

Terry Knight and
George Stiny

Authors' address
Department of Architecture
Massachusetts Institute of Technology
Cambridge, MA, USA
tknight@MIT.EDU
stiny@MIT.EDU

Classical and non-classical computation

Computers have come to play an important role in architectural practice. Nonetheless, the promise of computation as a creative partner in practice, and a means to better understand and support the design process has yet to be realized. This article considers aspects of computation, and alternative ways that these have been approached in order to make computation useful in architecture and other areas of spatial design.

Computation and creativity should be mutually supportive in the design process. For this to be possible, computation must be understood in a broad sense, encompassing both the narrow, digital kind of computation and the more general kind of computation in which objects may only have digital approximations. It is this broad approach that we have adopted.

The aspects of computation considered here are *representation* and *process*. Representation has to do with the way the objects in a computation are described. Process has to do with the computation itself and the rules that are used to carry it out. Both of these aspects have been approached in alternative ways – either *classically* or *non-classically*. These divisions are not hard and fast ones, but soft ones that are blurred at the boundaries.

We apply the classical/non-classical distinction because it provides a useful taxonomy both to see what is going on in computation, and to suggest areas for future investigation of importance in design. Certainly, as it is worked out below, the classical/non-classical split in process is already widely appreciated, even if it is not really discrete but more of a continuum. In representation, though, the split is more profound, even if much less appreciated. We are all used to computation in the classical mode – this is what is taught in school – and it takes some shifting of gears to see how computation can be more successfully used in design when representation is non-classical. For this reason, our main emphasis will be on non-classical representation. Any imbalance in presentation,

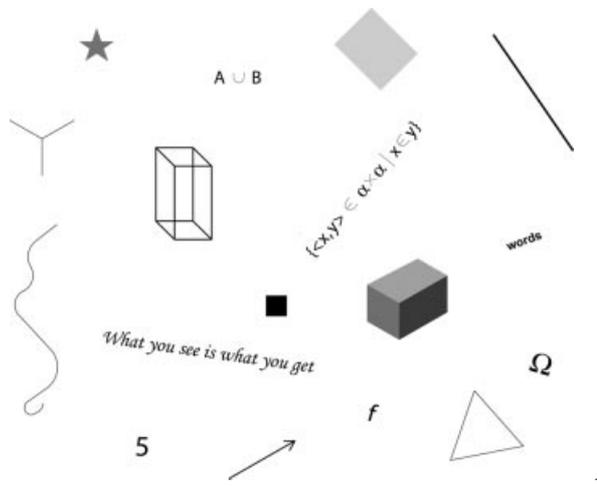
however, is not to reflect importance. All of the categories identified in our taxonomy provide valuable insights in design and computation.

Representation

Representation is usually divided into the verbal kind and the visual kind. The verbal kind is logical and scientific. Verbal representation is the very stuff of computation in the way most people think about it. It depends on a fixed set of primitives, and careful definitions that say how primitives are combined to do meaningful things and to make meaningful objects. Representation that is verbal is *classical* representation.

By contrast, visual representation is traditionally the stuff of creativity and the arts. It is characterized by its lack of primitives, and sometimes by a corresponding vagueness in presentation that leaves open exactly what is there. Visual representation is *non-classical*, at least from the computational point of view. However, this verbal/visual division is not categorical. In the end, both kinds of representations are interrelated.

Figure 1 shows an assortment of examples of classical and non-classical representations. The classical representations – that is, the verbal ones – are familiar to all of us. They are the kinds of things we learn to use in school. They include numbers, words, and symbols. They might include points and lines that are individually distinguished and invariant when they are combined, things like graphs and arrows. Then there are the non-classical representations – that is, the visual ones. These



- 1 Assorted classical and non-classical representations. Verbal and numerical ones are familiar – and classical. Visual ones are non-classical
- 2 A non-classical phenomenon explored
 - a This shape is difficult to describe
 - b ... it can be described as two squares, one in the other or ...
 - c ... as four triangles ...
 - d ... or two envelopes or ...
 - e ... two Ls and two overlapping Us and ...
 - f ... it can be divided arbitrarily

usually involve shapes made out of lines and planes and solids, things like boxes, curves, and triangles. They might come in different colours, and have different textures or other qualities. They can be seen in a variety of ways without being right or wrong.

Consider a classical representation, the sentence: 'I wish I were lying on a beach in Maui'. This is an example of the kind of representation used to write this paper. It is the kind of representation we use all the time to communicate to each other and to computers, and to conduct most of the affairs of the day. The sentence is divided into units – words – that are divided into smaller primitives – letters. And as will be shown shortly, the words can be composed into larger constituents, and these into still larger ones to get at the meaning of the whole sentence.

Despite its ubiquity, classical representation does not go unchallenged. Fifty years ago, Susan Langer (1982) argued for the advantages of presentational or non-classical representation over discursive or classical representation in the arts. And today, there are even reasons to believe that the sciences could profit from a non-classical approach. Hilary Putnam (1987), a philosopher of science, finds this possibility striking and convincing. He writes:

Since the end of the nineteenth century, science itself has begun to take on a 'non-classical' – that is, a non-seventeenth-century – appearance ... That there are ways of describing what are (in some way) the 'same facts' which are (in some way) 'equivalent' but also (in some way) 'incompatible' is a strikingly non-classical phenomenon. (p.29)

Now, the best that can be said about a non-classical phenomenon such as the shape in Figure 2a is to point to it. However, it is also possible to turn it into a classical representation by describing it verbally or by giving it a name. A verbal description of the shape would have to be a very complicated one to describe even a few of the many equivalent, possibly incompatible, facts about the shape. For example, the shape can be described as two squares, one inscribed in the other as in Figure 2b. But the shape is much more than this description or any other can convey. In fact, no finite description says everything

there is to say about the shape. The shape is also four triangles [Fig. 2c] and two envelopes, or two blunt arrows, or two pencil points [Fig. 2d]. Indeed, the shape is anything you want it to be that stays in the lines – such as two Ls and two overlapping Us [Fig. 2e]. Or the shape can be divided arbitrarily, in any way whatsoever. For example, it can be divided by drawing a crazy curve through it so that pieces on one side are one part, and pieces on the other side are another part [Fig. 2f].

In sum, representations come in two kinds: the non-classical ones made with things like lines and planes and solids in shapes, and the classical ones made with things like numbers, words, and symbols. Now how does the classical/non-classical distinction work for process?

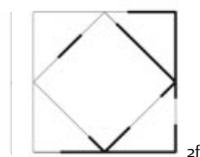
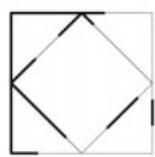
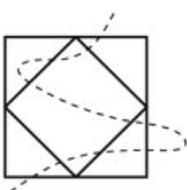
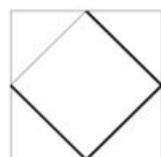
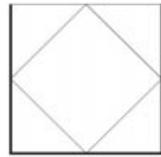
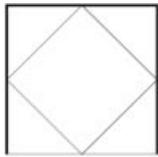
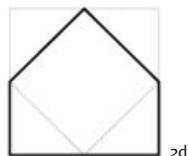
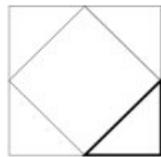
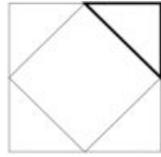
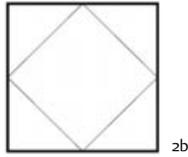
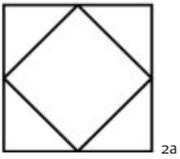
Process

We can think about a computational process in terms of *explanation* and *results*. If a computational process explains what is happening, if it provides the rules of the game and these are meant to be understandable, then the process is classical. But if what is of most interest is the result of a computation, and if there is little concern with understanding how a result is achieved, then the process is a non-classical one. Computation is indifferent to this distinction. Both kinds of process are equally computational.

The classical notion of classical process comes from the linguists – Noam Chomsky and others – and their friends, the cognitive scientists. The founding idea of a grammar in linguistics is its explanatory value. A grammar is judged by what the rules explain about a language and its structure, what the rules say about how sentences work. The MIT *Encyclopedia of Cognitive Science* (1999) is very clear about this:

The motivating idea of generative grammar is that we can gain insight into human language through the construction of explicit grammars. A language is taken to be a collection of structured symbolic expressions, and a generative grammar is simply an explicit theoretical account of one such collection.

The non-classical notion of process, on the other



hand, is exemplified by evolutionary or genetic computation, heuristic search, and optimization techniques. In these techniques, the emphasis shifts from rules to results. And these results are made possible only through the speed and power of current computers. Daniel Hillis (1998), a pioneer of massively parallel computing, and the inventor of the Connection Machine, talks about one of his experiments with evolving computer programs for sorting numbers. He doesn't know how his programs work, but he is absolutely confident that they produce the kinds of results he wants. Indeed, they have done it so many times in the course of their evolution that there is little doubt that they will continue doing it for ever. However, where Hillis is non-classical about process, he is classical about representation. To understand something is to describe it in a hierarchy of primitive parts that need no further explanation. He writes:

One of the interesting things about the sorting programs that evolved in my experiment is that I do not understand how they work. I have carefully examined their instruction sequences, but I do not understand them: I have no simpler explanation of how the programs work than the instruction sequences themselves. It may be that the programs are not understandable – that there is no way to break the operation of the program into a hierarchy of understandable parts. (pp.146–147)

But not everyone is happy about this lack of explanation. John McCarthy (1996), who with Marvin Minsky started the field of artificial intelligence or AI, recognizes the non-classical approach to process. But he finds it all a little odd. He says:

Logic-based AI ... proposes to understand the common sense world ... Anything based on neural nets, for example, hopes that a net can be made to learn human-level capability without the people who design the original net knowing much about the world in which their creation learns. Maybe this will work, but then they may have an intelligent machine and still not understand how it works. This prospect seems to appeal to some people. (pp.144–145)

Ronald Graham (quoted in Horgen, 1993), a mathematician at Bell Labs, finds the prospect of non-classical process possible, but perhaps discouraging. Here, he talks about mathematical proofs, an important kind of computation:

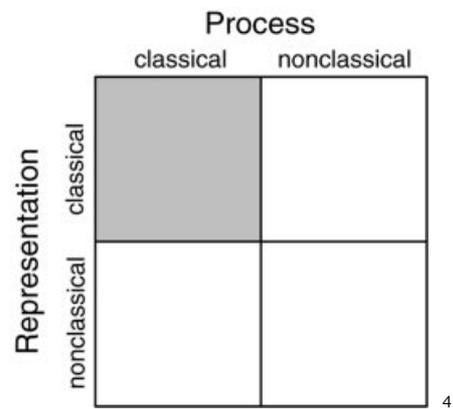
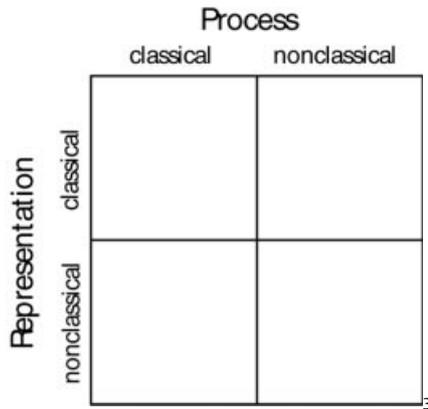
It would be very discouraging if somewhere down the line you could ask a computer if the Riemann hypothesis is correct and it said, 'Yes, it is true, but you won't be able to understand the proof.' (p.103)

In summary, the two aspects of computation laid out above, and the two approaches to each, divide computation into a matrix of four categories [Fig. 3]. These are:

- classical representation and classical process
- classical representation and non-classical process
- non-classical representation and classical process
- non-classical representation and non-classical process

Classical/classical computation

Computation that is classical in both representation and process is classical/classical computation as in



Start symbol: [SENTENCE]

- Rules:**
- [SENTENCE] → [NOUN PHRASE] [VERB PHRASE]
 - [NOUN PHRASE] → [ARTICLE] [NOUN]
 - [VERB PHRASE] → [VERB] [NOUN PHRASE]
 - [ARTICLE] → an
 - [ARTICLE] → the
 - [NOUN] → architect
 - [NOUN] → engineer
 - [VERB] → met
 - [VERB] → sued

Computation:

[SENTENCE] ⇒ [NOUN PHRASE] [VERB PHRASE] ⇒ [ARTICLE] [NOUN] [VERB PHRASE] ⇒ [ARTICLE] [NOUN] [VERB] [NOUN PHRASE] ⇒ [ARTICLE] [NOUN] [VERB] [ARTICLE] [NOUN] ⇒ [ARTICLE] architect [VERB] [ARTICLE] [NOUN] ⇒ . . . ⇒ the architect met an engineer

5a

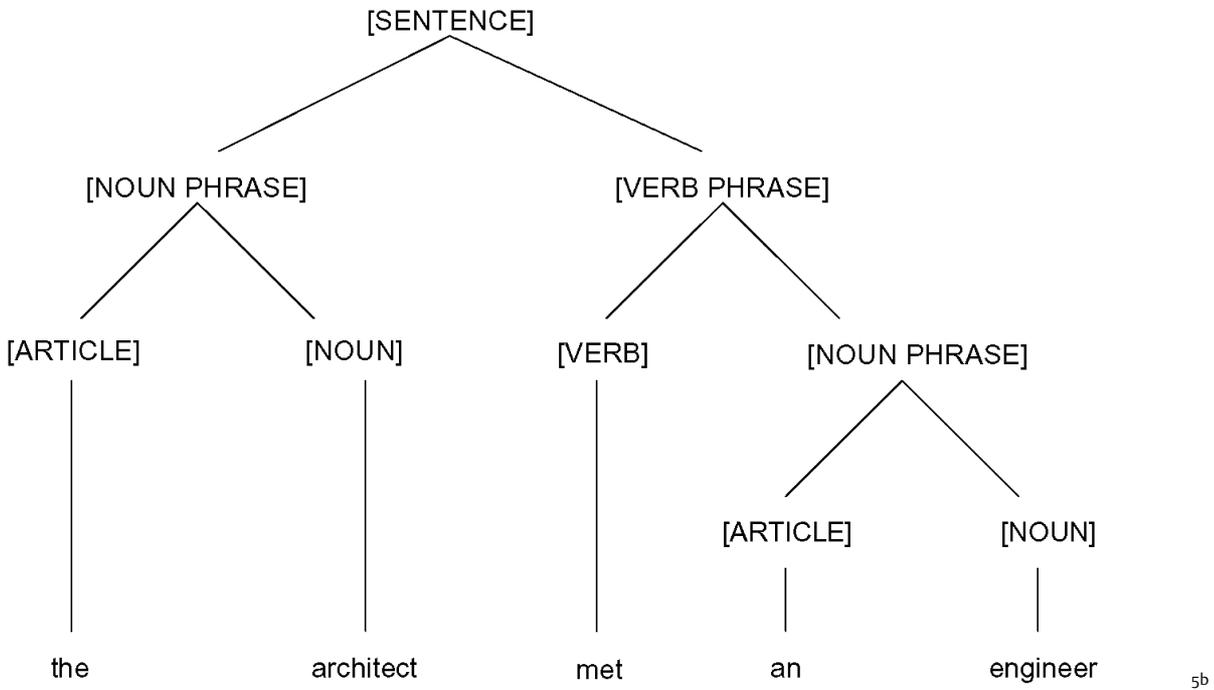
Figure 4. The paradigm for classical/classical computation comes from linguistics and, in particular, from Chomsky’s (1957) phrase structure grammar. A phrase structure grammar is based on a finite number of symbols specified in advance. It has a start symbol and finitely many rules that say how to put symbols together to make sentences, or more accurately, sentences and their structural descriptions. The seven rules shown in Figure 5a are applied recursively, beginning with the start symbol, to generate the sentence ‘The architect met an engineer’. The derivation, or computation, of the sentence provides an explanatory description of the sentence relative to the rules.

Figure 5b is a ‘tree’ (the linguistic concept behind Daniel Hillis’s notion of hierarchy) that recapitulates the application of the rules in the computation. The sentence and its parts are clearly – that is, classically – laid out in the course of applying rules. The representation, too, is classical, because all of the parts are combinations of antecedently specified symbols. There are no other possibilities. This view of computation is incredibly compelling, and almost impossible to shake. It goes back at least to Thomas

Hobbes in the seventeenth century. It is still the dominant view of computation today. In fact, the idea is tried usefully in other domains outside linguistics, including architecture.

Figure 5c is another kind of phrase structure grammar that incorporates architectural components (window elements and pediments) represented classically with symbols and combined in a classical, explanatory process. Symmetrical window patterns are generated by the rules and given a structural description in the course of rule applications. The first rule of the grammar $S \rightarrow aSa$ shows clearly the bilateral symmetry of the patterns produced. The structural description of each pattern generated is a tree in which the pattern and its equally symmetric parts are distinguished. For example, Figure 5d is the tree description for the pattern *aabaa*. This is the kind of description that makes further explanation unnecessary. We know how to get the pattern, how to distinguish its architecturally relevant parts, and how everything hangs together.

Classical computation also extends to non-linguistic computation, for example, cellular

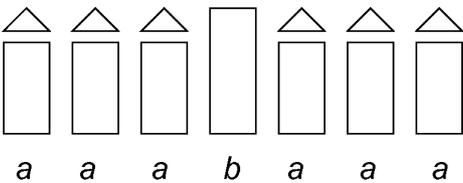


Start symbol: S

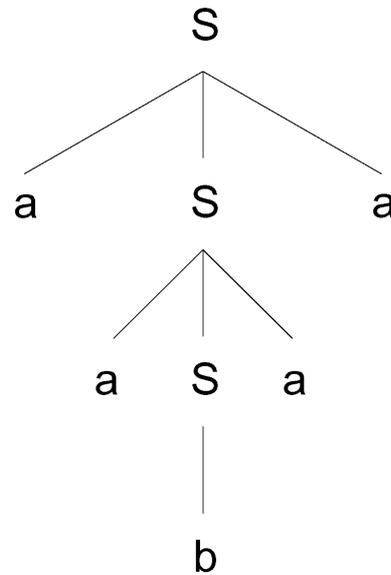
Rules: $S \rightarrow aSa$
 $S \rightarrow b$

Language of bilaterally symmetric strings:

$b, aba, aaba, aaabaaa, \dots$ or $a^n b a^n$



5c



5d

3 Computation can be divided into a matrix of four categories

4 Classical representation and classical process

5 The paradigm for classical/classical computation comes from phrase structure grammar.

a This example is based on simple sentences in English. The sentence derivation provides a description relative

to the rules

b A 'tree' recapitulates the application of the rules in the computation

c Another kind of phrase structure grammar uses architectural components represented classically and combined in a classical process

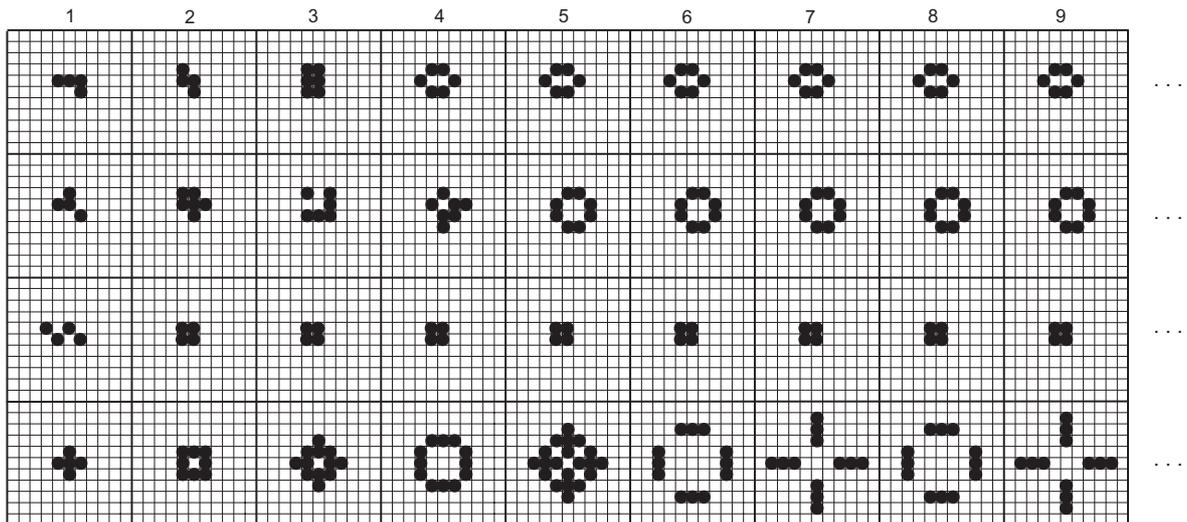
d The structural description for the patterns in 5c is a tree - shown here for the pattern *aaba*

automata. The representation for cellular automata is obviously classical. There are cells - the atomic units of computation - that are arranged in space. These cells talk to each other in their own neighbourhoods, and change in terms of what they are told. This idea of locally interacting primitives goes back to Stanislaw Ulam, one of the great mathematicians of the twentieth century. Today, the best-known example of a cellular automaton is the Game of Life invented by the mathematician John Conway in 1970 (Gardner, 1970) [Fig. 6].

The process of Life is indeed classical. There is survival, death, and birth. There is perhaps a little more - maybe some art - but remember this is a

Game of Life

- Survival If an occupied cell has two or three neighbors, it survives.
- Death If an occupied cell has four or more neighbors, it dies from overcrowding.
If an occupied cell has one or no neighbors, it dies from isolation.
- Birth If an unoccupied cell has exactly three neighbors, it becomes occupied.



The evolution of four starting configurations

6

mathematician's explanation. For any configuration of live cells in the plane, there is a sequence of rule applications that explains how the pattern evolved from a starting configuration. But explanation in this sense may not be all that it is cracked up to be. Just looking at the rules, it is not always possible to tell what will happen in the distant future. Explanation may sometimes entail prediction but need not always. There are different grades of explanation.

Cellular automata are used in the classical/classical sense in other places as well. In work done in the 1980s, Hillier and Hanson (1984) looked at village settlements in southern France. They observed the common characteristics of these villages, in particular the way they form beady rings [Fig. 7a]. Beady ring settlements are encapsulated in rules [Fig. 7b]. There are closed cells and open cells that interact locally just like in Conway's Game of Life. Interactions respond to certain kinds of adjacency requirements. These adjacency requirements are different than those for Life, but good for settlements. When rules are applied locally, global beady ring patterns are generated.

The idea of cellular automata can be relaxed a little, so that cells are not really in a grid, but still behave like interacting units in conversation and in response to local rules. A good example is the recent work of Peter Testa (2001), with what he calls Emergent Design. Emergent Design develops the

notions of artificial life, itself an outgrowth of cellular automata. In one of Testa's studio-directed projects, housing units are aggregated according to rules that specify possible spatial relationships. For example, if a house is three to five units of measure away from another house, the first house can be moved one unit of measure closer to the second [Fig. 8].

Classical/non-classical computation

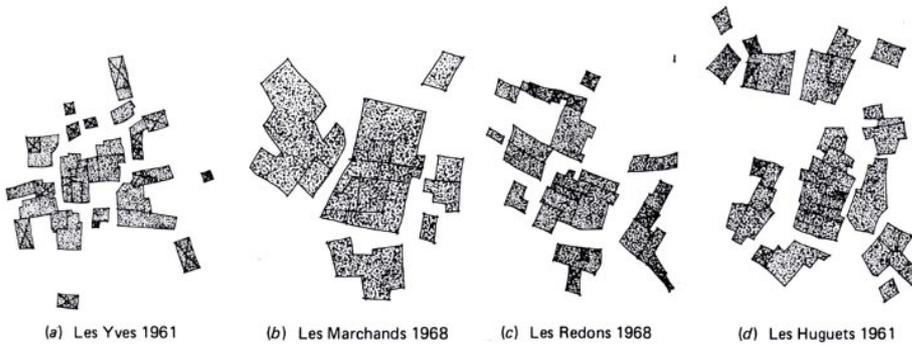
Testa's work leads nicely to computation in which representation is still classical, but in which process, or the act of computing, is non-classical. This is classical/non-classical computation [Fig. 9].

In classical/non-classical computation, explanation is divorced from computation. Typical of non-classical processes are evolutionary or genetic algorithms in which a host of outside criteria not given directly in the rules play a significant role in the outcomes. Another project [Fig. 10] from Testa's Emergent Design work is a good example of the classical/non-classical mode of computation. In this project, a genetic algorithm is used to array spaces in an office building, subject to certain criteria. The reasons for particular arrangements may be obscure in the genetic algorithm. With genetic algorithms in general, rules and constraints may be introduced simply to see what they do. In the end, there may be no explanation, but there may be plenty of effective results.

Some characteristics of **beady ring** settlements in southern France:

the open space is not in the form of a single central space with buildings grouped around it, but is rather like beads on a string: there are wider parts, and narrower parts, but all are linked together direct,

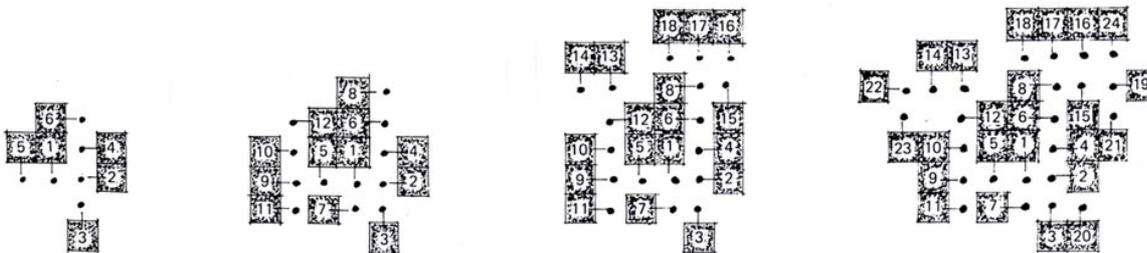
the beady ring is everywhere defined by an inner clump of buildings, and a set of outer clumps, the beady ring being defined between the two.



7a

Let there be two kinds of objects, closed cells with an entrance  and open cells . Join the two together by a full facewise join on the entrance face to form a doublet .

Allow these doublets to aggregate randomly, requiring only that each new object added to the surface joins its open cell full facewise onto at least one other open cell. The location of the closed cell is randomised, one closed cell joining another full facewise, but not vertex to vertex.



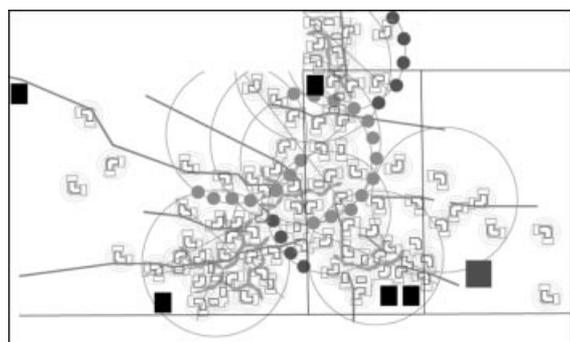
Four stages of a computer-generated beady ring structure.

7b

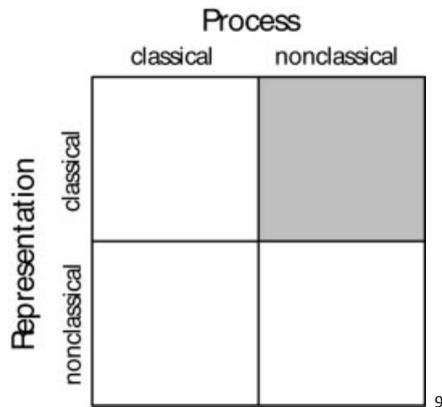
- 6 Classical computation extends to non-linguistic computation as in cellular automata. The Game of Life is a popular example of a cellular automaton
- 7 Cellular automata are used in the classical/classical sense as in this study

- by Hillier and Hanson of village settlements
- a Villages in southern France form 'beady rings'
- b Beady ring settlements can be encapsulated in rules
- 8 The idea of cellular automata can be relaxed as in Peter

Testa's Emergent Design work. Here, housing units are aggregated according to rules. The local organizations of houses are not predictable, regular patterns, but subtle arrangements, subject to the behaviour instilled in each of the houses



8



Non-classical/classical computation

Let's turn now to the inversion of classical/non-classical computation, to computation that uses non-classical representation – representation without units and primitives – in a classical process. This is non-classical/classical computation [Fig. 11].

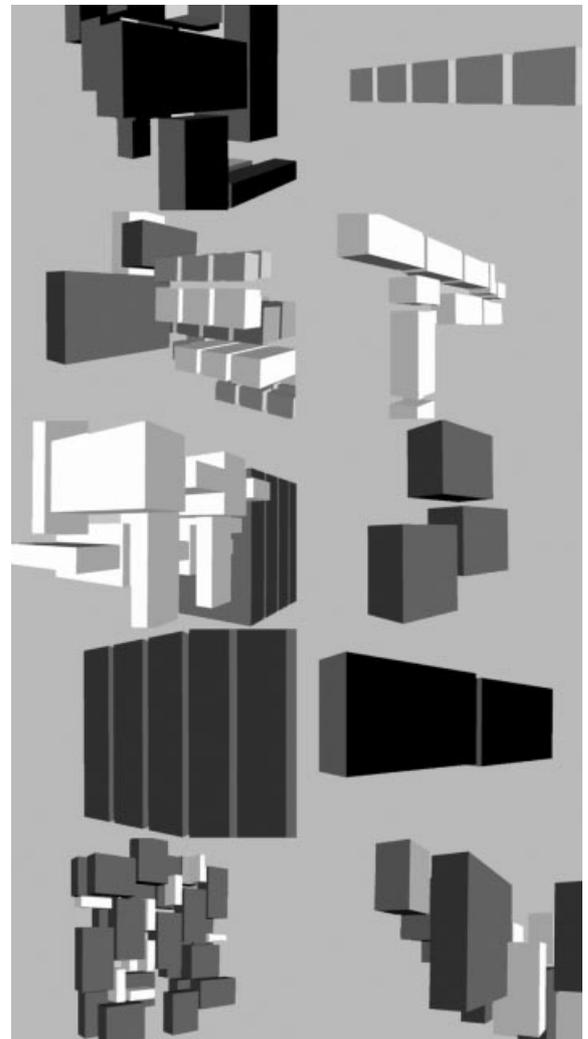
Shape grammars

The best example of this kind of computation is shape grammars. Shape grammars were invented about 25 years ago by two young engineering undergrads at MIT, George Stiny and Jim Gips (1972). Shape grammars were one of the earliest computational systems for doing design directly through computations on shapes, rather than indirectly through computations on text or symbols. This invention was a purely visual computational system for designing rather than a verbal, text-based one.

The representation of visual objects in shape grammars is strictly non-classical. However, the original conception of shape grammars and the many applications of shape grammars that followed are very classical in terms of process. Figure 12a is a simple example from Terry Knight's (1994) book, *Transformations in Design*, of what a shape grammar looks like and how it works. The grammar is based on a Greek cross, a common scheme for 'classical' architectural plans. A Renaissance church design based on a Greek cross is shown. The plan of the church can be decomposed into two 2x1 rectangles. To produce the Greek cross, the shapes are arranged to form the spatial relation shown.

The rectangles and spatial relation are used to define the initial, or starting, shape and the rule of the grammar. The rule says that *if* (the leftside of the rule) a 2x1 rectangle can be found in a design in any orientation or any size, *then* (the rightside of the rule) another 2x1 rectangle can be laid on top to form the Greek cross. The non-classical representation used by the rule allows it to recognize and to operate on shapes not explicitly defined or premeditated by either the author or the user of the grammar. In the computation shown, the rule applies recursively to generate a more elaborate Greek cross. Notice how the rule applies to smaller, 'emergent' rectangles.

The rule can be used to compute many other designs by applying or not applying it to different



10

rectangles in a computation. Figure 12b shows a few other designs that can be computed. All of these designs can be explained and understood in terms of the original Greek cross spatial relation and rule.

More ambitious grammars

The shape grammar above is very elementary. Over the past years, many more ambitious and specialized shape grammars have been developed in response to a variety of design problems. These problems can be divided roughly into two kinds: analysis problems and original design problems. In analysis applications, shape grammars are applied in the most classical sense. They are applied to explain design styles. The first analytic shape grammar was one for Chinese ice-ray designs written by Stiny (1977) [Fig. 13]. This grammar set the standards for the classical, descriptive power of a shape grammar. The four rules of the grammar (shown in abridged form) capture the compositional conventions of ice-ray designs. They even suggest the constructional conventions that Chinese artisans might have used to fabricate the designs as window grilles.

Many more analytic grammars followed from the ice-ray grammar. The first architectural shape grammar was one for Palladian villa plans (Stiny and Mitchell, 1978). Designs generated by the grammar

9 Classical representation and non-classical process

10 Evolutionary or genetic algorithms are typical of classical/non-classical computation as in the work of Peter Testa. Here, spatial arrangements are explored with a genetic algorithm, RhizomeGA (by Doug Rickett). The algorithm starts with a random

population of arrangements and sequentially refines the population using the best members of the population to create a new generation

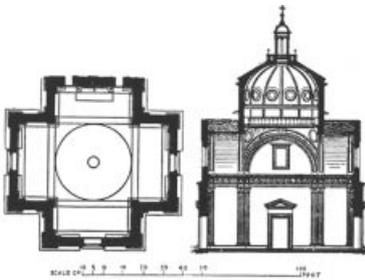
11 Non-classical representation and classical computation

12 Shape grammar representation is non-classical but the original conception and many subsequent

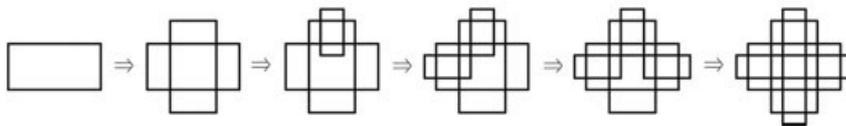
applications are very classical in terms of process
 a An example of what a shape grammar looks like and how it works. The grammar is based on a Renaissance church in the form of a Greek cross
 b The rule in 12a can be used to compute many other designs – all of which can be understood in terms of the original Greek cross

		Process	
		classical	nonclassical
Representation	classical		
	nonclassical		

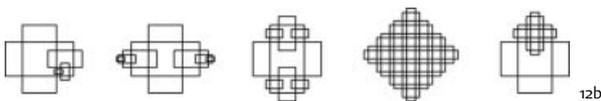
11



Computation:



12a



include original Palladian designs and new, hypothetical ones in Palladio's style [Fig. 14]. The first analytic three-dimensional shape grammar was a grammar for Frank Lloyd Wright's prairie houses (Koning and Eizenberg, 1981). Figure 15 shows some original and hypothetical houses computed by the grammar. Analysis work continues today, and continues to broaden in scope, with important new work coming from engineering and product design.

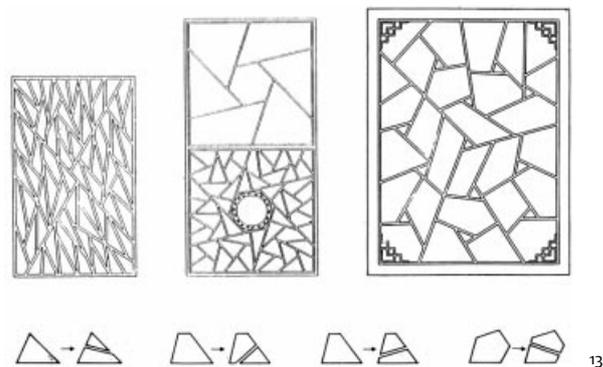
What about original design applications? These originate with George Stiny's (1980) work on abstract, kindergarten grammars using the Froebel blocks [Fig. 16]. Taken into graduate design studios, grammars like this have been the impetus for a remarkable series of design projects. These include projects for different building types, different sites, and different programmes. Figures 17a-f show some examples of design projects, completed in courses taught by Terry Knight at UCLA and MIT beginning in 1990. In all of these projects, the simple, visual rules used to develop designs are meant to explain, in the classical sense, how the designs come about. But is explanation as critical to design problems as it is to analysis problems like the ones shown earlier?

Design work with grammars has also raised questions of design process and design methodology. For example, what kind of methodology can be used to develop or design a grammar itself? To help answer this question, an MIT student recorded a protocol analysis of herself designing a housing complex with grammars. The different stages of grammatical design that were documented are given in Figure 18a. The end product of that design process is shown in Figure 18b.

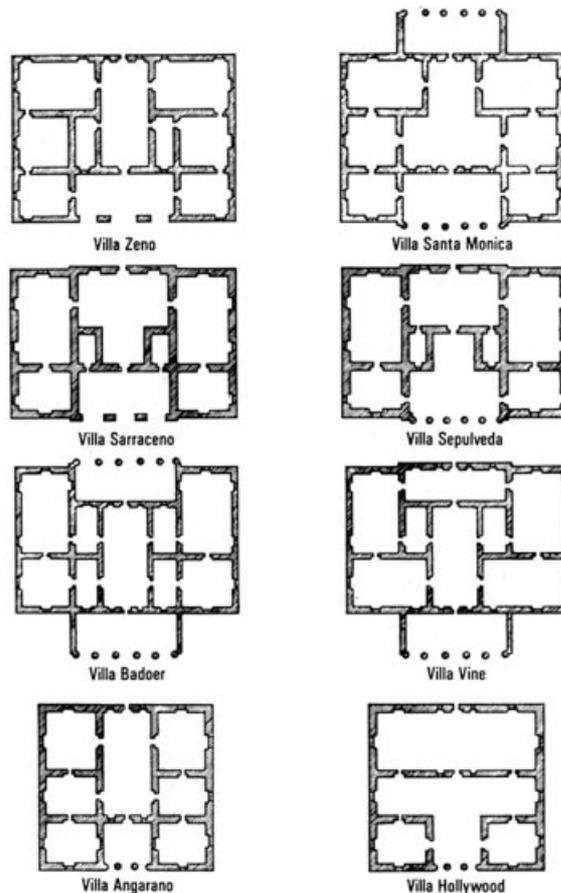
Emergence

What are the properties of shape grammars that have made all of this work possible? The non-classical representation employed by shape grammars is closely tied to a computational term popular in recent years: *emergence*. This is a term used today in many different ways. Often, it refers to the global, unexpected behaviour or outcomes that emerge from decentralized, local rules. Emergence is central to shape grammars. Shape grammars give rise to emergent forms and behaviours in the same way that other computational systems do. But there is more - as the example in Figure 19a shows. The rule in this Figure concatenates squares. After two steps in a computation, the rule produces an emergent shape, an L shape - not unexpected, but emergent nonetheless. This much happens in most computational systems. But with shape grammars we can do more than just recognize emergent forms like this one. We can use emergent forms themselves as input for further computations. We can use them actively. We can do things with, and do things to, emergent forms in an ongoing computation. For example, a new rule can be added to the one just shown that does something with the emergent L, for instance, shifts it diagonally and continues with the computation shown in Figure 19b.

Three kinds of emergence in a shape grammar can be distinguished: *anticipated*, *possible*, and



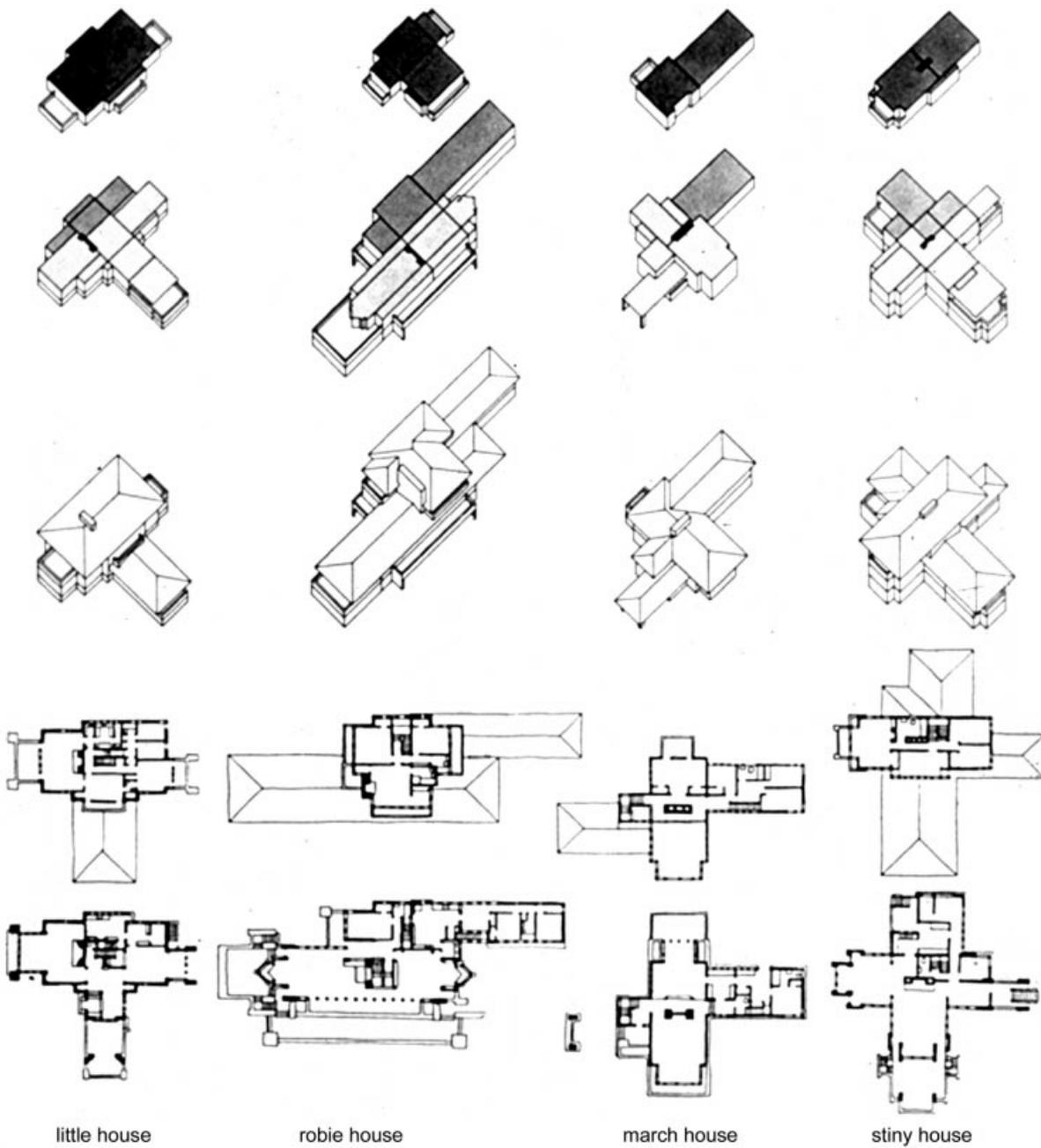
13



14

unanticipated.

With anticipated emergence (an oxymoron!), the author of a grammar writes rules and knows, by looking at the rules, that certain shapes will emerge. In anticipation of these emergent shapes, rules that operate on them are included in the grammar. With possible emergence, the author of a grammar writes rules and thinks that, perhaps, certain shapes might emerge. But this isn't known for sure. Contingency plans are then made for these shapes by writing rules that apply to them just in case they occur. With unanticipated emergence, the author writes rules and computes with the rules. Shapes emerge that were not anticipated or premeditated in any way. In order to compute with these shapes, the grammar needs to be updated with new rules. These rules are not easy to include at the outset because the shapes



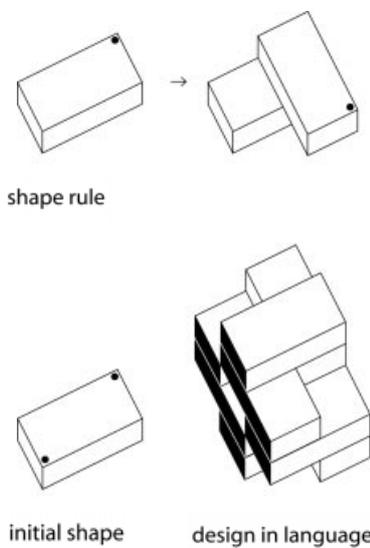
little house

robie house

march house

stiny house

15



shape rule

initial shape

design in language

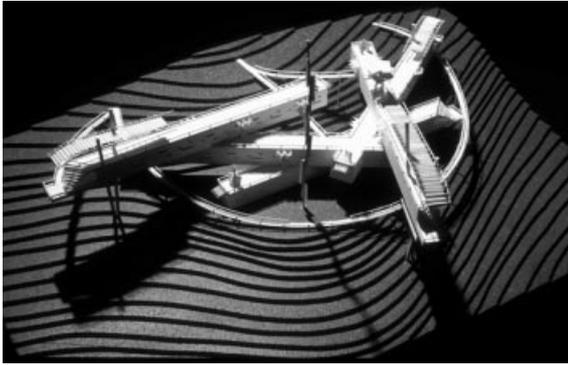
16

13 Many specialized shape grammars have been developed including this one for Chinese ice-ray designs. The grammar suggests the conventions for constructing the designs as window grilles

14 The first architectural shape grammar – for Palladian villa plans, by Stiny and Mitchell, 1978

15 The first analytic three-dimensional shape grammar was for Frank Lloyd Wright's prairie houses, by Koning and Eizenberg, 1981

16 An original shape grammar design application – an abstract grammar using Froebel blocks, by Stiny, 1980

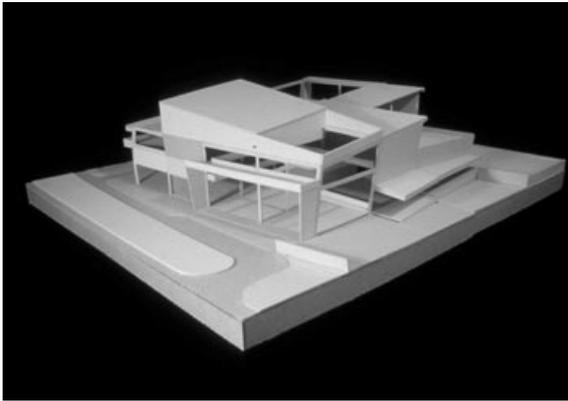


17a

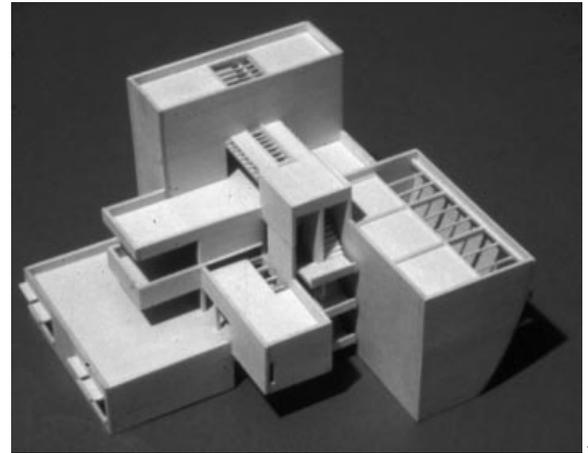
17 Examples of shape grammars applied in undergraduate projects
 a Cultural history museum in Los Angeles (Jin-Ho Park, UCLA, 1993)
 b ... the end product of the grammatical design process
 c Fine arts museum complex in Taipei (Wei-Cheng Chang, UCLA, 1992)
 d Apartment complex in Manhattan (Murat Sanal, UCLA, 1995)

e Cultural history museum in Los Angeles (Jin-Ho Park, UCLA, 1993)
 f Underground memorial to miners (Michael Wilcox, MIT, 2000)
 18 The use of shape grammars raises questions of design process and methodology
 a Student analysis of her process using shape grammars for the design of a complex housing project and ...

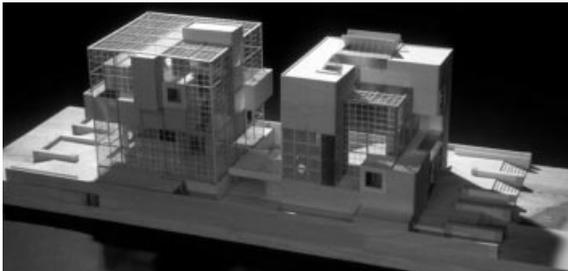
b ... the end product of the grammatical design process
 19 Shape grammars give rise to emergent forms that can be used for input for further computations
 a An example in which an emergent form is produced after two computation steps and, ...
 b ... with a new, emergent rule added, continues to develop



17b



17e



17c



17d

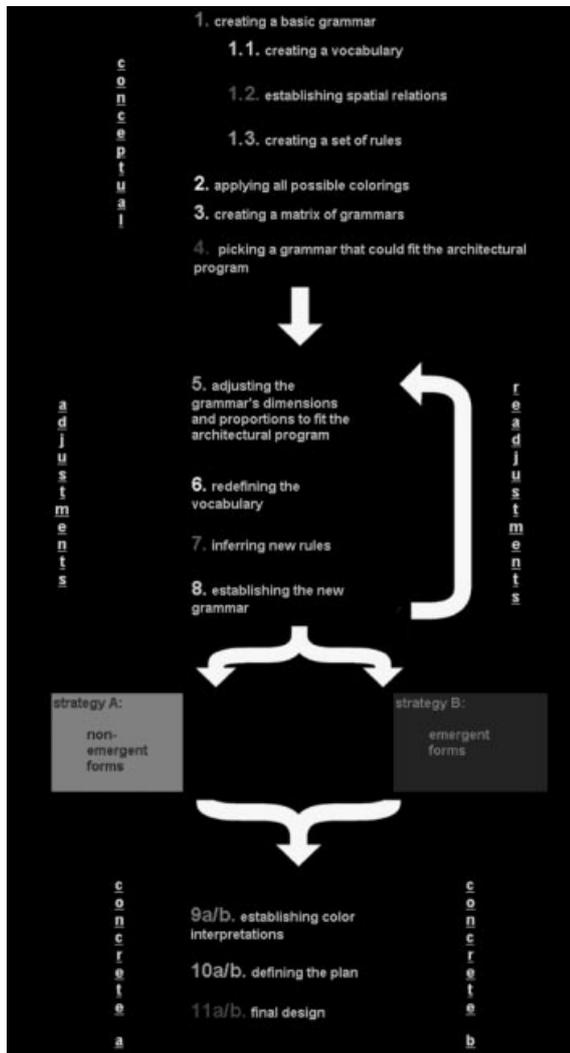


17f

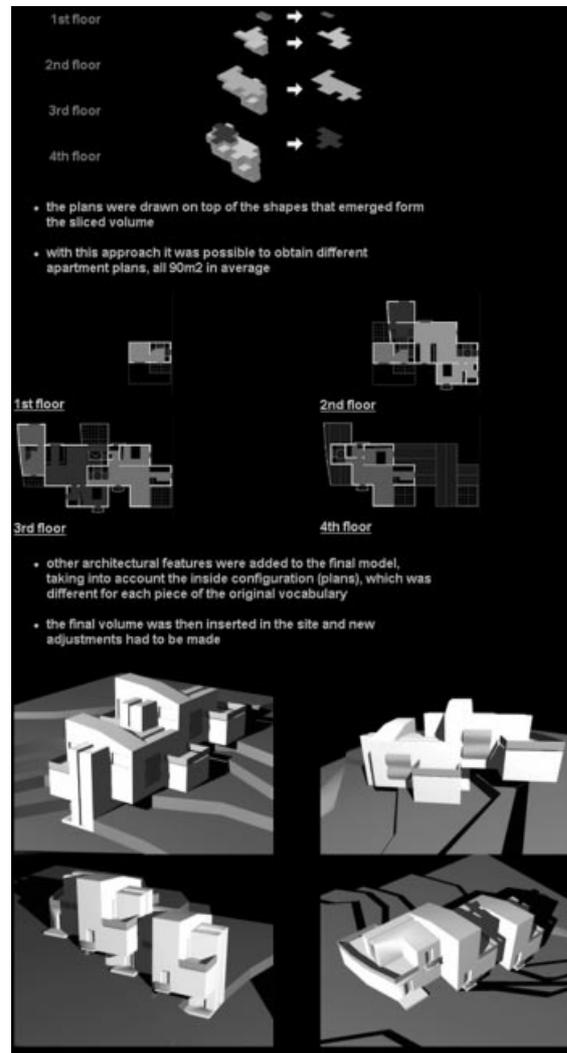
to which they apply were not expected.

Figure 20a is an example of anticipated emergence. The first rule of the grammar concatenates equilateral triangles. Knowing something about equilateral triangles, it is easy to predict that the rule will apply to create an emergent hexagon. In anticipation of emergent hexagons, a rule that picks out or does something to a hexagon is included in the grammar. This is the second rule – a hexagon finding rule. Anticipated emergence such as this is key to analysis applications of shape grammars. In analysis, emergence is necessary but must be anticipated, so that only a limited range of designs is computed.

Figure 20b is an example of possible emergence. The first rule overlaps a triangle on top of an existing triangle. Some experimentation with this rule reveals an emergent fish. It comes up in the fourth design computed. Knowing how this fish emerges from the triangles, it is easy to predict that an infinite number of such fish will emerge in computations. In fact, a much smaller fish emerges near the end of the computation. A rule that picks out emergent fish is included in the grammar. This is the second rule. It is applied in the fourth step and in the last step of the computation. The first, triangle rule generates a kind of triangular grid. A number of other creatures besides fish might emerge from this



18a

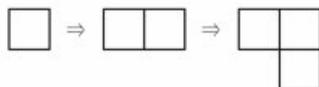


18b

rule:



computation:

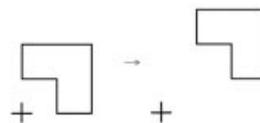


emergent L-shape:

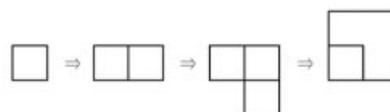


19a

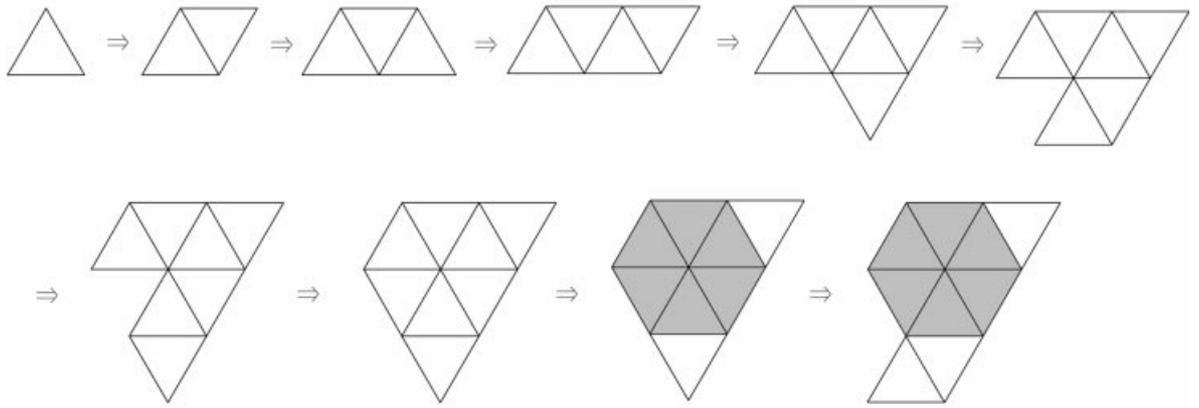
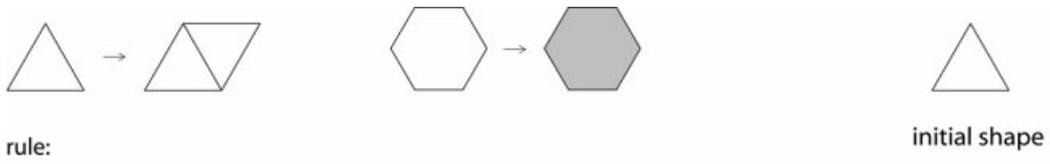
rule:



computation:

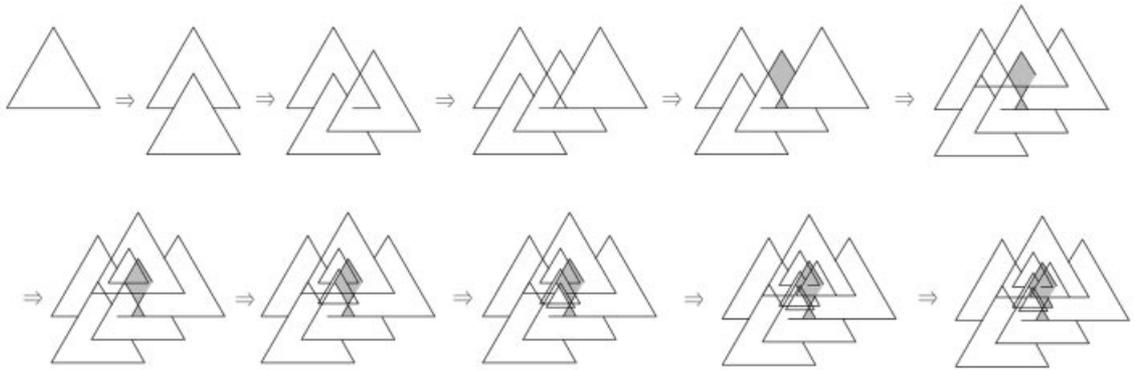
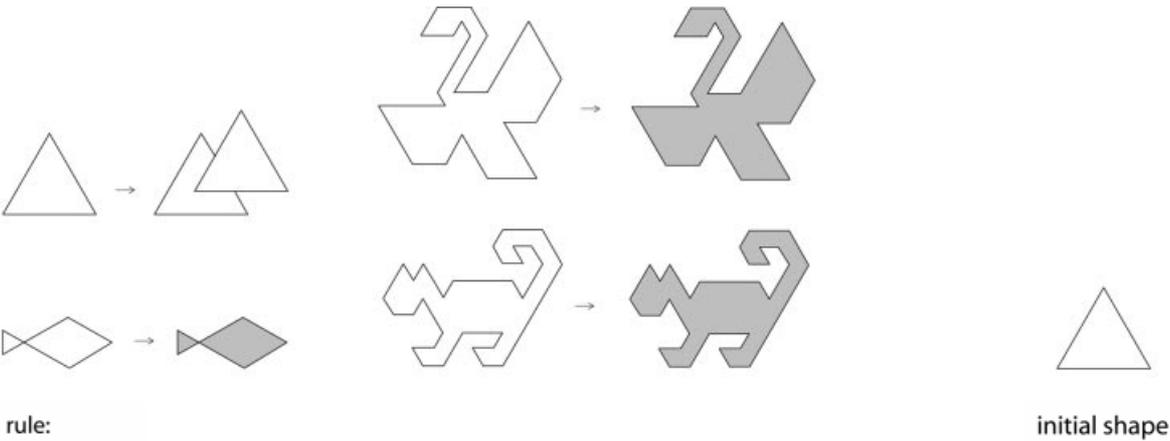


19b



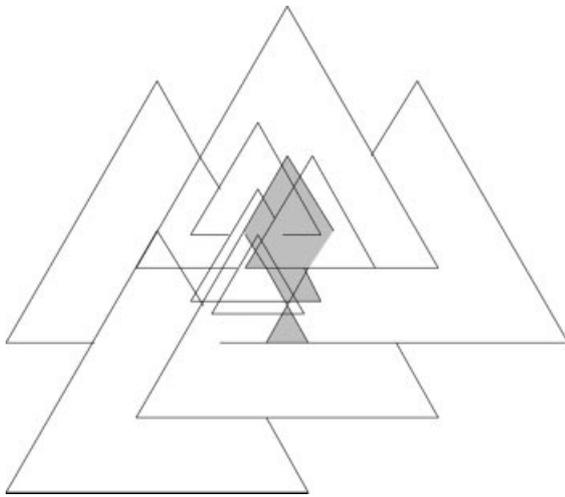
computation:

20a

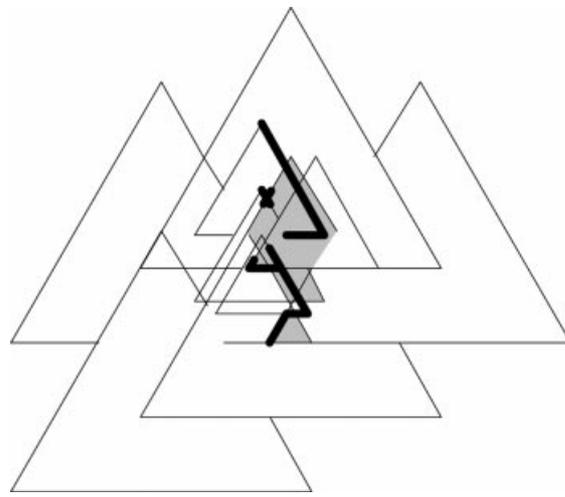


computation:

20b



20c



20e

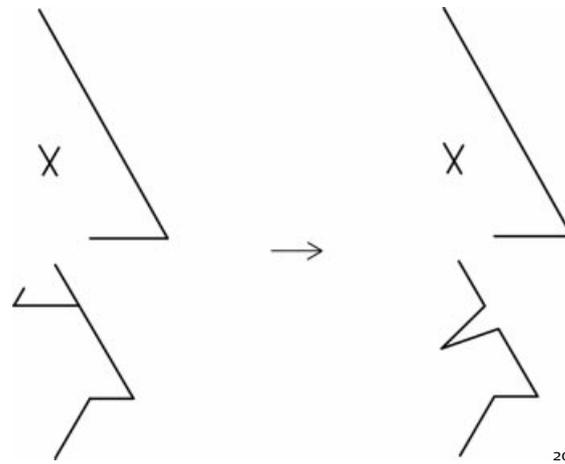


20d

20 With anticipated emergence the grammar author writes the rules and knows that certain shapes will emerge, with possible emergence the author thinks that certain shapes may emerge, and with unanticipated emergence shapes that were not anticipated emerge

a Anticipated emergence: a grammar based on equilateral triangles leads predictably to hexagons
b Possible emergence: a grammar based on overlapping triangles leads, predictably, to a fish, and possibly, to a swan or a cat
c Unanticipated

emergence: the grammar in 20c leads to an unanticipated complex fish
d An emergent rule that acts on the unanticipated fish and ...
e ... an unanticipated face also emerges...
f ... that enables a new, emergent rule to be added



20f

grid. Perhaps a swan, perhaps a cat. Just in case these creatures emerge, two other rules are included in the grammar. But will these rules ever apply?

Figure 20c – a fragment of Figure 20b – is an unanticipated emergence. Two overlapping fish combine in an unexpected way to form a more complex fish, a totally unanticipated fish. The emergent fish is outlined with bold lines. In order to do something with this emergent fish, a new, emergent rule [Fig. 20d] can be added to the grammar.

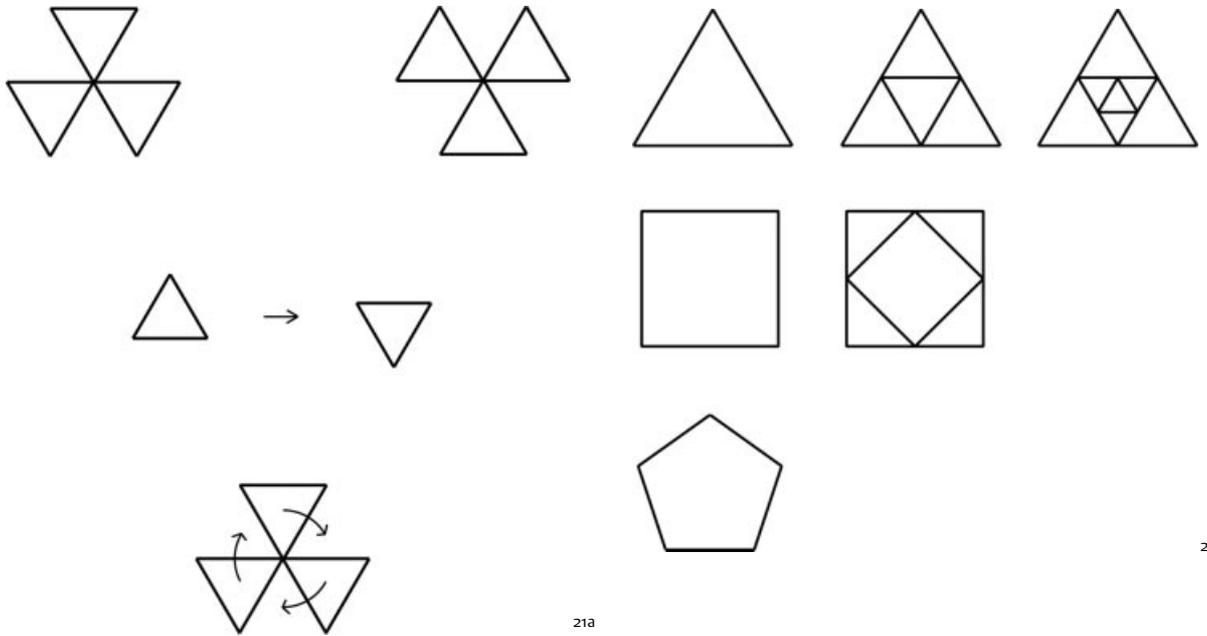
Another look at the design, and other unexpected shapes may show themselves – such as the face (outlined in bold) in Figure 20e. A new rule acting on this new form can be defined in Figure 20f. While perhaps unwelcome in analysis applications, unanticipated emergence like this is key to design applications. This is what happens when we design. The way shape grammars handle emergence may be viewed as a formal counterpart of Donald Schon's (1983) back talk and reflective interaction with the stuff of design. It shows that creative behaviour in design, at least in Schon's sense, is not beyond computation.

What makes emergence possible? The answer is unambiguous – *ambiguity*. Ambiguity is the special property of concrete things like shapes that lets you see them in different ways whenever you like. Ambiguity gets a bad press. One of the pioneers of cognitive science, George Miller (1983), thinks ambiguity is noise:

An interesting question for a theory of semantic information is whether there is any equivalent for the engineer's concept of noise. For example, if a statement can have more than one interpretation and if one meaning is understood by the hearer and another is intended by the speaker, then there is a kind of semantic noise in the communication even though the physical signals might have been transmitted perfectly.
(pp.495–496)

Perhaps Miller is right for cognition, at least if it is rationality, but there may be more to it than this. Ambiguity can be a designer's best friend.

But how does ambiguity work and what can one do with it in computations? Figure 21a is a good example (see Stiny, 2001 for background and more details). The figure on the left of this is rotated about its centre to produce the figure on the right. This is



21c



21b

points	U_{00}	U_{01}	U_{02}	U_{03}
lines		U_{11}	U_{12}	U_{13}
planes			U_{22}	U_{23}
solids				U_{33}

algebras of shapes U_{ij}
 $i \leq j$

simple enough, but try it with a rule that rotates triangles about their centres. It is impossible! The rule keeps the centres of triangles fixed when they are rotated. But these centres rotate when the left-most figure is rotated into the right-most one. How is this done? Calculating with the rule, one can use Figure 21b to find out. The trick is to see that the fourth shape in this computation is both three triangles – the rotated versions of the original triangles – and two triangles as well. With the second description, three triangles can be produced from two that are rotated appropriately. The sequence of triangles, without ever erasing or adding any, is 3 - 3 - 3 - 2 - 2 - 3 - 3 - 3 - 3. This is not magic. It is ambiguity. And this is just what non-classical representations in shape grammars are good for. Nothing is fixed in advance. There are neither units, primitives, nor definitions.

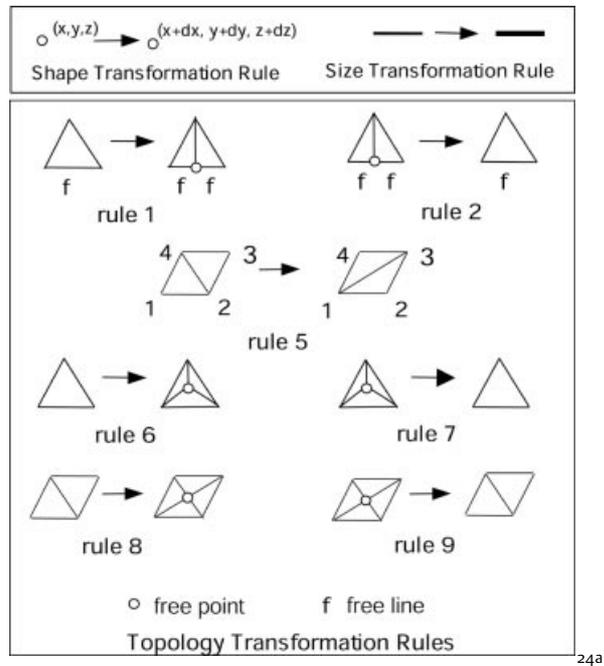
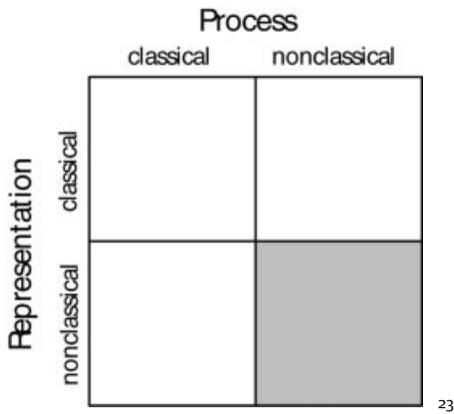
22

But perhaps this is just an isolated computation. How common is ambiguity anyway? It is everywhere, whenever shapes are involved. For example, it is easy to generalize the computation with triangles using the shapes in the potentially infinite table shown in Figure 22c. Ambiguity is anywhere you have a rule to find it, and that's everywhere. And if explanation is desired, there is an algebraic theory about how all of this works. Figure 22 depicts the algebras of shapes for points, lines, planes, and solids (Stiny, 2001). These algebras have a classification in terms of whether or not the indices i and j are 0. Most importantly, the atomic algebras are defined when $i = 0$. They formalize classical representation. There is

21 Emergence is made possible by ambiguity – the property of concrete things like shapes to appear in different ways
 a The top left figure can be rotated around its centre – but not around the centres of the individual triangles. The individual centres themselves rotate when the whole figure is rotated
 b Writing a rule for

21a: the fourth shape is both 3 triangles and 2
 c Ambiguity is everywhere: it is easy to generalize the computation with triangles using the shapes shown

22 Ambiguity is anywhere there is a rule to find it – and there is an algebraic explanation for this. Algebras for points, lines, planes and solids (see Stiny, 2001)



Modeling Structural Design Knowledge in Shape Annealing

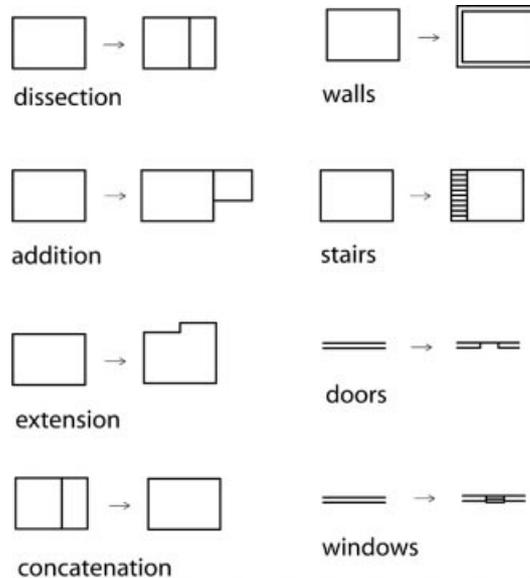
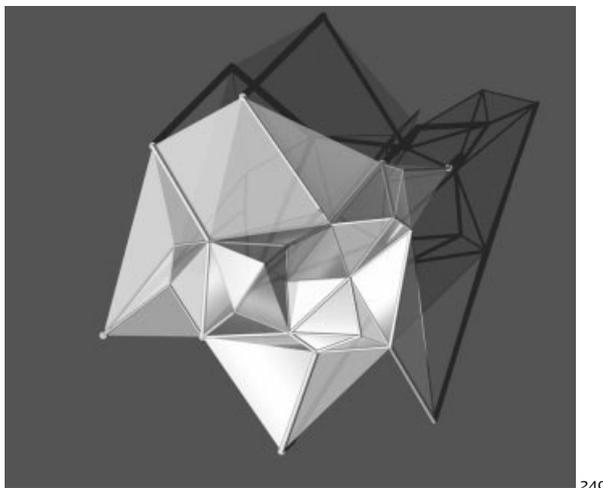
Specifications (Syntax)	Constraints (Semantics)
<ul style="list-style-type: none"> material properties number of supports and locations symmetry joint angles 	<ul style="list-style-type: none"> stress Euler buckling displacement geometric obstacles
Objectives (Semantics)	aesthetics
<ul style="list-style-type: none"> <u>efficiency</u> minimum mass <u>economy</u> minimum number of distinct cross-sections minimum number of distinct lengths <u>utility</u> maximum enclosure space minimum surface area 	<p>uniformity metric = $\sigma(\text{member lengths})$ golden ratio metric =</p> $\sum_{\text{members}} \left \phi - \frac{b}{a} \right + \left \phi - \frac{b}{c} \right + \left \phi - \frac{a}{b} \right $

23 Non-classical process and non-classical computation

24 Eiform (Shea) is an example of non-classical/classical computation a Shape rules used in Eiform generate forms that may or may not be structurally feasible

b Computations with these rules are controlled by shape annealing ... c ... that can lead to impressive, artistic, structurally sound forms d A collection of shape rules that could be developed in an evolutionary process to define grammars for floor plans

24b



no reason to think that verbal and visual devices are not both part of a single mathematical theory. The non-atomic algebras for lines, planes, and solids are defined when i is greater than o . They take care of non-classical representation.

Non-classical/non-classical computation

What happens when the kind of non-classical representation described here is used in a non-classical process? We have non-classical/non-classical computation [Fig. 23].

There are not yet many examples of this. In non-classical/non-classical computation, explicit explanation may be abandoned, but effective results are not. Kristina Shea's (1999) program for generating free form structures, called Eiform, is a good example of non-classical/non-classical computation. Figure 24a shows some shape rules used in Eiform.

The rules are general – so much so that they generate forms that may or may not be structurally feasible. However, computations with these rules are orchestrated by a host of outside criteria. These contribute to making the whole process non-classical. Factors like efficiency, economy, utility, and

aesthetics are put together in an optimization process called shape annealing [Fig. 24b]. The results of computations are no longer explainable by the shape rules or by the computations themselves. Figure 24c shows the kind of impressive, artistic, and structurally sound form that results. And, in a more architectural vein, it is easy to imagine putting together a collection of rules, say floorplan rules [Fig. 24d] that have been useful at one time or another individually, and then seeing what the rules do when they act and evolve together in an evolutionary process. This is exactly what Hillis did in evolving his number sorting programs. There is no claim to explanation, because it is almost impossible to know how the evolved grammars work.

There's more ...

Computation has been described in this paper in terms of representation and process. However, representation and process are just two aspects of computation among a host of many, many others. In many ways, computation is much like shapes. There is always some other way to describe it that may prove insightful. One's work is never done.

References

- Chomsky, N. (1957). *Syntactic Structures*, Mouton, The Hague.
- Gardner, M. (1970). 'Mathematical Games', in *Scientific American* 223:4, pp.120-123.
- Hillier, B. and Hanson, J. (1984). *The Social Logic of Space*, Cambridge University Press, Cambridge.
- Hillis, W. D. (1998). *The Pattern on the Stone*, Basic Books, New York.
- Horgen, J. (1993). 'The Death of Proof', in *Scientific American* 26:9, pp.92-103.
- Knight, T. (1994). *Transformations in Design*, Cambridge University Press, Cambridge, England.
- Koning, H. and Eizenberg, J. (1981). 'The Language of the Prairie: Frank Lloyd Wright's Prairie Houses', in *Environment and Planning B* 8, pp.295-323.
- Langer, S. K. (1982). *Philosophy in a New Key*, Harvard University Press, Cambridge, MA.
- McCarthy, J. (1996). 'Hubert Dreyfus, What Computers Still Can't Do', in *Artificial Intelligence* 80, pp.143-150.
- Miller, G. (1983). 'Information Theory in Psychology', in *The Study of Information: Interdisciplinary Messages*, Fritz Machlup and Una Mansfield, eds., 1983, John Wiley & Sons, New York.
- MIT *Encyclopedia of Cognitive Science*, (1999). MIT Press, Cambridge, MA.
- Putnam, H. (1987). *The Many Faces of Realism*, Open Court, LaSalle, IL.
- Schon, D. (1983). *The Reflective Practitioner*, Basic Books, New York.
- Shea, K. and Cagan, J. (1999). 'Languages and Semantics of Grammatical Discrete Structures', in *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Special Issue on Generative Systems in Design* 13, pp.241-251.
- Stiny, G. (1977). 'Ice-Ray: A Note on the Generation of Chinese Lattice Designs', in *Environment and Planning B* 4, pp.89-98.
- Stiny, G. (1980). 'Kindergarten Grammars: Designing with Froebel's Building Gifts', in *Environment and Planning B* 7, pp.409-462.
- Stiny, G. (2001). 'How to calculate with shapes', in *Formal Engineering Design Synthesis*, E. K. Antonsson and J. Cagan, eds., 2001, Cambridge University Press, New York.
- Stiny, G. and Gips, J. (1972). 'The generative specification of painting and sculpture', in *Information Processing* 71, C. V. Frieman, ed., 1972, North-Holland, Amsterdam.
- Stiny, G. and Mitchell, W. J. (1978). 'The Palladian grammar', in *Environment and Planning B* 5, pp.5-18.
- Testa, P. (2001). 'Emergent Design: a crosscutting research program and design curriculum integrating architecture and artificial intelligence', *Environment and Planning B: Planning and Design* 28(4), pp.481-498.

Illustration credits

arq gratefully acknowledges:
 B. Hillier and J. Hanson, 7a and b
 M. Auer, T. Braude, J. Nakagawa, 8
 Doug Rickett, 10
 G. Stiny, 13, 16
 G. Stiny and W. J. Mitchell, 24
 H. Koning and J. Eizenberg, 15
 Randy Brown, 17a
 Michael Brown, 17b
 Wei-Cheng Chang, 17c
 Murat Sanal, 17d
 Jin-Ho Park, 17e
 Michael Wilcox, 17f
 Gabriela Celani, 18a and b
 K. Shea and J. Cagan, 24a-c

Acknowledgement

This paper is a modified version of a keynote address given at the Greenwich 2000 International Symposium on Digital Creativity, 13-15 January 2000.

Biographies

Terry Knight is an Associate Professor in the Department of Architecture at the Massachusetts Institute of Technology. She conducts research and teaches in the area of design and computation.

George Stiny is a Professor in the Department of Architecture at the Massachusetts Institute of Technology. He has been working in the area of shape computation – especially as it applies to design – for many years.