Automatic Evolution of Conceptual Building Architectures

Corrado Coia
Dept. of Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
Email: year7c0@gmail.com

Brian J. Ross
Dept. of Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
Email: bross@brocku.ca

***Abstract*—An evolutionary approach to the automatic generation of 3D building topologies is presented. Genetic programming is used to evolve shape grammars. When interpreted, the shape grammars generate 3D models of buildings. Fitness evaluation considers user-specified criteria that evaluate different aspects of the model geometry. Such criteria might include maximizing the number of unique normals, satisfying target height requirements, and conforming to supplied shape contours. Multi-objective evaluation is used to analyze and rank model fitness, based on the varied user-supplied criteria. A number of interesting models complying to given geometric specifications have been successfully evolved with the approach. A motivation for this research application is that it can be used as a generator of conceptual designs, to be used as inspirations for refinement or further exploration.**

I. Introduction

The design of building structures is a complex endeavor requiring considerable architectural knowledge and experience. A successful building design requires attention to many requirements regarding functionality, structural integrity, cost, and aesthetics [1], [2]. Although computer technology is a fundamental tool of architects and designers, the entire architectural task is too complex and multifaceted to be completely automated at present time. On the other hand, specific aspects of architectural design can be effectively automated. For example, CAD systems aid in drafting exterior and interior designs, while structural analyses are supported by finite-element analyses software.

The external structural form of a building is a basic aspect of architecture. Computer tools are commonly used to aid in the design of architectural forms. An example of such a computational tool is the shape grammar [3]. Shape grammars are used for the computer-supported generation and editing of design ideas. They permit complex shapes to be refined via the structure of shape-generating rules in the grammar. Minor changes in the rules can result in new styles of generated models. This results in an efficient means for automating the generation of new design styles of building structures. For example, Stiny creates objects from blocks based on spatial relationships, using hand-coded shape grammars to extrude, split, add blocks, remove blocks, and other basic commands [4]. In another study, Stiny uses shape grammars to create paintings through the use of commands such as location, rotation and scale [5]. Tapia uses a visually guided shape grammar system to let the user specify the grammar, while the system displays the results, giving the user an interface to explore the designs from the specified language [6]. The designer specifies an initial shape on a 2D grid and at least one production rule prior to getting, exploring, and refining the grammar. Ando *et al.* use shape grammars to analyze well-known architectural structures [7]. The commercial application CityEngine [8] uses a shape grammar to create everything from building facades and structures [9], to entire layouts of cities [10]. Generated results can be used for architectural studies, computer games, and computer animations. Halatsch *et al.* use shape grammars to design city layouts [11].

Evolutionary design is a research field concerned with the application of evolutionary computation (EC) towards problems in modeling, structural engineering, image generation, and others [12], [13]. A comprehensive survey of evolutionary computation and structural design is by Kicinger *et al.* [14]. Since model topology design is the primary focus of this paper, the following is a brief survey of work in evolutionary design and model generation. Gero *et al.* use shape grammars with automated EC to produce topologies of a beam section conforming to specified physical criteria [15]. Buelow evolves trusses which are structurally optimal for withstanding loads [16]. Jackson evolves 2D L-systems with interactive genetic programming [17]. The Genr8 system of Hemberg *et al.* uses shape grammars and genetic programming to interactively evolve 3D surfaces [18]. O'Neill *et al.* use genetic programming to construct shape grammars that generate specified 2D images [19]. They extend their ideas to 3D in [20], and study the interactive generation of 3D shelters. Machado *et al.* use evolution with *Context Free Design Grammars*, which are image generating grammars akin to shape grammars [21]. EC and shape grammars are used to create 3D medieval Italian architectures [22]. Gero and Sosa use automatic EC to design adaptive automotive instrument panels [23]. Hornby uses EC and L-systems to evolve 3D table designs [24]. Jacob evolves models of flowers and plants using L-systems [25]. Beaumont and Stepley use grammatical evolution to evolve L-systems [26].

TABLE I
SELECTED FUNCTIONS OF SHAPE GRAMMAR LANGUAGE

Function	Meaning
$extrude(N)$	Extrudes the shape N units along the face normal.
$split(X)C$	Subdivides current shape along axis X, and applies command C to each subdivision.
$r(R_x, R_y, R_z)$	Rotates current shape around the pivot point using angles R_i.
$s(S_x, S_y, S_z)$	Sets size of the shape using scales S_i.
$insert(S)$	Adds an object S (3D model, polygon mesh).
[and]	Stack operations.

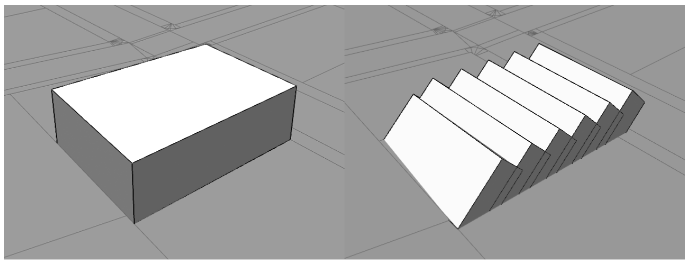


Fig. 1. (a) Model before initial extrude. (b) After split and rotate.

This paper presents research in which genetic programming (GP) is used to automate the construction of external building structures. The GP language used is CityEngine's shape grammar [8]. This language is a well-designed, sophisticated implementation of a shape grammar, that has an extensive track record of real-world applications. The shape grammar is denoted within the GP system as an ADF tree, where ADF modules represent shape grammar non-terminal rules. The GP system uses fully-automated fitness evaluation, where an assortment of geometric criteria is supplied by the user. Examples of these criteria include the characteristics of polygon normals, structure height, orthographic shape contours, and others. Multi-objective evaluation strategies are used to find the aggregate fitness scores for the multiple geometric specifications. A population diversity strategy helps promote genetic diversity during the run.

Our motivation is to use evolution as a means for automatic generation of concepts for 3D building design. Although shape grammars are powerful tools, they also require considerable technical expertise to use and debug effectively. We suggest that GP can be used to automatically explore shape-grammar generated building models without the need of shape grammar programming by the user. The intention is that evolved shape grammars will result in phenotypes (3D models) that comply to the given geometric specifications required by the user. We consider this approach as one that uses evolution to produce inspirational building concepts. The evolved structures might be the seeds of concepts, to be used as a basis for further design. Alternatively, models could be immediately used within computer animations and games.

The paper is organized as follows. Section 2 reviews background information in shape grammars and geometric specifications for 3D models. Section 3 describes aspects of the GP system and experiments, such as the multi-objective fitness strategy, shape grammar representation, and overall system architecture. Example experiments and results are given in Section 4. Section 5 gives concluding remarks, and directions for future work.

More details about this research are in [27].

II. BACKGROUND

A. Shape Grammars and CityEngine

A grammar is defined as follows:

- The alphabet $A = \{w_1, w_2, w_3, ..., w_n\}$, is a set of commands which can be used within the grammar.
- An axiom $I \rightarrow \alpha \in A$, which contains a string of commands from the alphabet which defines the initial state of the system.
- A set of production rules $R_n \rightarrow \alpha \in A$, where each production rule contains elements of the alphabet.

A shape grammar is a grammar in which the alphabet denotes the generation, manipulation, and alteration of 2D or 3D objects [3], [4], [15]. The rules within the grammar can specify shape replacements, translations, rotations, scaling, repetition, moving, splitting, and extruding. Once the grammar is defined, it is executed sequentially and the actions specified within the grammar are performed on the shape.

In the case of architectural design, the resulting 3D models arising from shape grammars represent building structures. The benefit of using a grammar to generate a building model is due to the fact that individual aspects of the building can be developed and reused multiple times during its construction. For example, a set of rules can be made which describe the structure of a floor within a building. This set of rules can be called multiple times to create multiple floors of the same or similar design, saving the designer from having to create each floor explicitly.

CityEngine is a commercial system used to create models of cities [8]. It uses grammars to create a system of roads [10] and 3D shape grammars to generate building models [9]. It is capable of generating huge city models, as well as many varieties of building styles. In order to create a building, a grammar must be hand-coded. A trail-and-error process is required, to inspect results and refine the grammar as required. Like any programming task, precision and care is required to obtain suitable results.

The CityEngine shape grammar functions used in our experiments are listed in Table I. This is a small subset of the full CityEngine language. The selected functions in this table define a small but powerful subset of useful expressions, that enable a variety of building structures to be constructed. The actual language implementation is much more detailed than shown. It supports such things as variables, mathematical expressions, and a comprehensive library of useful functions.

An example CityEngine grammar is as follows:

$$Start \rightarrow Extrude(10)\ Split(z)\{5 : RuleA\}*$$
$$RuleA \rightarrow Rotate(45, 0, 0)$$

The initial shape is the square lot. It is first extruded (Figure 1(a)). Then the extruded model is split along its z-axis into objects 5 units wide. $RuleA$ is applied to each resulting sub-object, which rotates each 45 degrees along the x-axis. The final result is in Figure 1(b).

B. Geometric Properties and Building Models

There are many factors an architect needs to consider when designing a building, due to the fact that every building is designed with specific goals in mind. Many of these goals can revolve around space, function, and form [1]. Space use may be reflected by maximizing the space allocated for the building, perhaps to fit the size of a lot. Function refers to constructing the building such that it is able to accomplish its task. Form refers to the external appearance of structures. Certain building designs could benefit from having a complex form. One example could be of an architect designing a building which is meant to represent the wealth of a particular financial corporation: the final design would require a high level of complexity to achieve the correct form. A design might be made more complex by considering offset levels, spiral and circular designs, and multi-tiered sections.

This research considers factors which fall into the space and form categories. Fitness evaluation considers different geometric properties of 3D models. For example, one geometric property of a sphere is that each surface normal is unique. One could also use this property as a criteria for generated building models. The greater the number of unique surface normals seen in a model, the closer it comes to matching this particular property of a sphere. Other properties would need to be satisfied, however, for a sphere model to arise. Another geometric property we use is to match the orthogonal projection of a structure to a target shape. If viewing the model from above, the result would be a 2D shape representing its silhouette or footprint. This silhouette can be compared to a specified shape, such as a triangle or a star. The goal is to have the model silhouette conform to the supplied shape.

The spectrum of results that can be obtained by different geometric requirements can be highly specific and well-defined, or vague and open-ended. For example, criteria specifying that the resulting building surface must contain 95% unique surface normals, be exactly 500 units tall, and fill a pentagon footprint 75 units wide, would over-specify the criteria, and possibly very challenging for evolution to satisfy. On the other hand, under-specifying the criteria permits too many extraneous solutions. A balanced set of criteria is preferable.

III. System

A. Architecture

The system architecture is depicted in Figure 2. Genetic programming is implemented with RobGP, a C++ based GP system [28]. RobGP has a particularly strong implementation of ADFs, which we use to denote the shape grammar. Looking at the figure, GP is used to evolve ADF trees that denote shape grammars. The GP tree is translated into a CityEngine-compliant shape grammar. The grammar is transferred to the CityEngine application. CityEngine parses the grammar file, and then executes the grammar, resulting in the generation of a 3D model. This model is exported as a Collada file [29], which is then read back into RobGP for fitness evaluation. A report file for each model is also generated by CityEngine, which can be used during fitness evaluation.

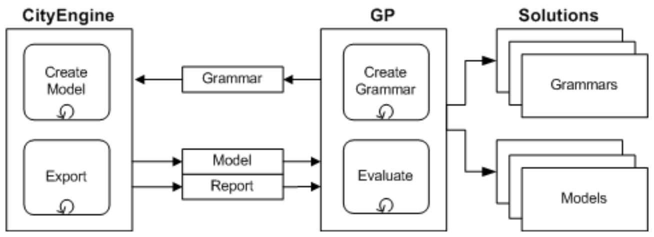


Fig. 2. System architecture.

Seven ADFs are defined in each GP tree. Each ADF represents one production rule within the shape grammar. This allows each production rule to evolve as its own unit, yet still able to reference other production rules. Program bloat is possible with the shape grammar. A tree may define unused production rules, which will simply be ignored during model generation by CityEngine. Certain sequences of commands can be ignored as well. For example, if a series of "size" commands are used in succession, only the final command has an effect. We rely on fitness pressure to penalize less productive grammars.

We use CityEngine's shape grammar language [8]. CityEngine implements a comprehensive shape grammar language, which can generate a huge variety of realistic building structures. A fully general shape grammar language such as this, however, will be ineffective for use with GP. A 3D shape grammar has the potential to create a vast number of structures. Many generated structures will contain disconnected components; for example, consider 100 cubes scattered randomly in space. Since we are primarily interested in models of buildings, such disconnected models are unlikely to be of interest. Furthermore, the generality of shape grammars permits the definition of virtually an infinite number of varied models, for example, buildings, furniture, cars, and people. Evolved models can diverge considerably from typical building structures, which can be a blessing or curse, depending on ones intended use of evolution.

To improve the likelihood of evolving models that look like buildings, the shape grammar language used by genetic programming is constrained. Rather than supply GP with the entire definition of CityEngine's shape language, we use a subset of useful CityEngine functions (Table I). These functions provide a powerful subset of geometric constructions, and when placed into shape grammar rules, permit a variety of

TABLE II
GP PARAMETERS

Parameter	Value
Runs	10
Generations	60
Population	300
Crossover	0.90
Mutation	0.08
Elite migration	0.02
ADFs	7
Initial tree method	Grow
Max tree size	50
Tournament size	3
Fitness	multi-objective
Individual ranking	ranked sum
Diversity penalty	20

interesting models to be generated. The exact set of primitives chosen for an experiment can be tailored to the style of models to be evolved. The degree to which the language is constrained depends upon the intended goals of evolution for the particular application being considered. A constrained language can focus search on a specific building style. Less constraints result in a larger variety of potential solutions, albeit perhaps at the cost of evolutionary performance and solution quality.

Language primitives themselves can be further constrained, depending on the goals of a particular GP run. Some function parameters can have sensible values hard-coded into them, or useful ranges of values predefined. The grammar itself can be partially predefined as well, in order to direct evolution towards sensible models of interest. For example, consider the situation in which a skyscraper or tower complying to user-supplied specifications is to be evolved. For this task, the base grammar expression in the GP tree can be preset to use a given primitive (eg. extrude in the y (up) direction). This helps the grammar generate models that are likely to be tower structures. Evolution can then concentrate on searching the space of towers, rather than search for tower structures themselves. Supplying problem-specific constraints to the shape grammar is a great benefit to the evolutionary design process. With minimal technical intervention, the user can predefine the desired family of structures to be investigated, thereby allowing evolution to concentrate on searching for more refined solutions to the problem at hand.

Table II summarizes some typical parameters used in our experiments. Most parameters are self-explanatory (see [30]). Note that although the maximum tree size of 50 nodes is small compared to most GP applications, this size limit is adequate for shape grammars. Unfortunately, the scope of our experiments (runs, population size, generations) was less than ideal, as we met system factors outside our control. One machine license of the CityEngine system was available, and only a single run on one CPU could be executed at a time. This limited the total runs possible. Also, known memory leaks in CityEngine severely handicapped the duration of runs and the size of the population. Nevertheless, experiments were performed within the limitations and resources available.

B. Multi-objective Fitness Evaluation

When evolving 3D building models, there will be different geometric criteria to use. These criteria can conflict with one another. For example, one goal might be to maximize the number of unique surface normals, while another goal can be to define the silhouette of the building shape. The larger the building is, the greater the surface area, which makes having a greater number of unique normals easier to obtain. However, this conflicts with the second goal, which limits the unbounded size of a model.

The above is a natural multi-objective problem [31], [32]. An advantage of using multi-objective evaluation is that different criteria can be evaluated independently of one another, without introducing user bias that arises with a weighted sum formula. This is especially relevant when different criteria use quite different measurements and metrics for fitness scoring. Also, multiple diverse solutions can be produced during a run using multi-objective evaluation.

We use a ranked sum fitness approach for multi-objective evaluation. This strategy was originally derived for high-dimensional multi-objective problems [33]. However, experience shows that it is also effective for low-dimensional problems [34], [35]. Consider a fitness vector $[f_1, \ldots, f_k]$ for a k-objective problem. Each fitness f_i has its rank r_i determined: $[r_1, \ldots, r_k]$, where $(1 \leq r_i \leq N)$ for a population of size N. Then the sum rank is:

$$fitness = \sum_{i=1}^{k} w_i r_i$$

where w_i is an optional weight (default 1). An option is to normalize each rank before summation:

$$[\frac{r_1}{R_1}, \ldots, \frac{r_k}{R_k}]$$

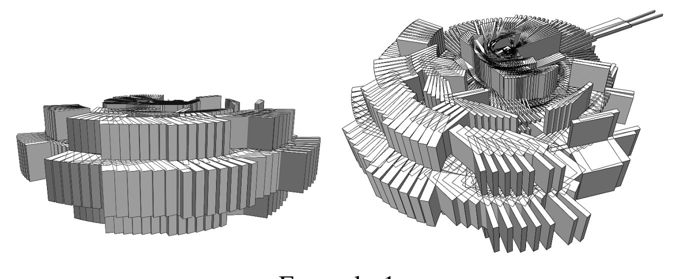
where R_i is the maximum rank found for objective i.

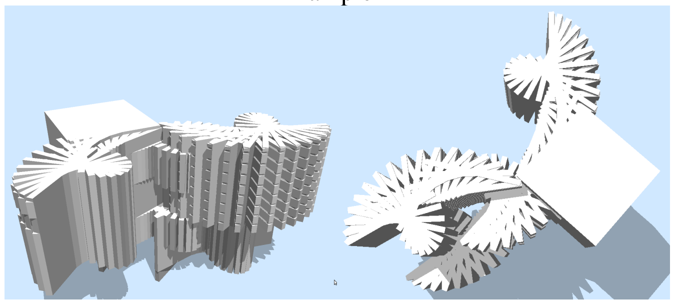
A diversity promotion strategy is used. A penalty factor (eg. 20 in Table II) is added to every rank r_i of an individual each time its raw fitness vector is found to be identical to another in the population. This has the effect of penalizing the overall sum of ranks score for that duplicate individual. Multiple duplicates are assigned even further penalized scores.

IV. RESULTS

A. Single-objective

This section shows selected results using single objective criteria. The architectural goal is to evolve geometrically complex building structures which at least subjectively, may be visually interesting. Initial runs maximized the polygon count as an objective. This resulted in unfeasibly large models evolving within a few generations, which also crashed the system. Maximizing the number of surface normals was also


Example 1


Example 2

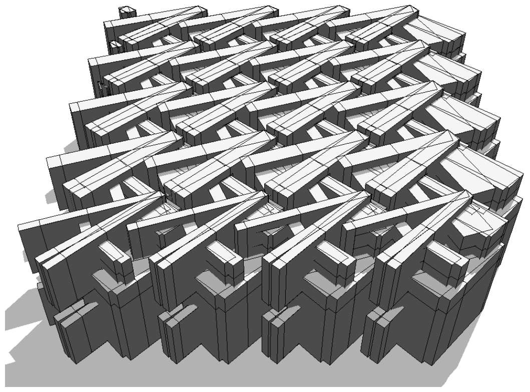
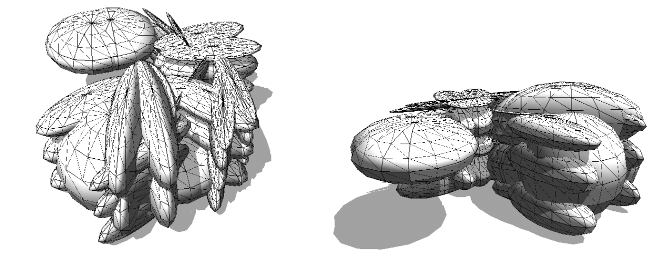

Example 3

Fig. 3. Maximizing unique normals.

tried, but without success. Usually, GP evolved shape grammars that repeatedly subdivided polygons into highly tiled planar surfaces, with one normal per polygon. The models, however, were not visually interesting. We discovered that maximizing the number of *unique* surface normals gave the most interesting results, since it requires shape grammars to orientate surfaces into new directions.

Figure 3 shows some results for maximizing the number of unique surface normals in the model. Example 1 has a total of 542 unique normals. This is less than number of polygons, as some structures (floors) are repeated, which results in duplicate normals. Example 2 has 987 normals. Although example 3 only has 61 unique normals, its repetitive structure is interesting. All 3 examples show symmetric, repetitive patterns, which arise directly from the calling patterns used in their shape grammars.

Figure 4 shows further results that search for a maximized unique normal count. Here, a 224-polygon sphere is made available as a basic building component. The resulting models include many spheres, in order to increase the unique normal count. The first example has 4184 normals, while the second has 1330. Example 2's simpler, balanced design is more visually appealing to our tastes. Note that spheres cannot be recursively split and duplicated, and so we did not obtain models with thousands of spheres (unlike the divided polyhedra in Figure 3).


Example 1

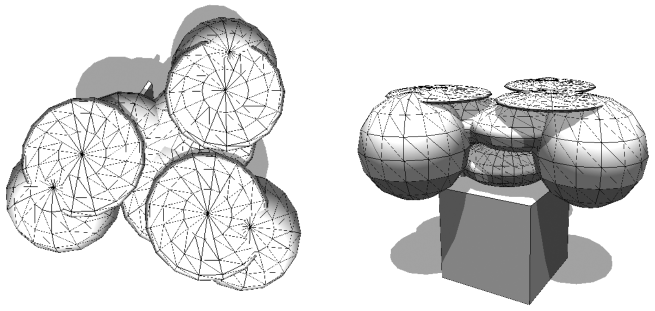
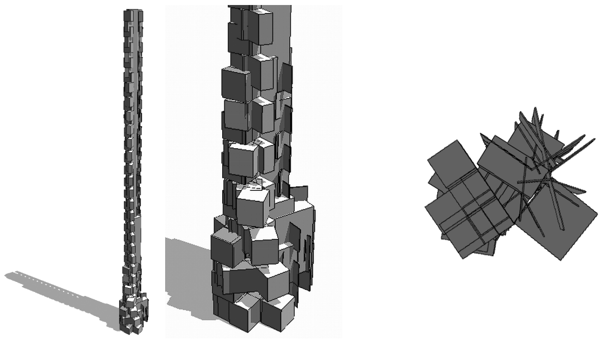

Example 2

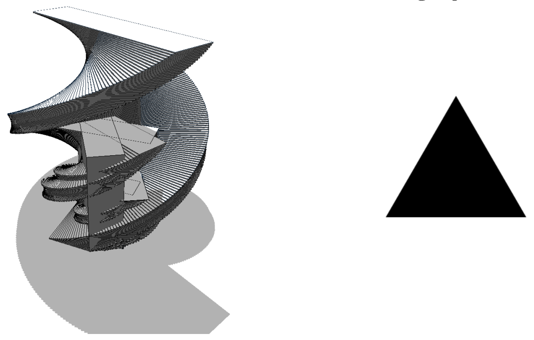
Fig. 4. Using spheres to increase unique surface normals.

B. Multi-objective evolution

More advanced multi-objective evolution was performed using image-based model shaping. The goal is to make a complex skyscraper whose vertical projection is as close as possible to a triangular boundary. The objectives are (i) reach a target height of 1500, (ii) maximize unique normals, and (iii) fit the top-view projection of the model to a given shape mask. The target height is actually higher than necessary, especially relative to the dimensions of the shape mask. Models even half this height will take the form of high towers. Note that the shape objective inhibits the maximizing of normals, since unbounded models are more likely to have many normals. A distance measure is used to evaluate the proximity of model heights to the target height. This will penalize models shorter and taller than the target. The vertical projection shape used is the equilateral triangle in (e). The fitness test for shape fitting first renders the model using an orthographic projection down the Y (vertical) axis. The resulting bitmap is then compared to the target triangle. A perfect score is if the rendered model image corresponds exactly to the triangle. Otherwise, the percentage of erroneous pixels are determined. These are pixels within the triangle that are not rendered, or



(a) Tower. (b) Detail. (c) Vertical projection.



(d) Spiral tower. (e) Vertical projection target shape.

Fig. 5. Multi-objective results: maximize normals, reach target height, shape fit vertical projection.

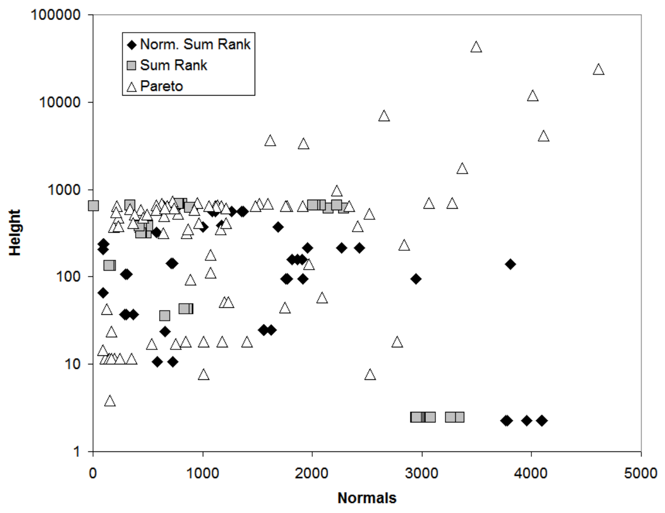


Fig. 6. Scatter plot of all multi-objective solutions.

pixels rendered outside the triangle. Normalized sum rank is used for scoring. The base model that the grammar works from is a box 50 units high, laying within the triangle.

A few results are shown in Figure 5. The tower (a) is one of the more pleasing solutions. Its vertical projection in (c) conforms fairly well to the target shape. Although its height is just 314, which is well below the target of 1500, its ability to conform to the vertical shape means that a tower structure has evolved. Its normal count is 538, due to rotated structures (see detail in (b)). For comparison, another evolved solution is in (d). Its height (189.8) and shape scores are much weaker, but its normal count of 1610 is very high. It also shows an intriguing "conch shell" shadow.

C. Comparing multi-objective strategies

TABLE III
MULTI-OBJECTIVE COMPARISON STATISTICS: SOLUTION QUALITY (N=NORMALS, H=HEIGHT)

	Norm. sum rank		Sum rank		Pareto	
	N	H	N	H	N	H
μ	1442.7	248.6	1332.6	342.8	1295.2	3165.5
σ	1191.0	237.8	1087.9	272.6	1108.5	15901.3
min	90	2.2	6	2.5	92	3.8
max	4098	650.0	3340	683	4961	138215

The multi-objective runs in Section IV-B use variations of sum rank scoring. To see how sum rank results compare with those using the more common Pareto ranking technique,

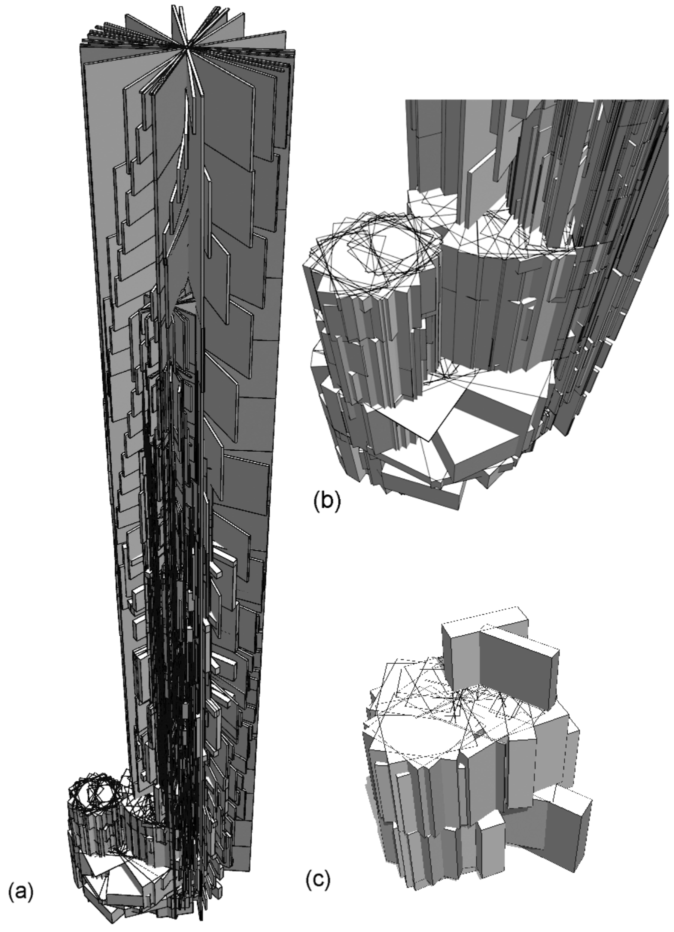


Fig. 7. Comparing sum rank and Pareto. Tower in (a), with detail in (b), from a normalized sum rank. Model in (c) is a Pareto outlier with low height and normal scores.

TABLE IV
GRAMMAR FOR TOWER IN FIGURE 7(A).

```
::: predefined variables
attr baseHeight = 50

::: Evolved rules...
Lot -->
  extrude(baseHeight) [ s(1.72999, 1.72999, 0.742476)
  [ r(scopeCenter, 0, 157*split.index/split.total, 0)
  [ split(y){ baseHeight : RuleC }* ] s(0.362006, 1.72999,
  1.72999) RuleC s(0.362006, 1.72999, 0.742476) s(0.362006,
  1.72999, 1.72999) [ split(y){ baseHeight : [ RuleB
  s(0.362006, 1.72999, 0.742476) s(0.362006, 1.72999,
  0.742476) RuleC [ RuleB RuleC s(1.72999, 1.72999,
  0.742476) RuleC RuleC ] ] }* ] s(0.362006, 1.72999,
  1.26481) RuleC RuleC RuleC ] s(0.362006, 1.72999,
  0.742476) RuleC RuleC ]

RuleA --> r(scopeCenter, 0, 183* split.index/split.total, 0)

RuleB -->
  [ [ r(scopeCenter, 0, 283*split.index/split.total, 0)
  split(x){ baseHeight : r(0, 108*split.index/split.total,
  0) }* [ [ r(scopeCenter, 0, 283*split.index/split.total,
  0) split(x){ baseHeight : r(0, 108*split.index/
  split.total, 0) }* split(y){ 18 : RuleA }* r(scopeCenter,
  0, 279* split.index/split.total, 0) RuleA ] split(x){
  baseHeight : RuleA } split(y){ 43 : [ RuleA split(y){
  baseHeight : RuleA } ] }* ] r(scopeCenter, 0, 279*
  split.index/split.total, 0) RuleA ] split(x){ baseHeight
  : RuleA } split(y){ 43 : [ r(scopeCenter, 0, 279*
  split.index/ split.total, 0) RuleA split(y){ baseHeight
  : RuleA } ] }* ]

RuleC --> RuleB split(x){ 16 : RuleB }* split(x){ baseHeight
  : RuleA }*

::: Unused rules...
RuleD --> r(scopeCenter, 0, 281*split.index/split.total, 0)
RuleE --> r(scopeCenter, 0, 252*split.index/split.total, 0)
RuleF --> s(1.13894, 0.433669, 0.554051)
RuleG --> s(0.284231, 1.81872, 0.815832)
```

runs were performed using the objectives of maximizing unique normals, reaching a target height of 750, and constrain the model to a given square boundary. A total of 10 runs were done for each of sum rank, normalized sum rank, and Pareto ranking. 10 elite individuals were saved from each run, resulting in 100 solutions from each of the 3 strategies.

After completing the runs, duplicate solutions, as indicated by identical fitness vectors, were pruned from the 100 solutions from each strategy. This resulted in 52 normalized sum rank, 38 sum rank, and 83 Pareto solutions. This immediately showed that the sum rank strategies were converging faster than the Pareto runs, the latter having significantly more genetic diversity. In hindsight, a stronger diversity penalty than what we used would have been useful.

Statistics for the normal and height scores for the pruned solutions are shown in Table III, and a scatter plot for all the solutions is in Figure 6. A t-test shows that the normalized sum rank results are statistically significant with respect to the height scores, in comparison to the sum ranks (92% significance). T-tests also show that both sum rank strategies produce statistically significant compared to the Pareto results, with respect to height scores (95% significance). Unique to the Pareto runs were models with heights far above the target of 750. These are easily seen in the top portion of the scatter plot. The strategies are generally comparable with respect to normal scores.

Not shown in the table is that Pareto runs were poor performers on the boundary condition. 60 out of 83 (or 72%) of the unique Pareto solutions violated the boundary test. In comparison, only 3 out of 52 normalized sum ranks failed the boundary test, while none of the sum rank solutions violated it. The Pareto boundary violators help allow many outliers, including those with excessive heights.

A few examples of results are shown in Figure 7. The tower in (a), with detail in (b), is a good quality solution from a normalized sum rank run. It has 1624 normals, its height is within 24 units of the 750 target altitude, and it lays within the boundary. Its evolved grammar is shown in Table IV. The Pareto solution in (c) is an example of an outlier. It has a good score only in the boundary condition, while having only 222 normals and a height of 99.

We find that the normalized sum rank generates the best quality solutions. Non-normalized sum rank solutions tend to be idiosyncratic, having converged to more unusual areas of the search space. Pareto results are the most varied, and outliers are common. Although our insights are based on limited runs, we conjecture that these results would be more apparent with larger populations and longer runs.

V. CONCLUSION

A method for evolving 3D building models using fully automated fitness evaluation has been presented. After specifying geometric criteria to be used as building specifications, GP evolves shape grammars whose phenotypes are 3D models that hopefully satisfy the specifications. A spectrum of possible building shapes will arise in this approach, depending upon the number and scope of the supplied geometric constraints. Our experience is that a balance must be maintained between under- and over-specifying building requirements. Multi-objective evaluation is particularly useful in this problem domain, since it does not introduce user bias as occurs with single-objective evaluation with weights. Our strategy is to constrain the shape grammar language to conform to the general style of building structures to be evolved.

Our approach is useful for exploring possible building structures, for use as concepts and inspiration by the user. Even when a solution satisfies the fitness criteria, however, it still may be unsatisfactory, due to a user's subjective opinions and aesthetic sensibilities. There is always be a large degree of subjectivity in art and architecture. In this case, it is up to the user to inspect solutions and select those that are most appealing. Depending on the experiment, we found that most single runs produced at least a few models that were interesting and pleasing to us. An architect might take an evolved model, and use it as a conceptual design basis for a more serious and functional end result. Alternatively, the models are suitable as-is for a computer animation or video game.

Our experiments used fitness objectives based on rudimentary geometric properties of 3D models. We were primarily motivated to evolve complex models, as determined by high unique normal counts. This usually results in models that are visually engaging, albeit perhaps impractical as real structures. Of course, many other styles of models are evolvable, should suitable fitness criteria be used. We have not considered functional requirements here, and it is indeed possible to do so. Structural analyses and other functional considerations might be included during fitness evaluation. In terms of functional usage considerations, an approach similar to that used by Flack for 2D floor plan evolution could be adopted [35]. He uses a number of efficiency considerations and room assignment analyses during evaluation of house floor plans. Similar analyses could be applied to 3D building models, to consider cost, structural integrity, component usage, energy efficiency, building codes, and other factors.

Fully automated evolution of 3D structures has not been extensively examined in evolutionary design. Rather, most research uses interactive evolution of 3D models, possibly guided by automatic fitness scoring of the population. For example, Genr8 evolves 3D structures using interactive genetic programming [18]. Hornby's work in evolution of table designs uses automatic evaluation of geometric and physical properties, which are similar in spirit to our geometric analyses [24]. O'Neill *et al.*'s work in using GP with 2D [19] and 3D [20] shape grammars also shares similarities with ours. Their 3D application requires interactive guidance by the user.

There are many future directions for this work. More advanced evaluation criteria are possible, including functional and structural evaluations. Enhancements and fine-tuning of the shape grammar would permit evolution of particular families of building shapes and styles. An interface that combines interactive and automatic evolution is worth considering. This would allow a user to become a more active participant in the evolutionary process, without discarding automatic evolution as a primary evolutionary force.

ACKNOWLEDGMENT

Research supported through NSERC Operating Grant 138467.

REFERENCES

[1] F. D. Ching, *Architecture - Form, Space, and Order*. Wiley, 2007.

[2] Vitruvius, *Ten Books on Architecture*. BiblioLife, 2009.

[3] G. Stiny, "Introduction to shape and shape grammars," *Environment and Planning B*, vol. 7, pp. 343–351, 1980.

[4] ——, "Kindergarten grammars: designing with froebel's building gifts," *Environment and Planning B: Planning and Design*, vol. 7, no. 4, pp. 409–462, July 1980.

[5] G. Stiny and J. Gips, "Shape grammars and the generative specification of painting and sculpture," in *Information Processing '71*, C. V. Friedman, Ed., Amsterdam, 1972, pp. 1460–1465.

[6] M. Tapia, "A visual implementation of a shape grammar system," *Environment and Planning B: Planning and Design*, vol. 26, pp. 59–73, 1999.

[7] N. Ando, N. Yamahata, S. Masumi, and M. Chatani, "Shape grammar and form properties of architectural figures," *Journal for Geometry and Graphics*, vol. 5, no. 1, pp. 23–33, 2001.

[8] "CityEngine," 2011, last accessed January 4, 2011. [Online]. Available: http://www.procedural.com/

[9] P. Muller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," in *Proc. SIGGRAPH '06*. New York, NY, USA: ACM, 2006, pp. 614–623.

[10] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 301–308.

[11] J. Halatsch, A. Kunze, and G. Schmitt, "Using shape grammars for master planning," in *Design Computing and Cognition '08*. Springer, 2008.

[12] P. Bentley and D. Corne, *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.

[13] J. Romero and P. Machado, *The Art of Artificial Evolution*. Springer, 2008.

[14] R. Kicinger, T. Arciszewski, and K. Jong, "Evolutionary computation and structural design: A survey of the state-of-the-art"," *Computers and Structures*, vol. 83, no. 23–24, pp. 1943–1978, 2005.

[15] J. S. Gero, S. J. Louis, and S. Kundu, "Evolutionary learning of novel grammars for design improvement," *AIEDAM*, vol. 8, pp. 83–94, 1994.

[16] P. von Buelow, *Genetically Engineered Architecture - Design Exploration with Evolutionary Computation*. VDM Verlag, 2007.

[17] H. Jackson, "Toward a symbiotic coevolutionary approach to architecture," in *Creative Evolutionary Systems*. Academic Press, 2002.

[18] M. Hemberg, U.-M. O'Reilly, A. Menges, K. Jones, M. da Costa Goncalves, and S. R. Fuchs, "Genr8: Architects' experience with an emergent design tool," in *The Art of Artificial Evolution*. Springer, 2008.

[19] M. O'Neill, J. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg, "Shape grammars and grammatical evolution for evolutionary design," in *Proc. GECCO '09*. ACM, 2009, pp. 1035–1042.

[20] M. O'Neill, J. McDermott, J. Swafford, J. Byrne, E. Hemberg, and A. Brabazon, "Evolutionary design using grammatical evolution and shape grammars: designing a shelter," *Intl. Journal of Design Engineering*, vol. 3, pp. 4–24, 2010.

[21] P. Machado, H. Nunes, and J. Romero, "Graph-based evolution of visual languages," in *Proc. EvoMusArt*, vol. 2. Springer, 2010, pp. 271–280, lNCS 6025.

[22] C. Soddu, "Recognizability of the idea: The evolutionary process of argenia," in *Creative Evolutionary Systems*. Academic Press, 2002.

[23] J.S.Gero and R. Sosa, "Complexity measures as a basis for mass customization of novel designs," *Environment and Planning B*, vol. 35, no. 1, pp. 3–15, 2008.

[24] G. Hornby, "Functional scalability through generative representations: the evolution of table designs," *Environment and Planning B*, vol. 31, no. 4, pp. 569–587, 2005.

[25] C. Jacob, "Evolving evolution programs: Genetic programming and L-systems," in *Proc. Genetic Programming 1996*. MIT Press, 1996, pp. 107–115.

[26] D. Beaumont and S. Stepney, "Grammatical evolution of l-systems," in *Proc. CEC 2009*. IEEE Press, 2009, pp. 2446–2453.

[27] C. Coia, "Automatic evolution of conceptual building architectures," Master's thesis, Department of Computer Science, Brock University, 2011, (expected).

[28] R. Flack, "Robgp - robust object based genetic programming system," Dept of Computer Science, Brock University, Tech. Rep., Nov. 2010. [Online]. Available: http://sourceforge.net/projects/robgp/

[29] "Collada," 2011, last accessed Jan 5, 2011. [Online]. Available: http://www.collada.org/

[30] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[31] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," in *Evolutionary Computation*, 1996, pp. 3(1):1–16.

[32] Coello, C. Coello, Lamont, G.B., and Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed. Kluwer, 2007.

[33] P. Bentley and J. Wakefield, "Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms," in *Soft Computing in Engineering Design and Manufacturing*. Spinger Verlag, 1997.

[34] S. Bergen and B. Ross, "Evolutionary Art Using Summed Multi-objective Ranks," in *Genetic Programming - Theory and Practice*. Springer, May 2010.

[35] R. Flack, "Evolution of architectural floor plans," Master's thesis, Department of Computer Science, Brock University, 2010.