What's in a Shape

An Algorithm for Finding Shapes in Shapes

Joaquim Reis
Instituto Universitário de Lisboa (ISCTE-IUL)
ISTAR-Iscte
Lisboa, Portugal
joaquim.reis@iscte-iul.pt

Abstract — **This paper describes a simple two stage algorithm for finding emergent sub-shapes in shapes, in the context of shape grammar systems. Matching the shape in the left side of a rule of a shape grammar with parts of a shape in a design in process to decide if the rule is applicable, is its main purpose. Shape grammars have been used to represent the knowledge behind the creative work of architects, designers and artists and allow the implementation of computational mechanisms to analyze and synthesize designs of visual languages, with obvious applications to design, including for marketing. Their computational mechanisms can include the detection of emergent sub-shapes. The algorithm we propose performs this task and is a core component of a system, described in our past work, that allows users to build their own shape grammars and use them.**

Keywords - Shape Grammars; Artificial Intelligence; Design.

I. INTRODUCTION

The shape grammar formalism can be used to synthesize, as well as, to analyze, designs of design languages. Shape grammars are related to design and, as well as the symbolic/text phrase grammars, they can be considered a member of the "family" of grammars. Both can be considered production systems, where replacement rules are used to recursively generate phrases of a language. But the similarities end up here. Firstly, shape grammars are inherently visual. And, secondly, they accommodate aspects of emergency, *i.e.*, the possibility of generating shapes not explicitly introduced by the application of the rules. Maybe the differences are not restricted to these two features, but these two are very important in the field of arts, especially in design.

Not infrequently, an artist stops in the middle of a creative work to look to, and appreciate, or assess, the work done so far, and it happens to discover some emergent detail, or shape, that wasn't there before. And that particular detail comes to be inspiration for him/her for the next step of the creative work. Many other activities, visual or not, involve discovering emergent patterns too, from the most noble scientific research to the most commonplace activities, including entertainment ones[1]. Emergency is important in creativity, in creating new (and useful) ideas, designs, realizations, artifacts[2].

[1] Yes, those! For instance, see the puzzles to discover "how many" triangles, or squares, or whatever "are in the picture?"

[2] Emergency is everywhere. It is present in the Nature too, so that's probably why it is the nature of Nature to be so creative!

This paper approaches the problem of discovering emergent shapes within given shapes. The human brain and eye seem efficient in solving it, although prone to failures and mistakes too, but the problem seems to be defying from a computational perspective. Extending the path other researchers followed, we propose an approach to the problem in the form of an algorithm, applicable in the context of shape grammars with shapes composed of some basic geometric elements. The internal computational infrastructure used by this algorithm is described in another paper, twin to the present one [1].

In the following, we summarize: what shape grammars are and a brief state of the art (section II); the short and relevant history of the proposed approaches to the problem, including why the problem is important for the application of rules (section III); our previous work in the area and our goals (section IV); then we expose the algorithm proposed, show some data from examples using it and compare it with alternatives from other researchers (section V). Finally, we draw conclusions and show intended future work (section VI).

II. THE SHAPE GRAMMAR FORMALISM

Shape grammars were introduced by George Stiny and James Gips in the 1970s, and the focus of the related research is in representing and applying knowledge about languages of design basically through the use of concepts from formal grammars and rule-based/production systems [2] [3]. A shape grammar is composed of (1) a set of basic shapes, the shape alphabet, (2) a set of rules, and (3) a special shape, the initial shape, used to trigger rule application.

The mechanics of rule application and shape generation is as follows. In a rule, A→B, the left side, or antecedent, A, and the right side, or consequent, B, are shapes. A rule, when applied, substitutes the shape on the right side for the shape on the left side, in the original shape, or design, or composition, as described further. Applicable rules may recursively be applied to a shape, until there are no more rules to apply, or some termination condition holds. A shape computation, or shape derivation, is a sequence of shapes in which each shape, except for the initial shape, is generated from the previous by the application of a rule of the shape grammar. A rule A→B is applicable to a shape, C, if there is a similarity geometric transformation (a translation, a rotation, an uniform scaling or a combination of these) T, which, when applied to shape A makes A equal to a part of C, *i.e.*, a geometric transformation T

such that T(A) ≤ C, where ≤ denotes a sub-shape relation[3]. Application of the rule results in a new shape, C', that is computed subtracting from C the result of applying the transformation T to A, and then adding to C the result of applying T to B, *i.e.*, the resultant design will be C' = (C – T(A)) + T(B), where + and – denote the shape sum and shape difference (or subtraction) operations. It can easily be seen that ≤ is a cornerstone for our sub-shape problem.

In very brief words, the research area of shape grammars has been focused in conceptual and theoretical aspects, as the ones exposed in [3], in analysis, *i.e.*, the development of specific shape grammars of languages of design extracted from corpuses of designs in architecture, product design or painting, see [4] [5] [6], for instance, and in synthesis, *i.e.*, building specific shape grammars to define original languages of designs, as in [7] [8] [9]. More research include the development of algorithms for shape manipulation and rule matching and application processes, which are very interesting for us in the present paper, as in [10] [11] [12] [13] [14] [15] [16] [17] [18], where some of this papers focus also on appropriate interfaces and generic and reusable shape grammar interpreters, including for didactical purposes.

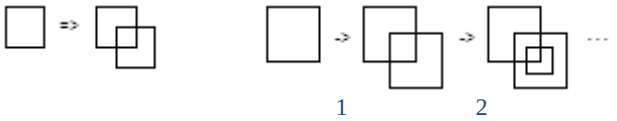


Figure 2- A shape grammar rule and results of 2 steps of a derivation.

III. An Interesting Problem and the Past Research

A classic example of emergence in shape grammars is shown in Figure 2. The rule says: if a square is found in the composition, add an equal square with its top and left sides intersecting the right and bottom sides of the one in the composition by its middle points. The shape after step 1 has two squares explicitly introduced, one is the initial shape, the other is the result of the first step. But there is also a third *emergent square*, which was detected and was the focus of application of the rule in step 2.

No object-oriented method[4], nor any classical CAD tool[5], just by themselves, can help to computationally implement such a detection mechanism. As found by Stiny and other researchers following the same research path, see [2] [3] [10] [11] [12] [13], the correct approach to this kind of problem lies on using operations (+, – and ≤) on, and representations of, shapes according to a special kind of algebras called algebras of *maximal shapes*. Then, the computational mechanism used to match the left side of a rule with the composition can be made to detect embedded emergent shapes.

The +, ⁻ and ⩽ operations are part of algebras usually classified as U_{ij} algebras, as its basic elements are points, lines, planes and solids that are defined in dimensions i = 0, 1, 2 or 3 and combined and manipulated in dimension j ≥ i, see [3]. For i > 0 these basic elements have finite non zero content (either length, area or volume) and boundaries that are shapes in the algebra $U_{i-1,j}$. A *maximal shape* is a shape that is composed by a finite set of basic elements, which are *maximal* in combination, each one being independent of the others (*i.e.*, with no overlap among them). This means that, except for points, any maximal element of a maximal shape is the representation of an infinite number of (non-maximal) elements, of the same dimension, contained in it. For instance, a maximal line represents an infinite number of line segments limited by any pair of non-coincident points on the maximal line.

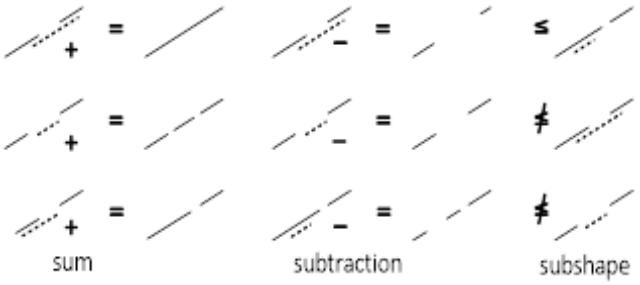


Figure 1- Some examples of maximal shape sum and subtraction and the sub-shape relation in a U_{12} algebra. The dashed line is a component line in a shape to sum to, subtract from, or test for sub-shape relation with, other lines component of another, target, shape.

Figure 1 shows examples of operations in a U_{12} maximal algebra (1 dimension max for shape elements in a 2-dimensional space), with lines only, in these example cases. Only cases with colinear lines are shown because it is the only situation in which operations can interfere with and modify the components and so be relevant for illustration[6].

The problem of sub-shape detection is, in the general case, a computationally hard one, see [18], and is equivalent in hardness to the subgraph isomorphism problem. In the compact summary research following, all papers address the problem we are interested in, of the recognition of emergent shapes. Papers [10] and [11] paved a research path to the appropriate computational representation of shapes in 2 dimensions and how shape operations should work on them. In [12], more precise definitions suggest computational representations for maximal shapes, including for more than 2 dimensions. In [13] the "formula" C' = (C – T(A)) + T(B) is analyzed, as well as the recognition of emergent shapes and the cases that occur in the determination of T. The following research papers we refer seem to assume and use always maximal representations. In [14], the first implementation of a shape grammar system able to detect emergent shapes, for rectangular shapes, is proposed. In [16] an algorithm is proposed for detecting emergent shapes which considers each, and every, intersection (concrete or

[3] From now on, we will treat ≤ as an operation, more specifically, a predicate that tests if the first operand is in the sub-shape relation with the second.

[4] If two objects are programmatically created, it's two objects, and no more.

[5] As said in [13], Computer-aided Design, or CAD, systems are often no more than systems that serve as repositories for *already designed* information. Classical CAD systems are helpful, but don't have the ability to accommodate the notion of change and rely only on a set of predefined shape static primitive elements, limited to no more than the *combination* decisions of the designer.

[6] Note as there can be some surprising cases. For instance, in the bottom example for the – operation, it happens that subtracting a line from another present in the target shape leaves the shape with more lines than before. And the first case for + sums a line but leaves the shape with less lines than before.

virtual/projected) between pairs of lines in the shape. In [17] an approach is proposed that uses graphs, more specifically, graph grammars, to represent shapes and shape grammars, relegating the problem to another of graph representation and manipulation. The computational complexity of the algorithms involved in different kinds of shape grammars, including in sub shape detection is analyzed in [18].

IV. The GSG System and Past Work

Our relevant past work includes GeoWin, a multi-agent system to build creative drawing compositions, where each agent has its own shape grammar defining its composition style and participates in a composition process [19]. The participation occurs with different coordination strategies, ranging from a totally cooperative and orderly way to an extreme competitive/antagonistic/egocentric way in a purely emergent manner. This system was supported by a primitive shape grammar interpreter built on top of an *ad hoc*, logic-like, forward-chaining rule language to express simple computations with shape grammar rules with predefined shapes and a small set of logic and arithmetic operators. This is still unfinished work as, in the meantime, more work needed to be done in refining the idea of a universal shape grammar interpreter in the realization of its fundamental mechanics of rule application including with less restricted kinds of shapes.

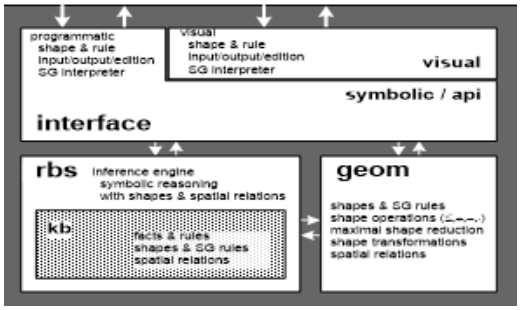


Figure 3- The GSG computational architecture.

In the path to improve in the direction mentioned above, we then embarked on a 'fundamental' approach on a project to build a prototype of a computational system centered around a "Generic Shape Grammar" interpreter, the GSG project, see the main initial work in [20] [21] [22]. This interpreter intends to be a core tool, a kind of an expert system *shell*, for shape grammar systems tasks for the use of students, artists, designers and architects, specifically allowing the definition of, and experimentation with, shape grammars, and is internally supported by an appropriate computational representation for shapes, shape rules and shape operations using the algebras of maximal shapes. This is ongoing work.

The GSG system computational architecture is depicted in Figure 3. The main components are a two-part *interface*, and two core sub-systems, a rule-based component, the *rbs*, centered around rules, sub-shape detection and rule application, and a computational geometry related one, the *geom*, centered in computational geometry methods. The visual interface is a part of the interface layer of the system, together with the symbolic/API (programmatic) interface and was appropriately described with examples in [23]. A third kind of interface, the textual/file interface, not shown, is also available. As described in [23], a notable point in GSG is that all shape grammar objects, *i.e.*, shapes, rules, and grammars, that come to existence in the system environment may have an independent (interface) representation in three possible formats: the symbolic (through programmatic objects), the visual (through graphical windows) and the textual (with an appropriate text/file external representation) format.

Sideway to the GSG system, work on the application of shape grammars to architectural project is shown in [24] [25] and work about the usability of interfaces of implemented shape grammar systems of different authors is shown in [26] [27]. In the latter, we have devised a set of requirements (see a summary of these in [23]) that can be used either to evaluate interfaces of existing shape grammar systems, or as a set of good rules to follow in the implementation of new ones for specific users (either students/beginners in the field of shape grammars, or architects, or designers, or artist specialists, or even users with additional programming expertise). Also, in another, twin, paper, the internal computational infrastructure needed by the algorithm presented here is described [1].

V. Finding Shapes in Shapes: An Algorithm

We now turn to the algorithm that is the subject of the present paper. We recall that the task of the algorithm to detect if a given shape, typically one in the left side of a rule, matches, or is contained, according to each and any similarity transformation to be determined, in another, target, shape, typically a shape of a composition, or design, be it the initial shape or a shape produced in the middle of a derivation process by the rules of a given shape grammar. The algorithm must identify all the sub-shapes in the target shape that match with the given shape, if there is any, and the corresponding similarity transformation associated to each matching case.

First, the limitations of the algorithm. In sub-shape detection and rule application, the similarity transformation T is the only kind of transformation applied to shapes[7]. Also, although a U_{12} algebra of maximal shapes is used, lines are the only type of shape component considered (points are not considered[8]). Additionally, component lines must bear, in the shape they belong to, at least two intersections, concrete or virtual/projected. Now, we present some definitions with illustrations to help clarify the structures used by the algorithm and the language used in its explanation. Respecting to internal representations used by the algorithm our options follow closely the proposals in [10] and [11]. As an illustration example, we show a shape, with seven lines, labelled line 1 to 7, in Figure 4-a).

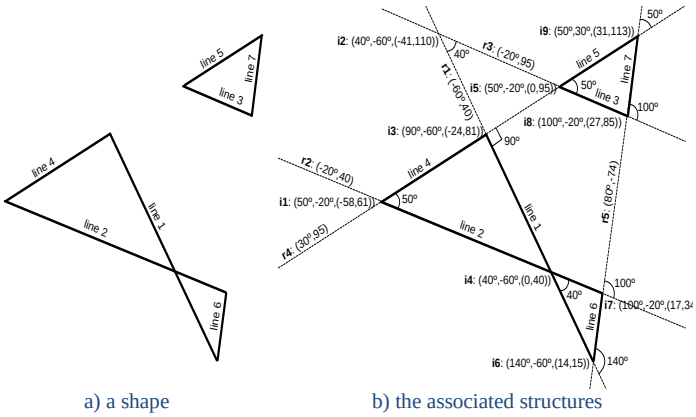
In GSG, a shape is a collection of maximal lines[9]. These are defined by its limiting points (a point is represented by its pair (x,y) of coordinates) and its slope and they are kept in a collection, sorted by its slope and coordinates of the limit points. As the algorithm deals with similarity, intersection

[7] This excludes the so-called parametric shape grammars (see [2]).

[8] Allowing points would, in fact, render the problem tackled simpler.

[9] There can be points too but, as said earlier, these are not considered here, as the algorithm works only with lines, at least for now.

angles and length proportions are very important, as their magnitudes are maintained through similarity transformations. So, at the time a shape is constructed, besides the data structures for representation of its maximal lines, certain additional data structures, described with more detail in [1], are internally created which are the appropriate representation infrastructure to support the algorithm, not only in sub-shape detection, but also further, in rule application. For a short illustration of the importance of the structures in the operation of the algorithm we depict them in Figure 4-b) for the shape in



a) a shape b) the associated structures

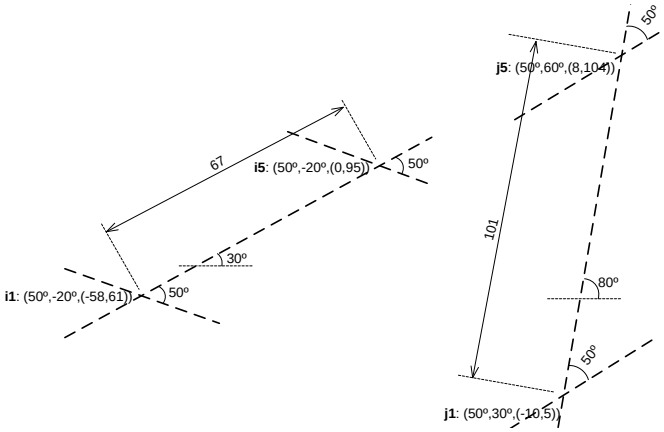
Figure 4- A shape and its associated internal structures in GSG.

Figure 4-a).

The structures are the straight-lines of support of (*i.e.*, containing) the lines of the shape and the straight-line intersections. The first are labelled r1, to r5 (there are five straight-lines) together with a descriptor indicating the slope and y-intercept of each one (for vertical lines, the x-intercept is used, instead). Intersections are pairs of straight-lines with an intersection point[10]. In the figure, intersections are labelled i1 to i9 (there are nine intersections) together with a descriptor indicating the primary angle of intersection, the slope of the straight-line with the smallest slope in the intersection and the pair (x,y) of the intersection point. Straight-lines and intersections are kept in specific sorted collections in the shape.

Suppose that we have a target shape, similar to the one in Figure 4-a), but translated and/or rotated and/or scaled, possibly containing more than one similar instance of it, and with more lines, and we want to determine if the shape in Figure 4-a) is a sub-shape of that target shape. How do we find if the first is contained in the second and what is the associated transformation, *i.e.*, the angle of rotation, scale factor and (Δx,Δy) translation? We have developed a two-stage approach where the first stage tests for similarity of the infrastructural elements, intersections of the straight-lines and then, only in the positive case, a second stage tests for containment of lines of the shape to match in the target shape. What we propose is first to try to match pairs of straight-line intersections, one pair in the shape to match and another in the target shape, instead of trying to match pairs of lines, as its done in [16], which would potentially increase the number of steps of the algorithm uselessly[11]. We consider this an advantage, as there will potentially be less intersections of straight-lines to consider than intersections (concrete or virtual/projected) for lines, at least in complex shapes[12], so the approach makes sense in terms of reduction of the number of steps of the algorithm. Moreover, our algorithm uses structures related directly with the shape elements (lines, in our case) and not any kind of additional intermediate data structure, like graphs, as used in [17] which (besides some advantage is terms of abstraction and some flexibility) can bring the disadvantage of precluding an easy use of domain heuristics in the search of possible sub-shapes[13].

As an example, in Figure 5-a), we show an intersection pair that is part of the shape in Figure 4-a) (with intersections i1 and i5) and another of a target shape, in Figure 5-b) (with intersections j1 and j5). An additional advantage of the algorithm is that of obtaining hints on the possible angles of rotation for T. Two intersections match if one has an angle of



a) Shape to match b) Target shape

Figure 5- Intersection pairs.

intersection, or its supplementary, that is equal to the angle of intersection, or its supplementary, of the other, apart an angle of rotation. This last angle is a hint. Still another advantage is having hints on the possible scale factors for T. A pair of intersections will match with another pair if each angle in the first is equal to another, different, in the other pair, apart from some angle of rotation of one pair. A pair of intersections has a shared segment of a shared straight-line, which has a length and, by matching two intersection pairs of different shapes, the ratio between the two lengths is a hint for a scale factor. For the translation, we can obtain a hint if, in the first place, we assume (Δx,Δy) displacements in a way to make an intersection point of one intersection pair coincident with an intersection point of the other intersection pair and, only after, determine the appropriate angle and scale factor hints. In the case shown in Figure 5, a match will be found with a translation from point (-60,65) to point (-10,5), followed by a rotation with a 50º angle,

[10] There will be more than one intersection when more than two straight-lines intersect at the same point.

[11] As correctly mentioned by [18] (section 6.1).

[12] If there are many colinear lines in a shape, considering intersections of lines (concrete or virtual/projected) leads to consider more times intersections than with straight-lines, although the total number of intersections in the shape is the same.

[13] If you use graphs to represent shapes the heuristics more easily usable are those of the graph domain, but not of the shape grammar domain.

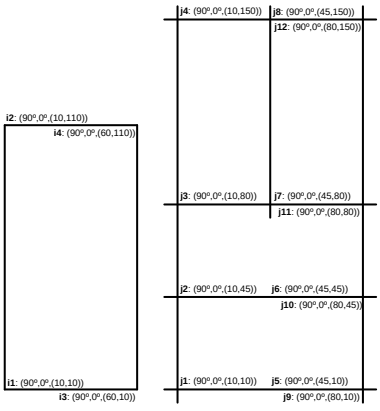
and a scaling with a 101/67 factor, both centered in point (-10,5). In this case both intersection pairs are symmetrical (each have both their intersection angles equal, of 50º), so, there is a second match with a translation from point (-60,65) to point (8,104), followed by a rotation with a 230º (*i.e.*, 180+50) angle, and a scaling with the same scale factor.

```
Algorithm structural-match(Input: subShape, targetShape)
Locals: tR1 = {}, tR2 = {}, tRstage1 = {}
For each is in intersections(subShape)
    For each it in intersections(targetShape)
        For each ips in intersection-pairs(straight-line1(is))
            For each ipt in intersection-pairs(straight-line1(it))
                tR1 = t-union(match-ip(ips, ipt), tR1)
            For each ipt in intersection-pairs(straight-line2(it))
                tR1 = t-union(match-ip(ips, ipt), tR1)
        For each ips in intersection-pairs(straight-line2(is))
            For each ipt in intersection-pairs(straight-line1(it))
                tR2 = t-union(match-ip(ips, ipt), tR2)
            For each ipt in intersection-pairs(straight-line2(it))
                tR2 = t-union(match-ip(ips, ipt), tR2)
        tRstage1 = t-intersection(tR1, tR2)
Return tRstage1

Algorithm sub-shape?(Input: subShape, targetShape, tRstage1)
Locals: tRstage2 = {}
For each t in tRstage1
    If t(subShape) ≤ targetShape Then tRstage2 = t-union(t, tRstage2)
Return tRstage2
```

Figure 6- The structural-match and sub-shape? algorithms (match-ip tests if there is any matches between two intersection pairs returning, in that case, a set of transformations for T; t-union and t-intersection return the union and the intersection of two sets of transformations; straight-line1 and straight-line2 return each of the straight-lines of a given intersection).

This, of course in only a small step, as the algorithm must try to match all the intersection pairs of the given shape with those of the target shape and see for which hints for the parameters for T consistency is maintained throughout all the process. One subtlety is worth to mention here. In order to find all possible matches, *i.e.*, all possible sub-shapes in the target shape, the concept of intersection pair has to be refined in a way as to consider, in the case of the target shape, intersection pairs with any pair of intersection points in each same straight line, but, in the case of the given shape, the shape to match, the concept must be restricted only to intersection pairs with adjacent intersection points in the same straight line. This is because any two points of intersection in the same straight line in the target shape can give a hint to a scale factor. In the case of the example in Figure 5, this would involve going on trying to match intersection pair in a) also with other possible additional intersection pairs in b) composed of intersection j1 and other intersections beyond and up the j5 intersection on the common straight line There are some other additional subtleties involved in the algorithm but, for the sake of simplicity and use of space, we will stick to the essentials and mention only this one.

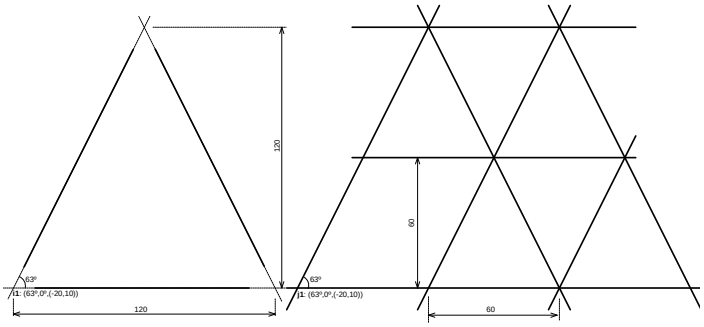


a) Shape to match b) target shape

Figure 9- First test example.

The algorithm structural-match of the first stage for sub-shape detection is the first shown in Figure 6. With the results of this algorithm (the set tRstage1) we then can apply sub-shape?, the second stage algorithm, the second shown in Figure 6. The containment operation ≤ is according to [10] [11].

We will now present some example test cases of application of the process of these two algorithms. The first test case involves rectangular figures and is illustrated in Figure 9. The results, in Figure 7-a)[14], show us ten possible matches, eight with the smaller four rectangles, with scale factor 7/10, and two with the bigger rectangle in the target shape, with scale factor 7/5[15].



a) Shape to match b) target shape

Figure 8- Second test example.

The second test case involves some oblique figures, with odd/infrequent angles (63º and 54º)[16], with a shape to match having no concrete intersections and with multiple intersection sharing the same intersection points. This is depicted in Figure

```
((SEQUENCE :SEQUENCE ((TRANSLATION :DX 20 :DY 70)
                      (SCALE :FACTOR 7/10 :X 80 :Y 80)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -15 :DY 70)
                      (SCALE :FACTOR 7/10 :X 45 :Y 80)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 20 :DY 70)
                      (ROTATION :ANGLE 90 :X 80 :Y 80)
                      (SCALE :FACTOR 7/10 :X 80 :Y 80)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 20 :DY 35)
                      (ROTATION :ANGLE 90 :X 80 :Y 45)
                      (SCALE :FACTOR 7/10 :X 80 :Y 45)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -15 :DY 140)
                      (ROTATION :ANGLE 180 :X 45 :Y 150)
                      (SCALE :FACTOR 7/10 :X 45 :Y 150)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -50 :DY 140)
                      (ROTATION :ANGLE 180 :X 10 :Y 150)
                      (SCALE :FACTOR 7/10 :X 10 :Y 150)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -50 :DY 35)
                      (ROTATION :ANGLE 270 :X 10 :Y 45)
                      (SCALE :FACTOR 7/10 :X 10 :Y 45)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -50 :DY 0)
                      (ROTATION :ANGLE 270 :X 10 :Y 10)
                      (SCALE :FACTOR 7/10 :X 10 :Y 10)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 20 :DY 0)
                      (SCALE :FACTOR 7/5 :X 80 :Y 10)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -50 :DY 140)
                      (ROTATION :ANGLE 180 :X 10 :Y 150)
                      (SCALE :FACTOR 7/5 :X 10 :Y 150))))
(10 items, computation time: 00:00:02).
```

```
((TRANSLATION :DX -31 :DY 0)
 (TRANSLATION :DX 31 :DY 0)
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 61 :DY -60)
                      (SCALE :FACTOR 1/2 :X 132 :Y 70)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 31 :DY 0)
                      (SCALE :FACTOR 1/2 :X 102 :Y 130)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 0 :DY -60)
                      (SCALE :FACTOR 1/2 :X 71 :Y 70)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX -31 :DY 0)
                      (SCALE :FACTOR 1/2 :X 41 :Y 130)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 31 :DY -120)
                      (ROTATION :ANGLE 180 :X 102 :Y 10)
                      (SCALE :FACTOR 1/2 :X 102 :Y 10)))
 (SEQUENCE :SEQUENCE ((TRANSLATION :DX 0 :DY -60)
                      (ROTATION :ANGLE 180 :X 71 :Y 70)
                      (SCALE :FACTOR 1/2 :X 71 :Y 70))))
(8 items, computation time: 00:00:02).
```

a) First test example b) Second test example

Figure 7- Results for test examples.

[14] For economy of space, sub-shapes associated to each transformation are not shown in the results, although also returned by the algorithm.

[15] Note that each of the rectangles in the target shape will have two matches, one for 0º and another for 180º rotation.

[16] Supposedly, an infrequent case in the literature, by the way, which seems to, not infrequently, resort to rectangular, 90º degree angles, shapes.

8. The results, in Figure 7-b), show us eight possible matches, six with the smaller triangles (four with 0º and two with 180º of rotation), with scale factor 1/2, and two with the two big triangles of the target shape, with 0º of rotation angle and scale factor 1.

VI. Conclusions and Future Work

After a brief state of the art and showing our goals and context, namely shape grammars and the GSG system, we have presented a two-stage algorithm to detect sub-shapes for use in that context. Some advantages of this algorithm were exposed in face of two other alternative algorithms presented in [16], namely a potential reduction in the number of useless steps, and in [17] namely the use of data structures directly related to shape elements which are expected to allow easier use of domain heuristics. Note that, although time/efficiency issues aren't manifest in our, actually toy, examples (2 seconds computation time for both, as seen in Figure 7), these domain heuristics would be very welcome, as the problem is computationally hard [18] in general. In terms of programming, all GSG components are built in Common Lisp/Common Lisp Object System, using the LispWorks® IDE system.

Future work will refine the stages of the algorithm, improve and finish the GSG system and, using the components and the experience gained with the development of GSG, our aim is to develop a multi-agent creative system in line with the ideas of the (primitive) GeoWin system of our past work.

Acknowledgment

This work was undertaken at ISTAR-Information Sciences and Technologies and Architecture Research Center from ISCTE-Instituto Universitário de Lisboa (University Institute of Lisbon), Portugal, and it was partially funded by the Portuguese Foundation for Science and Technology (Project "FCT UIDB/04466/2020").

References

[1] J. Reis, "Supporting Creativity with Emergent Shapes in Shape Grammars", *WAIM 2023 Workshop in the CISTI 2023 Conference*, Aveiro, Portugal, 2023.

[2] G. Stiny, "Introduction to Shape and Shape Grammars," *Environment and Planning B,* 7(3), 343-351, 1980.

[3] G. Stiny, Shape: Talking about Seeing and Doing, Cambridge, Massachusetts, USA: MIT Press, 2006.

[4] G. Stiny, W. J. Mitchell, "The Palladian Grammar," *Environment and Planning B,* 5, 5-18, 1978.

[5] G. Koning, J. Eisenberg, "The Language of the Prairie: Frank Lloyd Wright's Prairie Houses," *Environment and Planning B,* 8, 295-323, 1981.

[6] J. P. Duarte, "Towards the Mass Customization of Housing: The Grammar of Siza's Houses at Malagueira," *Environment and Planning B,* 32, 347-380, 2005.

[7] G. Stiny, "Kindergarten Grammars: Designing with Froebel's Building Gifts," *Environment and Planning B,* 7, 409-462, 1980.

[8] J. Heisserman, "Generative Geometric Design," *IEEE Computer Graphics and Applications,* 14, 37-45, 1994.

[9] M. Agarwal, J. Cagan, "A Blend of Different Tastes: The Language of Coffeemakers," *Environment and Planning B,* 25, 205-226, 1998.

[10] R. Krishnamurti, "The Arithmetic of Shapes," *Environment and Planning B,* 7, 463-484, 1980.

[11] R. Krishnamurti, "The Construction of Shapes," *Environment and Planning B,* 8, 5-40, 1981.

[12] R. Krishnamurti, "The Maximal Representation of a Shape," *Environment and Planning B,* pp. 19, 267-288, 1992.

[13] R. Krishnamurti, "Spatial Change: Continuity, Reversibility, and Emergent Shapes," *Environment and Planning B,* 24, 359-384, 1997.

[14] M. Tapia, "A Visual Implementation of a Shape Grammar System," *Environment and planning B,* 26, 59–73, 1999.

[15] S. C. Chase, "A model for User Interaction in Grammar-Based Design Systems", *Automation in Construction,* 11, 161–172, 2002.

[16] T. Trescak, M. Esteva, I. Rodriguez, "A shape grammar interpreter for rectilinear forms," *Computer-Aided Design,* vol. 44 (7), pp. 657-670, 2012.

[17] T. Grasl, A. Economou, "From Topologies to Shapes: Parametric Shape Grammars Implemented by Graphs," *Environment and Planning B, 40, 5,* pp. 905-922, 2013.

[18] T. Wortmann, R. Stouffs, "Algorithmic complexity of shape grammar implementation," *AIEDAM,* 32, 138-146, 2018.

[19] J. Reis, "Agents with Style − Multi-Agent Visual Composition with Shape Grammars," em *Proceedings of the Third Joint Workshop on Computational Creativity, Aug. 2006*, Riva del Garda, Italy, 2006.

[20] J. Reis, "GSG, A Tool for Knowledge-Based Visual Creativity," em *CISTI 2013 Proceedings, Vol. I, pp. 358-363.*, Lisboa, Portugal, 2013.

[21] J. Reis, "A Shell Tool for Visual Creativity Support," em *ISDOC 2013 Proceedings, pp. 56-63.*, Lisboa, Portugal, 2013.

[22] J. Reis, "Crossing Lines in GSG," em *ISDOC 2014, pp. 105-112.*, Lisboa, Portugal, 2014.

[23] J. Reis, "Shapes: Seeing and Doing with Shape Grammars," em *CISTI2022*, Madrid, Spain, 2022.

[24] J. Tching, A. Paio, J. Reis, "A Shape Grammar for Self-Built Housing," em *Proceedings of the SIGraDi 2012, pp. 486-490.*, Fortaleza, Brasil, 2012.

[25] J. Tching, J. Reis, A. Paio, "Shape Grammars for Creative Decisions in the Architectural Project", *CISTI 2013, Vol. I, pp. 389-394.*, Lisboa, Portugal, 2013.

[26] J. Tching, J. Reis, A. Paio, "A Cognitive Walkthrough towards an Interface Model for Shape Grammar Implementations," *Computer Science and Information Technology*, vol. 4(3), pp. 92-119, 2016.

[27] J. Tching, J. Reis, A. Paio, "IM-sgi – an Interface Model for Shape Grammar Implementations," *AIEDAM,* 33, Issue 1, February 2019, 24-39, 2019 .