Supporting Creativity

with Emergent Shapes in Shape Grammars

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

Abstract — **This paper describes a computational infrastructure used to support creative design in detecting emergent shapes in the specific context of shape grammar implementation. Shape grammars have been used to represent the knowledge behind the creative work of architects, designers and artists. This kind of grammars are inherently visual and they allow the implementation of computational mechanisms to either synthesize or analyze designs of visual languages, including the detection of emergent sub-shapes languages. They have obvious applications to design, including for marketing. The infrastructure presented, together with the algorithm to which it gives support, the latter proposed in another, twin, paper, is a core component of a system, described in our past work, that allows users to build their own shape grammars and experiment with and use them.**

Keywords - Shape Grammars; Artificial Intelligence in Design; Creativity.

I. INTRODUCTION

Many human creative activities, visual or not, can involve discovering emergent patterns and taking advantage of, and using them, from the most noble scientific research and the creative work of artists to the most commonplace day-to-day activities, including entertainment. Emergency is important in creativity, in discovering and creating new (and useful) ideas, designs, realizations, artifacts. Take, for instance, an artist's creativity process. It's not unusual to stop in the middle of his/her creative work to look to, and assess, the work done so far, and it happens to discover some emergent detail, or shape, that didn't seem to be (and that, in fact, it wasn't explicitly included) there before. And that detail comes to be inspiration for the next step in the creative work.

A similar phenomenon happens in the context of shape grammars. To apply shape grammar rules to a composition, or design, there must be some mechanism, either human or computational, that decides if a rule is applicable or not. For that purpose, it must detect if the shape in the left side of the rule, its antecedent, is contained in the composition. That can happen, for the same rule, in zero or more ways. The human brain and eye seem efficient in solving this emergent shape detection feature, although prone to failures and mistakes too. Nevertheless, the problem seems to be defying (and so, it is interesting) from a computational perspective.

Shape grammars and its formalism were introduced by George Stiny and James Gips in the 1970s. They can be used to synthesize, as well as, to analyze, designs, or styles, of design languages. Together with symbolic/text phrase grammars shape grammars 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 there. Firstly, shape grammars are inherently visual; secondly, they can accommodate aspects of emergency, *i.e.*, the possibility of generating shapes not explicitly introduced by the application of the rules, which, of course, will have to be detected further in the rule application process. Maybe the differences are not restricted to these two features, but these two are very important in the field of arts, especially in design.

This paper approaches the problem of discovering emergent sub-shapes in given shapes, specifically in what respects to the computational structures necessary to support a particular detection algorithm. Extending the path other researchers followed, we have proposed an approach through an algorithm, described in a twin paper [1], the operation of which needs specific data structures, described here, in the present paper. This approach is applicable in the context of shape grammars with shapes composed of some basic geometric elements.

In the following, we summarize: a brief state of the art and what shape grammars are (section II); then, an introduction to the sub-shape detection problem, illustrating its importance for rule application and including the short relevant history of the approaches proposed (section III), follows; our previous work in the area and our goals are described in (section IV). Then, we expose the structures necessary to give support to the algorithm mentioned above and compare our approach with alternatives from other researchers (section V). Finally, we draw conclusions and show intended future work (section VI).

II. SHAPE GRAMMARS? WHAT IS THAT?

A seminal paper by Stiny and Gips [2] introduced shape grammars and its formalism. The related research area can be described as being about representing and applying knowledge of languages of design, basically through the use of concepts from formal grammars and rule-based/production systems [3] [4]. In very brief words, the research has been focused in conceptual and theoretical aspects, as the ones exposed in [4], 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] [10]. More research include the development of algorithms and data structures for shape manipulation and rule matching and application processes, which are very interesting for us in the present paper, as in [11] [12] [13] [14] [15] [16] [17] [18] [19], where some of this papers focus also on appropriate interfaces and generic and reusable shape grammar interpreters, including for didactical purposes.

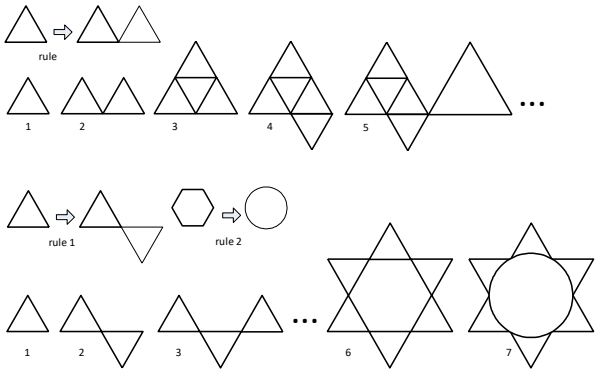


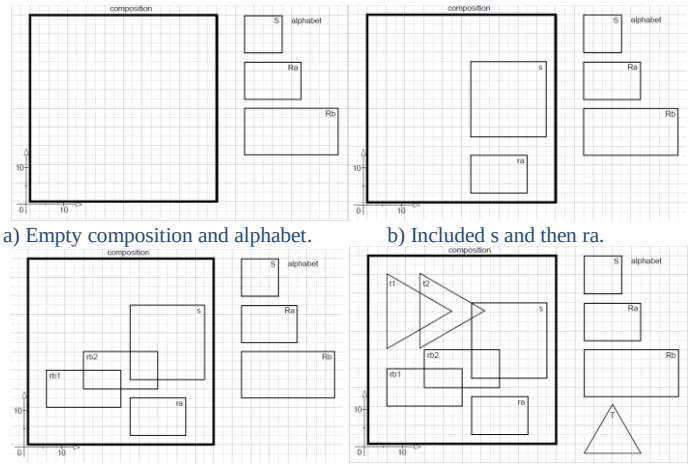
Figure 1- Two grammars with example derivations, exhibiting shape emergence. In the first, the (only) rule is applied to an emergent triangle in step 4 and then in step 5, too. In the second grammar, emergence also occurs, resulting in the shape shown in step 7, after the application of rule 2.

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 in the 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 the shape in a composition, C, if there is a similarity geometric transformation (a translation, a rotation, an uniform scaling or a combination of these) Tr, which, when applied to shape A makes A equal to a part of C, *i.e.*, a geometric transformation Tr such that Tr(A) ≤ C, where ≤ denotes a sub-shape relation[1]. Application of the rule results in a new shape, C', that is computed subtracting from C the result of applying the transformation Tr to A, and then adding to C the result of applying Tr to B, *i.e.*, the resultant design will be C' = (C – Tr(A)) + Tr(B), where + and – denote the shape sum and shape difference (or subtraction) operations, further described. It's easily seen that ≤ is important for the sub-shape problem.

In Figure 1 we show two shape grammars and derivation examples, in both cases exhibiting emergent shape detection. As can be seen, the computational mechanism that decides if a rule is applicable and, in that case, to perform its application, if it is chosen to be applied, must detect if Tr(A) ≤ C and to determine the set of possible transformations Tr, too. The mechanism must deal with cases of Tr(A) being an emergent sub-shape, which seems a feature difficult to be implemented.

III. An Interesting Problem and the Past Research

To illustrate the problem of emergent sub-shape detection, we now turn to a toy example of composing a simple design with an alphabet of simple predefined shape elements.



a) Empty composition and alphabet. b) Included s and then ra.

c) Included rb1 and rb2. d) Included t1 and t2.

Figure 2- Example of composition stages a), b), c) and d). Before stage d) the alphabet was expanded with T.

In Figure 2-a) we show a starting point for a composition, together with an alphabet of predefined shapes containing a square, S, and two different rectangles, Ra and Rb. In Figure 2-b) we included in the composition s, an instance of S, and then ra, an instance of Ra, respectively scaled by a 2 and a 1 scale factor[2]. In Figure 2-c) we further included rb1 and rb2, two instances of Rb scaled by a 4/5 scale factor[2]. In Figure 2-d) we further expanded the alphabet with an equilateral triangle, Tr, and then included in the composition t1 and t2, two instances of Tr, both rotated by 30º and scaled by a 4/3 scale factor (apart from appropriate translations). In stage c) we see that an emergent shape appeared resulting from the interference of rb1 and rb2 and still another from rb2 and s (a rectangle and a square, similar to Rb and S, respectively). Also, in stage d), we also find two emergent triangles, one resulting from the interference of t1 and t2 and another from t2 and s.

When looking for similarity transformations Tr such that Tr(A) ≤ C to verify if a rule A→B is applicable and in what ways, we could easily devise a mechanism that looks only for predefined shapes like the ones in our alphabet. And it would be easy to determine the appropriate scale factors (through distances between shape elements as well as their proportions), angles of rotation and amounts of translation for the Tr's. But, as we keep the demand of detecting also emergent sub-shapes, the mechanism fails as soon as the shapes being included in the composition begin to intersect. Also, the complexity of the

[1] 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.

[2] Apart, of course, a rotation of 0º and some appropriate translation.

problem increases when more complex shapes must be looked for, see examples in Figure 3.

A possible way to surpass this difficulty is considering the

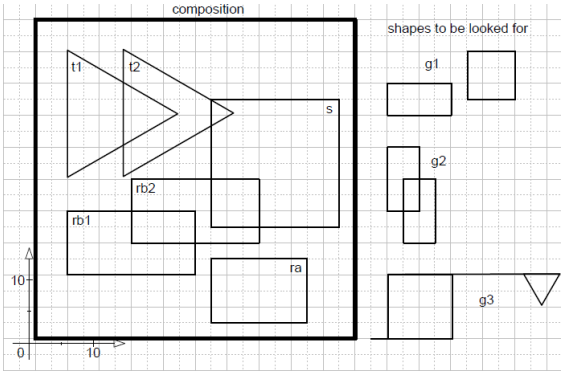


Figure 3- Example of more complex shapes to be looked for: g1, g2 and g3 are all sub-shapes of the shape in the composition.

predefined shapes as composed by line segments limited by its extreme points and, whenever intersections happen, restructure the presently stored composition representation in a way to consider every and each original line segment intersected as composed by smaller segments limited by original extreme and/or intersection points. But, as the number of lines increase in the composition, the computational complexity of the process can grow exponentially. See [18] for a review of this and other possible alternatives.

In fact, no object-oriented method[3], nor any classical CAD tool [4], just by themselves, can help to computationally implement the generic sub-shape detection mechanism we are talking about. As found by Stiny and other researchers following a common research path, see [3] [4] [11] [12] [13] [14], 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 and, at the same time, to mitigate the complexity issue.

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 $j \geq i$, see [4]. 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 line. Figure 4 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[5].

In the compact summary research following, all papers address the problem we are interested in, of the recognition of emergent shapes. Papers [11] and [12] paved a research path to the appropriate computational representation of shapes in 2 dimensions and how shape operations should work on them. In [13], more precise definitions suggest computational representations for maximal shapes, including for more than 2 dimensions. In [14] the "formula" C' = (C – Tr(A)) + Tr(B) is analyzed, as well as the recognition of emergent shapes and the cases that occur in the determination of Tr. In [15], the first implementation of a shape grammar system able to detect emergent shapes for rectangular shapes, is proposed. In [17] an algorithm is proposed for detecting emergent shapes which

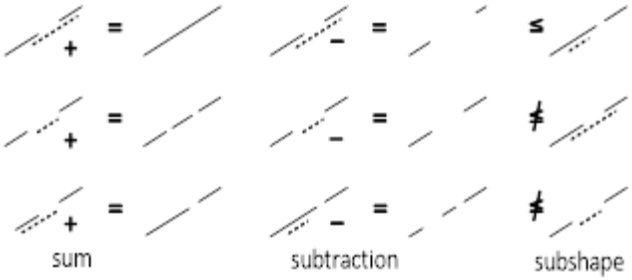


Figure 4- Some examples of maximal shape sum and subtraction and the sub-shape relation in a U_{12} algebra. The dashed line is a component line in a shape to sum to, subtract from, or test for sub-shape relation with, other lines which are components of another, target, shape.

considers each, and every, intersection (concrete or virtual/projected) between pairs of lines in the shape. In [18] 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 problem of sub-shape detection is a computationally hard one and is equivalent in hardness to the subgraph isomorphism problem, see [19], where the computational complexity of algorithms used in different kinds of shape grammars, including in sub-shape detection, is analyzed.

IV. Past Work and the GSG System

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 [20]. This participation occurs with different coordination strategies,

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

[4] 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.

[5] 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 in the target shape leaves the shape with more lines than before; also, in the first case for + a line is summed leaving the shape with less lines than before.

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.

In the path to improve in the direction mentioned above, we then embarked on an approach to build a prototype of a computational system centered around a "Generic Shape Grammar" interpreter, the GSG project, see the main initial work in [21] [22] [23]. 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 experimenting 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 main components of the GSG system are a two-part interface and two core sub-systems, a rule-based component centered around rules, sub-shape detection and rule application, and a third component centered on 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 [24]. A third kind of interface, the textual/file interface is also available. As described in [24], 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 [25] [26] and work about the usability of interfaces of implemented shape grammar systems of different authors is shown in [27] [28]. In the latter, we have devised a set of requirements (see a summary of these in [24]) that can be used to either 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).

V. Supporting Structures for Finding Sub-Shapes

We now turn to the computational structures which are the subject of the present paper and that give the necessary support to the sub-shape detection algorithm we have implemented. See [1], a twin paper to this, where we point that the task of the algorithm is to detect if a given shape, typically one in the left side of a rule, matches, *i.e.*, is contained, after any similarity transformation to be determined, in another, target, shape, typically a shape of a composition, or design. 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 Tr associated to each matching case.

Besides the adoption of an U_{12} algebra of maximal shapes restricted to lines (points are excluded[6]), and although internal representations we adopted follow closely the proposals in [11] and [12], additional option decisions were made, both in assumptions for the algorithm and in the data structures used to support the algorithm, that are described in the following. First, the limitations of the algorithm, as detailed in [1]. In sub-shape detection and rule application, the similarity transformation Tr is the only kind of transformation applied to shapes[7]. Additionally, component lines must bear, in the shape they belong to, at least two intersections, concrete or virtual/projected. In the following, we describe the structures in our terminology and then illustrate their role in the sub-shape detection algorithm in GSG.

In GSG, a shape is a collection of maximal lines[8]. 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 the slope and coordinates of the limit points. Because similarity is what we are dealing with, intersection angles and length proportions are very important for the algorithm to operate, as their magnitudes are maintained through similarity transformations. So, at the time a shape is being constructed, besides the data structures for representation of its maximal lines, certain additional internal/backstage data structures are incrementally created which are the appropriate representation infrastructure to support the algorithm, not only in sub-shape detection, but also further, in rule application. These structures represent the straight-lines of support of (*i.e.*, containing) the lines of the shape and straight-line intersections, which are pairs of straight-lines with an intersection point. There will be more than one intersection if more than two straight lines happen to intersect at the same point. In a shape, as well as lines, these two additional structures are maintained in specific sorted collections for fast access. Additionally, a volatile kind of structure we name as intersection pairs is also needed and used. So, summing up, the important structures for shape representation and operation of the sub-shape detection algorithm are:

- Lines – they are maximal lines, defined by its limit points and can be either created or destroyed when new lines are summed to, or subtracted from, a shape. Other relevant attributes are the slope (an angle in the interval [-89º, 90º]) and an intersect parameter (either x-intersect for vertical lines, or y-intersect for others).
- Straight-lines – they represent infinite lines, with slope and intersect, they support, or "contain", lines with equal slope and intersect and are created as soon as a line appears with non-preexistent slope and intersect and destroyed when operations with lines result in empty (*i.e.*, with no lines) straight-lines.
- Intersections – composed of a pair of non-parallel straight-lines and its intersection point, they are created when the creation of a new straight-line generates new intersections and destroyed when an

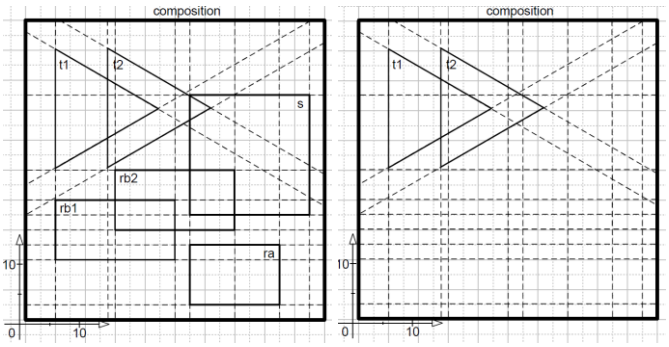
[6] Allowing points would, in fact, render the problem tackled simpler, by providing possible anchor locations for easier Tr parameter determination.

[7] This excludes the so-called parametric shape grammars (see [3]).

[8] 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.

intersecting straight-line is destroyed. They have, as attributes, the intersection angle (an angle in the interval [1º, 179º], the slope of the lower slope straight-line and the intersection point.

– Intersection pairs – pairs of intersections (i1, i2) sharing a common main straight-line. These are created on the fly and just temporarily stored as needed by the algorithm. While, for a shape to be looked for they are made strictly from consecutive intersections, for a target shape they are made from any pair of intersections, on the main straight-line. Relevant attributes are the two intersection angles of the pair, the slope of the main straight-line and the length of the pair, *i.e.*, the Euclidean distance between the intersection points.



a) backstage structures b) focus on t1 and t2

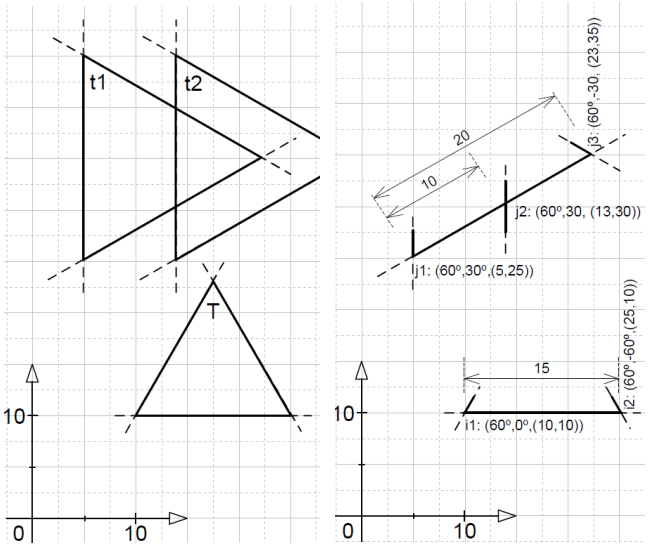
Figure 5- Composition example final state. All shape elements are maximal lines, represented with a full line style. Dashed line is used for straight-lines.

The implemented maximal algebra operations +, – and ≤ in the computational geometry GSG component make them possible to operate between lines and shapes and also between straight-lines, intersections and between mixed type operands. Also implemented were operations returning the result of applying any similarity transformations to any of the items described above, including intersection pairs. Additionally, operations to determine the possible similarity transformations necessary to match (two) intersections or to match (two) intersection pairs were also implemented based on the following. Two intersections match if one has an angle of intersection, or its supplementary, that is equal to the angle of intersection, or its supplementary, of the other, apart some angle of rotation. This one can be used as a hint for the angle of rotation of the similarity transformation. A pair of intersections will match with another pair if each angle in the first is equal to another, different, angle in the other pair, apart from some angle of rotation of one of the pairs. In this case, the ratio of the Euclidean distances between the intersections of each pair can be used as a hint for the scale factor of the transformation.

We now turn to a previous example to illustrate the role of these structures by considering the final composition state in Figure 2-d), seen also in Figure 3 and assuming the backstage representation scenario of GSG. This means that all shape elements are maximal lines and all the additional representational, backstage, permanent structures described above are established. These would look as represented in Figure 5-a). For further illustration, we now focus only on the triangles t1 and t2 (included in the last stage) as shown in Figure 5-b), as part of our target shape. Assume further that we have the triangle T of the alphabet as the shape to look for in a location as shown in Figure 6-a).

In a first stage, the sub-shape detection algorithm tries to match each consecutive intersection pair in the shape looked for with any intersection pair in the target shape obtaining, in the case of success, a non-empty set of similarity transformations. In case of success, on a second stage, using each of the transformations found it will test for containment of each line in the shape looked for in the target shape. Then, it will return the set of transformations associated to the cases of positive containment, if there are any.

To additionally illustrate in detail how the data structures used help to make the match, let us focus on a detail of part of the process of the algorithm, specifically on trying to match the base of triangle T with the 30º slope base of triangle t1. Matching will be tried between (i1, i2) and (j1, j2) failing (the two angles of each of the intersections don't match with the same rotation angle), between (i1, i2) and (j1, j3) with success for one of the (two) possible rotation angles, 30º, and a 4/3 scale factor (from the ratio of 20/15) and finally, between (i1, i2) and (j2, j3) with success for one of the (two) possible rotation angles, 30º, and a 2/3 scale factor (from the ratio of 10/15). This will, of course, be a part of the whole process of the algorithm, as it will find, in the first stage, all the similarity



a) looking for T in t1 b) detail of part of the process

Figure 6- Matching of intersection pair (i1, i2) of triangle T will be tried with intersection pairs (j1, j2), (j1, j3) and (j2, j3) of triangle t1.

transforms that allow to consistently transform all intersections of the shape looked for in different intersections present in the target shape. The second stage boils down to testing for line containment and discard the negative cases, as we said before.

Using and matching intersection pairs turns out to be a potentially more efficient way in terms of space/memory, at least for more complex shapes (*e.g.*, with more lines per straight-line) as it considers each intersection only once, instead of doing it repeatedly at the level of each intersecting line pair as done in [17]. Moreover, it can possibly be more efficient in terms of time/speed, as we use structures related directly with shape elements (lines, in the case) and not any kind of additional intermediate data structure, like graphs, as used in [18] which (besides some advantage is terms of

abstraction and flexibility) can bring the disadvantage of precluding an easy use of domain heuristics in the search of possible sub-shapes. Note that these domain heuristics would be very welcome, as the problem is computationally hard [19]. Finally, the method used directly produces hints for rotation angles and scale factors for the searched similarity transformations.

VI. Conclusions and Future Work

After a brief state of the art and showing our goals and the context, namely shape grammars and the GSG system, we have presented an infrastructure to support an algorithm to detect sub-shapes for use in that context. Some advantages of this algorithm were exposed in face of other alternative algorithms. 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 infrastructure and the algorithm, improve and finish the GSG system and, using the components and the experience gained with the development of GSG, our aim is also 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, "What's in a Shape, An Algorithm for Finding Shapes in Shapes," *WAIM 2023 Workshop in the CISTI 2023 conference*, Aveiro, Portugal, 2023.

[2] G. Stiny, J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," *Information Processing,* 71, 1460-1465, 1972.

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

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

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

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

[7] 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.

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

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

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

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

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

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

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

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

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

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

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

[19] T. Wortmann, R. Stouffs, "Algorithmic complexity of shape grammar implementation," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 32, 138-146, 2018.

[20] 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.

[21] J. Reis, "GSG, A Tool for Knowledge-Based Visual Creativity," em *CISTI 2013, Proceedings of the 8th CISTI, Vol. I, pp. 358-363.*, Lisboa, Portugal, 2013.

[22] J. Reis, "A Shell Tool for Visual Creativity Support," em *ISDOC 2013, Proceedings of the International Conference on Information Systems and Design of Communication, pp. 56-63.*, Lisboa, Portugal, 2013.

[23] J. Reis, "Crossing Lines in GSG," em *ISDOC 2014, Proceedings of the International Conference on Information Systems and Design of Communication, pp. 105-112.*, Lisboa, Portugal, 2014.

[24] J. Reis, "Shapes: Seeing and Doing with Shape Grammars," em *CISTI2022, 17th Iberian Conference on Information Systems and Technologies*, Madrid, Spain, 2022.

[25] 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.

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

[27] 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.

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