1. Integrating Shape Grammars and Design Analysis

William J. Mitchell, Robin S. Liggett, Spiro N. Pollalis, and Milton Tan

Graduate School of Design
Harvard University
48 Quincy St.
Cambridge, MA 02138, USA

This paper demonstrates how design problems can be solved by combining a shape grammar to generate alternatives with standard engineering analysis procedures to test them. It provides a detailed worked example, and discusses practical applications of the idea in design teaching.

Introduction: Generating and Testing Design Alternatives

If we take it that design is a generate-and-test process, then it follows that design knowledge may be embodied in the generator, in the tester, or in both (Mitchell 1990). In other words, effective design processes (whether manual or automated) may combine blind generation of alternatives with highly effective selection (evolution's strategy), or employ such foolproof generation techniques that there is no need for further testing to weed out mistakes (God's strategy), or divide the labor between generator and tester in some appropriate way (the demigod's strategy).

It has been shown that the shape grammar formalism (Stiny 1980) provides an effective way to encode knowledge of how to put various types of artifacts together so that they function as required, and that generation of designs in the languages specified by such grammars can be supported by appropriate computer software. It has also been shown, in many contexts, that standard engineering analysis procedures can be applied to computer models of design alternatives in order to determine whether these alternatives are satisfactory. These two observations suggest that a powerful knowledge-based computer-aided design system might result from combining a shape grammar as the generator with analysis procedures for testing. This approach has the additional technical advantage that the shape grammar rigorously and precisely specifies the range of inputs with which the analysis procedures must deal. In this paper we consider such a combination.

The Primitive Hut Problem

The problem chosen as a vehicle for the study was design of a fairly simple three-dimensional framed structure—a modern equivalent of Laugier's archetypal "primitive hut". The structure is rectangular in plan, with a column at each corner (figure 1). Primary structural members span between the columns and support secondary members, which in turn support a roof of planks. Length, width, and height of the enclosed volume can vary arbitrarily.
Normal roof loads for a small garden pavilion are assumed. The designer's problem is to choose, arrange, and connect primary and secondary roof members so as to transfer loads to the columns, to provide adequate lateral stability, and to size all members appropriately.

This is, of course, quite an elementary problem—but a moment's reflection reveals that the knowledge needed to solve it is by no means trivial. For a start, you have to know that beams and trusses are appropriate types of spanning members for use in this context. Next, you have to know something about the various types of flat and pitched trusses, the arrangements of their constituent members, and the reasons for choosing one type rather than another in a particular context. Once you have an idea of what the spanning members might be, you need some basis for choosing the direction in which the primary members will span, and for choosing the number and spacing of the secondary members. Given a basic topology, you need to know how to resolve connection details—particularly at the corners. You need to know the various ways that lateral stability can be provided. And finally, you need to know the rules for sizing beams, columns, and truss chords correctly.

Experienced designers have all this knowledge at their fingertips, and so can produce solutions very quickly and confidently. But inexperienced designers often lack some of the necessary knowledge, and it seems clear that even a small knowledge deficiency can produce considerable uncertainty and can greatly hinder the production of a sound solution. (Beginning students can often be observed struggling with this sort of problem.) The consequences of gaps in knowledge show up in several ways: a designer may fail to produce a solution at all, or may produce a solution without being confident that it is satisfactory, or (worst of all) may believe that a proposed solution is satisfactory when, in fact, it would fail.

The function of a knowledge-based computer-aided design system (in general, and for this problem in particular) is to supply knowledge that might be missing, and so to enable inexperienced designers to perform at the level of experts. An effective system should allow inexperienced designers to produce solutions quickly, and it should guarantee that these solutions are adequate.
Vocabularies and Rules of Usage

The knowledge needed to solve problems can often be encoded in the form of conditional rules, and knowledge-based computer-aided design systems usually incorporate such rules. Each such rule specifies a condition, and associates with that condition an action which may appropriately be taken in response. To apply knowledge that is expressed in this form, you first try to find a match between the conditions at hand and the condition part of a rule. Then, when you find a match, you follow the instructions in the action part of the rule. Consider, for example, the problem of crossing the road safely. A traffic light has three possible conditions: red, yellow, and green. The rules that you need to know are: if red then stop; if yellow then continue if started else stop; if green then go. To apply these rules you need to look at the light, find a match between the condition of the light and the condition part of a rule, then act as instructed by the action part of that rule. Notice that it does not help to know the rules when you are unable to recognize conditions—if, for example, you are color-blind. Nor does it help to recognize the conditions if you cannot associate actions correctly—if you can see that a light is red, but do not know what this means.

In architectural contexts, it is particularly convenient to express conditional rules graphically. In other words, you draw a context (the condition that is to be recognized), and you also draw an appropriate design response to that context. Some simple examples of such rules are shown in figure 2. These are rules of usage for the architectural elements that show up on the right (action) sides of rules. In other words, rules of usage specify the contexts in which elements such as pedestals, columns, and entablatures can be used, and show the possible relationships of these elements to other elements.

![Figure 2: Rules of usage for some architectural elements.](image)

The elements that appear in rules of usage may be actual construction elements and spaces, or they may be abstract shapes and volumes that serve as construction lines, grids, axes, placeholders for later substitution of something else, and other devices that guide the development of a design. These marker elements are removed when they no longer serve a useful purpose, they do not explicitly form part of the final design, and they do not show up in bills of materials or space inventories. We distinguish them from terminal elements, which do show up in final designs, and do appear in bills of materials. In figure 2, rule 1, for example, a simple rectangle initially serves as a marker for the whole design. The rule specifies
replacement of this marker by an assembly of lower-level markers showing the outlines of pedestal, column, and entablature. Then rules 2, 3, and 4 show how these lower-level markers can, in turn, be replaced by terminal vocabulary elements: cornice, frieze, architrave, capital, shaft, column base, cap, dado, and pedestal base.

Exhaustive specification of the ways to use some architectural element (whether a terminal or a marker) requires identification of all the contexts in which that element can appear, and specification of one or more rules for each of those contexts. By establishing a vocabulary or marker and terminal elements, and writing complete rules of usage for these elements, we can define a shape grammar that specifies a language of architectural form. Formally, such a grammar consists of a marker vocabulary $V_m$, a terminal vocabulary $V_t$, a set $R$ of rules of usage, and a starting shape $S$. Designs in the language specified by the grammar are derived by recursive application of the rules $R$ to the starting shape $S$. A rule applies when its left side matches a subshape of the current design in progress. The result of application is replacement of that subshape by the shape shown on the right side of the rule. A derivation terminates when no further rule applications are possible. If the resulting design contains no markers, then it is a complete design in the language specified by the grammar. (If it does contain markers it is a design dead-end—a configuration which cannot be developed into a complete design under the given rules of usage.) The language specified by the grammar consists of all the complete designs derivable in this way.

To describe how to put together a primitive hut, then, we need to specify an appropriate terminal vocabulary, a marker vocabulary, and rules of usage. A reasonable terminal vocabulary consists of the following: timber columns, masonry columns, timber beams, parallel-chord timber trusses, pitched timber trusses, bow timber trusses, knee braces, cross braces, one-way braces, and masonry shear walls (figure 3).

![Figure 3: Terminal vocabulary for the primitive hut.](image)

These terminal vocabulary elements can be put together to form subsystems that provide necessary functions (figure 4). For example, we can put together four columns to produce a supporting substructure that holds up the roof. We can put together beams, parallel-chord trusses, and pitched trusses in various combinations to become roof structures that hold up a plank roof and so provide shelter. And we can put together a support structure, a roof structure, and braces or walls to produce a system that has lateral stability.

![Figure 4: Some subsystems of the primitive hut.](image)
When we want to represent a subsystem without committing ourselves to details of its internal organization, we can show it as an abstract volume. Figure 5 illustrates the various subsystems of the primitive hut, depicted in this abstract way. These abstract volumes constitute the marker vocabulary of the primitive hut grammar. Markers serve as the intermediate abstractions that we use in thinking about how to put terminal vocabulary elements together in subsystems so that they function as required. By combining markers in different ways, and substituting different arrangements of terminals for the same marker, we can generate different hut designs.

**Figure 5: Marker vocabulary for the primitive hut.**

Notice that the markers correspond to architectural roles. The rules of usage, then, must specify ways to use terminals and combinations of terminals in these roles. They must show how to use timber and masonry columns as structural supports, beams and trusses as primary structural members, beams and trusses as secondary structural members, and knee braces, cross braces, and shear walls as lateral stability systems. Figure 6 shows our rules of usage for putting together terminal vocabulary elements to produce subsystems, and for putting together lower-level subsystems to produce higher-level subsystems. (The design variables introduced at each step are listed in italics on the right.)

**REFINEMENT OF THE PRIMARY STRUCTURE:**

```
Starting shape
Sub-1

```

```
Starting shape
Sub-2

```

```
Starting shape
Sub-3

```

```
Starting shape
Sub-4

```
ORGANIZATION OF MAJOR SUBSYSTEMS:

Flat primary structure
Rule 5
(applicable without primary overhang)

Flat primary structure
Rule 6

Pitched primary structure
Rule 7

Bowed primary structure
Rule 8

SUBSTITUTION OF SECONDARY STRUCTURE TERMINALS:

Flat-bottomed secondary roof structure
Rule 9

Pitched butteres
String
Such a depth
Secondary length
Secondary size height

Flat-bottomed secondary roof structure
Rule 10

Bowed butteres
String
Such a depth
Secondary length
Secondary size height

Flat-bottomed secondary roof structure
Rule 11

Bases
String
Such a depth
Such a width (calculated)

Flat-bottomed secondary roof structure
Rule 12

Flat butteres
String
Such a depth
Such a width
Secondary size height

Pitched secondary roof structure
Rule 13

Joints
- Such a base
- Aperture
- Width (calculated)

Bowed secondary roof structure
Rule 14

Joints
- Such a base
- Aperture
- Width (calculated)
SUBSTITUTION OF LATERAL STABILITY TERMINALS:

- **Primary lateral stability**
  - Rule 15
  - Diagonal brace

- **Primary lateral stability**
  - Rule 16
  - Cross braces

- **Primary lateral stability**
  - Rule 17
  - Knee braces

- **Primary lateral stability**
  - Rule 18
  - Shear wall

- **Primary lateral stability**
  - Rule 19
  - None
  (Applicable only with masonry columns)

- **Secondary lateral stability**
  - Rule 20
  - Diagonal brace

- **Secondary lateral stability**
  - Rule 21
  - Cross braces

- **Secondary lateral stability**
  - Rule 22
  - Knee braces

- **Secondary lateral stability**
  - Rule 23
  - Shear wall

- **Secondary lateral stability**
  - Rule 24
  - None
  (Applicable only with masonry columns)
SUBSTITUTION OF PRIMARY ROOF STRUCTURE TERMINALS:

![Diagram of pitched and bowed structures]

SUBSTITUTION OF COLUMN TERMINALS:

![Diagram of column and masonry/timber columns]

Figure 6: Rules of the primitive hut grammar. (Variables are in italics.)

Thus by drawing terminal vocabulary elements we specify their forms, and by defining rules of usage we specify their functions. (Similarly, a dictionary specifies the forms of words, while grammatical rules describe their functions in sentences—how to use them as nouns, verbs, adjectives and so on.) Of course you can argue with our rules: there may be ways of using these elements to make primitive huts in ways that we have not considered, and you may not like some of the usages that we allow. The rules simply represent some empirical knowledge of how things work, and like all empirical knowledge it may be incomplete and imperfect. If you know better, then you can change the rules.

The starting shape S to which these rules are applied is a simple rectangular box, as illustrated in figure 7. The length, width, and height of this box can be varied to specify huts of different sizes.

![Diagram of a box]

Figure 7: The starting shape of the primitive hut grammar

Under the rules of the primitive hut grammar, certain elements can be used in different roles. A beam, for example, can be used as either a primary structural element or as a secondary structural element. On the other hand, some contexts allow use of alternative elements in a role: roof support can be provided either by masonry columns or by timber columns, and so on. Thus the rules of usage allow different combinations to emerge: they
specify not just one primitive hut design, but a language of primitive huts. Figure 8 illustrates the derivation of one design in this language by recursive application of the rule system to the starting shape. Different sequences of rule applications are possible, and yield different designs. If we have written our rules correctly, all of these designs should prove functionally satisfactory in the sense that they shelter a rectangular volume, carry roof loads down to the ground, and provide lateral stability.

Figure 8: Derivation of a design in the language.
The rules of the grammar assign a hierarchical structure of elements and subsystems to any design in the language. Figure 9 illustrates this structure (the syntax tree) for the design that was derived in figure 8.

![Syntax tree for the derived design.](image)

**Figure 9**: Syntax tree for the derived design.

### Parametric Variation

All of the terminal and marker vocabulary elements in this language are defined as parametric objects. In other words, their dimensions are variable, and different instances can be specified by assigning different values to the variables. This is necessary if we want to be able to adapt hut designs to different functions and contexts, but it creates some complexities.

The basic problem is that the dimensions of some elements must depend on the dimensions of others if functionality is to be maintained as dimensions are varied. A beam, for example, must span between columns if it is to be able to hold up the roof. If we take the length of the beam as an independent variable, then column spacing must depend on it. Alternatively, we can take the column spacing as an independent variable and make the length of the beam depend on this. In any case, we establish chains of dimensional dependencies when we fit elements and subsystems together to produce hut designs.

The problem of tracing dependencies can be simplified considerably by assuming that dimensional dependencies are always propagated top-down through the syntax tree, so that
lower-level subsystems fit within the envelopes established by higher-level subsystems. (Figure 9 illustrates how variables may appropriately be introduced at various levels in the syntax tree that was shown earlier.)

**Structural Analysis of Alternatives**

In general, the dimensional variables of terminal vocabulary elements are subject to constraints. A timber beam, for example, must be large enough to support the load that it carries, but not so large that it exceeds the dimensions of available timber. Thus, when we attempt to trace a chain of dimensional dependencies and adjust a design accordingly, we may find that a constraint violation results in some terminal.

This problem can be handled by making some reasonable assumptions about loads on the system as a whole, then using standard formulae to check the adequacy of terminal members. Whenever an inadequate terminal member is found, the designer can attempt to adjust its dimensions to achieve adequacy. If this proves impossible within the space available, then the designer can back up the abstraction hierarchy, and attempt to adjust the higher-level subsystem so that the problem in the terminal becomes solvable. If this also fails, then the designer can back up again, and so on to the top of the tree.

Notice how formulae and grammar interact. First, the grammar establishes the structural roles of elements and hence the types of analyses that are relevant—whether, for example, a particular rectangular element should be analyzed as a column or a beam. Second, the grammar often can be (and in this case is) written to assure that no unanalyzable conditions develop: every design in the language is a well-formed object of analysis with the available formulae. (This is analogous to the property of a computer programming language that every syntactically correct command is processable, or the property of a spoken language that every grammatical declarative sentence is meaningful—either true or false.)

**The Topdown Primitive Hut System**

To explore the possibilities of combining a shape grammar and structural analysis of alternatives, the Topdown system (Liggett, Mitchell, and Tan 1990, Mitchell, Liggett, and Tan 1990) was used to implement the primitive hut grammar (as described above) for putting together timber and masonry columns, timber beams and trusses, and various kinds of lateral stiffeners in configurations such that shelter is provided to a rectangular area, vertical loads are transferred to the ground, and there is stability under lateral loading. (In other words, the generator encodes knowledge of how to satisfy these requirements, together with considerable knowledge of how to resolve the detailing problems that arise when elements are put together in different combinations.) Structural analysis calculations were integrated so that member sizes could automatically be tested for adequacy. (That is, knowledge of how to size structural members was built into the tester.)

The user of Topdown sees three windows on the screen (figure 10). The peek window always displays the current state of the design. The poke window depicts the design in terms of intermediate abstractions at any chosen level in the abstraction hierarchy—from a rectangular box (the starting shape) to the current configuration of terminal elements. The poke window also shows dimension line. Finally, the message window displays textual suggestions, reminders, critical comments, results of analyses, and so on.
All interaction takes place via the poke window. Clicking on an intermediate abstraction in the poke window brings up a dialogue box displaying possible substitutions—elements that can play the role defined by the intermediate abstraction (figure 11). Selecting one of these substitutions then results in performance of the substitution and depiction of the result in the peek window. Clicking on a dimension line brings up a slider which can be manipulated to vary the corresponding dimension (figure 12).

Figure 10: The Topdown interface.  
Figure 11: Topdown substitution dialog.  
Figure 12: Topdown parametric variation dialog.

The user of this system is able to select from among alternative elements and subsystems to use in various contexts, and to vary dimensions at any level within a hierarchy—from individual member sizes up to overall dimensions of the entire system. All operations are in realtime or close to it in versions implemented on Macintosh II and IBM PS/2 machines.
Realtime Visual Feedback

Load and strength calculations are carried out on the fly, in realtime, as dimensions are chosen and varied. (Reasonable simplifying assumptions, and use of elementary beam, truss, and column formulae, make this feasible even on a Macintosh or PS/2 (Pollalis 1990). Implementation on a more powerful workstation would make more sophisticated analysis feasible, but there would probably not be a lot of advantage in this.) Feedback of evaluation results is simple, direct, and graphic: adequately sized members are shown in green, inadequately or prohibitively sized members are shown in red, and members on the threshold of inadequacy or prohibitiveness are shown in yellow.

Red conditions can usually be eliminated by appropriately resizing elements or subsystems. For example, a red primary beam might be made green either by increasing its depth or by reducing the overall dimension of the hut and hence its span. Sometimes a green condition can be achieved by choosing and substituting a more appropriate element or subsystem for a particular role—a truss instead of a beam for a long span, for example.

Thus the designer's objective is to find a configuration that responds to whatever non-structural objectives are relevant, while keeping all the members green. (The system is, then, a demonstration of the Logic of Architecture thesis that design can be seen as search within a language to discover alternatives that satisfy specified predicates. The alternatives that satisfy the predicates have no red members.) Users typically find that the system allows them to explore a very wide range of alternatives in just a few minutes, and to converge quickly on one that is both technically adequate and aesthetically interesting (so refuting Kermit the Frog's well-known assertion that "It isn't easy being green.").

Renderings and Animations

The result of a successful Topdown session is a fully detailed three-dimensional model. This is represented internally, in extremely compact fashion, as a set of parameter values. Simple procedures can then expand these values into DXF, IGES, or other types of files for export to geometric modelers, renderers, or animators, so establishing a particular close linkage between exploration of construction possibilities and consideration of corresponding visual and spatial qualities.

Applications in Teaching

This system has been used by more than one hundred students in large introductory structures and computer-aided design classes at the Harvard Graduate School of Design. Most students grasped the idea of the system instantly, were able to begin working with it right away, and could produce satisfactory designs after just a few minutes of work. Thereafter, they were able to explore alternatives and variations freely—so developing an understanding of the interrelationships between element, arrangement, material, and dimension choices, and gaining a sense of the domain of possibilities structured by the grammar.

Our experience suggests that the approach may have wide have wide applicability in teaching construction, structures, and environmental systems. It has the advantage of allowing students to learn in a self-directed, exploratory way, by considering a wide variety of interesting and diverse examples. It combines some of the best features of the design studio (rich and complex problems and encouragement of formal exploration, but little analysis)
with those of technical classes (careful analysis, but often simplified examples, and little opportunity for wide-ranging exploration of alternatives), while mitigating the disadvantages of both.

**Reflecting on Rules**

It is sometimes objected that this type of knowledge-based system reduces design to rule-following. But this objection only has force if the grammar is closed and rigid, and if it is used unreflectively. A more interesting view is that a grammar expresses a set of working assumptions about how to put together a design with the properties that you want. These assumptions are provisional, and subject to change as you discover that certain choices and arrangements cause difficulties, that elements can usefully be arranged in ways that you had not anticipated, that the given rules do not provide sufficient adaptability to satisfy particular requirements of context and function, and so on. A grammar should be used reflectively, with constant attention to whether it is helping to solve the problem at hand, and it should be amended continually. In this way, a grammar becomes an evolving repository of explicit, machine-processable design knowledge.

One of the main problems in developing rules is that you must identify all the conditions requiring attention that can emerge in the derivation of a design, and specify rules for dealing with these conditions. Otherwise, if unexpected conditions emerge, derivations may not terminate. Thus, for example, you must solve the corner detailing problem in a general way—for all the different combinations of timber and masonry columns, beams and trusses as primary structure, beams and trusses as secondary structure, and different choices of lateral stability systems, that alternative derivations may produce. Consequently, rule systems must often be modified to respond to conditions that were not initially anticipated.

A second problem is that you must choose an appropriate hierarchy of intermediate abstractions, and the best way to do this may be far from obvious. Instead of parsing the hut as we have done, for example, you might parse it into two frames with a roof structure slung in between, as illustrated in figure 13. This structures a designer's thinking in quite a different way. Whenever you choose one particular way of parsing a design, you illuminate some issues and open up certain possibilities at the expense of others. In development of the primitive hut grammar, many alternative ways of parsing were considered. All had both advantages and disadvantages.

![Figure 13: Alternative support structures.](image-url)
And a third problem is that you must try to specify a language that is sufficiently extensive to allow response to different contexts and functional requirements, and provides a designer with adequate expressive power, but does not include a lot of irrelevant possibilities and architecturally meaningless variations. Our primitive hut grammar certainly errs on the side of restrictiveness. Everything that it produces is functionally sound, but it does not produce enough to establish an extraordinarily interesting domain for design exploration.

All of these issues provoked extensive and interesting discussions among the authors as the primitive hut grammar was under development. We explored the advantages and disadvantages of several different parsing schemes, and devoted a great deal of effort to finding general solutions to arrangement and detailing problems. A particular point of contention was the priority of spatial versus tectonic decisions. Should you first choose a general spatial arrangement then explore feasible tectonic realizations, or should you first choose a tectonic vocabulary then explore the spatial possibilities that it establishes? Such questions probably have no general answers: it seems that different designers take characteristically different attitudes toward them, and so produce characteristically different responses to similar problems.

**Limitations of the Approach**

The Topdown system has certain significant limitations. Most importantly, it imposes a predefined, fixed, hierarchical decomposition on designs. Thus it lacks the capacity to recognize and transform emergent shapes, as a fully general shape grammar system would have. However, there is considerable practical justification for treating terminal vocabulary elements as indivisible atoms, since they are both real physical components and natural units of structural analysis.

**Conclusion**

Implementation of the primitive hut grammar in Topdown has demonstrated the feasibility of capturing sufficient practical knowledge in a grammar to support very efficient solution of a fairly elementary, but certainly non-trivial, three-dimensional architectural design problem. In the implementation and testing process, this grammar evolved through many versions: in its current state it expresses a much more complete and refined theory of how to design a "primitive hut" than it did in earlier versions. Others might take it even further—so enacting a cycle of theory-building, criticism, and modification that produces an accumulation of practical design knowledge in accessible, computer-processable form.
References


