*Artificial Intelligence for Engineering Design, Analysis and Manufacturing*

**cambridge.org/aie**

# Algorithmic complexity of shape grammar implementation

Thomas Wortmann[1] and Rudi Stouffs[2]

[1]Singapore University of Technology and Design, Architecture and Sustainable Design, 20 Dover Drive, Singapore 138682, Singapore and [2]Department of Architecture, National University of Singapore, 4 Architecture Drive, Singapore 117566, Singapore

**Research Paper**

**Cite this article:** Wortmann T, Stouffs R (2018). Algorithmic complexity of shape grammar implementation. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **32**, 138–146. https://doi.org/10.1017/S0890060417000440

Received: 25 October 2016
Revised: 12 May 2017
Accepted: 12 May 2017

**Key words:**
Shape grammars; shape grammar implementations; algorithmic complexity; tractability

**Author for correspondence:**
Thomas Wortmann, E-mail: thomas_wortmann@mymail.sutd.edu.sg

**Abstract**

Computer-based shape grammar implementations aim to support creative design exploration by automating rule-application. This paper reviews existing shape grammar implementations in terms of their algorithmic complexity, extends the definition of shape grammars with sets of transformations for rule application, categorizes (parametric and non-parametric) sets of transformations, and analyses these categories in terms of the resulting algorithmic complexity. Specifically, it describes how different sets of transformations admit different numbers of targets (i.e., potential inputs) for rule application. In the non-parametric case, this number is quadratic or cubic, while in the parametric case, it can be non-polynomial, depending on the size of the target shape. The analysis thus yields lower bounds for the algorithmic complexity of shape grammar implementations that hold independently of the employed algorithm or data structure. Based on these bounds, we propose novel matching algorithms for non-parametric and parametric shape grammar implementation and analyze their complexity. The results provide guidance for future, general-purpose shape grammar implementations for design exploration.

## Introduction

Shape grammars are formal descriptions of visual design processes that have been especially successful in terms of recreating a corpus of existing designs in a systematic, rule-based fashion (e.g., Stiny, 1977; Duarte, 2001). More recently, Stiny (2006) has reemphasized that shape grammars can also be used for creative design exploration, a process he calls “visual calculating”.

Computer-based shape grammar implementations aim to support such visual calculation processes by automating rule-application (e.g., Tapia, 1999; Trescak et al., 2012; Grasl & Economou, 2013; Strobbe et al., 2015). However, although such efforts have been ongoing for more than three decades – Krishnamurti (1981) proposed the first algorithm – shape grammar implementations are hardly used in actual design processes. One explanation for this limited use are the limited capabilities of existing implementations (see section ”Shape grammar implementations: the state of the art”) and the computational difficulty of expanding these capabilities (Yue & Krishnamurti, 2013).

The paper reviews some existing shape grammar implementations in terms of their algorithmic complexity, extends the definition of shape grammars with sets of transformations for rule-application, and categorizes sets of transformations. It describes how different sets of (parametric and non-parametric) transformations result in different numbers of potential targets for rule application. In the parametric case, this number depends on the size of the individual target. The analysis yields lower bounds that hold independent of the employed algorithm and data structure. Based on these bounds, we propose novel matching algorithms for non-parametric and parametric shape grammar implementation and analyze their complexity.

## Background

This section reviews shape grammars and shape grammar implementations, introduces the sub-shape detection problem, introduces algorithmic complexity, and discusses existing implementations in terms of this complexity.

### *Shape grammars*

There are various grammar formalisms for design, with different requirements for the representation, interpretation, and generation of entities. Most examples of shape grammars rely on labeled shapes, a combination of – often two-dimensional – line segments and labeled points (Stiny, 1980). Stiny (1992) proposes numeric weights as attributes to denote line thicknesses or surface tones. Knight (1989, 1993) considers a variety of qualitative aspects of design as shape attributes. Stiny (1981) proposes to augment shape grammars with description functions in order to enable the construction of verbal descriptions of designs. Stouffs and Krishnamurti (2002) propose sortal grammars using a compositional approach to the representational structures underlying (augmented) shape grammars, allowing the definition and exploration of a variety of grammar formalisms.

Stouffs (2018) addresses the problem of implementing a shape grammar interpreter that supports varying shape grammar formalisms. He proposes an algebraic treatment that facilitates a modular approach based on a connection between algebraic abstraction and procedural abstraction (Frank, 1999). However, he does not address parametric shape grammars and only touches upon transformations other than similarity. Nevertheless, his approach could include parametric shape grammars and shape grammars allowing for arbitrary transformations as well. In this paper, we address the algorithmic complexity of implementing such transformations.

## Shape grammar implementations: the state of the art

Ideally, a shape grammar implementation should: (1) support visual computing, (2) allow emergence, (3) not rely on predefined parts, and (4) be parametric (Gips, 1999).

Shapes have explicit definitions, but indefinitely many "touchable" parts, all of which can become a focus of design intent. These parts *emerge* under the part relation, regardless of whether they were originally envisioned as such. The concept of emergent shapes describes creative design activities, in the sense that looking at a design provides new insights that lead to a new interpretation of existing design elements. Continuity of visual computations requires an anticipation of the structures that are to be changed (Krishnamurti & Stouffs, 1997), but creativity is devoid of anticipation and instead relies on a restructuring of information. The concept of emergent shapes thus is highly relevant for design search (Mitchell, 1993; Stiny, 1994). Emergence is challenging to implement, however, which is why many implementations do not support it (Chau et al., 2004).

A corollary to supporting emergence is that an implementation cannot rely on pre-defined geometrical parts since it is impossible to predict which parts of a shape a designer might choose to develop further. Employing pre-defined parts thus involves assumptions about what a designer might want to do. For example, the geometric primitives provided in CAD programs make certain shapes easier to draw and manipulate than others. Shape grammars eliminate hierarchies and overlaps between parts and allow arbitrary de- and recomposition (Stiny, 1994).

Parametric shape grammars allow more flexible rule applications compared with non-parametric ones. As discussed in the section "Sets of transformations", the distinction between parametric and non-parametric grammars can be understood as a special case of the type of transformations under which a visual calculation accepts shapes as isomorphic.

Non-parametric shape grammar implementation is well researched, with a successful implementation presented by Tapia (1999). Current research focuses on parametric shape grammar implementations based on graphs (Grasl & Economou, 2013; Wortmann, 2013; Yue & Krishnamurti, 2013; Strobbe et al., 2015), since graphs avoid pre-defined parts and provide the needed flexibility to support both emergence and parametric transformations.

## The subshape detection problem

The central problem for shape grammar implementations that support emergence is the matching problem. This problem is termed *subshape detection*. A solution to this problem consists of finding a correspondence between the spatial elements in the left-hand side of a shape rule (the input shape) and a part of a given design (the target shape) and of determining the transformation (matrix) that represents this correspondence. If such a correspondence is found, the input shape and the target shape are *isomorphic* to each other.

The equivalent problems for set and graph grammars, termed subset and subgraph detection, consist of searching for either a single entity or a group of entities within a set or graph. This search is straightforward – it requires a one-to-one matching of entities that are identical under a certain transformation – but not always computationally efficient. The subgraph isomorphism problem, for example, is NP-complete (Garey & Johnson, 1979), which means that there is no known algorithm that can solve problems of this type efficiently, i.e., in polynomial time. But the ability to replace spatially any subshape of a shape is a prime requirement for visual calculations.

## Algorithmic complexity

A fundamental issue for shape grammar implementations is algorithmic complexity, which is a measure of the number of steps required by an algorithm. Algorithmic complexity measures this number of steps not directly, but *asymptotically*, that is, it describes the type of growth of this number as the size of the input for the algorithm increases. [Algorithmic complexity is indicated in Big-O notation (Cormen et al., 2009), with $O(n^k)$ indicating an upper and $\Omega(n^j)$ a lower bound. In other words, for inputs of size $n$, the algorithm that takes at most $n^k$ and at least $n^j$ steps.]

A problem is *tractable* when an algorithm with polynomial complexity is available (i.e., the number of steps is bounded by some $n^k$, where $n$ is the size of the input and $k$ is a constant) and intractable when there is none. Intractable problems – which often also are NP-complete or NP-hard – can be very slow, or even impossible, to compute. In practice, such problems are solved by approximation algorithms, but such approximations might not be acceptable in terms of the envisioned use of shape grammar implementations. For example, a designer might want to see all ways to apply a rule, and not only some, or even require the full enumeration of a grammar. Tractability thus is a key consideration for interactive shape grammar implementations which aim to provide quick feedback to designers.

Even for tractable problems, it is important whether their algorithmic complexity is constant [$O(1)$], linear [$O(n)$], linearithmic [$O(n \log n)$], or another polynomial such as [$O(n^k)$]. Note that polynomial algorithms can still be slow, especially for larger inputs $n$ and values of $k$ >2.

## Algorithmic complexity of shape grammar implementations

Despite the great relevance of algorithmic complexity, the literature on shape grammar implementations discusses it only rarely. For subshape detection for non-parametric shape grammars, Krishnamurti (1981) provides a polynomial algorithm that tests matches between the target shape and the design in terms of comparing triplets of intersection points. For $n$ maximal elements, this algorithm tests $O(n^4)$ triplets (actually, combinations of two intersection points with the third point constructed from these two points) drawn from $O(n^2)$ intersection points. Trescak et al. (2012) propose an improved version of this algorithm that sorts the triplets in terms of their angles, which avoids more time-intensive tests but does not improve algorithmic complexity.

The parametric shape grammar implementation of Grasl and Economou (2013) achieves subshape detection through non-planar subgraph matching, which is NP-complete. Yue and Krishnamurti (2013) argue that subshape detection for parametric shape grammars is NP-complete in general. Their proof rests on the claim that – by converting a graph into a shape – one could use a hypothetical, polynomial-time subshape detection algorithm for subgraph matching. But the existence of such an algorithm would contradict the fact that subgraph matching is NP-complete. However, subgraph matching is NP-complete only for non-planar graphs (Dorn, 2010). For planar graphs, it is linear in the size of the graph and exponential in the size of the subgraph, or $2^{O(k)}n$. One could thus use the hypothetical, polynomial-time subshape detection algorithm for planar subgraph detection without contradiction. In other words, we hypothesize that there is no bijective mapping between graphs and shapes: for shapes, intersection points are an inevitable result of intersecting maximal elements, while for graphs, intersecting edges does not result in a new vertex (Wortmann, 2013). This paper presents a subshape detection algorithm whose complexity is similar to the result for subgraph isomorphism by Dorn (2010).

Beyond subshape detection, Yue and Krishnamurti (2013) identify three factors influencing the tractability – and thus algorithmic complexity – of shape grammar implementations: operations on shapes, the potential number of matching instances where a rule can be applied, and indeterminacy of rule application. Another factor for the algorithmic complexity of shape grammar implementations is the data structure used for representing shapes. In the analysis by Wortmann (2013) of different graph data structures used for shapes, the smallest representations are quadratic, or $\Theta(n^2)$. This paper discusses the algorithmic complexity of the number of matching instances in terms of different sets of (parametric and non-parametric) transformations.

## Shape grammars as sets of rules and transformations

According to Stiny (1980), a shape grammar consists of a set of shapes $S$, a set of labels $L$, a set of shape rules $R$ and a starting design $I$ ($I$ can be the empty shape). However, as shown below, to apply the shape rules in $R$ one also needs a set of admissible transformations $T$.

Although Stiny (ibid.) briefly discusses Euclidian and isometric transformations, these transformations are not an explicit part of his definition. This omission is typical of many shape grammars. Such grammars define the shape rules and starting design explicitly, but define the set of transformations only implicitly, either by example or by constraining shape rule application with labels. However, as discussed in the section "Lower bounds on the algorithmic complexity of shape grammars", the set of admissible transformations is a determining factor for the algorithmic complexity of shape grammar implementations. Additionally, most examples of shape grammars are expressed as parametric shape grammars, although shape grammar implementations often avoid the issue of parametric shape rules.

Shapes consist of maximal elements. In the most common case, the maximal elements are non-overlapping line segments in the plane (overlapping line segments are not maximal and are combined into one maximal element), and a shape is a set of maximal elements. Often, a shape is a closed polygon, or a combination of closed polygons. A shape rule $r$ can be understood as a function with an input shape $x$ and an output shape $y$:

$$r(x) = y, \quad r \in R.$$

Designers apply a shape rule by identifying an instance of the input shape $x$ in a design $D$, and replacing this target shape $s$ with the output shape $y$. Often, the target shape $s$ is a subshape, in other words, it is a part of a larger design $D$. Formally, one identifies a target (sub-)shape by defining an admissible transformation that maps the input shape onto a part of the design, that is, a target shape. When there is an admissible transformation between an input and a target shape they are isomorphic to each other (see Fig. 1).

$$t(x) = s, \quad s \in D, \quad t \in T.$$

One applies the shape rule by subtracting the target shape from the design and adding the transformed output shape.

$$D' = D - s + t(y), \quad t \in T.$$

This formalization of shape grammars clarifies the importance of admissible transformations for shape isomorphism, shape rule application, and shape grammar implementation. [Note that beyond their application to target shapes, shape rules also involve a transformation (or mutation) that maps the input onto the output shape (Stiny, 2011).] The following sections categorize admissible sets of transformations used in shape grammars and analyze these categories in terms of their algorithmic complexity.

## Sets of transformations

Existing shape grammars apply different sets of transformations to map input to target shapes, for example, non-parametric, and parametric transformations (Stiny, 1980). The authors of

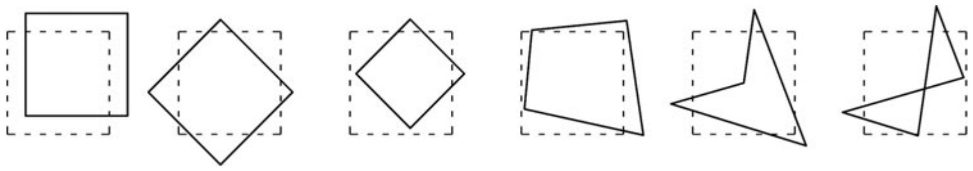

**Fig. 1.** Examples of quadrilaterals that are isomorphic under different sets of transformations: translated, isometric, similar, matrix (affine and projective), and two arbitrary transformations of shapes with four maximal elements. The fourth shape maintains convexity as an invariant, the fifth shape the topology of links and intersections, and the sixth shape only the number of maximal elements.

this paper are not aware of an established categorization of these transformations. To analyze the impact of different sets of transformations on the algorithmic complexity of shape grammar implementations, this paper proposes the following categories, listed from simple to complex: translated, isometric (Euclidean), similar, matrix (including affine and projective) and arbitrary (see Table 1 for an overview of these transformations). Note that every set of transformations also includes the ones from the preceding sets.

### Translated

In this category, the only allowed transformation is a translation. In other words, a shape can only be moved, but not rotated or mirrored. Consequently, there is only one way in which a rule that copies and moves a square diagonally can apply to the squares of the translated derivation in Figure 2, resulting in a purely diagonal arrangement. In practice, translation-only shape grammars are rare, most likely due to their limited formal possibilities.

### Isometric

The isometric, or Euclidean, transformations include translation, rotation, and reflection. As implied by the term "rigid", these transformations preserve a shape's size, proportions, and angles, though not necessarily its orientation. The isometric derivation in Figure 2 applies the rule as the translated derivation, but in different orientations. However, the rule cannot apply to the smaller and larger squares that emerge from the overlap of the original ones, since isometry does not include scaling.

Although one can achieve the formal possibilities of an isometric shape grammar also within a translated one (by adding individual rules for symmetries and rotations), isometric shape grammars allow geometric precision with a more economical set of rules. This precision makes them especially applicable to traditional architectural designs methods that emphasize proportions and measurements. A classic example of an isometric grammar is the Palladian Grammar (Stiny & Mitchell, 1978*b*). [Note that the Palladian grammar, in principle, allows for (non-uniform) scaling, as the size of the grid cell may vary.]

### Similar

Similarity transformations include uniform scaling in addition to the Euclidean transformations. Uniform scaling preserves a shape's proportions and angles, but not its size, as is visible in the similar derivation in Figure 2. In contrast to isometry, in this case, the same rule can apply to different-sized squares. Like fractal patterns, similarity-based shape grammars operate on different scales. Accordingly, they are suitable for abstract, scale-less pattern designs such as the Sierpinski gasket (Yue & Krishnamurti, 2014).

### Matrix

This category includes all transformations that can be expressed as a transformation matrix, such as affine and projective transformations (transformation matrices are convenient for computer implementation). Matrix transformations include translation, isometry, and similarity, as well as linear transformations. Linear transformations include both affine and projective transformations. Projective transformations, for example, perspective projections, do not preserve the angles and proportions of a shape, in contrast to more general characteristics, like maximal elements, intersections, and convexity. Affine transformations preserve parallel lines and ratios of distances as well. In the projective (matrix) derivation in Figure 2, starting with an initial shape including only horizontal and vertical lines, and applying a shape rule that adds only parallel lines, the derivation also includes only horizontal and vertical lines.

Matrix-based shape grammars are parametric in the sense that they guide rule-application not with numerical criteria such as lengths and angles, but with topological criteria such as the number of maximal elements, closure, convexity, etc. Stiny (1980) refers to such grammars as having open terms. For example, Tapia (1992) implements the ice ray grammar of Chinese lattice designs (Stiny, 1977) as a parametric grammar that allows triangles of any kind as inputs and preserves not the proportions, but only the number of sides of output shapes.

### Arbitrary

This last category includes all transformations, including ones that transformation matrices cannot express. Stiny (1990) discusses the application of visual rules in terms of arbitrary transformations *t*. For example, shape isomorphism can be defined topologically based on relations of connectivity between maximal elements. One can also preserve other characteristics such as the convexity or number of sides of a shape. Other possibilities include basing shape isomorphism on the area enclosed by a shape, or the fact that a shape is a polygon, etc. To analyze the

**Table 1.** The columns in this table represent the five sets of transformations

| Translated | Isometric | Similar | Matrix | Arbitrary |
|---|---|---|---|---|
| Translation | Translation | Translation | Translation | Translation |
| | Rotation | Rotation | Rotation | Rotation |
| | Reflection | Reflection | Reflection | Reflection |
| | | Uniform scaling | Uniform scaling | Uniform scaling |
| | | | Linear transformations | Linear transformations |
| | | | | Everything else |
| | | Non-parametric | Parametric | |

Each row represents an individual transformation. Note how each set contains progressively more transformations. Also, note the distinction between non-parametric and parametric (i.e., non-similar) sets of transformations.

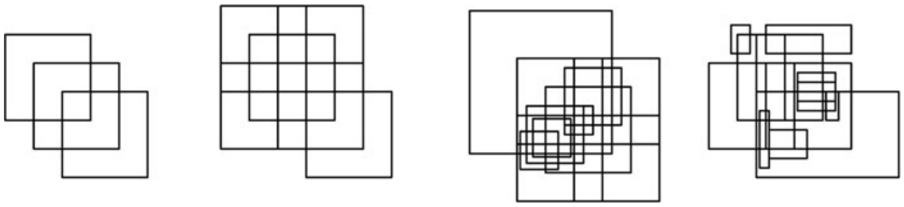
**Fig. 2.** Derivations based on the same initial shape (a square) and the same shape rule applied under different transformations: translated, isometric, similar, and matrix (from left to right).

algorithmic complexity of different sets of transformations, this paper examines only cases where the number of maximal elements is invariant.

With this flexibility of rule application, Stiny (2011) intends to preserve a maximum of design freedom. Per this view, shape grammars are a special case of calculating with visual rules, in that they restrict visual calculations to a set of rules and transformations. As a general design method, visual calculating does not require such restrictions. However, the next section shows how the increasing computational complexity of the sets of transformations results in algorithmic restrictions in terms of generalized shape grammar implementations.

## Lower bounds on the algorithmic complexity of shape grammars

The existing literature discusses the algorithmic complexity of shape grammar implementations primarily in terms of the algorithms for subshape detection. This paper proposes a different perspective: considering algorithmic complexity in terms of the number of potential target shapes.

If a general shape grammar implementation should allow rule-application for all target shapes, this potential number of shapes provides lower bounds for shape grammar implementations. In other words, the algorithmic complexity of a shape grammar implementation cannot be smaller than the largest number of potential target shapes it intends to present to the user for potential rule application.

The ability to find all potential target shapes – and thus all possibilities for rule application – in a reasonable amount of time is fundamental for shape grammar implementations in terms of securing the design freedom envisioned by Stiny (2011). This ability is for example necessary to let a user choose where to apply a rule based on a visual preview or to automate the enumeration of the complete design space of a specific grammar (Stiny & Mitchell, 1978*a*).

Table 2 presents the simple case of finding all quadrilaterals in a grid under different sets of transformations (see Fig. 3). The results represent upper bounds in the sense that they represent the largest number of subshapes we might want to find in such a case, even if some might consider it pathological. They represent a lower bound for shape grammar implementation in the sense that even with a linear time subshape detection algorithm, we would at least need that many steps to find all possibilities to apply a shape rule.

The following sections show that the number of potential target shapes depends on the set of admissible transformations for mapping input shapes onto target shapes, with more flexible transformations leading to higher algorithmic complexity. Although from a practical standpoint, the number of potential target shapes can be limited by asking the user to preselect an area for rule-application, this increasing algorithmic complexity remains a challenge for shape grammar implementations that intend to preserve as many design possibilities as possible.

### *Limitations*

The analysis has two limitations: (1) It assumes that, in the parametric case, all maximal elements intersect each other and (2) only considers closed quadrilaterals as target shapes. For the parametric case, the analysis generalizes to target shapes with arbitrary numbers of maximal elements. These limitations are justifiable in terms of proving lower bounds for algorithmic complexity because removing them would only increase this complexity and makes implementation more challenging. For example, graph-based shape grammar implementations often represent maximal line segments as portions of infinite lines. In this case, non-closed shapes become a special case of closed ones. In addition, note that the resulting bounds do not consider the complexity of representing shapes (Wortmann, 2013) and the complexity of subshape detection. (In other words, we consider how many subshapes one would need to find, not how to find them.)

**Table 2.** The numbers of intersections and isomorphic quadrilaterals resulting from a rectangular grid of maximal elements in terms of different sets of transformations as a function of the number of maximal elements

| Maximal elements | Intersections (similar) | Intersections (parametric) | Translated quads | Isometric quads | Similar quads | Parametric quads |
|---|---|---|---|---|---|---|
| $n$ | $(n/2)^2$ | $(n^2 - n)/2$ | $\Omega\ (n^2)$ | $\Omega\ (n^2)$ | $\Omega\ (n^3)$ | $\Omega\ (n^4)$ |
| 4 | 4 | 6 | 1 | 8 | 8 | 8 |
| 6 | 9 | 15 | 4 | 32 | 40 | 120 |
| 8 | 16 | 28 | 9 | 72 | 112 | 560 |
| 10 | 25 | 45 | 16 | 128 | 240 | 1.680 |
| 12 | 36 | 78 | 25 | 200 | 440 | 3.960 |

We motivate the asymptotic lower bounds below. Note that the numbers for non-similar intersections and parametric quadrilaterals assume that all maximal elements intersect each other.

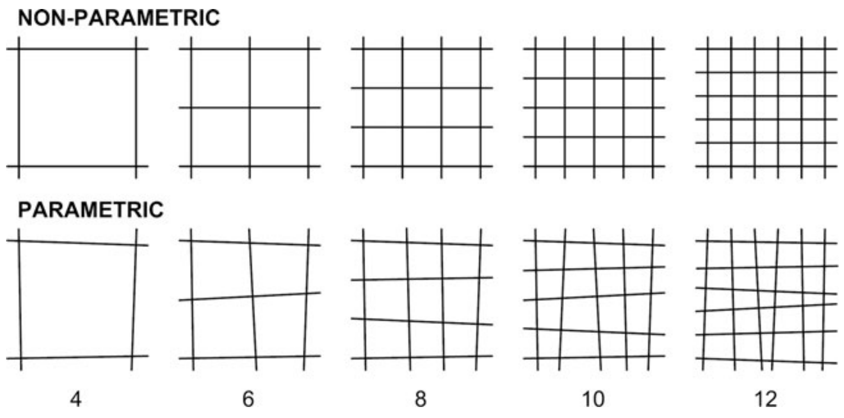

**Fig. 3.** Grids consisting of even numbers of maximal elements. The top row consists of parallel lines, resulting in different numbers of isomorphic quadrilaterals (i.e., squares) under translated, isometric, and similar transformations. The bottom row consists of skewed grids, which result in a factorial number of isomorphic quadrilaterals under affine and arbitrary (i.e., parametric) transformations. Table 2 tabulates the different numbers of intersections and isomorphic quadrilaterals under different sets of transformations.

Nevertheless, the resulting bounds provide guidance for the design of general shape grammar implementations.

## *Translated and isometric*

Let $n/2$ parallel, equidistant, horizontal maximal elements intersect with $n/2$ parallel, equidistant, vertical maximal elements. Then the total number of maximal elements is $n$, and the largest number of identical squares under translation is $(n/2-1)^2$ or $n^2/4-n+1$. Consequently, the algorithmic complexity to find all target shapes for a rule having such a square as an input is $\Omega(n^2)$. For example, the $4\times 4$ grid of eight lines in Figure 4a yields two (vertical and horizontal) sets of $8/2-1=3$ columns, in Figure 4b. Squaring, that is, intersecting the two sets of three columns yields nine identical, translated squares. When isometry also is admissible, every square can be rotated four times and mirrored, which yields eight target shapes per square. However, this linear increase does not affect the quadratic bound already established for translation.

## *Similar*

Once one admits also similarity transformations, target shapes of different sizes become targets for rule application. That is, we should consider not only the $(n/2-1)^2$ squares in the original grid, but also squares resulting from grids with a smaller number of maximal elements, such as $[(n-2)/(2-1)]^2$, $[(n-4)/(2-1)]^2$, …, $[4/(2-1)]^2$. In other words, one can combine different sets of smaller squares into bigger squares. For example, the $4\times 4$ grid of eight lines in Figure 4a yields two (vertical and horizontal) sets of $(8-2)/2-1=2$ overlapping columns, in Figure 4c. Squaring, that is, intersecting the two sets of two columns yields four identical, translated squares. Likewise, in Figure 4d, there are two sets of $(8-4)/2-1=1$ columns, which yield one square.

The resulting number of squares is a quadratic series. The sum of such a series (Sloane & Arndt, 2010) – also known as the sum of square pyramidal numbers – can be expressed like this:

$$1^2+2^2+3^2+\cdots+m^2=\sum_{1}^{m}k^2=\frac{m(m+1)(2m+1)}{6}$$
$$=\frac{m^3}{3}+\frac{m^2}{2}+\frac{m}{6}.$$

Considered in terms of the number of maximal elements $n$, $m=n/2-1$. Asymptotically, the potential number of target shapes under similarity transformations is cubic or $\Omega(n^3)$.

## *Matrix and arbitrary*

In the parametric case, any combination of four maximal elements forms a quadrilateral (if all maximal lines intersect each other), and thus is a potential target for rule application. Figure 4e shows how three intersecting lines always form a triangle, and how the fourth line cuts this triangle into two smaller triangles and one quadrilateral. In the worst case for affine transformations, all quadrilaterals could also be isomorphic in terms of affinity. An algorithm for subshape detection in the

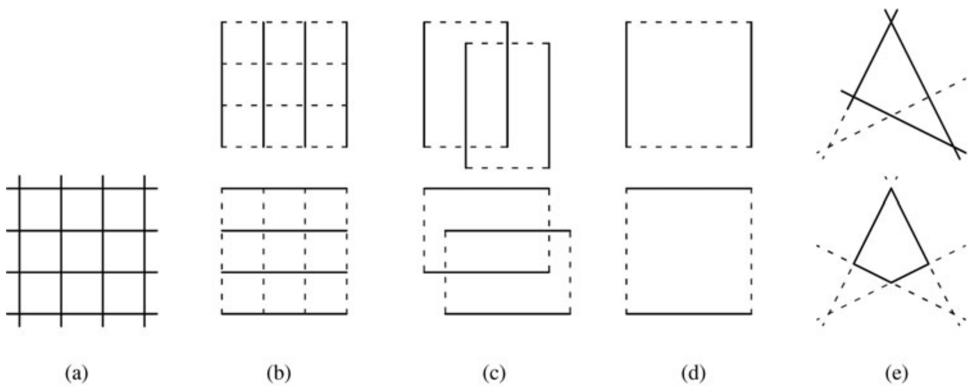

**Fig. 4.** Diagrams illustrating the derivation of the lower bounds. Panels (a) and (b) illustrate translation and isometry, (a)–(d) similarity, and (e) matrix and arbitrary.

affine case would thus have to test for every quadrilateral if there exists an affine transformation matrix that maps it onto an input shape.

Even under non-linear transformations, one might want to retain the distinction between convex and concave shapes. However, this distinction at best halves the number of potential target shapes, and thus has no impact on algorithmic complexity.

If one retains the number of maximal elements $k$ as an invariant between input and target shapes, one can express the number of potential target shapes for input shapes with $n$-maximal elements as a combination, multiplied by eight to account for rotations and symmetry:

$$8\binom{n}{k} = \frac{8n!}{k!(n-k)!}.$$

Asymptotically, for a quadrilateral input shape, the potential number of target shapes is $\Omega(n^4)$. In general, this expression is $\Omega(n^k)$. When $k$ grows close to $n/2$, the expression becomes non-polynomial. However, even in the polynomial case, one can expect the performance of a parametric shape grammar implementation to deteriorate rapidly as input shapes become more complicated and the overall design grows larger. Note that this result is similar to the bound for planar subgraph isomorphism by Dorn (2010).

Consequently, the most critical factor for the algorithmic complexity of parametric shape grammar implementations is the size of the input shapes. In the general case, parametric subshape recognition is non-polynomial, even with a hypothetical, linear time subshape detection algorithm. One can limit combinatorial explosion by introducing additional criteria for mapping shapes, such as convexity, proportions, closure, etc. Nevertheless, this result highlights an acute challenge.

Yue and Krishnamurti (2013) present a similar result in terms of the open terms of parametric shape grammars and argue that this number is "usually small enough for their time complexity to be relatively inexpensive". Indeed, for individual grammars, it is often possible to design more efficient parametric implementations (Stouffs, 2017). However, one cannot assume such efficiency for designers that use a parametric shape grammar implementation for design exploration. In this case, quite complex input shapes might arise easily.

Grasl and Economou (2013) report benchmark results of their parametric shape grammar implementation, searching for parametric quadrilaterals in a grid similar to the ones presented in Figure 3. They employ the subgraph detection algorithm by Batz (2006), which – in a square grid consisting of 22 maximal elements – finds all 3.025 quadrilaterals in about 500 milliseconds. However, for targets shapes with more than four maximal elements, the number of targets would increase rapidly, which would arguably affect the subgraph detection algorithm's performance.

Strobbe et al. (2015) employ a similar graph-matching algorithm for subshape detection. They report a running time that is exponential in terms of the size of the design but "reasonably low (<2 s)" for a floorplan with about 30 rooms. However, they also do not consider the size of the input shape, which again is only a quadrilateral.

One should also note that both benchmarks only use two-dimensional line segments. The impact of more complex geometries on the performance of parametric grammar implementations (Stouffs & Krishnamurti, 2006) would probably be considerable and deserves further study.

## Upper bounds on the algorithmic complexity of shape grammars

The analysis in the previous section suggests novel, straightforward algorithms for non-parametric and parametric subshape detection with an improved worst-case complexity. These upper bounds complement the lower bounds presented above.

### *Non-parametric (translated, isometric, and similar)*

In two dimensions, only two distinct points are needed to determine any similarity transformation, except for a reflection about the axis connecting both points. For shapes made up only of $k$ line segments, as considered here, three infinite lines, not all parallel, result in at least two intersection points, yielding two (or three) distinct points. Specifically, when no two lines are parallel, there are three distinct intersection points. Otherwise, a third distinct point can always be constructed at a perpendicular angle from the axis connecting both points and at a distance equal between the two existing points (Krishnamurti, 1981). In both cases, non-parametric subshape detection requires the matching of three maximal lines with at least two intersections. Once a similarity transformation is found, one can verify the matching in $O(k)$ steps. As such, the algorithm presented by Trescak et al. (2012) considering triplets of intersection points, while ordering them by angle and ratio of lengths, is unnecessarily complex. Considering $n$ maximal lines, the number of intersection points is $O(n^2)$, and there are $O(n^6)$ triplets to be considered. But the number of triplets of maximal elements is only $O(n^3)$ and angles and ratios of lengths can still be used to quickly reduce potential matches. Note that this is also the potential number of target shapes under similarity transformations, providing a tight bound.

### *Parametric (matrix and arbitrary)*

With $\Omega(n^k)$ as the lower bound for parametric target shapes established in the section "Matrix and arbitrary", one can sketch a subshape detection algorithm and analyze its complexity: Given an input shape $s$ with $k$ maximal elements, one can compare it with a potential target shape $t$ by a pairwise comparison of the elements and their labels. Such labels can denote invariants such as intersections with other elements, convexity, line segments, etc. (Wortmann, 2013). First, we compute the $O(k^2)$ intersections for $s$ and $t$. For every element of $t$, we then perform a pairwise comparison with the elements of $s$. This comparison verifies the connection between elements and matches between labels. Using tree search to traverse $t$, each comparison would take $O(k^2)$. These tree searches are easy to parallelize. Note that, in this case, we do not search for emergent subshapes, but check only whether two shapes are isomorphic, which is an easier problem. To perform comparisons for all elements then takes $O(k^3)$. We must perform this test for all candidate target shapes, which yields a tight bound of $O(n^k)$. Note that, depending on the size of $k$, this bound is exponential in the worst case, although one can use labels to limit the combinatorial explosion. Note that the above, non-parametric subshape detection algorithm is a special case of the parametric one since both rest on the general principle of comparing combinations of maximal elements for isomorphism.

A graph-based variant of this algorithm would convert $s$ and $t$ into planar graphs. In this case, a planar graph representation is possible, because the representation does not need to account for emergent subshapes (Wortmann, 2013). Isomorphism for planar, connected, undirected, and unlabeled graphs is possible in $O(k^3)$ as well (Kukluk et al., 2004), resulting in the same overall complexity (since labels would decrease, rather than increase, the complexity of the comparison).

## Conclusion

This paper identifies sets of transformations as a crucial factor for the algorithmic complexity of shape grammar implementations. As we have seen, this algorithmic complexity increases from cubic to non-polynomial as more transformations are allowed. For parametric shape grammar implementations, the paper identifies the size of the input shape as a critical factor that limits the capability of parametric shape grammar implementations in terms of supporting design exploration with real-time feedback. Nevertheless, we propose a subshape detection algorithm for parametric transformations that is polynomial for small input shapes.

As the lower bounds presented in this paper hold independent of specific subshape detection algorithms or (graph-based) data structures, parametric shape grammar implementations must rely on faster processing speeds and parallelization to achieve usable, interactive applications. One might also want to exploit the fact that planar subgraph detection is polynomial, for example by developing a planar graph representation for shapes. Even then, many types of visual calculations probably remain out of reach in terms of computer implementation, an assessment shared by Yue and Krishnamurti (2013). Further research would test implementations of the proposed algorithms and extend the presented bounds and algorithms – which apply to line segments in two dimensions – to higher dimensions and other types of geometries.

**Acknowledgments.** We thank the three anonymous reviewers for providing critical interactions that helped to improve and clarify this manuscript. The contribution of the second author received some funding support from Singapore Ministry of Education's AcRF start-up grant, WBS R-295-000-129-133.

## References

**Batz GV** (2006) *An Optimization Technique for Subgraph Matching Strategies* (Internal Report No. 7). Institut für Programmstrukturen und Datenorganisation, Fakultät für Informatik, Universität Karlsruhe (TH), DE.

**Chau HH, Chen X, McKay A and de Pennington A** (2004) Evaluation of a 3D shape grammar implementation. In Gero JS (ed.). *Design Computing and Cognition'04*. Dordrecht, NL: Springer, pp. 357–376.

**Cormen TH, Leiserson CE, Rivest R and Stein C** (2009) *Introduction to Algorithms*, 3rd edn. Cambridge, MA: MIT Press.

**Dorn F** (2010) Planar subgraph isomorphism revisited. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science.*

**Duarte JP** (2001) *Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses*, Ph.D. Dissertation. Boston, MA: Massachusetts Institute of Technology.

**Frank AU** (1999) One step up the abstraction ladder: Combining algebras-from functional pieces to a whole. In *Proceedings of the International Conference on Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science*. London, UK: Springer.

**Garey MR and Johnson DS** (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

**Gips J** (1999) Computer implementation of shape grammars. *NSF/MIT Workshop on Shape Computation.*

**Grasl T and Economou A** (2013) From topologies to shapes: parametric shape grammars implemented by graphs. *Environment and Planning B: Planning and Design* **40**(5), 905–922.

**Knight TW** (1989) Color grammars: designing with lines and colors. *Environment and Planning B: Planning and Design* **16**(4), 417–449.

**Knight TW** (1993) Color grammars: the representation of form and color in designs. *Leonardo* **26**(2), 117–124.

**Krishnamurti R** (1981) The construction of shapes. *Environment and Planning B: Planning and Design* **8**(1), 5–40.

**Krishnamurti R and Stouffs R** (1997) Spatial change: continuity, reversibility, and emergent shapes. *Environment and Planning B: Planning and Design* **24**(3), 359–384.

**Kukluk JP, Holder LB and Cook DJ** (2004) Algorithm and experiments in testing planar graphs for isomorphism. *Journal of Graph Algorithms and Applications* **8**(2), 313–356.

**Mitchell WJ** (1993) A computational view of design creativity. In Gero JS and Maher ML (eds). *Modeling Creativity and Knowledge-Based Creative Design*. Hillsdale, NJ: Lawrence Erlbaum, pp. 25–42.

**Sloane NJA and Arndt J** (eds) (2010) The On-Line Encyclopedia of Integer Sequences. Available online at https://oeis.org

**Stiny G** (1977) Ice-ray: a note on the generation of Chinese lattice designs. *Environment and Planning B: Planning and Design* **4**(1), 89–98.

**Stiny G** (1980) Introduction to shape and shape grammars. *Environment and Planning B* **7**(3), 343–351.

**Stiny G** (1981) A note on the description of designs. *Environment and Planning B: Planning and Design* **8**(3), 257–267.

**Stiny G** (1990) What is a design? *Environment and Planning B: Planning and Design* **17**(1), 97–103.

**Stiny G** (1992) Weights. *Environment & Planning B: Planning and Design* **19**, 413–430.

**Stiny G** (1994) Shape rules: closure, continuity, and emergence. *Environment and Planning B: Planning and Design* **21**(7), S49–S78.

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

**Stiny G** (2011) What rule(s) should I use? *Nexus Network Journal* **13**(1), 15–47.

**Stiny G and Mitchell WJ** (1978*a*) Counting palladian plans. *Environment and Planning B: Planning and Design* **5**(2), 189–198.

**Stiny G and Mitchell WJ** (1978*b*) The palladian grammar. *Environment and Planning B: Planning and Design* **5**(1), 5–18.

**Stouffs R** (2017) A practical shape grammar for Chinese ice-ray lattice designs. In *Cultural DNA Workshop: Computational studies on the cultural variation and heredity*. Daejeon, KR.

**Stouffs R** (2018) Implementation issues of parallel shape grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. doi: 10.1017/S0890060417000270.

**Stouffs R and Krishnamurti R** (2002) Representational flexibility for design. In Gero JS (ed.). *Artificial Intelligence in Design '02*. Springer Netherlands, pp. 105–128.

**Stouffs R and Krishnamurti R** (2006) Algorithms for classifying and constructing the boundary of a shape. *Journal of Design Research* **5**(1), 54–95.

**Strobbe T, Pauwels P, Verstraeten R, De Meyer R and Van Campenhout J** (2015) Toward a visual approach in the exploration of shape grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **29**(04), 503–521.

**Tapia M** (1992) Chinese lattice designs and parametric shape grammars. *The Visual Computer* **9**(1), 47–56.

**Tapia M** (1999) A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design* **26**(1), 59–73.

**Trescak T, Esteva M and Rodriguez I** (2012) A shape grammar interpreter for rectilinear forms. *Computer-Aided Design* **44**(7), 657–670.

**Wortmann T** (2013) *Representing Shapes as Graphs*, SMArchS Thesis. Cambridge, MA: Massachusetts Institute of Technology.

**Yue K and Krishnamurti R** (2013) Tractable shape grammars. *Environment and Planning B: Planning and Design* **40**(4), 576–594.

**Yue K and Krishnamurti R** (2014) A paradigm for interpreting tractable shape grammars. *Environment and Planning B: Planning and Design* **41** (1), 110–137.

**Thomas Wortmann** is a PhD candidate in the Architecture and Design Pillar at Singapore University of Technology and Design. He holds a MArch from the University of Kassel and a SMarchS in Design and Computation from MIT. His SMarchS thesis was on graph representations for shapes. He is an award-winning architectural designer and has been building computational design tools for over a decade. His research interests are computational design and interactive, visual optimization tools based on surrogate models.

**Rudi Stouffs** is an Associate Professor in the Department of Architecture at the National University of Singapore. He holds an MS in Architectural Engineering from the Vrije Universiteit Brussel, an MS in Computational Design, and a PhD in Architecture from Carnegie Mellon University. He has held previous appointments at the School of Architecture at Carnegie Mellon University, the Chair for Architecture and CAAD at ETH Zurich, and the Chair of Design Informatics at Delft University of Technology. His research interests include computational issues of description, modeling, and representation for design, mainly in the areas of shape recognition and generation, and building/city information modeling and analysis.