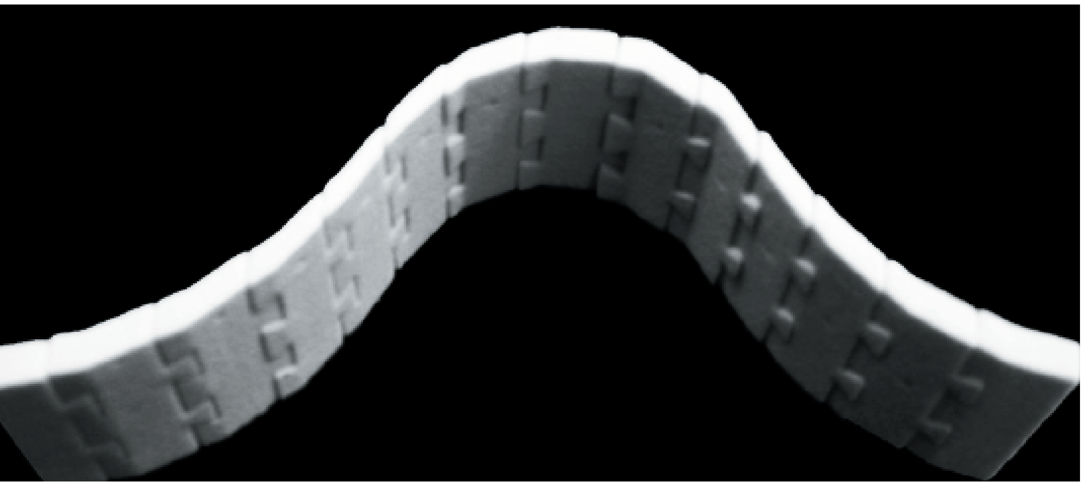
Rulebuilding: a generative approach to modeling architecture using 3D printers

Yanni Loukissas Massachusetts Institute of Technology
Lawrence Sass Massachusetts Institute of Technology



Abstract

3-D printers alter the speed, cost, complexity, and consistency with which physical architecture models can be crafted. If architects are to harness the unique abilities of this modeling process, it is necessary to find a complementary means of conceptualizing designs and generating the geometric data necessary for 3-D printing. This paper introduces a novel combination of 3-D printing and scripting through three examples of architectural surface models. In these examples, VBScript is used to write generative scripts for execution within the Rhinoceros modeling environment. The scripts produce digital geometric models which, in turn, are exported to a Z-Corp 3-D printer.

The merits of this methodology are demonstrated, in one example, through models of an architectural surface composed of light-modulating conical components. The design intent in this example is a grid of responsive components which ride on a complex curved surface and steer toward a light. The written script is an explicit representation of this intention. Methods in the script use

external parameters to generate a digital geometric model. The form of the subsequent printed model is calculated as a function of the initial parameters, two boundary splines and a vector indicating the orientation of the light. By varying these parameters, a set of design options can be generated and 3-D printed for comparison. The combination of scripting and 3-D printing allows complex design intentions to be managed in a concise, sharable format and modeled iteratively without manual intervention (Fig. 1).

Introduction

Architects make many artifacts along the way to a finished design. These objects are independent investigations which "can embody, symbolize, and mean in ways that are identical to the cultural artifacts we identify as buildings or paintings or other finished works." (Porter 2004) If architectural design is to be embraced by a broad culture as an important process and not just a product, architects should make use of artifacts which embody the design process. These artifacts will be referred to in this paper as process objects. They represent procedures and behaviors capable of generating the data necessary to construct physical objects. A process object might encode a set of design constraints for building a wall out of bricks or the instructions for folding a paper airplane. Process objects enhance collaboration between design participants by allowing people to share design intentions in an explicit format and reflect critically upon their own design process. In addition, process objects might be related explicitly to external factors defined by a physical design context.

A design context is described by Stiny and March as an understanding of the physical world as conditioned by a particular design intention. The design context determines how physical objects are represented and constructed in relation to a design. "They fix the requirements of meaning and purpose that designs must meet and the means available to build or manufacture objects to such designs" (Stiny and March 1981). Relating process objects to external factors enables architectural design to draw upon environmental performance criteria in early phases of the design process. In addition, process objects can help architects to explicitly relate design intentions to fabrication constraints.

This paper presents a framework for developing and discussing process objects as part of architectural design. In the examples presented, scripts written in Rhinoscript are used to define design intentions. The scripts generate digital geometric models which are in turn exported to a Z-Corp 3D printer. The scripts, digital geometric models, and 3D prints can all be regarded as process objects because they are linked in a cycle of design creation. The design context is defined by the representational system of the Rhinoceros modeling environment and the fabrication capabilities of the Z-Corp 3D printer.

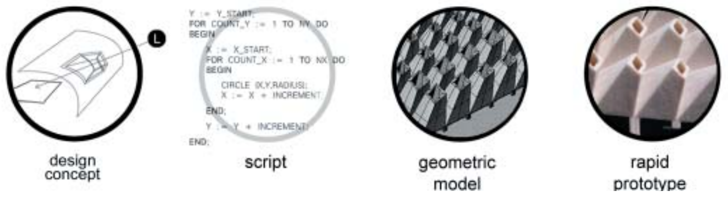
This framework has been used in an architectural workshop in the Department of Architecture at MIT. I was involved as a research assistant. The workshop was entitled "*Generative and Parametric Tools for Design and Fabrication*." It was led by Axel Kilian, Terry Knight, and Lawrence Sass. Carlos Barrios was a research assistant. In discussing the implications of the framework for architectural design, I will draw upon quotes from students and staff involved in the workshop (Fig. 2).

Methods

1. Generative Scripting

The notion of using scripts or programming to encode a design theory is described in William Mitchell's 1987 book, The Art of Computer Graphics Programming. Mitchell proposes procedural programming as a generative mechanism for architectural design. This book followed a tradition of treatises since Vitruvius which establish underlying compositional rules for architectural design. However, Mitchell's book describes a language for rulebuilding rather than the compositional rules themselves.

A script or end-user program (EUP) is a set of instructions written in computer code and executed within a specific software environment. Scripting languages are written using the same basic structures as full-fledged programming languages: variables, loops, conditionals, and functions. In addition, scripts can make use of functions already coded into the parent software environment or add new functionality. There are limitations to what can be scripted in any given software environment. However, scripting provides just the right amount of access to underlying structures, which allows one to represent personal and



```
Y := Y_START;
FOR COUNT_Y := 1 TO NY DO
BEGIN
    X := X_START;
    FOR COUNT_X := 1 TO NX DO
    BEGIN
        CIRCLE (X,Y,RADIUS);
        X := X + INCREMENT;
    END;
    Y := Y + INCREMENT;
END;
```

Figure 1: Design Artifacts

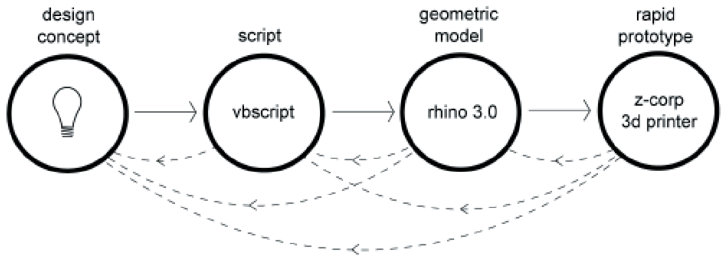


Figure 2: Framework for a Process Object

project specific design processes. The process objects described in this document were written in RhinoScript, the scripting language of Rhinoceros (Rhino), a 3D modeling environment. RhinoScript is based on the Visual Basic Programming Language developed by Microsoft. The techniques described here can be applied to architectural design using any end-user or conventional programming language.

2. Digital Fabrication

This exploration has been guided by an intention to design physical objects through procedural programming. The examples were developed using a consistent means of 3D additive fabrication with a Z-Corp 3D printer. The Z-Corp builds up solid models from many layers of bonded plaster. This machine houses two columns of plaster, a feed and a build, in side-by-side containers. Before printing, the feed container is filled below the surface of the machine with enough plaster to build a solid plaster block the height and width of the given model. The build container starts out nearly flush with the top of the machine. As the machine runs, the feed column rises and a mechanical arm spreads a thin layer of plaster from the feed across the surface of the build column. The cross section of the 3D model is printed onto this thin layer in bonding liquid. The build column is then depressed slightly to allow another layer of plaster to be spread on top. The feed container rises again and the process repeats. As subsequent layers are shifted from feed to build and the next cross section of plaster is bonded to the previous one, a solid model of bonded plaster is slowly assembled out of thin slices, surrounded in the build container by loose plaster.

Typically, a physical object produced on a 3D printer is designed manually as a digital geometric model. In order to be fabricated by the printer, the digital model must be translated into 2D printing paths. This requires that the geometric model be broken down into a series of paper thin horizontal slices. The printing paths are derived from these slices and used to drive the printer head. In contrast, the first computer numerically controlled (CNC) machines were developed to operate straight from symbolic code which described the machine paths. The framework presented here is a return to the use of code, in the form of scripts, to drive computer-controlled fabrication machines. However, scripts do not describe the 2D paths directly. Scripts rely on the geometric abstractions encoded into the parent modeling program. The scripts automate the use of the modeling program and not the printer. The process embodied in a script does not escape the constructs for handling forms within conventional 3-D modeling programs (points, lines, surfaces, solids). In this framework, the Z-Corp 3D printer and the Rhinoceros modeling program are both part of the design context as defined above. Rhinoceros defines how physical objects are represented in relation to the design intent. The Z-Corp machine determines how physical objects are fabricated.

3. Related Work

Several other researchers have worked on automating a design process through the use of rule-based design and rapid prototyping in conjunction. Notable contributions to this effort include those by Axel Kilian, Yufei Wang and Jose Duarte.

Axel Kilian's paper "*Fabrication of partially double-curved surfaces out of flat sheet material through a 3d puzzle approach*" explains how he bridged between a generative design script written in AutoLisp and

2D rapid prototyping machines. Killian used AutoLisp scripting as a schematic tool to make surfaces which have embedded joints. The joints have varying shapes which are responsive to the overall geometry of the sketch surface. The design of surfaces and accompanying joints are meant to take advantage of the characteristics of 2D rapid prototyping machines.

Yufei Wang and Jose Duarte have published work which couples the use of rule-based design and rapid prototyping with a 3-D printer. Their paper, "*Automatic Generation and Fabrication of Designs*," introduces an educational framework for using 3D printing to physically realize architectural massing designs developed using Shape Grammars (Stiny and Gips 1972). This is based on an earlier paper, "*Basic Grammars and Rapid Prototyping*," by Duarte and Simmondetti, which explains the use of 3D printing to fabricate designs produced using AutoLisp code and the Shape Grammar formalism. Their work addresses some of the difficulties in producing complex three-dimensional designs with Shape Grammar rules.

In contrast to the approaches summarized above, the framework presented in this paper is novel in its emphasis on relating the combination of programming and rapid prototyping to performance criteria through the design context. In addition, this study presents conclusions drawn from observations and interviews of designers using this framework.

Examples

This section presents three examples of context-specific process objects encoded as scripts and used to produce multiple physical design models using a 3D printer. These examples were developed by the author in an exploratory manner as a way of clearing paths for future work. As such, many of the scripts are intended to be educational. During the process of developing these scripts, I sought to bridge gestures, rules, and fabrication as distinct but relatable ways of making. Emphasis is placed on varying control systems in scripts. Presented here are scripts which respond to numeric, gestural, and environmental control mechanisms. This set of process objects gives some insight into the characteristics of computational, explicit rule-making as part of design.

1. Numbers
Process Object A

This process generates a grided membrane which modulates light. The membrane is composed of a matrix of extruded cells. Each cell has a variable depth and frame thickness. The amount of light that can pass through this membrane at any one cell is a function of the depth and frame thickness of that cell. This script was used to produce approximately 10 different physical models using the 3-D printer in which the depth and frame thickness of cells are varied. Each of these models performed differently when tested under similar conditions of natural light. Each model also had a different material strength, cost, and production time as a result of being printed physically. The models range from an instance in which all the cells have identical frames and are extruded to one consistent depth to an instance where each cell has a different frame size and depth (Fig. 3).

Numerically generated patterns are an obvious basis for controlling design output. Patterns are compositions based on numerical progressions or consistent geometric transformations. Patterns are often the first programmed models that people build, simply because they allow for variation with minimal effort. Patterns might use linear, quadratic, and wave functions in order to control the way in which repetitive elements change in an orderly way across the span of a surface. Direct numeric control of scripts can also be used to generate random compositions. However, in scripting, randomness is never uncontrolled. In Visual Basic, the parent language of RhinoScript, there is a predefined function, Ran, which generates random numbers between 0 and 1. It is possible to scale this range to any desired set of numbers through a simple multiplication. Therefore, the range of random numbers is precisely controlled by the designer. As a result, the parameters of a model which are random are quite controlled. In the example above, I chose to specify the majority of the dimensions explicitly including the number of columns and rows and the overall size of the model. Meanwhile, the size and depth of the individual grid cells remains variable. In other words, some aspects of the model are set but others allow room for play (Fig. 4).

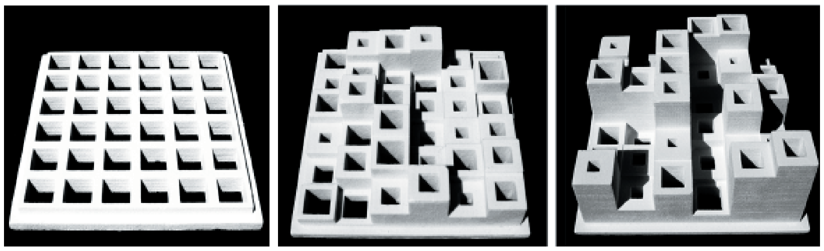


Figure 3: Plaster models from Process Object A. Z-Corp 3D printer

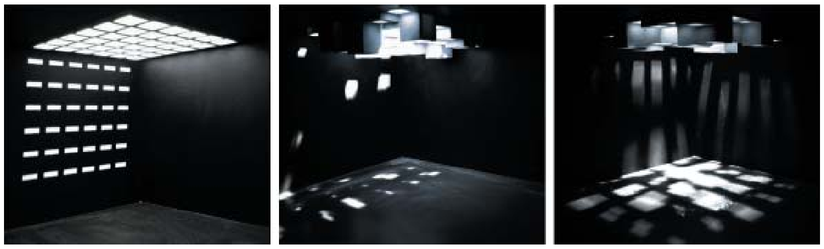


Figure 4: Lighting tests with Process Object A. Z-Corp 3D printer

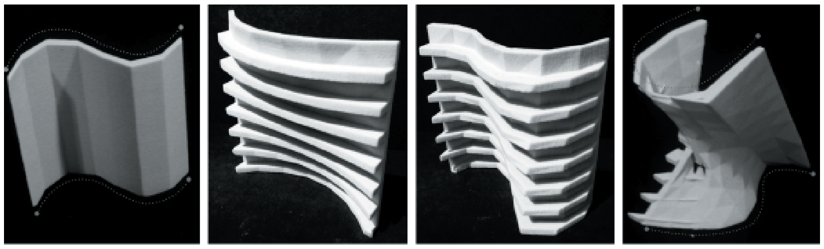


Figure 5: Plaster models from Process Object B. Z-Corp 3D printer

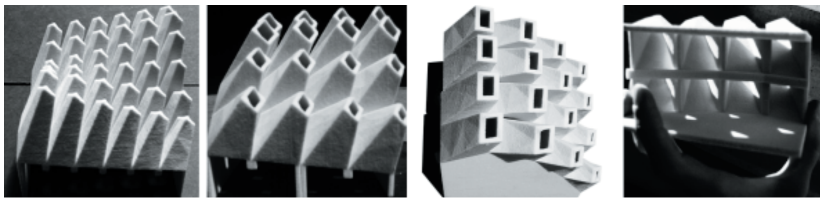


Figure 6: Variations from Process Object C. Z-Corp 3D printer

2. Gestures

Process Object B

This process generates a complex curved surface with supporting ribs from a pair of hand drawn profile splines. This script uses a simple interpolation algorithm to find intermediary points between the two input curves. These intermediary points are stored and later recalled to define the corner points for tiles that make up the final surface. This process was used to generate over one hundred digital models and to print 10 physical models (Fig. 5).

Process Object B demonstrates how a set of rules can respond to hand drawn input from the user. Gestural inputs may also be in the form of scanned images or digitized three dimensional objects. Because this process has more variable output than the previous example, many of the digital models generated by the script could be displayed on screen but not 3d printed successfully. This can be attributed to the geometric limitations of what can be fabricated on the Z-Corp machine. Only a limited set of solid, structurally stable forms can be printed.

Ideally, objects are designed to take into account the output method as part of the design context. The output method is the means of fabrication which will be used to craft a physical object. This object shows a clash between the characteristics of the process object and the design context as defined by the 3D printer. Many of the surfaces which result from this script have twists and folds which cannot be accommodated by the material system of the 3D printer. In this example, the process object does not account for all the limitations of the printer. As a result, some conditions in the initial profiles can result in surfaces which are too thin to be printed at all. Even some of the printed surfaces show tears and fragmentation.

3. Environments

Process Object C

This process generates a surface composed of light-modulating conical components. It combines functions from Process Object A, the first light modulating membrane, with the function for deriving surfaces from gestures explained in Process Object B. In addition, this process tracks the position of an adjustable vector representing the orientation of a light. Using the information from this vector, the script generates a matrix of cells which share the same orientation as the vector. The result is a grid of light tracking components which traverse a complex curved surface (Fig. 6).

The complexity of Process Object C is a function of a simple simulated environment. Patterns might arise from such a strategy, however, these will be informed contextual patterns and not based on abstract geometry. In Process Object C, the generated form is quite complex and would be difficult to model manually on a computer, let alone physically. Process Object C produces a matrix of light-responsive apertures, each with a unique geometry. This is not accomplished by isolating each local condition and resolving it. It is done by specifying a general method which is applied to many different conditions. As long as each condition has the appropriate defining parameters, the general method can be applied. This results in a different final geometry for each distinct condition. In this way scripting can elevate the design process to the level of relationships as opposed to individual components.

Discussion

1. Design Rules

'Rules in design are largely implicit, overlapping, diverse, variously applied, contextually dependent, and subject to exceptions and critical modification.' (Schon 1988)

The act of designing a building can be understood as the establishment of order-inducing rules on an existing site. According to former MIT Professor Don Schon, designers formulate and test rules implicitly in the form of drawings and models. In this investigation, the process object framework has served as an instrument through which to study the effect of using design rules which are purposefully external, unambiguous, and perhaps capable of generating unexplored physical forms.

In programming, rules always have to be mathematical, or about the geometry. This is not always true in rules outside of the machine.
—(Workshop Student)

The conflict between implicit and explicit means of representing design concepts permeates this work. I do not presuppose the dominance of one technique over the other. I have tried to explore the translation from one to the other and in doing so expose the characteristics of both as means of expression. In my investigations, I have found that it is difficult for most architects to separate these two ways of working.

Programming helps you or forces you to make your rules concrete. I used rules in the past but they were more vague, or I didn't realize that I was using them; but I wouldn't be able to understand the difference between how I make rules now and how I used to make rules unless I went back to my old way of working. And that would be very difficult after having programmed rules.
—(Workshop Student)

Computation describes *how to*. It is the expression of logic from an imperative point of view (Sussman and Sussman 1996). Scripting supports the structured development of form through repetitive procedures while allowing for variation through the adjustment of parameters on which those procedures rely. Computational ideas can be expressed as process objects through computer code and, more recently, as physical objects through rapid prototyping. This thesis explores the meeting between a way of thinking and a way of making which gives rise to an aesthetic of structure and variation.

Computational representations allow designers to define the conditions by which designs may be derived. This is a means of externalizing a process in an unambiguous way. However, transferring design ideas into computer code should not be misinterpreted as the recording of design knowledge. Knowledge about designs is a subjective construction that might be linked to a shared set of rules but ultimately evolves from the interaction of designers, clients, and consultants.

Designers who have collaborated using scripts have described this process as more transparent than collaborating through a conventional design representation. Conventional drawings make the shape of a design explicit, but leave out a definition of how the shapes are derived and related.

My partner and I shared code.... It was easy to see what the other person was trying to do in code. You could see exactly what had been added and you could leave comments for each other.
—*(Workshop Student)*

2. Design Context

Three primary ways of representing designs have sustained this exploration: computer code, computer models, and physical prints. Each of these representations has unique structural properties which determine how they might be composed and further manipulated. Together they define the design context.

Computer code has an explicit language of expression. It is composed of imperative statements which are arranged in a modular structure and are parametrically adjustable. Physical prints have a double structure; they refer to their generative code and an intrinsic material behavior. In the context of this exploration, 2D screen-based computer models have served as an intermediary between coded processes and physical products.

Creating scripts in Rhinoceros is largely the manipulation of symbols which refer to locations specified by x, y, z coordinates. At the base of any system created through scripting is a reliance on points as the base elements of composition. Objects which are defined in terms of points must be manipulated in terms of their defining coordinates.

There is a disconnect between what you need to know to design and what you need to know to code. You have to know too much to code all points…example of apertures. I can sketch a shape, but to codify it I need to know all of the geometry that makes up that shape.
—*(Workshop Student)*

This has many advantages in terms of precision and the ability to apply a generalized solution to many objects defined by the same number of points. However, many of the workshop students had difficulty thinking outside of the terms of higher level conventional modeling functions. In asking for help, students spoke in terms of such high-level behaviors as "*rotation,*" "*stretching,*" and "*solid boolean operations.*" Eventually it was necessary for students to explicitly define each of these procedures in terms of how the defining coordinates change in the object of manipulation. Eventually it is necessary to build one's own functions.

Scripts must be tailored for specific output devices. When fabricating from a digital file, one must take physical considerations into account. For example, in order to generate a printable model for the Z-corp printer, a script must specify only solid objects. Objects must have a minimum thickness (usually 1/8" depending on the shape of the object) and maximum size. The allowable thickness of a model is partly a function of the overall proportion of the model. However, the thickness of the form determines the amount of time is takes to dry and become strong. Forms which are very thick may be more likely than thinner forms to buckle because they have a wet, weak core. Surfaces or incomplete solids generated by the script will result in a damaged print. When developing the Process Object C script, I had problems printing the model because some of the generated digital geometric objects were not solids. This made the model impossible to print.

Models generated by scripts cannot usually be scaled and printed at different sizes easily. The physical demands of objects change with changes in scale. A script must be altered when the model is scaled in order to account for increased self-weight and new joint conditions. The objects described in the example section all account for some physical aspects of the 3D printer. However, many times the objects still failed to print successfully. When the geometry that I wanted to produce went beyond the ability of the printer, I had to find ways to rewrite the code which would enable my designs to be printed. It was only possible to understand the constraints of the fabrication system after extensive testing (Fig. 7).

Architectural constructions are generally component-based. However, the Z-corp produces only monolithic, solid objects. In order to study component-based designs, it is necessary to fabricate multiple objects that are computationally related but physically separate. Using 3D printing to produce these assemblies alone poses many problems. The Z-Corp printer lets one produce solid objects with complex curves but not thin or flexible surfaces. It yields the same structural resolution everywhere when you might get by with less material in places and consequently increase the speed of production or flexibility of the model. A hybrid approach to fabrication would ensure that you have the appropriate material behavior in different parts of a model.

Working between geometric rules and material results, it became obvious that there are identifiable constraints which can be taken into account during the process of scripting. Codifying these constraints is one way of defining a material system. This process helps design at a schematic phase stay within the bounds of what is buildable. Codifying material constraints may also suggest ways to design forms which are more aesthetically sensitive to material properties. A formal aesthetic may develop out of a creative consideration of materials and fabrication techniques.

3. Design Search

During the workshop, the teaching staff tried to encourage the perception of physical models generated from scripts as process objects, with associated ways of being interpreted and manipulated. By iteratively augmenting the process, multiple physical variations can be produced with a precision that allows them to be compared accurately. However, as soon as students had a physical model, most students reverted to discussing the design as a fixed object (Fig. 8).

Right now it is just an object. How do you get back to a stage in which you can find variations, or is that counter-productive when you are just trying to get to fabrication?
—(Workshop instructor)

There are two ways of augmenting scripts to create geometric variations. Variations can be produced in the geometry without disturbing the underlying topology. Variations in topology can also be arranged by changing the number of parts or underlying relationships between parts. There are also two evident directions in which to sort through these variations: optimization and exploration. Optimization is a continual narrowing of the search to a specific goal. This type of search is commonly called *hill climbing* and is used extensively in engineering applications.

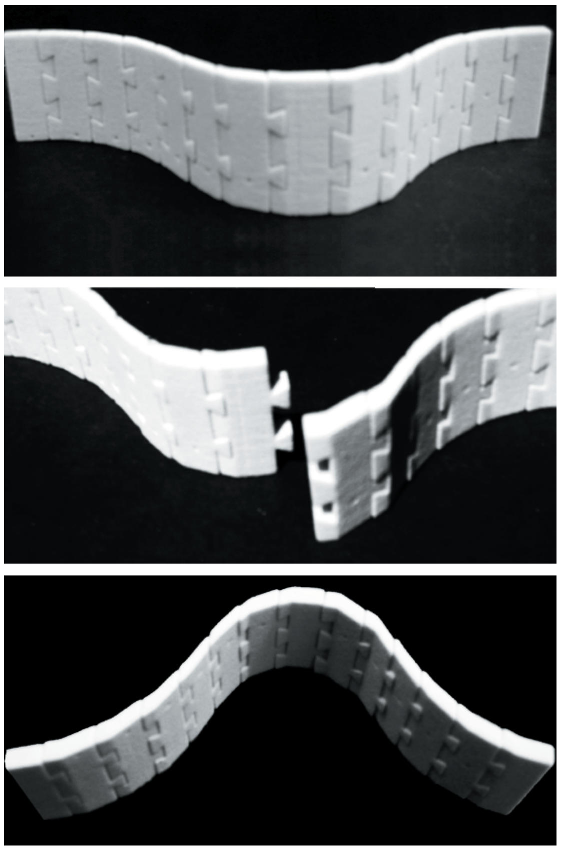


Figure 7: Component-based plaster models from Process Object B. Z-Corp 3D printer.

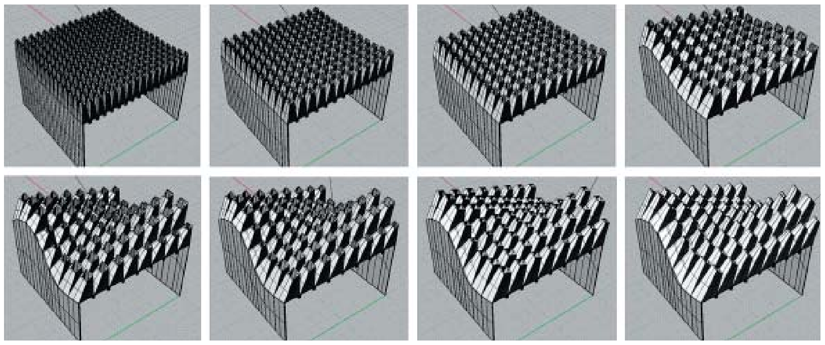


Figure 8: Geometric model variations from Process Object C. Z-Corp 3D printer.

Engineers often optimize for a limited number of variables which control important issues such as the strength of a system, its speed, or its cost. In searching for an optimized solution, engineers typically try to refine their representation of the problem. "If you find the right representation, the solution falls right out." (Winston 1992) However, a problem can only be correctly stated if there is some certainty about what the unknowns are. The kind of optimization that this statement implies can only be calculated for a limited set of variables. In a typical architectural brief there are countless variables.

Computers may introduce a false sense of having optimized a design, which may be fundamentally ill-conceived.
—(Frazer 1995)

As mentioned, variations can also lead to exploration. For a topologist, a coffee cup and a donut might possess the same structure. They are geometrical variations of the same topology. But of course this is a huge conceptual and pragmatic distinction. Exploration of variations can sometimes result in paradigm shifts which reframe how the initial problem is understood.

The problem associated with making too many variations is of course evaluation. It is possible to take a purely performance-driven view of evaluation. This can be helpful in narrowing the acceptable range of variations. However, the definition of satisfactory performance should not be a means to pass off the responsibility of design. Designers who accept this means of validating designs must investigate performance criteria critically. High performance in one area may mean low performance in another. As mentioned above, it is difficult to optimize more than a few variables at a time. There is also a tendency to weight the criteria most heavily which can be expressed explicitly. This creates a tension between performance-based evaluation and design poetics. Students who relied on metaphorical sources of control in their work produced results that could not easily be interpreted in terms of success and failure. These schemes were in danger of becoming meaningless in the context of the optimization paradigm.

Many practicing architects and researchers are working on ways to connect design systems and environmental analysis more strongly. In a recent lecture at MIT, an architect said that his ideal computer tool would be "*something like a slider*" that would allow him to experiment in the space between environmental optimization and aesthetic purity. In this scenario, the computer would be positioned, somewhat uncomfortably, between aesthetics and good policy.

Conclusion

Process objects can help architects to elevate the importance of design as a process. This paper identifies methods for handling both digital processes and products within the framework of process objects. Scripts, geometric models, and physical prototypes have very different structures. However, they are all relatable through the concept of a process object. Throughout this paper, the development of process objects has been shown to be structured both by personal design decisions and by the nature of the design context.

The examples provided show how this approach can generate physical forms through both direct instructions and responsive rule sets which follow gestural and environmental input. Ultimately, through process objects design investigations can be directed inwards towards optimization or outwards towards conceptual exploration.

This work has shown that process objects enhance collaboration in architectural design. They allow participants to share design intentions and constraints in an explicit format and reflect critically upon their own design process. In addition, process objects allow designers to explicitly define the design context and relate design decisions to external factors in a precise way.

In the transition from implicit to explicit representations of process, the nature of architectural design is likely to change. However, it is not likely that the computer will eradicate older media. More likely, it will shift the way that these media are understood and applied. The computer is a reflective medium which frames how designers work and, in doing so, allows architects to see themselves critically.

References

Abelson, Harold, and Sussman, Gerald Jay. (1996). *Structure and Interpretation of Computer Programs*. Cambridge, MA: MIT Press.

Duarte, Jose Pinto, and Simondetti, A. (1997). Basic Grammars and Rapid Prototyping. *Proceedings of the 4th Workshop of the European Group for Structural Engineering Applications of Artificial Intelligence:* 117-119. Lahti, Finland.

Fliesher, Aaron, and Gross, Mark. (1988). Designing with Constraints. In *Design Studies*, 9, no. 3

Frazer, John. (1995). *An Evolutionary Architecture.* London: Architectural Association.

Kilian, Axel. (2003). *Fabrication of Partially Double-Curved Surfaces out of Flat Sheet Material through a 3d Puzzle Approach*. Indianapolis: ACADIA'03 Proceedings.

Mitchell, William J. (1987). *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers.* New York: Van Nostrand Reinhold.

Mitchell, William J. (1990). *The Logic of Architecture Design, Computation, and Cognition.* Cambridge, MA: MIT Press.

Porter, William L. (2004). Designers' Objects. In *Design Representation,* eds. G. Goldschmidt and W. Porter. London: Springer-Verlag.

Rowe, Peter G. (1987). *Design Thinking.* Cambridge, MA: MIT Press.

Schön, Donald A. (1988). Designing: Rules, types and worlds. In *Design Studies*, 9, no. 3:133-43.

Stiny G., and Gips, J. (1972) Shape Grammars and the Generative Specification of Painting and Sculpture. In *Information Processing 71*, ed. C.V. Freiman: 1460-1465. Amsterdam: North-Holland.

Stiny, George, and March, L. (1981). Design machines. In *Environment and Planning B*, vol. 8: 245-55.

Winston, Patrick Henry. (1992). *Artificial Intelligence*. USA: Addison-Wesley Publishing Company.

Yufei, Wang, andDuarte, Jose Pinto. (2001). Automatic Generation and Fabrication of Designs. In *Automation in Construction,* vol. 11, no. 3 April: 291-302.

Acknowledgements

Sincere thanks to the Center for Bits and Atoms at MIT for sponsoring this research.

Yanni Loukissas is a PhD Scholar in Computation at the Department of Architecture of MIT. He also holds a Master of Science from MIT and a Bachelor of Architecture from Cornell University. He has practiced as an architect in London and New York City and taught design and digital media at Cornell University and MIT. He is a currently a Visiting Professor at the School of the Museum of Fine Art in Boston. His doctoral research is an exploration of architectural thinking from a computational point of view. This work seeks to open up new ways of seeing in relation to the design of built environments.

Lawrence Sass is an Assistant Professor in the Department of Architecture at MIT, where he conducts advanced research and teaching in the field of Digital Fabrication. His ongoing research demonstrated that buildings can be designed and constructed in paperless environments - using CAD/CAM for generating buildings and their components, and rapid-prototyping to test the design of these components prior to construction. His current research, therefore, is focused primarily on the advancement of newer software tools and rapid-prototyping devices that will help foster greater establishment of such paperless design offices. Professor Sass worked for several large architectural practices in Boston and New York between 1990 and 2001. He received his B. Arch. from Pratt Institute, and his M.S. Arch. and Ph.D. degrees from MIT.