Parametric Modeling Implementation for Kinetic Systems Simulation:

Programmable Matter Matters

Nelson Montás [1], Pablo Baquero[2], Effimia Giannopoulou[3]

[1]*ESARQ-UIC, Dominican Republic,*[2,3]*ESARQ-UIC, Faberarium, Colombia, Greece*

[1]*https://narchitecture.wordpress.com/,* [2, 3]*www.faberarium.org,*

[1]*arq.montas@gmail.com,* [2]*paniba@faberarium.org* [3]*efeminno@faberarium.org*

Abstract. This paper focuses on programmable matter (PM), paying attention to anisotropic material behavior in shape memory alloys (SMA) and its applications within kinetic systems and architecture (KA), using parametric modeling code structures for subsequent architectural application development. Updates were done optimizing and expanding code functionality. Utilizing agile software development strategies, we derived new code that expands Grasshopper and Kangaroo's software functionality (SF) concerning digital design and previewing digital fabrication. We evaluated previous and optimized code using code correctness (CC) indicators as defined by Davis (2013) and hereby present its results so that further research and development projects concerning PM and molecular scale material design (MD) can be speculated upon and discussed. These could be used as basis for future research regarding both the theoretical and building application scopes of integrated material design, where simulation is used as a means of quasi-fabrication of kinetic architecture.

Keywords. Programmable matter; kinetic systems; digital simulation; flexinol; parametric modeling.

Introduction

In order to simulate the material's programmed behavior to properly predict critical function, actuation and physical properties, Grasshopper (GH) + Kangaroo was used to bridge the design-simulation tool's workflow in a single stream and to optimize and protocolize a smoother and more fluent decision making process. What this paper proposes is that, using designs built, tested and published by Chun Yi Wu on Youtube.com1 and laboratory results from *Dynalloy*'s *Flexinol* published literature to model macroscopic manifestation and mechanical phenomena, it is possible to approximate reality at the architectural scale. This way building an approach to design that is based and evolved not just within the 4th dimension (time), but actually using it to the designer's advantage thus making it possible and reliable to create complex self-assembly, self-organizing and intelligent kinetic systems based in PM simulation modeling instead of animation or static CAD modeling. We ask further questions about performance, code correctness and re-usability concerning previously developed PM based, digital simulations that replicate physical models and simulations done by Chun

[1] Original video available at: https://www.youtube.com/watch?v=1FoaIgKY01U&index=1&list=PLmnBQW0k0x1Gy19QXR9ZRmyPYJ-YzP0tT&t=387s (retrieved: 31/03/2017)

Yi Wu in which we used macroscopic mechanical data (of Flexinol) from Dynalloy [1], an industry SMM manufacturer.

Material Design

During the past ten (10) years, attempts to program not only software and machines, but matter itself, have defined the convergence of material science and the design disciplines, henceforth outlining a boundary-less, transdisciplinary continuum that suggests that, in the future, we might be able to design architectural structures emerging from molecular (10^{-6}) toward building-size scales (10^{2}), hence outlining what we consider to be material design (MD). MD is, within material science, the process of configuration, reconfiguration or modification, through programming or other methods like physical form finding, of any material to meet a certain set of criteria ranging from macroscopic manifestation to emergent behavior properties (Montás, 2015, 290). In this manner, it is close to what programmable matter is defined as "the science, engineering, and design of physical matter that has the ability to change form and/or function (shape, density, moduli, conductivity, color, etc.) in an intentional, programmable fashion." (Thomas, Tibbits, Banning, 2014, 3)

As Casey Reas (2006) has established, "Emergence refers to the generation of structures that are not directly defined or controlled. Instead of overtly determining the entire structure, I write simple programs that define the interactions between elements. Structure emerges from discrete movements of each element as it modifies itself in relation to its environment."

Following this principle, we wrote simple programs to define *Flexinol* behavior under different temperatures and to various geometric definitions. The emerging structure depends of the initial geometry and force arrangement, but the subroutines used stayed the same with, in principle, little changes made to adapt them to different situations and were therefore reusable.

Agile Software Development Applied to Parametric Modeling

In February 2001, a group of concerned software development professionals and representatives from different programming "schools of thought" who were all sympathetic to the need for an alternative to documentation driven, heavyweight software development processes held a gathering to see how to overcome the "rigidity" and low efficiency of what is commonly known in programming as the waterfall method, which was considered hindering by this group of software developers due to its overly rigid planning stages, bureaucratic structure and front-loading data modeling (Beck et al., 2001, 1).

Out of this reunion came out what is now known as the "Manifesto for Agile Software Development", convened and signed by Kent Beck and eighteen (18) other software engineers and developers. This manifesto predicated four (4) main tenets that countered what at the time was the hegemonic approach to the practice of software architecture and

engineering.From these core canons, twelve (12) working principles were drawn, out of which two (2) in particular directly address and define our team's developing method:

"...7) Working software is the primary measure of progress.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." (Beck et al., 2001)

These tenets played a pivotal role in the programming community's shift in the early 2000's towards the advent of better built, more flexible, changeable and interchangeable code, applications and so forth as Davis has noted (2013).
Following these guidelines and for this paper we built software tools that can help designers with little knowledge about SMA or programmable matter to approximate material behavior and design principles. Henceforth, technical data informed the employed methods and workflow selection.

Methodology to Design a Workflow Framework

Following Daniel Davis, we argue that using visual programming to bypass what is commonly known as the edit-compile-run loop (Davis, 2013, 159) is a relatively efficient way for visualizing algorithms working on demand and applying them to specific circumstances thus circumventing the aforementioned loop and hereby bypassing time consuming processes that hinder intuitive design decisions that mostly happen on the fly (Montás, 2016, 381), which is consistent with Schön's reflective practice and also shares a similar pattern of iterative prototyping as well present in agile development (Davis, 2013, 66).

Applying this code writing and analysis method has also shown to be instrumental in changing the stages of Raviv et al's approach to programmable matter described as a design-fabrication-simulation workflow in their paper "Active Printed Materials for Complex Self-Evolving Deformations" in Nature magazine (2015, 1), whose order we aim to alter into becoming a design-simulation-fabrication workflow (Montás, 2016, 378), which is shown here:

Combining Imperative + Declarative Methods for Parametric Modeling

Programming languages normally available to architects are usually classified in two (2) main paradigms: imperative and declarative programming (Davis, 2013, 62) (see figure 1), each with their own pros and cons, depending on a particular programmer's taste and skills in order to be selected for use.

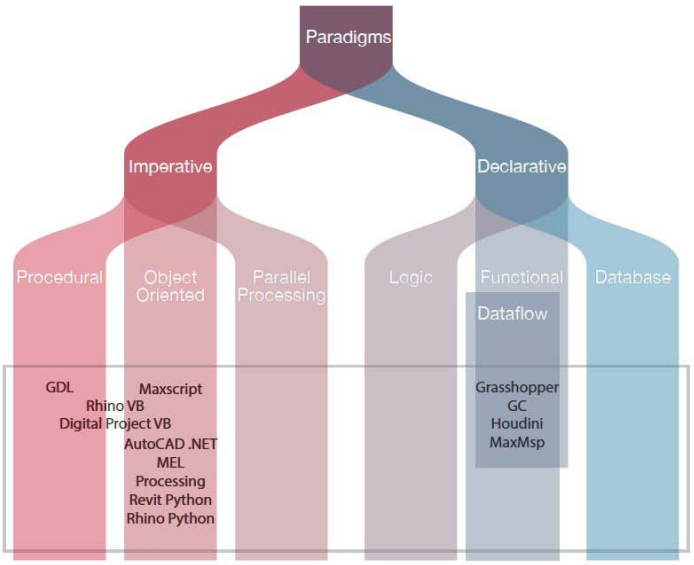


Figure 16: The programming languages architects use categorised by Appleby and VandeKopple's (1997, xiv) taxonomy of programming paradigms.

Figure 1 Davis, Daniel (2013, 62), Programming languages architects use categorized by Appleby and VandeKopple's (1997, xiv) taxonomy of programming languages.

Whereas declarative, visual code languages are excellent at bypassing the edit-compile-run loop which helps when you are consumed in intuitive design tasks and modeling, however, in other scenarios and situations, imperative programming has several advantages when compared to the declarative paradigm. One of these is that it supports structured code more elegantly than declarative, thus being more useful for automated tasks and repetitive event handling, hence event driven programming. It can be implemented to realize accurate yet time consuming or “boring” tasks, letting the designer worry about more metaphysical aspects (Montás, 2016).
Combining object oriented (OOP) and visual programming paradigms, exploiting Grasshopper’s ability to embed python code in it’s visual code, we proceeded to write a Python script in the Grasshopper interface that lets you write automated object oriented programming (OOP) chunks and integrate them into a GH definition as components that have inputs and outputs. Imperative programming is used, in this context, to automate the material's macroscopic, mechanical behavior like rigidity, rest length, stiffness, stress and, in this particular case, Flexinol’s heating and cooling resultant pull-forces, proportional to its diameter in the C.G.S. measurement system (grams). These characteristics inherent in the material, although they can be modeled using data-flow (declarative paradigm), hold better and are more reusable if implemented in a component that reproduces the material’s laboratory tested mechanical data.

Expanding GH and Kangaroo’s Software Functionality: Pull Force Calculation Subroutine

As a part of these experimental cases, we proceeded to reverse engineer, formalize and program a scalar formula that integrates proportions between the material’s diameter and pull force, in accordance with the Dynalloy Technical Data Table [1], and hereby outputs

a scalar resultant ready to be given vector format. This was realized by writing a subroutine in Grasshopper's Python editor and using the Unit Vector components to turn the values into proper vector form, therefore expanding GH's SF by producing a priorly non-existent component called *SMM NiTi Flexinol Mecca*. The general formula was implemented using the following simplified form:

$\big(2(ax2)\big) * c = F(x)$ (1) {Equation}

Where "a" is the nominal term, "x" is equal to the wire's diameter, "c" is a constant equal to 100 and "F(x)" equals the resultant pull force. Setting the "a" term to 71 gives the pull force for heating actuation and setting it to 29 gives the cooling actuation pull force, is shown below for heating:

$(2(71x2)) * 100 = F(x)$ (1) {Equation}

And cooling:

$(2(29x2)) * 100 = F(x)$ (1) {Equation}

The simplified equation can also be expressed in algebraic form as follows:

$(ax2 + ax2) * c = F(x)$ (2) {Equation}

For heating:

$(71x2 + 71x2) * 100 = F(x)$ (2) {Equation}

And cooling:

$(29x2 + 29x2) * 100 = F(x)$ (2) {Equation}

This function allowed us to make an approximate, but correct simulation to an error margin within 0.25% for small diameters (0.028mm) and 3.60% for bigger ones (0.51mm) when compared to mechanical macroscopic data observed in laboratory experiments, published by Dynalloy. The results from this simple operation were then used to tell actives vectors points in any such given geometry how to behave realistically in reaction to the material's stiffness factor, which in the case of Ni-Ti alloys and Flexinol in particular, is of about 172 MPa while heating (deploying or shape A to B) and 70 MPa while cooling (retracting or shape B to A). A temperature conditional statement has been also added to control one way (shape A to B) and two way2 (shape A to B, then B to A) shape memory effect behavior that acts as a boolean to toggle between states at will, the temperature threshold were set to 90c° which is a Flexinol standard commercial wire's martensitic phase temperature (the other one is 70c°).

---

[2] *The one way and two way memory effect are characteristics of certain SMA by which to, according to their chemical configuration or "recipe", undergo either a one way (going from shape A to B) or two way shape memory martensitic phase transitions (shape change), which equals going from shape A to B and back to A.*

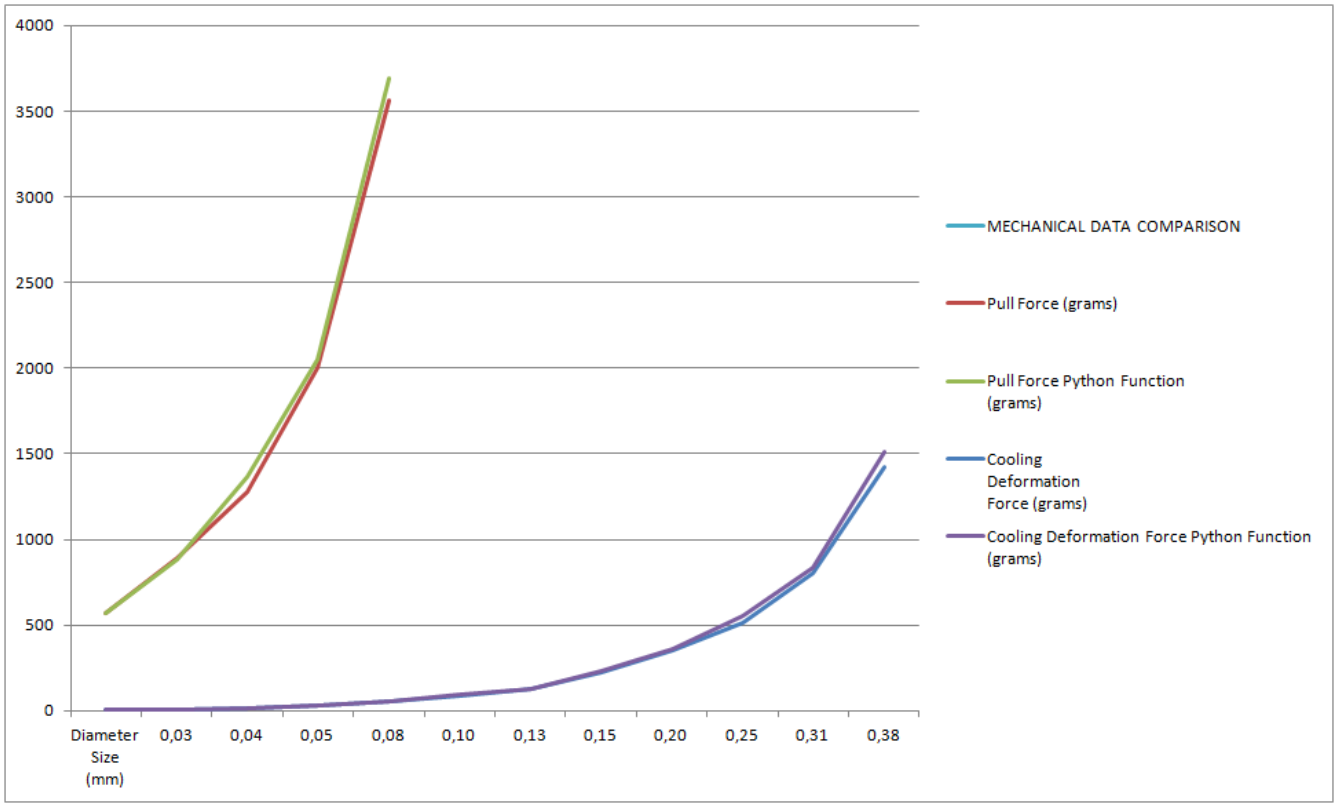


Figure 2.Montás, Baquero, Giannopoulou. Mechanical data comparison between Dynalloy's laboratory tests and our team's Python implemented function. Diameters are shown in millimeters and forces in grams.

Diameter Size (mm)	Pull Force (grams)	Pull Force Python Function (grams)	Cooling Deformation Force (grams)	Cooling Deformation Force Python Function (grams)
0.03	8,90	8,88	3,60	3.63
0.04	20,00	20,50	8,00	8,38
0.05	36,00	35,50	14,00	14,50
0.08	80,00	82,02	32,00	33,50
0.10	143,00	142,00	57,00	58,00
0.13	223,00	239,98	89,00	98,02
0.15	321,00	319.5	128,00	130,50
0.20	570,00	568,00	228,00	232,00
0.25	891,00	887,50	356,00	362,50
0.31	1280,00	1364,62	512,00	557,38
0.38	2004,00	2050,48	802,00	837,52
0.51	3560,00	3693,42	1424,00	1508,58

Table 1.Montás, Baquero, Giannopoulou. Mechanical data comparison between Dynalloy's laboratory tests and our team's Python implemented function.

Conceptual Model of the System

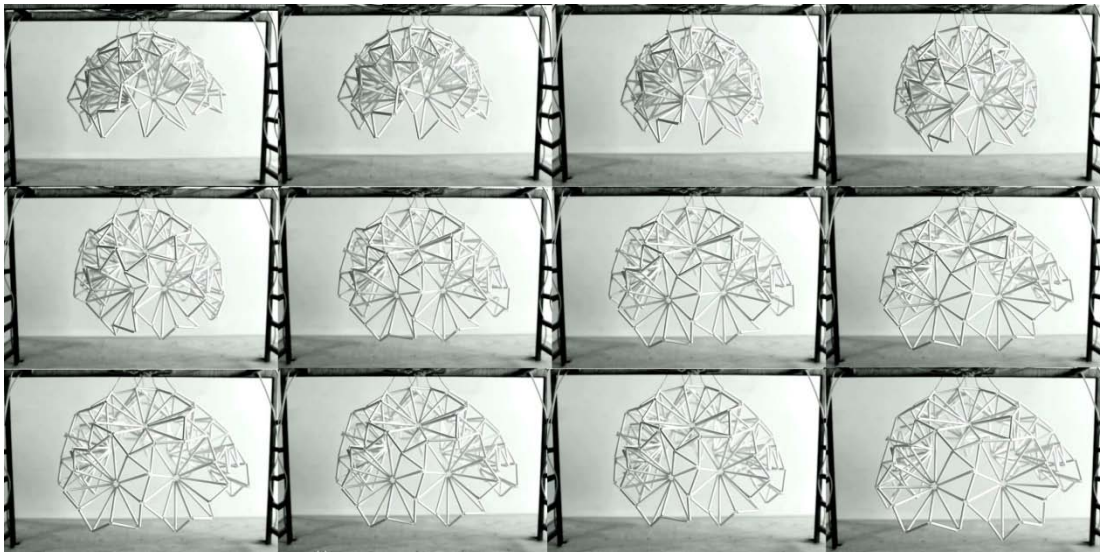
To define a simplified representation of a system, approximations are introduced to reduce complexity, computational requirements and solution time. The model's time span is dynamic, so behaviour changes over time. The changes over time are represented as both discrete events and continuous ones.
The below case studies are experiments in order to obtain a better understanding of the system. The Kangaroo physics engine is used to visualize the emergence of possible geometrical configurations of SMA under specific temperature changes to arrive at complex self-assembly, self-organizing and intelligent kinetic systems based in PM simulation modeling.

Case Studies

In order to find a way to prove our OOP subroutine in realistic approximations to actual physical models using SMA, we selected two (2) designs built, tested and published by Chun Yi Wu on Youtube.com. These are the following: “hexagonal composition” and “membrane composition”.

Hexagonal Composition

In order to do the simulation of the physical behaviour of the chosen geometrical configuration, we used Rhinoceros to set up the initial conditions of twelve (12) flat hexagons with their radial lines and put them as input lines and points for the GH definition. Using the Kangaroo physics engine, we applied the pull force calculation subroutine to the actuated lines while the rest kept their original length and rigidity unchanged, producing a dynamic tensegrity system. Changing the temperature we recorded the movement. At less than 90° the hexagons kept closed, at more than 90° they opened as seen in the stop motion sequence below. (See figure 3).



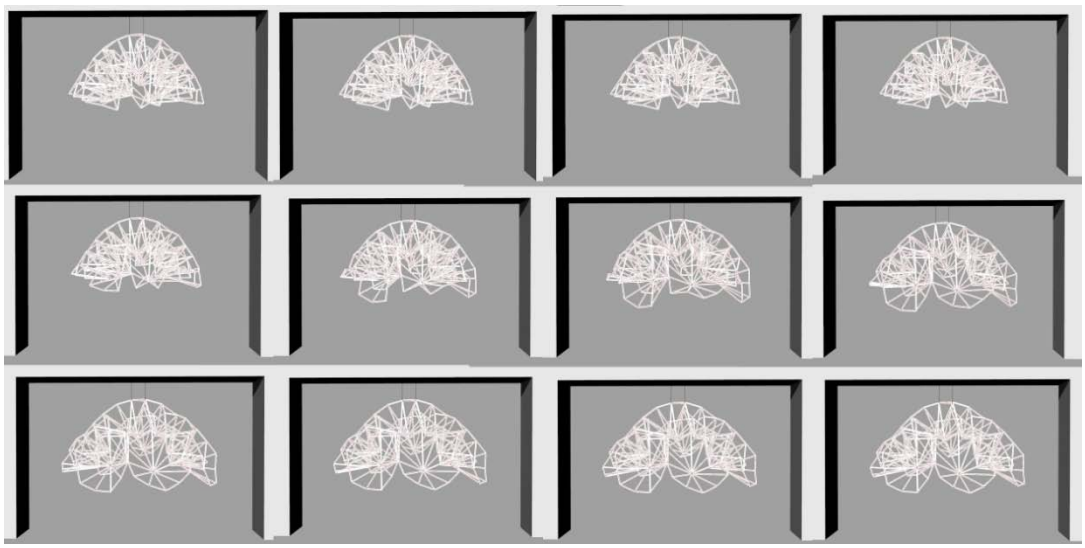
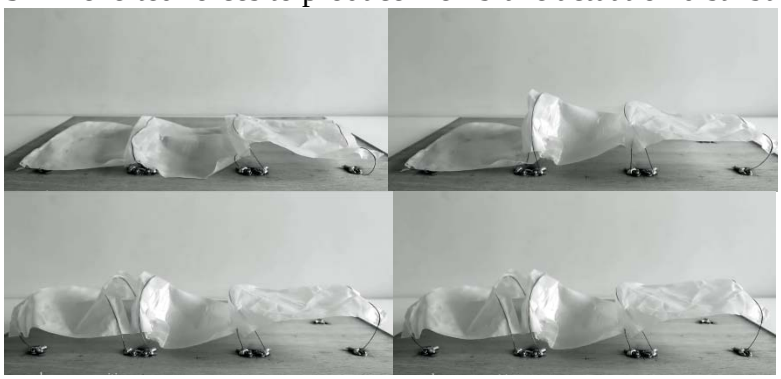


Figure 3.Montás, Baquero, Giannopoulou. Hexagonal composition comparing physical model (above) and simulation actuation (below).

2. Membrane Composition

To simulate Wu's membrane composition, after trying a range of pull components in GH, we settled for a vector field matrix approach, an idea borrowed from material science used to describe how materials bend and deform under forces acting upon them. This idea tells us that a field is like a piece of cloth covering an area of space with a value assigned at each point in its surface area (in this case, the membrane). When that value takes the form of a vector, it becomes a vector field and, if we interlock them tracing a lattice all over its surface area (in this case a two -2- dimensional one), this field becomes controllable as a whole with a matrix-like vector definition: a combination of values produces a resultant form. We implemented the python pull force subroutine to apply the SMA exerted forces to produce membrane actuation distributed into X, Y, Z unit vectors.



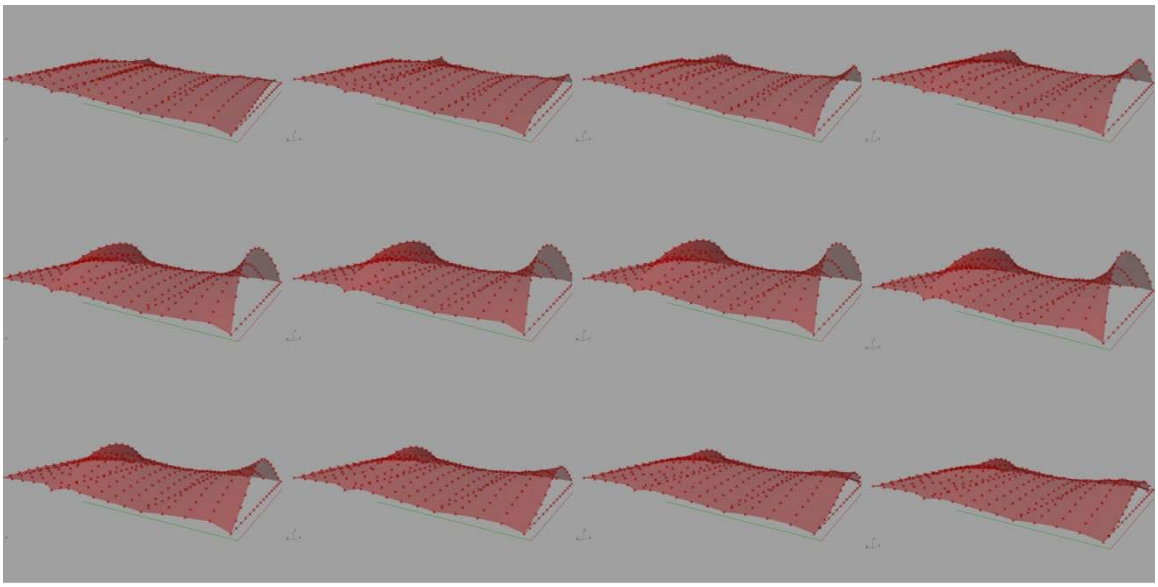


Figure 4. Montás, Baquero, Giannopoulou. Membrane composition comparing physical model and simulation actuation.

Conclusion

The foreshown experiments make the case for possible, contingent and creative SMA material applications that can be used to ameliorate architectural spaces and even entertain the notion of scaling them up to the building scale by conceiving material matrices that combine SMA with other more structurally implementable materials. We have expanded SF in GH to produce a tool that architects can use in their design strategies to address KA and PM applications and projects. Next is a set of questions that aim to evaluate parametric modeling and its effectiveness (Davis, 2013) while also serving as a benchmark for qualitatively measuring code correctness (CC):

Functionality: Are all the modeling (and coding) tasks able to be performed by every programming method? Yes, they can be done by both declarative and imperative programming editors and languages, yet, the tasks are better handled by imperative if repetitive and exponential and by declarative if more intuitive and ad hoc.

Correctness: Do programs do what is expected? Mostly, Yes, with some delays. In the case of the membrane composition, some vectors showed an unexplained malfunction. Our a priori assumption is that the aforementioned lacks polishing. On the other hand, the OOP code worked as expected and results approximated the physical model.

Ease of use: Are the modeling interfaces easy to use? Not established as a sure fact, lacking group experiments to confirm, but it appears to be the case. Generally they are easy to engage, but it could be easy to get disorganized with many connections. It is good practice to keep notes to remember functionalities.

Construction time: How long did the respective models take to build? Not measured.

Lines of Code: How verbose were the various programming methods? The hexagonal composition was able to be built using relatively few components and code lines, but in the membrane composition case, the definition was substantially more verbose due to the vector field approach selected.

Latency: How quickly did code changes become geometry? The interactivity exhibited is considered sufficient for these models specifically. More complex models should see their interactivity decrease proportionally.

These conclusions clearly indicate that GH and Kangaroo, when combined with Python scripting language, which embeds imperative within declarative programming, make up a reliable method to design PM based kinetic systems and applications which therefore represents a step forward in this direction. While group experiments can enlarge our understanding of how to model these complex kinetic systems, they remain a future endeavor in the path of revealing if parametric modeling is able to be broadly disseminated to address PM in the wider discipline of architecture without resorting to engineering specialists, at least in the design stages of a project.

What Lies Ahead

We need to define a parametric model method that effectively reconciles molecular and building scales, hereby achieving multi-scalar material modeling. We are confident that the combination of data-flow and OOP (declarative and imperative) can bring about parametric models that can do this. Every paper we have known about examines either microscopic or macroscopic behavior models. For example, Otsuka and Wayman (1998) explain SMA's microscopic structure mathematically, yet macroscopic behavior is left to explanations with mechanical equations and matrices but no reconciliation with its subatomic theory. We are looking for ways to try to reconcile both molecular (10 -6) and building-size scales (10 2) yet we are aware that this is a current and difficult conundrum in physics and hope to contribute to or learn from the situation in that discipline and advance our work on programmable matter and material design altogether.

References:

Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, “Manifesto for Agile Software Development”, pp. 1-10, February, 2001, (https://www.google.fr/search?q=Manifesto+for+Agile+Software+Development&oq=Manifesto+for+Agile+Software+Development&aqs=chrome..69i57.418j0j4&sourceid=chrome&ie=UTF-8) retrieved: 31/03/2017.

Davis, Daniel, Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture, School of Architecture and Design College of Design and Social context, RMIT University, Melbourne, 2013.

Montás, Nelson, Performance Software Approaches for Kinetic Architecture: Programmable Matter Based

Simulations, ESARQ, Universitat Internacional de Catalunya, Barcelona, 2015.

Otsuka, Kazuhiro, Marvin Clarence Wayman, Shape Memory Materials, Cambrige University Press, 1998.

Raviv, Dan, Wei Zhao, Carrie McKnelly, Athina Papadopoulou, Achuta Kadambi, Boxin Shi, Shai Hirsch, Daniel Dikovsky, Michael Zyracki, Carlos Olguin, Ramesh Raskar & Skylar Tibbits, “Active Printed Materials for Complex Self-Evolving Deformations”, Scientific Reports 4, Nature, Article number: 7422, pp. 1-8, December 2014.(http://www.nature.com/articles/srep07422) retrieved:29/11/2016.

Reas, Casey, “Process/Drawing”, Programming Cultures: Art and Architecture in the Age of Software, Architectural Design, Wiley Academy Press, United Kingdom, 2006, pp. 27

Campbell, Thomas, Skylar Tibbits, Banning Garrett, “The Next Wave: 4D Printing and Programming the Material World”, Atlantic Council, Washington, DC, USA, May 2014, pp. 3

[1]http://www.dynalloy.com/tech_data_wire.php