A NOTE ON GENERATIVE DESIGN TECHNIQUES:

S G G A

A USER-DRIVEN GENETIC ALGORITHM FOR EVOLVING NON-DETERMINISTIC SHAPE GRAMMARS

Benjamin A Loomis
Massachusetts Institute of Technology, Cambridge MA, USA

ABSTRACT

We propose a model for generative design which synthesizes two separate but well-established forms of computational design. It is argued that shape grammars and genetic algorithms address complementary aspects of the generative design problem, and that injecting the theory and research from each field into the other will promote the development of better generative design machines. We also present a prototypical system which synthesizes these strands of research, sketched out within AutoCAD's programming environment.

Note: the included Figures and Tables are mock-ups only

INTRODUCTION

Efforts in computational design can largely be characterized as stemming from one of two positions, which might be referred to as the logical and biological. Broadly speaking, the logical strand of research focuses on systems of production and analysis, such as grammars, syntaxes, and similar rigorously defined languages. The biological strand of research, on the other hand, tends to focus on adaptation and evolution, along with the related processes of complexity, self-organization, and emergence. Among the work in each of these strands of thought about computational design methods, shape grammars and genetic algorithms stand out as two of the most well-established fields of research in their respective domains.

Both first made known in the early seventies, shape grammars and genetic algorithms have each established a solid body of knowledge and community of researchers over the past thirty years. Shape grammars have proven capable of producing complex and meaningful design languages, as exemplified by the Palladian, Prairie, and Queen Anne house grammars, and are theoretically capable of producing any design (Stiny, 1975). Similarly, genetic algorithms are recognized as a powerful and robust problem-solving method, with a wide range of theorems and applications which suggest their optimality for many types of problems (Goldberg, 1989; Mitchell, 1996). However, as the two areas of research stand on opposite poles of the computational design spectrum, the few examples of explicitly combining grammars and search algorithms (Shea, 1998; Rosenman and Gero, 1999) have yet to incorporate the established fields of shape grammars and genetic algorithms. In the field of genetic algorithms and other evolutionary computing techniques, for example, it is often thought that shape grammars are too simplistic or restrictive to make use of the full potential inherent in the evolutionary paradigm (Frazer, 1995). Similarly, it might be argued that the structure of genetic algorithms do not lend themselves to the full range of possibilities inherent in visual calculation, or that the evolutionary approach tends to focus on results to the detriment of understanding.

Yet, as this paper shows, the two fields are in fact complementary, and could benefit from research which explores the points of convergence between them. The most obvious point of convergence lies in the fields' respective strengths with respect to problem spaces. Genetic algorithms are an advanced search mechanism ideal for exploring large and complex problem spaces, though the research to date leaves the structure of the problem spaces themselves less understood than the searching mechanisms. On the other hand, shape grammars provide a rigorous method for defining and constraining design spaces, though they put aside issues of exploring those spaces. Thus, insofar as computational design methods focus on the construction and exploration of design spaces, these two forms of computation address opposite sides of the same coin.

The work we present takes a designer's approach, and is about practical synthesis and gains in understanding rather than strict adherence to theoretical possibilities. Our prototype uses a very simplified and limited shape grammar and genetic algorithm, and makes some naive assumptions about each system in order to test their combination. However, though limited, the theory behind our grammar and genetic algorithm is solid, and the model we outline should be extensible into more practical design contexts. Similarly, the prototype does provide insights into each technique that would not otherwise be obvious, and suggests that each research paradigm could benefit by exploring the gains made so far by the other.

SHAPE GRAMMARS

A shape grammar consists of shapes, labels, shape rules, and an initial shape. Shapes and their labels are the basis for the definition of shape rules. A shape rule has two parts - a left side shape(s) and a right side shape(s), separated by an arrow. A rule states that the shape(s) on the left side is transformed or replaced by the shape(s) on the right side. Given an initial shape, one transforms it using the rules of the grammar to produce a new shape or shapes. A transformation in this generalized sense could include subtracting part of the shape on the left side, adding a new shape to it, dividing it, or so on. Successive use of the grammar's rules on an initial shape produces designs. Though we will now introduce an example of a two-dimensional shape grammar to illustrate the basic mechanisms, shape grammars can be of any dimension. The grammar we will describe in our prototype is a three-dimensional shape grammar very similar to the one here.

Consider Figure 1a, which shows a two-dimensional shape grammar with one rule. The rule adds a rectangle to another in the relationship shown on the right side of the rule. Given the rectangle on the left side as the initial shape, two possible designs generated by the grammar are shown in Figure 1b. Each design is generated by four rule applications. In each step, the rule applies to the rectangle added previously. The difference between the two designs lies in the decision of where to apply the rule, which in turn is predicated by the symmetry of the rectangle (i.e., the number of Euclidean transformations which can be applied to the shape without changing it).

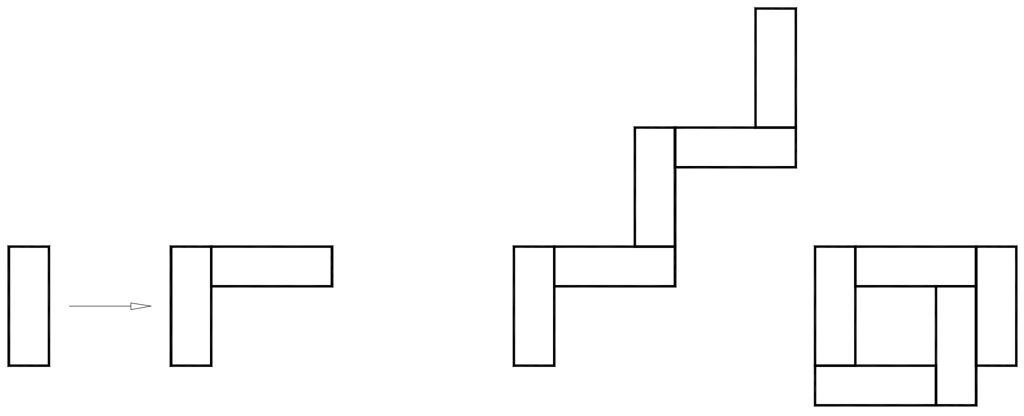

Figure 1. A simple shape grammar that creates chains of rectangles. (a) shape rule (b) designs

Given the spatial relationship defined between the two rectangles shown in the right side of the shape rule, there are four different ways in which the rule can be applied. Figure 2a illustrates the ambiguity that is created by the rule in Figure 1a. In order to create a deterministic set of rules, we can label the rectangles in the shape rules, as shown in Figure 2b. This label in the corner of each rectangle effectively reduces the symmetry of the rectangle to one, and removes all ambiguity about where to apply the next rule. Now a rectangle can only be added to a side which has a dot on the corner, and we must define four shape rules if we want to cover each possible application of the rule. Figure 3 shows the derivations of the two designs in Figure 1b, along with the shape rules applied at each step.

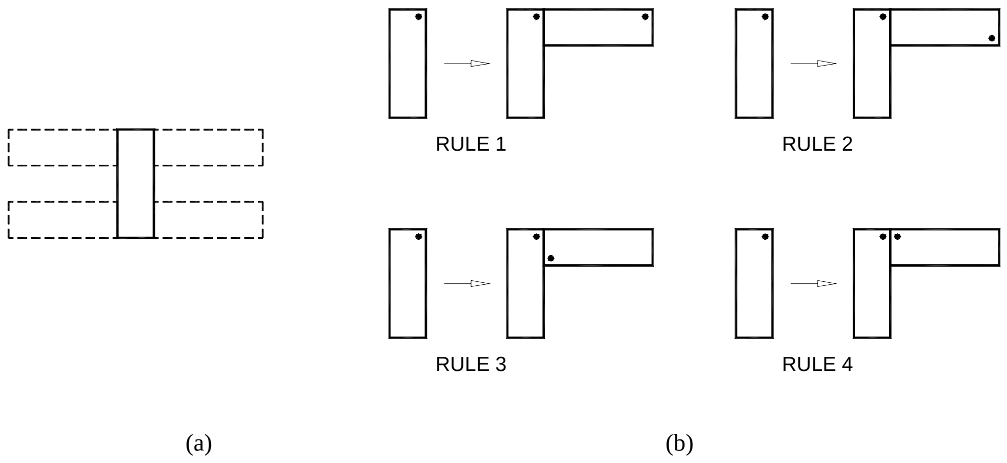

Figure 2. Ambiguity of where to apply a rule. (a) ambiguity of original rule (b) four labeled shape rules

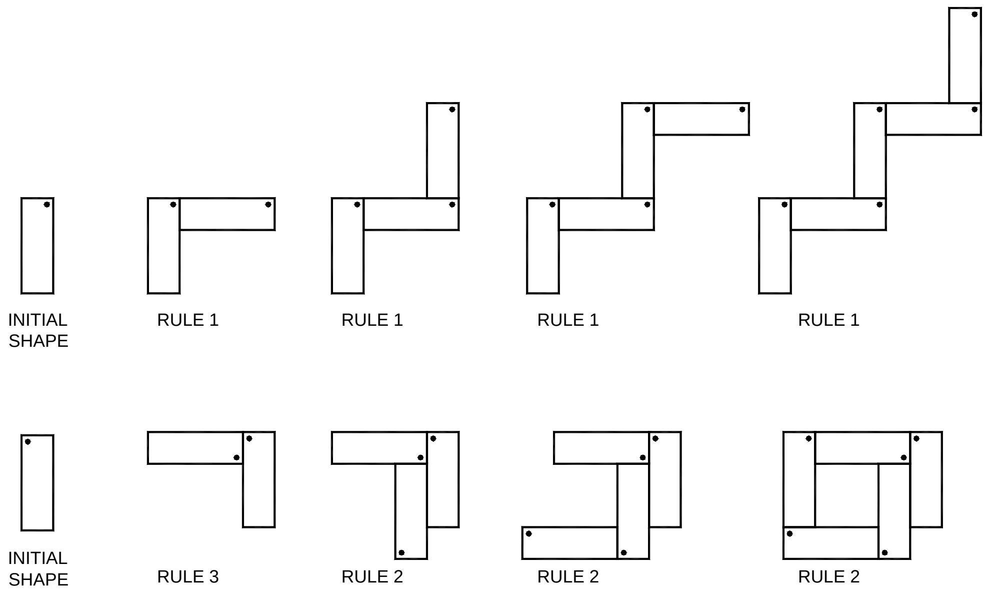

Figure 3. Derivations of designs in Figure 1, using the rules of Figure 2b.

The shape rules in Figure 2b define a shape grammar, and the shape grammar defines a style, which we might call a design space. In this sense, a design space is the set of all possible designs which can be generated by the grammar, and can contain an infinite number of designs. However, the size of the design space generated by application of a finite number of steps can be defined by a simple combinatorial equation:

$$D=L^N$$

where D represents the number of possible designs that can be generated after N steps, and L is the number of label positions (or more generally, rules) which could apply at each step. Thus, there is an exponential explosion in the design space, depending on how many times a rule is applied. The simple four-step designs shown in Figure 1b are two of a possible 256 designs, and if we were to double the number of rule applications to eight, we would now have a space of some 65,000 designs, even with this extremely limited grammar. This exponential increase of possible designs in the shape grammar system might be recognized as analogous to a problem often faced by designers: the rules of thumb used in the design process become progressively more difficult to apply as the problems they are facing becomes more complex.

GENETIC ALGORITHMS

Genetic algorithms are a search method rooted in the mechanisms of natural selection and the notion of evolution. They consist of some type of population (such as candidate solutions to a problem), some method of selection (such as an exogenous fitness calculation), and operators (such as crossover and mutation) which transform one population into another. Genetic algorithms are stochastic in the sense that all three operators use some form of randomness in their process. However, the genetic algorithm's iterative structure of selection, based on survival of the fittest, and crossover, a form of sexual reproduction, transforms the probabilistic nature of the process into a directed and adaptive mechanism which exhibits the characteristics of evolution. The process of transforming one population into another over many generations results in the evolution of the general population towards the goals of the fitness function (Goldberg, 1989, pp1-14).

Clearly, this technique was developed with the advent of fairly powerful computers, as the process would be interminable if done by hand. However, there is an intuitive example one can proverbially do at home with a set of dice, creating a small population and adapting it over just a few generations. Here's the set up: imagine wanting to find seven numbers between 1 and 6, whose sum is as high as possible. Obviously, the answer is seven sixes, which sum to 42, and the problem is trivial. However, approaching the problem using a simple GA (genetic algorithm) will nicely illustrate the basic mechanisms. Note that this "simple GA" is different from the standard "simple GA" (Goldberg; Mitchell) which is based on bits flipping between states of 1 and 0, but we illustrate with the dice example as it is closer to the type of genetic algorithm employed by our prototype.

Genetic algorithms typically start with a set of random answers to a problem, so we will roll seven dice to come up with population members. For this example, let's do it six times - we now have six population members each composed of seven bits, as shown in Table 1. Table 1 also shows their fitness score, which is simply their sum, and the percent of the total population fitness which each member contributes (ie, their relative fitness score, shown in the last column, which is simply the members fitness score divided by the populations total fitness). This relative fitness score is important because we will use it in the selection process to generate a new population.

Member	String							Fitness	Relative
I	2	1	3	1	6	5	3	21	14.9%
II	2	2	1	5	4	2	1	17	12.1%
III	5	4	3	4	1	3	6	26	18.4%
IV	4	5	1	6	2	3	3	24	17.0%
V	5	6	6	2	1	5	5	30	21.3%
VI	5	6	4	1	4	1	2	23	16.3%
							Total	141	100.0%

Table 1. An initial randomly created population

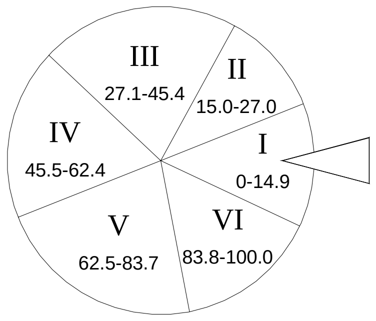

Figure 4. A biased roulette wheel based on Table 1

Before we move to the selection process, though, we should make a quick note of some of the basic terminology in genetic algorithm research. First, the split between genotypes and phenotypes is a critical division made in most GA work. As Holland laid out in the initial text on genetic algorithms, the genotype is the underlying genetic structure of a population member, whereas the phenotype is the member's "amalgam of observed characteristics" (Holland, pp 9-10). Typically, the genotype is what is manipulated by the crossover and mutation mechanisms, while the phenotype is used in the selection process. In the case here, the genotype would be the string of seven numbers, and the phenotype would be the fitness score itself (or, more precisely, the relative fitness score). Further, the genotype in our example consists of several parts analogous to biological systems. The string itself can be referred to as the chromosome, where the individual numbers are each a gene, and the gene's value is an allele. Thus, for example, the fifth gene in member III's chromosome has an allele value of 1. Now that we've established the terminology behind Table 1, we can move to the selection process by which the next generation of solution candidates will be produced.

The wheel in Figure 4 shows a relative fitness range for each member, based on the cumulative relative fitness scores of Table 1. We will spin this wheel to determine the next population. Note that this selection process is similar to a biased roulette wheel, and favors the more fit members of the population, which have a larger slice of the roulette wheel. Thus, each successive generation is more likely to contain the fittest member of the previous population, while the less fit members are slowly weeded out.

Based on this selection method, we spin the roulette wheel six times to select the members who will become the basis for the next population. Doing this, we (hypothetically) arrive at a new population derived from Table 1's members I, V, III, VI, II, and V (see the first part of Table 2). However, we still need to apply the crossover and mutation operators before the second generation is ready for testing. These are the operators which give the genetic algorithm its evolutionary characteristics. Crossover, involves crossing the characteristics of two population members to create new members. To crossover two population members in this example, we will randomly select a site between two numbers in each string (and since there are six such spaces, we can roll a die and use the result to determine the point of crossover). For this population of six, we will roll the die three times, pairing up two chromosomes each time. For example, if the first roll turns up a 3, we count over three spaces in members I and V, cut each at that point, and interchange the rear part of the chromosomes. See Table 2 for the pairings, crossover sites, and resulting members.

Finally, we will apply the mutation operator, which will take a gene's allele and replace it with a new, randomly generated allele. This operator should occur very infrequently, or else the process will become overly randomized, approaching a purely random process when the mutation probability is 50%. In our example, we will provide about a 0.5% chance for mutation of each bit. Thus for every bit in the population, there is a 0.5% chance it will be replaced with a new random bit. This process corresponds to stepping through each bit in each population member and rolling three dice. When three sixes come up (roughly a 0.5% chance), we must roll another die to determine what that bit will become. Here, only the fourth gene in chromosome II is mutated, in this instance to a 1. See Table 3 for the final outcome of all these processes, and the resultant population.

Old	Old String	New	New String
I	2 1 3 | 1 6 5 3	I	2 1 3 2 1 5 5
V	5 6 6 | 2 1 5 5	II	5 6 6 1 6 5 3
III	5 4 3 4 | 1 3 6	III	5 4 3 4 4 1 2
VI	5 6 4 1 | 4 1 2	IV	5 6 4 1 1 3 6
II	2 2 1 5 4 2 | 1	V	2 2 1 5 4 2 5
V	5 5 6 2 1 5 | 5	VI	5 5 6 2 1 5 1

Member	String	Fitness	Relativ
I	2 1 3 2 1 5 5	19	12.9%
II	5 6 6 2 6 5 3	33	22.4%
III	5 4 3 4 4 1 2	23	15.6%
IV	5 6 4 1 1 3 6	26	17.7%
V	2 2 1 5 4 2 5	21	14.3%
VI	5 5 6 2 1 5 1	25	17.0%
	Total	147	100.0%

Table 2. A single-point crossover mechanism **Table 3.** The population's second generation

Member	String	Fitness	Relative
I	5 6 6 2 4 1 2	26	16.1%
II	5 4 3 4 6 5 3	30	18.6%
III	5 6 1 5 4 2 5	28	17.4%
IV	2 2 6 2 6 5 3	26	16.1%
V	5 6 4 1 1 3 1	21	13.0%
VI	5 5 6 2 1 5 6	30	18.6%
	Total	161	100.0%

Member	String	Fitness	Relative
I	5 6 6 2 4 1 2	26	15.8%
II	5 4 3 4 6 5 3	30	18.2%
III	5 4 3 4 1 3 1	21	12.7%
IV	5 6 4 1 6 5 3	30	18.2%
V	5 6 1 5 4 1 2	24	14.5%
VI	5 6 6 2 4 6 5	34	20.6%
	Total	165	100.0%

Table 4. The third generation **Table 5.** The fourth generation

Table 3 shows the second generation of this population, along with their raw and relative fitness scores. Tables 4 and 5 show the third and fourth generations. Note that in each generation the total population fitness has increased slightly, as has the raw fitness of the fittest member. Even this small population illustrates the general behavior of a genetic algorithm, in which generations of solution candidates gradually become more and more fit. The genetic algorithm, through a random search of possible gene combinations, is directed by the fitness function, transforming this blind mechanism into a directed and adaptive process which gradually evolves well-fit individuals, even from completely random starting points. As made clear by the fundamental theorem of genetic algorithms, the Schema Theorem (Goldberg, 1989), short strings of genes which consist of above-average allele combinations will occur exponentially more frequently during subsequent generations. This results in a nearly optimal process for searching through solutions to difficult problems, trading between exploring for new knowledge and the exploitation of knowledge previously gained in the search (Goldberg, 1989). This approach of the adaptive system parallels our intuitive sense of the designer's process of slowly developing more sophisticated and appropriate solutions to a design problem as more time is spent designing.

DESIGN SPACE AND SEARCH HEURISTICS

The complementary nature of the shape grammar formalism and the heuristic of the genetic algorithm should now be evident. Shape grammars define design spaces, though they do not provide a method for exploring them. Conversely, genetic algorithms have been invented to search through problem spaces, although there is as yet no consensus for appropriate methods of representing or defining design spaces.

The Palladian grammar (Stiny and Mitchell, 1978a) developed in the late seventies is a simple illustration of a shape grammar in a practical context. Much more complex than the grammar we illustrated previously, this eight stage, 72-rule grammar produces architectural plans in the style of Palladio. It also demonstrates the formalism's ability to embed design knowledge and constraints into a set of rules and create meaningful design spaces. By following the grammar's rules, one can produce the plans for all of

Palladio's villas as well as new hypothetical plans which fit into the existing corpus stylistically. The Palladian grammar also reveals the potential enormity of a design space: the number of potential Palladian villa partis grows exponentially as the plan size increases. Without allowing for parameterization of dimensions or articulations such as porches and windows, there are 210 distinct Palladian partis based on a 5x3 grid and 25,017 distinct 5x5 plans (Stiny and Mitchell, 1978b). When parameterization of dimensions and various articulations are accounted for, the number of design possibilities quickly approaches infinity. Even with the severe formal restrictions of a classical grammar, the authors only enumerated a small subset of partial design candidates. And while a mechanical process can be used to produce designs ad infinitum, the skill and knowledge of an architect would be required to produce only functional, criteria-meeting designs. In order to automate such a grammar in a practical context, some efficient method for exploring and rating the design possibilities is necessary.

Genetic algorithms are an excellent general method for searching for specific kinds of designs, especially in the context of ill-defined problems, where they have many advantages over other search heuristics. In most design situations, a robust method is critical and the satisfaction of multiple requirements tends to dominate the need to optimize single functions. Much of the early research into genetic algorithms established their robust nature and satisfaction abilities in various basic contexts (Goldberg, 1989), and some recent research has turned to more difficult issues, such as design. Bentley's GADES program illustrates the genetic algorithm's basic capabilities in a design context (Bentley, 1999). As with the Palladian shape grammar, GADES is substantially more sophisticated than the simple GA we presented: it uses a hierarchical crossover mechanism, multi-objective optimization techniques, and parallel populations. Even so, the system exploits the generality of genetic algorithms through modular evaluation capabilities and "blobby" agglomerations of a block primitive, and has been used to evolve simplistic designs for a variety of tasks, such as boat hulls, heat sinks, prisms, and floor plans.

In the process of creating a design-oriented GA, a developer must contend with the issue of design representation. In this respect, shape grammars deserve consideration. At a general level, the shape grammar formalism provides a refined method for representing and manipulating shapes that goes well beyond the translation, rotation, reflection and similar transformations with which we are familiar. At a more sophisticated level, a shape grammar, embedding stylistic and/or functional design constraints, can be established for most any design problem. This provides a level of restraint in the design generation process - an important step in problem-solving of any real complexity, and one which is likely to be critical before a GA would be used in a design situation.

THE SGGA PROTOTYPE

As an initial attempt to fuse these shape grammars and GAs into one system, we implemented a genetic algorithm which searches through a shape grammar design space as directed by user inputs. The prototype, SGGA (for Shape Grammar Genetic Algorithm), is written as a plug-in to AutoCAD using AutoLISP. AutoLISP is a dialect of the LISP programming language which includes some scripting capabilities to access AutoCAD's built-in commands. Working within AutoCAD's established environment allowed us to quickly sketch programs for debugging and testing. And, as we describe below, given the restrictions which we placed on the SG and GA, working within AutoCAD's framework and datastructure caused no substantial limitations.

The SGGA prototype outlined here uses a single shape and single spatial relationship, which together define a distinctive language of possible designs and a plainly structured design space. The simplicity of the grammar at this point was paramount, as we wanted to understand how the mechanisms of the grammar and genetic algorithm would interact. Any further complexity to the grammar itself would impede our ability to rigorously review the genotype and phenotype solutions, as well as the evolutionary mechanisms of the system. It is a difficult and tedious process to review each design to ensure that the system is working as intended. While increasing the grammar complexity and the capabilities of the genetic algorithm's operators is a current concern (see Figure 10 at the end of the paper), it is critical at the early stages of the genetic algorithm design process to understand exactly how the system's mechanisms are working.

As Figure 5a shows, the shape with which we begin is a pillar with a length-width-height ratio of 1:1:3. The single spatial relationship of the grammar is shown in Figure 5b. The shape rule, shown in Figure 5c, adds a pillar to an existing pillar in the spatial relationship specified. Except for the proportions of the pillar, this type of grammar is essentially a kindergarten grammar, as described by Stiny (Stiny, 1980b), and can be thought of as a simple block-stacking procedure. As with the two-dimensional rectangle grammar described earlier (see Figure 1a), this grammar also involves an ambiguity about where to add a new block. In this case, there are 16 places to stack a new block on each pillar and while maintaining the spatial relationship of Figure 5b. There is also an ambiguity concerning which of the existing blocks to add the new block. We have restricted the grammar in this system to only allow a new block to be added to the most recently placed block, though it can be placed in any of the 16 corners which replicate the spatial relationship. If no labels are added to restrict rule applications, this is a non-deterministic basic grammar (Knight, 1999). This type of grammar is one of the most restrictive types of grammars. Still, even it contains enough complexity for our purposes: the grammar defines a design space with over one million possible five-step designs and over a trillion possible ten-step designs. Obviously, a grammar such as this cannot be enumerated. Even more daunting, though, is that the combinatoric complexity of this grammar prevents a designer from being able to predict what a sequence of rule applications will produce.

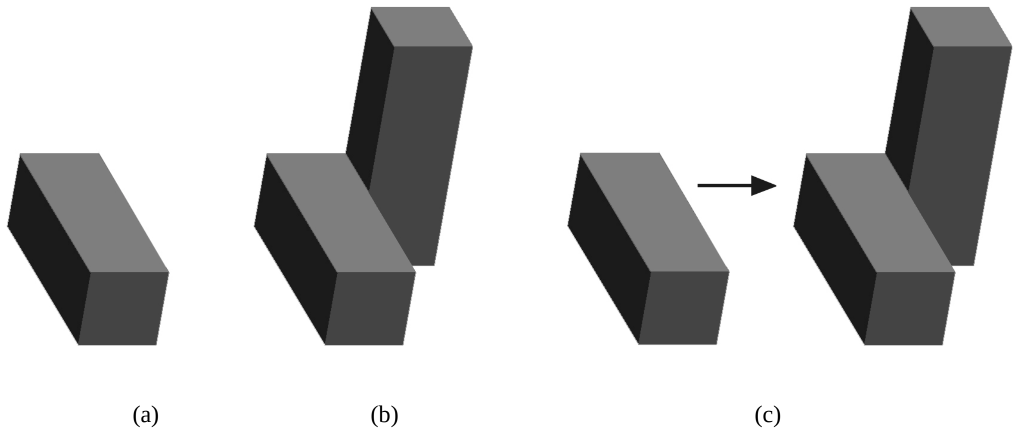

Figure 5. SGGA's shape grammar. (a) shape   (b) spatial relationship   (c) shape rule

The genetic algorithm we employ assists the user in navigating the space of possible designs, searching through different combinations of rule applications. Though based on the simple, and fairly standard, GA we described earlier, our genetic algorithm differs in several important ways. Most importantly, the SGGA is user-driven - at each population step, the user selects the population members which will be used to generate the next population. Consequently, the user acts as the fitness function. This prototype is geared towards exploration rather than optimization, and the user-directed nature of the selection process is invaluable in providing an exploratory tool. Moreover, we believe that as GAs gradually become employed as design tools, they will be, to a greater or lesser degree, user-directed. In most design contexts, optimizing some single, objective function is less of a concern than the satisficing of several concerns, and an intuitive feel for the direction that a design is taking is an invaluable tool. Employing a genetic algorithm for architectural designs, for example, will likely involve a selection process that combines constraints returned from simulations with selections made by the eye of the architect. In the case here, the shape grammar design space is largely of theoretical and aesthetic interest, and so the user-directed selection process is appropriate to the types of possible designs and ideal for pure exploration.

As required by the selection process, our population is sized so that it can be readily scanned by the user. SGGA thus maintains a population of nine designs from which the user selects. Though this population is quite small compared to most genetic algorithms, it does resemble other early experiments in the area, such as Dawkins biomorphs, and is well-suited to exploring the design space we have created. In every generation, nine designs, beginning with random designs at the start of a session, are presented to the user. The user chooses two to become the "parents" for the next generation. This next generation will consist of the two parents, thus maintaining some direct continuity with the previous generation, along with seven offspring formed through the crossover and mutation mechanism. This procedure is iterated in the typical manner of a genetic algorithm, and the populations tend to evolve into designs with some obvious family resemblances. Note, though, that since the GA selection process is user-driven, "evolution" is not as inherent to the process as in a typical GA. If the user maintains very high mutation rates with each generation, or selects designs with different criteria at each step, the process will come to resemble a random walk rather than an evolutionary process. Similarly, in keeping with the user-driven nature of the GA, we allow the user to inject a basic design (Knight, 1999) of their choosing as one or both of the parents at any generation. A basic design is a design generated by applying a single rule in the same way repeatedly, and can consist of any number of rule applications. The injection of these designs into the population amounts to a simple form of "genetic engineering" and allows the user more control over the resulting designs, which serves both the pedagogical and design purposes of the tool.

Another critical elaboration of the simple GA in our tool occurs in the genotype, which we have set up to favor repetition. Given the size and nature of our populations, it makes sense to artificially inject some form of repetition into the designs rather than wait for it to evolve over the course of time. Repetition improves the character and legibility of the abstract designs produced by the grammar here. In order to provide the user adequate means for choosing designs and navigating the design space in a goal-directed way, it becomes necessary to establish the genotype in such a way that repetition is more easily afforded. Thus the genotype in SGGA can be broken down into two parts. The first, which we will call the motif, is part of the design determined by a sequence of rule applications which describe where each new block should be set in relation to the preceding block. The second part consists of two numbers, establishing levels of repetition within the motif and within the ultimate design. So, a typical genotype in our scheme might look like this:

$$((M_1\ M_2\ M_3\ ...\ M_n)\ N\ X)$$

where each M_i ($1 \leq i \leq 16$) corresponds to one of the 16 ways a rule can be applied, and X and N are integers which determine the levels of repetition within and of the motif. Note that our GA can handle motifs of variable length, as the motif is not restricted to any set length. For example, both ((12 4) 2 2) and ((8 14 14 8 6) 1 3) would be legitimate chromosomes in SGGA. Expanding the first genotype, ((12 4) 2 2), would obtain the following rule sequence: 12 12 4 4 12 12 4 4. While the tail end of the genotype establishes an artificial level of repetition, the algorithm is still theoretically capable of evolving any design. Similarly, though we have reduced the space of *likely* designs by externally introducing repetition into the genotype rather than allowing it to naturally evolve, the search space is still enormous, and we have made it more likely that the user will select and search through designs of long length, as repetition makes them more legible.

The other important differences between SGGA and the simple genetic algorithm we outlined previously occur in the mechanisms of crossover and mutation. SGGA employs a type of uniform crossover, in which crossover can occur at every point along the chromosome string. Simply put, uniform crossover allows every point between two genes to become a potential crossover point, rather than just choosing one point, as we did in the previous example. While atypical in genetic algorithm research, and possibly disruptive of potential schemas, uniform crossover has been established in certain applications. And, after some experimentation with different crossover methods, we determined that an alternative to single-point crossover was required due to the typical genotype length, the population size, and our genotype structure.

Also different from the typical genetic algorithm, SGGA employs a variable mutation probability rate. In our prototype, the user selects a mutation rate along with the parents for the next generation. With this additional control, the user is able to control the extent to which a new generation is based on the previous. At early stages in the search, a higher mutation rate is usually preferred, in order to widen the range of potential designs. However, as the user finds a region of the design space that seems interesting, the mutation can be lowered or reduced to zero, ensuring that all designs have a direct relationship with the parents.

Figures 6-8 show a sample of three successive populations. The highlighted designs in each population are the designs selected to become parents for the next population. An examination of the designs in the three populations reveals a definite sense of movement within, and narrowing of, the design space. Figure 6 shows a population of completely randomly generated designs, whereas the population in Figure 8, two generations removed from the first, is composed of only 3 different kinds of rule applications and most of the designs share visible similarities. These three generations are typical of a beginning run of SGGA in that, through careful selection of parents, the search space under exploration can be quickly narrowed down. It normally takes many more generations, however, before more purposeful or obviously ordered designs can be evolved. But with a little patience anyone using the system can develop an intuition for the direction a population is heading and is soon capable of cultivating very surprising and stylistically similar families of designs.

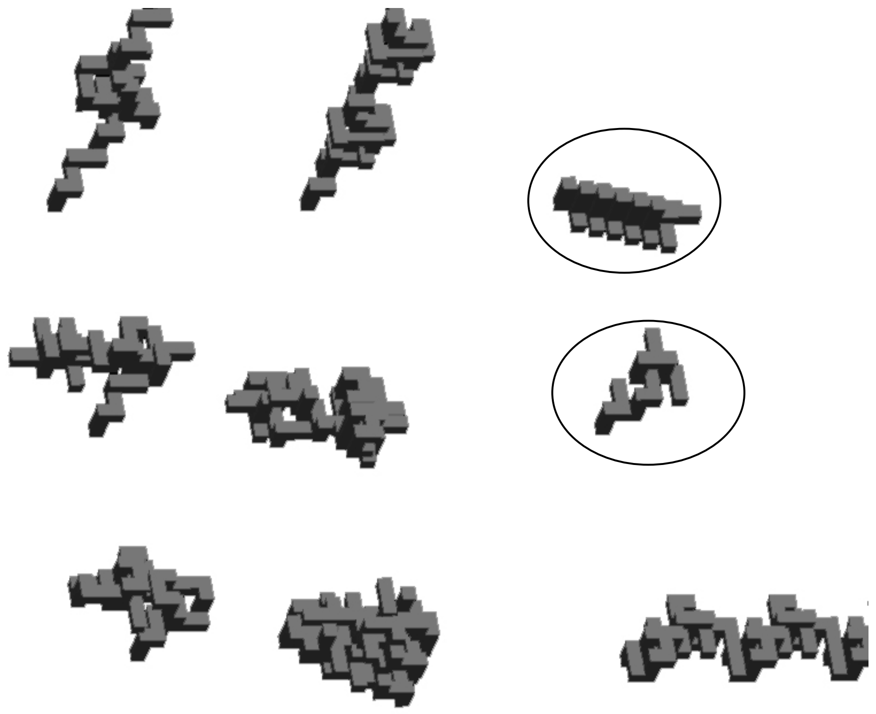

Figure 6. Population, generation 1. Circled designs will become parents for the next generation.

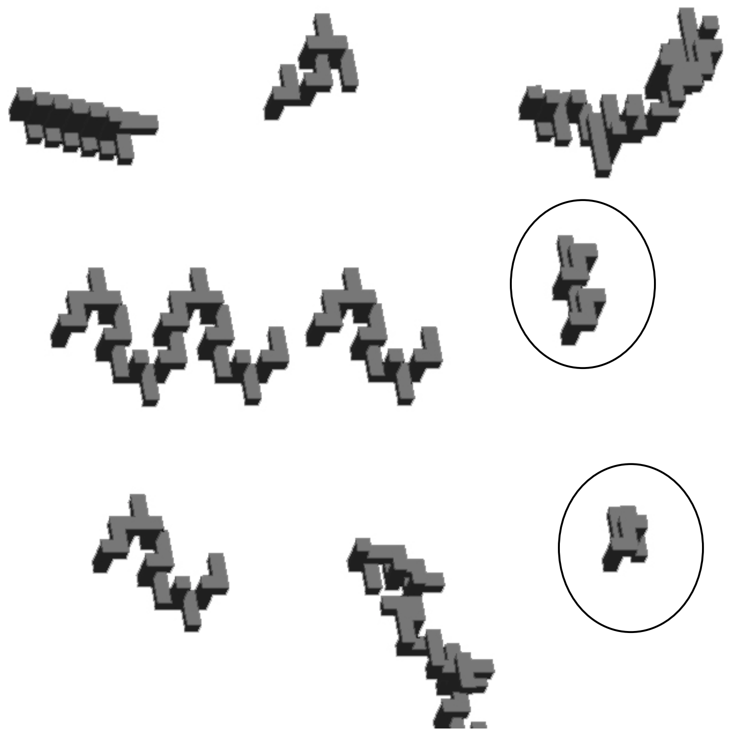

Figure 7. Population, generation 2. Circled designs will become parents for the next generation.

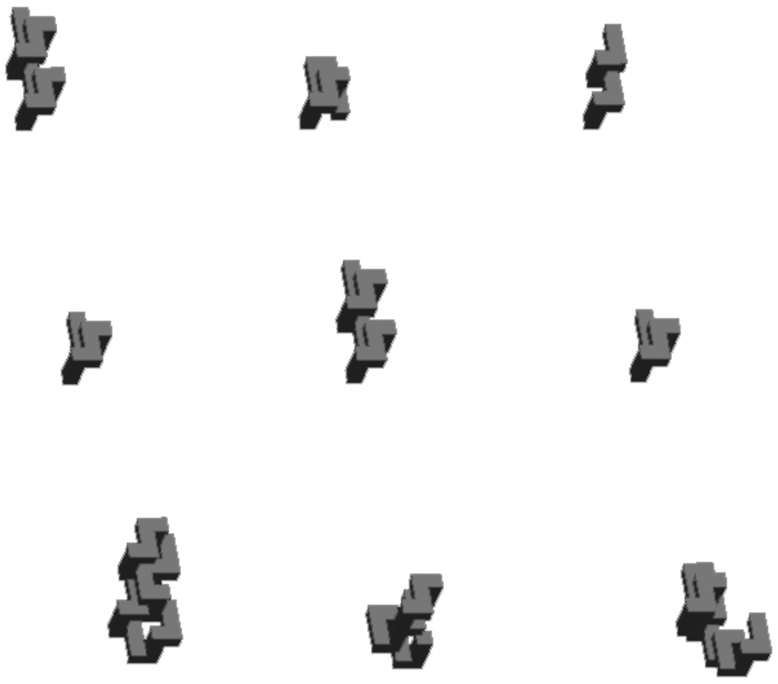

Figure 8. Generation 3. Some family resemblances are already apparent

Figure 9 shows several examples of the many design families that have been cultivated with SGGA. Some design types, such as the pinwheels, tend to appear sporadically during different runs and have very tenuous structures. This means that generations of new designs which use a pinwheel as a parent are unlikely to have offspring with the characteristic pinwheel structure. These types of designs seem to occur with a very precise convergence of rule applications and repetition, and even a minute change in the genotype will radically alter the phenotype. Conversely, other design types, such as the waffles and one-legged designs shown in Figure 9, share obvious resemblances even as the genotype undergoes the various transformations engendered by crossover and slight mutation. In the case of these two families, all of the designs shown are direct descendants of the underlined design.

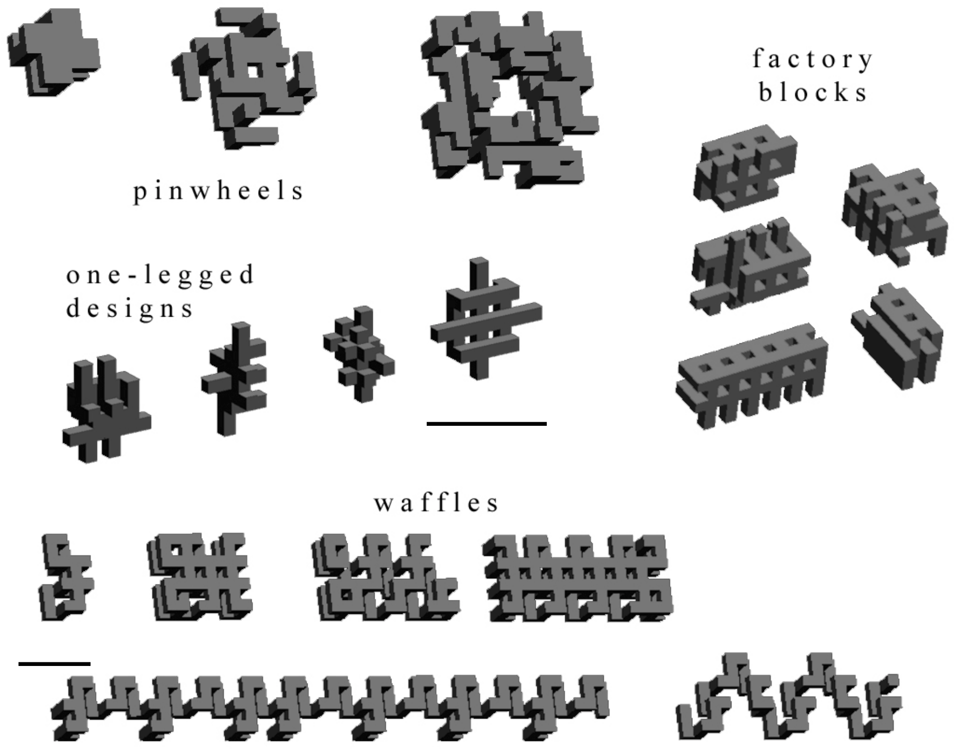

Figure 9. Various design families cultivated with SGGA

KEY ISSUES AND NEXT STEPS

As a pedagogical tool, SGGA is promising even at this stage of development. In addition to introducing students to the theoretical underpinnings of shape grammars and genetic algorithms, the system also engenders in its users a clear understanding of the somewhat abstract concept of a design space. Though generalizations about it's efficacy over a broad range of grammars would be difficult to justify given the limited grammar on which it is based, some specific potentials of the system are clear. The SGGA's capabilities for directed search illustrate aspects of shape grammars and genetic algorithms that would not be obvious using either method by itself. For example, after running a few generations of different populations, both novices and experienced shape grammarians quickly develop an intuitive feel for what small changes to rule sequences imply for the design. This intuition is simply not possible to develop easily when working by hand or with a computer program that automatically generates designs from a user specified rule sequence. And since this intuition can be developed quickly with SGGA, the structure of the design space created by the grammar is illuminated in a way not otherwise possible. Also, the user-driven nature of the genetic algorithm provides a powerful introduction to evolutionary search methods, especially since the user can act as the selection function, adjust mutation levels, and inject basic grammars into the population.

While such pedagogically interesting results can be obtained with the single spatial relationship we have described, turning the plug-in into a true design tool will involve adding the capability to explore other relationships and grammars. Current experiments with SGGA allow the genetic algorithm to also alter the proportions of the shapes and the spatial relationship in the grammar. Figure 10 shows a population of designs using the original spatial relationship next to a population where the shapes and spatial relationship have been mutated (though the rule applications remain the same). This additional capability will make the tool useful for design assistance in massing studies and other schematic design ideas. With this system, students, such as those at MIT who experiment with similar "block-stacking" shape grammars for massing studies, would be able to evaluate hundreds or thousands more variations of their design ideas than otherwise possible.

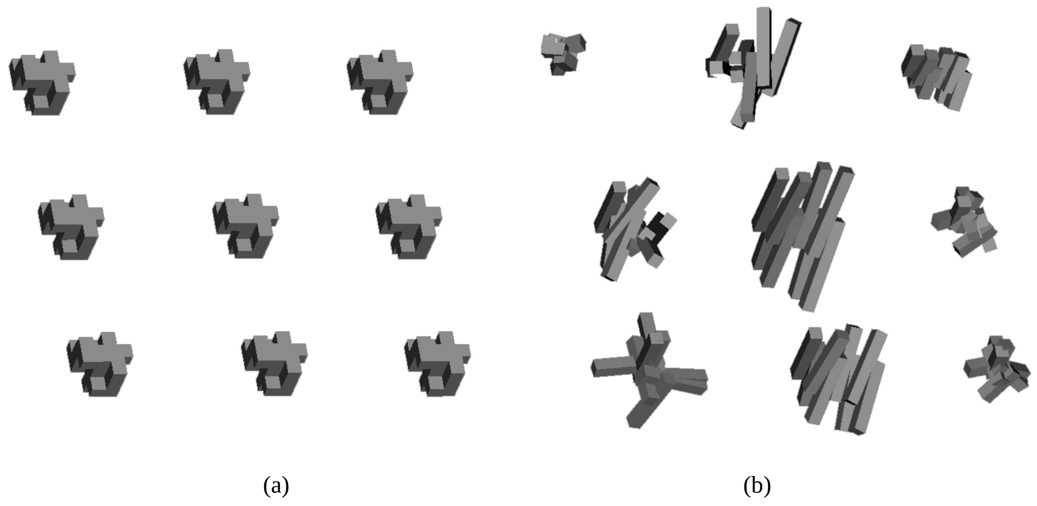

Figure 10. The spatial relationships of the design in (a) are transformed with SGGA in (b). The rule applications remain the same.

Further extensions of the SGGA system would make the program capable of even more design assistance. Obvious improvements such as enlarging the population, allowing for the selection of more than two parents, and evolving the grammars themselves could be combined with the provision of plug-in fitness functions (for example, symmetry, structural constraints, etc) and multiple populations to speed and improve the search process according to different design goals. To some degree, these and similar improvements have all been researched by others and would be somewhat straightforward extensions (Bentley, 1999; Rosenman and Gero, 1999). There are, however, deeper issues that will also need to be addressed in attempting to synthesize the full power of shape grammars and genetic algorithms. Foremost is the issue of resolving the inherent non-determinism in the "shape recognition" aspect of shape grammars and the traditional division in genetic algorithms between the genotype instructions and phenotype object. This is an issue beyond the scope of this project, but one which would need to be addressed in applications which attempted to make full use of both systems.

That said, SGGA is a promising first step synthesizing these two computational systems for the purposes of developing a generative design engine. Our system does not address some of the difficult issues involved in supporting the full theoretical possibilities inherent in each domain. Yet it does point to several critical paths that need to be researched in order to create a generative design system that can be used on a practical basis by artists and designers.

BIBLIOGRAPHY

Bentley, Peter. "From Coffee Tables to Hospitals: Generic Evolutionary Design." in Bentley, Peter, ed. 1999. *Evolutionary Design by Computers*. Morgan Kaufmann.

Frazer, John. 1995. *Themes VII: An Evolutionary Architecture*. Architectural Association.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. MIT Press.

Knight, T. W. 1999. "Shape grammars: six types" *Environment and Planning B* **26:** 15-31.

Mitchell, Melanie. 1996. *An Introduction to Genetic Algorithms*. MIT Press.

Rosenman, Mike & Gero, John. "Evolving Designs by Generating Useful Complex Gene Structures." in Bentley, Peter, ed. 1999. *Evolutionary Design by Computers*. Morgan Kaufmann.

Shea, Kristina & Cagan, Jonathan. 1998. "Generating Structural Essays from Languages of Discrete Structures" *AI in Design '98*: 365-383.

Stiny, George & Mitchell, W. J. 1978a. "The Palladian grammar" *Environment and Planning B* **5:** 5-18.

Stiny, George & Mitchell, W. J. 1978b. "Counting Palladian plans" *Environment and Planning B* **5:** 189-198.

Stiny, George. 1980a. "Introduction to shape and shape grammars" *Environment and Planning B* **7:** 343-351.

Stiny, George. 1975. *Pictorial and Formal Aspects of Shape and Shape Grammars*, Basel, Birkhauser,

Stiny, George. 1980b. "Kindergarten grammars: designing with Froebel's building gifts" *Environment and Planning B* **7:** 409-462.