated on a Nanoscribe Photonic Professional GT using a technique called "two-photon lithography" to a resolution of 150nm. Designed in DMDesign, with geometry exported as STL mesh. *Image Credit: Prashant Patil 2015*

Currently, DMDesign supports STL export of assembly geometries. Users can export geometry by material, and choose between the abstracted cell meshes or the more detailed part geometry. These STLs have been used in 3D printing workflows, including nanoscale 3D printing (Figure 3-12), photorealistic rendering in Blender, and as reference geometry for simulations in COMSOL.

Chapter 4

Digital Materials and Self-Assembling Assemblers

The future research trajectory of CBA explores ways to build self-assembling systems from micron-scale electronic and mechanical digital materials. We imagine this work will involve new part types with higher-level functionality than the bulk material parts discussed in Chapter 3. Building on previous work with DMDesign, I'm creating CAD and simulation tools around these more functionally-dense assemblies of parts. My aim is to provide a means of virtually exploring this new design space through realistic, physically-based simulation techniques. The remainder of this thesis discusses:

- Nomenclature and hierarchical scaling of complexity in self-assembling systems (this chapter)
- Simulation methods (Chapter 5)
- Progress toward a CAD/simulation tool for self-assembling assemblers (Chapter 6)
- Comparison with current state-of-the-art simulation techniques and tools (Chapter 7)
- Future work in this research roadmap (Chapter 8)

Self-assembly is particularly interesting in the context of digital assembly because it provides a means of exponential scaling. For example, if we image $1\mu m^3$ digital material feedstock assembled together in a serial assembly process, the time it would take to assemble something on the order of $1m^3$ would be approximately:

$$assembly\ time\ (seconds) = 10^{18}/f$$

where f is the frequency of assembly in Hz.

Assuming a 10Hz assembly rate, the total time required is 10^{17} seconds, or about 3 billion years. Clearly, this process only becomes tractable through the coordinated

assembly of millions or billions of serial assemblers in parallel. In order to produce assemblers in this quantity, the assemblers must be made from their own feedstock so that one assembler can construct a working replica of itself. Given the ability to duplicate itself and sufficient feedstock, it would only take 30 duplication cycles for one assembler to spawn more than 1 billion child assemblers.

In our proposed self-assembly system, we've identified a four-layer hierarchical breakdown of machine components from the full assembler, to its subsystem modules, all the way down to its raw feedstock (Figure 4-1). The four layers of hierarchy are based on a similar hierarchical structure found in analogous biological systems, explained in more depth in Section 4.6. Each level of hierarchy contains machine components of similar size and complexity. This chapter introduces the four levels of hierarchy in terms of the system we plan to build. Section 4.6 and Section 4.7 explore these hierarchical delineations in biological and virtual self-assembling contexts.

It is worth noting that the design and fabrication of this self-assembling system is actively underway at the time of writing this thesis, but currently in the very early stages of development. The details of the fabrication and joining strategies for these physical parts is beyond the scope of this thesis, though current ideas are briefly addressed here. Initial thoughts about length scales are included here, but are meant only as a starting point for discussing these systems.

4.1 Elements

At the lowest level, bulk materials in the form of *elements* are assembled together to form multi-material assemblies. Our notion of "element" is not an atomic element, but rather a homogenous volume of material with characteristic material properties. A finite set of elemental types will be chosen based on desirable physical properties, cost, ease of fabrication, and compatibility with other elemental types and assembly processes.

For example, an aluminum element type would have the properties of being conductive, stiff, and lightweight and a rubber element type would be insulating and flexible. Current elements types of interest are described in Figure 4-2. Other element types may include magnetic and magnetically permeable types, additional actuation types, and thermally conductive and insulating types.

Elemental components will be fabricated to a size on the order of $\sim$1-10μm^3. Some, as yet to be determined, form of joining interface must be incorporated into the geometry of the elemental parts or applied (in the form of an adhesive) to the parts during the assembly process. At this point in time, we envision a permanent joining strategy between elemental parts.

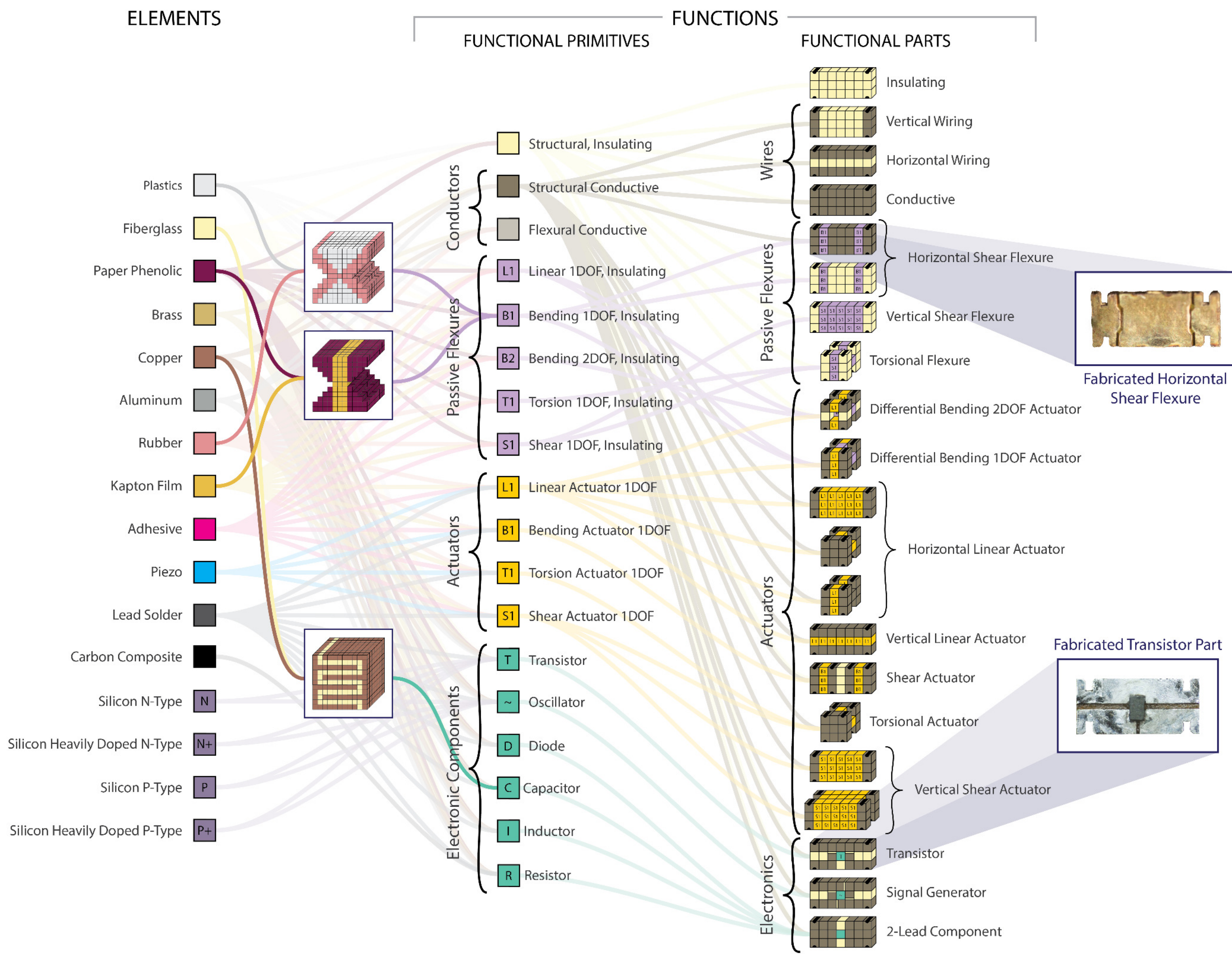


Figure 4-1: Diagram of the hierarchical breakdown of functional parts into functional primitives and elements. Examples of the geometric layout of elements to form functional primitives are indicated for a 1DOF bending flexure and a capacitor. Images of fabricated functional parts are shown alongside their functional primitive decompositions. More detailed views of the transition from elements to functional primitives and from functional primitives to functional parts are shown in Figures 4-2 and 4-3. *Image Credit (for photos of fabricated functional parts): Will Langford 2016*

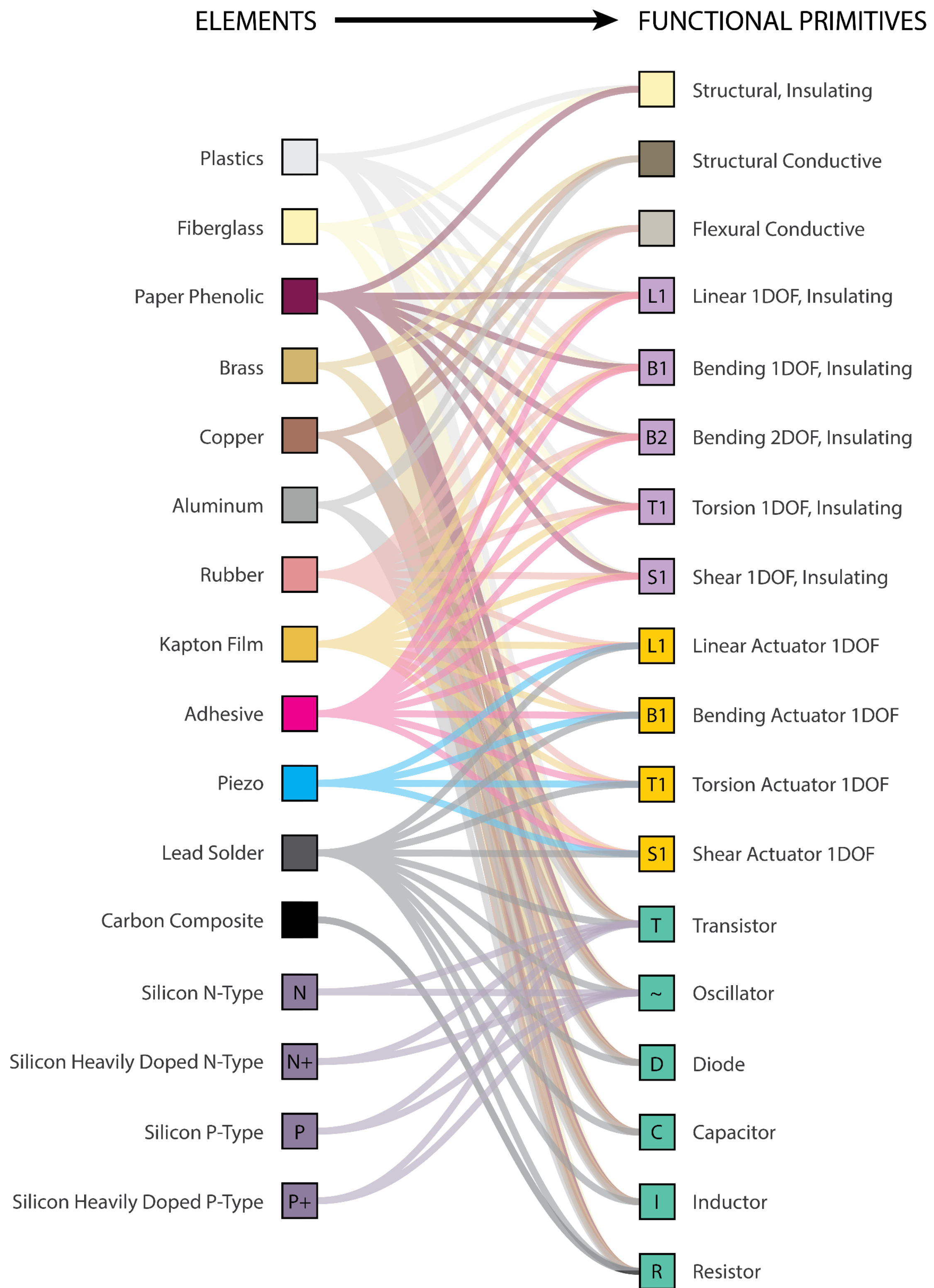


Figure 4-2: Detail view of Figure 4-1, explaining the elemental materials that go into each functional primitive.

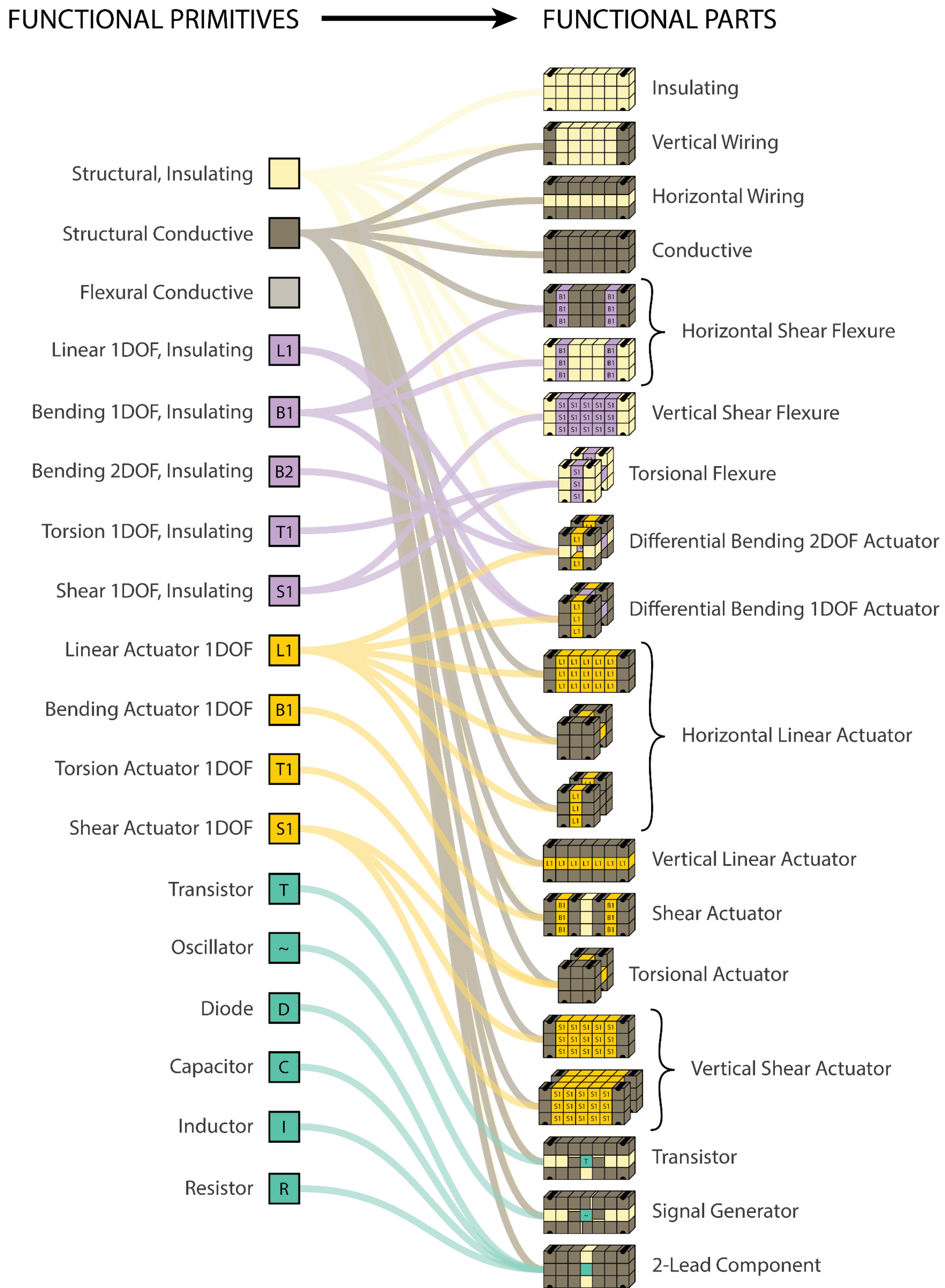


Figure 4-3: Detail view of Figure 4-1, explaining the decomposition of functional parts into functional primitives. Part-part interface of functional parts is indicated in black.

4.2 Functions

Assemblies of elements form *functions* - larger scale part types with material properties defined by their constituent elements. As the name suggests, functions are categorized based on higher-level interactions with neighbors, rather than their bulk material properties.

From a software perspective it makes sense to break down the function category into two layers: *functional primitives* and *functional parts*. Functional primitives are a useful abstraction that serve as the fundamental unit for simulation of functional parts, described in more detail in Chapter 5. Functional parts, on the other hand, are small motifs of functional primitives, complete with interfacing joinery, that form the basis of function-level fabrication and assembly. This distinction is analogous to the cell/supercell delineation used for GIK parts, described in Chapter 3.

Functional primitives fall into one of several functional archetypes, including n-degree-of-freedom mechanical flexures and actuators, and simple passive and active electronic components. These archetypes are represented chromatically in Figure 4-3. The "function" of a functional primitive is determined by the 3D composition of its constituent elements, depicted in Figure 4-1. Anisotropic patterns of elemental types within a functional primitive may give it tunable anisotropic behaviors. A more complete discussion of the diversity of mechanical and electronic part types at the function-level is given in Chapter 5. In the fullness of time, the geometric composition of functional primitives from elements would be a good candidate for constrained topological optimization.

Functional parts are small assemblies of functional primitives, plus a joining interface. Because functional parts are not dramatically more complex than function primitives, their behaviors are more or less the same. A selection of potential functional parts are diagramed in Figure 4-3, though that list is not meant to be exhaustive. Examples include routing components, which pass one or several electronic signals to neighboring parts, anisotropic flexural and actuated parts, and simple passive and active electronic components.

Functional parts are currently fabricated using irreversible, bulk 2D processes on the order of $\sim$1mm^3. Some form of common interface must allow for both mechanical loads and analog electronic signals to pass from one functional part to another. At the function-level, a reversible joining interface is preferred so that reconfigurable behavior and recycling is possible. Currently, press-fit or reversible soldered interfaces are being explored. The first attempts at fabricating functional parts (by Will Langford) is depicted in Figure 4-1.

4.3 Modules

Modules are assemblies of function-level components, joined together to form robotic subsystems. Modules combine electronic and mechanical functionality to achieve singular, high-level robotic tasks. Modules will communicate with each other through an abstracted, digital interface. Each module will own its own microprocessing unit that coordinates its function-level actuators and other active components. This way, the low-level description of the structure within a module remains internal to the module itself.

Examples of modules include end effectors like grippers, clamping mechanisms, and sensors, actuation/locomotion systems, energy storage and generation, and large memory or programmable logic banks. Joining interfaces between modules will allow mechanical forces, power, and a few digital signals to pass from one module to the next. As with functions, modules should be reversibly joined. The interface between modules may consist of many parallel function-level interfaces, or a bulk, module-level interface.

Modules will be built on the order of $\sim$1cm^3. Though modules may use mechanical compliance at the function-level to increase their internal degrees of freedom, at larger scales modules act as rigid bodies with internal motion. That is, internal mechanical compliance within a module does not propagate to interactions with neighboring modules.

4.4 Complexes

Assemblies of modules form *complexes*, self-contained robotic systems that coordinate their own sensing, memory, logic, power, and/or actuation. Though a module should be sophisticated enough to control its own actuators, a complex ties together input and output modules to produce interactive behavior. Instructions may be fed to complexes through their environment in order to direct large scale processes involving many complexes in parallel. In this case, complexes will need to maintain electronic contact with environmental signaling wires. Initial complex design may also depend on power sourced through the complex's environment.

A complex could be a single, locomoting robot or manipulator, or an active, environmental system that performs sensing, actuation, or logical tasks across space. Complexes will interact with function-level and module-level feedstock to accomplish assembly tasks. Many complexes will work together to shuttle these feedstock around space and pick and place them in a directed, programmable fashion.

Complexes will be comprised of $\sim$10-100 individual modules, spanning length scales on the order of tens of cm to m. Complexes should be considered independent, self-contained units with quantized motions. The interface between complexes is governed

by end effector modules that the complex owns.

4.5 Systems of Complexes

Many complexes of various forms and functions may work together at a *system*-level to accomplish large scale tasks. Additional layers of hierarchy could be introduced to manage these systems. For now, a systems-level discussion is beyond the scope of this work.

4.6 Design Hierarchy in Biology

Currently, the only known physical example of a self-replicating system is the biology that has evolved on Earth. The primary drivers of biological self-replication are protein machinery (called "protein complexes") built primarily from a small set of amino acid building blocks.

The construction of protein complexes takes place in a hierarchical fashion. Figure 4-4 describes the hierarchical breakdown of protein complexes in terms of the four levels of hierarchy established earlier in this chapter; protein complexes (complex-level objects) are decomposed into proteins (modules), then into amino acids (functions), and finally into atoms (elements). The similarity of some of the biological nomenclature to our own hierarchical nomenclature is due to the fact that it was derived from the biological model.

4.6.1 Elements, Functional Groups, and Functions

At the most fundamental level, biological structures are composed of atoms of various elemental types. On Earth, biological molecules are made primarily from combinations of carbon, hydrogen, nitrogen, oxygen, phosphorus, and sulfur (CHNOPS).

Each element's unique position on the periodic table dictates its physical properties, which have implications in higher-level structures formed from the element. These properties include mass, the number of covalent bonds it can form, the number of paired electrons it contains in its outermost electron shell, the polarization of the bonds and molecules it forms with other elements, and the stability of its bonds in various environments.

Certain motifs of covalently bonded atoms form *functional groups* within molecules. Functional groups determine the ability of a molecule to undergo various archetypal reactions (addition, substitution, elimination, etc) with itself or other molecules. A subset of the important functional groups found in biochemistry are indicated in Figure 4-4. "R" indicates an arbitrary side-chain, where the functional group connects to the rest of the molecule.

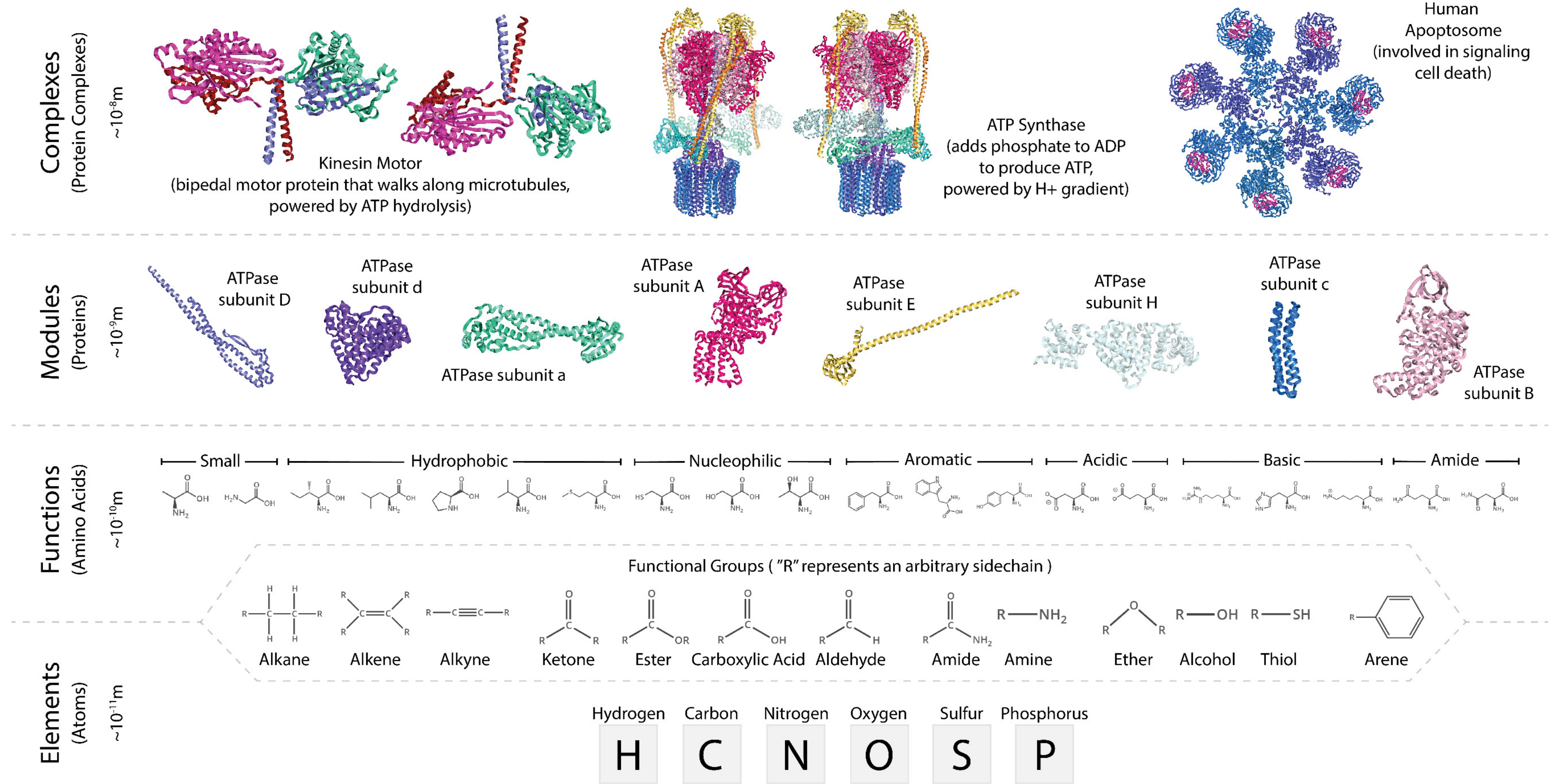


Figure 4-4: Hierarchical breakdown of protein complexes (complexes) into proteins (modules), amino acids (functions), and atoms (elements). 3D renderings of protein complexes and subunits were made in Pymol using data from the RCSB Protein Data Bank [6].

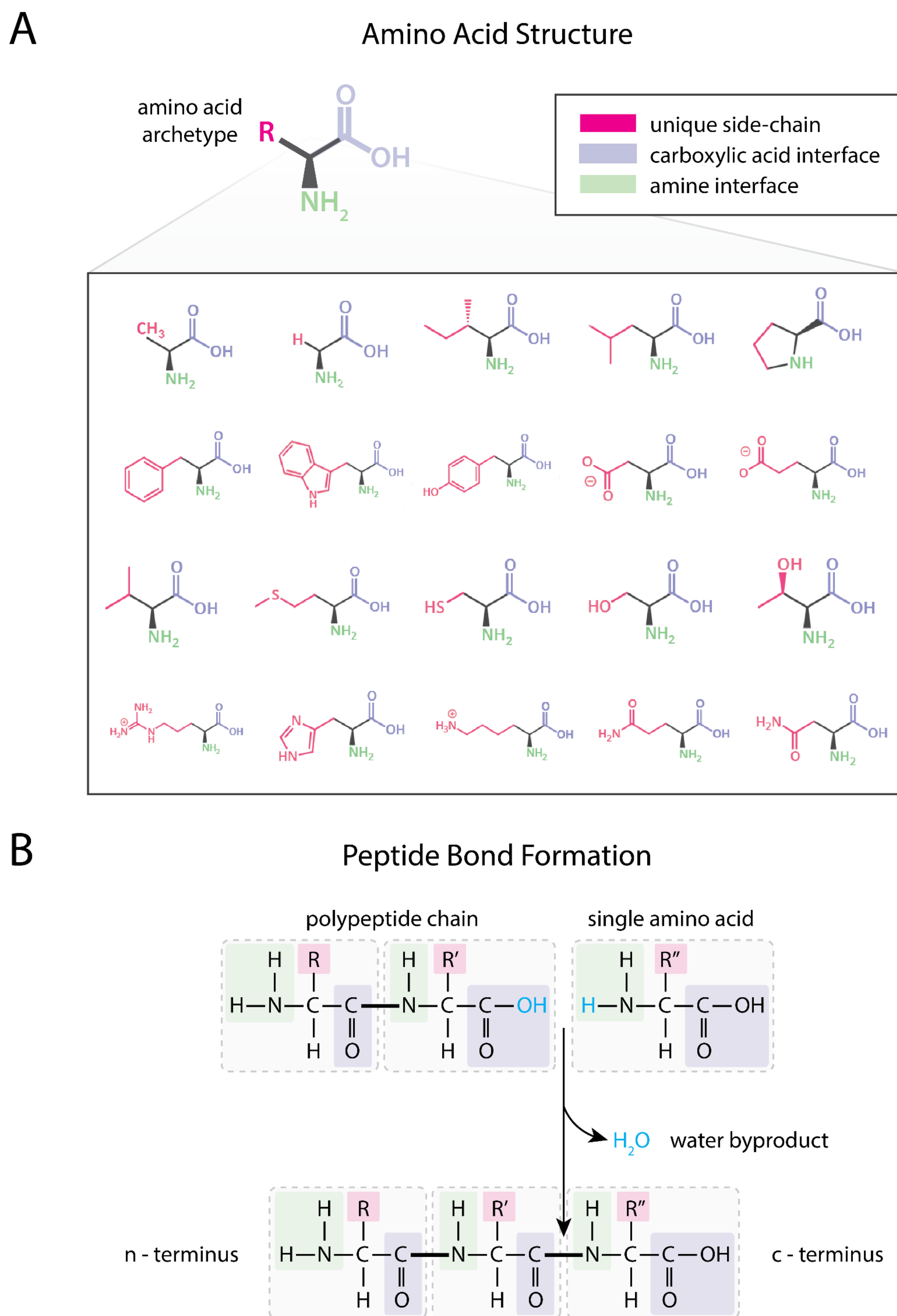


Figure 4-5: (A) Decomposition of amino acids into carboxylic acid and amine interfaces and unique side-chain. (B) Formation of peptide bond at c-terminus (carboxylic acid group) of polypeptide chain with the n-terminus (amine group) of an amino acid. One molecule of water is produced as a byproduct of the peptide bonding reaction. R, R', and R" represent arbitrary amino acid side-chains. Peptide bonds bolded.

Amino acids are molecules which contain several functional groups. As indicated in Figure 4-5A all amino acids contain a carboxylic acid (COOH) and an amine group (NH_2 or NRH). The purpose of these groups is to form the interface between amino acids in a polypeptide chain. Amino acids are attached to each other through a covalent bond, called a "peptide bond", with a single molecule of water created as a byproduct of each bonding event4-5B.

The remainder of the amino acid molecule (indicated by "R" in the amino acid archetype in Figure 4-5A, we'll call it "*the* side-chain") determines its unique properties. Each type of amino acid has a different side-chain, which may consist of one or more distinct functional groups. Amino acids are characterized according to their side-chains in several ways; one grouping is shown in Figure 4-4, with amino acids described by size, polarity, nucleophilicity, the presence of aromatic rings, acidity or basicity, and the presence of amide groups.

4.6.2 Modules and Complexes

Polypeptide chains typically consisting of hundreds of amino acids fold to form three dimensional *proteins*. Hydrogen bonding and other non-covalent interactions between amino acid side-chains, along with external effects, determine the three dimensional structure of the protein. At the protein's active site, amino acid sidechains interact with molecules in the environment. Sometimes, these interactions result in global deformations of the protein's three dimensional structure, called "conformational changes".

Multiple proteins join together through non-covalent interactions to form a *protein complex*. A protein within a complex is usually referred to as a subunit. Protein complexes sometimes contain many identical subunits.

4.6.3 Insights

The main take-away from the biological model is proof of principle. Biology demonstrates how a small basis set of simple, functional feedstock can be reversibly assembled to create more complex structures and mechanisms - ultimately, self-replicating systems. It should be noted, however, that analogies between biological systems and our proposed assembly system are limited due primarily to differences in scale (and therefore, different types of dominant physics), rates of interactions between parts, and mechanism of interaction between parts.

Scaling

In the biological example, each level of hierarchy introduces a factor of about 10x in scaling. The atoms that compose the lowest level of hierarchy have covalent diameters ranging from 50-200pm [69]. Amino acid diameters can be roughly calculated from

atomic radii and three-dimensional structure to a range of 0.42-1.2nm [53]. The average protein length across prokaryotes and eukaryotes is about 200-400 amino acids, with a mass of about 20-40kDa [10]. Assuming a simple spherical shape, this mass translates to a typical protein diameter of about 3-4nm [23]. Known protein complexes are comprised of two to several hundred protein monomers with typical complex diameters ranging from about 8-100nm [84].

Design and Fabrication

Despite the striking complexity gap between biological systems and the nutrients required to sustain them, we should not think of biological systems as making everything they need "from scratch". The raw material feedstock of biology contains some inherent structure and function. Humans only synthesize 11 of the 20 standard amino acids; we must absorb the remaining "essential" amino acids from food. Even considering organisms that do make all their amino acids, biological synthesis pathways begin with reactive precursors rather than exclusively elemental forms of atomic feedstock [76]. Disassembly takes place with similar granularity. Cells break down proteins into amino acids in a process called "proteolysis"; the freed amino acids are then used to construct new proteins. Typically, amino acids are not metabolized into smaller components unless the cell has an excess of amino acids or lacks the glucose needed meet its energy needs [76].

Following this biological model, our lowest-level, reversibly assembled parts should also have some built-in structure and function. As described in Figure 4-5A, we can think of an amino acid as being composed of a unique, functional side-chain and a standard, reversible interface. The functional parts depicted in Figure 4-3 share this same basic structure. I hesitate to draw the same analogy with the element-level, bulk material parts described in Section 4.1. Differences in interaction properties (elasticity, fracture susceptibility) and available manufacturing processes for each of the elemental types listed in Figure 4-2 pose significant challenges to a universal, reversible joining interface between single-material parts.

If we designate functional parts as the lowest-level reversibly assembled parts in our assembly system, interfaces between elements could include permanent adhesives or other chemical bonding techniques. Functional parts could be produced in large-scale, batch processes without a disassembly strategy in mind.

4.7 Design Hierarchy in Conway's Game of Life

In the 1940's Von Neumann began studying the requirements for self-replication and evolvability in artificial, CA worlds [50]. One such world that has gained widespread notoriety is Conway's Game of Life ("Life"). Since its inception in 1970, Life has developed a cult following of researchers, engineers, and hobbyists, pushing each other to construct increasingly elaborate "machines" from pixels on a screen. These in-

vestigations have resulted in a lengthy taxonomy of motifs, reactions, mechanisms, and complex engineered systems within Life. Some notable accomplishments include Gemini (an oblique spaceship encoded by a long glider tape) [80], the Linear Propagator (a self-replicating machine) [31], a universally extensible Turing Machine [57], the Minsky Register Machine (a finite universal computer) [14], and the OTCA Metapixel (a structure that behaves like a large-scale Life pixel) [22].

The analogies between machines in Life and our proposed assembly system are limited, due to the fact that Life is inherently non-physical; it violates basic conservation principals, has no notion of cell joining, and is based on highly abstracted interactions. However, it provides an example of how humans can engineer non-trivial self-replication from extremely simple building blocks. A proof of the existence of self-replicating patterns in Life was first published by Conway, Berekamp, and Guy in 1982 [8], and several decades later the first implementations of replicators began emerging on online Life forums [32]. This work argues that the basic requirements for self-replication - universal computation and universal construction - are not only *not* unique to biology, but are, in fact, quite prevalent in virtual and physical systems. An excellent analysis of Conway's existence proof, as well as an introduction to important concepts in Life can be found in *The Recursive Universe* by William Poundstone [55].

In the next sections, I'll analyze a particularly complex Life pattern in terms of the hierarchies I introduced earlier in the chapter. The OTCA Metapixel was designed by Brice Due in 2006. Though not the first *metacell* (a structure that can mimic a CA cell), it is the first *metapixel* (a structure that looks and behaves like a macro-scale CA pixel) built in Life, and it can be programmed to perform any CA ruleset that's based on a summation of a cell's 8 local neighbors. Many metapixels can be patterned together on a grid to play out a macro-scale CA (Figure 4-6). The following analysis is based off of information on Brice Due's website [22] and by watching the pattern run in Golly; I wrote a more detailed discussion of the inner workings of the pattern on Instructables [27].

(I chose to analyze this pattern because its square aspect ratio makes it well suited for displaying in a static text like this writeup; known self-replicating patterns do not fit onto one piece of paper at a reasonable scale.)

4.7.1 Elements, Motifs, and Functions

The first few levels of hierarchy within Life are illustrated in Figure 4-7. At the most basic level, structures within Life are constructed from *cells*. Each cell is represented by one pixel on a Life grid and stores a binary state - "living" or "dead". Cells interact with each other according to Conway's rules: living cells with 2 or 3 neighbors stay alive, dead cells with exactly 3 neighbors become living, and all other configurations die or stay dead. Though the logic of cell-cell interactions is simple, the long term behavior of a given initial condition is difficult to intuit.

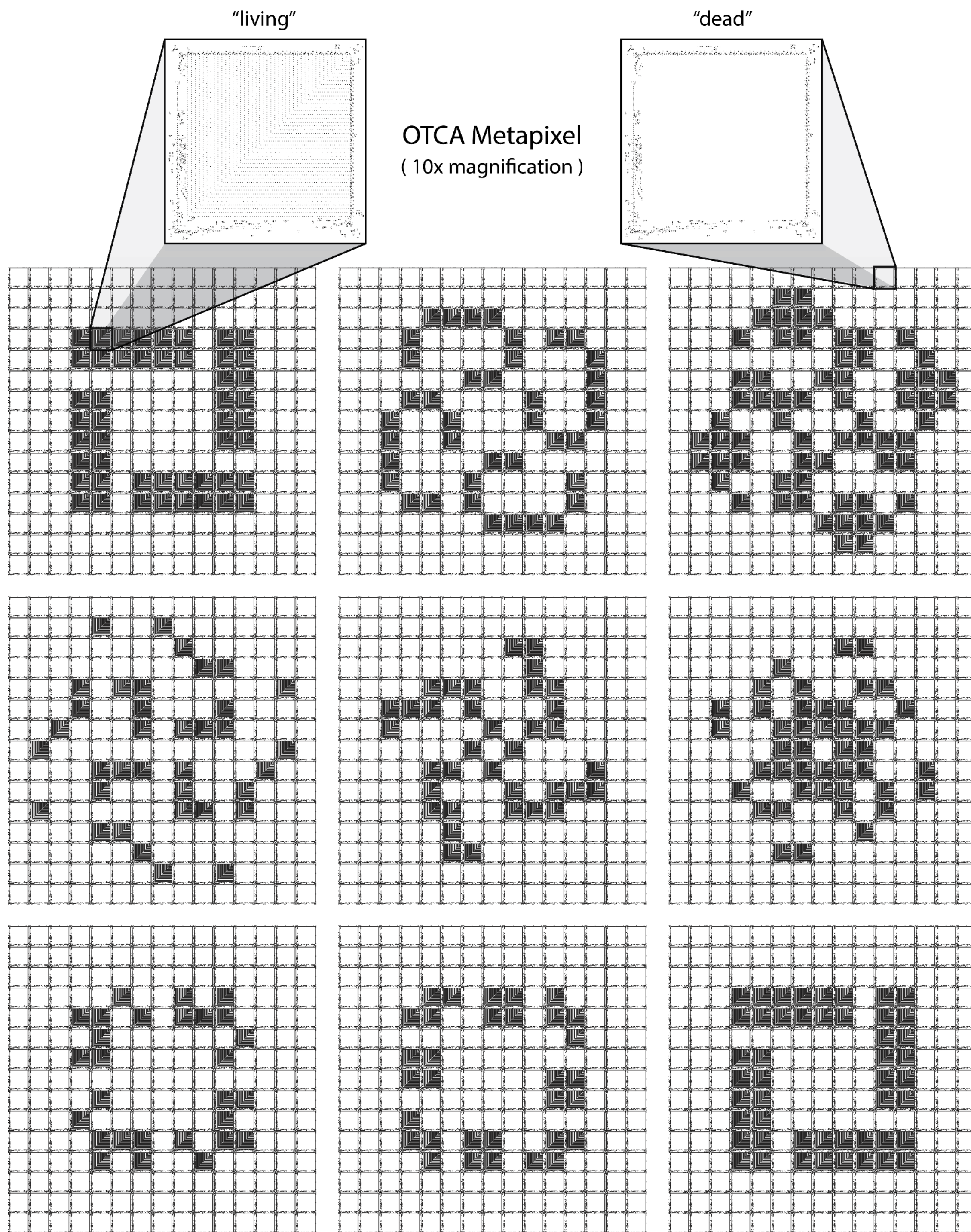


Figure 4-6: Nine timesteps of a 15x15 array of OTCA Metapixels playing out a period-8 oscillating pattern called "Kok's Galaxy". Each time-step represents 35,328 generations of Life; the entire 8 step sequence takes 282,624 generations to complete. Each metacell occupies 2058x2058 Life cells; the complete 15x15 metapixel array totals 30,800x30,800 Life cells (accounting for a 5 cell overlap between adjacent metacells). 10x magnification of a "living" and "dead" cell are shown at the top of the image.

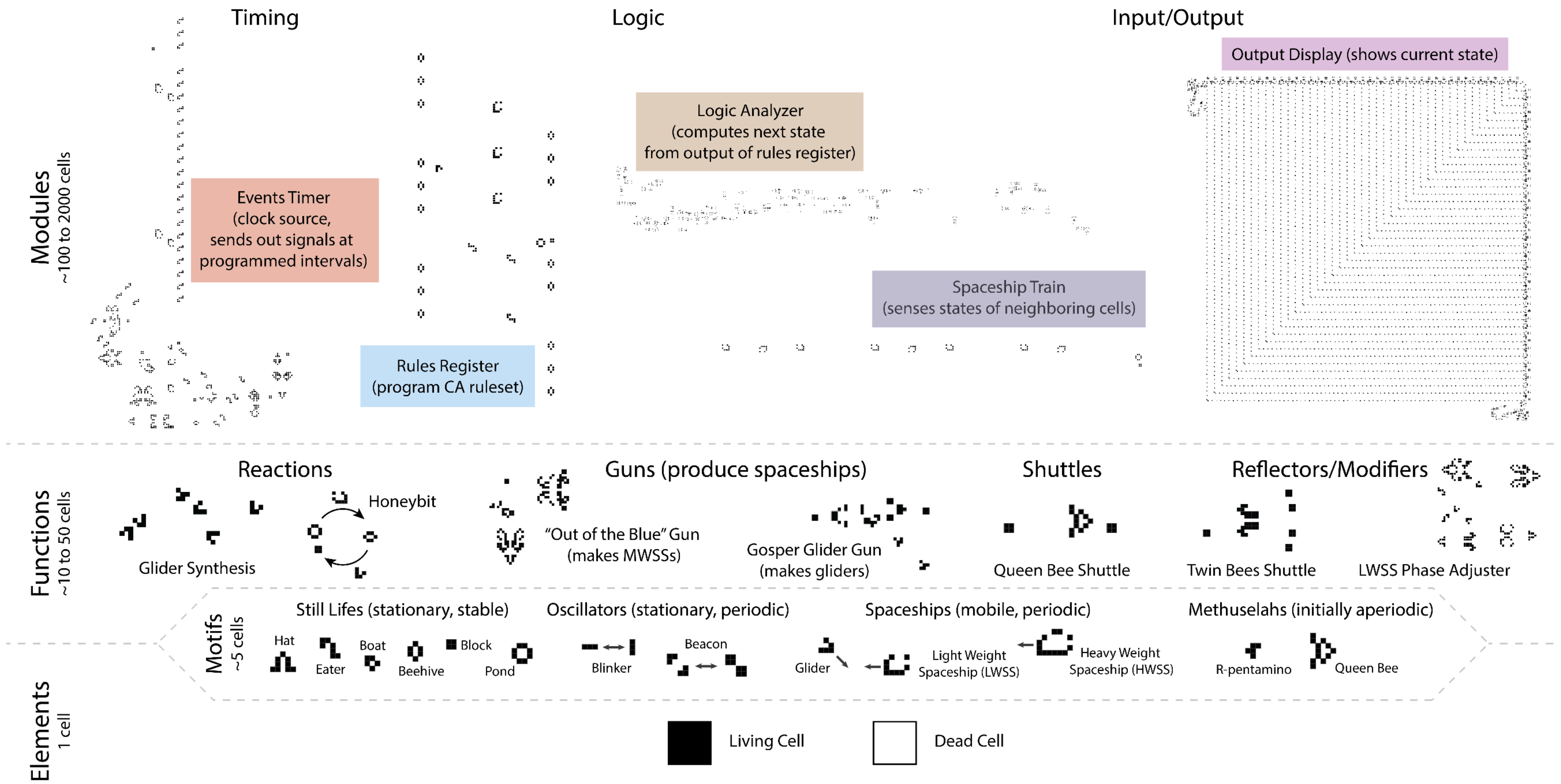


Figure 4-7: Hierarchical breakdown of OTCA Metapixel into modules, functions, motifs, and elements. Complex-level diagram is shown in Figure 4-8.

Small assemblies of cells form *motifs* on the order of about 5 cells across. *Still Lifes* are a category of motif that are static across time unless acted upon by another pattern. *Oscillators* repeatedly morph between several conformations at some regular time interval. Oscillators are categorized based on their period, for example, the Blinker and Beacon are period-2 oscillators, and Kok's Galaxy (Figure 4-6) is a period-8 oscillator. *Spaceships* are oscillators that move across the Life grid as they oscillate. Gliders, the smallest type of spaceship in Life, move across space in diagonal directions. Other spaceships, such as Light Weight, Medium Weight or Heavy Weight Spaceships (LWSS, MWSS, and HWSS, respectively) move in cartesian directions. Spaceships are categorized based on their period and on their speed. *Methuselahs* are small patterns that take a large number of generations to eventually stabilize. R-Pentamino is a Methuselah of only 5 initial cells that stabilizes in 1103 generations.

At the function-level, Life patterns on the scale of about 50 cells interact with each other in more structured and predictable ways. In order for two function-level patterns to be compatible, they must have compatible periods. For example, a train of period-4 LWSSes is compatible with a period-46 Twin Bees Shuttle as long the LWSSes in the train are spaced out by 11.5 cycles or some multiple of 11.5 cycles.

Guns are oscillators that create a train of regularly-spaced spaceships. Guns are categorized based on the type of spaceships they produce and the frequency at which they produce them. *Shuttles* are patterns comprised of an active region that oscillates between two stabilizing ends. Interactions with shuttles can serve a wide variety of functions. For example, the Twin Bees Shuttle can reflect (change the direction of) gliders, LWSSes, and MWSSes, and it can convert gliders to LWSSes or LWSSes to MWSSes. The Twin Bees Shuttle forms the basis of the "Out of the Blue" MWSS gun as well as many other period-46 oscillators. The Queen Bee Shuttle is made from a Queen Bee that oscillates between two stabilizing blocks. Two Queen Bee Shuttles form the basis of the Gosper Glider Gun and many other period 30 oscillators.

Reactions are collisions of Life objects that produce useful outcomes. The Honeybit reaction is used several times in the OTCA Metapixel to write, store, and read a single bit of data. In the Honeybit reaction, a glider collides with a Beehive to produce a Pond and a Block - this sets the bit. Then a LWSS collides with the pond, destroying the LWSS and returning the Pond/Block back to its original Beehive state. If the bit is not set, the LWSS passes by the beehive unharmed. *Glider Synthesis* is a process by which complex Life patterns are created solely through the collisions of gliders. Glider synthesis forms a large subset of known and actively studied Life reactions, and it was a critical piece of Conway's existence proof of a self-replicating universal constructor in Life. The initial glider configuration pictured in Figure 4-7 is a 6-glider synthesis for the Queen Bee Shuttle; an 8-glider synthesis is known for the Gosper Glider Gun and the Twin Bees Shuttle.

A more complete discussion of Life patterns and their history can be found on the

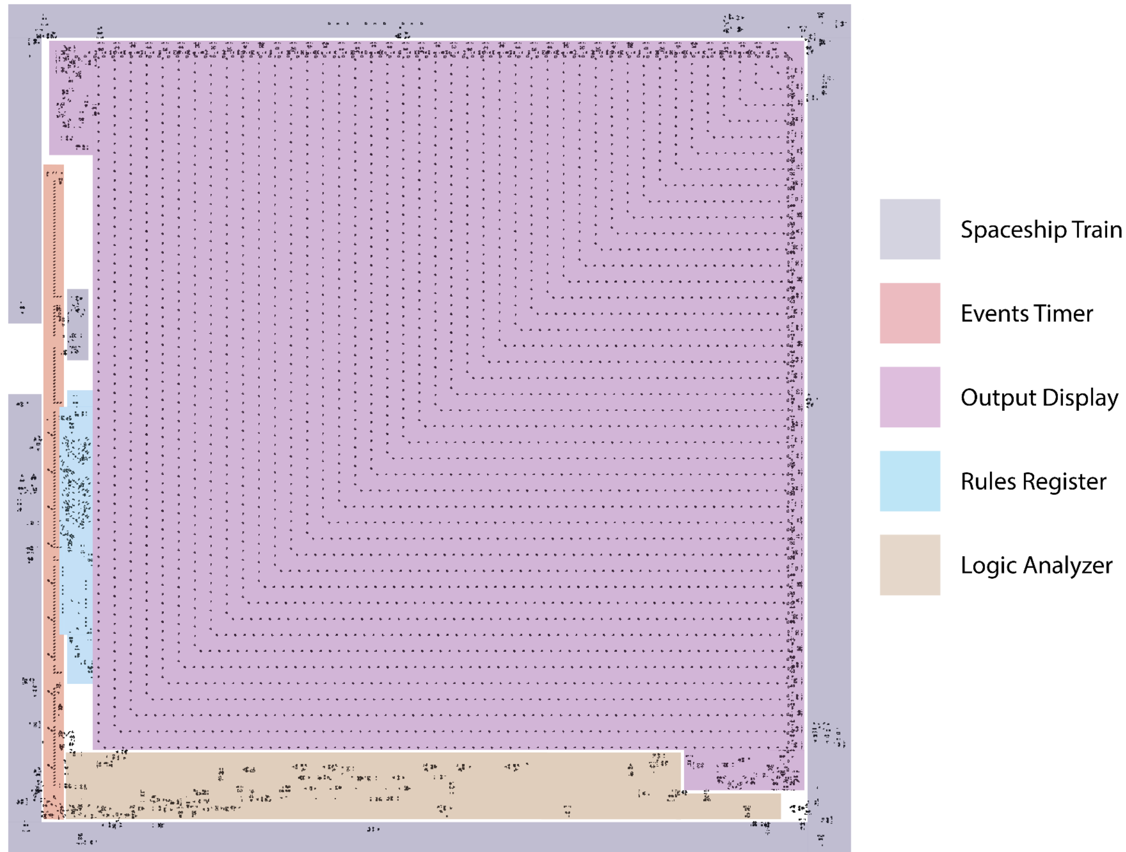


Figure 4-8: Complex-level diagram of OTCA Metapixel with the most important modules highlighted. Further breakdown of modules into functions, motifs, and elements given in Figure 4-7.

Life Wiki [2] or the Life Lexicon [66].

4.7.2 Modules and Complexes

Modules in Life are designed from functions and motifs to complete a singular, high-level task in a Life machine. Modules typically range in size from 100 to several thousand cells across. As with functions, modules must have compatible periods in order to be compatible with one another. Modules dictate the overall frequency at which a Life machine operates.

Spaceship *tapes* are often used to store, move, and execute binary information in Life machines. These tapes are read through collisions with other objects; the presence or absence of a spaceship in a given position on the tape will dictate the outcome of the collision. Collisions between two tapes form basic bit-wise logical operations.

Information on the tapes is often duplicated prior to being operated on in order to preserve the original data. The Spaceship Train (of LWSSes) in the OTCA Metapixel travels in a loop around the perimeter of the cell, collecting a tally of the neighboring states through collisions with Honeybits. The components of this module include the train itself, the reflectors and other modifiers used to direct the train along its course, and the Honeybit reactions set by neighboring cells.

Machines often require a central *events timer* to trigger the start and stop of key tasks. A Tractor Beam clock is typically used for this purpose in Life. Tractors beams are a type of gun aimed at a still life, such that each collision between the tractor beam's spaceship stream and the still life moves the still life some increment closer to the tractor beam's source. In a sense a tractor beam "pulls" the still life toward it. When the still life reaches some critical distance to the tractor beam source, it is destroyed and re-synthesized in its original starting position. The time it takes for the still life to complete one cycle of this behavior determines the overall period of the tractor beam clock; this period is programmed by controlling the initial distance of the still life from the tractor beam. The OTCA Metapixel's events timer shoots a stream of LWSSes and MWSSes towards a Block, moving the Block down by 8 cells in each collision; a complete cycle of the events timer takes 35,328 generations. The events timer has an added feature that a glider is ejected to the right during each collision with the Block. A fence of Eaters typically annihilates the glider immediately, but programmed holes in the fence allow the glider to pass through and trigger events at arbitrary phases of one complete cycle.

The OTCA Metapixel *displays its current state* by toggling on and off two banks of MWSS guns, whose streams intersect and mutually annihilate each other in the center of the pattern. Each bank is controlled by a simple latching mechanism, which toggles a LWSS gun on and off. When this gun is on, it interacts with a bank of shuttles in an "Out of the Blue" reaction, effectively turning them into MWSS guns.

A rules *register* is a type of persistent memory that is used, in the case of the OTCA Metapixel, to encode the rules of a CA. Inside the register, a collision with the Spaceship Train and a Honeybit reaction compare the number of living neighbors with the rules stored in the register. The OTCA metapixel's register capacity tops off at 16 bits of information. In order to store arbitrarily large numbers in this type of register or on one of the tapes described above, the register/tape would need to be infinite. Marvin Minsky described a type of finite-sized memory register with infinite memory capacity that solves this problem in 1967 [49]. Dean Hickerson designed the Minsky register in Life [34] in 1990 based on a description given by Conway in Winning Ways [8]. This paved the way for the development of a universal computer with finite size, built by Paul Chapman in 2002 [14].

A static *logic bank* is constructed within a machine to interpret data, typically from incoming spaceship tapes. Simple bitwise logical operators (not, and, or, nor, etc) can be computed on one or more pieces of data. The output from the OTCA Metapixel's

rules register is fed into a logic bank in order to determine the next state of the metacell.

4.7.3 Insights

The previous sections illustrates design hierarchy in one example of a complex, programmable Life machine. Moving from elemental cells, to motifs and functions, to modules, and finally to complexes introduces scaling on the order of 10-50x at each level. As with biological systems, function and motif-level structures are highly reusable in the design of module-level objects. Modules tend to be more highly tuned to the specific needs of the larger complex they belong to, both in terms of their time-dependencies and geometric layout; in general this means a module may need some editing to work within a complex it was not originally designed for.

Design

Human-directed design in Life occurs at the level of motif and function-level building blocks, based on a knowledge of codified interactions between them. It is entirely possible that a seemingly random assortment of Life cells produce desired high-level behaviors such as programmable self-replication. However, it would not be possible for a human to design a complex machine in Life by placing elemental parts without some notion of hierarchical abstraction. As with functional groups in organic chemistry, motifs and functions in Life provide a useful abstraction from the low level physics of the system. Without this abstraction, it is nearly impossible to develop an intuition about how a complex Life pattern will behave. Furthermore, it is likely that we would recognize familiar motifs within any complex Life pattern, human-designed or not. Small still lifes, oscillators, and spaceships spontaneously arise from random initial conditions with high frequency [24].

Evolutionary Potential

Interactions between a Life machine and debris in the environment could edit spaceship tapes, change logical outcomes, or affect other programmed functions of the machine. Some of these interactions could result in permanent changes to the functioning of the machine. As with genetic mutations in coding regions of the genome, most changes to the machine's inner workings would be deleterious to the overall functioning of the machine, however, some changes could eventually endow it with new behaviors.

Chapter 5

Simulation of Functional Digital Materials

Simulation of functional parts is carried out via a dynamic model at the granularity of functional primitives. In this model, neighboring face-connected functional primitive nodes apply forces and torques on one another through linear elastic interactions. Translational and rotational positions and velocities are calculated from these forces using discrete-time integration techniques. Evaluating the local interactions between nodes iteratively, subject to their initial state and boundary conditions (external forces, fixed regions, collisions, etc.) gives the state of the assembly over discrete time. Integration and other treatments of orientation are handled in spherical space to allow for potentially large angular displacements. Internal damping is modeled as a linear function of velocity. Anisotropic behaviors of *functional primitives* are parameterized by 15 stiffness and damping coefficients, k and d. Actuation is achieved by modulating the nominal distance of adjacent cells along a particular degree of freedom.

5.1 Solid Mechanics

Solid mechanics is the study of the motion and deformation of solid bodies under external forces, changes in temperature, and other stresses. Solid mechanics differentiates itself from other branches of continuum mechanics because solids are assumed to have a preferred rest shape. This thesis is concerned with a regime of solid mechanics called elasticity, where materials return to their initial rest shape after all external forces are removed.

The behavior of solids is typically characterized in terms of *stress* and *strain*. Stress describes the internal and external forces acting on a material, measured as force per area in pascals (Pa) or N/m^2. Strain describes the deformations of a material in response to stress, measured as a unit-less ratio of a length of deformation per unit of unstressed length. Figure 5-1 shows an example of a typical stress/strain curve for a solid material.

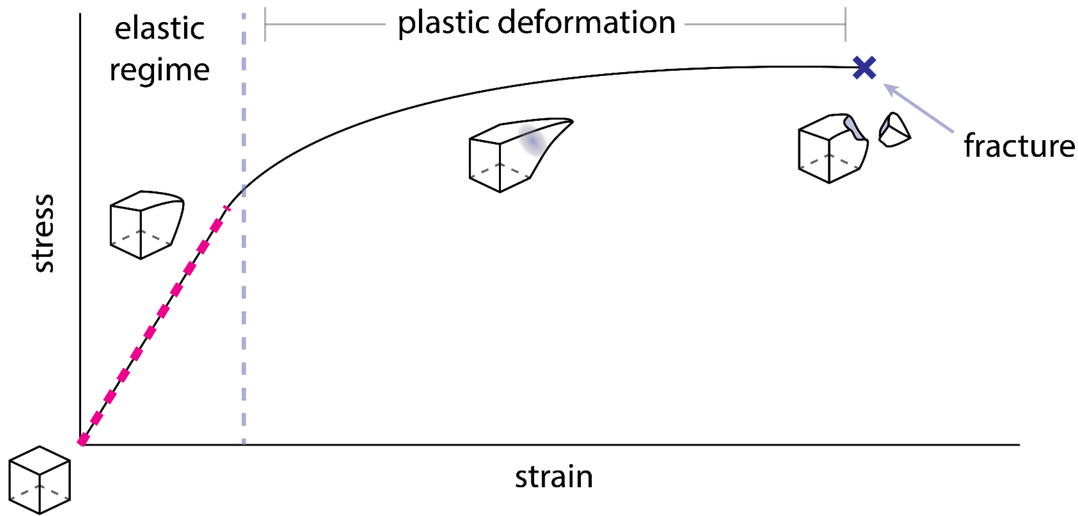


Figure 5-1: Stress/strain curves of materials reveal the linear elastic regime (pink), characterized by linear relationship between stress and strain. Under larger stresses, materials enter non-linear elastic and plastic deformations, and eventually fracture.

Solids deform in an elastic regime until stresses acting on or within them become so large they result in irreversible, plastic deformations or fracture. The modeling in this chapter will deal exclusively with *linear* elastic deformations of solids. This is the region of the stress/strain curve with constant slope, indicated by the pink dotted line in Figure 5-1. In this region, stress and strain have an approximately linear relationship to each other, in other words, materials obey Hooke's law:

$$F = kx$$

where F is a force, x is a displacement, and k is a measure of geometric stiffness. We can rewrite Hooke's law in terms of stress (σ) and strain (ϵ) as follows:

$$\sigma_{axial} = E\epsilon_{axial} \tag{5.1}$$

$$\sigma_{shear} = G\epsilon_{shear} \tag{5.2}$$

where E is the elastic modulus (also called "Young's modulus") and G is the shear modulus of a material. As long as the forces acting on and within a solid are sufficiently low, we can assume that these linear elastic deformation models apply.

So far we have discussed the ways that a solid's material properties, E and G, affect its behavior, next we must consider its geometry. Starting with Equations 5.1 and 5.2, we could try modeling the deformations of a linear, elastic solid as a multi-dimentional Hookian spring with axial and shear stiffnesses E and G. Though this strategy could yield reasonable results for simple geometries and loading patterns, like a rubberband in pure tension, it is not accurate for more complex situations. Instead, modern sim-

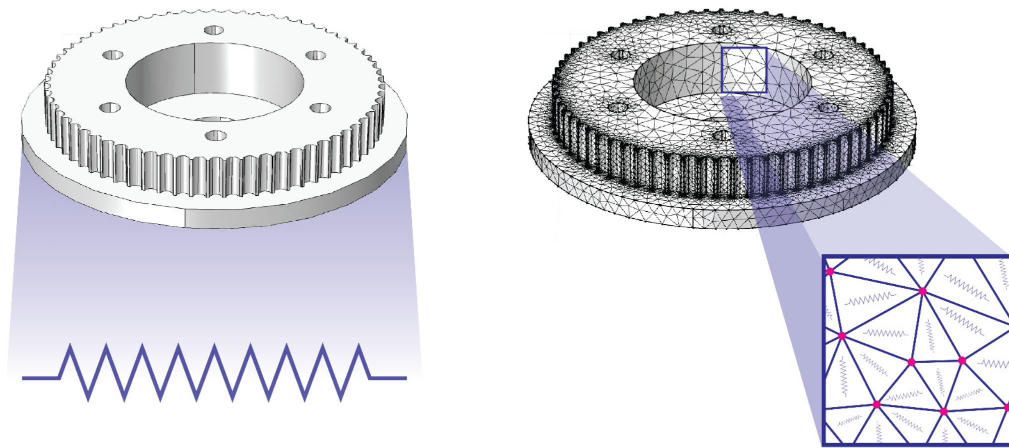


Figure 5-2: Two potential ways to model linear elastic deformations of a solid. The strategy on the left treats the entire geometry as a multidimensional Hookian spring, resulting in erroneous modeling of the solid's behavior. The strategy on the left breaks up the solid into small regions which can individually be modeled with Hooke's law to a reasonable degree of accuracy. The individual behaviors of the small regions are combined to synthesize a model of the global behavior of the entire geometry. Note - the regions being modeled with Hooke's law on the right are not treated as literal springs, just 3 dimensional regions modeled with linear elastic relationships between nodes (pink). The method on the right summarizes the main idea behind FEA of linear elastic solids.

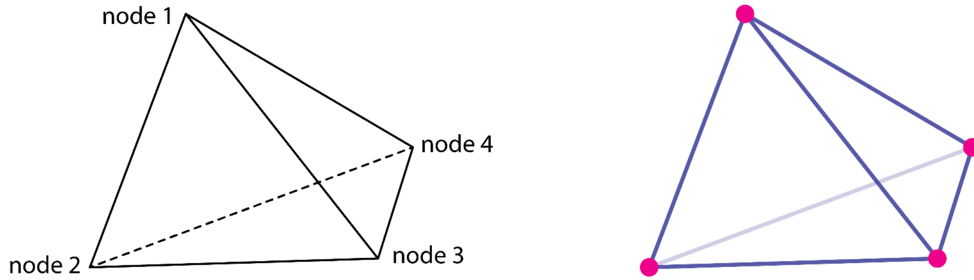


Figure 5-3: A tetrahedral element in 3D FEA. The vertices of each element are called nodes. In linear elastic solids, the relationships between forces (stress) and displacement (strain) of the nodes is described by variations on Hooke's law.

ulation techniques break up complex geometries into many simple, discrete regions whose individual behavior can be accurately modeled with Hooke's law (Figure 5-2). Combining the solutions to these discrete regions approximates the net behavior of the more complex geometry. The process I've just described is typically referred to as *Finite Element Analysis* (FEA).

Now we can model the global deformations of a solid as the summation of deformations of many smaller, discrete volumes, or *finite elements*. In 3D, these elements might have any number of shapes, a common element shape is the tetrahedral illustrated in Figure 5-3. The vertices of each finite element are called nodes. Nodes connect to other nodes through finite elements to form a mesh. No assumptions are made about the length of the edges of a finite element; in fact one of the main advantages of FEA is the ability to locally alter the resolution of the mesh in a particular region of interest.

We must also consider that materials may respond differently to different types of forces. Forces acting on finite elements fall into four categories: axial (tension and compression), shear, bending, and torsion. Each type of force causes a characteristic deformation, illustrated in Figure 5-4. Applying multiple types of forces on a finite element will cause it to exhibit a combination of deformations. We've seen how E determines the behavior of a material in response to axial stress and G for shear stress in Equations 5.1 and 5.2. Using information about the geometry of an element, we can derive equations for bending and torsional response in terms of torque (T) and angular displacement (θ):

$$T_{bending} = \frac{EI}{l}\theta_{bending} \tag{5.3}$$

$$T_{torsion} = \frac{GJ}{l}\theta_{torsion} \tag{5.4}$$

where I and J are area moment of inertias (a measure of the distribution of material in a cross section - convention uses J for torsional cross section and I for bending, but they are calculated the same way) and l is the length of the element in the relevant

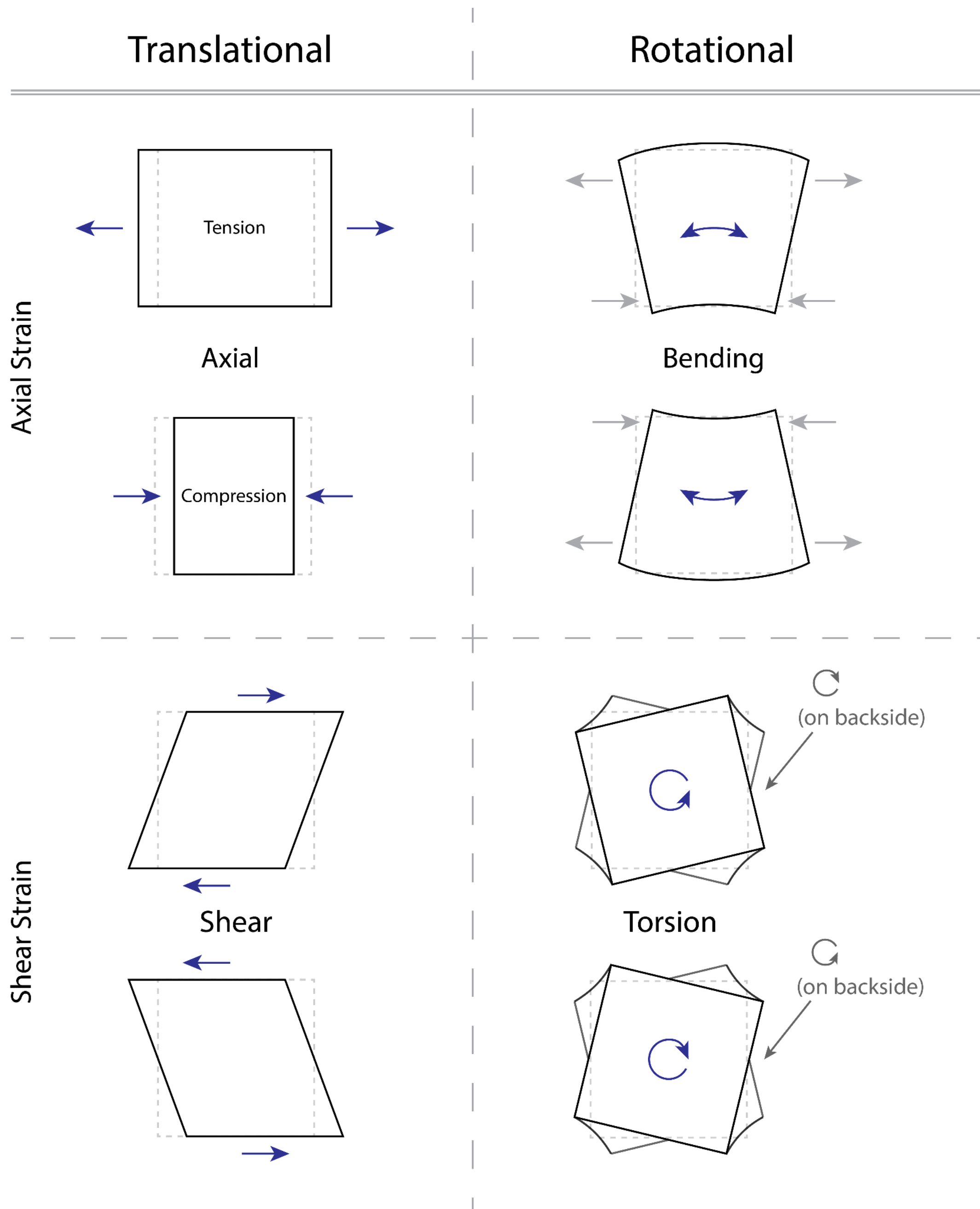


Figure 5-4: Deformations of a finite element under four types of forces. In modeling assemblies of *functional primitives*, these characteristic deformations are referred to as "internal degrees of freedom". Within an assembly of cells, axial (compression and tension) and shear stresses cause translational displacement and bending and torsion cause rotational displacement.

direction. Often in solid mechanics literature, the word moment is used in place of torque in this context. I'll be using the term torque for the remainder of this thesis because I'm more familiar with that term.

Equations 5.1, 5.2, 5.3, and 5.4 work nicely for homogenous materials, but don't address situations where a material's response depends on the direction of the applied force: when the material is *anisotropic*. Materials comprised of oriented fibers or other internal structures typically display anisotropic behaviors. In these cases, bulk E and G are not sufficient to describe a material's behavior.

5.2 Modeling Setup

Simulation of an assembly of *functional parts* happens at the granularity of identically-sized *functional primitives*. Functional parts, their decomposition into functional primitives, and the physical parts being modeled are illustrated in Figure 5-5. The number of functional primitives shown in each functional part is meant only as an illustration for now; the joining strategy and dimensions of functional parts are still in development. Once these parameters have been settled, the model can be adjusted to fit the parts simply by varying the scale and aspect ratio of the cubic lattice.

In the next sections, I'll demonstrate how to extend the ideas from FEA to model the dynamic behavior of multimaterial assemblies of identically-sized, anisotropic functional primitives, which I'll call "cells" for the remainder of this chapter. The primary motivations for this work are listed below:

Design description = simulation description: Functional parts are assembled on a regular, cubic lattice. Rather than using an arbitrary mesh (as is the case in Figure 5-2), assemblies are decomposed into their functional primitive representation for simulation. This makes the meshing the model trivial (a step which often requires some user skill), puts a lower bound on the size of the smallest finite element, makes all elements the same size and shape which allows many parameters to be precomputed in advance (increasing computational efficiency), and should result in better agreement with empirical data [12].

Parallelization: The behavior of this system is determined by local interactions between cells and their six face-connected neighbors. In each iteration of the solver, cells are treated as differential elements; differences in position, velocity, orientation, and angular velocity of a cell and its neighbor are translated into forces and torques. Each of these calculations only depends on the state of the two cells involved in the interaction and cells are evaluated in no particular order, so each interaction may be computed on a separate core of the GPU for increased performance.

Non-linear treatment of angular displacements: Some flexures and actuators may exhibit large angular deformations that must be handled using non-linear (spher-

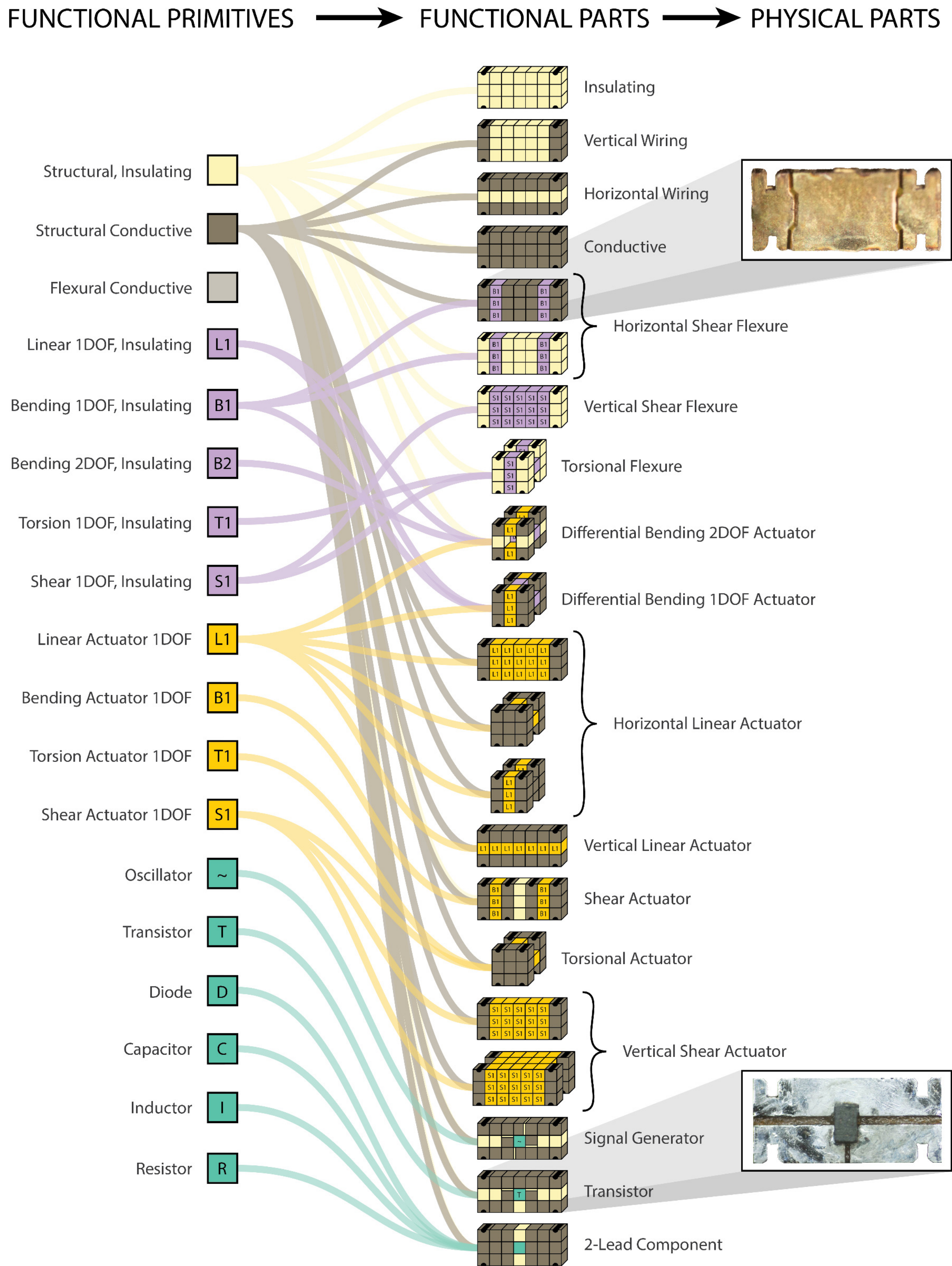


Figure 5-5: Illustration of various functional parts, showing their decomposition into functional primitives. Simulation of functional parts occurs at the granularity of functional primitives. Fabricated transistor and shear flexure shown alongside its virtual representation. *Image Credit: (photos of physical parts) Will Langford 2016*

ical) techniques. FEA typically relies on small angle approximations to handle small angular deformations and costly remeshing to handle larger deformations without introducing error.

Multimaterial interactions: Cells in this simulation are made from several material types which may be assembled in dithered patterns. In this model, dissimilar material properties are combined and applied to interactions between different cell types.

Anisotropy: Cells may exhibit both isotropic and anisotropic behaviors. To handle anisotropy, cell materials are parameterized by 15 stiffness and damping constants rather than just E and G.

Dynamic Simulation: We are interested in the time dependent behaviors of assemblies of parts. The framing of this simulation model makes computing dynamics trivial.

Electronics and actuation: Cells at the function-level are defined not only by their mechanical properties, but also by their ability to transmit electronic signals from one face to another, and their active properties in response to a signal. This model provides a computationally efficient way to integrate electronic effects with mechanical simulation.

Interaction with environment: We are interested in the interactions between functional parts and the environment, specifically for the tasks of locomotion and material handling. This model implements basic collision detection with the ground and will be extended in the future to handle collisions between cells.

The combinatorial space of mechanical, electronic, and actuated cell types is described in Figure 5-6. A selection of these cell types are included in Figure 5-5 and have been implemented in simulation. When describing mechanical behavior of a particular cell type, I'll refer to its four deformation modes (Figure 5-4) as "internal degrees of freedom" (DOF). For example, a 1DOF bending cell will have large deformations in bending along one axis, but relatively small deformations in response to other types of applied forces. Section 5.8 describes the process of electronic simulation in more detail, the remainder of this chapter will focus on the passive and active mechanical simulation of functional primitive cells.

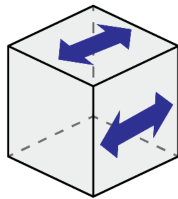
5.3 Spring-Damper Characteristics

The geometric stiffness of a structure relates the bulk properties of a material to its 3d geometry. For example, an I-beam has a higher geometric bending stiffness than an equal length rectangular bar made from the same amount of the same material. Unless otherwise noted, "stiffness" in this analysis refers to geometric stiffness.

Combinatorial Space of Functional Primitives

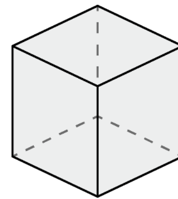
Axial	Shear	Bending	Torsional
0 DOF	1 DOF	2 DOF	3 DOF
Passive		Actuated	
Conductive		Insulating	

Linear Actuator



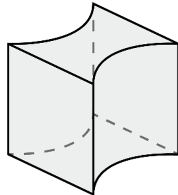
1 Actuated DOF Axial
0 DOF Shear
0 DOF Bending
0 DOF Torsional
Insulating

Conductive Structural



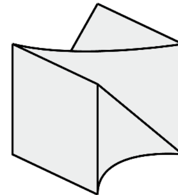
0 DOF Axial
0 DOF Shear
0 DOF Bending
0 DOF Torsional
Conductive

1 DOF Hinge



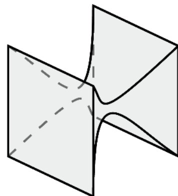
0 DOF Axial
0 DOF Shear
1 Passive DOF Bending
0 DOF Torsional
Insulating

1 DOF Torsinal Flexure



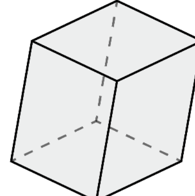
0 DOF Axial
0 DOF Shear
0 DOF Bending
1 Passive DOF Torsional
Insulating

2 DO F Hinge



0 DOF Axial
0 DOF Shear
2 Passive DOF Bending
0 DOF Torsional
Insulating

1 DOF Shear Flexure



0 DOF Axial
1 Passive DOF Shear
0 DOF Bending
0 DOF Torsional
Insulating

Figure 5-6: Combinatorial space of functional primitives with six examples explicitly described in terms of their electronic and mechanical properties. Note - any individual degree of freedom within a cell falls on the spectrum described in Figure 5-8. A cell is described at a high level as having a particular degree of freedom if its corresponding stiffness in that dimension is sufficiently flexible to allow for significant deformation under applied load. There are no infinitely stiff or fully unconstrained degrees of freedom in this system.

Stiffness and damping are used to characterize the response of cell's internal degrees of freedom to applied external forces. Stiffness has the units N/m and damping N·s/m. In three dimensions, each cell's passive mechanical properties are parameterized by 15 stiffness and damping constants:

$$k = \begin{cases} k_{axial_x} \\ k_{axial_y} \\ k_{axial_z} \\ \\ k_{shear_{xy}} \\ k_{shear_{xz}} \\ k_{shear_{yx}} \\ k_{shear_{yz}} \\ k_{shear_{zx}} \\ k_{shear_{zy}} \\ \\ k_{bending_x} \\ k_{bending_y} \\ k_{bending_z} \\ \\ k_{torsional_x} \\ k_{torsional_y} \\ k_{torsional_z} \end{cases} \qquad d = \begin{cases} d_{axial_x} \\ d_{axial_y} \\ d_{axial_z} \\ \\ d_{shear_{xy}} \\ d_{shear_{xz}} \\ d_{shear_{yx}} \\ d_{shear_{yz}} \\ d_{shear_{zx}} \\ d_{shear_{zy}} \\ \\ d_{bending_x} \\ d_{bending_y} \\ d_{bending_z} \\ \\ d_{torsional_x} \\ d_{torsional_y} \\ d_{torsional_z} \end{cases}$$

k_{shear} and d_{shear} are broken out into six parameters of the form $shear_{nm}$ because the shear response depends both on the direction of shear displacement between two cells (m) and on the axis along which the cells are connected (n). The $shear_{nm}$ notion used above is described graphically in in Figure 5-7.

The stiffness and damping constants of each cell type may be calculated from bulk properties of the constituent materials that make up the cell, or they may be measured empirically. For example, an isotropic cell made from a bulk material with elastic modulus E and shear modulus G has stiffnesses along the x axis defined by [40]:

$$k_{axial_x} = \frac{EA_x}{l} \tag{5.5a}$$

$$k_{shear_{xy}} = \frac{GA_{sy}}{l} \tag{5.5b}$$

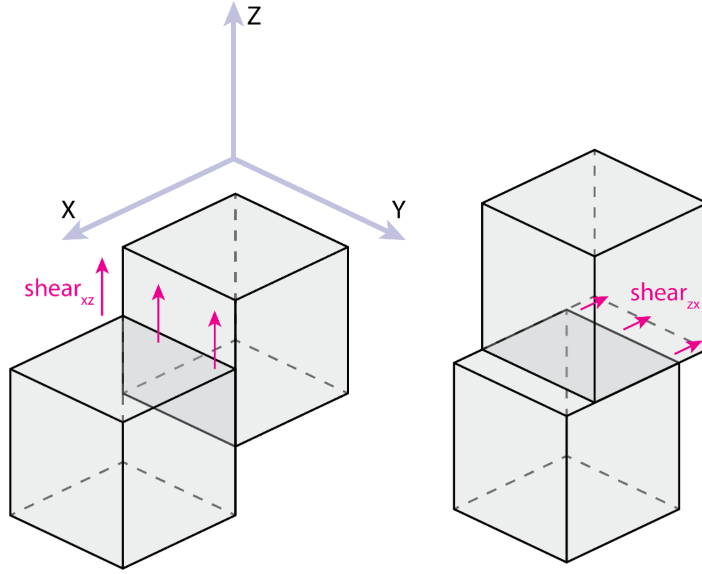


Figure 5-7: Illustration of shear subscript notation. First subscript describes the direction of neighboring cell connection, second describes the direction of shear displacement.

$$k_{bending_x} = \frac{EI_x}{l} \tag{5.5c}$$

$$k_{torsion_x} = \frac{GJ_x}{l} \tag{5.5d}$$

where A is the cross sectional area, l is the length of the cell, and A_s is a correction on A to account for the assumption of a constant shear profile as apposed to a parabolic profile. For rectangular cross sections, $A_s = \frac{5}{6}A$ [19].

The natural frequency (ω_n) of any of the translational degrees of freedom within a cell is calculated by:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{5.6}$$

where m is the cell's mass. And for the rotational degrees of freedom by:

$$\omega_n = \sqrt{\frac{k}{I}} \tag{5.7}$$

where I is the cell's moment of inertia around the axis of interest. Linear damping for any degree of freedom is calculated from natural frequency by:

$$d = 2\zeta\omega_n$$

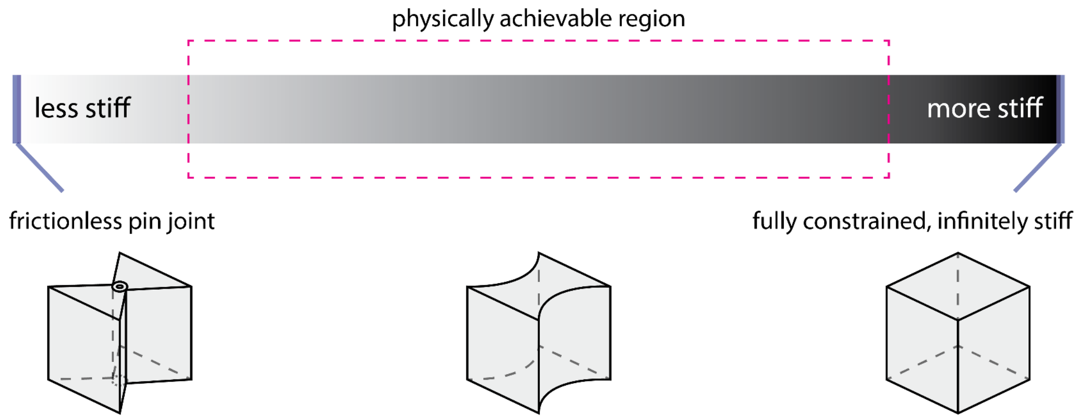


Figure 5-8: Continuum of bending stiffness in a 1DOF bending flexure. Region which can be physically achieved through existing fabrication techniques is highlighted in pink.

where ζ is a constant called the damping ratio, with:

$$\begin{aligned} \zeta &= 0 && \text{undamped} \\ 0 < \zeta &< 1 && \text{underdamped} \\ \zeta &= 1 && \text{critically damped} \\ \zeta &> 1 && \text{overdamped} \end{aligned}$$

A linear scale showing the range of physically achievable 1DOF bending cell types compared with the range of theoretically possible types is shown in Figure 5-8. Due to manufacturing and material constraints, we do not envision functional primitives that occupy the region near zero bending stiffness (e.g. a frictionless pin joint) or infinite bending stiffness (zero compliance material).

In the case that neighboring cells have different stiffness or damping constants, a composite stiffness or damping constant must be calculated in order to properly model the junction between the cells. Composite stiffness can be calculated according to the following formula:

$$k_{composite} = \frac{2k_1k_2}{k_1 + k_2} \tag{5.9}$$

similarly, composite damping is calculated by:

$$d_{composite} = \frac{2d_1d_2}{d_1 + d_2} \tag{5.10}$$

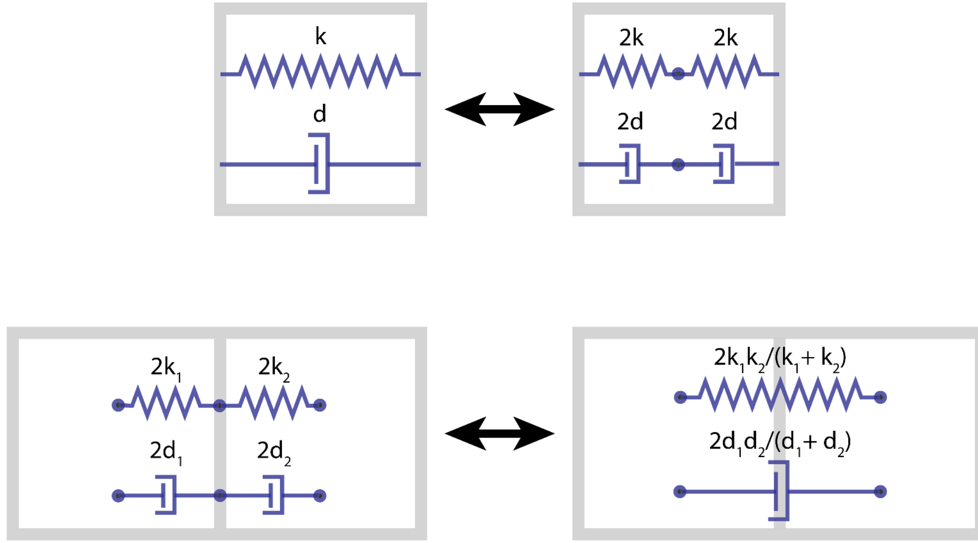


Figure 5-9: Calculation of composite stiffness and damping of two cells with different k and d constants.

Equations 5.9 and 5.10 are equivalent to the formulas for two springs or two dampers of half length in series. This is demonstrated graphically in Figure 5-9. Notice, for example, in the case of $k1 = k2$, Equation 5.9 reduces to $k_{composite} = k1 = k2$.

5.4 Translational Forces

In the simplest case without rotation, the force $\vec{F}_{1\to2}$ applied to cell 2 by cell 1 is given by:

$$\vec{F}_{1\to2} = \vec{k} \circ (\vec{p}_1 - \vec{p}_2) + \vec{d} \circ (\vec{v}_1 - \vec{v}_2) \tag{5.11}$$

where $\vec{p}_1$ and $\vec{p}_2$ are the displacements of cell 1 and cell 2 from their nominal position in the lattice, $\vec{v}_1$ and $\vec{v}_2$ are the cells' translational velocities, $\circ$ is multiplication of two vectors by element, and $\vec{k}$ and $\vec{d}$ are 3D vectors containing the appropriate composite stiffness and damping constants for the interaction between the cells. For example, given the scenario illustrated in Figure 5-10, where two cells are connected along the x axis, an axial constant should be used for displacements along x and shear constants should be used for displacements along y and z:

$$\vec{k} = \begin{bmatrix} k_{axial_x} \\ k_{shear_{xy}} \\ k_{shear_{xz}} \end{bmatrix} \qquad \vec{d} = \begin{bmatrix} d_{axial_x} \\ d_{shear_{xy}} \\ d_{shear_{xz}} \end{bmatrix}$$

If we wish to consider the orientations of the cells, denoted by the unit quaternions $q1$ and $q2$, we will need to adjust equation 5.11.

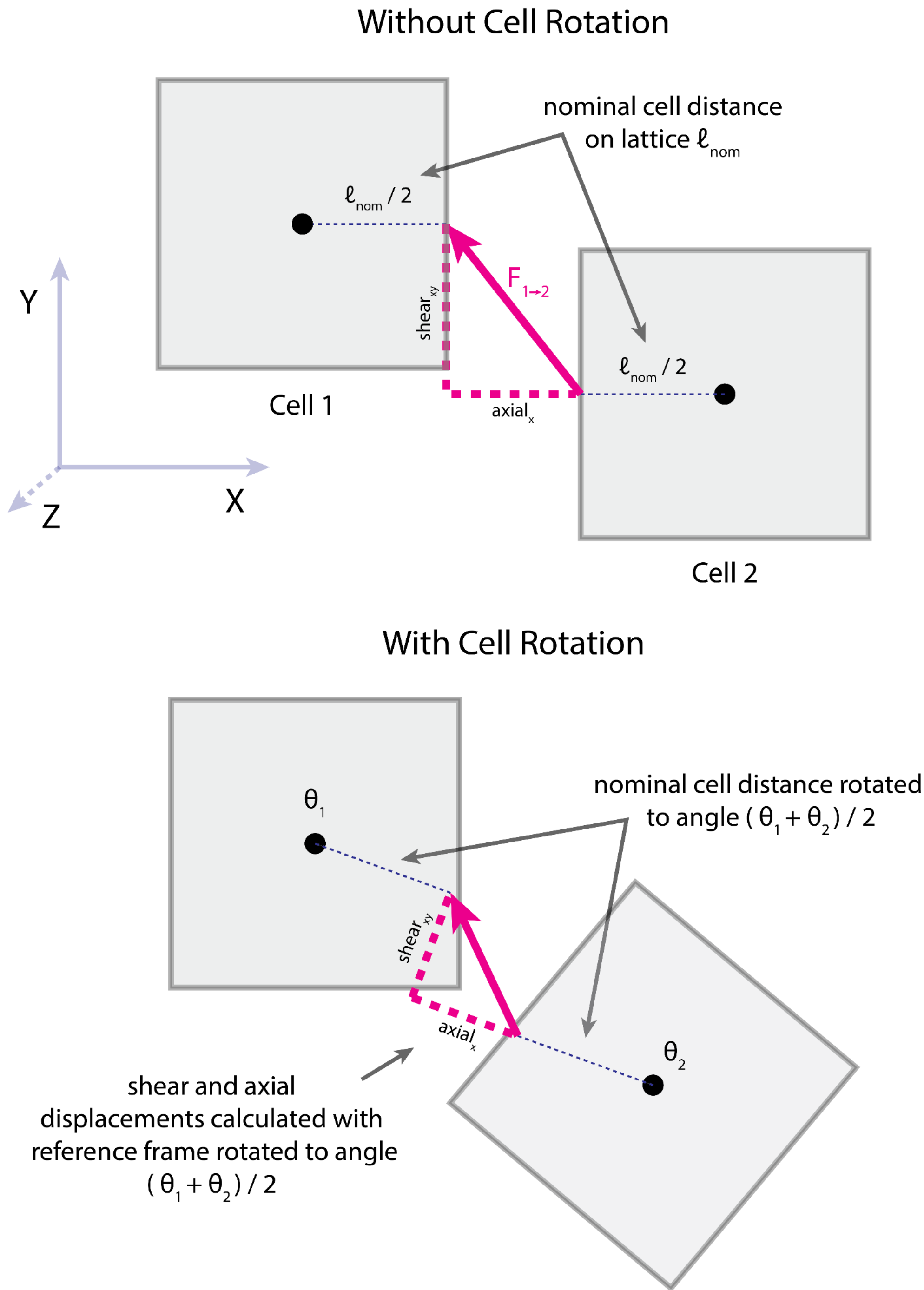


Figure 5-10: Diagram of translational force calculations, with and without relative rotations between cells.

We can rotate a vector $\vec{v}$ in 3-space by a quaternion q in 4-space:

$$\vec{v}_{rotated} = q * \vec{v} * q^* \tag{5.12}$$

by treating $\vec{v}$ as a 4-space vector $[x, y, z, w]$ with $w = 0$. The operator $*$ denotes the Hamilton product:

$$a * b = \begin{bmatrix} a_w b_x + a_x b_w + a_y b_z - a_z b_y \\ a_w b_y - a_x b_z + a_y b_w + a_z b_x \\ a_w b_z + a_x b_y - a_y b_x + a_z b_w \\ a_w b_w - a_x b_x - a_y b_y - a_z b_z \end{bmatrix}$$

and q^* denotes the conjugate of q:

$$q^* = \begin{bmatrix} -q_x \\ -q_y \\ -q_z \\ q_w \end{bmatrix}$$

By performing a spherical linear interpolation (slerp) halfway between $q1$ and $q2$, we can calculate the average orientation of the two cells as a unit quaternion:

$$q_{avg} = slerp(q_1, q_2; 0.5)$$

Using q_{avg}, we can rotate the stiffness and damping vectors from Equation 5.11 into this average reference frame, and substitute the rotated vectors back into equation 5.11:

$$\vec{k}_{rot} = (q_{avg} * \vec{k} * q^*_{avg}) \tag{5.13a}$$

$$\vec{d}_{rot} = (q_{avg} * \vec{d} * q^*_{avg}) \tag{5.13b}$$

$$\vec{F}_{1\to 2} = \vec{k}_{rot} \circ (\vec{p}_1 - \vec{p}_2) + \vec{d}_{rot} \circ (\vec{v}_1 - \vec{v}_2) \tag{5.14}$$

Finally, we need to make an adjustment to the nominal differential position between the cells, indicated in Figure 5-10. The form above assumes the nominal distance between the centers of the cells is the unrotated distance from cells 2 to cell 1 in their initial lattice configuration, $\vec{l}_{nom21}$:

$$\vec{l}_{nom21} = \vec{p}_{abs2} - \vec{p}_{abs1} \tag{5.15}$$

where $\vec{p}_{abs}$ is the absolute position of a cell in the world reference frame. Rotating $\vec{l}_{nom}$ into the average rotational reference frame of the two cells gives

$$\vec{l}_{rot21} = q_{avg} * \vec{l}_{nom21} * q^*_{avg}$$

introducing this correction to Equation 5.14 gives the final form:

$$\vec{F}_{1\to 2} = \vec{k}_{rot} \circ (\vec{p}_1 - \vec{p}_2 + \vec{l}_{nom21} - \vec{l}_{rot21}) + \vec{d}_{rot} \circ (\vec{v}_1 - \vec{v}_2) \tag{5.16}$$

The force $\vec{F}_{2\to 1}$ exerted on cell 1 by cell 2 is equal in magnitude to $\vec{F}_{1\to 2}$ and opposite in direction:

$$\vec{F}_{2\to 1} = -\vec{F}_{1\to 2} = \vec{k}_{rot} \circ (\vec{p}_2 - \vec{p}_1 + \vec{l}_{nom12} - \vec{l}_{rot12}) + \vec{d}_{rot} \circ (\vec{v}_2 - \vec{v}_1) \tag{5.17}$$

Only $\vec{F}_{1\to 2}$ is indicated with a solid pink arrow in Figure 5-10.

Dividing by the mass, we can calculate the acceleration of cells 1 and 2 due to interactions between them:

$$\vec{a}_{1\to 2} = \frac{\vec{F}_{1\to 2}}{m_2} \qquad \vec{a}_{2\to 1} = \frac{\vec{F}_{2\to 1}}{m_1}$$

5.5 Rotational Forces

So far, this model uses shear and axial stiffness and damping constants to compute the translational interactions between neighboring cells in 3D. In order to incorporate bending and torsional forces, we must develop a method of for cells to apply torques to one another. These torques result in rotational motion of a cell about its center of mass (assumed to be the center of the cell).

A sketch of the translational and rotational model for cell interaction is shown in Figure 5-11. In this model, we consider the fact that shear and axial forces between cells are not applied directly to the center of mass of a cell, but rather at some moment arm from the center of mass. We define the moment arm $\vec{r}_{moment2}$ of cell 2 to be half the nominal distance from cell 2 to cell 1, rotated to the average reference frame of the two cells:

$$\vec{r}_{moment} = \frac{\vec{l}_{rot21}}{2}$$

Then the torque $\vec{T}_{1\to 2}$ can be calculated by the cross product:

$$\vec{T}_{1\to 2} = \vec{r}_{moment} \times \vec{F}_{1\to 2} \tag{5.18}$$

If we stop now, we have set up a simulation of an arbitrary linkage of cells tied together with frictionless, spherical joints at their centers of mass.

Next we'll add in bending and torsional stiffness, which create reaction forces that counteract relative rotations between cells (T_{rxn} in Figure 5-11). For neighboring cells

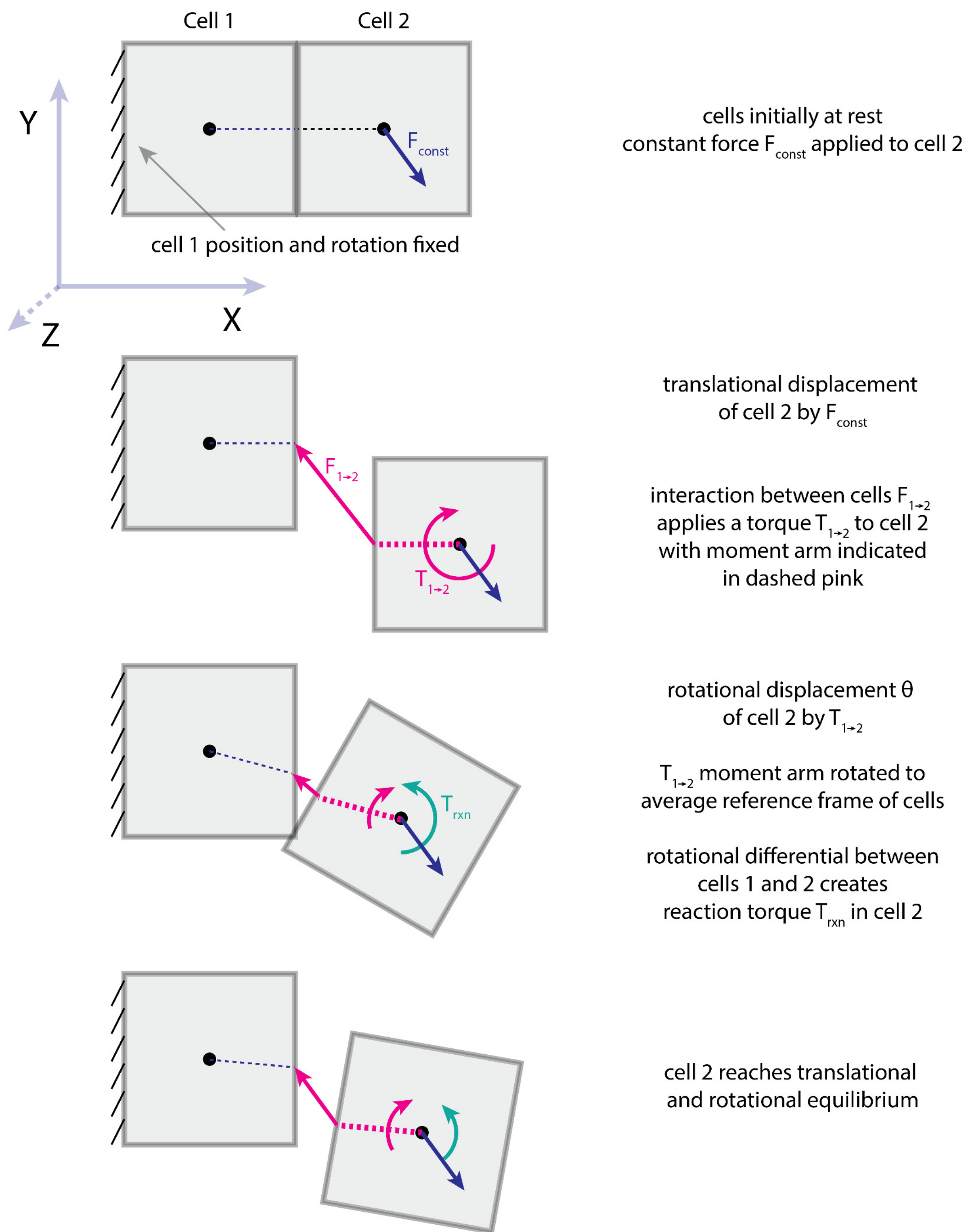


Figure 5-11: Diagram of translational and rotational forces between cells. Translational forces due to shearing cause a torque in cell 2 ($T_{1\to2}$). Relative rotation between the cells induces a reaction torque (T_{rxn}).

oriented along the x axis, we have composite stiffness and damping vectors:

$$\vec{k} = \begin{bmatrix} k_{torsion_x} \\ k_{bending_y} \\ k_{bending_z} \end{bmatrix} \qquad \vec{d} = \begin{bmatrix} d_{torsion_x} \\ d_{bending_y} \\ d_{bending_z} \end{bmatrix}$$

Then we rotate the composite stiffness and damping constants into the average reference frame between the cells as we did in Equations 5.13a and 5.13b to get $\vec{k}_{rot}$ and $\vec{d}_{rot}$.

Next we calculate the angular displacement and relative angular velocity between the two cells. We have to be careful in this step to remember that angular velocities (and torques) are vectors in cartesian space, but euler rotations ($\vec{\theta}$) are vectors in spherical space. So while vector addition and subtraction of angular velocities is valid:

$$\vec{\omega}_{diff} = \vec{\omega}_1 - \vec{\omega}_2$$

Vector addition and subtraction of euler rotations is not valid (though it might be ok for small angles):

$$\vec{\theta}_{diff} \neq \vec{\theta}_1 - \vec{\theta}_2$$

Instead, we'll use the two cells' quaternions to calculate the axis ($\vec{A}_\theta$) and angle of rotation ($a_{\theta 21}$) from cell 2 to cell 1:

$$q_{diff} = q_1^* * q_2 \tag{5.19}$$

$$a_{\theta 21} = 2 * acos(q_{diff_w})$$

(if $a_{theta} == 0$ then the axis does not matter, we can pick something arbitrary)

$$A_\theta = \frac{1}{\sqrt{1 - q_{diff_w}^2}} \begin{bmatrix} q_{diff_x} \\ q_{diff_y} \\ q_{diff_z} \end{bmatrix}$$

A_θ, $a_{\theta 21}$, ω_1, and ω_2 are used to calculate reaction torque of cell 2 ($T_{2_{rxn}}$), following a similar spring/damper setup as we used in Equation 5.16:

$$\vec{T}_{2_{rxn}} = \vec{k}_{rot} \circ (a_{\theta 21} \vec{A}_\theta) + \vec{d}_{rot} \circ (\omega_1 - \omega_2) \tag{5.20}$$

The total torque on cell 2 is the sum of the torque applied by shear and axial interactions with cell 1 (Equation 5.18) and the reaction force of cell 2 (Equation 5.20):

$$\vec{T}_2 = \vec{T}_{1\to 2} + \vec{T}_{2_{rxn}} \tag{5.21}$$

more verbosely:

$$\vec{T}_2 = \frac{\vec{l}_{rot21}}{2} \times (\vec{k}_{rot} \circ (\vec{p}_1 - \vec{p}_2 + \vec{l}_{nom21} - \vec{l}_{rot21}) + \vec{d}_{rot} \circ (\vec{v}_1 - \vec{v}_2))$$

$$+ \vec{k}_{rot} \circ (a_{\theta 21} \vec{A}_\theta) + \vec{d}_{rot} \circ (\omega_1 - \omega_2) \quad (5.22)$$

The torque on cell 1 from the interaction with cell 2 is equal in magnitude and opposite in direction:

$$\vec{T}_1 = -\vec{T}_{1\to 2} - \vec{T}_{2_{rxn}} = \vec{T}_{2\to 1} + \vec{T}_{1_{rxn}} \quad (5.23)$$

$$\vec{T}_1 = \frac{\vec{l}_{rot12}}{2} \times (\vec{k}_{rot} \circ (\vec{p}_2 - \vec{p}_1 + \vec{l}_{nom12} - \vec{l}_{rot12}) + \vec{d}_{rot} \circ (\vec{v}_2 - \vec{v}_1))$$
$$+ \vec{k}_{rot} \circ (a_{\theta 12} \vec{A}_\theta) + \vec{d}_{rot} \circ (\omega_2 - \omega_1) \quad (5.24)$$

To calculate angular acceleration ($\vec{\alpha}$), we divide by moment of inertia. For a 3D rectangular cell with sides l_x, l_y, l_z and mass m, the tensor moment of inertia is given by:

$$I_{inertia} = \frac{m}{12} \begin{bmatrix} l_y^2 + l_z^2 & 0 & 0 \\ 0 & l_x^2 + l_z^2 & 0 \\ 0 & 0 & l_x^2 + l_y^2 \end{bmatrix}$$

where:

$$\vec{T} = I_{inertia} \vec{\alpha}$$
$$\vec{\alpha} = I_{inertia}^{-1} \vec{T}$$

$$I_{inertia}^{-1} = \frac{12}{m} \begin{bmatrix} \frac{1}{l_y^2 + l_z^2} & 0 & 0 \\ 0 & \frac{1}{l_x^2 + l_z^2} & 0 \\ 0 & 0 & \frac{1}{l_x^2 + l_y^2} \end{bmatrix}$$

For cells 1 and 2:

$$\vec{\alpha}_1 = \frac{12}{m_1} \vec{T}_1 \circ \begin{bmatrix} \frac{1}{l_y^2+l_z^2} \\ \frac{1}{l_x^2+l_z^2} \\ \frac{1}{l_x^2+l_y^2} \end{bmatrix} \qquad \vec{\alpha}_2 = \frac{12}{m_2} \vec{T}_2 \circ \begin{bmatrix} \frac{1}{l_y^2+l_z^2} \\ \frac{1}{l_x^2+l_z^2} \\ \frac{1}{l_x^2+l_y^2} \end{bmatrix}$$

where, again, $\circ$ is multiplication of two vectors by element.

5.6 Actuation

Mechanical actuation is achieved by changing the nominal translation/rotation of a particular degree of freedom. For example if the nominal distance (l_{nom21} from Equation 5.15) between two cells oriented along the x axis at a distance l is given by:

$$\vec{l}_{nom21} = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix}$$

An axial actuator cell with strain ϵ would be modeled with the following nominal distance to its neighbor:

$$\vec{l}_{nom21} = \begin{bmatrix} l + \frac{1}{2}\epsilon \\ 0 \\ 0 \end{bmatrix}$$

where the factor of $\frac{1}{2}$ comes from the strain being split between the neighbor to the +x direction and the -x direction equally.

For a shear actuator, with shearing strain ϵ occurring in the y direction, the nominal distance to the neighboring cells becomes:

$$\vec{l}_{nom21} = \begin{bmatrix} l \\ \frac{1}{2}\epsilon \\ 0 \end{bmatrix}$$

And, similarly, for a shear actuator in the z direction:

$$\vec{l}_{nom21} = \begin{bmatrix} l \\ 0 \\ \frac{1}{2}\epsilon \end{bmatrix}$$

For rotations, we subtract the actuated angular offset in the calculation of the difference in orientation between two cells (Equation 5.19):

$$q_{diff} = q^*_{actuated} * (q^*_1 * q_2)$$

where the actuator strain between two cells is given by the 3D euler angle ϕ and:

$$q_{actuated} = \begin{bmatrix} sin(\phi_x) * cos(\phi_y) * cos(\phi_z) + cos(\phi_x) * sin(\phi_y) * sin(\phi_z) \\ cos(\phi_x) * sin(\phi_y) * cos(\phi_z) - sin(\phi_x) * cos(\phi_y) * sin(\phi_z) \\ cos(\phi_x) * cos(\phi_y) * sin(\phi_z) + sin(\phi_x) * sin(\phi_y) * cos(\phi_z) \\ cos(\phi_x) * cos(\phi_y) * cos(\phi_z) - sin(\phi_x) * sin(\phi_y) * sin(\phi_z) \end{bmatrix}$$

For a torsional actuator with angular strain ϵ:

$$\phi = \begin{bmatrix} \frac{1}{2}\epsilon \\ 0 \\ 0 \end{bmatrix}$$

And for a y-axis bending actuator:

$$\phi = \begin{bmatrix} 0 \\ \frac{1}{2}\epsilon \\ 0 \end{bmatrix}$$

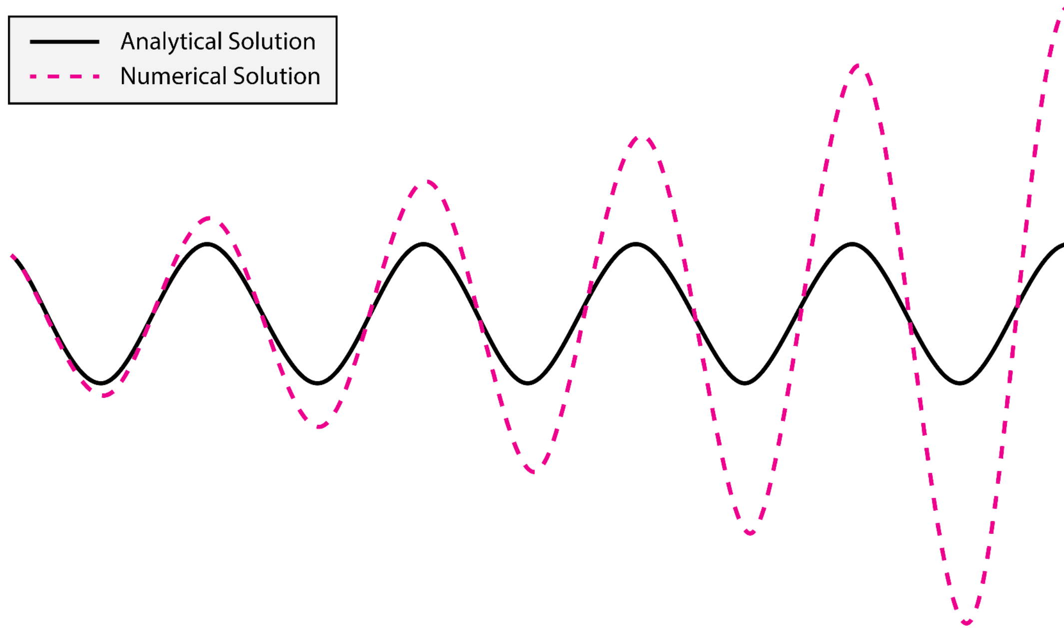


Figure 5-12: An example of inaccurate numerical integration causing instability, especially in undamped/underdamped/stiff systems.

And, similarly, for a z-axis bending actuator:

$$\phi = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}\epsilon \end{bmatrix}$$

5.7 Sources of Error

5.7.1 Numerical Time Integration

Numerical time integration usually introduces error into the system's state. For undamped, underdamped, or very stiff oscillating systems, this error can lead to numerical instability (Figure 5-12).

For linear systems, explicit forward time integration can be achieved using a variety of methods, each with associated error and computational cost. The simplest and most computationally efficient approach is forward Euler (1st order):

$$\vec{v}_{t+1} = \vec{v}_t + \vec{a}_t \Delta t$$

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t \Delta t$$

Verlet integration requires storing the previous two position calculations (at time t and $t-1$) in order to calculate the next position. The following form of Verlet gives 2nd order position and 1st order velocity calculations:

$$\vec{p}_{t+1} = 2\vec{p}_t - \vec{p}_{t-1} + \vec{a}_t \Delta t^2$$

$$\vec{v}_{t+1} = \frac{\vec{p}_{t+1} - \vec{p}_t}{\Delta t}$$

Higher order methods such as Runge-Kutta (RK4, 4th order) reduce error further, but require multiple calculations in order to solve for a single time step. Lower order Runge-Kutta methods, such as the midpoint method, may also be worth exploring.

Time integration of rotations is a bit trickier. The methods described above are meant for intergration in linear space, so they can be applied to the transformation between angular acceleration and velocity. For example, forward euler of angular acceleration is:

$$\vec{\omega}_{t+1} = \vec{\omega}_t + \vec{\alpha}_t \Delta t$$

3D rotational space in inherently spherical, and applying linear integration techniques to integrate angular velocity will create large errors when the angular displacement is large. Instead we can convert angular velocity into quaternion space:

$$\dot{q} = 1/2\vec{\omega} * q$$

with $\vec{\omega}$ as a 4-space vector with $w = 0$ and $*$ denoting the Hamilton product.

Euler integration of $\dot{q}$ is as easy as multiplying by Δt and normalizing to a unit quaternion:

$$q = normalize(\dot{q}\Delta t)$$

The time step Δt should be chosen as large as possible to maximize computational efficiency without causing numerical instability. For the interaction of two cells with different masses and stiffnesses, the maximum natural frequency ($\omega_{n_{max}}$) of the bond can be calculated from Equations 5.6 and 5.7 by:

$$\omega_{n_{max}} = \sqrt{\frac{k_{composite}}{m_{min}}} \quad \text{and} \quad \omega_{n_{max}} = \sqrt{\frac{k_{composite}}{I_{min}}} \tag{5.25}$$

where m_{min} is the smaller of the two masses and I_{min} is the smaller of the cells' moment of inertias. The optimal time step for numerical integration is calculated from the maximum natural frequency ($\omega_{n_{globalMax}}$) of any translational or rotational interaction of cells within the entire assembly:

$$\Delta t < \frac{1}{2\pi\omega_{n_{globalMax}}} \tag{5.26}$$

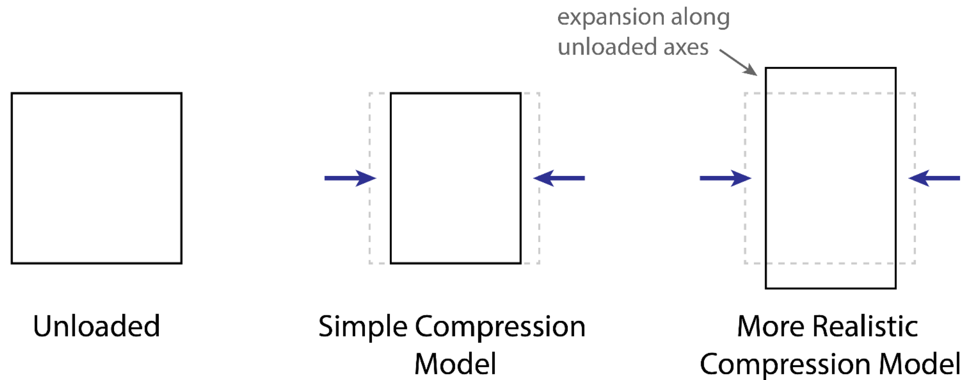


Figure 5-13: Deviation from reality in simple *function* simulation. Some coupling between degrees of freedom is expected, for example, compression on one axis will result in expansion along other, unloaded axes.

5.7.2 Poisson's Ratio

In reality, the degrees of freedom illustrated in Fig 5-4 are not completely orthogonal from one another. For example, compression along one axis of a solid will cause some degree of expansion along the unloaded axes (Fig 5-13). For axial deformations, this effect can be modeled by an additional linear elastic relationship:

$$\epsilon_{transverse} = -\nu\epsilon_{axial}$$

where ν is Poisson's ratio and ϵ are strains in the transverse and axial directions.

Though I am not currently modeling this effect in simulation, it could be added in the fullness of time.

5.7.3 Floating Point Operations

Finally, the accumulation of error in floating point calculations can potentially throw systems unstable, especially when damping is absent. In its current JavaScript implementation, floating points numbers are stored as 64 bit in the CPU and 32bit in the GPU. I'm not currently using any tricks to mitigate floating point error in my code.

5.8 Electronic Simulation

The physics engine currently supports simulation of various types of oscillators (sine, triangle, PWM, and saw) whose frequency, phase, pulse-width, and direction can be adjusted. Oscillators have two polar outputs, which are connected through wires to actuators. At each time step, the state of the electronic system is evaluated, and the analog voltage across each actuator is fed into the mechanical simulation. Errors are

thrown when a user incorrectly wires components and renders the electronic simulation unsolvable.

In the system we are modeling, electronic effects are orders of magnitude faster than mechanical effects. Rather simulating the propagation of electronic signals across cells through iterative, local interactions, the state of all electronics is completely solved before each step of mechanical simulation. This means that it is not currently possible to simulate things like ring oscillators, which rely on a time dependance to be properly modeled. Depending on the future trajectories of fabrication of electronic components for this system, a time-dependent simulation of electronics may be worth revisiting.

5.9 Collision Detection

Collision detection is needed to simulate interactions with the environment, such as gripping and locomotion. Simple collision detection with the ground has been implemented using a spring/damper system to model penetration of the ground plane. Calculation of normal force and friction is also achieved through these interactions. Currently, the edge of a cell is calculated by the position of its center plus a constant radial offset. A more accurate model could compute deformations of the cell's mesh using shape functions to determine where its boundaries lie.

Care must be taken to avoid inefficient calculation of collisions as it can easily turn into an n-body problem. The regular structure of the systems we intend to model can be leveraged in computational speedups. Because everything lies on a regular grid, it is trivial to precompute the cells located on the boundary of a solid object, and only evaluate collisions between these boundary cells. Additionally, it might be possible to group cells spatially on an octree for collision detection speedups.

The goal of collision detection within this thesis is to provide a qualitative assessment of interactions between assemblies and their environment. Current methods weigh efficiency and simplicity over accuracy. In its current state, no quantitative conclusions should be drawn from the collision detection. For example, evaluating this system at different time increments may alter the long term course of the state of the simulation. Accurate, quantitative modeling of interactions between solids is beyond the scope of this work.

Chapter 6

Implementation

Anisotropic Modeling of Engineered Bits and Atoms (AMOEBA), is a 3D CAD and simulation tool, based off the ideas from Chapters 3 and 5. In this tool, users construct assemblies of functional primitives and simulate their electronic and mechanical behaviors. It is available to demo online and the source code can be found on Github. AMOEBA primarily serves as a research tool for quickly simulating assemblies of functional digital material parts that we are developing in the lab. However, it's a very flexible platform, and I'm curious to see what other people might use it for. When it's in a more stable state I will write up some documentation and push it out to a wider audience. This chapter introduces information about the software implementation of AMOEBA.

6.1 Javascript/WebGL

AMOEBA was written in HTML5/JavaScript and currently hosted online. I chose the web as a development platform for this project because it allows easier access to the code than any other platform. Though user studies are not a component of this work, there is a long history of communities of users building things in sandbox environments that surpass anything the developers were able to imagine. There is a lot of talent beyond the immediate neighborhood of CBA, and I'd like to try to make this codebase accessible to anyone interested in exploring the design space around digital materials.

The web app was written with the following dependencies:

- Three.js is a library containing lots of useful classes for interacting with WebGL without getting bogged down in the details or sacrificing too much in performance.
- RequireJS is a framework for asynchronously loading JavaScript modules and dependencies.
- Backbone.js is a framework for managing UI events and giving structure to complex, interactive applications.

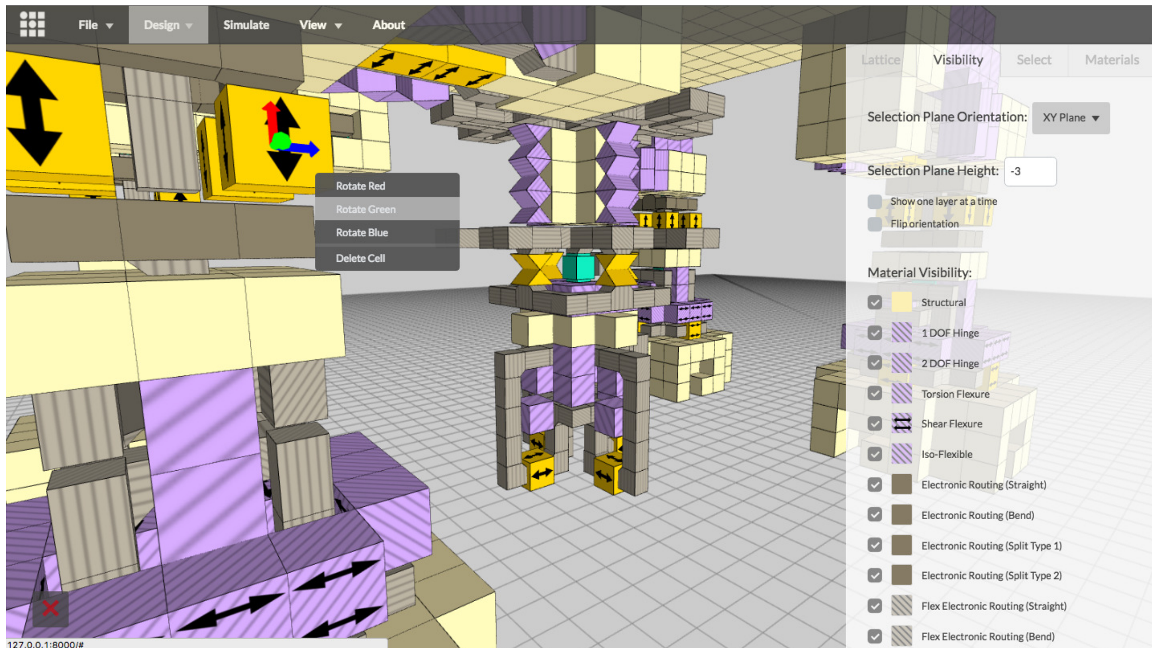


Figure 6-1: A schematic diagram of a robotic "pick and place" in AMOEBA. Cell rotation GUI allows anisotropic cells to be oriented in any direction.

- JQuery is a library that handles interactions between HTML and Javascript and helps maintain cross browser support of UI elements.
- Underscore is a library with lots of useful functions for dealing with arrays and JavaScript objects. Underscore also provides templating for Backbone.

6.2 GUI

AMOEBA borrows most of its GUI from DMDesign (Chapter 3), with a few exceptions. A cell rotation interface (Figure 6-1) allows users to rotate cells in 3D, so anisotropic cells may be oriented in any direction. Absolute cell orientations are saved as JSON when the assembly is saved. A 3D selection tool allows users to select large rectangular regions of space to fill with material, cut away material, and clone or mirror an existing structure into another region (Figure 6-2).

Anisotropic cells in AMOEBA are represented graphically with special meshes, illustrated in Figure 6-3. Some cells (e.g. the straight and bent conducting cells) appear to occupy a smaller volume than other cells, but this is meant only as a visualization of their properties. All cells fill the same volume and are joined mechanically with six face-connected neighbors. Isotropic and anisotropic materials may be edited or defined through the material editor interface (Figure 6-4).

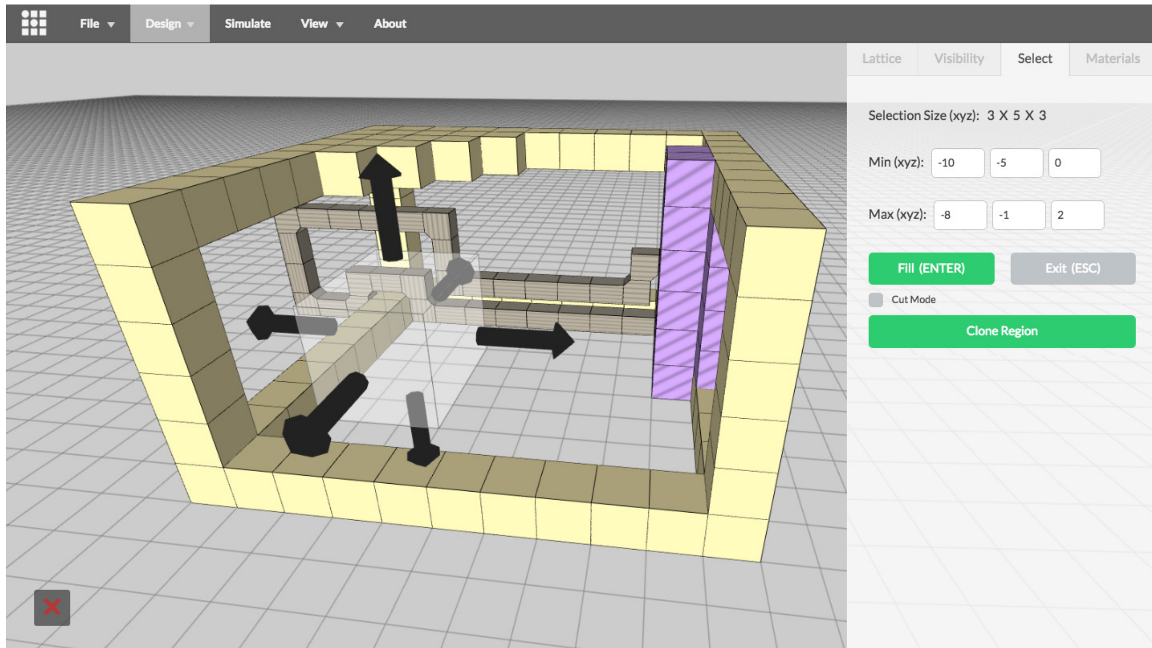


Figure 6-2: 3D selection tool allows for bulk cell additions and removals, and cloning or mirroring existing geometry.

6.3 GPU Programming

During the development of AMOEBA, typed arrays [1] were not performant for assemblies larger than ~30 cells (100 time steps forward per render), so I began running the bulk of the mathematical calculations on the GPU. Figure 6-5 shows a schematic representation of the GPU programming.

At the moment of writing this, mathematical programming on the GPU through the WebGL API is a bit of a hack. Data is passed in and out of the GPU cores as 2D arrays - RGBA image textures that would normally be used for rendering purposes. WebGL currently supports reading and writing Int8/16/32, uInt8/16/32 and Float32 textures. GPU programs are precompiled as *fragment shaders*, processes that are executed by a single core of the GPU to create a single pixel of output texture data. Each GPU core may access a number of input data textures but may only output one RGBA pixel to an output texture buffer. During the execution of a fragment shader program, a complete output texture is calculated by the available GPU cores in a highly parallel, pixel-wise process. A concise reference for the WebGL API is provided by the Khronos Group [3].

In AMOEBA, cell state and precomputed constants are stored in several data textures (Figure 6-5), and the next state of a given cell is evaluated in a single core of the GPU. It is not possible to fit all necessary updated state variables (position, velocity, orientation, and angular velocity, all in 3D) in a single RGBA pixel. This means that calculations for a single time step of the physics engine must be split into several,

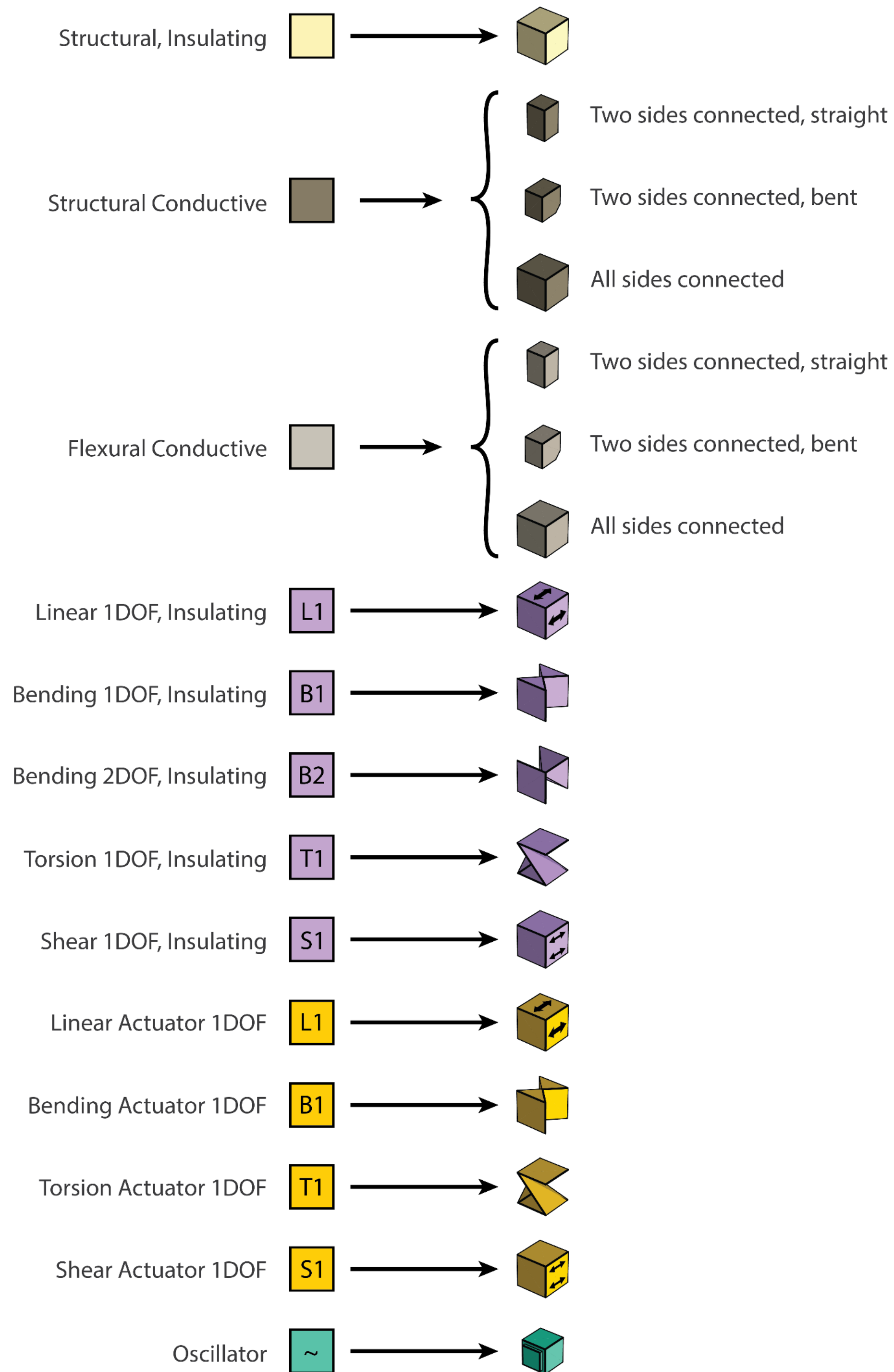


Figure 6-3: Meshes and textures used to represent different material types and convey the orientation of cell anisotropy.

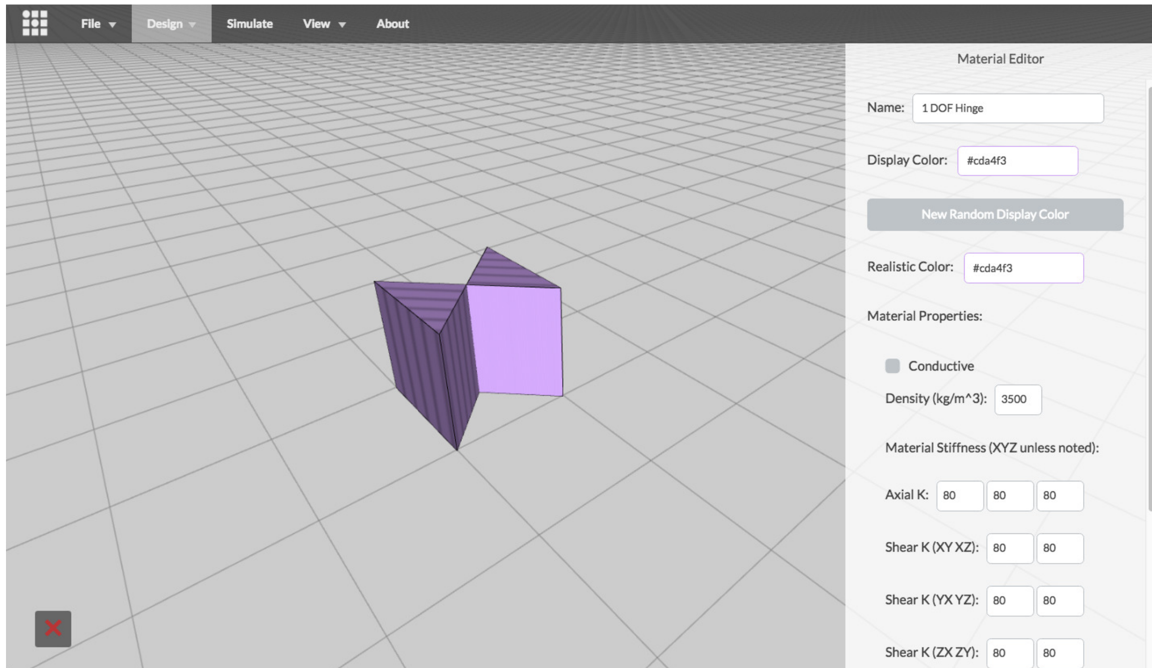


Figure 6-4: Material editor interface for a 1DOF bending flexure. All 15 stiffness and damping constants may be edited, along with density and conductivity.

sequential fragment shader programs (Figure 6-5).

After moving the simulation forward by some number of time steps, position and orientation information is passed out of the GPU to the main thread for a render call. Unfortunately, at this time only uInt8 textures may be transferred from the GPU to the CPU. An additional fragment shader program called "PackToBytes" converts the Float32 arrays storing position and orientation into uInt8 arrays, where each float in the original array is converted to four uInt8's. The process of transferring data from the GPU to the CPU is expensive, and in the fullness of time, could be avoided completely. AMOEBA is still very much under development and is not ready for these types of low level optimizations yet, but it is something to keep in mind in the following months.

Computing in the GPU speeds up the code significantly, but also creates some hardware dependencies that would not have otherwise been present. For example, some GPUs have a hard limit of 8 textures that can be loaded into memory at a time. In case of compatibility issues, a backup typed array implementation may be used at the expense of performance.

6.4 Other Performance Speedups

Additional speedups in mathematical operations increase runtime speed of the code. As demonstrated in Chapter 5, the math behind the mechanical modeling involves

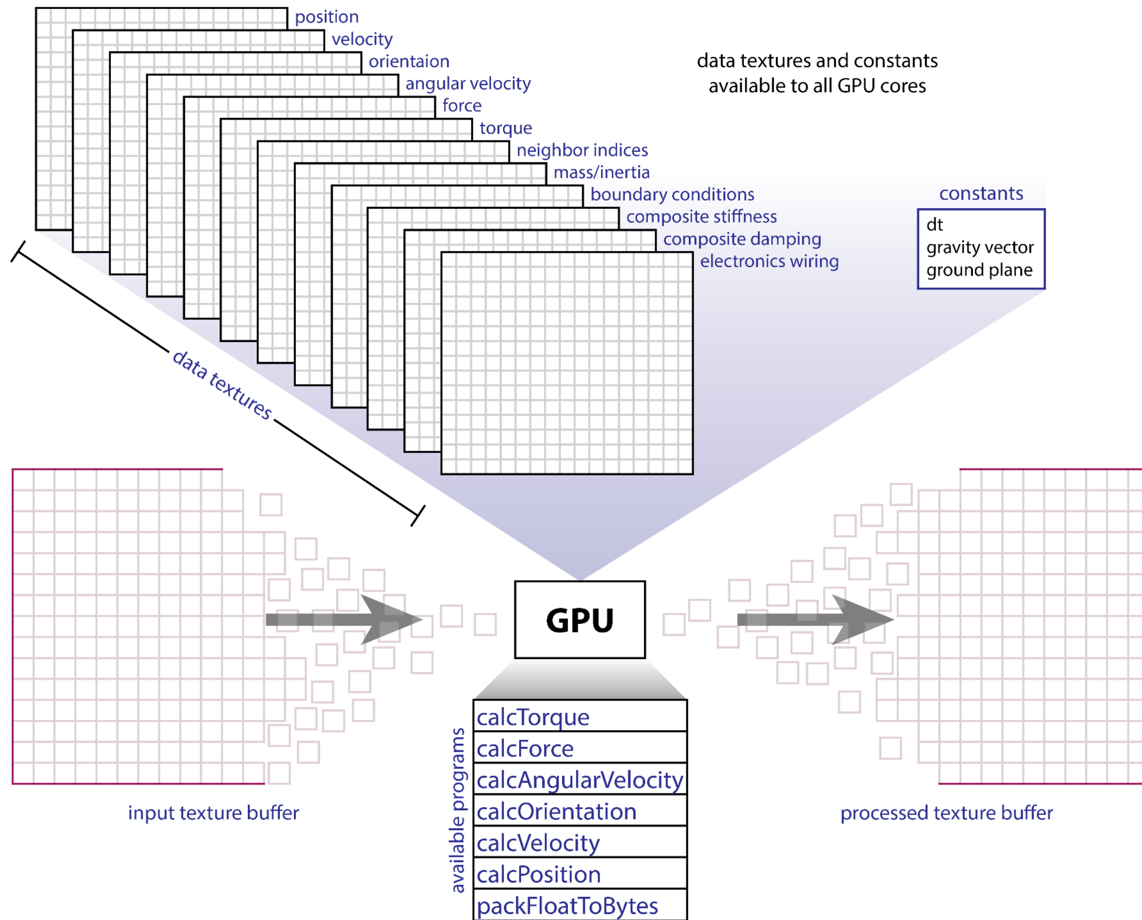


Figure 6-5: Schematic diagram of GPU programming in AMOEBA. 2D data textures store the current state of the cells, lookup tables for neighboring indices, and pre-computed composite stiffness and damping constants. These data textures as well as some global constants are passed into the GPU and are available to all cores. An input texture buffer is sent into the GPU, split up into RGBA pixels, and used to store the output state from each core. Several pre-compiled programs are available to the GPU, and are called sequentially during each time step of the simulation (with the exception of PackFloatToBytes, a program used to convert a Float32 texture to a uInt8 texture so that position and orientation can be extracted from the GPU for render).

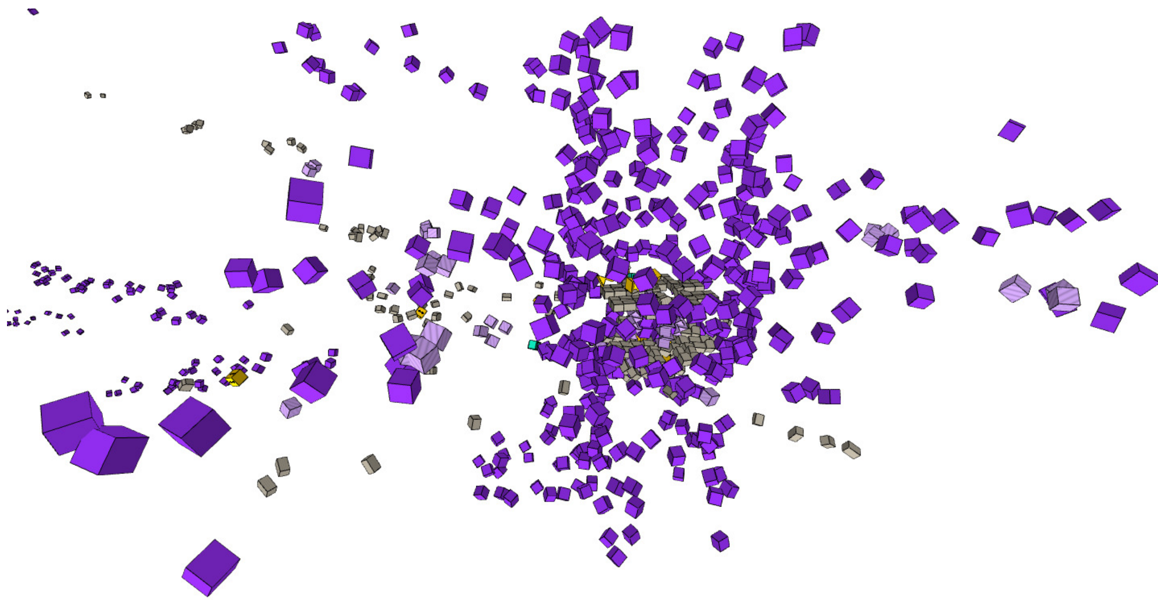


Figure 6-6: In its current state, AMOEBA is suffering from instability originating in bugs in the simulation code.

liberal use of quaternions. An efficient method of applying quaternions to vectors is given below:

$$t = 2 \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \times v$$

$$v_{rotated} = v + q_w t + \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \times t$$

This method uses fewer floating point operations than the standard Hamilton product from equation 5.12 [56].

6.5 Instability

In its current state, AMOEBA is suffering from issues of instability related to bugs in the simulation code. Additionally, rotational damping has not yet been implemented and its absence tends to throw the system unstable, especially in large, dense assemblies. This instability tends to look like an explosion (Figure 6-6); as spring systems go unstable they result in massive displacements. I expect this instability to be resolved in the coming months.

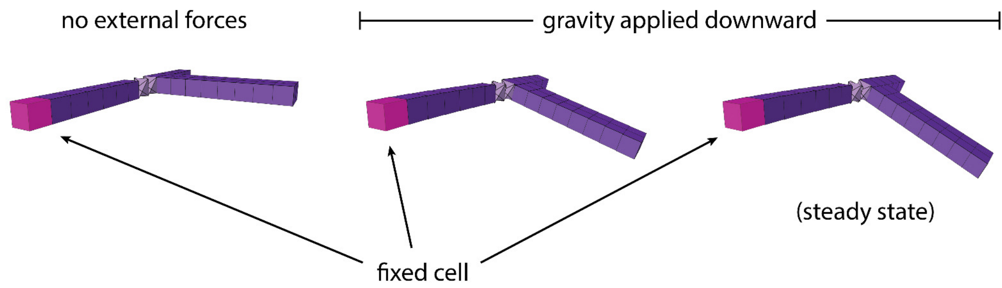


Figure 6-7: Simulation of a torsional flexure under a downward force from gravity shows appropriate anisotropic behavior. Purple are stiff cells, light purple is torsional flexures. Pink cell is fixed.

6.6 Examples

Due to instability, only certain configurations of parts are currently happy in AMOEBA. One example is a torsional flexure shown in Figure 6-7. An older version of the AMOEBA engine lacked a complete model of torsional and bending stiffness, but tended to be much more stable than the current state of development. Several examples of simple robotic elements were constructed within this older version of the physics engine (Figures 6-8 through 6-12).

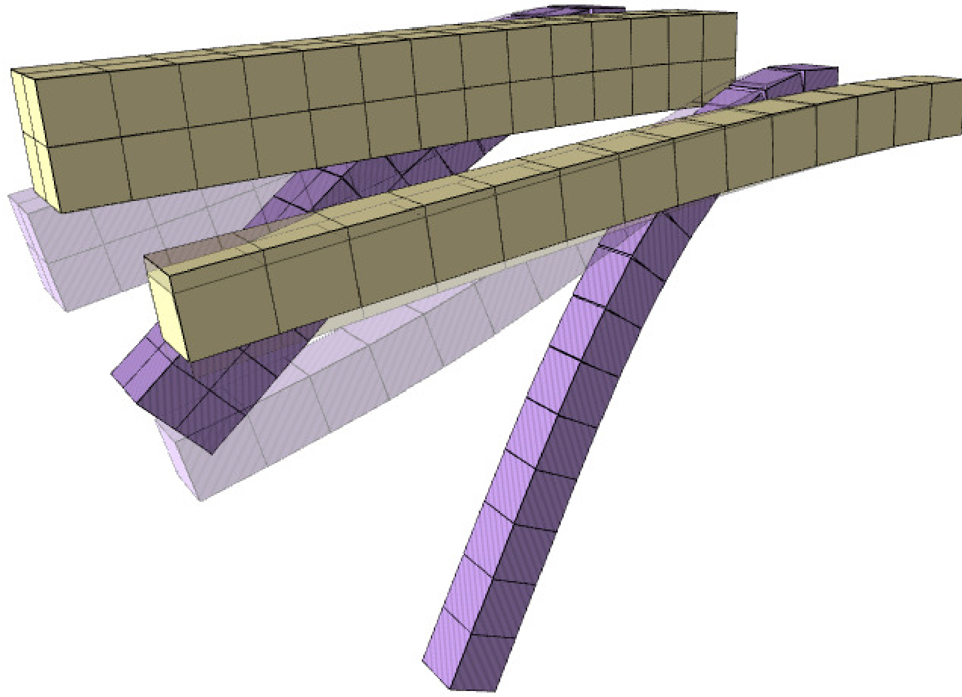


Figure 6-8: Beam bending simulation sets one end fixed and allows the rest to deform under gravity. Both material types shown are isotropic: white material is stiffer than purple.

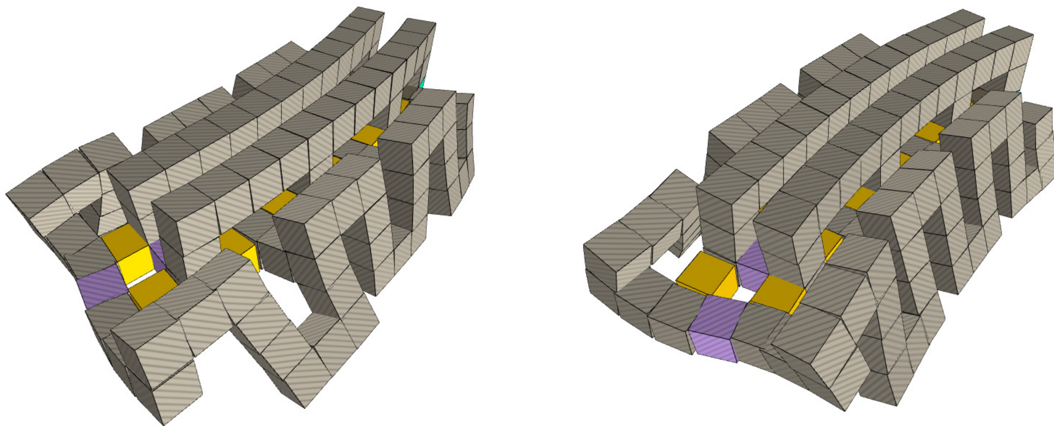


Figure 6-9: A bending structure made from two stacks of linear actuators driven out of phase from each other.

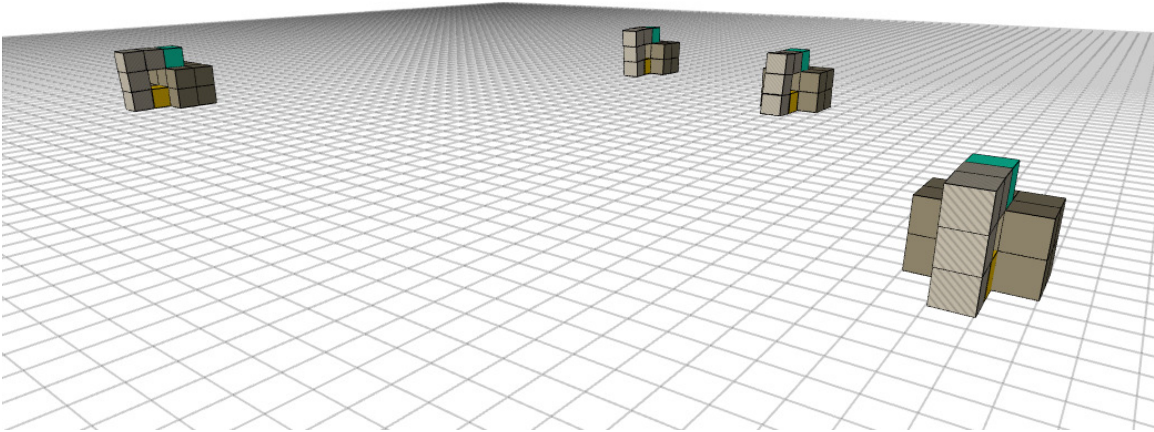


Figure 6-10: Simple locomotion demonstration shows four simple robots, each with a single linear actuator and oscillator. Differences in speed are due to different oscillator waveforms driving each robot's circuit. From left to right: sine, square, saw, triangle.

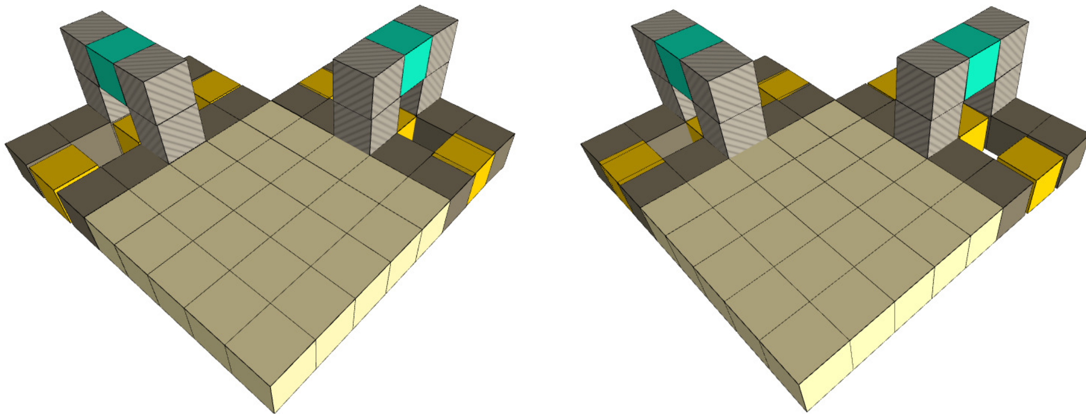


Figure 6-11: XY stage driven in a circular motion by two stacks of linear actuators a quarter cycle out of phase from each other.

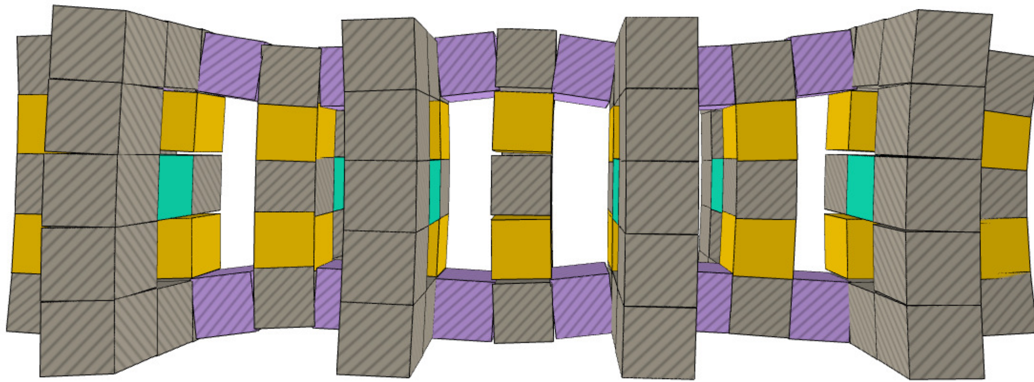


Figure 6-12: Undulating structure uses several stacks of linear actuators driven a third of a cycle out of phase from each other to create a sinusoidal motion up its spine.

Chapter 7

Evaluation

Comparison with existing FEA methods revealed a similarity between the cell-cell interaction model I developed in Chapter 5 and a 12DOF Timoshenko Beam Element (Section 7.1). My model ignores some Euler-Bernoulli bending components of the Timoshenko model, but also extends Timoshenko with non-linear handling of large angular deformations, rather than using small angle approximations. I argue that this is an appropriate model given the aspect ratio of the parts I am modeling (Figure 7-2) and the expected large angular deformations (Figure 7-3) of flexural joints. My model showed good qualitative agreement with COMSOL simulations, but more work is needed to run a quantitative analysis. Performance metrics are explored in Section 7.4.

7.1 Comparison with FEA Techniques

We can perform an analysis of the interactions between two cells using standard techniques from FEA to see how they compare with what we've derived in the previous sections. Within FEA, many types of models are used to describe the behavior of linear elastic solids with varying degrees of accuracy and computational complexity. One model that computes translational and rotational degrees of freedom between two nodes is the 12DOF Timoshenko beam. Using this model, we can construct a beam element between two nodes that represents the material joining two adjacent cells. The model determines the translational and rotational displacements of both nodes in 3D (Figure 7-1). The Timoshenko beam element takes into account axial deformation, bending deformation with shear effects, and torsional deformation.

Shape functions are polynomials in 1D, 2D, or 3D that describe how the behavior of the nodes should be interpolated in the regions between the nodes (e.g. the inner volume of the tetrahedron in Figure 5-3). Hermitian cubic shape functions are typically used for beam models; they are third order polynomials that provide continuity between discretized solutions along nodes in a beam. A graphical description of shape functions is shown in Wolfram [83]. The shape functions expressed in terms

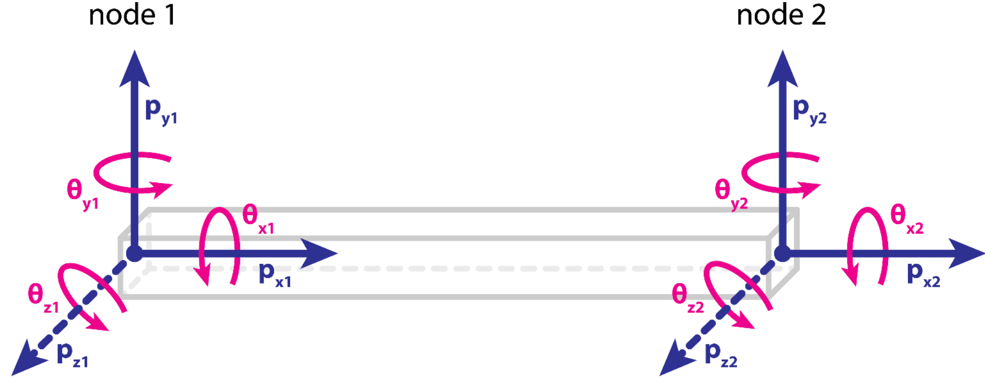


Figure 7-1: Setup of 12DOF beam model connecting two nodes (nodes 1 and 2). Translational displacement indicated by p_x, p_y, p_z and angular displacement indicated by θ_x, θ_y, θ_z.

of a dimensionless coordinate ξ are:

$$
\begin{aligned}
N_{x1} &= \tfrac{1}{4}(1-\xi)^2(2+\xi) \\
N_{x2} &= \tfrac{1}{4}(1+\xi)^2(2-\xi) \\
N_{\theta 1} &= \tfrac{1}{8}l(1-\xi)^2(1+\xi) \\
N_{\theta 2} &= \tfrac{1}{8}l(1+\xi)^2(1-\xi)
\end{aligned}
$$

where $-1 \leq \xi \leq 1$ and $\xi = -1$ at node 1 and $\xi = 1$ at node 2

Using these shape functions we can calculate the *stiffness matrix* (K) of the 12DOF Timoshenko beam oriented along the x-axis:

$$
K = \begin{bmatrix}
\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & \frac{-EA}{l} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{l^3(1+\phi_y)} & 0 & 0 & 0 & \frac{6EI_z}{l^2(1+\phi_y)} & 0 & \frac{-12EI_z}{l^3(1+\phi_y)} & 0 & 0 & 0 & \frac{6EI_z}{l^2(1+\phi_y)} \\
 & & \frac{12EI_y}{l^3(1+\phi_z)} & 0 & \frac{-6EI_y}{l^2(1+\phi_z)} & 0 & 0 & 0 & \frac{-12EI_y}{l^3(1+\phi_z)} & 0 & \frac{-6EI_y}{l^2(1+\phi_z)} & 0 \\
 & & & \frac{GJ}{l} & 0 & 0 & 0 & 0 & 0 & \frac{-GJ}{l} & 0 & 0 \\
 & & & & \frac{(4+\phi_z)EI_y}{l(1+\phi_z)} & 0 & 0 & 0 & \frac{6EI_y}{l^2(1+\phi_z)} & 0 & \frac{(2-\phi_z)EI_y}{l(1+\phi_z)} & 0 \\
 & & & & & \frac{(4+\phi_y)EI_z}{l(1+\phi_y)} & 0 & \frac{-6EI_z}{l^2(1+\phi_y)} & 0 & 0 & 0 & \frac{(2-\phi_y)EI_z}{l(1+\phi_y)} \\
 & & & & & & \frac{EA}{l} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_z}{l^3(1+\phi_y)} & 0 & 0 & 0 & \frac{-6EI_z}{l^2(1+\phi_y)} \\
 & & & & & & & & \frac{12EI_y}{l^3(1+\phi_z)} & 0 & \frac{6EI_y}{l^2(1+\phi_z)} & 0 \\
 & & symmetric & & & & & & & \frac{GJ}{l} & 0 & 0 \\
 & & & & & & & & & & \frac{(4+\phi_z)EI_y}{l(1+\phi_z)} & 0 \\
 & & & & & & & & & & & \frac{(4+\phi_y)EI_z}{l(1+\phi_y)}
\end{bmatrix}
$$

where

$$\phi_y = \frac{12EI_z}{GA_{sy}l^2} \qquad \text{and} \qquad \phi_z = \frac{12EI_y}{GA_{sz}l^2}$$

The stiffness matrix gives us a way to convert between translational/rotational nodal displacements and forces, according to the following equation:

$$\vec{F} = -K\vec{u} \tag{7.1}$$

where:

$$\vec{F} = \begin{bmatrix} F_{x1} \\ F_{y1} \\ F_{z1} \\ T_{x1} \\ T_{y1} \\ T_{z1} \\ F_{x2} \\ F_{y2} \\ F_{z2} \\ T_{x2} \\ T_{y2} \\ T_{z2} \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \\ \theta_{x1} \\ \theta_{y1} \\ \theta_{z1} \\ p_{x2} \\ p_{y2} \\ p_{z2} \\ \theta_{x2} \\ \theta_{y2} \\ \theta_{z2} \end{bmatrix}$$

where F_1, F_2 are translational forces acting on nodes 1 and 2, T are torques, p are translational displacements, and θ are rotational displacements. The negative sign was put in front of the stiffness matrix in Equation 7.1 to keep the same convention of restoring forces used in Chapter 5.

Multiplying through Equation 7.1 gives us 12 equations:

$$F_{x1} = -\frac{EA}{l}p_{x1} + \frac{EA}{l}p_{x2} \tag{7.2a}$$

$$F_{y1} = -\frac{12EI_z}{l^3(1+\phi_y)}p_{y1} - \frac{6EI_z}{l^2(1+\phi_y)}\theta_{z1} + \frac{12EI_z}{l^3(1+\phi_y)}p_{y2} - \frac{6EI_z}{l^2(1+\phi_y)}\theta_{z2} \tag{7.2b}$$

$$F_{z1} = -\frac{12EI_y}{l^3(1+\phi_z)}p_{z1} - \frac{6EI_y}{l^2(1+\phi_z)}\theta_{y1} + \frac{12EI_y}{l^3(1+\phi_z)}p_{z2} - \frac{6EI_y}{l^2(1+\phi_z)}\theta_{y2} \tag{7.2c}$$

$$T_{x1} = -\frac{GJ}{l}\theta_{x1} + \frac{GJ}{l}\theta_{x2} \tag{7.2d}$$

$$T_{y1} = \frac{6EI_y}{l^2(1+\phi_z)}p_{z1} - \frac{(4+\phi_z)EI_y}{l(1+\phi_z)}\theta_{y1} - \frac{6EI_y}{l^2(1+\phi_z)}p_{z2} - \frac{(2-\phi_z)EI_y}{l(1+\phi_z)}\theta_{y2} \tag{7.2e}$$

$$T_{z1} = -\frac{6EI_z}{l^2(1+\phi_y)}p_{y1} - \frac{(4+\phi_y)EI_z}{l(1+\phi_y)}\theta_{z1} + \frac{6EI_z}{l^2(1+\phi_y)}p_{y2} - \frac{(2-\phi_y)EI_z}{l(1+\phi_y)}\theta_{z2} \tag{7.2f}$$

$$F_{x2} = -F_{x1} \tag{7.2g}$$

$$F_{y2} = -F_{y1} \tag{7.2h}$$

$$F_{z2} = -F_{z1} \tag{7.2i}$$

$$T_{x2} = -T_{x1} \tag{7.2j}$$

$$T_{y2} = \frac{6EI_y}{l^2(1+\phi_z)}p_{z1} - \frac{(2-\phi_z)EI_y}{l(1+\phi_z)}\theta_{y1} - \frac{6EI_y}{l^2(1+\phi_z)}p_{z2} - \frac{(4+\phi_z)EI_y}{l(1+\phi_z)}\theta_{y2} \tag{7.2k}$$

$$T_{z2} = -\frac{6EI_z}{l^2(1+\phi_y)}p_{y1} - \frac{(2-\phi_y)EI_z}{l(1+\phi_y)}\theta_{z1} + \frac{6EI_z}{l^2(1+\phi_y)}p_{y2} - \frac{(4+\phi_y)EI_z}{l(1+\phi_y)}\theta_{z2} \tag{7.2l}$$

Combining Equations 7.2a and 5.5a gives us the x-axis spring component of Equation 5.17, assuming no relative rotation between the nodes:

$$F_{x1} = k_{axial_x}(p_{x2} - p_{x1})$$

Equations 7.2b and 7.2c have the same general form. Rearranging Equation 7.2b gives:

$$F_{y1} = \frac{12EI_z}{l^3(1+\phi_y)}(p_{y2} - p_{y1}) - \frac{6EI_z}{l^2(1+\phi_y)}(\theta_{z1} + \theta_{z2}) \tag{7.3}$$

The force F_{y1} acting on node 1 is a combination of contributions from bending and shear stiffnesses. Assuming shear dominance, we can reduce Equation 7.3 to:

$$F_{y1} = \frac{12EI_z}{l^3\phi_y}(p_{y2} - p_{y1}) - \frac{6EI_z}{l^2\phi_y}(\theta_{z1} + \theta_{z2})$$

$$F_{y1} = \frac{GA_{sy}}{l}(p_{y2} - p_{y1}) - \frac{GA_{sy}}{2}(\theta_{z1} + \theta_{z2})$$

Substituting Equation 5.5b gives:

$$F_{y1} = k_{shear_{xy}}(p_{y2} - p_{y1}) - k_{shear_{xy}}l\frac{(\theta_{z1} + \theta_{z2})}{2}$$

$$F_{y1} = k_{shear_{xy}}(p_{y2} - p_{y1} - l\theta_{z_{avg}})$$

Which is a small angle approximation of the y-axis spring component of Equation 5.17.

We'll account for the bending components of Equation 7.3 in Section 7.1.1.

As discussed in Equation 5.17, the translational forces acting on node 1 are equal and opposite those acting on node 2. This is summed up by Equations 7.2g, 7.2h, and 7.2i.

Combining Equations 7.2f and 5.5d gives us a small angle approximation to the x-axis rotational spring component of Equation 5.24 (in this case the small angle approximation reduces the first term of Equation 5.24 to zero, assuming that torques applied to node1 by node 2 have no x component):

$$T_{x1} = k_{torsion_x}(\theta_{x2} - \theta_{x1})$$

Equations 7.2e and 7.2f have the same general form. Rearranging Equation 7.2e gives:

$$T_{y1} = -\frac{6EI_y}{l^2(1+\phi_z)}(p_{z2} - p_{z1}) - \frac{EI_y}{l(1+\phi_z)}((4+\phi_z)\theta_{y1} + (2-\phi_z)\theta_{y2}) \tag{7.4}$$

As with Equation 7.3, this is a combination of shear and bending effects. Assuming shearing dominance reduces 7.4 to:

$$T_{y1} = -\frac{6EI_y}{l^2\phi_z}(p_{z2} - p_{z1}) - \frac{EI_y}{l}(\theta_{y1} - \theta_{y2}) \tag{7.5}$$

$$T_{y1} = -\frac{GA_{sz}}{2}(p_{z2} - p_{z1}) + \frac{EI_y}{l}(\theta_{y2} - \theta_{y1})$$

Substituting Equation 5.5b and 5.5c gives:

$$T_{y1} = \frac{-l}{2}k_{shear_{xz}}(p_{z2} - p_{z1}) + k_{bending_y}(\theta_{y2} - \theta_{y1})$$

which is a small angle approximation of the y-axis spring component of 5.24 with:

$$\vec{l}_{rot12} \approx \vec{l}_{nom12} = \begin{bmatrix} -l \\ 0 \\ 0 \end{bmatrix}$$

Following this same procedure with Equation 7.2f gives a similar result, but with an extra negative sign:

$$T_{z1} = (-)\frac{-l}{2}k_{shear_{xy}}(p_{y2} - p_{y1}) + k_{bending_z}(\theta_{z2} - \theta_{z1})$$

This negative is a byproduct of the crossproduct in Equation 5.24, according to the following unit vector relationships:

$$\hat{x} \times \hat{y} = \hat{z} \qquad \hat{x} \times \hat{z} = -\hat{y} \tag{7.6}$$

We'll account for the bending components of Equation 7.4 in Section 7.1.1.

Finally, Equation 7.2k and 7.2l have the same general form. Rearranging Equation

7.2k gives:

$$T_{y2} = -\frac{6EI_y}{l^2(1+\phi_z)}(p_{z2} - p_{z1}) - \frac{EI_y}{l(1+\phi_z)}((2-\phi_z)\theta_{y1} + (4+\phi_z)\theta_{y2}) \tag{7.7}$$

In the case of shearing dominance, 7.7 reduces to:

$$T_{y2} = -\frac{6EI_y}{l^2\phi_z}(p_{z2} - p_{z1}) - \frac{EI_y}{l}(\theta_{y2} - \theta_{y1})$$

$$T_{y2} = \frac{GA_{sz}}{2}(p_{z1} - p_{z2}) + \frac{EI_y}{l}(\theta_{y1} - \theta_{y2})$$

Substituting Equation 5.5b and 5.5c gives:

$$T_{y2} = \frac{l}{2}k_{shear_{xz}}(p_{z1} - p_{z2}) + k_{bending_y}(\theta_{y1} - \theta_{y2})$$

which is a small angle approximation of the y-axis spring component of 5.22 with:

$$\vec{l}_{rot21} \approx \vec{l}_{nom21} = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix}$$

Following this same procedure with 7.2l, we get:

$$T_{z2} = (-)\frac{l}{2}k_{shear_{xy}}(p_{y1} - p_{y2}) + k_{bending_z}(\theta_{z1} - \theta_{z2})$$

where again, the extra negative sign is accounted for by the cross product in Equation 5.22 and the relationships 7.6.

Again, we'll account for the bending components of Equation 7.7 in Section 7.1.1.

Through this process, we have also shown that for shearing dominance, y and z components of torque (Equations 7.2e, 7.2f, 7.2k, 7.2l) of one node are equal and opposite to the other node (Equation 5.23).

7.1.1 Conclusions

Compared to Timoshenko's equations, my model has the advantage of not using any small angle approximations. Since we are modeling flexural joints rather than structural solids, this non-linear treatment of angles is an asset. Figure 7-3 demonstrates the large angular deflections we expect to see in our parts.

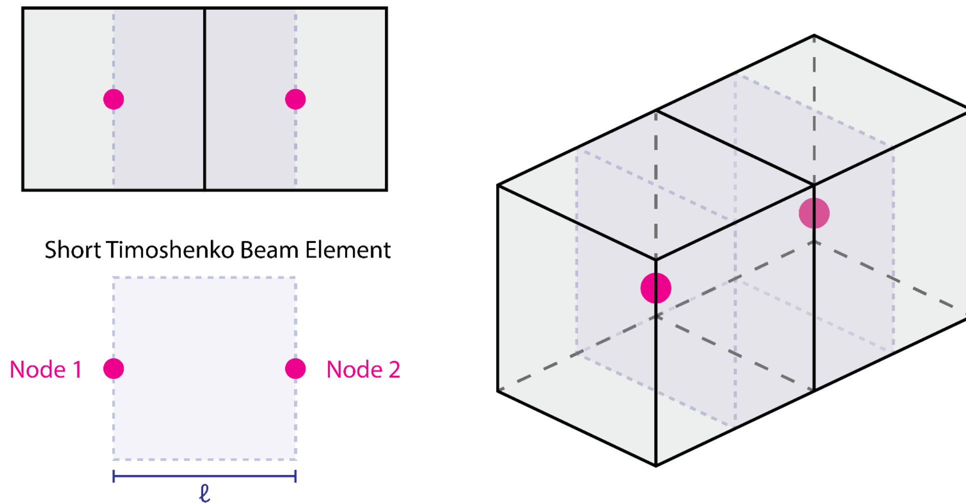


Figure 7-2: A diagram of the Timoshenko Beam Element joining two cells in my model. Due to the low length and relatively high cross sectional area of this "beam", I argue that shearing effect dominate bending effects in interactions between cells.

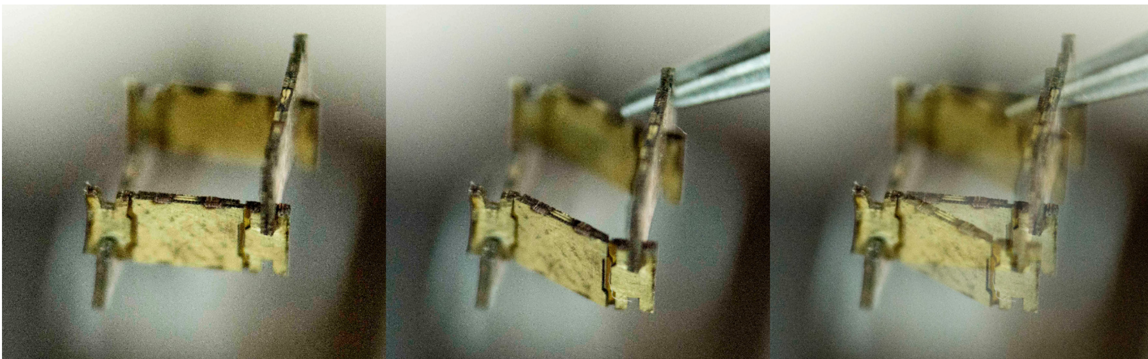


Figure 7-3: Large angular deflection of bending flexures in a small assembly of functional parts. *Image Credit: Will Langford 2016*

Missing Bending Contributions

Some bending components of Equations 7.2b, 7.2c, 7.2e, 7.2f, 7.2h, 7.2i, 7.2k, and 7.2l are absent from the model described in Chapter 5. Though the model still has a bending stiffness component, it is a special case of the 12DOF Timoshenko Beam Element where shearing dominates bending, and we can safely ignore the extra bending effects in a single cell-cell interaction. This seems appropriate considering the aspect ratio of the beam that we are using to model the two cells. As beams become shorter/thicker shearing dominates, and as beams become longer/thinner bending dominates [9]. If we are considering a perfectly cubic lattice, the length of the beam element that we can draw between two cells has a length equal to its height and width: the "beam" is a cube (Figure 7-2). According to Bower [9] there is no known accurate approximate theory of short beams.

However, I am somewhat concerned about potential effects of the missing bending components from my model and Timoshenko's. Though these components should not have much an effect on a single interaction between two cells, if we were to form a long thin beam of cube-shaped elements using my model, it's possible that the missing bending elements would stack up to create noticeable deviation from the Timoshenko model. Looking toward the future development of this model, it would be interesting to try using the complete Timoshenko Beam Element (Equations 7.2a through 7.2l) to model cell-cell interactions to see how the results compare for long, thin beams. My only concern is that as they are written in Equations 7.2a through 7.2l, they have many small angle approximations throughout. Another option could be to derive a model equivalent to Equations 7.2a through 7.2l without any small angle approximations.

That being said, it's also possible that adhering strictly to the Timoshenko model is not the best way to treat this system, especially since we will typically not be creating long, thin beams in practice. We should also consider the tradeoff between computational efficiency and accuracy - introducing extra bending components will add more floating point operations into each cycle of the simulation. If the effects of these bending components will rarely be seen, or result in a sufficiently small correction to the model, it might be fine to omit them. Further development of parts and empirical measurements will help to guide future modeling decisions.

7.2 Comparison with VoxCAD Physics Engine

The model behind VoxCAD uses a special case of the 12DOF Timoshenko Beam Element called the Euler-Bernoulli Beam Element [37]. The Euler-Bernoulli model assumes no shearing of a beam element joining two cells - it is a bending-dominated form of Timoshenko. For a beam oriented along the x-axis, using the same Hermitian cubic shape functions, we get the following stiffness matrix for an Euler-Bernoulli beam:

$$
K = \begin{bmatrix}
\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & \frac{-EA}{l} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{l^3} & 0 & 0 & 0 & \frac{6EI_z}{l^2} & 0 & \frac{-12EI_z}{l^3} & 0 & 0 & 0 & \frac{6EI_z}{l^2} \\
 & & \frac{12EI_y}{l^3} & 0 & \frac{-6EI_y}{l^2} & 0 & 0 & 0 & \frac{-12EI_y}{l^3} & 0 & \frac{-6EI_y}{l^2} & 0 \\
 & & & \frac{GJ}{l} & 0 & 0 & 0 & 0 & 0 & \frac{-GJ}{l} & 0 & 0 \\
 & & & & \frac{4EI_y}{l} & 0 & 0 & 0 & \frac{6EI_y}{l^2} & 0 & \frac{2EI_y}{l} & 0 \\
 & & & & & \frac{4EI_z}{l} & 0 & \frac{-6EI_z}{l^2} & 0 & 0 & 0 & \frac{2EI_z}{l} \\
 & & & & & & \frac{EA}{l} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_z}{l^3} & 0 & 0 & 0 & \frac{-6EI_z}{l^2} \\
 & & & & & & & & \frac{12EI_y}{l^3} & 0 & \frac{6EI_y}{l^2} & 0 \\
 & & \textit{symmetric} & & & & & & & \frac{GJ}{l} & 0 & 0 \\
 & & & & & & & & & & \frac{4EI_y}{l} & 0 \\
 & & & & & & & & & & & \frac{4EI_z}{l}
\end{bmatrix}
$$

This is equivalent to the 12DOF Timoshenko stiffness matrix with the shear terms ϕ removed. Multiplying through Equation 7.1 gives us 12 equations:

$$F_{x1} = -\frac{EA}{l}p_{x1} + \frac{EA}{l}p_{x2} \tag{7.8a}$$

$$F_{y1} = -\frac{12EI_z}{l^3}p_{y1} - \frac{6EI_z}{l^2}\theta_{z1} + \frac{12EI_z}{l^3}p_{y2} - \frac{6EI_z}{l^2}\theta_{z2} \tag{7.8b}$$

$$F_{z1} = -\frac{12EI_y}{l^3}p_{z1} + \frac{6EI_y}{l^2}\theta_{y1} + \frac{12EI_y}{l^3}p_{z2} + \frac{6EI_y}{l^2}\theta_{y2} \tag{7.8c}$$

$$T_{x1} = -\frac{GJ}{l}\theta_{x1} + \frac{GJ}{l}\theta_{x2} \tag{7.8d}$$

$$T_{y1} = \frac{6EI_y}{l^2}p_{z1} - \frac{4EI_y}{l}\theta_{y1} - \frac{6EI_y}{l^2}p_{z2} - \frac{2EI_y}{l}\theta_{y2} \tag{7.8e}$$

$$T_{z1} = -\frac{6EI_z}{l^2}p_{y1} - \frac{4EI_z}{l}\theta_{z1} + \frac{6EI_z}{l^2}p_{y2} - \frac{2EI_z}{l}\theta_{z2} \tag{7.8f}$$

$$F_{x2} = -F_{x1} \tag{7.8g}$$

$$F_{y2} = -F_{y1} \tag{7.8h}$$

$$F_{z2} = -F_{z1} \tag{7.8i}$$

$$T_{x2} = -T_{x1} \tag{7.8j}$$

$$T_{y2} = \frac{6EI_y}{l^2}p_{z1} - \frac{2EI_y}{l}\theta_{y1} - \frac{6EI_y}{l^2}p_{z2} - \frac{4EI_y}{l}\theta_{y2} \tag{7.8k}$$

$$T_{z2} = -\frac{6EI_z}{l^2}p_{y1} - \frac{2EI_z}{l}\theta_{z1} + \frac{6EI_z}{l^2}p_{y2} - \frac{4EI_z}{l}\theta_{z2} \tag{7.8l}$$

7.2.1 Conclusions

As I described in Section 7.1.1, a bending dominated beam model is more appropriate for long, thin beams. Given the aspect ratio of the elements being modeled in VoxCAD (very similar to what I've been modeling), the absence of shearing likely causes deflections to be underestimated [9]. Additionally, VoxCAD relies on the same small angle approximations present in the Timoshenko model.

That being said, my model draws enormous inspiration from VoxCAD, specifically in the handling of composite stiffness and damping constants for multimaterial junctions (Equations 5.9, 5.10), calculation of maximum time step Δt (Equations 5.25 and 5.26), and translational actuation model. Furthermore, seeing the performance that VoxCAD was able to achieve made me more confident to embark on this work in the first place.

7.3 Comparison with COMSOL Simulation

COMSOL is a professional multiphysics package that uses FEA to solve problems in solid mechanics, optics, analog electronics and more. I ran a simple passive flexural part model (Figure 7-4) and an actuated model (Figure 7-5) in COMSOL and in AMOEBA to compare their results.

I started by using the same geometric representation I'm using in AMOEBA, shown in Figure 7-4B. Using this minimal geometry, I had trouble decoupling bending and axial stiffness in COMSOL (explained in Section 7.3.1), so I switched to a more geometrically detailed model, shown in Figure 7-4C. In the second model, the geometry lowers area moment of inertia (I) and causes lower bending stiffness with the same E. In the second simulation I was forced to increase E beyond reason for both my material types to make the simulation work. I'm still not sure why COMSOL was having so much trouble with this.

Additionally in both my COMSOL simulations I could not find an appropriate constraint to fix the angle of the free end of the part, so its face is always oriented normal to the applied force. Instead I used a roller constraint, which has the added effect of constraining the part to be 9mm wide. This added extra axial strain to the part as it deforms, which we would not expect to see if the part were in the configuration shown in Figure 7-3. The dotted line in Figure 7-4A indicates lateral displacement of the free end of the part from its starting width of 9mm.

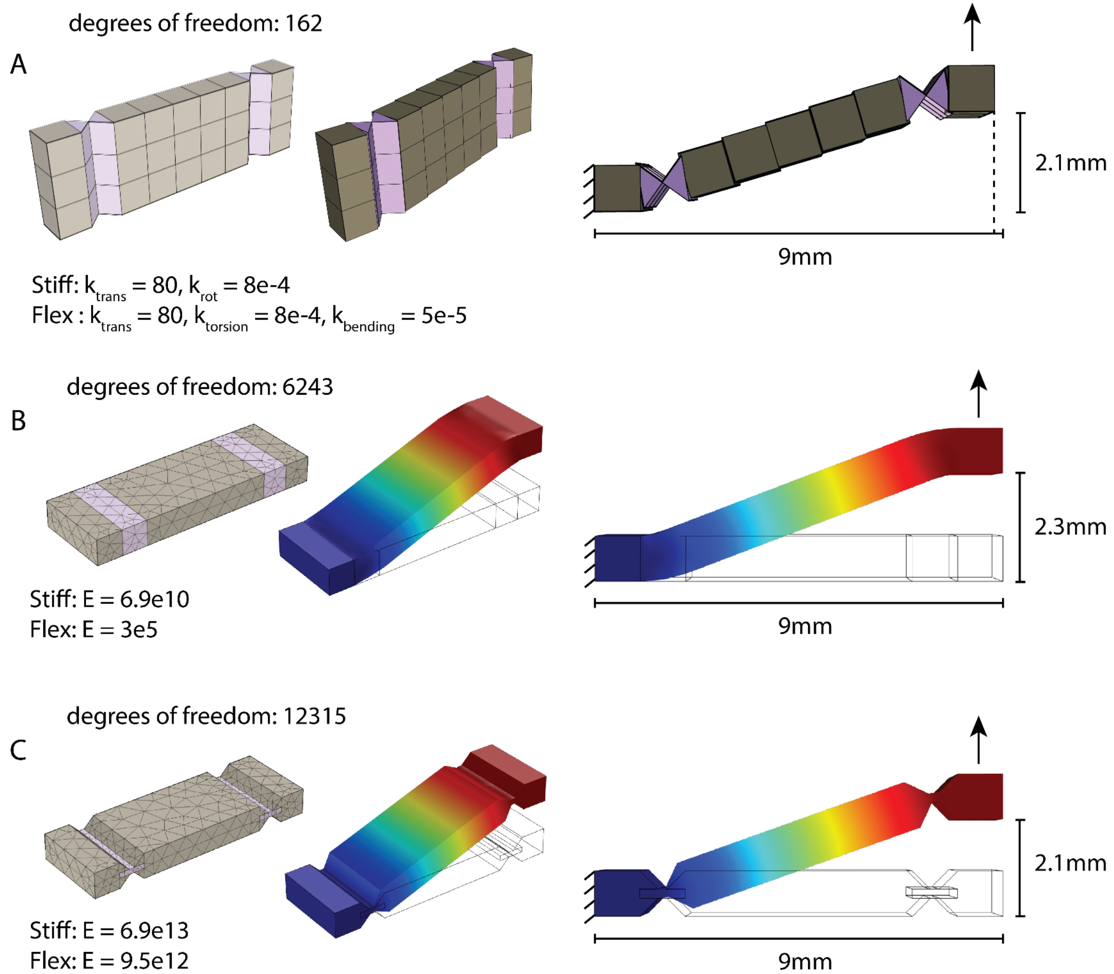


Figure 7-4: Simulation of a 0.012N translational force on a shear flexure functional part in AMOEBA **(A)** and in COMSOL using a simple geometric model **(B)** and a more detailed model **(C)**. Real parts shown in Figure 7-3. Note - the special meshes used for the 1DOF bending flexures in **A** are meant for visualization only, the underlying geometry is assumed to be identical to **B**.

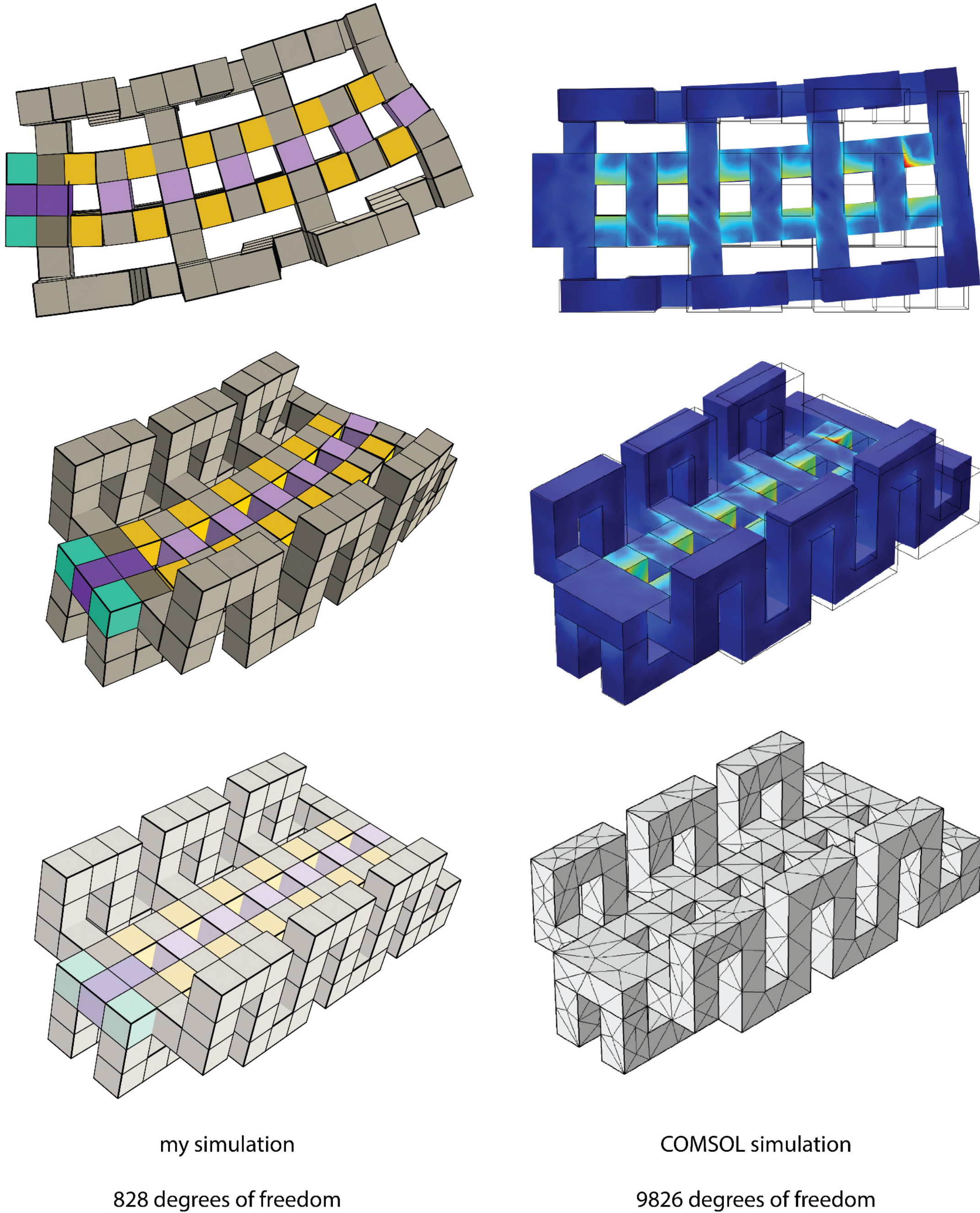


Figure 7-5: Qualitative comparison of actuated behavior in a bending actuator made from two stacks of piezos. I was not able to get enough control over the strain of the piezos in COMSOL to due a meaningful qualitative analysis on this system.

I ran an additional actuated simulation in COMSOL using its piezoelectric module. Unfortunately I found it difficult to control the strain of the piezos and couldn't achieve nearly the same strain that I've been modeling in my linear actuator block types. Still, I found nice qualitative agreement between the COMSOL model and my simulation (Figure 7-5).

7.3.1 Conclusions

Though I was able to run the shear flexure part simulation in AMOEBA without running into instability errors, through the process of doing this I noticed that the stiffnesses I'm using in my simulation are significantly less stiff than the materials we will be modeling in reality. For example, the elastic modulus of aluminum is 6.9e13Pa. For a 1x1x1mm cube that's an axial stiffness of 69MN/m, I'm using an axial and shear stiffnesses of 80N/m in the model in Figure 7-4. You can see a noticeable amount of shearing occurring from this low stiffness, but stiffnesses approaching anything near 69MN/m will run extremely slow. I saw the same difference of about 5 orders of magnitude in my bending stiffnesses as well. I will have to dig into this number more to be sure it's not a scaling error on my part, but it's possible we will have to scale down some of the stiffnesses in our model to help the solver move along. Because of this difference in scaling (and also previously mentioned roller constraints), it is difficult to do a quantitative analysis, but my modeling does appear to be giving the correct qualitative behavior in the examples shown in Figure 7-4 and 7-5.

In AMOEBA I represent functional primitives as geometrically simple elements with isotropic and anisotropic material properties deriving from their internal structure. This way we can discretize the model with a relatively low resolution mesh and increase the efficiency of the simulation. This is reflected by the number of degrees of freedom solved using each simulation technique (Figure 7-4); my method resulted in only 162 degrees of freedom for this part, while the full geometrically detailed model required 12315.

My model gives a convenient abstraction of degrees of freedom within flexural systems with hooks into low level control over material stiffnesses and other physical properties. Though it is entirely possible to simulate large assemblies of our functional parts in COMSOL or other multiphysics packages, the time required to setup these models makes them rarely worthwhile. My tool is bounded by its assumption of voxelized geometry, but in systems for which this voxelization is appropriate, it offers a simple way to compose assemblies of parts in minutes rather than hours.

Decoupling Axial/Bending and Shear/Torsion

Axial and bending stiffness typically use the same elastic modulus (E) in their formulations (Equations 5.5a and 5.5c) and are therefore coupled to each other (e.g. you can't pick a value of E that gives low bending stiffness and high axial stiffness at the same time for simple cubic geometry). However, some composite materials exhibit

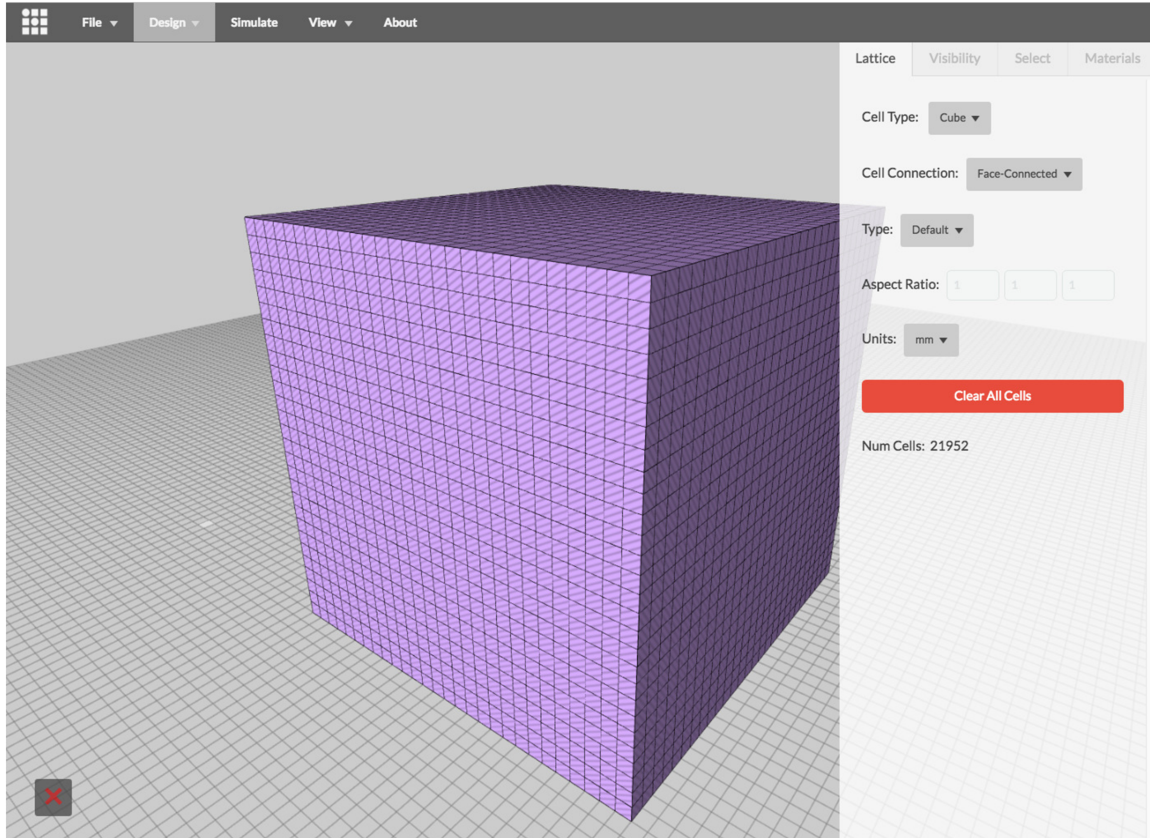


Figure 7-6: A 28x28x28 cube containing 21952 cells.

these behaviors - Calisch showed that it is possible to build a structure that is stiff in axial compression and flexible in bending around one axis from four part types [12]. This DOF coupling applies similarly to shear and torsional stiffness sharing the same shear stiffness G (Equations 5.5b and 5.5d). If we are trying to model a functional primitive that has complex internal geometric structure, we may need to decouple these degrees of freedom.

The only way to do this in COMSOL is to derive your own custom stiffness matrix for each of the anisotropic parts you wish to model or explicitly model their anisotropic geometries. Defining your own stiffness matrix for each of these cell types requires a high degree of technical knowledge from the end user and essentially defeats the purpose of using a professional package; at that point you might as well write your own solver (the situation I'm in). Building a model of the substructures of each anisotropic cell type requires setting up even more material properties, geometries, and finer meshing of the model - slower from a user perspective and slower computationally.

7.4 Performance Metrics

I performed a series of measurements to get a better understanding of where time was being spent during each complete render cycle of the simulation (simulation + threejs rendering). I performed these tests on assemblies ranging from 1 cell to 21952 cells. Since the amount of operations per simulation steps is proportional to the number of connections between cells, I ran these tests on cube-shaped assemblies (Figure 7-6), so the inner cells are all fully connected to their six neighbors. This represents a "worst case scenario" performance-wise because cells are maximally connected, assemblies we will typically be modeling will not be as dense. Smaller assemblies have the extra advantage that they have a greater surface area to volume ratio and are therefor proportionally more sparsely connected than larger assemblies of cells, but these effects drop off quickly.

I ran these tests using a AMD Radeon HD 6490M graphics card with 160 cores, clocked somewhere between 700-800MHz. I set the threejs rendering window size to 1200x650px on a 1440x900px monitor and zoomed out so that all blocks were displayed in the rendering window. Assemblies ranged in size from 1x1x1 to 28x28x28, or 21,952 cells.

Each complete render cycle consists of 3 stages: simulation, data transfer, and rendering. The length of the simulation stage depends on the number of time steps that are being solved in the current render cycle. Figure 7-7 shows plots for 50, 100, and 150 steps per cycle. The data transfer stage is when the data from the final time step of simulation is pulled off the GPU and onto the CPU. The rendering stage is when a render call is sent to threejs and graphics are rendered to the screen.

7.4.1 Conclusions

Since I was timing my code in the CPU, my attempts at separately timing the simulation stage and data transfer stage were unreliable. However, for a given assembly the time required to complete the data transfer stage is not dependent on the number of steps computed in the simulation stage. So I timed the total length of both stages while varying the number of steps per simulation stage and and plotted it is Figure 7-8. I fit a line to these plots and extracted a slope from each line. From this I was able to determine that the time to solve a single cell's state for one time step on the GPU is approximately 0.2μs. I used the following equation:

$$\text{sim time per cell per } \Delta t = \frac{m_{150} - m_{50}}{100}$$

Figure 7-9 shows the breakdown of a complete cycle in terms of time spent in simulation, data transfer, and rendering. Data transfer takes about 30ms; this could be cut down by updating mesh positions and orientations on the GPU and bypassing data transfer completely. Rendering speed could be improved greatly by removing invisible internal geometry and merging multiple geometries into a single threejs mesh. I'm

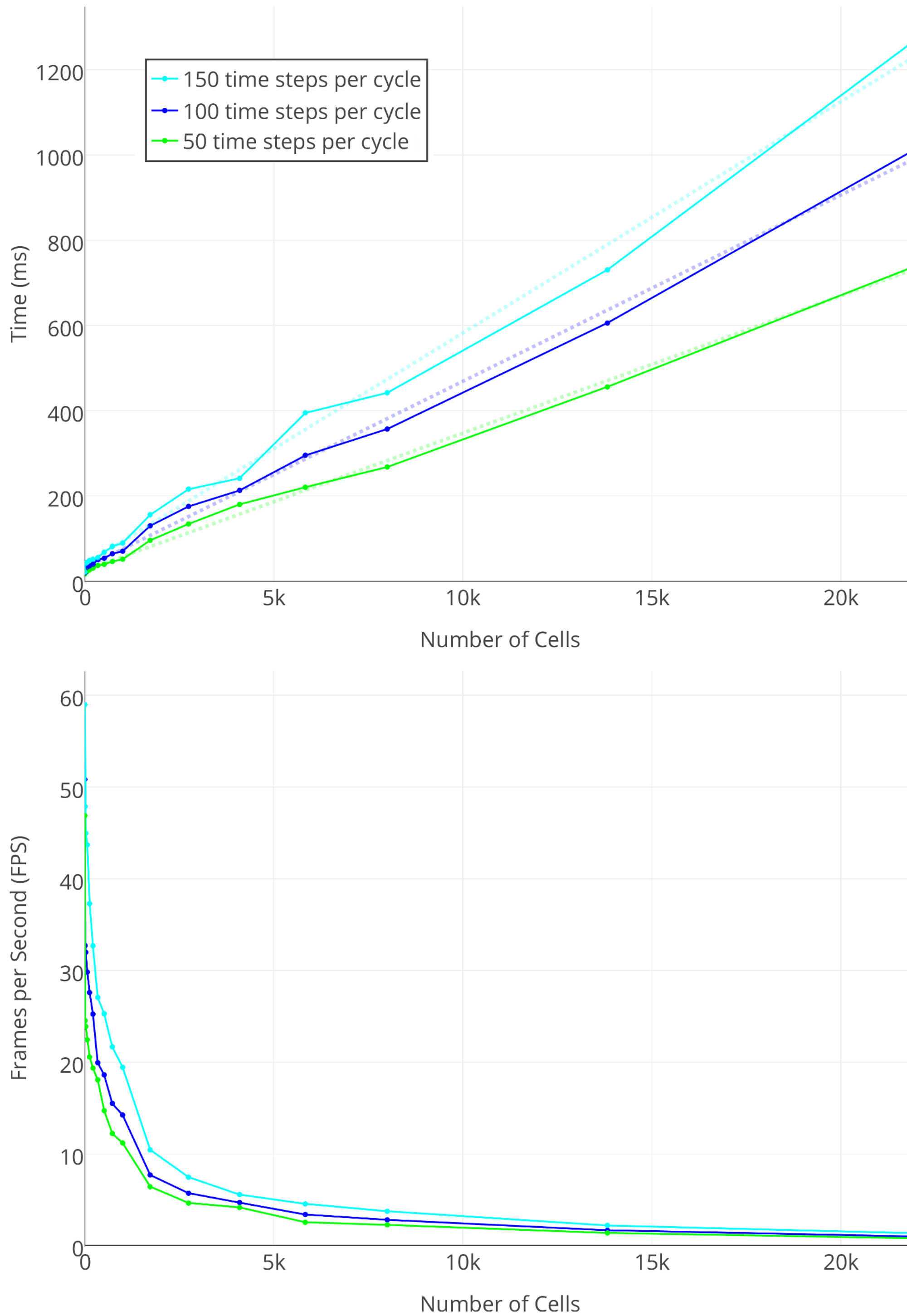


Figure 7-7: Render cycle speed shown in ms (top) and in FPS (bottom) for a series of assemblies spanning dimensions 1x1x1 to 28x28x28. The number of simulation time steps solved per cycle was varied: 50 time steps (green), 100 time steps (navy), and 150 time steps (light blue). Linear fits to top plot shown with dotted lines.

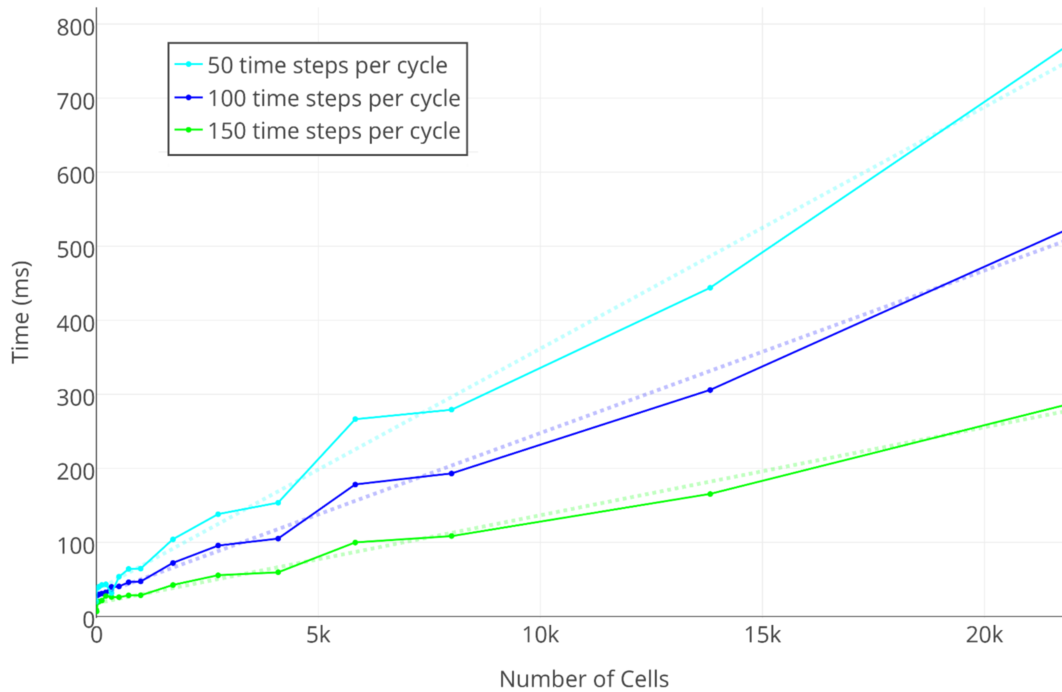


Figure 7-8: Duration of simulation and data transfer stages plotted for 50 time steps (green), 100 time steps (navy), and 150 time steps (light blue). Linear fits plotted with dotted lines.

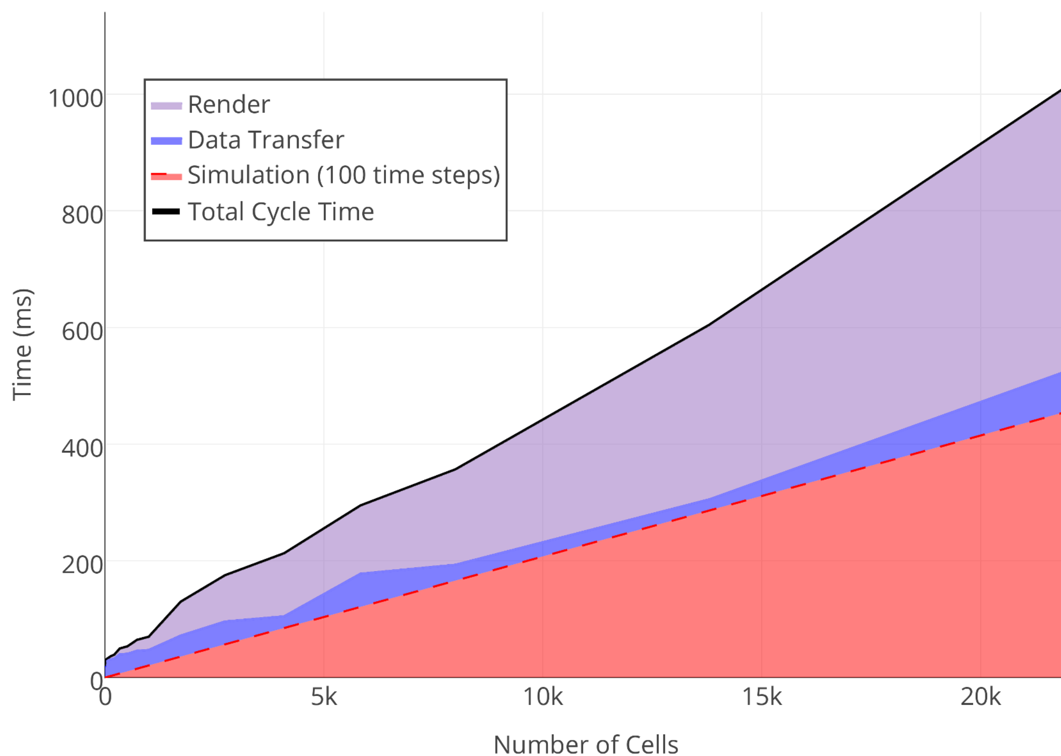


Figure 7-9: A breakdown of a full cycle of simulation (simulation + data transfer + rendering) shows where time was spent. Time spent in 100 cycles of simulation (salmon), data transfer (blue), and rendering in threejs (purple) are shown in milliseconds. Time to complete simulation was not measured directly, it was determined from analysis of Figure 7-8.

actually computing each interaction between cells twice - once when I process one cell, and again when I process its neighbor. Since the forces and torques between neighboring cells are symmetric, no new information is gleaned from these extra calculations. Removing this redundancy will decrease simulation processing time by some factor less than 0.5 depending on the connectivity of the assembly. These optimizations will come after I'm through the debugging phase of this project.

7.5 Final Conclusions

The model developed in Chapter 5 contains most of the components of the 12DOF Timoshenko beam model, but it is missing some Euler-Bernoulli bending components. Though the contributions from these components are small for any interaction between two cells, they may result in noticeable accumulated error in long, thin beams formed from functional primitive parts. Further development of the model could introduce these extra bending terms while still maintaining non-linear handling of angular deformations. An advantage of my model for simulating flexural joints is its non-linear treatment of angular displacements - allowing for large angular deformations to be simulated without costly remeshing. Comparison with COMSOL simulation revealed that current material stiffness are far lower than stiffnesses to be used in our proposed system. In order to maintain efficient solving of the dynamic simulation, we should not increase stiffnesses more than about 1 order of magnitude. Empirical testing is needed to determine how much of an effect this is having on our simulation results. As the parts are scaled down and geometric stiffness of our parts decreases, this problem disappears, but at the current scale, we may need to develop methods to overcome this discrepancy.

Chapter 8

Future Work

Future work revolves around hierarchical extensions of the current CAD/simulation model, experiments in computational design optimization, and development of a video game around digital materials.

8.1 Hierarchical Simulation

The hierarchical scaling of structure and function in our assembly system (Chapter 4) hints to a hierarchical framing of design and simulation in software. The bulk of this thesis explores CAD/simulation at the function-level, especially the decomposition of functional parts into functional primitives. Future work will explore design and simulation at other levels of hierarchy and translate behaviors of assemblies at one level into higher-level building blocks.

Current thoughts around the translation of *assemblies of elements* to *functional primitives* are depicted in Figures 8-1 and 8-2. Mechanical parameters describing a functional primitive are calculated from static simulations of assemblies of elements. Boundary conditions applied to elements at each face of an assembly induce steady-state deformations; 15 sets of boundary conditions apply forces from which $F = kx$ and $T = k\theta$ may be solved for all 15 stiffness constants. Additionally, the inertia tensor and mass are computed directly from constituent material densities and geometry. For example, differences in composition between the two 1DOF bending flexures depicted in Figure 8-1 are expressed as differences in stiffness constants, mass, and inertia at the functional primitive level of description (Figure 8-2). In the case that mechanical deformations are not linear over a range of applied forces, a constant k is calculated to best fit the data.

Electronic properties may be determined as well. Conductance between faces of a functional primitive are trivially calculated from assemblies of conducting and insulating elements. Static capacitance and inductance are computed using steady-state solutions to Finite Difference Time Domain (FDTD) simulations; previous work in DMDesign used FTDT to calculate potential fields (Figure 3-1E).

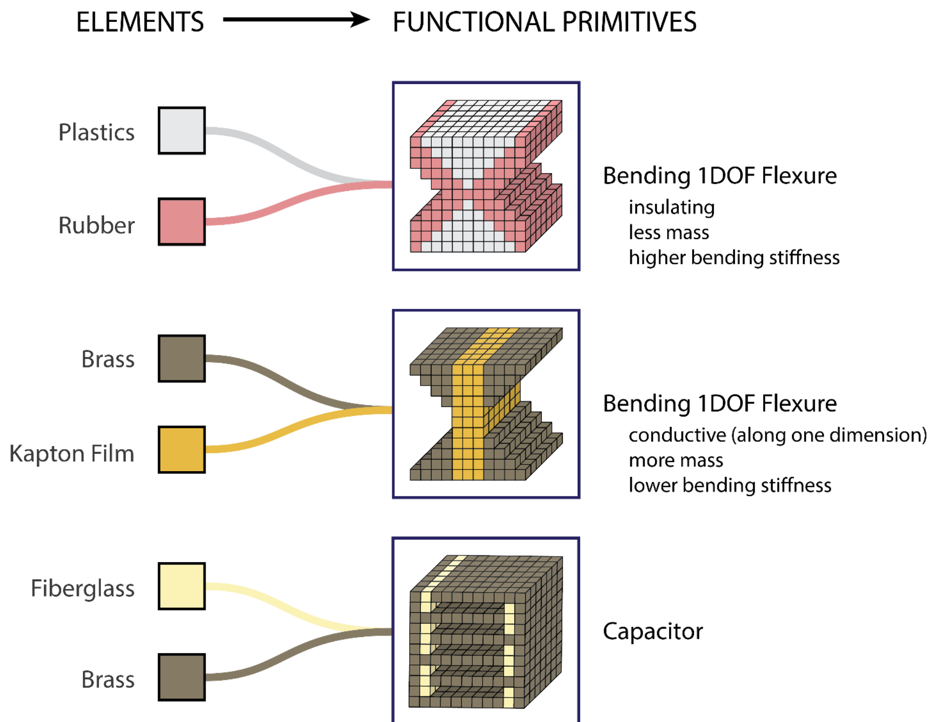


Figure 8-1: Three functional primitives decomposed as assemblies of elements. The geometrical layout of elements within a functional primitive establish its mechanical and electronic properties. Two sketches of a 1DOF Bending Flexure have different mass, moment of inertia, conductivity, and bending stiffnesses (top, middle). Parallel plate construction of a capacitor from conducting and insulating parts determines the functional primitive's global capacitance (bottom).

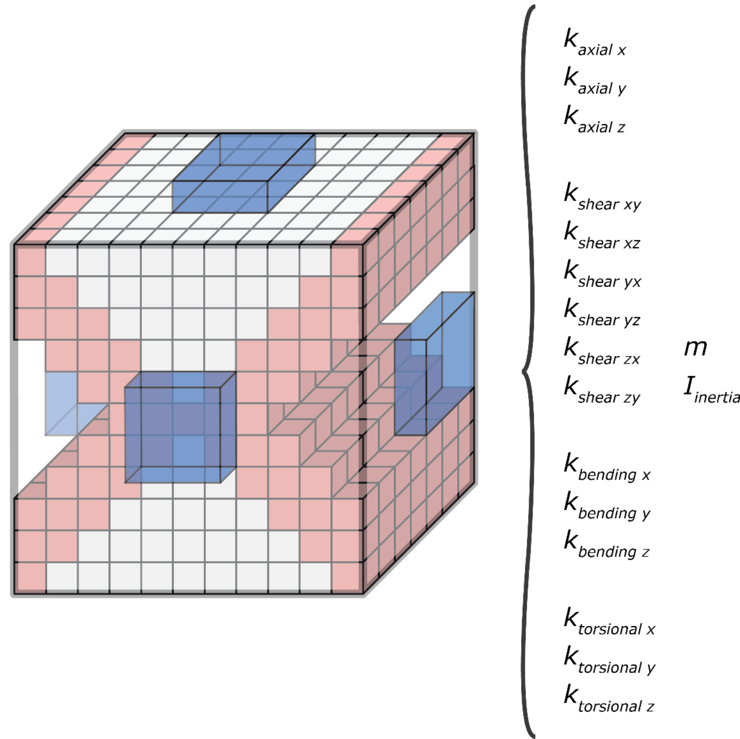


Figure 8-2: Schematic view of the translation of an assembly of elements into functional primitive parameters: 15 stiffness constants, mass, and moment of inertia tensor. Blue boxes indicate "attachment points" for applied boundary conditions in multidimensional stiffness measurements.

Modules and complexes exhibit high-level behaviors that are abstracted from the low-level physics of the system. Simulation of modules and complexes may reflect this abstraction by modeling interactions with a discretized, kinematic CA ruleset (somewhat like the work of William Stevens with CBlocks3D [74]). At this time it is still unclear how to generate a module's governing ruleset from behaviors of assemblies of function-level parts.

8.2 Declarative Design

A long term goal of this work is to close the loop between CAD and simulation and move towards *declarative design*. In a declarative design workflow, users specify high-level goals and a constrained optimization process searches across parameter space for a solution. The discretization of space and finite set of parts types at each level hierarchy within our assembly system sets it up nicely for constrained design optimization.

Once a method of translating assemblies of elements into functional primitives has

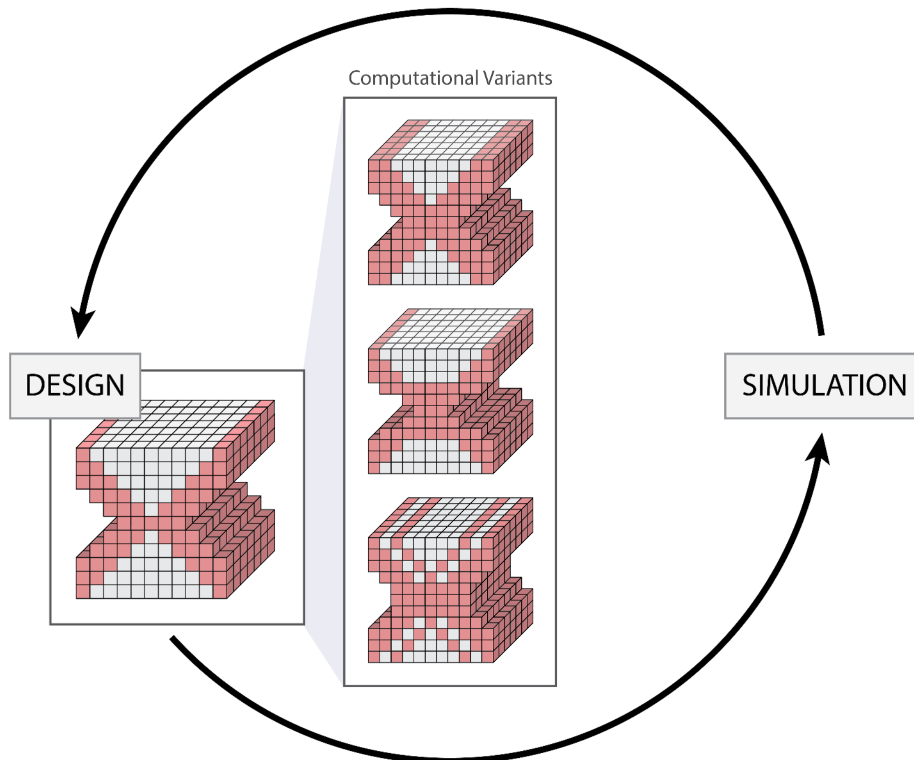


Figure 8-3: A 1DOF bending flexure is optimized in a topology optimization process. Variations on initial user-generated design are evaluated in simulation. This design/simulation loop is repeated until fabrication constraints and high-level goals are satisfied.

been established (see discussion from previous section), we can perform topology optimization across assemblies of elements to discover new types of functional primitives (Figure 8-3). Fabrication limitations should be translated into geometric constraints to ensure that the outcomes of the optimization are reasonable. Future work could also explore optimization of control strategies for actuated assemblies, similar to previous work by Sims [67].

8.3 "Fab the Game"

An offshoot of the work described in this thesis is a collaboration with E-Line Media on a game. Tentatively called "Fab the Game", it explores an aspirational future where digital materials are used to construct nearly everything. In the game, players construct assemblies from functional primitives in an open-ended sandbox environment and in more directed challenges. We envision a large component of gameplay revolves around constructing robots, operating them, and using them to shape the surrounding environment.

The game will extend the physical simulation outlined in Chapter 5 through realtime user interaction. For example, actuators in the game can be mapped to the keyboard and other gaming controllers so players can operate their robots in realtime. Complex robots may be driven by a combination of preprogrammed controls and user input. Using these controls, players test the agility and speed of their robot in arena challenges, drawing inspiration from the competitions of First Robotics.

More advanced modes of gameplay borrow from the ideas discussed in the previous two sections. Players dive into functional primitive definitions and alter their elemental composition to unlock new functional properties beyond those available by default.

Through this game, we hope to familiarize players with CAD and simulation tools, constraint-based machine design, controls, and design optimization. We will also provide off-ramps to digital fabrication workflows so that designs may be realized IRL ("in real life") by gamers/makers. In order to ease the burden of building large assemblies of parts, we may introduce concepts in hierarchical and parametric design within the crafting interface. We are curious to see what players are able to construct within the game and hope it might inform future research trajectories at CBA.

Concept art by Eli Gershenfeld explores the look and feel of the game as well as potential gameplay scenarios (Figures 8-4 through 8-9).

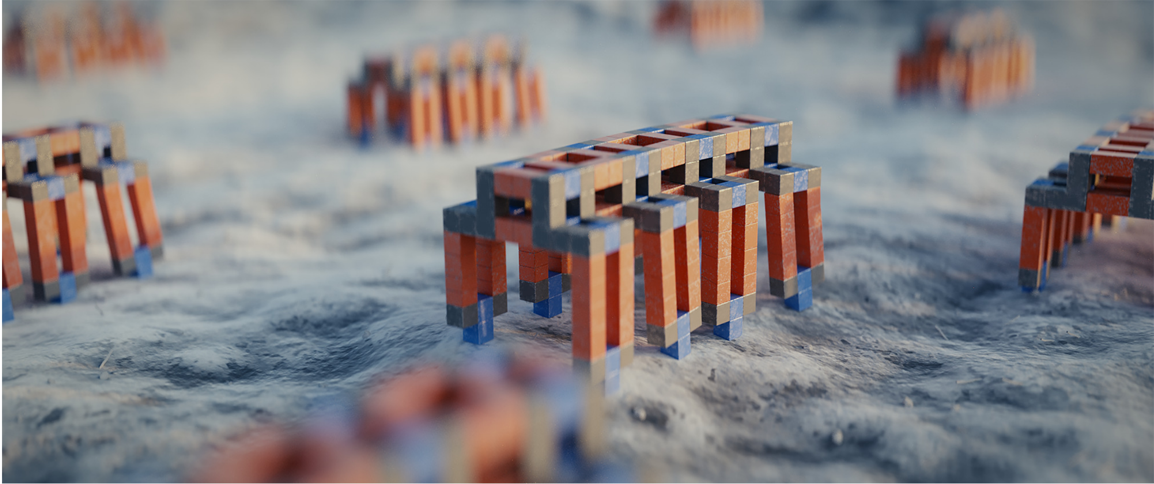


Figure 8-4: A swarm of locomoting robots. *Image Credit: Eli Gershenfeld 2016*

Figure 8-5: Assembler assembling an assembler. *Image Credit: Eli Gershenfeld 2016*

Figure 8-6: Assembler “factory” feedstock. *Image Credit: Eli Gershenfeld 2016*

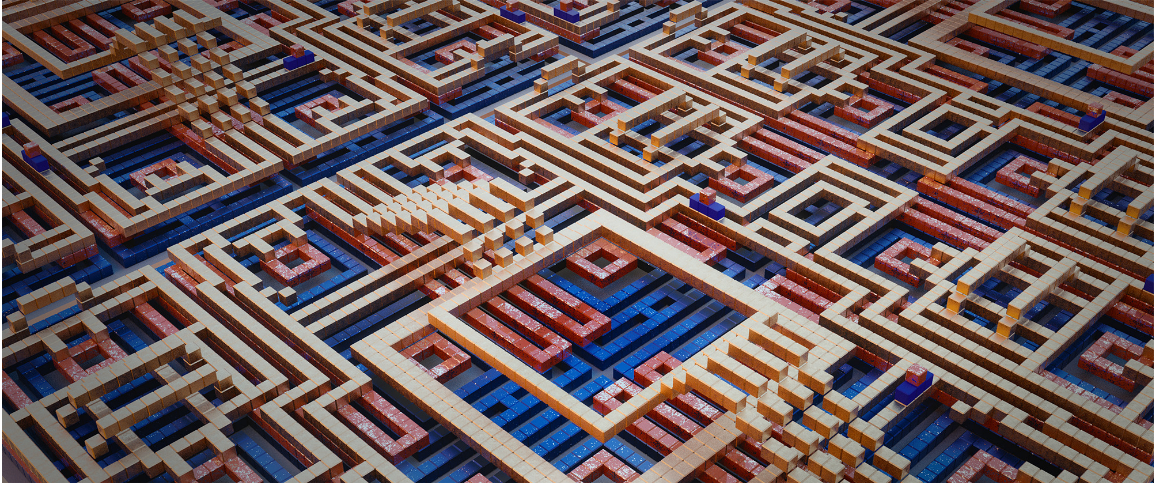


Figure 8-7: Distributed “factory” automation. *Image Credit: Eli Gershenfeld 2016*

Figure 8-8: “Living” macro-scale structure. *Image Credit: Eli Gershenfeld 2016*

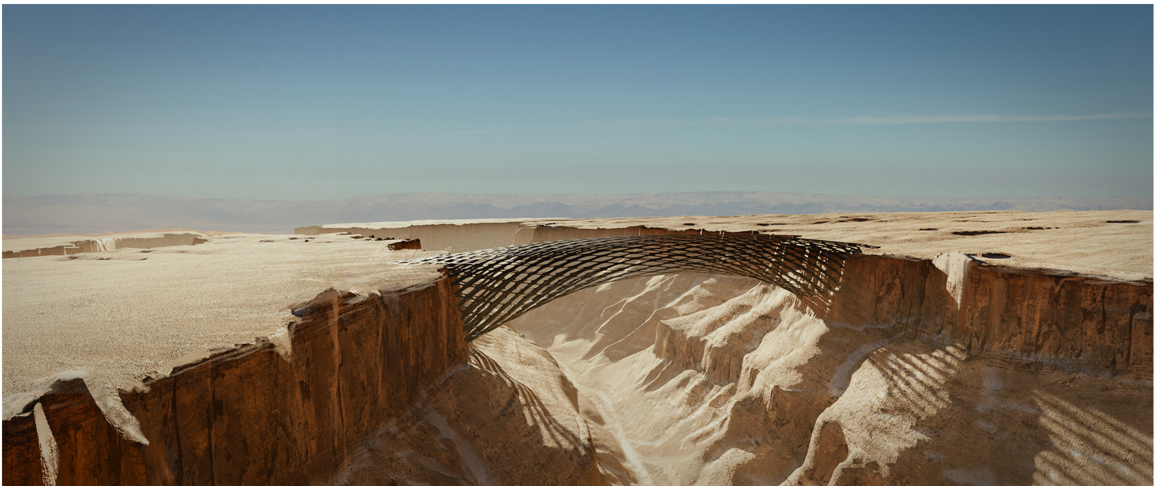


Figure 8-9: Self-assembling bridge. *Image Credit: Eli Gershenfeld 2016*

References

[1] JavaScript Typed Arrays, 2016. [url].

[2] Life Wiki, 2016. [url].

[3] WebGL Reference Card, 2016. [url].

[4] L M Adleman. Molecular computation of solutions to combinatorial problems. *Science (New York, N.Y.)*, 266(5187):1021–1024, 1994. [url].

[5] Bok Y Ahn, Eric B Duoss, Michael J Motala, Xiaoying Guo, Sang-Il Park, Yujie Xiong, Jongseung Yoon, Ralph G Nuzzo, John a Rogers, and Jennifer a Lewis. Omnidirectional printing of flexible, stretchable, and spanning silver microelectrodes. *Science*, 323(5921):1590–1593, 2009. [url].

[6] Rutgers and UCSD/SDSC. RCSB Protein Data Bank, 2016. [url].

[7] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-It: Optimizing Moment of Inertia for Spinnable Objects. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 33(4), 2014. [url].

[8] Elwyn R. Berekamp, John H. Conway, and Richard K. Guy. *Winning Ways For Your Mathematical Plays*. Academic Press, 1982. [url].

[9] Allan F Bower. *Applied Mechanics of Solids*. 2009. [url].

[10] Luciano Brocchieri and Samuel Karlin. Protein length in eukaryotic and prokaryotic proteomes. 33(10):3390–3400, 2005. [url].

[11] Arthur W. Burks and John Von Neumann. *Von Neumann's Self-Reproducing Automata*. 1969. [url].

[12] Samuel Eli Calisch. Physical Finite Elements, 2014. [url].

[13] Matthew Eli Carney. Discrete Cellular Lattice Assembly, 2015. [url].

[14] Paul Chapman. Live Universal Computer, 2002. [url].

[15] Nicholas Cheney, Jeff Clune, and Hod Lipson. Evolved Electrophysiological Soft Robots. *ALIFE 14: Procedings of the Forteenth International Conference on the Synthesis and Simulation of Living Systems*, 2014. [url].

[16] Nick Cheney and Josh C Bongard. Evolving Soft Robots in Tight Spaces. *GECCO*, 2015.

[17] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation - GECCO '13*, page 167, 2013. [url].

[18] Kenneth Cheung and Neil Gershenfeld. Reversibly Assembled Cellular Composite Materials. *Science*, pages 1219–1221, 2013. [url].

[19] G R Cowper. The Shear Coefficient in Timoshenko's Beam Theory. *Journal of Applied Mechanics*, 6(4):5–10, 1966. [url].

[20] Hendrik Dietz, Shawn M Douglas, and William M Shih. Folding DNA into twisted and curved nanoscale shapes. *Science (New York, N.Y.)*, 325(5941):725–730, 2009.

[21] Shawn M. Douglas, Adam H. Marblestone, Surat Teerapittayanon, Alejandro Vazquez, George M. Church, and William M. Shih. Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Research*, 37(15):5001–5006, 2009.

[22] Brice Due. OTCA Metapixel - How does it work?, 2006. [url].

[23] Harold P Erickson. Size and Shape of Protein Molecules at the Nanometer Level Determined by Sedimentation , Gel Filtration , and Electron Microscopy. *Biological Proceedures Online*, 11(1):32–51, 2009. [url].

[24] Achim Flammenkamp. Most seen natural occuring ash objects in Game of Life, 2004. [url].

[25] Anthony C Forster and George M Church. Towards synthesis of a minimal cell. *Molecular Systems Biology*, 2, 2006.

[26] Martin Gardner. Mathematical Games: The fantastic combination of John Conway's new solitare game "life". *Scientific American*, 1970. [url].

[27] Amanda Ghassaei. OTCA Metapixel - Conway's Game of Life, 2015. [url].

[28] Daniel G Gibson, John I Glass, Carole Lartigue, Vladimir N Noskov, Ray-Yuan Chuang, Mikkel a Algire, Gwynedd a Benders, Michael G Montague, Li Ma, Monzia M Moodie, Chuck Merryman, Sanjay Vashee, Radha Krishnakumar, Nacyra Assad-Garcia, Cynthia Andrews-Pfannkoch, Evgeniya a Denisova, Lei Young, Zhi-Qing Qi, Thomas H Segall-Shapiro, Christopher H Calvey, Prashanth P Parmar, Clyde a Hutchison, Hamilton O Smith, and J Craig Venter. Creation of a bacterial cell controlled by a chemically synthesized genome. *Science (New York, N.Y.)*, 329(5987):52–56, 2010. [url].

[29] John I Glass, Nacyra Assad-Garcia, Nina Alperovich, Shibu Yooseph, Matthew R Lewis, Mahir Maruf, Clyde a Hutchison, Hamilton O Smith, and J Craig Venter. Essential genes of a minimal bacterium. *Proceedings of the National Academy of Sciences of the United States of America*, 103(2):425–430, 2006. [url].

[30] R.Wm. Gosper. Exploiting regularities in large cellular spaces. *Physica D: Nonlinear Phenomena*, 10(1-2):75–80, 1984. [url].

[31] Dave Greene. Linear Propagator, 2013. [url].

[32] Dave Greene. Re:Geminoid Challenge, 2013. [url].

[33] E Hawkes, B An, N M Benbernou, H Tanaka, S Kim, E D Demaine, D Rus, and R J Wood. Programmable matter by folding. *Scientific American*, 107(28), 2010. [url].

[34] Dean Hickerson. Description of sliding block memory, 1990. [url].

[35] Jonathan Hiller and Hod Lipson. Design and analysis of digital materials for physical 3D voxel printing. 2(November 2008):137–149, 2009. [url].

[36] Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012. [url].

[37] Jonathan Hiller and Hod Lipson. Dynamic Simulation of Soft Multimaterial 3D-Printed Objects. *Soft Robotics*, 1(1):88–101, 2014. [url].

[38] Jonathan D Hiller and Hod Lipson. Fully Recyclable Multimaterial Printing. 2005. [url].

[39] Clyde A Hutchison, Ray-yuan Chuang, Vladimir N Noskov, Nacyra Assad-garcia, Thomas J Deerinck, Mark H Ellisman, John Gill, Krishna Kannan, Bogumil J Karas, Li Ma, James F Pelletier, Zhi-qing Qi, R Alexander Richter, Elizabeth A Strychalski, Lijie Sun, Yo Suzuki, Billyana Tsvetanova, Kim S Wise, Hamilton O Smith, John I Glass, Chuck Merryman, Daniel G Gibson, and J Craig Venter. Design and synthesis of a minimal bacterial genome. *Science*, 2016. [url].

[40] R. Juvinal and K. Marshek. *Fundamentals of Machine Component Design (Table 5.1)*. 2006. [url].

[41] Yonggang Ke, Luvena L Ong, William M Shih, and Peng Yin. Three-Dimensional Structures Self-Assembled from DNA Bricks. *Science*, 338(November):1177–1183, 2012. [url].

[42] Yonggang Ke, Luvena L Ong, Wei Sun, Jie Song, Mingdong Dong, William M Shih, and Peng Yin. DNA brick crystals with prescribed depths. *Nat Chem*, 6(11):994–1002, 2014. [url].

[43] Mohammadreza Khorasaninejad, Wei Ting Chen, Robert C. Devlin, Jaewon Oh, Alexander Y. Zhu, and Federico Capasso. Metalenses at visible wavelengths: Diffraction-limited focusing and subwavelength resolution imaging. *Science*, 352(6290), 2016. [url].

[44] Do-nyun Kim, Fabian Kilchherr, Hendrik Dietz, and Mark Bathe. Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures. 40(7):2862–2868, 2012. [url].

[45] Do Nyun Kim, Fabian Kilchherr, Hendrik Dietz, and Mark Bathe. Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures CANDO. *Nucleic Acids Research*, 40(7):2862–2868, 2012.

[46] Will Langford, Amanda Ghassaei, and Neil Gershenfeld. Automated Assembly of Electronic Digital Materials. *ASME*, pages 3–12, 2016. [url].

[47] William Kai Langford. Electronic Digital Materials, 2014. [url].

[48] Minecraft Wiki. Minecraft Blocks, 2016. [url].

[49] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc, Englewood Cliffs, N. J., 1967. [url].

[50] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966. [url].

[51] L. S. Penrose. Mechanics of Self-Reproduction. *Annals of Human Genetics*, 1958. [url].

[52] Justin P Peters and L James Maher. DNA curvature and flexibility in vitro and in vivo. 43(1):23–63, 2014. [url].

[53] Charles P. Pool and Frank J. Owens. *Introduction to Nanotechnology*. John Wiley & Sons, 2003. [url].

[54] George A Popescu, Tushar Mahale, and Neil Gershenfeld. Digital materials for digital printing. pages 1–4. [url].

[55] William Poundstone. *The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge*. Dover Publications, 1985. [url].

[56] Stefan Reinalter. A Faster Quaternion-Vector Multiplication, 2016. [url].

[57] Paul Rendell. A Turing Machine in Conway's Game of Life, 2000. [url].

[58] John W Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3D M-Blocks : Self-reconfiguring Robots Capable of Locomotion via Pivoting in Three Dimensions. *IEEE International Conference on Robotics and Automation*, pages 1925–1932, 2015. [url].

[59] ROS. Universal Robot Description Format, 2016. [url].

[60] Paul W K Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006. [url].

[61] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, (March), 2014. [url].

[62] Christian Schumacher and Bernd Bickel. Microstructures to Control Elasticity in 3D Printing. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH*, 34, 2015. [url].

[63] Nadrian Seeman. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 1982. [url].

[64] Luke Shaeffer. A Physically Universal Cellular Automaton, 2014. [url].

[65] Shopbot. OpenSBP, 2016. [url].

[66] Stephen A. Silver. No Life Lexicon, 2016. [url].

[67] Karl Sims. Evolving Virtual Creatures. *Siggraph*, 1994. [url].

[68] Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. Computational design of actuated deformable characters. *ACM Transactions on Graphics*, 32(4):1, 2013. [url].

[69] J C Slater. Atomic Radii in Crystals. 3199(May 2016), 1964. [url].

[70] D R Smith, Willie J Padilla, D C Vier, and S Schultz. Composite Medium with Simultaneously Negative Permeability and Permittivity. *Physical Review*, pages 1–4, 2000. [url].

[71] Statasys. Objet1000 Data Sheet. *Stratasys*, pages 1–2, 2012. [url].

[72] William M. Stevens. Simulating Self-replicating Machines. *Journal of Intelligent and Robotic Systems*, 49(2):135–150, 2007. [url].

[73] William M Stevens. Adapting Gospers Hashlife Algorithm for Kinematic Environments. *CoSMoS 2010: Proceedings of the 2010 Workshop on Complex Systems Modelling and Simulation*, 2010. [url].

[74] William Michael Stevens. A Kinematic Model of a Self-Replicating Programmable Constructing Machine. *ECAL 2009 10th European Conference on Artificial Life*, 2009.

[75] William Michael Stevens. *Self-Replication, Construction and Computation*. PhD thesis, Open University, 2009. [url].

[76] Lubert Stryer. *Biochemistry*. W. H. Freeman and Company, New York, 3 edition, 1988.

[77] Synthetos. TinyG, 2016. [url].

[78] Skylar Tibbits. 4D Printing. *Architectural Design*, pages 116–121, 2014. [url].

[79] Tommaso Toffoli and Norman Margolus. Programmable Matter: Concepts and Realization. *Physica D: Nonlinear Phenomena*, 47:263–272, 1990. [url].

[80] Andrew J. Wade. Gemini, 2010. [url].

[81] Jonathan Ward. Additive Assembly of Digital Materials, 2010. [url].

[82] Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. Printed optics: 3D printing of embedded optical elements for interactive devices. *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, pages 589–598, 2012. [url].

[83] Wolfram. Finite Element Method, 2016. [url].

[84] Haixuan Yang, Andrei L Turinsky, Zhihua Li, Pierre C Havugimana, G Traver Hart, Peggy I Wang, Daniel R Boutz, Vincent Fong, Sadhna Phanse, Mohan Babu, Stephanie A Craig, Pingzhao Hu, Cuihong Wan, James Vlasblom, Vaqaar-un-nisa Dar, Alexandr Bezginov, Gregory W Clark, Gabriel C Wu, Shoshana J Wodak, Elisabeth R M Tillier, Alberto Paccanaro, Edward M Marcotte, and Andrew Emili. Resource A Census of Human Soluble Protein Complexes. 2010. [url].

[85] Xin-She Yang and Y Young. Cellular Automata, PDEs, and Pattern Formation. *Handbook of Bioinspired Algorithms and Applications*, page 12, 2010. [url].

[86] Qian Zhao, Weike Zou, Yingwu Luo, and Tao Xie. Shape memory polymer network with thermally distinct elasticity and plasticity. *Science Advances*, (January):1–8, 2016. [url].

[87] Victor Zykov, Efstathios Mytilinaios, Adams Bryant, and Hod Lipson. Self-reproducing machines. *Nature Communications*, 435(May):163–164, 2005. [url].