THE DEFINITIVE, PEER-REVIEWED AND EDITED VERSION OF THIS ARTICLE CAN BE FOUND IN:
Yue K, Krishnamurti R, 2013, "Tractable shape grammars" *Environment and Planning B: Planning and Design* **40**(4) 576 – 594
[DOI http://dx.doi.org/10.1068/b38227]

Tractable shape grammars

Kui Yue

kuiyue@microsoft.com, Microsoft, Redmond, WA 98052

Ramesh Krishnamurti [(1)]

ramesh@cmu.edu, Carnegie Mellon University, Pittsburgh, PA 15213-3890

February 2010 (Revised) December 2011, May 2012, April 2013

Abstract: This paper explores the theoretical basis for a concept of "computation-friendly" shape grammars, through a formal examination of tractability of the grammar formalism. Although a variety of shape grammar definitions have evolved over time, it is possible to unify these to be backwards compatible. Under this unified definition, a shape grammar can be constructed to simulate any Turing machine from which it follows that: a shape grammar may not halt; its language space can be exponentially large; and in general, its membership problem is unsolvable. Moreover, parametric subshape recognition is shown to be NP. This implies that it is unlikely in general to find a polynomial time algorithm to interpret parametric shape grammars, and that more pragmatic approaches need to be sought. Factors that influence the tractability of shape grammars are identified and discussed.

Keywords: shape grammar definitions, Turing machine, parametric shape recognition, computational complexity, tractability

---

[(1)] Author for correspondences

1 Introduction

Implementing a parametric shape grammar interpreter has long been considered difficult (Chau et al., 2004; Gips, 1999), for reasons not always apparent. In this paper we consider the issue of implementation through a formal examination of tractability of the shape grammar formalism for which we invoke both formal language theory and asymptotic analyses of algorithms. Specifically, a computational problem is deemed *tractable* whenever there is a polynomial algorithm; and *intractable* when it requires super-polynomial, typically, exponential, time. As a result we are better positioned to formulate a concept for a shape grammar that is "computation-friendly."

The basic formalism of a shape grammar has remained largely unchanged, although, over time, there have been changes in definition and development. Factors that have influenced these changes relate to the scope of permissible shape elements and possible augmentations. The early focus in shape grammars was on two-dimensional shapes made up from finite lines, specified, representationally, in terms of maximal elements (Stiny, 1980a). In the kindergarten grammar (Stiny, 1980b), basic shapes were extended to three-dimensional rectilinear solids, albeit drawn as line shapes. In a subsequent paper, Stiny (1991) considered shapes made up from points, lines, planes, or solids as the main elements for a shape grammar. Krishnamurti (1992a) examined shape arithmetic on shapes made up from finite planes; with Earl, he considered subshape recognition for three-dimensional shapes under linear transformations (Krishnamurti and Earl, 1992); and with Stouffs, he extended the arithmetic to higher-dimensional shape algebras (Krishnamurti and Stouffs, 2004), described algorithms for three-dimensional shape arithmetic and analyzed their

computational complexity (Stouffs and Krishnamurti, 2006) and considered subshape recognition over the Cartesian products of differently dimensioned shapes (Krishnamurti and Stouffs, 1997). A three-dimensional shape grammar implementation based on a commercial solid modeling kernel has been described in Piazzalunga and Fitzhorn (1998), and grammars over curves have been considered by several authors (Chau, 2002; Jowers et al., 2004; Jowers&Earl, 2010, 2011; McCormack and Cagan, 2003; Prats et al., 2004).

Geometric shapes can be augmented by symbols, numbers, attributes, in general, *weights* (Stiny, 1992), in this way, connecting shapes of various kinds (Stiny, 1991). Shapes so augmented can be further extended, by open parameters, such that a family of shapes can be defined. A shape $s \approx s(x)$ can be associated with a finite but possibly empty set of variables, x, for example, coordinates of points that describes a family of shapes. When the set of variables x is empty, then a shape is given automatically. Otherwise, a shape can be 'fixed' by using a function g to assign values to the variables x as $g(s) \approx g[s(x)]$. Shape grammars with parameters have been, historically, termed *parametric shape grammars*. To distinguish, grammars without parameters are referred to as *non-parametric shape grammars*.

2 Evolution of shape grammar definitions

A rigorous unified general definition of shape grammars is essential for theoretical analyses. In this regard, a detailed review of past definitions is essential to enable us to explore important characteristics over the time, capture tendencies of development, and obtain insights on a definition of shape grammars, which will be appropriate for complexity analysis.

The first formal definition, *SG-DEF-1971* [(2)] was given in the seminal article by Stiny and Gips (1971). Since then, several other definitions have appeared in the literature, each reflecting either the understanding at a particular time, or reflecting a specific research flavor. These include *SG-DEF-1974* (Gips, 1974), *SG-DEF-1975* (Stiny, 1975), *SG-DEF-1977* (Stiny, 1977), *SG-DEF-1980* (Stiny, 1980a), *SG-DEF-1991* (Stiny, 1991), *SG-DEF-1992* (Stiny, 1992) and the implied definition *SG-DEF-2006* (Stiny, 1992). All but *SG-DEF-1974,* and *SG-DEF-1992* are reviewed in detail.

In these definitions, a shape grammar $SG = <S, M, P, I>$ is typically specified as a 4-tuple made up from a set of *vocabulary* shapes S, a set of *markers* M, a set of *productions* P and an *initial shape* I together with a notion of shape rule application. The set S^* is formed by finite arrangements of one or more elements of S in which any element may be used in a multiple number of times with any scale or orientation. Typically, the elements of S^* and M have nothing in common. That is, $S^* \cap M = \varnothing$. Each element in P consists of an ordered pair (u, v) typically written as $u \rightarrow v$, represents a *shape rule*. The initial shape I normally contains an A such that there is an applicable rule, (u, v), which is an element of P. Elements of S^* appearing in certain rules (u, v) of P, or in I are *terminal* elements. By contrast, elements of M are *non-terminals*. Variations between the definitions rest on the way in which the sets and shape rule application are specified.

A shape is generated from a shape grammar by starting with the initial shape and recursively applying the rules. The result of applying a rule to a given shape is another shape consisting of the given shape with the right side of the rule substituted in the shape for an

[(2)] In this paper, the different definitions are named using the format, *SG-DEF-year*.

occurrence of the left side of the rule. In principle, shape rule application proceeds as follows: (1) find a part of the shape that is geometrically similar to the left side of a rule in terms of both spatial and marker elements, (2) find a geometric transformation that makes the left side of the rule identical to the corresponding part in the shape, and (3) apply those transformations to the right side of the rule and substitute the right side of the rule for the corresponding part of the shape. The shape generation process terminates when no shape rule in the grammar can be applied. The set of shapes generated by the shape grammar is defined to be its *language*.

2.1 SG-DEF-1971

The earliest definition for a shape grammar comes from the seminal article by George Stiny and James Gips (1971), who employ an analogy to phrase structure grammars (aka generative grammars).

Here, the marker set M is a finite set of shapes that is distinct from the vocabulary set S. Each rule (u, v) satisfies the following: u is a shape in $(x \in S^*) \times M$, and (i) v is x or, (ii) v is x augmented by a shape in M (that is, $x \in (\{s\} \times M)$) or, (iii) v is x augmented by a shape in $S^* \times M$. That is, $v \in ((\exists y, x \subseteq y \in S^*) \times M)$. I is a shape in $S^* \times M$.

In many ways this definition reflects the infancy of shape grammars in the sense that grammars are purely shape-based, daringly moving away from symbols in a direct manner. Markers, which are used to guide the application of shape rules, are also shapes distinguishable from the principal shape, avoiding the use of any non-shape symbols. There are no restrictions on the types of shapes used; that is, in principle, shapes can be

combinations of straight lines or curves, two- or three-dimensional, whatsoever. Analogous to phrase structure grammars, the * operator is defined over *S*, which is interpreted as a finite arrangement of elements of *S* under transformations of similarity, including the empty shape ø[(3)]. This enables one to define a shape vocabulary succinctly. However, on careful examination, there is no * operator defined over *M*—in all probability, this was a typo—however, this results in a definition that is not completely consistent with the shape grammar examples (Urform I, II, and III) given in the paper. In particular, the marker for RULE 1 (pp. 1461) changes in size as well as orientation while the set *M* is defined to contain a single marker. Noticeably, here shape rules can, in effect, only add more terminal shapes, with no capacity for subtraction, although markers can be eliminated, revised, or exchanged during the application of shape rules. Implicitly, the application of a shape rule involves the shape operations of Boolean sum and difference; moreover, recursive application of shape rules requires that both Boolean operations be closed over the types of shapes involved.

2.2 SG-DEF-1975

In his dissertation, Stiny (1975) investigated the concept of shape grammars in terms of two models, pictorial and formal. The definition of shape grammars in the pictorial model is essentially the same as Gips' definition, *SG-DEF-1974*. However, the definition for the formal model (*SG-DEF-1975*) is 'custom-designed' for analysis, analogously to phrase structure grammars. Here, a shape is restricted to consist of only line segments. Such shapes are most common, and have certain nice properties. For example, all shapes belongs to the

[(3)] Notationally, we distinguish between Ø, the empty set and ø, the empty shape.

set defined by a unit line segment under the * operator. In contrast, this is not the case for shapes made out of arcs. As before, a shape grammar is a 4-tuple: $<S, M, P, I>$, with the following conditions: Each production in P is of the form $<s_u, u_1, \ldots, u_n> \rightarrow <s_v, v_1, \ldots, v_n>$ such that (a) $s_u, s_v \in S^{*R}$; (b) for all i, $1 \le i \le n$, $u_i \in M^{*R}$, or $u_i = e$, for all i, $1 \le i \le n$, $v_i \in M^{*R}$; and (c) there is an i, $1 \le i \le n$, such that $u_i \neq \emptyset$ and $u_i \neq e$. The initial shape I is an $n+1$ tuples of shapes $I = <s_0, m_{0_1}, \ldots, m_{0_n}>$ such that (a) $s_0 \in S^{*R}$; (b) for all i, $1 \le i \le n$, $m_{0_i} \in M^{*R}$; and (c) there is an i, $1 \le i \le n$, such that $m_{0_i} \neq e$.

In comparison to *SG-DEF-1971* and *SG-DEF-1974*, the ^{R} operator, which enforces shapes to be in a reduced form (maximal lines), is new in this definition. Implicit in the definition of the ^{R} operator is the notion of embedded shapes, more specifically of proper line (segments) embedded in the maximal lines. A subshape of a shape is then made up from these embedded (line) segments. The restriction of shapes made out of straight lines makes it nearly impossible to distinguish S^* from M^*. The technique to deal with such difficulty is to employ shapes with multiple tuples; that is, shapes on different tuples are on different 'channels,' thereby not interfering with one another. The use of $n+1$ tuples of shapes, together with the symbol e (which behaves as the empty shape ø), enables shape grammars to be combined to form a new shape grammar. However, to our knowledge, no further work along these lines can be found in the subsequent literature.

This definition also distinguishes a special case of shape rule application. When the union of the left hand side of a shape rule has fewer than two points of intersection, there are potentially infinitely many ways to apply such a shape rule. It appears that Stiny, at the time,

regarded the 'infinitely many ways' unfavorably, defining the transformation to be the one which transforms the left hand side in such a way that each element has an identical, rather than a subshape, counterpart in the configuration.

With this definition, Stiny was able to show that shape grammars so defined are as equally powerful as Turing machines. Algorithms for shape rule application, and Church's thesis demonstrate that a Turing machine can simulate any shape grammar so defined. Likewise, a shape grammar can be constructed to simulate any Turing machine.

2.3 SG-DEF-1977

In his paper on Chinese ice-ray lattice grammars, (Stiny, 1977) introduces labeled shapes and parametric shape grammars briefly explaining the corresponding definition for non-parametric shape grammars in the appendix. As parametric shape grammars are further elaborated in a subsequent paper (Stiny, 1980a), we postpone discussion to section *SG-DEF-1980*; the focus here is on labels and markers.

In this definition, shapes are finite arrangements of straight lines of limited but nonzero length specified in some Cartesian coordinate system, assumed given. A family of shapes is defined by associating parametric expressions satisfying certain conditions with a limited number of points coincident with the lines in a given shape. A particular member of this family is specified, by giving an *assignment* of real values to parameters that satisfies the conditions. The result of applying an assignment g to a *parameterized shape* s is the shape denoted by $g(s)$. A non-parameterized shape is a special case of a *parameterized shape*—here g is always the identity assignment, that is, $s = g(s)$. In this case, s has no variables.

The set M consists of labeled points of the form $p{:}m$, where p is a point with the symbol m associated with it. It is not necessary for labeled points to be coincident with lines in a shape. A transformation t of a labeled point $p{:}m$ is the labeled point $t(p){:}m$, where $t(p)$ is the image of p under t. A *labeled shape*, $\sigma = \langle s, l \rangle$, consists of a shape s and an unordered set of labeled points, l. Note that an unlabeled shape s is the labeled shape with an empty set of labeled points; that is, $s \approx \langle s, \varnothing \rangle$. When s or l are parameterized, $\sigma = \langle s, l \rangle$ is a *labeled parameterized shape*. An assignment g to the parameters in s and l specify a specific labeled shape $g(\sigma)$ in the family of labeled shapes defined by σ.

Here, a parametric *shape grammar* has five parts: $\langle S, M, P, I, T \rangle$. T is a collection of allowable transformations. Here, M is a finite unordered set of labeled points. P is a set of shape rules $u \rightarrow v$ where u and v are labeled parameterized shapes in $S^+ \times M^*$ and $S^* \times M^*$ respectively. $M^* = M^+ \cup \{e\}$, where e denotes the empty labeled point. The initial shape is an element of $S^+ \times M^*$. Again, shapes are generated by a shape grammar by beginning with the initial shape I, and recursively applying the shape rules in the set P.

A shape rule $u \rightarrow v$ applies to a labeled shape s when there is an assignment g and a transformation t such that $t[g(u)] \leq s$. The result of applying shape rule $u \rightarrow v$ to labeled shape s under g and t is the labeled shape given by $[s - t[g(u)]] + t[g(v)]$. The expression for rule application for a non-parametric shape grammar is obtained by substituting for g by the identity function. That is, applying $u \rightarrow v$ to a shape s under t is the shape given by $[s - t(u) + t(v)]$.

This definition of shape grammars uses labeled points instead of markers, in contrast to definitions *SG-DEF-1971*, *SG-DEF-1974*, and *SG-DEF-1975*. Nevertheless, labels and markers are equivalent to some extent. Stiny explains that labeled points function in the same way as markers to guide shape generation; however, labels are invariant under Euclidean transformations whereas markers are not. However, such a distinction might be construed as overly simple, and depends on the design of shape rules. For example, in *SG-DEF-1974*, Gips employs certain geometrical characteristics, such as the asymmetry of the markers, to control shape rule application. By replacing markers with labels, the only important geometry information is position. As most grammars do not rely on the geometric characteristics of markers or labels beyond their position, we may use markers and labels interchangeably unless stated otherwise; consequently, the phrases 'marker-driven' and 'label-driven' mean exactly the same thing.

Knight (1983) provides an extensive discussion on the usage of labels. Labels in a shape rule normally supply additional information not provided by the shapes themselves and indicate (1) how, (2) where, or (3) when a shape rule may apply to the design being generated. *How* labels specify under which Euclidean transformations a rule can apply (usually by altering the symmetry). *Where* labels specify the subshapes in the design to which a shape rule can apply. *When* labels are associated with the design instead of with any particular point or points. This last kind of labeling is most frequently used to indicate successive stages in the generation of a design. Here, labels serve as status markers, regulating the sequence and repetition of rule applications. How and where labels are spatial

as their location is important. When labels are non-spatial as their presence rather than location is more important.

2.4 SG-DEF-1980

Stiny (1980a) elaborates on labeled shapes, non-parametric, and parametric shape grammars. This version (*SG-DEF-1980*) has subsequently become the standard definition, and is most widely quoted. In this definition, a shape is specified by its *maximal* line representation, and every line (segment) of a *subshape* of a shape is embedded in a maximal line of the shape. As before, a labeled shape consists of two parts: a shape and a set of labeled points. Parameterized labeled shapes are similarly defined. Labeled points may be parameterized. That is, the coordinates of a labeled point may be variables

Unlike *SG-DEF-1977*, in this definition, a shape grammar reverts back to comprising four components: S, a finite set of shapes; M, a finite set of labels; P, a finite set of rules of the form $u \rightarrow v$, where u is a labeled shape in $(S, M)^+$, and v is a labeled shape in $(S, M)^*$; and I, the initial shape, a labeled shape in $(S, M)^+$.

For non-parametric shape grammars, a shape rule $u \rightarrow v$ applies to a labeled shape s when there is a transformation t such that $t(u)$ is a subshape of s. The labeled shape produced by applying the shape rule $u \rightarrow v$ to the labeled shape s under the transformation t is given by $[s - t(u)] + t(v)$. Parametric shape grammars are extensions of non-parametric shape grammars in which shape rules are defined by filling the open terms (point variables) of a general schema.

A *shape rule schema* $u \rightarrow v$ comprises a pair of parameterized labeled shapes, u and v, where no member of the family of labeled shapes specified by u is the empty labeled shape. When specific values are given to the variables of u and v by an assignment g, to determine specific labeled shapes, a new shape rule $g(u) \rightarrow g(v)$ is defined. This shape rule can then be used to change a given labeled shape into a new shape in the usual way. That is, rule application is expressed as $[s - t(g(u))] + t(g(v))$.

In comparison to *SG-DEF-1975* and *SG-DEF-1977*, this definition is much more succinct and allows for more flexible shape rules. In *SG-DEF-1975*, markers are just shapes on different channels from the principal configuration, and labeling is implicit. In *SG-DEF-1977,* labels replace markers. In this definition, shape rules without symbols are supported; subshape matching drives shape rule application rather than markers or labels, whereupon, shape emergence becomes the factor to be considered during shape rule application. While this allows new types of shape rules, there is a price to pay. Computationally, determining applicability of shape rules as well as their corresponding transformations become much more complicated.

Note that, in this definition, the allowable transformations can be restricted to special kinds, although this facility seldom features in the subsequent literature. The restriction on the transformations in the case of the infinitely many ways of applying a shape rule, which appears in *SG-DEF-1975*, is not singled out here.

The introduction of parametric shape grammars basically extends the scope of allowable transformations. While providing for more flexible and natural design of shape rules,

function g for assigning parameters implicitly implies computational difficulty. Such functions are those allowing the points of a shape as variables (open terms) and the space of such functions is infinitely large. This means searching an infinite space. Indeed, Stiny (2006: pp 280) states that devising an algorithm to find the transformations under which a parametric shape rule applies to a configuration is an open question. As shown in this paper, the number of candidates to be tested increases exponentially fast as the number of open terms increases, making this problem NP (Garey and Johnson, 1979), perhaps NP-hard, in general.

2.5 SG-DEF-1991

An obvious deficiency of *SG-DEF-1980* is the limitation on shapes requiring them to be composed of straight lines. Shapes, in general, are formed as arrangements of points, lines, planes, solids, and even exotic curves and surfaces. Stiny (1991) generalizes definition *SG-DEF-1980* in terms of shape algebras.

Mathematically, if there is a t such that $t(u) \leq s$ is satisfied, then an object is produced according to the formula $[s - t(u)] + t(v)$. t, $\leq$, $+$, and $-$ are operators defined over a shape algebra, where t is a transformation function over a shape and can be generalized as a *being alike* function, $\leq$ is a partial order relation in terms of subshape, and $+$ and $-$ are Boolean sum and difference. All these operators are applied recursively until reaching the basic elements, on which these operators are directly defined.

In short, in a shape grammar, any pair of objects u and v defines a rule $u \rightarrow v$. The rule applies to an object s in a two-stage process involving a transformation t. The transformation is used in both stages, once with the subshape relation $\leq$ to distinguish some part of s, and then again with the arithmetic operations + and – to replace the part that has been picked out.

Under this definition, shapes are readily extensible. A shape can be *simple*—formed from basic elements of a single kind; or *compound*—a mix of various elements, optionally augmented in some way, for example, by colors. The only condition is that the operators of any shape algebra are defined on all its elementary objects, are recursively applicable, and are closed. In contrast to definition *SG-DEF-1980,* indeterminacy, that is, the infinitely many ways of applying a shape rule, is encouraged rather than restricted. While this causes little trouble for designers, indeterminacy is a tough issue for computer implementations. Additionally, shape emergence is regarded as a way of producing novel designs. As an extension to this definition, *SG-DEF-1992* (Stiny, 2006, 1992) formally includes labels and weights in algebraic terms.

2.6 SG-DEF-2006

In his monograph, *Shape: Talking about seeing and doing*, Stiny discusses shape grammars in terms of drawing shapes and calculating by seeing. The historical analogy of shape grammars to phrase structure grammars is re-examined, with the conclusion that the analogy is inappropriate; it implies a lot more than it should. As a matter of fact, during the design process, a designer's vocabulary of shapes is typically not prescribed; instead, new types of shapes are defined on the fly. Noticeably, in this book, a definition for a shape grammar is

actually never given, and only alluded to informally, with the basic formalism remaining the same as *SG-DEF-1992*.

2.7 Trends in the development of shape grammars

The evolutionary development of shape grammars falls into two categories: marker-driven and subshape-driven grammars. Computationally, this distinction is important; in comparison to marker-driven shape grammars, there are harder computational issues with subshape-drive shape grammars, in particular, parametric subshape recognition and indeterminacy.

The first four definitions: *SG-DEF-1971*, *SG-DEF-1974*, *SG-DEF-1975* and *SG-DEF-1977* belong to the former category in the sense that shape rule application in grammars so defined are controlled by markers. It is the markers, which play a pivotal role in determining both the applicability of shape rules as well as their corresponding transformation. Markers can be designed in a way that the determination of applicability and transformation is relatively straightforward to compute. In later developments of shape grammars, markers evolve as alphanumeric symbols, which make determination even simpler (albeit while losing power). All subsequent definitions belong to the subshape-driven category during which marker-driven (aka label-driven) and subshape-driven rule application can coexist. In other words, the definitions support both marker-driven and subshape-driven shape grammars. This coexistence between marker-driven and subshape-driven rules is explicit in *SG-DEF-1975*.

Chronologically, the above definitions exhibit backwards compatibility. That is *SG-DEF-1971 << SG-DEF-1975 << SG-DEF-1977 << SG-DEF-1980 << SG-DEF-1991 << SG-DEF-1992 << SG-DEF-2006*, where the right side of << is more general than the left side. Historically, it is significant to note that *SG-DEF-1971 << SG-DEF-1974*. However, there is a discrepancy between *SG-DEF-1974* and *SG-DEF-1975,* which were developed independently by the two principal authors of shape grammars, for very distinct research purposes, albeit from the same root, *SG-DEF-1971*. The discrepancy is reflected in the evolutionary development of shape grammar definitions.

The evolutionary development shows a trend from 'rigid' to 'soft'. 'Rigid' here means that the shape grammars are defined in a way that is closer to phrase structure grammars. Such shape grammars are more machine-bound in the sense that they are relatively easy to carry out (compute) on a computer, but harder to use to generate novel designs. As a matter of fact, there is very limited novelty involved. *SG-DEF-1974* falls within this category. A recent series of notable shape grammar implementations based on the CGA shape falls within the rigid category of shape grammars (Müller et al., 2007; Pascal Müller et al., 2006; Watson et al., 2008; Weber et al., 2009).

On the other hand, 'soft' is more human-centered, showing more concern and consideration on how to use shape grammars to generate novel designs. This explains, in part, the importance of subshape-driven grammars, concepts of indeterminacy and shape emergence, and the support for ambiguity in shape grammar research. Humans have little trouble handling such concepts. Moreover, human designers actually benefit from them. However, these concepts are problematical when considering computer implementation.

Recent developments attempt to subvert some of the more difficult issues through alternative representations (Keles, Ozkar and Tari, 2010). The features are summarized in Table 1.

Table 1. Evolution of shape grammar definitions

	Definition	**Features**
Marker-driven shape grammars	SG-DEF-1971	Based on shapes as both vocabulary and marker elements. Introduces the analogy to generative string grammars. Emergent shapes are implicit in 'surprises.'
	SG-DEF-1974	Based on closed polygons, curves. No treatment of emergent shapes. Theoretical basis of the first ever shape grammar interpreter implemented (Gips, 1974). Certain elements of the shape grammar were treated symbolically in Gips' implementation.
	SG-DEF-1977	Introduces labels and labeled points. Outlines the elements of parametric shape grammars.
	(*Above*) Marker-driven shape grammars based on an analogy to generative grammars. Grammars tend to be more tractable and easier to implement.	
	SG-DEF-1975	Based on two-dimensional rectilinear shapes. Implicit introduction to maximal lines. Mainly used to prove equivalence between shape grammars and other formal language formalisms. Emergent shapes are referred to as 'surprises.' Theoretical basis of the first shape grammar interpreter that properly took into consideration emergent shapes (Krishnamurti, 1982).
Subshape-driven shape grammars	(*Below*) Subshape-driven shape grammars. Progressively shy away from the generative grammar analogy. Grammars tend to be human-centered, less tractable, and harder to implement. Emergence is central to such grammars.	
	SG-DEF-1980	Introduces parametric shape grammar definition for shapes based on a maximal line representation. First definition for rule application explicitly based on the subshape relationship.
	SG-DEF-1991	Extends the definition to apply to shapes defined on different algebras, e.g., points, lines, planes and volumes. Introduces a being-alike function.
	SG-DEF-1992	Extends the algebraic definition to include weighted shapes. This definition subsumes a host of other independently defined weighted shape grammars, e.g., color grammars (Knight, 1989).
	SG-DEF-2006	Implicit definition of shape grammars considered in Stiny (2006). Essentially dismantles any vestiges of a connection to generative grammars. Indeed, generative grammars can be considered as a special case of shape grammars.

2.8 A unified definition of shape grammars

Traditionally, the non-parametric shape grammars formalism is defined as follows: for a shape rule $u \rightarrow v$ and a configuration c, if $t(u) \leq c$, then the result of applying the shape rule on c is $[c - t(u)] + t(v)$, where t is a transformation of similarity, $\leq$ is a part relation, $-$ is the operation of Boolean difference, and $+$ is the operation of Boolean sum. Note that, the operations of Boolean sum and difference implicitly involve an operation of reduction R, which is used to maintain the maximal representation (Krishnamurti, 1992b).

For parametric shape grammars, the formalism is defined as follows: for a shape rule schema $u(x) \rightarrow v(x)$ and a configuration c, if $t[g(u(x))] \leq c$, then the result of applying the shape rule on c is $[c - t[g(u(x))]] + t[g(v(x))]$, where g is a function which makes an assignment to the open terms (aka variables) of the schema.

Since t can be generalized to a being-alike function (Stiny, 1991), the function g can be combined with, thus subsumed by, t to form a new being-alike function. In this way, the formalisms of non-parametric and parametric shape grammars are unified.

For a shape rule $u \rightarrow v$ and a configuration c, if $t(a) \leq c$, then the result of applying the shape rule on c is $[c - t(u)]+t(v)$, where t is a being-alike function, $\leq$ is a part relation, $-$ is the operation of Boolean difference, $+$ is the operation of Boolean sum.

Implicit in this definition, as the last step in applying a shape rule, is the reduction operator, R, which is needed to maintain a maximal representation.

Following this definition, the scope of basic shape elements can be extended arbitrarily; the bottom line is that all operators of t, $\leq$, $-$, $+$, and R are well defined. In particular, elements are implicitly typed in a way that operators of $\leq$, $-$, $+$, and R only operates on two elements of the same type (that is, the elements are co-equal); For example, for two line segments, these operators are only meaningful when two have the same slope. The unified definition above is backwards compatible with the other definitions reviewed in this section.

The unified definition holds for shape rules defined across shape algebras. In a recent article, Stiny (2011) considers the specification of shape rules in terms of general transformations, part and boundary relationships. In his classification, Stiny considers three basic kinds of rule constructs: $x \rightarrow t(x)$, $x \rightarrow prt(x)$ or $x \rightarrow b(x)$, where $t(x)$ represents a transformation, typically geometrical, in general, a parametric variation of x; in other words, t is a being-alike function. The part relation satisfies $prt(x) \leq x$ with the reduction operation R relying on there being a shape y such that $x = prt(y)$, or $y = prt^{-1}(x)$. The inverse part relationship prt^{-1} is computationally interesting as it essentially specifies a data structure that hosts any of its embedded shapes. Certain prt^{-1} relations have been referred to as *carriers* (Krishnamurti and Stouff, 2004). An example is shown in Figure 1. Elements of $prt^{-1}(x)$ share descriptors with subshapes of $carrier(x)$, which can be effectively employed in computation (Stouff and Krishnamurti, 2006).

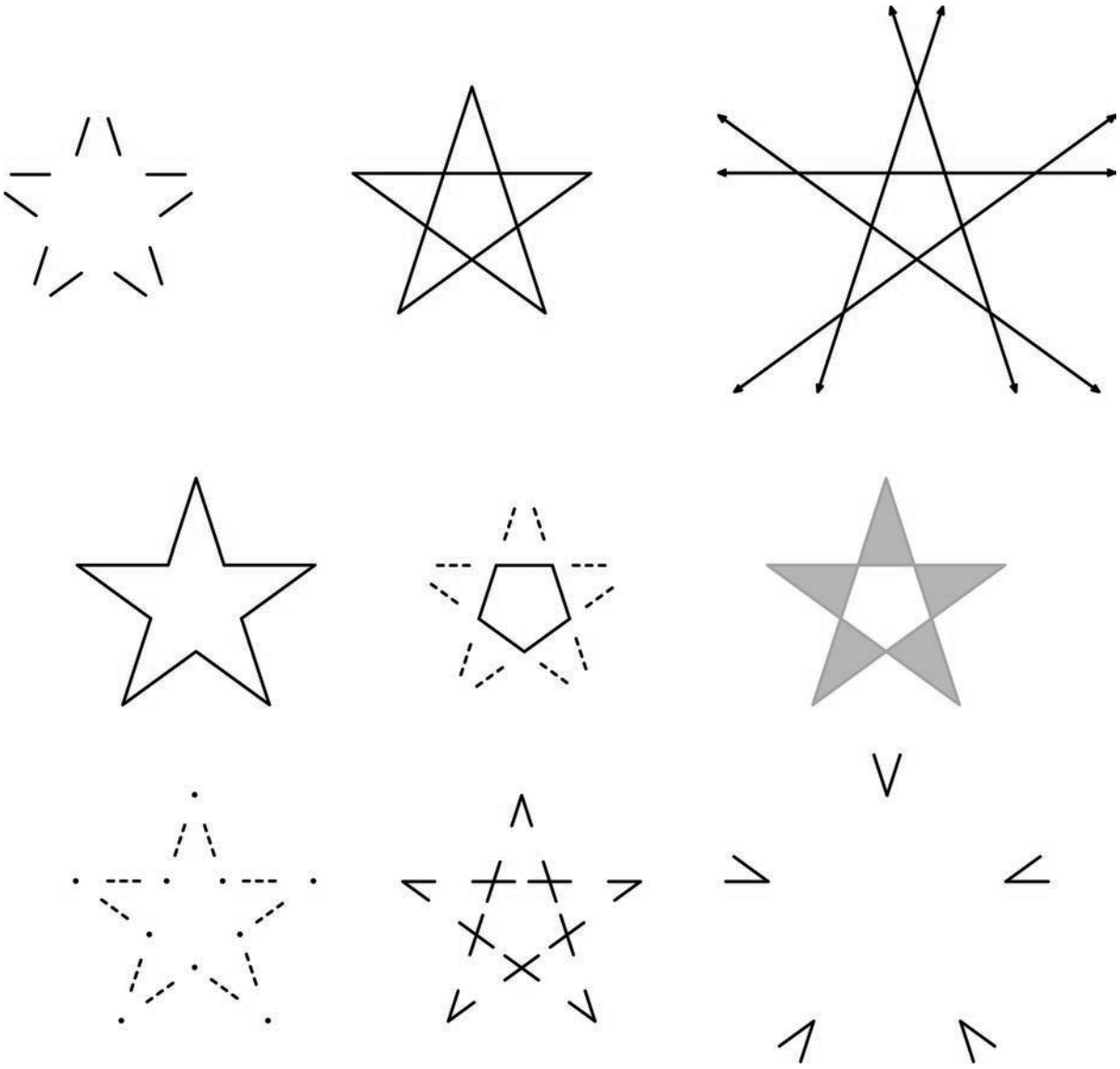


Figure 1. A shape x, a shape in $prt^{-1}(x)$, the *carrier* of x, and interesting subshapes of *carrier*(x)

Note that *carrier*(x) is a restriction on $prt^{-1}(x)$. There are subshapes y of *carrier*(x) that are not elements of $prt^{-1}(x)$. However, since $x + y$ is an element of $prt^{-1}(x)$, *carrier*(x) enables

us to explore shapes that appear 'unrelated' to *x* so to speak, that is, even having no shape in common with *x*. However, these shapes are parts of the same carrier, and this opens up an interesting question, which is beyond the scope of this paper to address. As Stiny (2004) remarks "seeing makes it worthwhile to calculate with shapes," it is worth exploring calculations with the 'unrelated' shapes of a shape. The answer may well lie within the carriers of shapes. For example, in Figure 1, for the left-most shape, from among possible shapes we see immediately, are the star, five triangles, the pentagon in the middle, ten distinct points of intersection, and at least 25 different angles. Some shapes are seen in combination with parts of the left-most shape; others have an identity all their own.

The boundary is a cross algebra operator. The *b* operator and its inverse b^{-1} define a shape, may resolve parameters and can be used in conjunction with *t*, *prt*, and prt^{-1} to specify particular shapes. Computationally, in a maximal element representation, the combination of a *b* and a prt^{-1} operator specify a shape (Krishnamurti and Stouff, 2004). Other kinds of shape rules such as erasure, identity, and in general, unrestricted parametric rules can be created from these basic constructs. See Stiny (2011).

3 Three corollaries

In his dissertation, Stiny (1975) concluded that for shape grammars under *SG-DEF-1975*, theorems on Turing machines naturally hold. Certain direct extensions of theorems on Turing machines to shape grammars are helpful in understanding their computational complexity. These are worth discussing in some detail. Using constructions similar to Stiny's, it is possible to show that a shape grammar under the unified definition can be constructed to

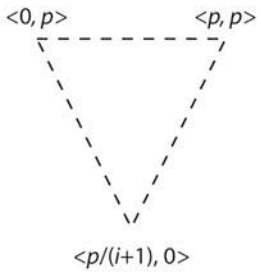
simulate any Turing machine. For this, it is essential to consider four aspects. The following briefly explain each and illustrate the corresponding simulation.

(1) *The states of a Turing machine can be encoded as shapes in reduced form, such that no two similar shapes represent distinct states. The set of shapes corresponding to the set of states of the Turing machine will form the main part of the set of markers for the constructed shape grammar.*

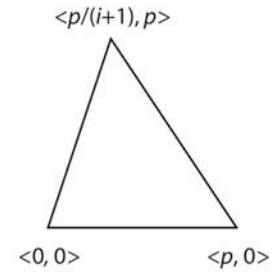
Consider a Turing machine with states q_i, $0 \le i \le n$. Each state q_i can be encoded by a triangle shape s_i with points $\{<0, p>, <p, p>, <\frac{p}{i+1}, 0>\}$, where $0 \le i \le n, p \ne 0$. Notice that for states q_i and q_j, if $q_i \ne q_j$, then s_i is not similar to s_j. For the shape rules simulating transitions, the states serve as markers. Figure 2a shows an example of one such state.

(2) *The tape symbols, including the blank symbol, of a Turing machine can be encoded as shapes in reduced form such that no two similar shapes represent different tape symbols. The set of shapes corresponding to the set of tape symbols of the Turing machine form the main part of the set of terminals for the constructed shape grammar.*

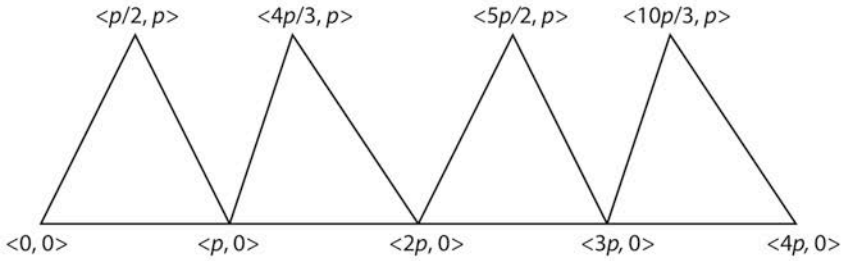
Tape symbols can be defined similarly to state symbols. Assume the Turing machine to have the set of tape symbols $\Sigma = \{a_i \mid 1 \le i \le m\}$. Let the blank symbol be a_0. Each symbol in the set, $\Sigma \cup \{a_0\}$, can be uniquely encoded by a triangle with points in the set $\{<0, p>, <p, p>, <\frac{p}{i+1}, 2p>\}$, where $0 \le i \le m$, $p \ne 0$. Figure 2b is an example of such a symbol.



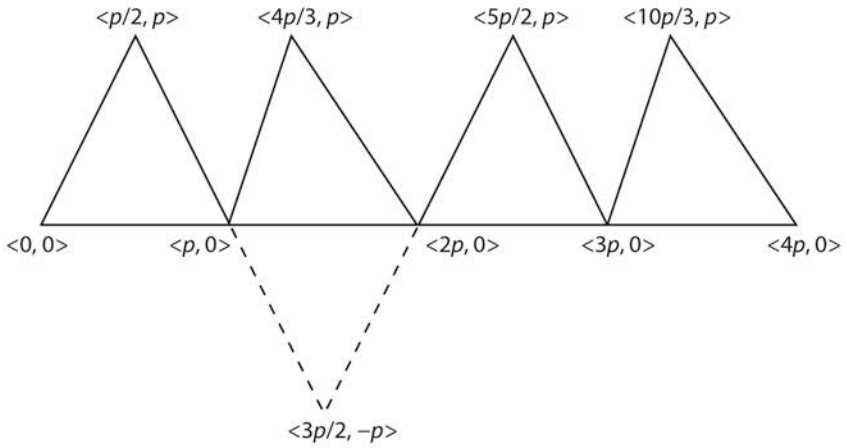
(a) Parametric shape encoding state q_i



(b) Parametric shape encoding symbol a_i



(c) A parametric shape encoding the tape $a_1\, a_2\, a_1\, a_2$



(d) A parametric shape encoding a configuration $a_1a_2a_1a_2$ in state q_1

Figure 2. Encoding symbols and states for a Turing machine by parametric shapes. Adapted from Stiny (1975)

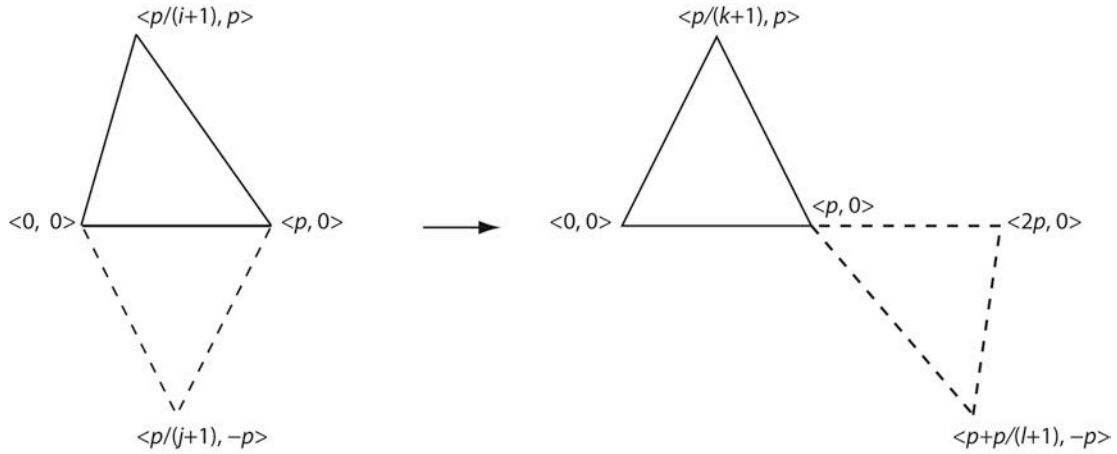
(3) *Turing machine tapes and configurations can be represented by shape grammars.*

Consider the Turing machine tape $a_{i_0} \ldots a_{i_k}$ where all symbols to the left of a_{i_0} and to the right of a_{i_k} are the blank tape symbol a_0. The tape can be represented by the shape $t_{i_0} \cup trans(t_{i_1}, p) \cup \ldots \cup trans(t_{i_k}, pk)$, where $trans(t, x)$ means translating shape t by x along the X-axis. Figure 2c illustrates an example of such a tape.

Assume that the Turing machine is in state q_i and is scanning the tape symbol a_{i_j} occurring in the tape $a_{i_0} \ldots a_{i_j} \ldots a_{i_k}$. The configuration can be represented by the pair of shapes $<T, trans(s_i, pj)>$ where T is the shape representing the tape $a_{i_0} \ldots a_{i_j} \ldots a_{i_k}$. Figure 2d illustrates an example of such a configuration.

(4) *Turing machine transitions can be represented as shape rules. The set of shape rules corresponding to the set of transitions of the Turing machine form the main part of the set of shape rules for the constructed shape grammar.*

A transition $<q_i, a_j, a_{j'}, q_{i'}, L>$, which reflects a Turing machine in state q_i scanning symbol a_j, replacing it by symbol $a_{j'}$, subsequently, going into state $q_{i'}$, and moving its tape one tape cell to the left, can be represented by the shape rule $<t_j, s_i> \rightarrow <t_{j'}, trans(s_{i'}, p)>$. Figure 3 depict two shape rules that simulate such transitions.



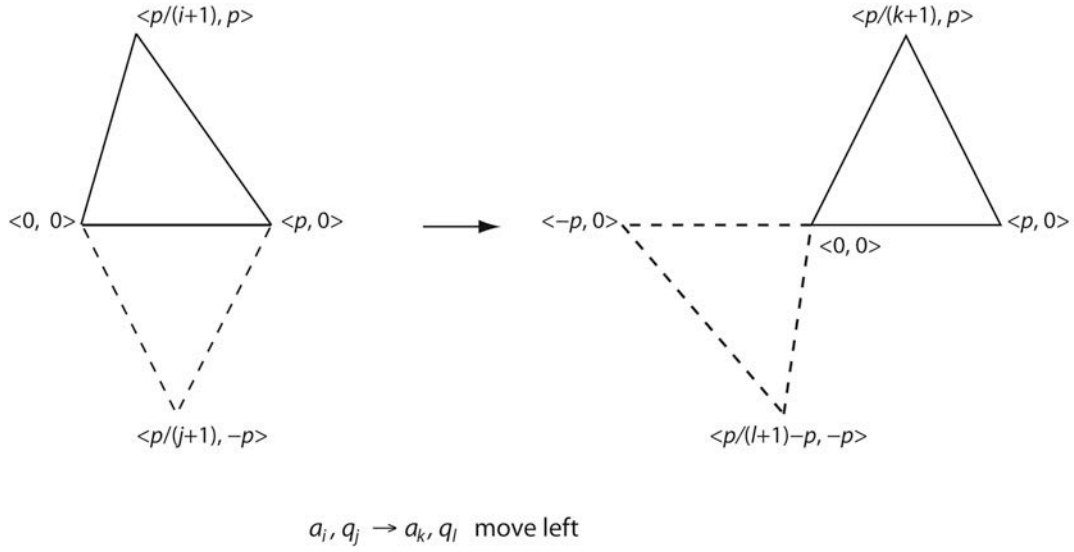


Figure 3. Simulating Turing machine transitions by shape rules. Adapted from Stiny (1975)

With the above setup, it is easy to see that the constructed parametric shape grammar simulates the computation of a Turing machine by derivation. The following are three relevant direct extensions from the theory of formal languages (Harry and Christos, 1997).

Firstly, it is well known that a Turing machine may not halt. Assume there is a computer program, which recursively applies the shape rules of a shape grammar until no shape rules can apply. As a result, this computer program will not halt for any shape grammar that simulates a non-halting Turing machine. In other words, there are non-halting shape grammars.

Secondly, a simulation of a non-deterministic Turing machine (abbreviated as NTM) with n steps by a deterministic Turing machine (abbreviated as DTM) requires exponentially many steps in n. Naturally, a shape grammar can be designed in a non-deterministic fashion, for example, the sports figure grammar (Carlson et al., 1991). Thus, a shape grammar can be designed to simulate any NTM in a fashion similar to simulating a DTM. This is equivalent to the problem of simulating a NTM by a DTM; that is the language space of a shape grammar can be exponentially large.

Lastly, another well-known theorem for unrestricted string grammars is that the membership problem—that is determining whether a string belongs to the language defined by a grammar or not—is undecidable. A shape grammar can be designed to simulate a string grammar in a similar manner to simulating a Turing machine. For such a shape grammar, the membership problem is equally undecidable—the proof, by contradiction, is trivial. In other words, in general, determining whether a configuration (aka. shape) belongs to the language defined by the shape grammar is unsolvable; that is, the problem of parsing a configuration against a shape grammar is unsolvable in general. Whether it is possible to restrict shape rules, to restriction categories similar to those defined for string grammars, for example, context-free grammars, remains an open problem.

4 Recognition in parametric shape grammars

The three corollaries show that, in principle, there are shape grammars, which do not halt and whose language spaces are exponentially large. Such shape grammars are unquestionably intractable. The following question is immediate: are all halting shape grammars with (even large) polynomial language space tractable? For ease of discussion, such grammars are termed *practical shape grammars*. See Figure 4.

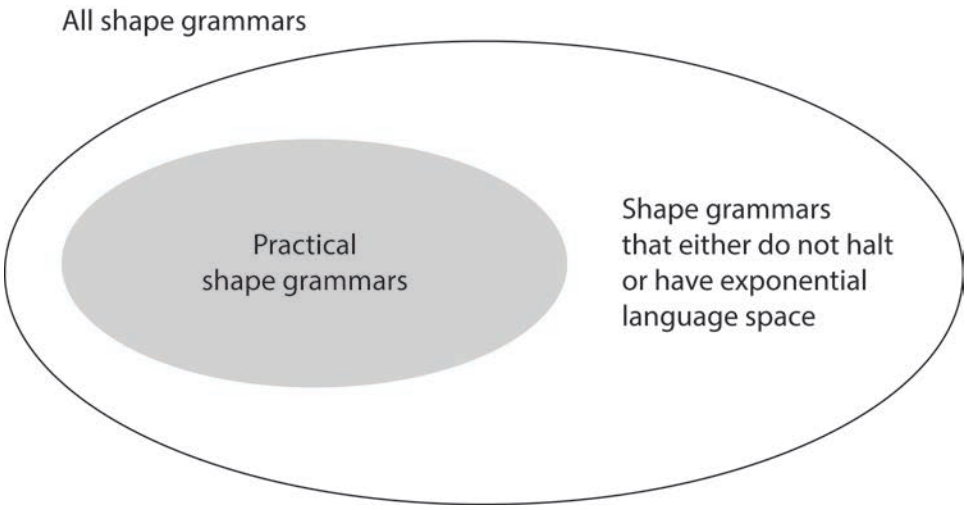


Figure 4. Practical shape grammars

A shape grammar is interpreted through the application of its shape rules. A shape grammar contains a finite set of shape rules. This fact underlies the fundamental basis for a shape grammar, namely, that of using a small number of shape rules to realize many, potentially, infinitely many, design possibilities (Stiny, 2006). For practical shape grammars, this fact implies that shape rules are applicable, at most, in polynomial time. It therefore

follows that tractability is determined by the application of each shape rule. However, it is known that parametric subshape recognition is difficult.

In the sequel, the tractability of a specific problem, that is, subshape recognition of parametric shape grammars over two-dimensional rectilinear shapes is examined. This is done by considering a polynomial-time reduction on the maximum clique problem (Cormen et al., 2004). The conclusion reached is that it is computationally expensive even for shapes of a relatively small size, and it is NP-hard for shapes with an arbitrary number of open terms.

4.1 Parametric two-dimensional rectilinear subshape recognition

For non-parametric subshape recognition of two-dimensional rectilinear shapes, the transformation t can be determined by matching three distinguishable points of a left side shape u to three distinguishable points of a configuration c (Krishnamurti, 1981). In fact, two points are all that are required with a third distinguishable point constructed from these two in four possible ways by considering reflections about the axis through the two distinguishable points. However, for parametric subshape recognition, this is not necessarily the case.

It is possible that the parametric shape u has a certain number of fixed points (non-open terms). If there are more than two fixed points (distinguishable by definition), the above 3-point algorithm is still applicable, with $O(n^2)$ possibilities to initially test against. For shapes with a single fixed point, this is identical to the situation when all points are open as similarity is subsumed by the assignment. When there are open points, the shape transformation may not be describable by a homogeneous transformation matrix. For

example, in Figure 5, (i) matches (ii) under a parametric shape rule, but there is no 3×3 homogeneous matrix which describes the transformation. In every case, open terms have to be determined, point-by-point, for each candidate subshape in c.

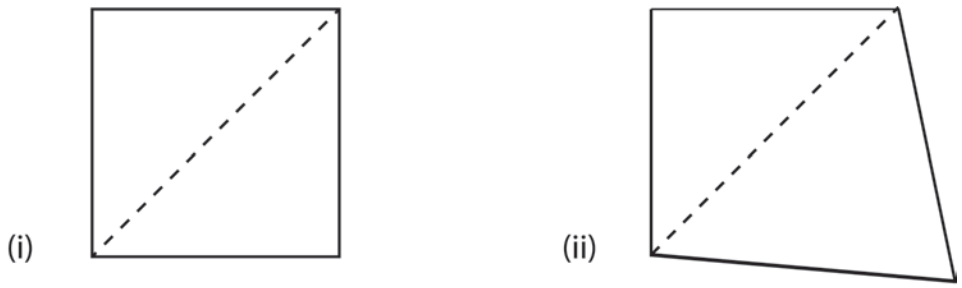


Figure 5. Example of parametric subshape matching

In general, when there are k open terms, there are $\binom{n}{k}$, or $O(n^k)$ possibilities. Even assuming that testing against each possibility costs unit time (typically, this is much more expensive in reality), when k is close to $n/2$, the time complexity is a super-polynomial. To illustrate with concrete examples, the possible number of tests is 7.5×10^7 when $k=5$, $n=100$; 1.7×10^{13} when $k=10$, $n=100$; and 1.0×10^{29} when $k=50$, $n=100$. It takes a computer, with performance of thousands of millions of instructions per second, several minutes to test all the possibilities when $k=10$, $n=100$. Note that when k exceeds $n/2$, the number of possible tests begins to decrease. It should be noted that in practice n and k are small numbers.

4.2 Parametric subshape recognition (PSR) is NP

Suppose that a subshape s is found under a parametric schema function t. We need to verify that s is the same as the left-hand side shape u under t. Let n be the maximum of the number

of points in s and u. As both shapes are two-dimensional and rectilinear, the verification algorithm first computes $u' = t(u)$ by applying t to each point in u. This takes $O(n)$ time, and u' so obtained has at most n points. The verification algorithm picks a point p in s, and compares the neighbors of p against the neighbors of point p' in u', which has the same coordinates as p. This process ends when all points of s have been compared. For each point p in s, it takes $O(n)$ time to find the corresponding point p' in u'. Since there are at most $(n-1)$ edges incident with p as well as p', it takes $O(n^2)$ time to verify equality of neighborhoods. Therefore, in total, it takes $n(O(n) + O(n^2)) = O(n^3)$ time to verify the equality of s and u'. That is, verification takes polynomial time.

We next show that PSR in general is NP by reducing the problem of finding certain cliques in a graph to the problem of PSR. That is, if we can solve parametric subshape recognition in polynomial time, then we can solve the graph theoretical clique problem in polynomial time, which is known to be NP, actually NP-hard (Cormen et al., 2004).

A *clique* in an undirected graph $G = (V,E)$ is a subset of vertices, $V' \subseteq V$, in which each vertex pair is connected by an edge in E. That is, a clique is a complete subgraph of G. The size of a clique is the number of its vertices. Figure 6c is an example of a clique of size 4. The clique problem corresponds to the optimization problem of finding a clique of maximum size in a graph. For example, in the graph of Figure 6a, the maximum clique is 4. The subgraph is shown bold.

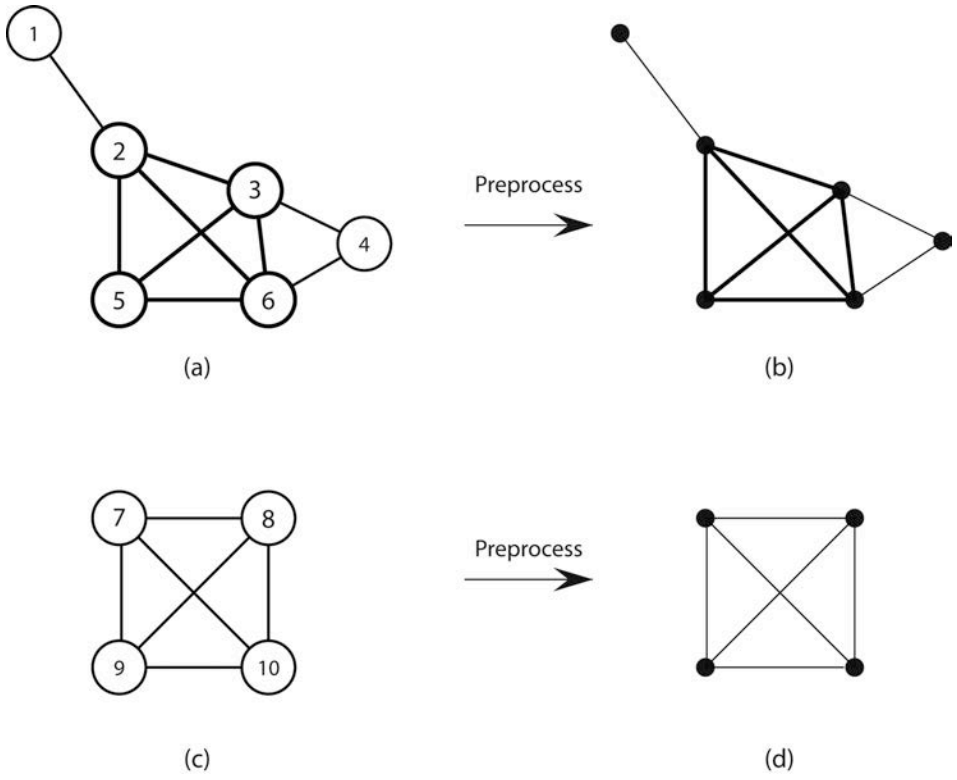


Figure 6. Example of finding a clique of size 4

Firstly, let us preprocess graph G (Figure 6a) to get G' (Figure 6b) by treating vertices in G as end points of incident edges, assigning unique x- and y-coordinates to all vertices so that no three vertices are collinear, and enforcing all arcs to be straight lines. This can be done in $O(|V|^2)$ time. Note that G' is actually a two-dimensional shape, and we can use it as the configuration shape c.

We generate a complete graph G_k with k vertices (Figure 6c) and similarly preprocess it to obtain G_k' (Figure 6d). This can be done in $O(k^2)$ time. Note that G_k' is another two-dimensional shape, and we may use it as the left side shape u. The points of u have their

counterparts in the vertices of G_k. Note that as u is a parametric shape, certain points, hence certain vertices in G_k, have coordinates with open terms.

If there is an algorithm capable of detecting the existence of subshape $g(u)$ in c by automatically finding an appropriate assignment of g in a polynomial time, then we can use the algorithm to detect the existence of subshape $g(G_k')$ in G' by automatically finding an appropriate assignment of g in a polynomial time, say, T_k. By the particular way that we processed graph G and G_k, the existence of subshape $g(G_k')$ in G' is identical to the existence of G_k in G. That is, we can use the algorithm to detect the existence of G_k in G in a polynomial time of T_k plus an added preprocessing time.

By the above preprocessing, and detecting sequentially for $k = \{1 \ \ldots \ |V|\}$ until the answer is false, we can find a clique of maximum size in time of $O(O(|V^2| + \sum_{k=1}^{|F|}(T_k + O(k^2)))$, which is a polynomial. This is a contradiction since the clique problem is known to be NP.

However, this result is not as bleak as it seems. The above proof is based on the assumption that the number of open terms k, and the number of points n, $n \geq k$, against which these open terms are matched are unbounded. In fact, the proof relies on the fact that n and k be as large as possible. Typically, n and k are bounded—we can employ a variant of the subgraph isomorphism algorithm, which, using brute force comparison, in the worst case, requires $O(n^k)$ time, although, as n grows indefinitely large, subgraph isomorphism too tends to be exponential in time. In reality, the open terms are topologically (and geometrically) dependent on the fixed terms. Matching can be resolved by following procedure. We first

match the subshape constructed from the fixed points. This yields a mapping between a spanning forest of the fixed points and corresponding points in the given shape, where two points are adjacent whenever there is a line in the shape between them. Next, the open terms are resolved by depth first search to grow this spanning forest. Implicit in this two-step matching process is a tree isomorphism test, which, if successful, is followed by a graph isomorphism check in the form of a simpler test for subshape relationship (Krishnamurti and Stouffs, 2004). We leave the details to the reader.

On the other hand, there are subshape situations, which do require exponential time. As a concrete example, consider a shape on $2n$ points such that it contains no n-sided polygon. It is possible to construct such a polygon quite easily. Let the points be numbered $1, 2, \ldots, 2n$. Construct lines pairwise among points $i, i+1, \ldots, i+(n-2) \bmod 2n$ and ensure that i and $i+(n-1)$ are not connected by a line. Then, for any sequence of n points it will not contain a line between, at least, one pair of points. That is, there are no n-gons in this polygon. However, determining that the shape does not contain an n-gon will require looking at $\binom{2n}{n}$ possible candidates, which requires $O(2^n)$ time.

The above example illustrates a situation where there are no matching subshapes. We have yet to find an example where a matching subshape exists, but one which requires exponential time to determine, although if the above example was reconstructed so that there was a single n-sided polygon, it is possible that it may still require $O(2^n)$ time to determine the subshape. [(4)]

[(4)] It is possible that we may not be able to construct just one single n-sided polygon within this configuration.

Taken altogether, we can conclude that parametric subshape recognition is NP in terms of the number of open terms. That is, in general, it is unlikely that there is a polynomial algorithm for parametric subshape recognition for two-dimensional rectilinear shapes. From this, we formally know that the problem of implementing a parametric shape grammar interpreter is NP, as parametric subshape recognition is a necessary step.

It follows then that some practical shape grammars are likely to be intractable. It is therefore important to know the factors that influence shape grammar tractability. In doing so, we would be in a position to manage and control the design of shape grammars to avoid these possible intractable situations.

5 Factors influencing tractability

The following analysis is built on top of the unified definition for shape grammars so that the results are as general as possible; that is, the factors influencing tractability are applicable to a variety of different kinds of shape grammars.

For practical shape grammars, tractability is determined by shape rule application. By definition, application of a shape rule involves operations of t, $-$, $+$, $\leq$, and R on elementary objects. If any of these operations takes superpolynomial time, then the shape rule application becomes intractable. In common design practice, the complexity of these operations may seem trivial, since the operations are not difficult to specify for rectilinear shapes. As shown by Stouffs and Krishnamurti (1993), the asymptotic upper bounds of comparing two co-equal spatial elements in d-dimensional space, $0 \leq d \leq 3$, is a polynomial in the maximum boundary element size n. In particular, when $d = 0,1$, the upper bound is a

constant; for $d = 2$, it is $\Theta((m+n) \log n)$, with $m = O(n^2)$; and for $d = 3$, $\Theta((Km + kn) \log n)$, with $K = O(k)$, $k = O(n)$ and $m = O(n^2)$.

However, for certain kinds of shape objects, some of these operations can be difficult, even intractable. An example is the Boolean operation on two solids with rational curved surfaces, which involves finding the intersection of two rational surfaces. The intersection of two smooth surfaces is one of the following: i) empty; ii) a collection of points; iii) a collection of smooth curves; iv) a collection of smooth surfaces; or, v) any combination of ii), iii), and iv) (Barnhill et al., 1987). Traditionally, analytical approaches by variable elimination have been the means to solving this kind of intersection problem. However, the degree of the resulting polynomial can be too high to solve. For instance, two generic bicubic patches can intersect in a curve of degree 324. Moreover, it has been shown that the intersection curves cannot be exactly represented by parametric equations even of degree 324 (Katz and Sederberg, 1988). Therefore, numerical methods have to be used and only curves under certain approximations are obtained. Although surface-to-surface intersection is still an active area of research (Hur et al., 2009; Patrikalakis et al., 2004), there are no good, general solvers for solving systems of multivariate polynomial equations, the equivalent problem to surface-to-surface intersection (Press et al., 2007).

The implication of this is that one cannot arbitrarily expand the scope of shapes. Basic operations of certain shape elements can become so complicated as to make them intractable. As a guideline, in order to design tractable shape grammars, the basic operations of the allowable elementary objects are required to take polynomial time. In the following discussion, we assume that this is the case.

The application of a shape rule $u \rightarrow v$ to configuration c involves two steps: searching the configuration c for applicable regions according to the left-hand side u, and rewriting the configuration with the right-hand side v. Rewriting a configuration involves two steps: subtracting (−) the left-hand shape under a known t, and adding (+) the right-hand shape under the same t. By our previous assumption, the operations of $-$, $+$, t, and R for each allowable elementary objects are in polynomial time. As there are a fixed number of elementary objects involved, the overall time complexity of rewriting still has an upper bound in polynomial time. It should be noted that the algorithm here is brute force, given simply for the purpose of deriving a polynomial upper bound—seeking efficient, uniform algorithms for rewriting is still valid research (Jowers, 2006; Stouffs, 1994). In our discussion, however, rewriting is 'easier', in the sense that there is always a brute-force polynomial algorithm.

On the other hand, searching a configuration for possible rule applications can be much 'harder'. In effect, the searching procedure includes two steps: using certain criteria to identify possible matching candidates, and then verifying the exact matching of each candidate under all allowable t. Even with the optimal searching criteria, the number of matching candidates can be super-polynomial. This is exactly the case for parametric subshape recognition over two-dimensional rectilinear shapes; the number of candidates increases exponentially as the number of open terms increases.

In the verification step for exact matching of a candidate, it is possible that there are infinitely many t's, which are impossible to compute in finite time. For example, for the candidate shape found in Figure 7 (marked with a dashed circle), the possible

transformations, up to scale, are indefinitely many. This phenomenon is known, in the literature, as *indeterminacy*, and viewed as an advantage where unexpected variations can be introduced (Stiny, 1991). However, it is hard for a computer implementation to appreciate this advantage. The basic question then is: which is the best way to choose one or a subset of possible candidates from infinitely many? Random choice provides a solution, but relying upon randomness to create novel designs is probably not always a good idea. Manual selection is another option, although this is counter to the goal of a computer implementation. What is certain is that it is impossible to elaborate all the indefinitely many possibilities; we have to assume that the grammar designer specifies a way of selecting a finite subset so that the implementation is tractable.

Figure 7. A candidate with infinitely many matching transformations under scaling

To sum up, there are three factors which influence tractability of a shape grammar: i) computational complexity of the basic operations t, $-$, $+$, $\leq$, and R; ii) number of matching candidates; iii) indeterminacy—number of possible t's for each matching candidate.

Factor i) is the most controllable in terms of computer implementation by ensuring that the system only supports basic elementary objects for which there are efficient algorithms (at most polynomial time complexity) for these operations.

Factor ii) probably influences certain extreme cases. In general, the number of open terms in parametric shape rules is usually small enough for their time complexity to be relatively inexpensive, thereby making shape recognition still tractable.

Factor iii) is somewhat controllable. As Stiny (1991) remarks, detailed conditions for indeterminacy are more complicated and vary from algebra to algebra and from dimension to dimension; Cartesian products are recommended as a useful way to avoid indeterminacy in general. However, in practice, chances for indeterminacy are much less. Shape grammars are seldom designed based purely on geometry—typically, they are imbued with semantics in the form of labeled points or elements. The semantics are usually sufficiently rich enough to permit only a limited number of possible transformations.

6 Computation-friendly shape grammars

The existence of both tractable and intractable shape grammars, together with other computation difficulties mentioned at the beginning of this paper negates the possibility of a single general shape grammar interpreter. As is shown in (Yue, 2009), characteristics of tractable shape grammars can vary significantly. In response, a paradigm for practical, 'general', shape grammar interpretation is proposed in a sequel (Yue and Krishnamurti, 2011), and shape grammars following such a paradigm can be said to be *computation-friendly*—that is, tractable with polynomial time and language space complexity.

Acknowledgement

This research was supported in part by a grant from US Army Corps of Engineers, Engineer Research and Development Center – Champaign, IL. Any opinions, findings, conclusions or recommendations presented in this paper are those of the authors and do not necessarily reflect the views of CERL. The authors are grateful to the referee for suggestions to improve the paper.

References

Barnhill R E, Farin G, Jordan M, Piper B R, 1987, "Surface/surface intersection" *Comput. Aided Geom. Des.* **4** 3-16

Carlson C, McKelvey R, Woodbury R, 1991, "An introduction to structure and structure grammars" *Environment and Planning B: Planning and Design* **18** 417-426

Chau H-H, 2002 *Preserving brand identity in engineering design using a grammatical approach* PhD dissertation, School of Mechanical Engineering and Keyworth Institute of Manufacturing and Information Systems, The University of Leeds

Chau H H, Chen X, McKay A, Pennington A, 2004, "Evaluation of a 3D shape grammar implementation", in *Design Computing and Cognition '04* Ed J S Gero, Boston pp 357-376

Cormen T H, Leiserson C E, Rivest R L, Stein C, 2004 *Introduction to Algorithms, Second Edition* (The MIT Press)

Garey MR, Johnson DS, 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (WH Freeman and Company)

Gips J, 1974 *Shape Grammars and Their Uses* PhD dissertation, Computer Science Department, Stanford University, Stanford, California

Gips J, 1999, "Computer implementation of shape grammars", (Cambridge, MA: NSF/MIT Workshop on Shape Computation)

Harry R L, Christos H P, 1997 *Elements of the theory of computation* (Prentice Hall PTR)

Hur S, Oh M-j, Kim T-w, 2009, "Approximation of surface-to-surface intersection curves within a prescribed error bound satisfying G2 continuity" *Computer-Aided Design* **41** 37-46

Jowers I, 2006 *Computation with Curved Shapes: Towards Freeform Shape Generation in Design* PhD dissertation, Department of Design and Innovation, The Open University, Milton Keynes

Jowers I, Prats M, Earl C, Garner S, 2004, "On curves and computation with shapes", in *Generative CAD systems symposium: G-CAD 2004*, Carnegie Mellon University, Pittsburgh, PA

Jowers I, Earl C, 2010, "The construction of cruved shapes" *Environment and Planning B: Planning and Design **B*** **37** 42 – 58

Jowers I, Earl C 2011, "Implementation of curved shape grammars" *Environment and Planning B: Planning and Design **B*** **38** 616– 635

Katz S, Sederberg T W, 1988, "Genus of the intersection curve of two rational surface patches" *Computer Aided Geometric Design* **5** 253-258

Keles HY, Ozkar M, Tari S, 2010, "Embedding shapes without predefined parts" *Environment and Planning B: Planning and Design **B*** **37** 664– 681

Knight T W, 1983, "Transformations of languages of designs" *Environment and Planning B: Planning and Design* **10** 125-177

Knight T W, 1989, "Color grammars: designing with lines and colors" *Environment and Planning B: Planning and Design* **16** 417-449

Krishnamurti R, 1981, "The construction of shapes" *Environment and Planning B: Planning and Design* **8** 5-40

Krishnamurti R, 1982, "SGI: An interpreter for shape grammars" Technical Report, Centre for Configurational Studies, The Open University, August 1982 (http://www.andrew.cmu.edu/user/ramesh/pub/distribution/technical/SGI.pdf)

Krishnamurti R, 1992a, "The arithmetic of maximal planes" *Environment and Planning B: Planning and Design* **19** 431-464

Krishnamurti R, 1992b, "The maximal representation of a shape" *Environment and Planning B: Planning and Design* **19** 267-288

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

Krishnamurti R, Stouffs R, 1997, "Spatial change: continuity, reversibility and emergent shapes" *Environment and Planning B: Planning and Design* **24** 359-384

Krishnamurti R, Stouffs R, 2004, "The boundary of a shape and its classification" *The Journal of Design Research* **4**

McCormack J P, Cagan J, 2003, "Increasing the scope of implemented shape grammars: a shape grammar interpreter for curved shapes", in *ASME 2003 International Design Engineering Technical Conferences and Information in Engineering Conference*, Chicago, Illinois, USA

Müller P, Zeng G, Wonka P, Gool L V, 2007, "Image-based procedural modeling of facades" *ACM Trans. Graph.* **26** 85

Pascal Müller, Wonka P, Haegler S, Ulmer A, Gool L V, 2006, "Procedural modeling of buildings" *ACM Trans. Graph.* **25** 614-623

Patrikalakis N M, Maekawa T, Ko K H, Mukundan H, 2004, "Surface to surface intersections" *Computer-Aided Design and Applications* **1** 449-458

Piazzalunga U, Fitzhorn P, 1998, "Note on a three-dimensional shape grammar interpreter" *Environment and Planning B: Planning and Design* **25** 11-30

Prats M, Jowers I, Earl C, Garner S, 2004, "Generative curves in product design", in *Design Computing and Cognition DCC´04*, MIT, Cambridge, MA

Press W H, Teukolsky S A, Vetterling W T, Flannery B P, 2007 *Numerical Recipes: The Art of Scientific Computing. Third Edition* (Cambridge University Press)

Stiny G, 1975 *Pictorial and formal aspects of shape and shape grammars and aesthetic systems* PhD dissertation, System Science, University of California, Los Angeles, CA

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

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

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

Stiny G, 1991, "The algebras of design" *Research in Engineering Design* **2** 171-181

Stiny G, 1992, "Weights" *Environment and Planning B: Planning and Design* **19** 413-430

Stiny G, 2004, "How to calculate with shapes" in *Formal Engineering Design Synthesis,* Eds E K Antonsson and J Cagan (Cambridge University Press) pp 20-64

Stiny G, 2006 *Shape: Talking about seeing and doing* (MIT Press, Cambridge)

Sting G, 2011, "What rule(s) should I use?", *Nexus Network Journal*, **13**(1) 15-47

Stiny G, Gips J, 1971, "Shape grammars and the generative specification of painting and sculpture", in *Information Processing 71, North Holland, Amsterdam* Ed C V Freiman pp 1460-1465

Stouffs R, 1994 *The algebra of shapes* PhD dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA

Stouffs R, Krishnamurti R, 1993, "The complexity of the maximal representations of shapes", in *The IFIP WG 5.2 Workshop on Formal Design Methods for CAD*, Talinn, Estonia

Stouffs R, Krishnamurti R, 2006, "Algorithms for the classification and construction of the boundary of a shapes" *Journal of Design Research* **5** 54-95

Watson B, Müller P, Veryovka O, Fuller A, Wonka P, Sexton C, 2008, "Procedural Urban Modeling in Practice" *IEEE Comput. Graph. Appl.* **28** 18-26

Weber B, Müller P, Wonka P, Gross M, 2009, "Interactive Geometric Simulation of 4D Cities" *Computer Graphics Forum*

Yue K, 2009, *Computation-Friendly Shape Grammars: With application to determining the interior layout of buildings from image data,* PhD dissertation, School of Architecture, Carnegie Mellon University, Pittsburgh, PA

Yue K, Krishnamurti R, 2013, "A paradigm for interpreting parametric shape grammars", *Environment & Planning B*: *Planning and Design*, forthcoming