Revisiting the use of generative design tools in the early stages of design education

Scott C. Chase

University of Strathclyde, United Kingdom
http://homepages.strath.ac.uk/~cas01101

Abstract. Computer based generative design tools can help elucidate the nature of design, but are often restricted in their scope due to implementation issues. These 'toy' applications are often developed as proof of concept software, but have the potential to serve as teaching aids in early design education. A number of such tools will be described and the case made for their continued use in design education.

Keywords. generative design tools, design education, computer programming, parametric variation, shape grammars.

Introduction

Research into the development and use of generative design tools has been ongoing since the early days of computer aided design. Given the complex nature of design, development of tools that truly automate the design process (as is often the designer's fear) has been slow. Tools developed to date can help elucidate the nature of design, but are often restricted to a limited number of design issues or a portion of the design process. In many cases they are 'toy' applications. Some of these applications were developed as teaching aids, but many were simply developed as initial proof of concept software, with the potential of becoming a more powerful design tool left for the future.

These applications can be useful for teaching design concepts at an introductory level. As they are limited in scope, the student is able to comprehend specific design concepts without the distraction of a complex set of design issues. This parallels some philosophies of operating systems development (e.g. Unix), with simple tools that do one task but do it well.

Traditional computer aided design tools such as geometric modellers can help students with some understanding of their designs and develop their knowledge and skills in areas such as geometry, but they act only as aids; the student must provide the input and directly manipulate the forms. The power of generative design tools is that they can guide the learner through a design exploration.

Rule based design

The tools described in this paper are rule based. According to March (1996), "…rules liberate. They provide the language in which the designer speaks. They give the designer 'style'. Freedom comes from following the rule. Bucking a rule is simply to follow another." This is a powerful concept to introduce to a student at the beginning of their design education. Introductory design teaching can involve the use of game playing as a means of providing a structured method of design exploration via rules, thus providing a low risk (and fun) design environment (Woodbury, Shannon, and Radford 2001).

Rule based generative design systems were more in vogue in schools of architecture over a decade ago, when CAD systems were not as powerful as they are today, and researchers were

still very interested in computational design tools. However, the expressive power of current CAD systems has encouraged designers to produce radically new forms, such as those by Frank Gehry, Greg Lynn and Daniel Liebeskind. This, in turn, is leading to renewed interest in generative design tools that can produce such forms parametrically and algorithmically, as evidenced by the recent establishment of the SmartGeometry group (www.smartgeometry.org: May 2003).

Attributes of generative design tools

Woodbury et al. (1992) provide a useful characterisation for interactive generative design tools by three attributes of their representations:

a clear representation has an understandable syntax, creates few exceptional conditions, and relies on a small number of concepts;

a fast representation relies upon algorithms that perform all pertinent operations within acceptable time limits for human interaction;

an applicable representation is useful in a domain, i.e. its features map well to observed features of the domain

A highly interactive system will have a representation that is clear and fast. A useful system will have an applicable representation. Due to the complexity of most architectural design problems, there are few generative design tools that satisfy all of these criteria, and most likely none that can be used at an introductory level. The tools described in this paper for the most part have representations that are clear and fast, key qualities for tools that introduce new concepts. Many may not be considered applicable, by nature of their output being limited in scope and generally abstract. It is then the responsibility of the user to make the connection between an abstract composition and a useful design.

Computer programming

There has been an ongoing debate in architectural schools whether computer programming should be taught within the curriculum. As much CAD development has moved from architectural and engineering schools to organisations that specialise in advanced visualisation, architectural computing research has tended to shift a bit from fundamental CAD development towards the application of CAD tools. This in effect has meant less computer programming instruction, and indeed, its disappearance from the curricula of many architecture schools, as many feel that architects need to use CAD tools effectively for design, thereby leaving programming to professional programmers. However, programming provides an understanding of computation (which is one view of design), and thus can be useful in teaching design concepts.

Programming at an introductory level can consist of fast and clear representations that might be of limited applicability (e.g. the Logo programming language often taught in primary schools). Once key concepts (e.g. structure, reiteration, recursion, conditionals) have been learned, they can be applied in more complex environments that produce more expressive and applicable results for architectural design (e.g. GDL, Custom Objects, Java, C).

The book The Art of Computer Graphics Programming (Mitchell, Liggett, and Kvan 1987) describes a method for teaching a theory of hierarchical, top down design with parametric variation in conjunction with programming paradigms. This later became encapsulated in the TopDown software (Mitchell, Liggett, and Tan 1990), which provided a graphic front end to allow designers easy access to the resultant programs. (see Parametric variation below).

Current examples of introductory programming taught in architecture schools include

FormWriter (Gross 2001), a simple programming environment based on Logo concepts, and the work of Burry et al. (2002), who advocate programming as a means for extending spatial and temporal understanding. The latter argue the case for teaching low-level computer programming: "…as an introduction or at least a consolidation of the realm of descriptive geometry and in providing an environment for experimenting in morphological time-based change… When computer programming is used as a platform for instruction in logic and spatial representation, the waning interest in mathematics as a basis for spatial description can be readdressed using a left-field approach. Students gain insights into topology, Cartesian space and morphology through programmatic form finding, as opposed to through direct manipulation."

A sampling of generative design tools

This section describes a number of generative design tools that have been or could be used for teaching at an introductory level. They specifically deal with form generation. A number of these are probably no longer in use but still have pedagogic value (and will run on today's computers!). Others are more contemporary and development is ongoing. The list here is by no means exhaustive but includes tools with which the author is familiar. Not all generative design paradigms are included (e.g., tools employing an evolutionary approach using genetic algorithms were not investigated). The author's strong interest in grammar based design generation should be evident in the extended discussion of grammar systems.

Pattern generation

Composition is a significant part of architectural design, and formal composition should be an important part of an architectural curriculum. There have been a number of tools that allow rapid exploration of abstract form generation, some developed within architectural schools. Such tools include

the programming language Logo: through its use, the student learns basic programming concepts as well as those of geometry and form generation. Many free tools are available; the author has used MSWLogo in his teaching (www.softronix.com/logo.html: May 2003).

cellular automata: the Game of Life (Gardner 1970) can serve as an introductory tool for teaching growth simulation (Figure 1). Krawczyk (2002) has developed methods to impose architectural interpretations upon cellular automata, thus allowing students to see the architectural possibilities in what may have been an abstract form generator.

recursive patterns: There are many programs which generate recursive patterns, including fractals. Krawczyk and has implemented several routines as Java applets (www.iit.edu/~krawczyk/java.html: May 2003). DiscoverForm (Carlson and Woodbury 1994), is a good example of a tool that generates fractals as well as other types of recursive patterns in a high-

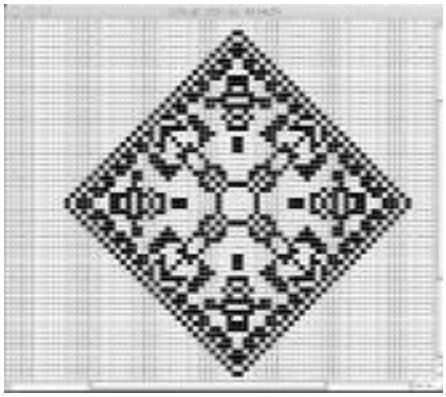

Figure 1. A Java applet to generate configurations from Conway's Game of Life.

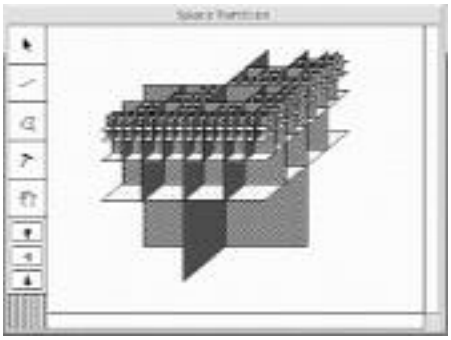

Figure 2. DiscoverForm application for generating recursive patterns.

ly interactive manner. This program provides a fast and clear representation to allow the user to not merely draw patterns, but learn about them and discover new ones (Figure 2).

Parametric variation

As mentioned above, the teaching of structured decomposition and parametric variation via computer programming led to the development of the TopDown software, which provided a graphical user interface for a number of applications supporting parametric variation (Mitchell, Liggett, and Tan 1990). This enabled students (designers) without any programming experience to manipulate and explore designs parametrically. The paradigm also incorporated elements of shape grammars. An example using a primitive hut grammar was used in introductory structures and CAD classes at Harvard University (Mitchell et al. 1992). Students were able to explore alternatives and variations freely, thereby developing an understanding of the relationships among element, arrangement, material and dimension, gaining a sense of the spectrum of possibilities structured by the grammar. It was felt that such an approach could have wide applicability in teaching construction, structure and environmental systems, combining features of the design studio (rich, complex problems encouraging formal exploration with little analysis) and technical classes (careful analysis on simple examples, with little opportunity to explore alternatives). However, a disadvantage of this approach is in its restricted representation: a predefined, fixed, hierarchical decomposition on designs, lacking the capacity to transform emergent shapes, as a general shape grammar would allow.

Grammars

Shape grammars (and other design grammars) have been in use for over three decades. Their utility in design generation and analysis has been well documented. Stiny (1980) has made a number of observations on the benefits of using rules to produce languages of designs:

they are less complicated than the designs they produce;

they open up new directions for design within a given vocabulary;

rules shift the emphasis away from individual designs to languages of designs;

use of a language of designs means that a designer can examine a design and its variation without loss of understanding simply by applying the rules to construct them;

rules can be modified systematically to define new languages of design.

Although computer implementations of grammar based systems have been few, due to the computational complexity of grammars and the difficulty in developing useful interfaces (Chase 2002), a few grammar implementations exist that have proven useful in teaching grammars. Their potential for teaching design concepts is slowly being realised.

Tartan Worlds (Woodbury et al. 1992) is a highly interactive generative symbol grammar system that generates designs as 2D configurations of symbols on tartan grids. It has been used in the teaching and illustration of form generation

in a first year design subject, where it served as a means for students to learn about design derivation, grammatical metaphor, and working with spaces of designs rather than individual designs.

Shaper2D (McGill 2002) is an interactive Java application for generating designs using basic grammars (a very restricted type of shape grammar). The system deliberately constrains the user to the exploration of a small portion of the shape grammar paradigm. While limited in its scope, the tool encourages the exploration of 2D spatial relations that can serve as the basis for more complex designs by exporting the results to CAD applications (Figure 3).

Figure 3. Shaper2D application. Two alternate designs are shown in the right panels, created by varying a label position interactively.

Shaper (Wang and Duarte 2002), an earlier tool than Shaper2D for exploring basic grammars, provides a less interactive user interface but a richer palette of shapes and spatial relations (3D shapes and transformations of the user's specification). While facilitating learning of spatial relations and shape grammars, the developer's research has also included fabrication of designs produced with this tool via export to rapid prototyping facilities.

GEdit (Tapia 1999) is perhaps the only shape grammar implementation to provide a user friendly interface and support emergence. The system allows the user to specify spatial relations and rules for 2D shapes of one's own design, and allows the viewing of all possible rule applications and results. This can encourage the beginning designer in their design exploration, as one can backtrack and try alternative paths to a design (Figure 4).

Figure 4. GEdit program. Windows shown include a) rule window abcd; b) current shape in the derivation, with result of a rule application; c) alternate ways in which the rule can apply; d) possible results of a rule application.

A different kind of grammar based design tool has been developed by Andrew Li (2002). It encapsulates a grammar for generating sections of Chinese wood frame buildings according to the Yingzao fashi building manual. The grammar and the system itself are very deterministic. Use of the system suggests that grammars can in fact be useful for describing or guiding a user's participation in a design process, especially if those parts the process which do not require his participation can be hidden. Such a tool is useful for learning a particular design process, and thus shares some characteristics with the predefined, fixed structure of the TopDown system.

Design teaching with grammars

MIT has been among those at the forefront of current design teaching using grammars. The two Shaper tools described above were among those used as studio design aids during the 2001 MIT-Myagi workshop (Celani 2001). Although the workshop was for advanced design students, it appears that these tools could also be used for simple exploratory design tasks, given the nature of their interfaces.

Terry Knight's work with shape grammars has

included their use for teaching design composition. She raises several questions about their pedagogical use (Knight 2000). Should students be introduced to grammars through abstract ones such as those described here? An advantage of abstract grammars is that they easily demonstrate the mechanics of shape grammars and do not limit the possible interpretations of the designs they generate. For beginning students, the connection between an abstract form and an architectural interpretation may be difficult to make. She cites the example of Flemming (1990), who proposed a model for teaching architectural composition using grammars that generate languages such as wall architecture, mass architecture, panel architecture, layered architecture, structure/infill architecture, and skin architecture. Grammars like these that have direct architectural interpretations may be easier for beginning students to comprehend.

A concern with the use of computer implementation for teaching grammars is that they might not be as effective as manual techniques, which require slow, careful deliberations about how rules work. Computer generation of designs might encourage mindless defining and application of rules without a clear understanding of how they operate. The key is to maintain a balance between manual and computer usage, as both have their advantages. As an example, the author initially found it difficult to visualise rules and their possible applications using Shaper and ended up constructing a set of blocks to better understand the operation of specific rules. Once that understanding was achieved, the software was very useful for quickly generating designs. The same applies for the other tools mentioned in this paper: a balanced approach between computer and hand methods produces a better understanding of the methodology.

Summary and conclusions

Tools that provide a limited range of expressive output can be useful as teaching aids to beginning designers if

they teach a specific design paradigm. A good tool should also allow the user to quickly explore a number of design possibilities as well as generate designs that may not have easily been considered using hand methods.

the tool has a clear and fast representation, i.e. the concepts are easy to understand and the user interface is easy to use. If it does not have an applicable representation (i.e. one that directly represents architectural forms and concepts), then an understanding of the tool's limitations should be conveyed to the student. The student should thus have the ability to interpret an abstract design in architectural terms as well as knowing when to follow the rules and when to break them;

in many cases, a computer tool is best used in conjunction with manual methods, which may need to be taught first, so as to give the student an understanding of the theoretical concepts of the computational paradigm and its mathematics. In this way, the student acquires the fundamental knowledge in geometry and programming structures that is often lacking in today's curricula.

The tools described here are specifically for form generation and could serve well in introductory theory and composition classes, as well as for some design studio exercises. This does not preclude the use of tools for other applications (e.g. the hut grammar implemented in TopDown was also used to teach structures and construction).

Generative design tools can introduce key mathematical concepts useful for design, e.g. geometry, transformations, spatial relations, recursion, reiteration, procedures, encapsulation.

One benefit of in-house tool development is

that the tools can serve teaching and research at several levels. Novice users learn basic design paradigms; intermediate designers may be able to use the tool in conjunction with other methods for more complex design (McGill 2002), and advanced/research students become tool developers, gaining insight into theories of design computation and software development. This approach has proven successful in the past and should be encouraged for the future.

References

Burry, M., Datta, S. and Anson, S.: 2002, Introductory computer programming as a means for extending spatial and temporal understanding, in M. J. Clayton and G. P. Vasquez de Velasco (eds), Eternity, Infinity and Virtuality in Architecture: Proceedings of ACADIA 2000, Association for Computer Aided Design in Architecture, pp. 129-135.

Carlson, C. and Woodbury, R.: 1994, Hands-on exploration of recursive patterns, Languages of Design 2, pp. 121-142.

Celani, G.: 2001, MIT-MIYAGI Workshop 2001: An educational experiment with shape grammars and computer applications, International Journal of Design Computing 3, http://www.arch.usyd.edu.au/kcdc/journal/vol3/celani/abstract.htm: May 2003.

Chase, S. C.: 2002, A model for user interaction in grammar-based design systems, Automation in Construction 11(2), pp. 161-172.

Flemming, U.: 1990, Syntactic structures in architecture: teaching composition with computer assistance, in M. McCullough, W. J. Mitchell and P. Purcell (eds), The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era, MIT Press, Cambridge, Mass., pp.31-48.

Gardner, M.: 1970, Mathematical games: the fantastic combinations of John Conway's new solitaire game "life", Scientific American 223, pp. 120-123.

Gross, M. D.: 2001, FormWriter: a little programming language for generating three-dimensional form algorithmically, in B. de Vries, J. van Leeuwen and H. Achten (eds), Computer Aided Architectural Design Futures 2001, Kluwer Academic Publishers, Dordrecht, pp. 577-588.

Knight, T.: 2000, Shape grammars in education and practice: history and prospects, International Journal of Design Computing 2, http://www.arch.usyd.edu.au/kcdc/journal/vol2/articles/knight/index.htm: May 2003.

Krawczyk, R. J.: 2002, Experiments in architectural form generation using cellular automata, in K. Koszewski and S. Wrona (eds), Design e-ducation: Connecting the Real and the Virtual, Proceedings of the 20th Conference on Education in Computer Aided Architectural Design in Europe, eCAADe, Warsaw, pp. 552-555.

Li, A. I-K.: 2002, A prototype interactive simulated shape grammar, in K. Koszewski and S. Wrona (eds), Design e-ducation: Connecting the Real and the Virtual, Proceedings of the 20th Conference on Education in Computer Aided Architectural Design in Europe, eCAADe, Warsaw, pp. 314-317.

March, L.: 1996, Rulebound unruliness, Environment and Planning B: Planning and Design 23(4), pp. 391-399.

McGill, M. C.: 2002, Shaper2D: visual software for learning shape grammars, in K. Koszewski and S. Wrona (eds), Design e-ducation: Connecting the Real and the Virtual, Proceedings of the 20th Conference on Education in Computer Aided Architectural Design in Europe, eCAADe, Warsaw, pp. 148-151.

Mitchell, W., Liggett, R., and Kvan, T.: 1987, The Art of Computer Graphics Programming, Van Nostrand Reinhold, New York.

Mitchell, W. J., Liggett, R. S., Pollalis, S. N. and Tan, M.: 1992, Integrating shape grammars and design analysis, in G. N. Schmitt (ed), Computer Aided Architectural Design Futures: Education, Research, Applications, proceedings of CAAD Futures '91, Viewweg, Braunschweig, pp. 17-32.

Mitchell, W. J., Liggett, R. S. and Tan, M.: 1990, Top-down knowledge-based design, in M. McCullough, W. J. Mitchell and P. Purcell (eds), The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era, MIT Press, Cambridge, Mass., pp. 137-148.
Stiny, G.: 1980, Kindergarten grammars: designing with Froebel's building gifts, Environment and Planning B 7, pp. 409-462.
Tapia, M.: 1999, A visual implementation of a shape grammar system, Environment and Planning B: Planning and Design 26(1), pp. 59-73.
Wang, Y. and Duarte, J. P.: 2002, Automatic generation and fabrication of designs, Automation in Construction, 11(3), pp. 291-302.
Woodbury, R. F., Radford, A. D., Taplin, P. N. and Coppins, S. A.: 1992, Tartan worlds: a generative symbol grammar system, in K. M. Kensek and D. Noble (eds), Computer Supported Design in Architecture: Mission, Method, Madness, proceedings of ACADIA '92, Association for Computer-Aided Design in Architecture, pp. 211-220.
Woodbury, R. F., Shannon, S. J. and Radford, A. D.: 2001, Games in early design education: playing with metaphor, in B. de Vries, J. van Leeuwen and H. Achten (eds), Computer Aided Architectural Design Futures 2001, Kluwer Academic Publishers, Dordrecht, pp. 201-214.

Additional Internet resources

All websites listed were accessible as of May 2003.

GEdit:
http://www.shapegrammar.org/GEdit/GEdit.html

Hilbert's Building Blocks:
http://www.iit.edu/~krawczyk/spcrv00.html

LifeLab: http://www.trevorrow.com/lifelab

MIT-Myagi workshop: http://www.mit.edu/~4.184/

Shaper2D:
http://architecture.mit.edu/~miri/shaper2d/

Yingzao fashi grammar:
http://www.arch.cuhk.edu.hk/serverd/staff/andrew/yzfs/yzfswebp/