Five Criteria for Shape Grammar Interpreters

Tzu-Chieh Kurt Hong and Athanassios Economou

Abstract Shape grammar interpreters have been studied for more than forty years addressing several areas of design research including architectural, engineering, and product design. At the core of all these implementations, the operation of embedding—the ability of a shape grammar interpreter to search for subshapes in a geometry model even if they are not explicitly encoded in the database of the system—resists a general solution. Here, a detailed account on various constructions of embedding is provided, including determinate and indeterminate ones, to give a sense of the rising complexity of their implementation in a shape grammar interpreter, and to provide a visual map of the work accomplished in the field so far, and the work ahead too.

1 Introduction

Shape grammar interpreters have been studied for more than forty years addressing several areas of design research. Several useful accounts of existing general shape grammar interpreters and purpose-built shape grammar interpreters are readily available in the literature [1–4]. An updated view of these lists is given below in Table 1 featuring 61 applications including implementations of general and specific purpose shape grammar interpreters. Note, however, that this list—and most of its predecessors—includes very few actual shape grammar interpreters (in the general and technical sense of the word), and that many of the references in the list are just

Be it as it may, the sheer increase of the number of applications should suggest an optimistic state of affairs on the current state-of-the-art of the field, and yet, the evidence of the impact of shape grammar interpreters in practice and academia does not readily support such a view.

T.-C.K. Hong (✉) · A. Economou
Georgia Institute of Technology, Atlanta, USA
e-mail: khong@gatech.edu

A. Economou
e-mail: thanos@gatech.edu

J. S. Gero (ed.), *Design Computing and Cognition'20*,
https://doi.org/10.1007/978-3-030-90625-2_11

Table 1 List of shape grammar implementations

No	Name	Author	Year
1	Shepard-Metzler Analysis	Gips	1974
2	Simple Interpreter	Gips	1975
*3	Shape Grammar Interpreter	Krishnamurti	1982
*4	Shape Grammar Interpreter	Krishnamurti & Giraud	1986
5	Queen Anne Houses Grammar	Flemming	1987
6	SEED	Flemming & Woodbury	1987
*7	Shape Grammar System	Chase	1989
*8	Genesis (CMU)	Heisserman	1991
9	GRAIL	Krishnamurti	1992
10	Grammatica	Carlson & Stouffs	1993
*11	Genesis (Boeing)	Heisserman	1994
12	Shape Grammar Editor	Shelden	1996
13	Implementation of Basic Grammars	Duarte & Simondetti	1997
14	3D Shape Grammar	Piazzalunga & Fitzhorn	1998
15	SG Clip	Chien	1998
16	Coffee Maker Grammar	Cagan & Agawal	1998
*17	GEdit	Tapia	1999
18	3D Shaper	Wang & Knight	1999
19	Shaper2D	McGill	2001
20	EifForm	Shea	2002
21	A Simulated Shape Grammar Yingzao Fashi	Li	2002
22	3D Architecture Form Synthesizer 3D Shaper	Wang & Duarte	2002
*23	Parametric Shape Grammar Interpreter	Cagan & McCormmack	2002
*24	Shape Grammar Implementation U_{13}	Chau	2004
25	Coca-Cola Grammar	Chau	2004
26	Computational Environment for Learning	Wong & Cho	2004
27	MALAG	Duarte	2005
28	Shape Designer V.2	Wong	2005
29	Bracket System	Wu	2005
*30	Curve-based SGI	Cagan & McCormmack	2006
31	Implementation of Description Grammar	Correia & Duarte	2006
32	Marrakech Medina Grammar	Duarte	2007
33	Sub-shape Detector	Jowers	2008
34	CityEngine	Mueller	2008
35	Grammar Environment	Li	2009
36	Classique Ottoman Mosques	Sener & Gorgul	2009
37	SGMP (for CNC Router)	Ertelt & Shea	2009
*38	Parametric SG Interpreter	Yue & Krishnamurti	2009
39	Shape Grammar and Augmented Reality	Chen	2009
*40	SD2	Jowers	2010

(continued)

Table 1 (continued)

No	Name	Author	Year
41	GraphSynth	Campbell	2010
42	Embedded Shape Detector	Keles, Ozkar & Tari	2010
43	Shape Design V.2 - SGI	Trescak	2010
44	Baltimore Row-house	Aksamija	2010
*45	QI	Jowers	2011
46	Thonet Chair Grammar	Barros	2011
47	SG Parsing via Reinforcement Learning	Teboul	2011
48	GRAPPA	Grasl	2012
49	SGI for Rectilinear Forms	Trescak, Esteva & Rodriguez	2012
*50	Visual Interactive 3D Spatial Grammar	Hoisl & Shea	2012
*51	DESIGNA	Correia	2013
*52	Shape Grammar Implementation	Strobbe	2013
*53	GRAPE: U_{12} and U_{13} SG Interpreter	Grasl & Economou	2013
54	GRAPE: Agent-based Rule Decision	Grasl & Economou	2014
*55	ShaDe	Ruiz-Montiel	2014
56	Dirksen Grammar	Park & Economou	2015
57	Rabo-de-Bacalhau Grammar	Strobbe & Eloy	2016
*58	SortAL GI	Dy & Stouffs	2017
59	Multipurpose Chair Grammar	Garcia & Letao	2018
60	Portmino	Ligler & Economou	2019
*61	Shape Machine	Hong & Economou	2020

Note: Those interpreters listed above with the marks "*" are general-purpose shape grammar interpreters

implementations of very specific grammars for a very specific design or research purpose. This is not an accident; the very core values that their underlying shape grammar formalism has promised, the miraculous calculations with shapes, the visual treatment of emergence and ambiguity, the seamless interface in design workflows, are all still in want. Surprisingly, the original account of the list of the shape grammar interpreters by Gips [1] already accounted for applications that claimed recognition of subshapes and deployment in 2D and 3D space and yet, even after all these years, general-purpose shape grammar interpreters seem still limited by shape types, types of transformation, complexity of geometry, matching conditions, counting of non-equivalent parts, semantic information, interface design and so forth. Still, the situation is not as grave as it may seem. It is suggested here that beyond this seemingly long list of technological hurdles, the operation of embedding, that is, the implementation of the mathematical concept of the "part relation" between two shapes, or equivalently, between two drawings, or between a shape and a design, is the single major obstacle to take on.

The work here focuses exactly on this front foregrounding the criteria that characterize the underlying machinery for the most important aspect of the shape grammar interpreter implementation, namely, the conventions of matching under which a shape can be a part of a design. These calculations follow the general structure of the calculations involved for tackling embedding outlined in Stiny [5, 6] and Krishnamurti [7, 8] but are recast here in a slightly different format following in part the lattice of schemata rules outlined in Stiny [9]. This modified structure consists of three matching conditions starting from simple queries of determinate matchings of embedded shapes under restricted conditions, to a rising complexity of determinate and indeterminate matchings without any restrictions, all characterized by four types of transformations under which each matching occurs. It is further suggested here that these general calculations for these three conditions of embedding (including their twelve subcases for the four types of transformations), along with the calculation for the determination of the non-equivalent mappings for each type of embedding, plus the familiar calculations for the characteristic signature of the shape grammar formalism—the maximal representation of shape (for each type of shape)—altogether do provide a map of the five families of calculational obstacles that general-purpose shape grammar interpreters face. The work here considers all three embedding conditions cast within the singular rule schema $x \rightarrow y$ [6]. A visual map of the work accomplished in the field in terms of the current state of embedding and the work ahead is given in the end. Aspects of interface design and integration to current work design workflows are deliberately left aside.

2 Requirements of a Shape Grammar Interpreter

A computer implementation of a shape computation requires the implementation of five distinct processes, all intimately involved in the recognition and replacement of a shape under a given transformation and all encoded in the structure of the shape algebras U_{ij} that shape rules are defined in [6]. More specifically, for u, v and W shapes, a shape rule $u \rightarrow v$ and the shape W defined as the current design, the operation that a shape grammar interpreter should process is:

$$\begin{aligned} &\mathit{if}\ f(u) \leq W : \\ &\qquad W = W - f(u) + f(v) \end{aligned}$$

or, a) Encode the shapes u, v and W in the smallest number of basic elements that can specify them; b) Inquire whether there is transformation f that embeds the shape $f(u)$ in W, and if yes; c) Subtract the shape $f(u)$ from W; d) Add the shape $f(v)$ in W; and e) Repeat the above processes for all applicable transformations of the shape $f(u)$ in W. A visual example is shown in Fig. 1 to demonstrate the five procedures outlined above underlying a shape replacement.

The example illustrated in Fig. 1 features a shape rule applied under an isometry transformation, that is, a transformation that keeps shape and size invariant while varying handedness and position. In this case, there are eight f transformations that

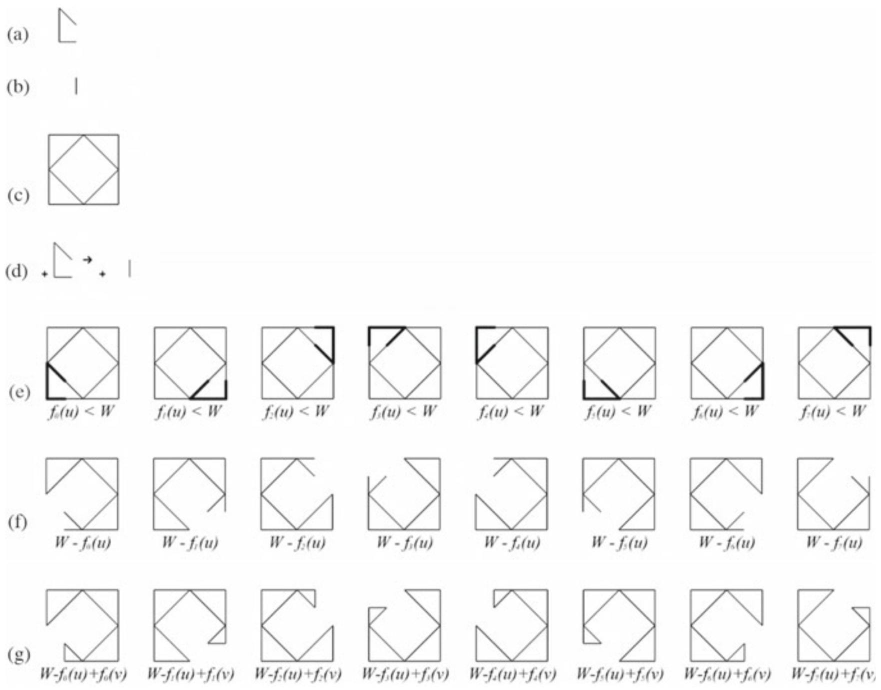

Fig. 1 A shape computation. **a–c** Shapes u, v, and W; **d** Shape rule $u \rightarrow v$; **e** Eight matches of the shape $f(u)$ in W; **f** Subtractions of the eight instances of the shape $f(u)$ from W; **g** Additions of the eight instances of the shape $f(v)$ to the corresponding eight instances of the shape $W - f(u)$

embed the shape $f(u)$ in the shape W (that is, make the shape $f(u)$ part of the shape W. The implementation of each of these five processes in a shape grammar interpreter brings its own set of problems and some more than others. A brief description of each process is given below.

The first process of encoding the shapes u, v and W in maximal representation, that is, in the smallest number of basic elements that can specify a shape [6], is to ensure that the shapes have a unique specification so that they can be compared and acted upon. The maximal representation of shape is typically defined in different ways depending on the dimensionality of the shape, that is, 0-, 1-, 2- and 3-dimensions, and its type, that is, line, arc, conic, Bezier, etc., requiring in essence different algorithms for a maximal point representation, maximal line representation, maximal curve representation, maximal plane representation, maximal surface representation, maximal solid representation, and so on [6]. For most shape grammar implementations, the maximal representation of shape is implemented with various approaches and typically, by a combined usage of operations (computer programs) of shape instantiation and shape addition or shape subtraction [10]. However, for more complex geometries, such as curves, surfaces, and solids, it is difficult to obtain the maximal representations of the corresponding elements of the shapes [11] and there is still a large number of shape types to be addressed. A different kind of problem might arise when shapes are perceptually similar but mathematically different and the implementation might seem to fail or otherwise cause confusion to users.

The second process of inquiring whether there is a transformation f that embeds the shape $f(u)$ in W is the most critical—and elusive—process for most of the shape grammar interpreters. The part relation between shapes is typically achieved by checking the boundaries of shapes [12]. The matching transformation, f is typically a Euclidean transformation—but more generally, a transformation belonging to the Euclidean, affine and projective geometries. Most detrimentally, most of the interpreters adopt database query [13] to simulate the desired transformation but this method, powerful as it may appear, it is severely limited because it assumes that a shape can be decomposed and represented as a finite set of subshapes and therefore violates the fundamental definition of shapes. Additionally, most of these matching calculations in affine and projective spaces accumulate a rounding error so fast that the matching results are often useless.

The third process of subtracting the shape $f(u)$ from W is based on the detection of shape boundaries. Chase [15] has listed 13 cases of two input lines with labeled endpoints so that the system can derive the results with three different procedures. The implementation of shape subtraction for lines has been done by Krishnamurti [7, 10] and has been broadly adopted in other interpreters. However, this shape operation is highly related to shape type and the complexity of implementation increases as the dimensionality of shape and the corresponding dimensionalities of space that the shapes are defined in are both increasing too. As above, shape types captured by higher degrees might cause severe precision errors and make the system unstable. For instance, the calculation of descriptors of high degree curves such as Bezier curves can be heavy because the system has to resolve the coefficients of high degree polynomials [16].

The fourth process of adding the shape $f(v)$ to $W - f(u)$ is similar to subtracting. As above, Chase [15] has listed out 13 cases of two input lines with labeled endpoints so that the system can derive the results with three different procedures. Similarly, the implementation of shape addition for lines has been done by Krishnamurti [7, 10] and has been broadly adopted in other interpreters as well. The same problems that are encountered in the implementation of the subtraction operation are encountered here too.

The fifth process of repeating the above processes for all applicable transformations of the shape $f(u)$ in W is straightforward for all shape grammar interpreter. The recursion can be implemented by re-assigning the result $W - f(u) + f(v)$ from the previous iteration back to the same variable W for the next iteration. The formal expression can be written as:

$$W_i = W_{i-1} - f_{i-1}(u) + f_{i-1}(v)$$

so that the result is:

$$W_N = W_{N-1} - f_{N-1}(u) + f_{N-1}(v)$$

after N iterations, where $f_i()$ represents the applied transformation for the i^{th} iteration.

3 Calculating Embedding

Shape matching in computer-aided design (CAD) systems is enabled by a database query requesting the retrieval of shapes from a CAD database. Surprisingly, shape matching under a given Euclidean, affine or linear transformation (visual matching), the most characteristic part of the shape grammar formalism, is entirely absent from current CAD systems. It is argued here that the conditions under which these visual matchings can occur and the calculations to implement them are the most important requirements for shape grammar interpreters to process rule applications and the single obstacle to merge shape grammar interpreters with generative CAD modelers.

The first condition to specify is the transformations themselves: for a shape u, the shape $f(u)$ to be embedded in a shape W can be modeled by four types of transformations: a) isometry transformations including translations, rotations, and reflections; b) similarity transformations including isometry transformations, scale transformations and their combinations; c) affine transformations including similarity, stretch, compress, and shear transformations and their combinations; and d) linear or projective transformations including affine transformations, one-point, and two-point perspective transformations and their combinations. The rising hierarchy of the matching transformations f is given in Fig. 2 for a shape in the form of a capital K.

The calculations for the transformations f of a shape u so that the shape $f(u)$ can be embedded in a shape W involve the following five processes:

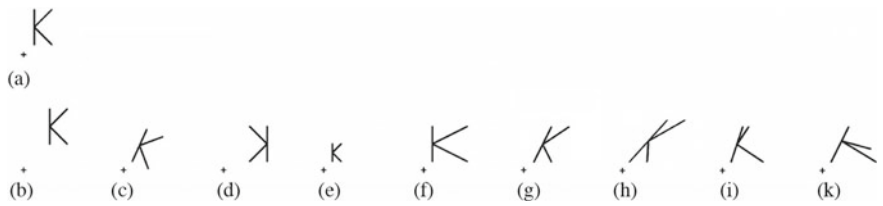

Fig. 2 Types of linear transformations. **a** Identity; **b** Translation; **c** Rotation; **d** Reflection; **e** Scale; **f** Stretch; **g** Shear; **h** Stretch and shear; **i** One-point perspective; **k** Two-point perspective

1) Encode the shapes u and W in their maximal representation;
2) Calculate the determinate match of embedded shapes f(u) whose boundaries are all recorded in the dataset of W (restricted embedding);
3) Calculate the determinate match of embedded shapes f(u) whose boundaries are not recorded in the dataset of W (unrestricted embedding);
4) Resolve the indeterminate matching for embedded shapes whose boundaries are not recorded in the dataset of W that in addition can be embedded into W in infinite ways (indeterminate embedding);
5) Count all non-equivalent matchings of the Left-Hand Side (LHS) and the Right-Hand Side (RHS) of the rule.

A brief description of the processes and conventions pertaining to the calculation of the maximal shape representation is given in the previous section. Here the focus is given in the calculation of the transformations f that make the shape u embedded in a shape W. A pictorial description of the types and instances of visual matching follows below.

3.1 Restricted Embedding

The calculation of the determinate embedding of shapes whose boundaries are defined in the target shapes (restricted embedding) has been viewed as the most important criterion for the calculation of the inverse transformations because the ways digital tools represent shapes are discrete and object-based [3, 17]. An example of a query of a restricted embedding is shown below in Fig. 3. Note that the shape $f(u)$ has boundary points that are all well defined as registration points in the shape W.

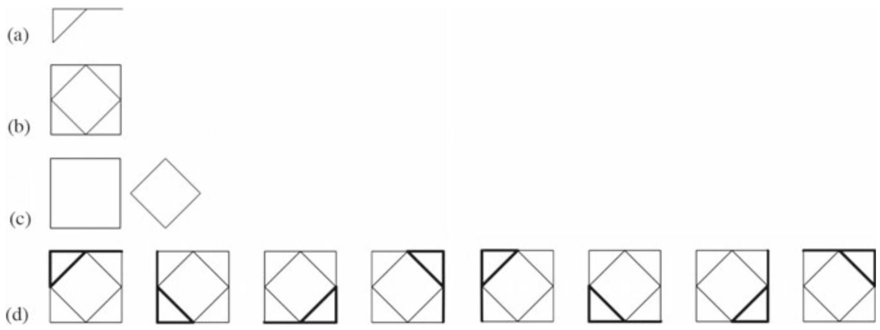

Fig. 3 An example of a restricted embedding of a subshape $f(u)$. All boundary points of the shape $f(u)$ are registration points in the shape W. **a** Shape u; **b** Shape W; **c** Registered objects; **d** Eight transformations f embedding the shapes $f(u)$ in W

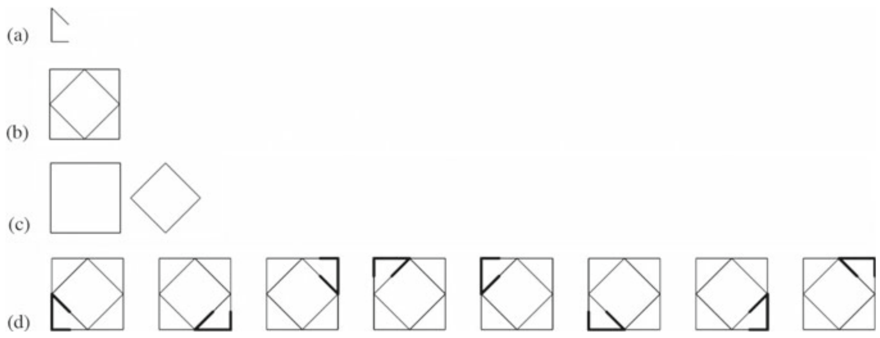

Fig. 4 An example of an unrestricted embedding of a shape $f(u)$. The boundary points of the shape $f(u)$ are not registration points in the shape W. **a** Shape u; **b** Shape W; **c** Registered objects; **d** Eight matchings of the shapes $f(u)$ in W

3.2 Unrestricted Embedding

The calculation of the determinate embedding of shapes whose boundaries are not recorded in the dataset of W (unrestricted embedding) is more involved and limited progress has been recorded on this front. An example of a query of an unrestricted embedding is shown below in Fig. 4. The query consists of a composite line of three segments showcasing two vertices that are registered in the dataset of the two squares and two that are not.

3.3 Indeterminate Embedding

The calculation of the indeterminate embedding of shapes whose boundaries are not recorded in the dataset of W can become involved too [5–8]. For such cases of shape matching, the system should be able to detect the indeterminate condition and offer processes to resolve the infinite possible matches. An example of an indeterminate query is shown in Fig. 5. The LHS shape “k” can be matched in infinite ways under a similarity transformation.

3.4 Counting Non-equivalent Embeddings

Finally, the counting of all non-equivalent embeddings of the LHS and the RHS of the shape rule completes the requirements for the calculations of the embedding operation and its interface with the transformations under which a shape rule applies. An example of a calculation of non-equivalent matchings of the LHS and

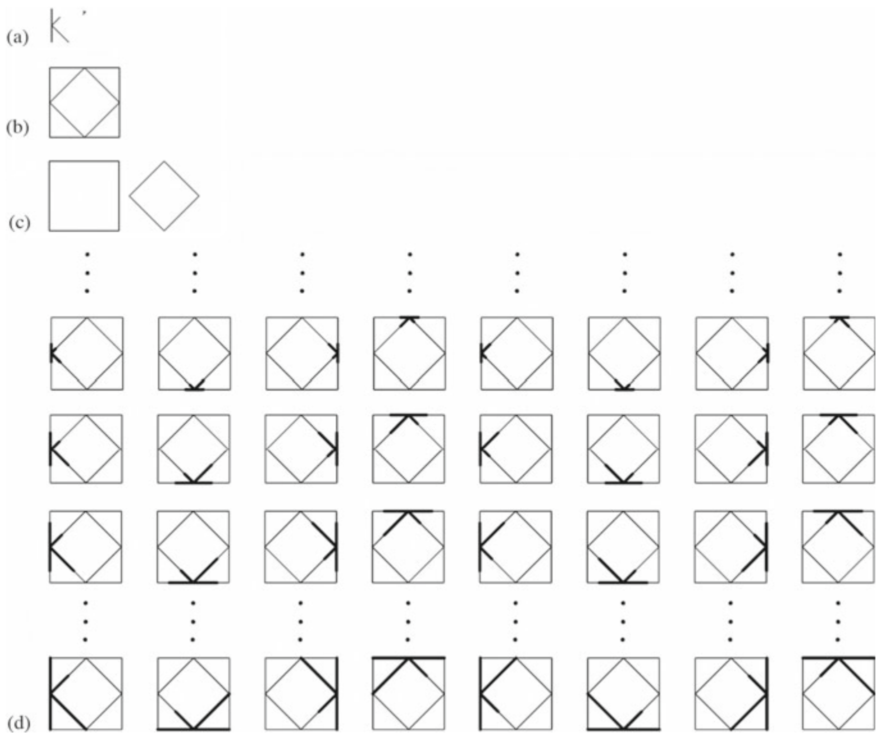

Fig. 5 An example of an indeterminate embedding of a shape $f(u)$. **a** Shape u; **b** Shape W; **c** Registered objects; **d** Eight infinite families of similarity transformations f embedding the shapes $f(u)$ in W

the RHS of the shape rule is shown in Fig. 6. In this example, the eight matches of the LHS shape are reduced to four non-equivalent matches, which are expanded again to eight matches of the RHS shape.

4 Three Systems

A sketch of the increasing complexity underlying the implementation of the modules required for the calculation of shape recognition and replacement, along with the modules required for the calculation of the various implementations of the mathematical concept of the part relation between two shapes, is given in Table 2. Significantly, the arrangement of these modules in three successive sets provides a common framework of rule-based computation that foregrounds the similarities and differences between generative geometric modelers, set grammar interpreters and shape grammar interpreters, respectively.

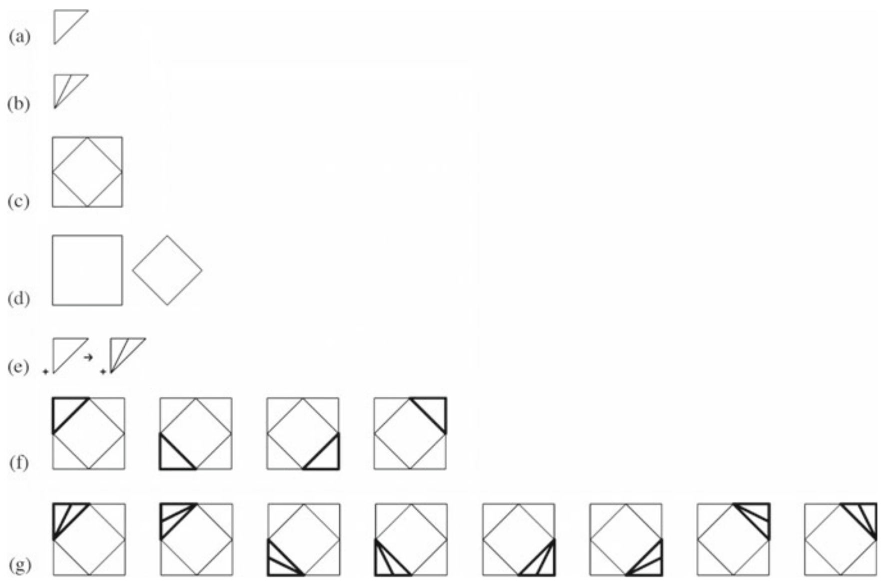

Fig. 6 An example of a calculation of counting non-equivalent embeddings and replacements of a shape $f(u)$ shape by the shape $f(v)$. **a** Shape u; **b** Shape v; **c** Shape W; **d** Registered objects; **e** Shape rule $u \rightarrow v$; **f** Four embeddings of the LHS in W; **g** Eight embeddings of the RHS in W

The two modules given in the first part of Table 2, the rule editor and the rule compiler, provide the underlying functionality for the implementation of generative systems (rule-based systems) irrelevant of the actual symbols, strings, shapes and so on, involved in the computation [17]. A symbolic rule editor and a symbolic rule compiler can be extended to a shape rule editor and a shape rule compiler by implementing the five modules in the second part of Table 2.

The five modules given in the second part of Table 2 are commonly found in most geometric modelers (CAD systems). These modules include geometric modeling functions to allow instantiation of shapes, modification of shapes and database query. The interpreters based on the integration of these five modules with the computational framework of the rule editor and the rule compiler provide rule-based systems for symbolic shapes and are typically classified as generative geometric modelers, see for example, Cellular Automata [18], L-system [19], CityEngine [20], and several more.

The five modules in the third part of the Table 2 outline the fundamentals of an advanced shape query system to make a symbolic generative modeler a shape grammar system. The major area of this part of the table—and the least populated region of the whole table—focuses on the implementation of the mathematical concept of the part operation ($\leq$) for shapes of the shape grammar interpreter, including the three subcategories of matching and the four transformations under which the matching is enabled. Note that these modules will be different for

Table 2 Requirements of a shape grammar interpreter

				Generative system	Generative geometric modeler	Set grammar interpreter	Shape grammar interpreter
Rule Editor				●	●	●	●
Rule Compiler				●	●	●	●
Shapes					●	●	●
Relations (Database query)					●	●	●
Operation: Transformations					●	●	●
Operation: Subtraction (−)					●	●	●
Operation: Addition (+)					●	●	●
Maximal Representation of Shape						●	●
Embedding (≤)	Determinate	Restricted	Isometry			●	●
			Similarity			●	●
			Affinity			●	●
			Linearity			●	●
		Unrestricted	Isometry				●
			Similarity				●
			Affinity				●
			Linearity				●
	Indeterminate		Isometry				●
			Similarity				●
			Affinity				●
			Linearity				●
Counting Non-equivalent Mappings					●	●	●

different types of shapes because the implementation of maximal representation of various types of geometries (lines, arcs, etc.), their embedding conditions, the definitions of addition and subtraction in terms of their parts, and even the transformations required for different dimensions, requires often radically different solutions. Clearly, a general account for the state-of-the-art of shape grammar interpreters requires different accounts of the implementation of distinct types of shapes, for example, lines, conics, Bezier curves, NURBS, and so forth in the algebra U_{12}, and other types of shape in different algebras too. A brief discussion of the current state of general-purpose shape grammar implementations of these five modules for shapes made up of lines in the algebra U_{12} is given below.

4.1 Case Studies of Shape Grammar Interpreters of Lines in U_{12}

The maximal representation of shapes consisting of lines has been successfully implemented in SGI by implementing line addition operation (Boolean union for lines), and most of the interpreters have been following this method—see for example, SGS [15], GEdit [13, 14], ShaDe [21] and Shape Machine [22]. Some interpreters have adopted a graph representation of maximal lines such as GRAPE [23] and SortAl GI [24] and use hypergraphs [25] to simulate the procedure. And still others, such as Curve-based SGI [26] and QI [16] use algorithms to achieve the maximal representation for Bezier curves and in doing so they solve the problem of maximal representation for lines because straight lines can be viewed as degree one Bezier curves.

The restricted embedding of shapes consisting of lines has been successfully implemented by adopting the 3×3 matrix algorithm in SGI and in particular for Euclidean and affinity transformations. SGI uses the 3×3 matrix method to derive the possible transformations that can make $f(u) \leq W$ true with two given registration points, and the algorithm of the 3×3 matrix is adopted by most of the interpreters such as SGS, GEdit, Curve-based SGI, QI, SGIRF [27], ShaDe and Shape Machine. Significantly, the two registration points, which are the points registered in the database of the design, can provide enough information to calculate transformations up to the Euclidean transformations and SGS, Curve-based SGI, Shape Machine and other interpreters have adopted three registration points to do so. Still, three points are not enough to calculate the complete range of all linear transformations: The new transformations that are added in the list, the one-point and two-point perspectivities, require four distinguishable points and a 9×9 matrix. Despite the seemingly straightforward extension of the approach in this new domain, the 9×9 matrix requires a heavy computation load taxed by severe precision or rounding errors. Some interpreters such as SortAl GI and GRAPE are looking for the data description of shapes to simulate the transformations. Shape Machine uses a non-numerical algorithm to derive the linear transformation and successfully avoids these issues. Significantly, the interpreters that provide maximal representation of shapes and restricted embedding are classified as set grammar interpreters [3, 17, 28] following the theoretical distinction between set grammars and shape grammars [29].

The unrestricted embedding foregrounds the main difference between the set grammar interpreters and the shape grammar interpreters as the productions of the former (and the generative geometric modelers at large) are symbolic and thusly indifferent to the richness of shape recognition and the open-ended calculations with shape rules. The unrestricted embedding of shapes consisting of lines has been partially implemented in SGI, GEdit and SGIRF. SGI provides a partial foundation for the unrestricted embedding. GEdit and SGIRF includes projection intersections as the registration points to achieve a partial unrestricted embedding only for a limited range of shapes. Shape Machine appears to succeed in this front integrating and extending SGI and GEdit's existing algorithms to offer a general solution of this type of matching for lines.

The indeterminate embedding of shapes consisting of lines is a set of cases that the matching results are indeterminate until users provide more information to consolidate the results. The indeterminate embedding of shapes consisting of lines has been partially implemented in Grape and SortAl GI by specifying floating endpoints but only for a specific class of shapes, including the K-shape. Shape Machine appears to be the only interpreter able to deal with the indeterminacy of rules by detecting the special cases and offering a structure for users to pass the parameters to the system.

The enumeration of non-equivalent matchings of shapes consisting of lines for all three types of embedding has been implemented in various ways using diverse approaches pertinent to the representation of the shape and the type of embedding. GEdit uses diagonal vectors to manage the matching results and remove the visual equivalent results. GRAPE eliminates the equivalent results by checking the symmetry of the graph representations. SortAl GI uses a predefined description of shapes to prevent the system from equivalent counting. Curve-based SGI, QI, ShaDe and Shape Machine remove the equivalent matches by checking the pictorial equivalency [5] between the matches.

5 Discussion

The review of the shape grammar interpreters within the lens provided in this work showed the profound complexities that are involved in the implementation of the part operator "$\leq$" (embedding)—and the different ways that this operator can be implemented. Unrestricted and/or indeterminate embedding are two of the hallmarks of the shape grammar formalism and the only general-purpose shape grammar interpreter that appears that successfully tackles this problem for shapes consisting out of lines is the Shape Machine. Shapes consisting of lines, arcs and their combinations—the subject matter of Euclidean geometry and an expressive space for design—also appear that they have been successfully tackled in Shape Machine so far for all Euclidean transformations along with some promising work on conics in affine and projective geometries [30].

Future development of shape grammar interpreters will be highly related to the types of geometry they will support. The implementation of the corresponding maximal representations and Boolean operations for different kinds of geometries requires the descriptors of corresponding geometries as their underlying structures. Still, finding the geometry descriptors can be a difficult task because there are many different types of shapes—lines, arcs, conics, Bezier, etc. Defining the range of geometry types that are commonly used in a design process might help reduce the complexity of the implementation. Another possibility is to use rational elements to approximate complex geometries. For instance, a NURB curve might be hard to model with the descriptor but it can be approximately decomposed into a composition of arcs. By adopting this concept, a complex geometry might be decomposed into a composition with rational elements such as lines, arcs, conic elements, planes, spheres, ellipsoids and so forth. Along with the increased complexity of the geometries, management of the performance will be one of the main tasks in the future. A major bottleneck of performance will surely be related to the calculations involved in the various procedures of embedding outlined above. The complexity of embedding follows the number of the registration points of the current design, W. The method of 3×3 matrix allows the system to use two registration points to achieve a Euclidean transformation, thus, the complexity of the embedding under Euclidean transformation is $O(n^2)$ where n is the number of the registration points of W. For affinity transformation, the complexity increases to $O(n^3)$ for requiring three registration points. For a linear transformation, the complexity increases to $O(n^4)$.

A second trajectory for the future development of shape grammar interpreters will be highly related to parametric shape representations. The interpreters reviewed in this paper are mostly based on transformational geometry because geometric transformations provide a precise recognition match. As a production system, the accuracy of subshape matching is important because the system has to guarantee that the computational tasks can be precisely executed. Thus, parametric shape grammar interpreters will require a certain level of accuracy to make sure the parametric computation process is precise too. Graph representation of shape cannot satisfy this accuracy in advance because it is too abstract to guarantee the uniqueness of shapes. For example, the k-shape in Fig. 7 cannot be matched through a linear transformation: the connection of its boundaries makes a concave quadrilateral and there is no geometric transformation that can match a convex quadrilateral to a concave quadrilateral; and the graph representation could not help either because it would not be able to distinguish between a k-shape and a ψ-shape. As in shape grammar interpreters, a unique representation of a parametric shape is required to implement a parametric shape grammar interpreter [31].

The expansion of the range of geometry descriptors for various types of shapes in different dimensions and the unique representation of parametric shape are two of

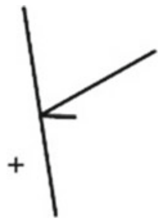
Fig. 7 Parametric deformation of the k-shape

the possible directions for the implementation of shape grammar interpreters. Additional directions pertaining to the design of interfaces for these systems and their seamless integration with current and future modes of practice provide a bright future for their development.

References

1. Gips J (1999) Computer implementation of shape grammars. In: Proceedings of the workshop on shape computation, MIT. https://www.academia.edu/3089939/Computer_Implementation_of_Shape_Grammars. Accessed 20 Apr 2020
2. Chau HH (2004) Evaluation of a 3D shape grammar implementation. In Gero JS (ed) Design computing and cognition 2004. Kluwer Academic Publishers, Dordrecht, pp 357–376
3. Chase SC (2010) Shape grammar implementations: the last 35 years. In: Proceedings of the 4th international conference design computing and cognition, Stuttgart. https://www.researchgate.net/publication/227441758_Spatial_grammar_implementation_From_theory_to_useable_software. Accessed 20 Apr 2020
4. Eloy S, Pauwels P, Economou A (2018) A Promises of shape grammars in advances in implemented shape grammars: solutions and applications. AI EDAM Special Issue: Artif Intell Eng Des Anal Manuf 32(2):131–137. Cambridge University Press
5. Stiny G (1975) Pictorial and formal aspects of shape and shape grammars. Interdisciplinary Systems Research, Birkhäuser
6. Stiny G (2006) Shape: Talking about Seeing and Doing. MIT Press, Cambridge
7. Krishnamurti R (1982) SGI: a shape grammar interpreter. Centre for Configurational Studies. The Open University, Milton Keys
8. Krishnamurti R, Stouffs R (1997) Spatial change: continuity, reversibility and emergent shapes. Environ Plann B Plann Des 24(3):359–384
9. Stiny G (2011) What rule(s) should I use. Nexus Netw J 13(1):15–47
10. Krishnamurti R, Giraud C (1986) Towards a shape editor: the implementation of a shape generation system. Environ Plann B Plann Des 13(4):391–404
11. Krishnamurti R (2015) Mulling over shapes, rules and numbers. Nexus Netw J 17(3):927–945
12. Earl C (1997) (1996) Shape boundaries. Environ Plann B Plann Des 24:669–687
13. Tapia M (1999) A visual implementation of a shape grammar system. Environ Plann B Plann Des 26:59–73
14. Tapia (1996) From shape to style. Shape grammars: issues in representation and computation. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada
15. Chase SC (1989) Shapes and shape grammars: from mathematical model to computer implementation. Environ Plann B Plann Des 16:215–242
16. Jowers I, Earl C (2011) Implementation of curved shape grammars. Environ Plann B Plann Des 38(4):616–635

17. Krishnamurti R, Stouffs R (1993) Spatial grammar: motivation, comparison, and new results. In: CAAD Future 1993, conference proceedings, Pittsburgh, 1993
18. Wolfram S (2002) A new kind of science. Wolfram Media Inc
19. Lindenmayer A (1968) Mathematical models for cellular interaction in development I. Filaments with one-sided inputs. J Theor Biol 18:280–289
20. Muller P, Wonka P, Haegler S, Ulmer A, Gool LV (2006) Procedural modeling of buildings. In: SIGGRAPH 2006: ACM SIGGRAPH 2006 papers, July 2006, pp 614–623
21. Ruiz-Montiel M, Belmonte MV, Boned J, Mandow L, Millán E, Badillo AR, Pérez JL (2014) Layered shape grammars. Comput Aided Des 56:114–119
22. Hong TK, Economou A. Shape machine. Accessed 14 Apr 2020. https://shape.design.gatech.edu/Research/Projects/2019_ShapeMachine/index.html
23. Grasl T, Economou A (2013) From topologies to shapes: parametric shape grammars implemented by graphs. Environ Plann B Plann Des 40(5):905–922
24. Dy B, Stouffs R (2018) Combining geometries and descriptions: a shape grammar plug-in for Grasshopper. In: eCAADe 2018, Lodz, Poland, vol 2, pp 499–508
25. Earl C, Johnson J (1981) Graph theory and Q-analysis. Environ Plann B Plann Des 8:367–391
26. McCormack JP, Cagan J (2006) Curve-based shape matching: supporting designers' hierarchies through parametric shape recognition of arbitrary geometry. Environ Plann B: Plann Des 33(4):523–540. Des Stud 25(1):1–29
27. Trescak T, Rodriguez I, Esteva M (2009) General shape grammar interpreter for intelligent designs generations. In: Proceedings of the 2009 6th international conference on computer graphics, imaging and visualization: new advances and trends, CGIV 2009, pp 235–240
28. McKay A, Chase S, Shea K, Chau HH (2012) Spatial grammar implementation: from theory to usable software. Artif Intell Eng Des Anal Manuf 26:143–159
29. Stiny G (1982) Spatial relations and grammars. Environ Plann B Plann Des 9:113–114
30. Economou A, Hong TK, Ligler H, Park J (2019) Shape machine: a primer in visual composition. Cultural DNA: Computational Studies on Cultural variation and Heredity. KAIST, South Korea, pp 79–102
31. Economou A, Yu J, Park J. Shape signature. https://shape.design.gatech.edu/Research/Projects/2019_ShapeSignature/index.html. Accessed 20 Apr 2020