Environment and Planning B: Planning and Design 2002, volume 29, pages 913 – 931

DOI:10.1068/b12839

Supporting designers' hierarchies through parametric shape recognition

Jay P McCormack, Jonathan Cagan
Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA;
e-mail: jmeq+@andrew.cmu.edu, cagan@cmu.edu
Received 27 July 2001; in revised form 2 April 2002

Abstract. The need to implement shape grammars efficiently, rather than hardcode them, in a way that supports creativity through shape emergence is an ongoing research challenge. This paper introduces a shape grammar interpreter that supports parametric subshape recognition, and thereby shape emergence. The approach divides shapes into hierarchies of subshapes based on specified geometric relationships within the shape. A default hierarchy based on geometric relations often found in engineering and architectural designs is presented as an efficient example of one appropriate hierarchy. The interpreter's shape recognition and generation abilities are demonstrated with two examples: a new engineering shape grammar for the design of vehicle inner panels and a modified version of the classical ice-ray shape grammar.

1 Introduction

Shape grammars, with their roots in the architectural literature (Stiny and Gips, 1972; Stiny, 1980), have recently found application in engineering. For example, Agarwal and Cagan (1998) introduced the coffeemaker grammar, Brown et al (1994) the lathe grammar, Shea and Cagan (1997) the truss grammar, and Agarwal et al (2000) the MEMS resonator grammar. Shape grammars can most easily be thought of as a type of expert or production system based on geometry (Gips and Stiny, 1980). Shape grammars, however, have succeeded in engineering applications where traditional expert systems have failed because of (1) their direct handling of and reasoning about geometry, (2) their ability to operate on a parametric geometric representation, and (3) their ability to support emergence of shape (Agarwal and Cagan, 2000). Emergent shapes are produced by rule applications in which the emergent shape arises from right-hand shape interactions and is more than a predetermined outcome from combined shapes.

In the past, shape grammars have been limited by the difficulty and time intensity in their implementation. Implementations have not allowed for parametric shape recognition and parametric subshape recognition in a single implementation, where subshape recognition (enabling search for emergent shapes) is a subset of shape recognition and implies the ability to perform shape recognition as well. Engineering shape grammar implementations in particular have been restricted to limited shape recognition and often were hardcoded and label driven, where a label is a letter or symbol associated with geometry and matched to determine rule applicability. This is unfortunate because much of the potential power of shape grammars lies in the ability to recognize emergent shapes, which can lead to novel designs. A system based on shape grammars that uses subshape recognition and parametric subshape recognition could provide a richer approach to design generation.

The general shape grammar interpreter introduced in this paper allows for engineering shape grammars or other types of shape grammars to be implemented in a fraction of the time that it currently takes to hardcode them. This quick turnaround from rules on paper to an implemented engineering system allows shape grammars to

be adjusted, fine tuned, and adapted to the changing design scenario presented to the rule writer. This interpreter possesses the features desired in an engineering grammar implementation, including parametric subshape recognition, providing designers the possibility to explore the promising potential of engineering shape grammar systems.

This paper presents a method for parametric subshape recognition based on a hierarchical decomposition of two-dimensional shapes that lie in a single plane ($U_{12}XV_{12}$). The decomposition is formulated to support design intent from specific to general. The hierarchy in turn decomposes the subshape recognition process into a successive search for each part of the decomposed shape. Matches of each part of the decomposed shape are then assembled for an entire shape match. Details of this process will be explained and illustrated through a version of a classic example from the shape grammar literature and through an industry-based engineering example.

2 Background

A shape grammar (Stiny, 1980) is a set of rules, based on shape, that is used to generate designs through rule applications. Rules take the form of $a \rightarrow b$, where a and b are both shapes. A rule is applicable if the left-hand shape, a, can be found in the design shape c. If the rule is applied, the left-hand shape is subtracted from the design and the right-hand side is added to the design $c - \mathrm{t}(a) + \mathrm{t}(b)$, where shapes a and b undergo a transformation t according to the transformation required to make shape a a subshape of shape c.

Parametric shape grammars are an extension of shape grammars in which shape rules are defined by a general schema and are applied by filling in the open terms. Given a rule $a \rightarrow b$, an assignment g which gives specific values to all the variables in a and b determines a shape $\mathrm{g}(a) \rightarrow \mathrm{g}(b)$ which can then be applied on a shape to generate a new shape.

A shape grammar interpreter is a computer program that can perform the operations necessary for determining if a rule can be applied (shape recognition) and can perform the operations needed to apply a rule (shape addition and shape subtraction). Krishnamurti (1980) described the algorithms needed for the implementation of shape grammars by a system that performs subshape recognition of an entire shape by using three transformations: rotation, translation, and isotropic scaling to match similar shapes. Although this work provided a solution for the subshape recognition problem, it did not allow parametric rules to be implemented. Attempts at creating engineering shape grammars able to be implemented by nonparametric interpreters were not successful because of the rigid nature of these schemes. Instead, engineering grammars have relied heavily on labels to determine the rule application, avoiding the difficulties with subshape recognition, but not taking full advantage of shape grammar properties such as emergence. Additionally, each rule in engineering grammars required text to describe the rule and additional text to explain where the designer intended this rule to be applied. For example given the rule in figure 1 the designer may state that a match to the left-hand shape (shape a) is a rectangle with the same ratio of lengths of sides. Another designer may want to match any rectangle or even any quadrilateral. Each of these cases increases the number of parameters.

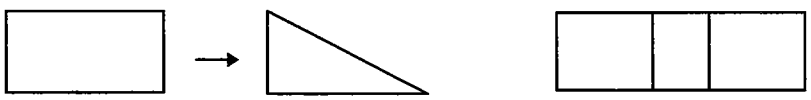


Figure 1. Designer must determine what is a match of shape a in shape c.

The general shape grammar interpreter introduced in this paper differs from other interpreters. Previous approaches used a combination of Euclidean transformations (that is, translation, rotation, and isotropic scaling) and required that the transformations be applied to the entire shape when performing shape searches. Our approach supports general parametric relations by decomposing a shape into a hierarchy of subshapes thereby allowing for any parts of the shape to be transformed with any possible transformation. The ordering of the shapes into a hierarchy provides an efficient shape-search order. In this paper a default hierarchy based on our own observation and interpretation of engineering and architectural grammars is introduced that is applicable in many cases and efficient in general; other decompositions could be applicable as well and still use the basic approach.

Krishnamurti (1980) was the first to describe the algorithms necessary for creating a shape grammar interpreter or implementing any shape grammar by using subshape recognition. This was limited to using the Euclidean transformations and applying those transformations to the entire shape. Eventually these ideas were extended to include three dimensions (Krishnamurti and Earl, 1992) with the same restrictions. Flemming (1987) has created a two-dimensional parametric shape grammar interpreter intended for architectural applications; however, it does not use subshape recognition for determining rule applicability. Tapia (1999) created a two-dimensional shape grammar interpreter with a well-designed graphic user interface that used subshape recognition and addressed issues of rational shapes and implementation efficiency but, again, the work was limited to the Euclidean transformations applied to entire shapes.

Previously implemented engineering shape grammars have been hardcoded meaning that a computer program was created entirely to implement a single set of rules. These grammars often did not rely on shape addition and subtraction for rule application or subshape recognition for rule applicability in their implementations. In order to determine if a rule can be applied, searches are instead performed for necessary labels. The coffeemaker grammar (Agarwal and Cagan, 1998) was hardcoded as a web-based Java applet with extensive use of labels. Other hardcoded engineering shape grammars include the MEMS grammar (Agarwal et al, 2000) that also relied on labels and the truss grammar (Shea and Cagan, 1997) that used only triangles and did not take advantage of potential emergent shapes.

Because of limited technology, implemented engineering shape grammars have not been able to include parametric subshape recognition until now. The technology presented in this paper allows designers to write innovative grammars that rely on parametric subshape recognition and that can have emerging characteristics resulting in creative designs.

3 Shape decomposition

In order to implement a parametric shape grammar three operations are necessary: parametric subshape recognition, shape addition, and shape subtraction. The methods for addition and subtraction have been well documented previously (Krishnamurti, 1980), while a method for parametric subshape recognition is introduced in this paper.

In our approach, parametric subshape recognition is achieved through a deomposition of shapes into a hierarchy of subshapes ordered by their decreasing restrictions. Instances of each of the subshapes are individually located in the design shape and then reconstructed to form an instance of the entire shape. The basis for the hierarchy of subshapes can be specified by the designer or based on the default spatial relations established in this paper that come from architectural and engineering knowledge. The levels of the hierarchy are defined so that the most constrained lines of a shape are those lines that the designer intended exactly. These most constrained lines have

specified parametric relations to other line segments and those relations, if altered, will compromise the designer's intentions. Conversely, the lowest level of the hierarchy, which contains the least constrained line segments, implies only a specific connectivity between line segments, implying that a more extensive search will be necessary.

This section lays out the details of the spatial relations used to form our default hierarchy of shape decomposition. Consider the shape in figure 2 as a starting point. The goal is to capture the intent of the designer that created this shape to define a part of a rule. The lines in the shape that the designer specified exactly must be separated from the lines in the shape that were intended as a general scheme. Separating the parts of the shape based on the designer's intent creates a hierarchy of subshapes that parametrically represents the whole shape. There is a limited set of transformations that can be applied to a set of straight lines that lie in a single plane. The possible transformations are translation, rotation, scaling, and shearing. In order to choose the features that are important compare the square in figure 2 with the shapes in figure 3.

Figure 2 Decomposition beginning.

Figure 3. Four different quadrilaterals.

The shapes in figure 3 are a quadrilaterial (a), a rhombus (b), a trapezoid (c), and a rectangle (d). All five of these shapes are quadrilaterals, but they each have features that make them unique. The square (figure 2) and the rectangle [figure 3(d)] have lines that intersect at right angles to each other. The square and the rhombus [figure 3(b)] each have reflection symmetry about imaginary lines of symmetry through the diagonals of each shape, as seen in figure 4. For our purposes, a line of symmetry is defined as being a bisector of a pair of line segments; the intersection point may be on the line segments or the intersection of the infinite line carriers of the line segments or between parallel lines.

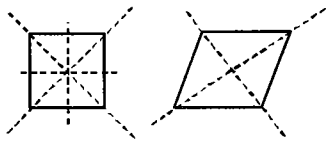


Figure 4. The symmetry in a square and a rhombus.

So despite an equal number of line segments, the shapes are distinct. Of the possible transformations, some will not destroy certain features and some will. For example, no amount of translation or rotation will destroy a specific feature, but only move the features from place to place, but shearing will eliminate any perpendicular intersections. Anisotropic (different scaling factor in all directions) scaling will destroy symmetry unless the scaling is along or perpendicular to the line of symmetry. Isotropic scaling and reflection (scaling by a factor of -1) does not, however, affect the symmetry of a shape. From this simple example, groupings can be formed. The set of translation, rotation, and isotropic scaling (same scaling factor in all directions) will preserve all symmetry and perpendicular intersections. Translation, rotation, and anisotropic scaling will preserve the perpendicular intersections and symmetry in shapes that have parallel lines of symmetry or only one line of symmetry when the axes of scaling are chosen properly. Translation, rotation, anisotropic scaling, and shearing can be applied to any intersecting line pair.

This leads to a four-level hierarchy or decomposition into subshape groupings as shown in table 1, which constitutes the default case. The default hierarchy is based on the assumption that the designer will specify a rule in such a way the relations between line segments present in the rule require. Upon decomposition, line segments will be placed into the most highly constrained subshape grouping for which they qualify.

Table 1. The four subshape groupings of the default hierarchy.

Subshape grouping	Feature	Transformation
s_1	(1) Lines that intersect perpendicularly and are the same length, (2) lines that are symmetric to two or more lines that are not parallel	Translation, rotation, isotropic scaling
s_2	(1) Lines that intersect perpendicularly, (2) lines that are symmetric to one line, or (3) more than one line that are parallel	Translation, rotation, anisotropic scaling
s_3	Intersecting lines	Translation, rotation, anisotropic scaling, shearing
s_4	None	None

Grouping s_1 consists of the most highly constrained line segments that intersect perpendicularly and are the same length. Additionally the s_1 grouping also contains any line segment that is symmetric to two or more other line segments and the line segments are not parallel. Two examples of lines that meet the symmetry criteria of groupings s_1 are the sides of a square and the legs of an equilateral triangle as seen in figure 5 along with other shapes that meet the s_1 criteria.

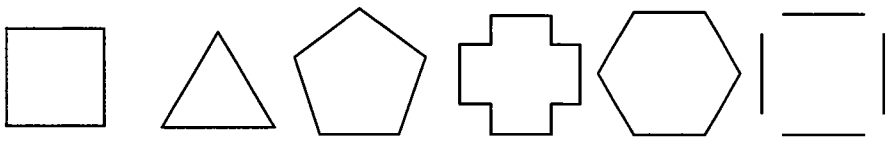


Figure 5. Example of s_1 lines.

Grouping s_2 contains the line segments that intersect perpendicularly. Any line segment that is symmetric to another line segment is also included in grouping s_2, including line segments that do not intersect. Grouping s_1 is a subset of s_2. Some examples of s_2 lines (that are not s_1 also) are the sides of a rectangle, the two equal length legs of an isoceles triangle, and the other shapes in figure 6.

Figure 6. Examples of s_2 lines.

Grouping s_3 contains the line segments that intersect. This makes groupings s_1 and s_2 subsets of s_3. Examples of three lines that are s_3 and not s_1 or s_2 are the three line segments that make up the triangle in figure 7 (over). The other shapes in figure 7 are also members of the s_3 grouping.

The line segments in grouping s_4 have no discernable spatial relation to any other line segments, essentially those lines not found in s_1, s_2, or s_3. An example can be seen in figure 8 (over). A single line segment requires additional information and

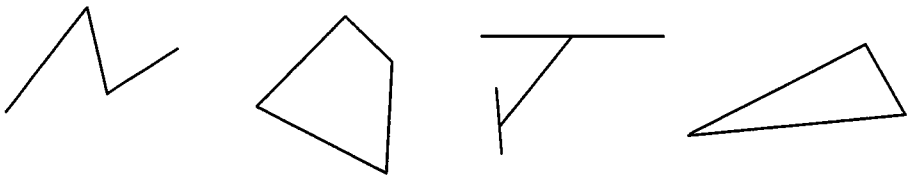


Figure 7. Example of s_3 lines.

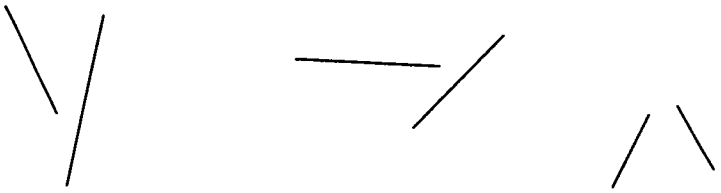


Figure 8. Example of s_4 lines.

assumptions to be made in order to find a match. It is likely that the use of an s_4 line as a left-hand shape in a rule would also include labels that give the line meaning. It is also possible that the user requires endpoint matching to reduce the number of shape matches to a finite value.

Having established the set of parametric features that define the default hierarchy, it should be restated that the parametric relationships that define each part of the hierarchy of shapes can be customized by the designer to reflect his or her need. This method of shape decomposition has now enabled a shape to be divided into pieces, turning a difficult parametric subshape recognition process into a manageable task. The subshape pieces of a whole shape can now be dealt with individually when the subshape recognition process is performed and yet allow an entire shape to be parametrically recognized through their combination.

4 Shape decomposition example

Every shape consisting of lines can be broken down into subgroups as introduced in section 3. By using the default decomposition outlined in section 3, the shape in figure 9 will be broken down into four subshapes as an illustration of the technique.

The approach first searches for all s_1 type lines in the original shape and then combines them to form subshape s_1. Subshape s_1 is then subtracted from the original shape and the process repeats. The shape s_2 lines are found from the remaining shape. The s_2 lines are combined to form the shape s_2 and are then subtracted from the current shape. The process continues through the hierarchy, which includes subshape groupings s_3 and s_4 in the default.

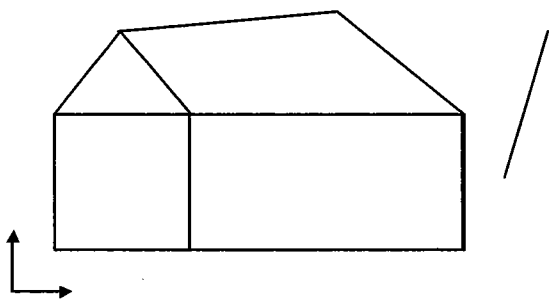


Figure 9. Shape decomposition example.

First a search is performed to find all lines of symmetry. By examining the example shape, the lines of symmetry (dashed lines) can be drawn according to the definition (figure 10).

Each line in the square is symmetric with the two lines of the square that it intersects. The same is true for each of the legs of the triangle. Each of the line segments in the square and the equilateral triangle is symmetric with more than one line. This meets the requirement to fit in the s_1 subshape grouping. The subshape s_1 (figure 11) can now be subtracted from the example shape (figure 12) and the search for the s_2 subshape can begin.

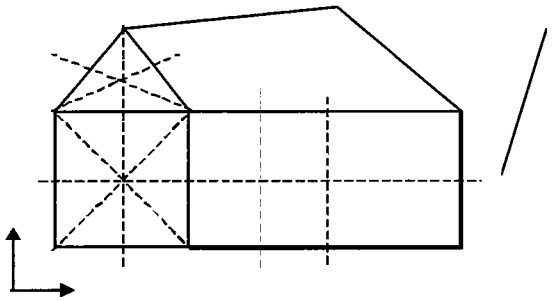


Figure 10. Symmetry in the example shape.

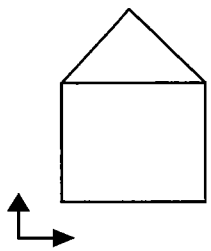


Figure 11. The s_1 subshape found in the example shape.

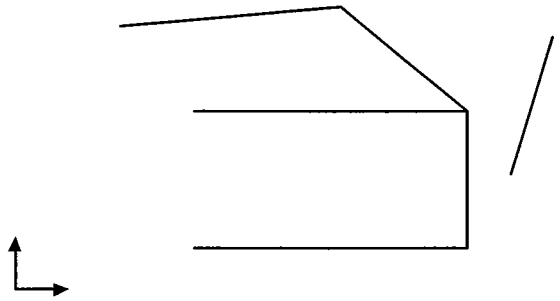


Figure 12. The example shape after the s_1 shape has been subtracted.

The remaining symmetric lines are symmetric to only one other line, placing them in the s_2 grouping. There are also two perpendicular intersections present, made up of three line segments. These lines make up the subshape s_2 (figure 13).

The subshape s_2 can now be subtracted from the example shape (figure 12), leaving the combination of subshapes s_3 and s_4 (figure 14, over).

The subshape s_3 can now easily be found in the example shape. Subshape s_3 (figure 15, over) is simply the intersecting line segments, which leaves the line farthest to the right in the s_4 (figure 16, over) subshape grouping. The original design shape (figure 9) can be reconstructed by combining the four subshapes.

Figure 13. Subshape s_2 from the example shape.

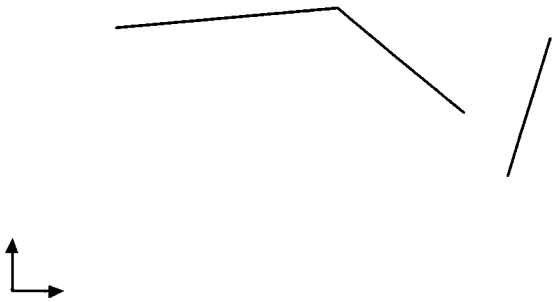


Figure 14. The example shape after shapes s_1 and s_2 have been subtracted.

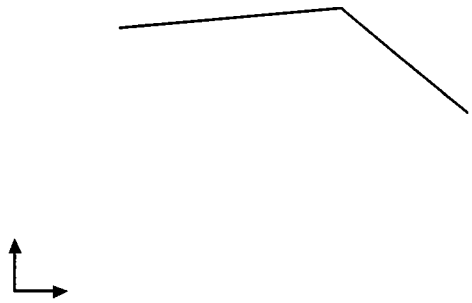


Figure 15. Shape s_3 found in the example shape.

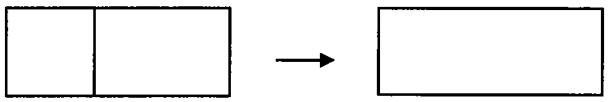
Figure 16. Shape s_4 found in the example shape.

The presentation of the hierarchical decomposition of the shape in figure 9 is done in a manner that implies that the shapes that define rules are decomposed in an effort to determine the intentions of the rule writer. In fact, the rule writer has knowledge of and even determines the hierarchy while writing the grammar rules, allowing the writer to specify an entire shape and its decomposition based on that hierarchy of spatial relations.

5 Shape recognition

Section 3 established the technique to decompose a shape into subshapes grouped by their properties. This method allows for parametric subshape recognition, which is needed to determine if and where a rule is applicable.

Subshape recognition is accomplished by repeating a three-step process for each of the subshapes of the decomposed left-hand side shape (shape *a* in figure 17) of the rule. The three steps of the process are (1) finding subshapes in the active shape, (2) subtracting the subshapes from the active shape, and (3) then adding labels to (a) the overlapping points (or projected intersections if needed) of the decomposed left-hand



shape a shape b

Figure 17. Rule $a \rightarrow b$.

shapes and (b) points in the active shape equal in location to the transformed, labeled points in the decomposed left-hand shape.

The process begins with the most constrained subshape and progresses down the hierarchy of subshapes towards the least constrained. Beginning with the most constrained subshape, the subshape is searched for in the design shape c, and those subshape matches make up the set of shapes S. Subshape matches are found by applying appropriate transformations to distinct points (line segment and projected line segment intersections) in the subshape in order to match distinct points in shape c. The shapes in the set S are each subtracted from the shape c, producing another set of shapes C. In order to maintain the connectivity between subshapes, shared points between the decomposed parts of shape a are labeled. The subshapes of a decomposed shape will overlap each other only at points, if at all, because the definition of the decomposition of shapes requires that the subshapes share no line segments. Additionally, the points in shape c that coincide with the labeled points in the subshape after transformation are also labeled. The process begins again with the next subshape in the hierarchy; this time the subshape search is performed in each of the altered, labeled versions of the shape c in the set C. The process ends and the rule can be applied when all of the decomposed parts of the left-hand shape have been found. When one of the shape searches finds no subshapes then the rule cannot be applied.

The advantage of this approach is that this decomposition maintains an interpretation of the designer's intent throughout the rule applications and the eventual design generation. To aid in explanation, the shape recognition strategy is presented in pseudo-code form in figure 18.

As an illustration of the shape-recognition process, the rule in figure 17 will be applied to the design shape in figure 19 (see over). In order to apply the rule $a \rightarrow b$

Notes for pseudo-code:
Lower case represents a single shape
UPPER CASE represents a set of shapes

Given shape a, find an instance of shape a in shape c_0 by using a user-defined shape hierarchy of n levels
  Decompose shape a into n subshapes where a_i, $i = 1, \ldots n$, corresponds to subshape grouping s_i from the user's hierarchy

  Shape c_0 is placed in the set of shapes C_0

  $m = 0$
  For $i = 1$ to n
    For $j = i+1$ to n
      Mark shared points between a_j and a_i with label $x_{i,m}$
      $m = m+1$
  For $k = 1$ to n
    For each shape c that is a member of the set of shapes C_{k-1}:
      Search for instances of shape a_k in shape c
      Store all instances of a_k in c whose labels at levels 1, ..., k match in S_k
      For each shape s that is a member of the set of shapes S_k:
        $c' = c - s$
        Copy the share point labels between c' and s
        Put c' in set of shapes C_k
        If $k = n$
          For each shape c_k that is a member of the set of shapes C_k:
          An instance of the original shape is equal to $c_0 - c_k$

Figure 18. Shape recognition pseudo-code.

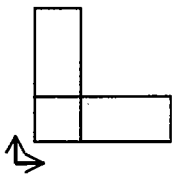


Figure 19. Shape c_0, coordinate system provides spatial reference.

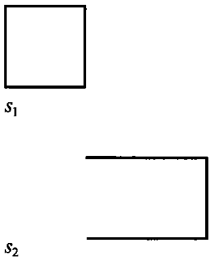


Figure 20. The decomposition of shape a in figure 17 where s_3 and s_4 are null.

(figure 17) to the shape c_0 (figure 19), shape a must be found to be a parametric subshape under various transformations (τ) of shape c_0. By using the technique from section 3, shape a is decomposed into the four subshapes where $a = s_1 + s_2 + s_3 + s_4$ (figure 20).

The shape recognition process begins with the most highly constrained subshape that is not null and skips any downstream subshape that is null. This produces a more efficient search because the more highly constrained shapes have fewer possible transformations. For example, if s_1 and s_3 are null, then the process will begin with s_2 and progress to s_4. In the example, shape a is decomposed into two subshapes, s_1 and s_2. For the purpose of explanation, the general mathematical technique for any combination of subshapes will be provided.

The transformations appropriate for subshape grouping s_i are represented by τ_i and those subshapes found in shape c are placed in the set S_i, where i is the subshape grouping level.

Beginning with the most highly constrained subshape, which is s_1 in the example, find

$$S_i = \{\tau(s_i) \leqslant c | \forall \tau \in \tau_i\}.$$

The symbol $\leqslant$ represents the subshape operation, where $x \leqslant y$ would determine if shape x is a subshape of shape y. The shapes in figure 21 are the set of shapes that make up S_1. The four shapes are geometrically equal but are found differently by the rotation of s_1 four different ways (0°, 90°, 180°, and 270°). The same shapes can be found with different transformations, but the repeated shapes are removed to improve efficiency.

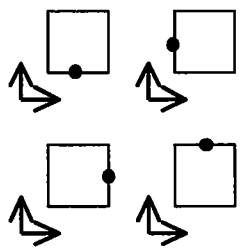


Figure 21. The set of shapes S_1; the dots show orientation.

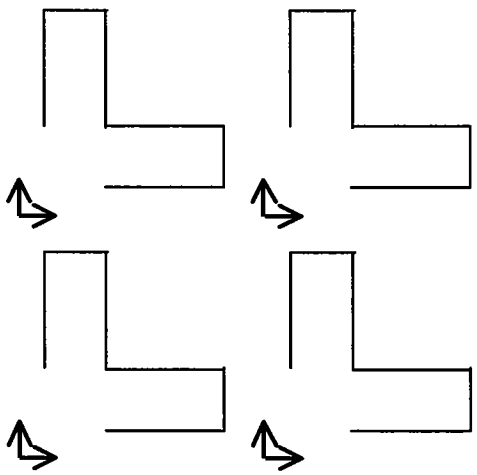


Figure 22. Shapes in the set C_1.

Having found the set of shapes S_1, find the set of shapes C_1 (figure 22) resulting from the subtraction of the shapes in S_1 from the shape c_0; that is,

$$C_i = \{c_0 - s | \forall s \in S_i\}.$$

By the definition of the shapes S_1, S_2, S_3, and S_4, it can be seen that no two of them will share any common line segments. They will, however, share common line segment endpoints. In order to mark points of shape intersection, the labeling operation g(p), where p is a point, can be performed. The labeling operation associates a marker (a letter, word, or symbol) with a geometric entity in order to provide nongeometric information. Beginning with shapes s_1 and s_2, label the points where they intersect. The intersecting points between shapes s_1 and s_2 are labeled on the shapes in figure 23.

The labeling operation must also be performed on the remaining portions of the design shape c_0 (found in the set C_1) in order to maintain the connectivity between the occurrences of s_1 found in c_0 and the instances of s_2 that will be searched for next. The labeled points of shape s_1 that coincide with the points in the member shapes of the set C_1 will be marked, where s_1 is transformed according to the transformation that made it a subshape of c_0. The equality of points is based on point location. For this example, the shapes in C_1 are shown in figure 24, after being marked. It is possible

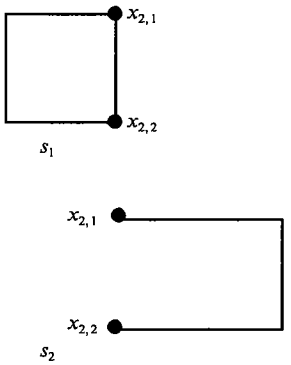


Figure 23. shapes s_1 and s_2 with shared endpoints labeled.

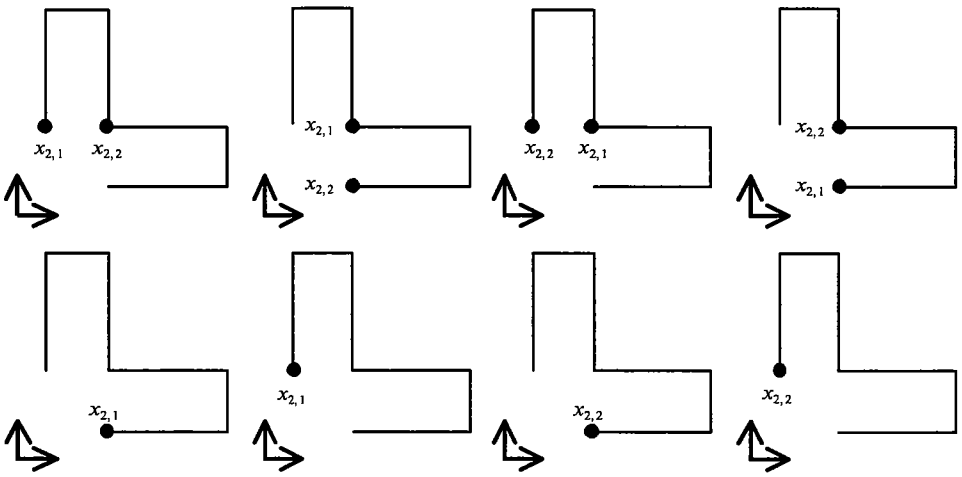


Figure 24. The shapes in set C_1 with shared points labeled.

and valid to have a point marked several times because of intersections with lines from more than one subshape grouping.

Continuing onto the next highest constrained subshape, find C_i such that

$$\forall c \in C_{i-1},\ C_i = \{c - \tau(s_i) | \tau(s_i) \leqslant c,\ \forall \tau \in \tau_i\},$$

where i is the number of the subshape grouping. In the example, i would be equal to 2. All of the subshape groupings of shape a have been searched for in shape c_0 leaving the set of shapes C_2. The shape search process concludes when the subshape search has been performed for the least constrained shape. The set of shapes A, which are instances of shape a are revealed by

$$A = \{c_0 - c | \forall c \in C_i\}.$$

The shape search resulted in four matches, of which two (figure 25) were repeated shapes. If there were additional subshapes of shape a (s_3 and/or s_4 for the default hierarchy) the search would continue, following the process previously outlined until the subshape groupings are exhausted or subshapes cannot be found for a grouping.

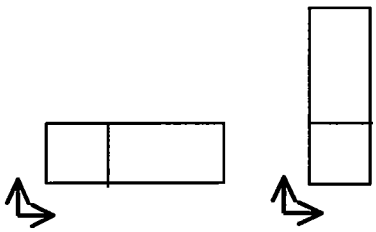


Figure 25. Instances of shape a.

6 Rule application example

Once the left-hand shapes have been recognized, the rule can be applied wherever appropriate. The process involves subtraction of the left-hand shape and then adding in the right-hand shape. Consider the following rule, an example of parametric rule application. The left-hand side of the rule in figure 26 has constraints that limit the shape search to perpendicular intersections, but with no lines being symmetric to more than one line, the left-hand shape is entirely in the s_2 subshape grouping. The goal is now to find shape a in the design shape (figure 27). The general parametric method described in section 5 is able to find twelve locations to apply the rule to three different subshapes in the design shape (figure 28).

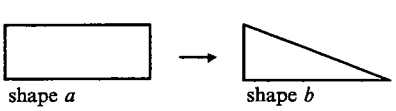


Figure 26. The rule to be applied

Figure 27. The design shape.

The right-hand side of the rule, shape b in figure 26, must then be transformed before being added to the difference of the design shape and the instance of the left-hand shape found. The perpendicular intersections of the triangle's legs of shape b places those line segments in the s_2 grouping. The transformations of the s_2 line segments of shape b come directly from those used to find a match for shape a. The hypotenuse of the triangle is transformed as an s_2 line as well because its endpoints are shared with other s_2 line segments. Applying the rule in the twelve different possible ways results in the shapes in figure 29.

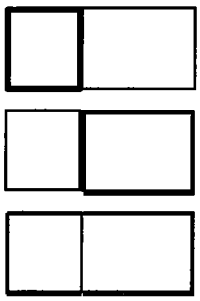


Figure 28. Locations in the design shape where the rule can be applied are in bold.

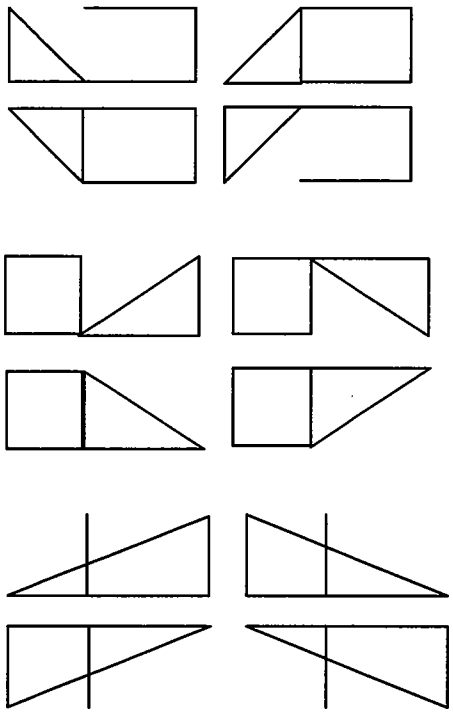


Figure 29. The design shape after the rule has been applied in all twelve possible ways.

There could exist a right-hand shape, which has line segments in a grouping not in the left-hand shape. Other means must be used to determine the transformation of these right-hand points. The logic behind creating a rule indicates that there is a relation between the left and right sides of the rule through their shapes. One method is if both shapes are defined on the same coordinate system, points that exist in both sides of the shape can be transformed similarly. If a point in the right-hand shape of a rule cannot be related to the left side, it could require additional user input to instantiate the shape. An interactive shape grammar system would prompt the user to locate the point, applying any constraints implied by the geometry.

The right-hand side of the rule in figure 30(a) (over) includes line segments in the s_1 and s_2 groupings whereas the left-hand side has only s_1. The s_1 and s_2 subshapes that constitute the right-hand shape contain two shared points (labeled x). The two shared points of the s_2 shape are constrained to be transformed according to the

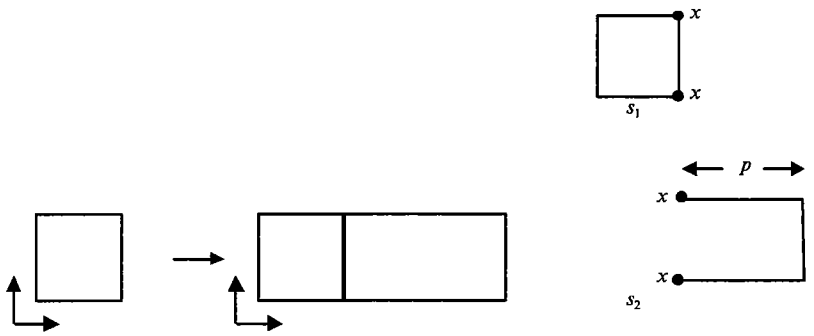
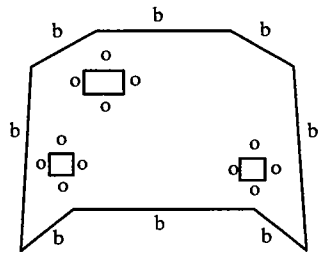


Figure 30. A rule requiring input for right-hand side instantiation.

transformation of the s_1 shape. This of course dictates a portion of the transformation of the nonshared points of the s_2 shape, leaving only a single scaling factor (which sets length p) to be determined. The optimization scheme or designer working interactively with the interpreter determines the final parameter.

7 Inner hood panel shape grammar example

The inner hood grammar was written to aid in the design of the inner hood panels for General Motors cars. It is a two-dimensional parametric grammar, consisting of twenty-eight rules, that is concerned with the layout of the structural beams of the inner hood panel as well as operational features such as the striker, hinges, etc. One difficulty in designing the hood panel is the limited space available for structural support because of the internal packaging (engine, shock towers, battery, etc). The shape in figure 31 is the initial shape for the hood grammar. The lines labeled b represent beams and the lines labeled o are the obstacles that the beams must be placed around. The rules in figure 32 are two of the rules from the grammar used to create and modify the structure of the inner hood. Rule 1 will place a beam between two other beams. Rule 2 recognizes the intersection of a beam and an obstacle. The application of rule 2 moves the endpoint of the intersecting beam to avoid the obstacle. The lines in the left-hand shape of rule 1 fall into the s_4 grouping, meaning that any



Figurer 31. The initial shape.

Figure 32. (a) Rule 1; (b) rule 2.

lines with matching labels will be valid matches. The lines in the left-hand shape of rule 2 fall into the s_3 grouping, meaning that a shape match will require matching labels and line segment connectivity. The complete inner hood shape grammar was implemented in a few hours with a text-based interface. The implemented grammar has been used to generate both standard and novel hood designs. McCormack and Cagan (2002) present the complete grammar and example applications.

By using only rule 1, a number of good designs can be created such as shown in figure 33, but the application of the rule could also result in a beam passing through an obstacle. The designer could take care so as to not let this interference occur when working interactively with the grammar, but there are situations where interference could arise, such as if an obstacle is moved, if a beam is being placed by a designer as a quick sketch with the intention of correction later, or if the rule is being applied by another computer program, where the knowledge to avoid obstacles is not inherent. Hence the presence of rule 2 allows for infeasible designs to be corrected.

Figure 34 shows a situation where a beam modification rule such as rule 2 is needed. To apply the rule a match must be found for the left-hand shape. The description of what is a match for rule 2 would read as: a beam intersects another beam at their endpoints. One of the beams must also intersect an obstacle. That intersection point may not be at the endpoint of either line segment. A method to perform the search for the left-hand shape of rule 2 represents the shape with more line segments than are present in the maximal line representation. Line segments are divided into the minimum number of line segments such that line segments intersect with other line segments only at endpoints. The left-hand side of rule 2 represented with this convenient set of line segments in shown in figure 35, where the line segments are numbered for reference.

The search begins with line segment 1 and matches will be any line segment labeled o. Line segment 2 is searched for next with the requirements that it is collinear with the first line segment and that they share an endpoint. The search for line segment 3 is next; matches must be labeled b and share an endpoint with line segments 1 and 2.

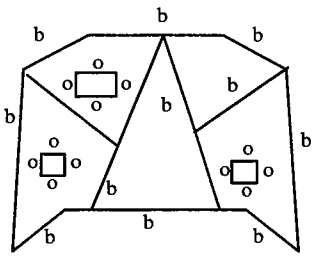


Figure 33. Hood resulting from multiple applications of rule 1.

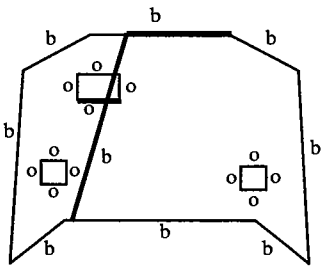


Figure 34. Beam interfering with obstacle is corrected through application of rule.

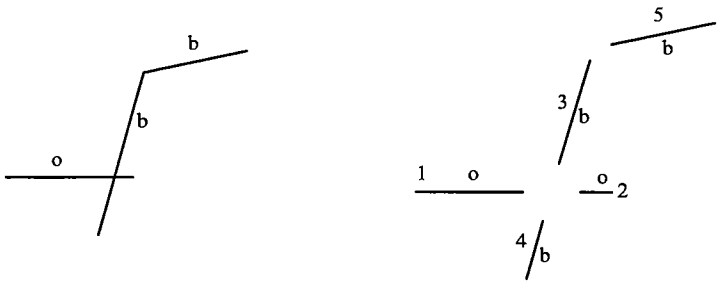


Figure 35. Left-hand shape of rule 2 broken down into a convenient set of line segments for an s_3 shape search. Line segments are numbered for reference.

The process continues until all line segments are accounted for, leaving the user with a shape match. Figure 36 shows a shape match for rule 2 with the line segments of the matching shape represented in a similar manner to the rule's left-hand shape in figure 35. Differences and improvements to the s_3 matching routine could be made while the meaning and significance of the s_3 grouping remains unchanged.

Application of rule 2 to the bold subshape produces the corrected hood in figure 37. Rules such as rule 2, which is meant to match any beam interfering with an obstacle, could not be implemented without a parametric subshape recognition scheme, such as the one presented in this paper, to match a variety of possible obstacle intersections that could occur.

An alternative but less preferable set of two rules composed of s_2 and s_3 lines can be introduced to replace rule 2 in figure 32, as shown in figure 38. These rules are given only to show the use of s_2 lines in the context of the interpreter. Two rules are necessary to account for the situation where the beam intersects opposite and adjacent sides of an obstacle.

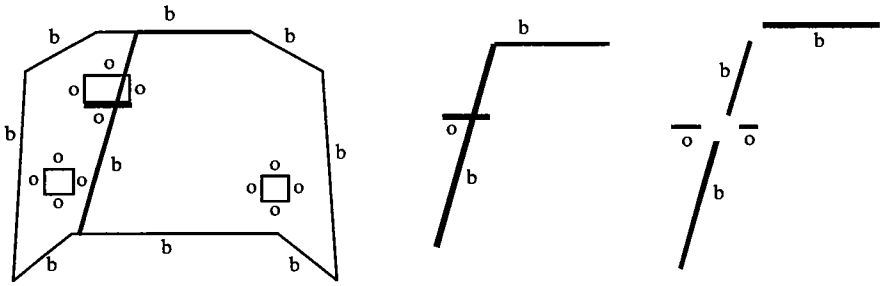


Figure 36. Matching subshape found in the sample hood broken down into line segments.

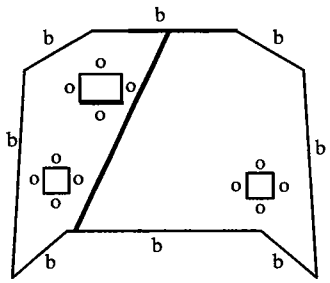


Figure 37. The result of the application of rule 2 to the shape in figure 33.

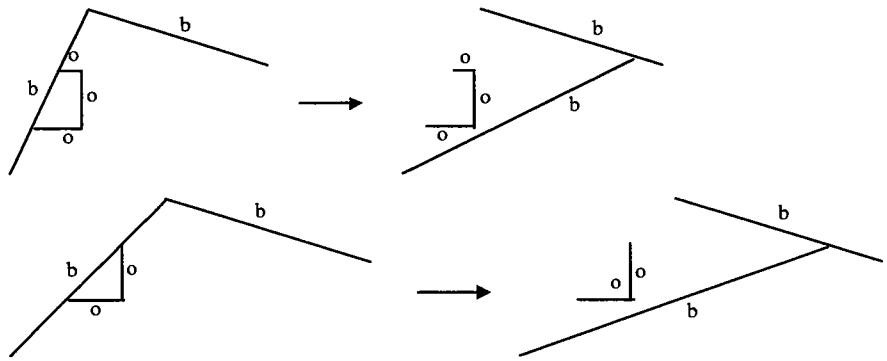


Figure 38. Alternative rules to rule 2.

8 Ice-ray example

The ice-ray grammar is a parametric shape grammar written by Stiny (1977) that can generate lattice patterns that are found in the traditional architecture of China.

A modified rule set was implemented with our interpreter in just a few minutes. Rules of the original ice-ray grammar were written such that the instantiated right-hand shape was divided into halves of approximately equal area by the internal dividing line. Our implementation randomly placed the dividing line endpoints along the adjoining line segments. The initial shape is found in figure 39 and the four rules are found in figure 40.

Upon examining each of the rules it is clear that the left-hand side of each rule falls into the s_3 shape grouping because of the lack of symmetry and perpendicular intersections of the shapes. Therefore, in general terms each rule can be applied if a shape with the proper number of sides is available. For example, rule 1 is applicable if any triangle can be found, rules 2 and 3 can be applied if any quadrilateral can be found, and rule 4 can be applied if any five-sided shape can be found. The progression of

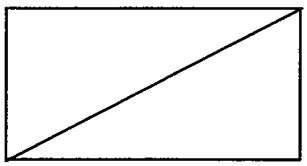
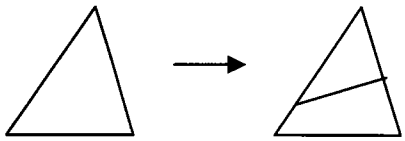
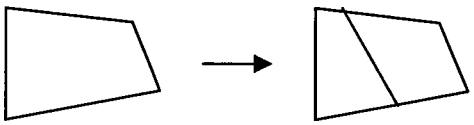


Figure 39. The initial shape for the ice-ray grammar.

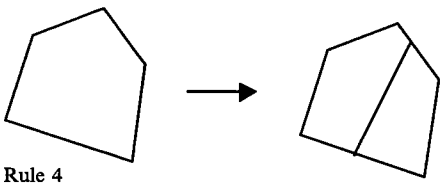


Rule 1



Rule 2

Rule 3



Rule 4

Figure 40. The four rules of the ice-ray grammar.

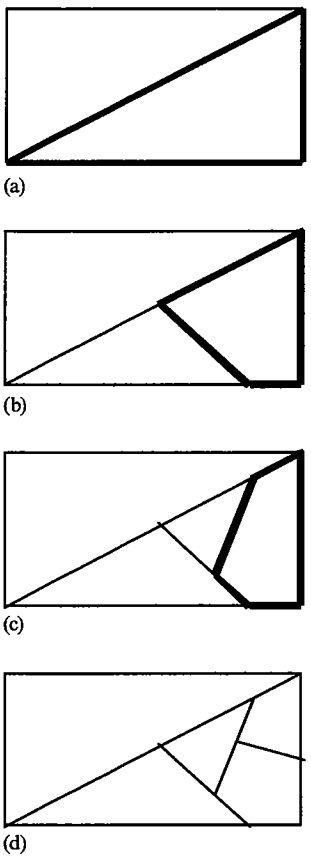


Figure 41. A series of rule applications: (a) initial shape, (b) apply rule 1, (c) apply rule 3, (d) apply rule 4.

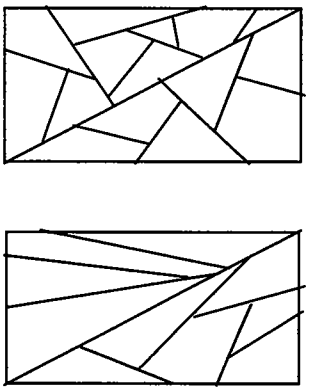


Figure 42. Examples of completed ice-ray patterns.

shapes in figure 41 comes from the application of a series of rules that we applied using the parametric shape grammar interpreter. The subshape that we chose to apply the rule to is highlighted in bold.

The ice-ray grammar, despite its simplicity, is an excellent example of a parametric grammar. The rules were chosen randomly from those that were applicable, as were the parameters, and after a number of rule applications, final designs such as those in figure 42 were produced. Many other parameters could have been used for each rule application and, generally, more than one rule was applicable at each stage. Note that, as the generation sequence evolves, shapes that were not explicitly specified emerge, enabling new shapes to be generated.

9 Concluding remarks

This paper introduced a shape grammar interpreter using parametric subshape recognition. The interpreter has been implemented in $C++$ and compiled and run on a Windows workstation. Unlike previous interpreters that are limited to nonparametric subshape matching or parametric shape matching (no emergence), our interpreter can work on general parametric features through the decomposition technique allowing subshape matching. A decomposition based on a default set of parametric relations was derived for general use in engineering, architectural, or other applications, but hierarchies based on the specific interests of a designer may be preferred and would be valid.

Ordering the groupings of parametric relations into a hierarchy of decreasing specificity results in a more efficient subshape search order. Although in the worst case an exhaustive check of all distinct (that is, intersection and end) points is required, in practice labels restrict the number of points that must be examined for any shape matching process. Other steps could be taken to improve performance such as restructuring the subshape groupings.

The need for parametric subshape recognition in engineering shape grammars has been considered. There is a lack of required subshape recognition in previously written engineering shape grammars, but it is unclear whether the designer avoided the issue because of the difficulty in the implementation or if subshape recognition was unnecessary. The introduction of this parametric shape grammar interpreter using subshape recognition will enable a new breed of engineering and other shape grammars that rely on form and are not reduced to an exercise in label searching. Additionally, this process opens up the possibility of supporting creativity in the design process by enabling shape emergence.

The ideas presented in this paper form the basis for the parametric interpreter to be extended into three dimensions and to include curves. We foresee this as the next step in the continuation of our work.

Acknowledgements. The authors would like to thank the National Science Foundation under grant DMI-9713782 and General Motors for its support of this work. The authors would also like to thank Ramesh Krishnamurti for his in-depth comments and suggestions as well as Su Yin for her discussions and Jim Elshoff and Mary Pickett for their input on the hood grammar.

References

Agarwal M, Cagan J, 1998, "A blend of different tastes: the language of coffeemakers" *Environment and Planning B: Planning and Design* **25** 205 – 226

Agarwal M, Cagan J, 2000, "On the use of shape grammars as expert systems for geometry based engineering design" *Artificial Intelligence in Engineering Design, Analysis and Manufacturing* **14** 431 – 439

Agarwal M, Cagan J, Stiny G, 2000, "A micro language: generating MEMS resonators by using a coupled form – function shape grammar" *Environment and Planning B: Planning and Design* **27** 615 – 626

Brown K N, McMahon C A, Sims Williams J H, 1994, "A formal language for the design of manufacturable objects", in *Formal Design Methods for CAD (B-18)* Eds J S Gero, E Tyugu (North-Holland, Amsterdam) pp 135 – 155

Flemming U, 1987, "The role of shape grammars in the analysis and creation of designs", in *Computability of Designs* Ed. Y E Kalay (John Wiley, New York) pp 245 – 272

Gips J, Stiny G, 1972, "Shape grammars and the generative specification of painting and sculpture", in *Information Processing 71* Ed. C V Freiman (North-Holland, Amsterdam) pp 1460 – 1465

Gips J, Stiny G, 1980, "Production systems and grammars: a uniform characterization" *Environment and Planning B: Planning and Design* **7** 399 – 408

Krishnamurti R, 1980, "The arithmetic of shapes" *Environment and Planning B: Planning and Design* **7** 463 – 484

Krishnamurti R, Earl C F, 1992, "Shape recognition in three dimensions" *Enviornment and Planning B: Planning and Design* **19** 585 – 603

McCormack J, Cagan J, 2002, "Designing inner hood panels through a shape grammar-based framework" *Artificial Intelligence in Engineering Design, Analysis and Manufacturing* **16**(4) forthcoming

Shea K, Cagan J, 1997, "Innovative dome design: applying geodesic patterns with shape annealing" *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing* **11** 379 – 394

Stiny G, 1977, "Ice-ray: a note on the generation of Chinese lattice designs" *Environment and Planning B: Planning and Design* **4** 89 – 98

Stiny G, 1980, "Introduction to shape and shape grammars" *Environment and Planning B: Planning and Design* **7** 343 – 351

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