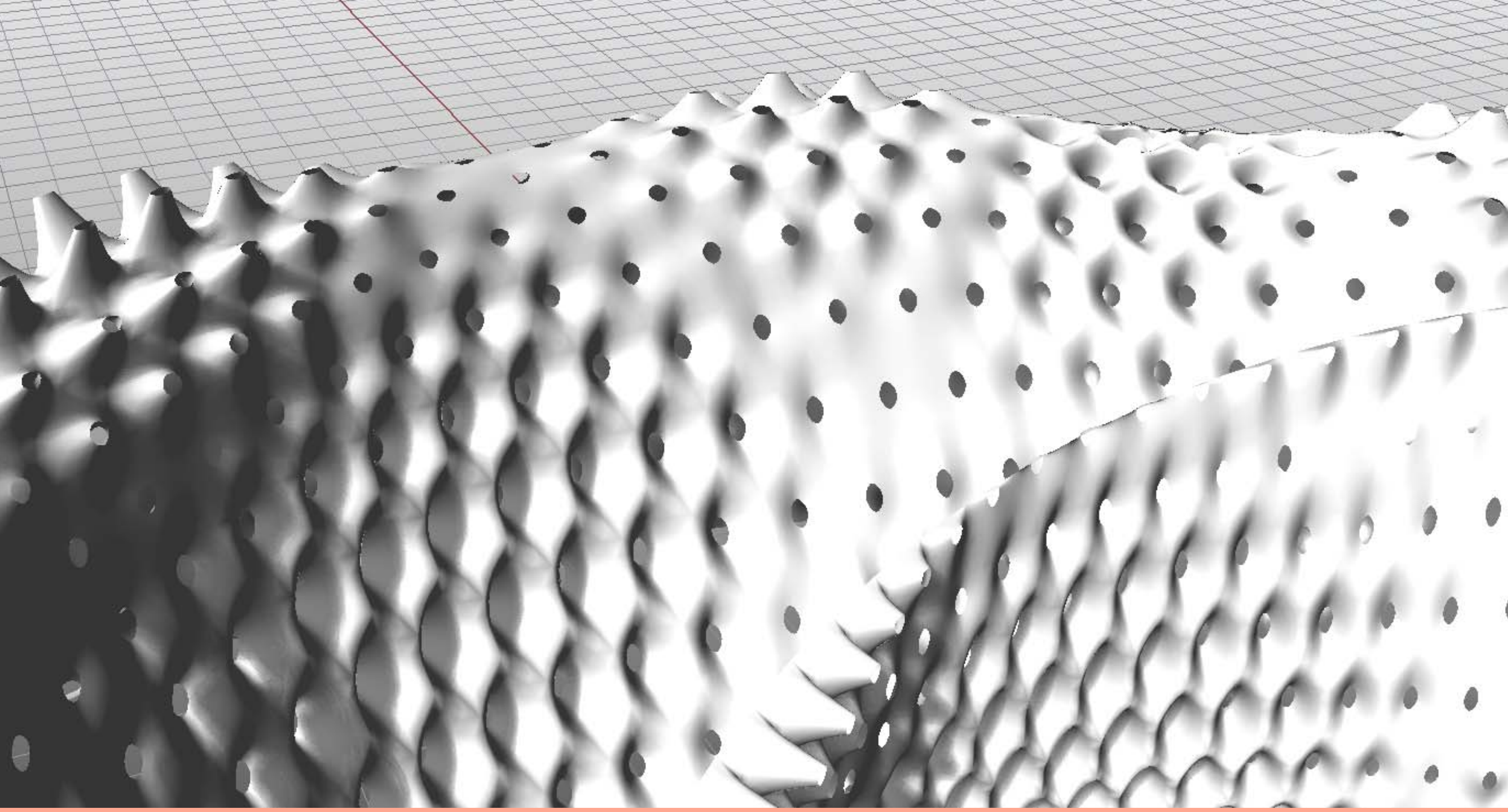

Design Modeling Terminology

By David Stasiuk

Abstract

Digital design modeling in architecture has matured in both practice and theory such that its discussion in almost any form comes loaded with competing claims and assumptions related to its associated terminologies. This territory for competing claims extends to definitions of different model typologies. In this context, this paper engages discourse surrounding different types of procedural modeling approaches, focusing specifically on the distinctions and inter-relations associated with definitions for *parametric*, *computational* and *generative* design models.

Read More →

proving ground

Procedural Modeling

A procedural modeling process is one that uses an explicit instruction set to produce a model outcome. Model outcomes can be highly varied in regards to the larger context of architectural modeling and especially when considered through the lens of procedural modeling techniques. For the purposes of simplicity in this section about procedural processes, architectural modeling outcomes will refer to design geometries as expressed in a descriptive, primarily digital medium. Design geometry in this context nonetheless remains very general and can refer to an object as simple and abstract as a point in space or as explicit and complex as a collection of highly differentiated, materially-specific interacting assembly of parts. Regardless of its ultimate expression, the outcome of a procedural model can be recognized for its *second-order* formational nature.

In contrast, *first-order* modeling techniques produce outcomes through a sequence of commands that are deployed "directly" by the modeler. In a physical design space this includes such actions as drawing with a pencil or pen, or directly manipulating materials such as clay or cardboard. In the digital design space, first-order drawing techniques include the use of direct CAD commands for creating such objects as polylines or splines by means of mouse-clicks. In contemporary practice, even in the computer and for the generation of three-dimensional design objects, first-order modeling techniques then are usually referred as being "hand-drawn".

Second-order modeling processes instead describe a removal from the direct modeling techniques described above. In place of direct manipulation of design geometries, the procedural modeler rather directly manipulates an instruction set – or algorithm – that in turn produces design geometries.

Procedural techniques then by definition demand an abstract mode of thinking in the realization of design goals: rather than directly engaging with an outcome, the designer develops a set of rules and processes that produce an outcome by transforming data from an idea space into the design geometry. This indirect engagement with modeling output has been identified as both a source of great potential in procedural modeling and as a significant pedagogical challenge (Aish 2011). Procedural modeling allows for the designer to dynamically engage with complexity in more efficient ways than those afforded through hand modeling.

This distinction between first-order (hand) and second-order (procedural) modeling techniques provides a point of departure for a deeper examination of what distinguishes different types of procedural models, how these different model types function as engines for producing outcomes, and why understanding their diversity for the purposes of classification merits careful consideration. The following sections will describe three consequential *types* of procedural architectural modeling: parametric, computational, and generative.

These types are here described as consequential because, in this listed order, each subsequent modeling type necessarily embodies the requisite characteristics of those that precede it. In the process of typing these models, we will then introduce the key features that delineate this hierarchy of nested and inter-related functional and operational components. Finally, through this understanding of how different types of procedural modeling are activated, I will begin to discuss how these distinctions between model types– both as conceptual and practical frameworks – partner with the designer for the execution of designs that are better able to produce but also synthesize and manage complexity.

Parametric Modeling

"Parametric Models" are the first type of procedural modeling discussed here, as they operate as the basis upon which each subsequent type is founded. "Parametric" is the term that is most commonly used to describe those techniques that define the largely digital form of design practice that privileges and deploys procedural modeling. A great deal has been written about parametric modeling, and because different texts produce different interpretations, the usage of "parametric" has become fraught with potential for epistemological loading. A broad range of definitions abound, each engaged in its own focus on either generalization or specificity. For the purposes of this research, I will largely defer to the excellent historical categorization, analysis and interpretation presented by Daniel Davis in the section of his PhD thesis titled "What is Parametric Modeling?" In this section he expands upon the term from its origins into the breadth of its various practical and theoretical applications specifically in the field of architecture.

In setting up and addressing a sequence of its common usages, Davis effectively counters any obfuscation embedded there in either the rhetorical device or theoretical bombast exhibited by practitioners by returning to a clear definition based specifically on the origins of the term in mathematics. He arrives at the direct and productive definition of a parametric model as "a set of equations that express a geometric model as explicit functions of a number of parameters". (Davis 2013)

This research similarly defines the term, with one small change regarding the model expression, or its output. Davis limits his notion of the parametric model output as being purely geometric in nature, which is fair when considering that most architectural models are recognizable for their articulation of form, organization and assembly as three-dimensional instruments. However, it is worthwhile to more explicitly outline that model outputs may also include such essential information as descriptive labels, material specification or other purely numerical data elements: it is common for a parametric model to generate not only geometry but also a host of associated information. This heterogeneity of modeling outputs becomes important when considering the relationships that models will frequently have with one another both within and across design projects. Therefore, we will consider that parametric models have the ability to deliver not just geometries, but systematic collections of information relevant to the delivery of an architectural solution. So here, a parametric model will be concretely defined as "a set of equations that express information regarding the deployment of an architectural information system, as explicit functions of a number of parameters." A parametric model can then be seen as being composed of parameters, translational functions, and their expressed information. These three primary components will be restated here as being *parameters*, *algorithms* and *outcomes*.

In a mathematical equation, the parameter supplies an initial informational construct to the translational function such that the outcome is dependent upon it, but it itself is not dependent upon the outcome. Thus, input parameters are, by definition, independent variables, and outcome parameters are by definition dependent variables. If in a mathematical equation the function is the means by which independent variables are transformed into dependent variables, in an architectural parametric model, the algorithm is the means by which parameters are transformed into model outcomes. The key to the concerted operation of these components in a parametric model is the explicit nature of this relationship: through the variation of the parameter values, the algorithm parses different outcomes. In a simple sense, once the designer has determined this explicit setup, his or her role shifts from model design to model operation.

Consider a simple parametric model for the geometric representation of elements in a façade. The façade elements arrayed in a digital environment will reflect the model outcome. The parameters for the model will be defined by elevation height, elevation width, number of vertical subdivisions and number of horizontal subdivisions. With these determined input parameters and intended model outcome, the algorithm then must be developed as the mechanism for enacting the transformation between these two information states. Here the algorithm might first (1) create a two-dimensional domain from the input parameters associated with the building width and height. It may then (2) divide this domain into a collection of smaller sub-domains, according to the respective numbers of horizontal and vertical subdivisions. It may then (3) convert these locally discrete sub-domains into an array of three-dimensional geometric representations of the façade elements.

In the case of this example, the algorithm is utterly simplistic and, would certainly only ever be used as a demonstration of the most rudimentary model setup. It is rare to find a parametric model comprised of only four parameters, and even rarer to find fully a developed parametric model that might rely on an algorithm that effectively deploys only three related translational operations. In practice, models may rely upon a great variety of input parameters, in both number and type. And models may be managed through algorithms that will execute hundreds of transformations during their deployment. Nonetheless, the simple example outlined above fully meets the criteria for operating as a parametric model and effectively demonstrates its chief identifying characteristics. In this most basic type, a parametric model operates as a one-way system, with parameters feeding an algorithm that directly enacts outcomes through explicit transformation.

One of the chief claims made about parametric modeling is that it enables the designer to operate in a dynamic environment that is particularly fruitful for the purposes of exploration and rapid versioning. It is true that the variation of input parameters – either systematic or otherwise – will allow for the designer to enact multiple expressions of the same system. Depending upon the model setup in the definition of both parameter space and algorithmic process, efforts to thus convert multiple parameter settings into taxonomies of outcomes can successfully produce highly varied results. Nonetheless, each instance of such a parametric model as outlined above remains singular as an expression of a collection of static parameters.

The problem space that emerges from such parametric models revolves around questions of creating incremental complexity and in particular in developing more effective means of producing information from parametric modeling systems. Direct transformation is surely useful for a host of applications, but the idea of effecting new knowledge about the design system from the modeling space engenders an opportunity to activate the next nested type in the hierarchy of procedural models.

Computational Modeling

Today, the term "computation" generally refers either the operation of or the operations performed by any of the multitude of digital computers with which we interact on a regular basis. When considering procedural modeling, this usage is problematic for at least two reasons: First, the explicit association of computation with digital computers designates a limiting specificity that belies the more general and pliable definition of computation as “a system of reckoning,” as it has been defined by the physicist and philosopher Heinz von Foerster. By his definition he asserts that “computing…literally means to reflect, to contemplate (putare) things in concert (com-), without any explicit reference to numerical quantities.” Computation then reflects “any operation, not necessarily numerical, that transforms, modifies, re-arranges, or orders observed physical [or symbolic] entities…” (von Foerster 2003) Here it is useful to note the specific detachment of computational dependency from the digital computer.

Naturally, computation can readily be performed on digital computers; but it is crucial – and perhaps particularly so in the field of architectural design – to not only be aware of alternative means to execute computation but also to understand the specifics of and potentialities embedded in their differentiation. Secondly, while the common definition of computation necessarily being related to digital computers is overly reductive in the above regard, it conversely is overly general in its failure to clarify the critical distinction between *computation* and *computerization*. This is no small matter to theorists and practitioners of architectural design, and the point has been made repeatedly, perhaps first by Nicholas Negroponte in early 1970’s (Negroponte 1973), and more recently by Kostas Terzidis. (Terzidis 2006) A more comprehensive investigation of the distinction between the two terms has been performed by Sean Ahlquist and Achim Menges, who assert that “the distinction between computation and computerization…can be broken down as methods which either deduce results from values or sets of values, or simply compile or associate given values or sets of values. One increases the amount and specificity of information, while the other only contains as much information as is initially supplied.” (Ahlquist and Menges 2011) In this way, computational design must be seen as being *information-producing* in makeup, moving beyond translational representation through a process of transformation. So computation is the expression of a system of reckoning that informs through operations of transformation, modification, or the ordering of observed physical or symbolic entities.

Ahlquist and Menges further narrow their definition: “one critical aspect of working in a computational manner is the processing of information algorithmically [which means developing design systems that operate according to] a set of procedures consisting of a finite number of rules, which define a succession of operations for the solution of a given problem.” (Ahlquist and Menges 2011) The computer scientist Jeannette Wing defines “computational thinking” as “the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.” (Wing 2006) The system of reckoning, then, is encoded through the information-processing agency of algorithms.

An explicitly computational modeling approach then fully embeds the characteristics of the parametric model in its makeup, performing explicit transformative operations on parameters through the use of algorithms in pursuit of a design outcome. Critically, though, computational design models are defined by incremental criteria and performances in both a model’s organization and in its expected output: they must compute *new information about an architectural system*. So then if the nature of the algorithm in a procedural model is such that its conversion of an input to an output is wholly translational and in no way transformative, its procedural functioning as a parametric model may be established, but it will reflect a process of computerization rather than computation. The parametric model described in the previous section fits this description of such a design model: both number and size, and orientation of the elements are wholly predetermined, but they are procedurally drawn through the model. Because it is not generating new information – operating as a system of reckoning – about an architectural outcome and is instead operating as a translational mechanism for the purpose of representation, this parametric model is not computational in nature. Thus, computational models can always be characterized as being parametric in nature without the converse necessarily being true.

In fairness, it is more common in practice for a well-developed parametric design modeling strategy to embody the functional criteria for computation. That said, parametric models are deployed to great effect for purposes other than strict design modeling, as evidenced through the representational capacities intrinsic to BIM systems (Aish 2011) In this capacity less as form generators or response systems for adaptation, parametric models demonstrate great capability for information collection and collation, and as power-drawing instruments.

The utility in making these distinctions for both functions and capabilities attributable to different model types lies in both clarifying the purpose of and optimizing strategies for the making of models deployed during different phases of project design and realization. The chief difference then between a purely parametric model and a computational model regards its parameterization: the former sees their parameters deployed as single static entities during model execution, whereas the latter have the capacity for their parameters to be actualized as dynamic and responsive entities during model execution, not only informing the algorithm but in turn being recalibrated themselves.

Consider the following example, of a computational design model for designing a series of vaults defined by a network of catenary geometries. This model may transform the geometric qualities associated with an established topological setup by simulating catenary action through a particle-based dynamic relaxation engine. Such a model may be composed of geometric, topological, and numerical parameters that describe the initial boundaries, networks of relationships between the particles, and multiple values that activate the forces in the simulation. The algorithm is comprised of the simulation system and its parameter-driven transformation of the particles either over a set number of time intervals or potentially during a real-time visualization for the designer, and the translation of these particles into useful output geometries. The algorithm here is necessarily time-based. The state of each particle at any given time step is recalculated in relation to those constituents with which is shares relationships and as part of the algorithm is used to re-parameterize the model with each subsequent time step. This dynamic and stochastic re-parameterization process is particularly direct in describing a computational approach, as the algorithm is used to effectively read from and write back to its parameter space.

Generative Modeling

Philip Galanter is an artist and researcher focused on "generative art," whose key characteristic is its "use of an autonomous system" in its production. His explicit definition is that "generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine or other procedural invention, which is set into motion with *some degree of autonomy* contributing to or resulting in a completed work of art." (Galanter 2003) By this definition, the boundary conditions for what can be considered a generative design process become fairly open.

There is little specificity here beyond the demand for a procedural application that introduces some note of algorithmic autonomy. Other definitions for the term "generative" as it pertains to design modeling embody more specific criteria. For example, generative systems are defined according to their ability to deploy algorithmic transformations and "re-use embedded sub-systems" specifically incremental morphogenesis (Hornby, Lipson and Pollack 2001). And generative processes have also been considered as "an active space of progressive formation and mutation" in the digital design process (Attar, et al 2009). These ideas of *progressive formation* and the re-use of embedded sub-systems drive the definition presented here: that a "generative" model will be understood as one that not only meets all of the criteria for being a computational model, but which during a single execution instance additionally incorporates a capacity for *unfixed topological relationships* between model elements, and/or is actualized through the step-wise accretion of new model elements that are morphogenically recursive, with incremental stages of formation dependent upon preceding steps enacted during the model execution.

An example of a fully generative system is given in the paper cited above by Gregory Hornby, Hod Lipson and Jordan B. Pollack. In it, they describe the modeling process of a locomotive robot whose design is evolved through the application of a stochastic l-system algorithm. In essence, their model development relies on the establishment of a framework of possible moves negotiated through an l-system for the generation of a geometric representation of a robot, along with a parallel definition for how certain components of its geometry may be afforded movement. This representation and definition for movement can then be assessed through a simulation model that tests for its locomotive performance, in terms of speed and efficiency of movement. Evolutionary algorithms are deployed on the rule sets that "grow" the digital representations of the robot. The process operates in sequence. Entities are first randomly generated through the arbitrary ordering of rules to be parsed through the l-system. These randomly generated entities are then tested within the simulation, and through the evolutionary algorithm the representations from each generation that demonstrate the greatest capacity for locomotion are "mated" with one each other by replacing bits of the l-system algorithm from one with bits from the other. Over hundreds of generations, unanticipated forms are fully generated, and the instruction set for the l-system is continuously redeveloped, intrinsic to the model itself.

Of the three primary terms that constitute the model typing hierarchy, "generative" is the one that seems to be most loosely defined in general usage. For this reason it has afforded practitioners a great deal of flexibility in its application, and also has become the most loaded with epistemological variation. The synthesis for each of the terms applied has been done as much through a process of elimination as possible. For example, other terms that may be considered in place of "generative" here include "emergent", "recursive", and "morphogenetic", but each of these is, respectively, either fraught with additional meanings, denotes more specific performance capacities than are necessary, or is only partially descriptive of the modeling characteristics it must embody. So even though "generative" appears to be the least specific in its broad application, this same critique can be applied to each of these terms, that usage varies from the highly general to the highly specific, and as one moves toward more specific definitions, greater potentials for disagreement emerge.

A Typological Spectrum

Thus far, this text has been careful to designate the differences that are embedded in those distinct procedural design model types as they embody parametric, computational or generative characteristics. For each of these types, the example presented has been selected for the clarity of its embodiment of the characteristics as defined above. The façade is quite simply parametric in the direct representational nature of the relationship between its inputs and outputs; the catenary vaults activate computation through the agency of a simulation process that operates in a clearly-contained but constantly self-informing feedback loop; the generative robotic model based on the evolution of a stochastic l-system exhibits characteristics that are not only fully morphogenetic in their recursive geometric formation, but sees its parameter space demonstrate adaptive behaviors through the application of evolutionary algorithms in the rewriting of the growth algorithm.

However, with these idealized examples of each in place and an understanding of the hierarchy of difference that describes their consequential and nested relationships, it finally becomes critical to recognize them as operating along a spectrum, and with procedural models in practice more often demonstrating tendencies toward each of these types rather than fully embodying the characteristics of a single one at any given moment. At some point, then, a model makes the jump from being purely parametric to exhibiting computational functionality. However, locating the particular moment or reason for this shift in the design model's epistemological makeup may present a problem. Indeed, individual models so analyzed may themselves be further discretized into subdivisions that identify variable degrees of complex characteristics as denoted through this typological hierarchy.

Finally, it should be noted that while the specificity of definitions developed in this article for different model types is not necessarily meant as a direct criticism of the variety of well-considered usages that have proliferated among academics and practitioners, it does aim to cut through their application as "buzzwords" and is in some ways a reaction to their proliferation as such. So although there are necessarily moments of disagreement between the categorical descriptions applied here and their usage elsewhere, this text is meant to identify, simplify and synthesize for each as many similarities as possible that it finds embedded in this observed variety. In so doing, it develops a graded framework for thinking about procedural design modeling systems. The aim is to identify difference and facilitate a categorical understanding of their diverse functional capacities.

References

Ahlquist, Sean, and Achim Menges. 2011. Computational Design Thinking. John Wiley & Sons, Ltd.

Aish, Robert. 2011. "DesignScript: Origins, Explanation, Illustration." In Computational Design Modeling, 1–8. Springer.

Attar, Ramtin, and Robert Aish. 2009. "Physics-Based Generative Design." CAAD Futures …. http://cumincad.architexturez.net/system/files/pdf/cf2009_231.content.pdf.

Davis, Daniel. 2013. "Modelled on Software Engineering : Flexible Parametric Models in the Practice of Architecture."

Foerster, Heinz von. 2003. "On Constructing a Reality." In Understanding Understanding, 78:211–27. Springer. doi:10.1006/geno.2001.6652.

Galanter, Philip. 2003. "What Is Generative Art ? Complexity Theory as a Context for Art Theory." In GA2003 - 6th Generative Art Conference.

Hornby, GS, H Lipson, and JB Pollack. 2001. "Evolution of Generative Design Systems for Modular Physical Robots." Robotics and Automation, 5–10. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=933266.

Negroponte, Nicholas. 1973. The Architecture Machine: Toward a More Human Environment. Computer-Aided Design.

Terzidis, Kostas. 2006. Algorithmic Architecture. Oxford: Architectural Press.

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.