Grasl and Economou, 2013. The definitive, peer-reviewed and edited version of this article is published in Environment and Planning B: Planning and Design, 40, 5, 905-922, 2013, doi:10.1068/b38156

From topologies to shapes: Parametric shape grammars implemented by graphs

Thomas Grasl and Athanassios Economou

Abstract
A method for implementing parametric shape grammars is presented. Subgraph detection is used to find subshapes. Parametric shapes are described by restricting topologies.

Keywords: Graph grammar, parametric shape grammar.

1 Introduction

Pattern matching is something that biological minds excel at. This is true for patterns in general, but nowhere more so than in image recognition. A beautiful example was the discovery by the bohemian engineer Ržiha that the mason marks found in gothic cathedrals and elsewhere can be traced back to a set of common carrier grids (Ržiha, 1883). Whether all of Ržiha's claims hold true or not, the discovery of the fourteen carrier grids was a remarkable feat. Each mason was assigned a mark that is embedded in one of these grids. Four of these mason marks and their carrier grids are given in Figure 1. This ability, to recognize subshapes in a clutter of lines, never mind what their proportion, size and orientation is the foundation of shape grammars. It is also the reason why it is challenging to implement them on a digital computer.

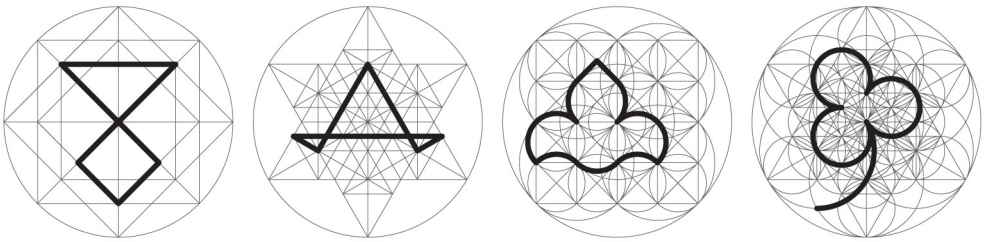

Figure 1. Four of the mason marks described by Franz von Ržiha

Shape grammars provide a powerful generative approach for the description, interpretation and evaluation of existing designs as well as new designs (Stiny and Gips 1972; Stiny 2006). A nice account of various applications of shape grammars for the study of existing and new languages of design can be found in Knight (2003).

While the shape grammar formalism has produced a long and excellent series of formal studies the corresponding efforts for automation of the tasks that shape grammars accomplish have not met the same success. Two

computational challenges of implementing shape grammars are detecting emergent shapes and enabling parametric rules. A general solution to both of these is presented here.

Graphs are a popular choice for representing the structure of shape. Heisserman (1991; 1994) uses boundary graphs to guarantee well-formed solids during the execution of his boundary solid grammars. Kelles et al. (2010) present the overcomplete graph as a tool to search for embedded shapes. Furthermore, Grasl (2012) presented Grappa, an implementation of the Palladian Grammar (Stiny and Mitchell, 1978) based on graph grammars.

Typically graphs are seen as multidimensional data structures that, with an appropriate embedding, can formally structure and visually resemble the shapes they represent. Still, the encoding of shapes in terms of these alternative graph representations is not straightforward. Various approaches to representing shapes as graphs are described here and one of them is selected as the most useful for a shape grammar interpreter. The resulting formalism uses two parallel representations, a graph theoretic which is used for the subshape detection, and a visual one that illustrates the shape and which can be queried for intersections. The implementation presented here also maps the shape rules into graph rules. Once a rule is selected matching subgraphs are mapped back to shapes for further user interaction, such as the picking of one of the shapes to apply the rule to. The shape grammar library has been implemented in *C#* and it is code-named GRAPE (Figure 2) to affectionately recall the parallel representations of GRAphs and shAPEs.

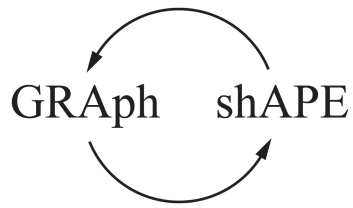

Figure 2. Graphs are informed by shapes are informed by graphs.

1.1 Existing implementations

Gips (1999) gives a good overview of different kinds of shape grammar applications and discusses current issues. Chau et al. (2004) present an extensive list of previous implementations before turning to their own work. More recent projects include Duarte et al. (2006), Ertelt and Shea (2009), Hoisl and Shea (2011), Jowers and Earl (2011), Jowers et al. (2010) and Tresak et al. (2009).

A great number of the above implementations are based on the algorithms devised by Krishnamurti (1981). Krishnamurti uses maximal lines and a search based on three points to find subshapes under similarity transformations. Krishnamurti and Earl (1992) extended this originally two-dimensional algorithm to three dimensions. Jowers and Earl (2010) have taken on the task of extending the algorithms to comprise parametric curves. Additionally Jowers et al. (2010) are experimenting with image recognition

routines for the subshape-matching problem. Similarly Keles et al. (2012) use image recognition based on genetic algorithms. However, while these approaches enable emergent subshapes to be recognized, parametric grammars cannot be implemented with them.

So interestingly, most implementations support emergent shapes but not parametric rules although most grammars described in the literature, such as the ones for Palladian villas, Mughul gardens, Ice Ray lattices or Hepplewhite chairs to name a few, require parametric rules rather than emergent shapes. Notable exceptions are Flemming (1987) and the work by McCormack and Cagan (2002; 2006). But while Flemming's implementation can handle parametric rules, it does not recognize emergent shapes. McCormack and Cagan's approach can recognize only anticipated and predefined parametric features for shapes in the plane. Shapes are decomposed into fragments and each is searched for individually.

Hence, one thing that distinguishes GRAPE from the above implementations is that emergent shapes and general parametric rules are both supported. This does come at a cost. As Yue et al. (2009) demonstrate, by reducing the maximum clique problem to it, parametric subshape recognition is NP-hard. That practical solutions can nevertheless be devised with the presented methodology is shown in section 4.3.

2 Graph theoretic representation of shape

Emergence in the algebras of design U_{1j}, shapes consisting of lines in two- or three-dimensional space, is strongly connected to intersections; whenever two shapes overlap new intersections are produced that offer the opportunity of emergence.

There are many examples that can do to show the underlying representation, a useful one is depicted in Figure 3. An early and detailed account of the emergence of this specific shape and its role in shape grammar scholarship is given elsewhere (Stiny, 1991).

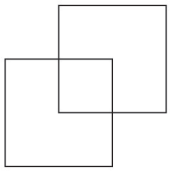

Figure 3. A shape.

The questions that arise looking at this shape quickly supersede the apparent simplicity of the argument. The careful observer will immediately concede to the ambiguity that is hidden behind this and many other, in fact all, pictorial settings. The structure of the shape, that is, the parts of the shape and their relations, denies any final and resolute configuration. This is in direct opposition to how the majority of symbols function: a word, a number, or a musical motif notated in classical western notation, are all unambiguously notated, communicated, and used. And even if one commits

to a singular representation, say, the shape as an intersection of two squares [Figure 4 (a)] then the emergent smaller inner square [Figure 4(b)] is unaccounted for.

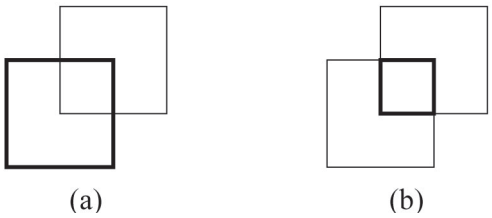

Figure 4. The shape seen as two overlapping squares (a) and the emergent third square (b).

These issues can best be resolved by representing shapes by their maximal lines (Stiny, 1980a). The maximal lines of a shape are the set of lines created by combining all collinear line segments that touch or overlap. Modeling shapes with maximal lines offers a unique representation for each shape and facilitates computations with shapes without resorting to combinations of atomic parts to account for the shapes involved in the computation. Here a graph theoretic structure is proposed to uniformly represent all maximal lines and their intersections.

Since the idea is to implement shape grammars using graph grammars as the computational engine, an unambiguous mapping between shapes and graphs must be found. Once a shape is translated into a graph, the graph is manipulated by a graph grammar and the result is mapped back into a shape. As a minimum the model should enable implementations of parametric shape grammars in U_{13}. Higher dimensional elements (U_{23}, U_{33}), as well as the algebras of labels (V_{ij}) and weights (W_{ij}) are possible extensions. Subshape recognition must be supported as well as the possibility to pick up shapes that are somehow implied, but not fully pictured, as described by Stiny (2006).

The following section shows various approaches to model the structure of the shape in Figure 3 in a graph theoretic manner; it provides a brief account of each and discusses the search patterns necessary to find the subgraphs of all three squares in the figure. Generalization to all others shapes in U_{13} is straightforward.

2.1 Plan graph

The most straightforward approach is to map points to nodes and lines to edges (Figure 5). In this mapping the edge of the graph essentially represents a line segment of the shape. Using a suitable embedding the graph may look much the same as the shape it represents. In architecture this kind of graph is perhaps best known as dual of the adjacency graph, it is sometimes called plan graph (Steadman, 1976).

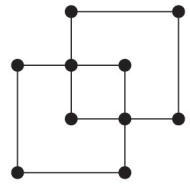

Figure 5. Plan graph: Mapping points to nodes and lines to edges.

The biggest drawback of this approach is its inability to model maximal lines. If a line is divided into several segments by crossing lines, it becomes difficult to query the graph for lines consisting of two or more segments within a rule since they differ topologically. This results in difficulties finding emergent shapes.

There is no single search pattern that can return all three squares embedded in the shape. Intuitively one might try searching for the pattern in Figure 6(b), but while the eye might find three matches in the above plan graph, a computer will find only one. Interestingly only the emergent square of Figure 4(b) will be returned. In order to find the two bigger squares the search pattern shown in Figure 6(a) is required. And while these patterns will return the targeted quadrilaterals in our example, they would also return other topologies, such as triangles, pentagons or hexagons under other circumstances.

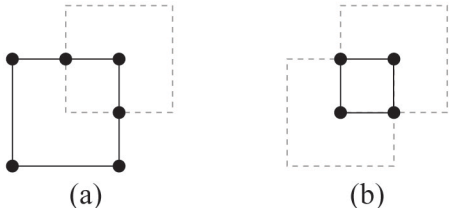

Figure 6. The two search patterns required to return all three squares.

2.2 Plan hypergraph

An approach suited better for the representation of maximal lines would be one using hyperedges (Figure 7). One hyperedge can connect more than two nodes, making it easy to simply connect all points along a line by one edge. Graph grammars have dealt with them extensively (Drewes et al. 1997) and it would be an approach true to the maximal line concept.

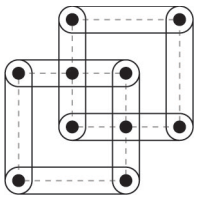

Figure 7. Plan hypergraph: Mapping points to nodes and maximal lines to hyperedges.

Now the same pattern will return all three squares, the respective matches are shown in Figure 8(a) and (b). Labels could be added using dedicated node and edge types. It is definitely a model that could be used, but due to

the limited availability of hypergraph grammar packages other possibilities will be explored first.

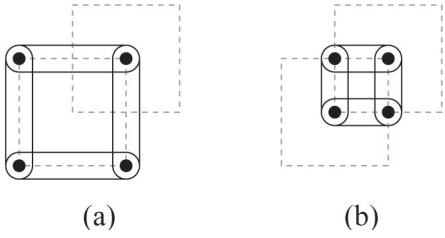

Figure 8. Pattern used to find quadrilateral topologies in plan hypergraphs.

2.3 Edge graph

Another approach could invert the mapping of elements and represent each maximal line of the shape as a node in the graph and each intersection in the shape as an edge (Figure 9). This mapping would seem to have all the advantages of the hyperedge approach in Figure 7, without the potential drawback of limited availability.

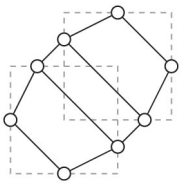

Figure 9. Maximal line edge graph: Mapping maximal lines to nodes and intersections to edges.

However, problems arise as soon as more than two maximal lines intersect in one point, hence it can only be used on a very limited set of grammars, such as orthogonal grammars. Finding triangles in a triangular grid is impossible. As above, the search pattern in Figure 10(a) and (b) will find all three squares.

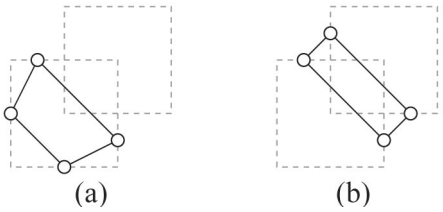

Figure 10. Pattern used to find quadrilateral topologies in maximal line edge graphs.

2.4 Edge hypergraph

Again the only possibility to satisfy all needs is the introduction of hyperedges (Figure 11) with all the reservations mentioned above.

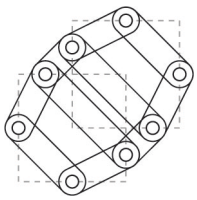

Figure 11. Edge hypergraph: Mapping maximal lines to nodes and intersections to hyperedges.

This is another possible model to be used for shape grammar implementations. Again labels could be added in a similar fashion as described above.

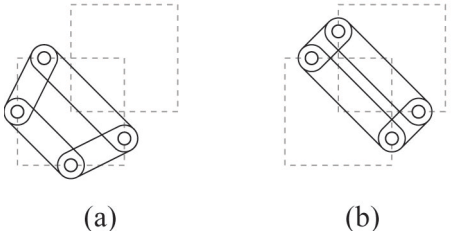

Figure 12. Pattern used to find quadrilateral topologies in edge hypergraphs.

2.5 Part-relation graph

Hypergraphs seem to be ideally suited to model maximal lines and enable subgraph detection. The question thus becomes, which characteristics of hypergraphs enable this behavior and can these be transferred to normal graphs? It is the ability to connect an edge to more than two nodes, in the current context this corresponds to connecting a maximal line to more than two points. Normal edges are restricted to binary relations, if such an edge represents a line it can only be connected to two points. Thus a different approach is proposed, we will represent geometric elements as nodes and relations between these elements as edges. For our example two different kinds of nodes need to be distinguished, point nodes (black) and line nodes (white) (Figure 13). Heisserman (1994) uses a similar graph-based boundary representation for boundary solid grammars. The graph here is called part-relation graph because it captures explicitly the topology of the shape in terms of its basic elements and their relations to one another.

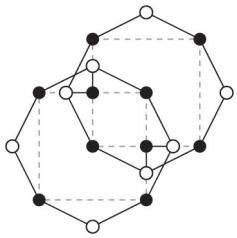

Figure 13. Boundary graph: Maximal lines and points to nodes and relations to edges.

In this case there is no drawback. One search pattern is enough to capture all the squares in the shape [Figure 14(a) and (b)].

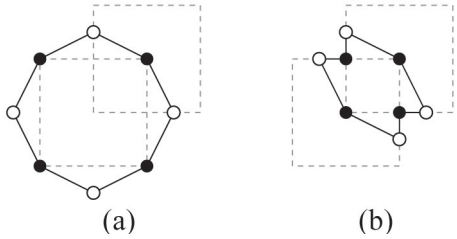

Figure 14. Patterns needed to return the three squares in the boundary graph.

Here it must be pointed out, that so far we are dealing with topologies only. So while the pattern in Figure 14 will indeed return the squares shown in Figure 4, it will also return parallelograms, rhombs and all other quadrilaterals under other circumstances. Restricting the topologies to specific geometries is something that will be dealt with in section 3.

2.6 Extensions

This approach can in fact be extended to any number of nodes that can be associated with a shape. Mapping several classes to nodes increases the total number of elements needed to represent a shape, but it adds flexibility because additional classes can be added. Depending on the grammar at hand classes of interest could be labels, faces, solids, colors, carrier lines and more. For example, the two squares and the emergent inner square in the center can be unambiguously described in terms of their vertices, edges, faces and their relations with a slightly extended part-relation model shown in Figure 15.

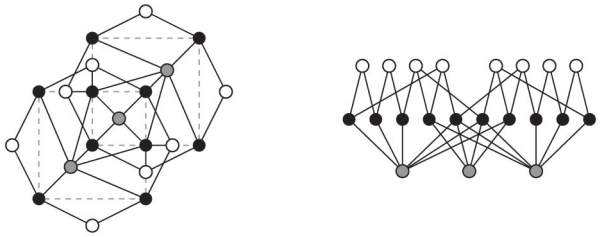

Figure 15. Graph representation of the shape in Figure 3 using three layers of objects: points (black), lines (white) and faces (grey).

Furthermore the approach can be extended in a straightforward way to account for projected intersections too. Projected intersections (Figure 16) are treated here in the same way as real intersections: point nodes connected to line nodes. In order to be able to distinguish between them though several types of edges are introduced: exterior, interior and boundary edge.

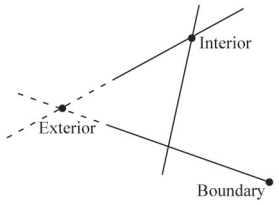

Figure 16. Examples for the three types of point-line relations.

Finally the extension of the approach to model shapes in algebras of labels (V_{ij}) is straightforward too. The model can be extended here to implement labels as new kind of nodes attached to shape nodes such as point or line nodes. And it is certainly possible to implement algebras such as the direct product $U_{13} \times V_{03}$. The graph model that can capture these calculations is shown in Figure 17. In addition to geometry nodes the model allows nodes for both shape and state labels [Figure 17(a)]. The model's edges are all undirected. They are divided into part edges, for various relations between

geometry nodes, and label edges, used to attach labels to objects [Figure 17(b)]. Defining hierarchies of nodes and edges is useful for creating the search patterns. For example, searching for a part edge will return boundary edges, interior edges or exterior edges.

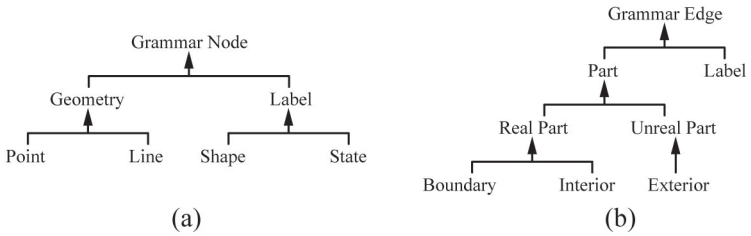

Figure 17. (a) The node hierarchy and (b) the edge hierarchy of the graph model.

3 Parametric grammar engine

The graph representation as described above is the foundation for the recognition of emergent shapes. Now we will turn to the reason the method enables parametric shape grammars. This ability is based on isomorphic subgraph recognition (Read and Corneil, 1977). The work here assumes some familiarity with group theory, permutations and combinatorics. A gentle introduction to some basic constructs and their usage in design is given in March and Steadman (1974), Economou (1999, 2006) and Economou and Grasl (2009).

As an example we will discuss variations of a rule to find quadrilaterals. Therefore we will first introduce the graph *Q* [Figure 18(a)], which represents all quadrilaterals. One must remember that the layout of the graph in the plane has nothing to do with the shape it represents.

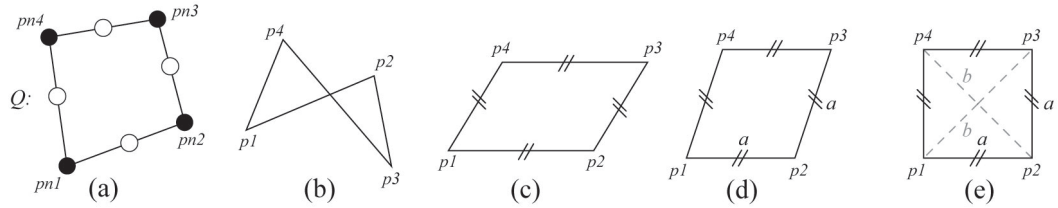

Figure 18. The graph *Q* (a) and some of the quadrilaterals it can represent (b-e).

To begin with we will examine which matches are returned if rule 1, the identity rule shown in Figure 19, is applied to *Q*. The graph is searched for four point nodes but nothing is changed.

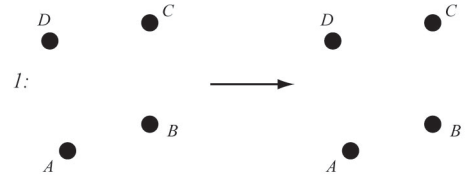

Figure 19. Rule 1, an identity rule used to search for four point-nodes.

One match would be to map *A→pn1, B→pn2, C→pn3* and *D→pn4*, or in short the permutation *1234*. Another would be to map *A→pn3, B→pn2, C→pn4* and *D→pn1*, or again in short *3241*. In fact there are 24 ways of mapping the four nodes of rule 1 onto the nodes of *Q*.

S_4 = *{1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2341, 2413, 2431, 3124, 3142, 3214, 3241, 3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321}*

That is, the subgraph is found in all its variations, or in group theoretic terms, all twenty-four permutations of the symmetric group S_4 for four elements are found. While this can undoubtedly be of use for certain rules of the form $U_{1j} \rightarrow U_{2j}$ in most cases such an abundance of matches will be overwhelming. The issue here, and the one which will be discussed in this section, is how to constrain the result set to a more specific subset of S_4.

3.1 The constrained topology patterns

The first constraint we can and most likely will want to introduce is to restrict the points to a certain topology, to introduce a sense of neighborhood. This can be done by connecting the points with lines, in the graph the respective nodes and edges are added.

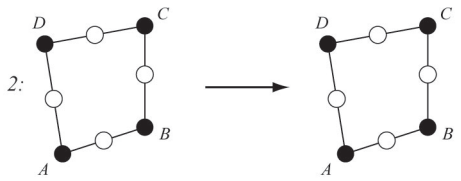

Figure 20. Rule 2 will find all general quadrilaterals.

Searching for the pattern shown in rule 2 (Figure 20) will result in eight matches per quadrilateral. At this point it is important to distinguish between the automorphism group of the graph and the symmetry group of the shape it represents. The automorphism group captures the symmetry of the graph, it contains all mapping of the graph onto itself. Using the part-relation graph model the symmetry group of the shape will always be a subgroup of the automorphism group. The eight matches correspond to the dihedral group D_4, the automorphism group of the graph describing quadrilaterals. A general quadrilateral has no symmetries except for the identity, thus its symmetry group is the cyclic group C_1. The results are reconciled by interpreting them as the cosets of C_1 in D_4. Geometrically it means that eight general quadrilaterals *Q1-Q8*, topological variations of each other, are returned.

D_4 = *{*C_1 *= Q1 = {1234}, Q2 = {1432}, Q3 = {2143}, Q4 = {2341}, Q5 = {3214}, Q6 = {3412}, Q7 = {4123}, Q8 = {4321}}*

While it is important to be able to query the graph for topologies, often this will not suffice. It is important to be able to further restrict the query to

return only certain shapes. This will result in more control over how parametric a rule should be. With enough restrictions set in place it is even possible to formulate an entirely static, non-parametric rule.
However, topologies are all that can be searched for using only the nodes and edges described in section 2. Everything that goes beyond topology is beyond the realm of the proposed model. Several additional kinds of nodes and edges could be introduced, such as nodes representing lengths and directions, to extend the possibilities, but this approach would be somewhat counter intuitive and would lead to rather complex graphs. Instead it is easier to add attributes for the *x*-, *y*- and *z*-coordinates to the point nodes. Once this is done attribute conditions can become part of the search pattern. Geometrically attribute conditions can be interpreted as constraints, as known from parametric modeling.

So if we are looking for a more specific geometry than just a general quadrilateral, additional attribute conditions will be necessary. Creating a topology and then constraining it to a specific geometry is much like one would proceed if describing a shape in a parametric modeling application. The advantage is that like one parametric model can represent numerous geometries; in this case one search pattern can find numerous shapes.

In order to find parallelograms only, one would have to constrain opposite edges to being parallel.

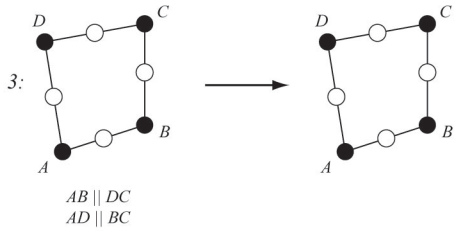

Figure 21. Rule 3 will find all parallelorams.

While the pattern in rule 3 will now only return parallelograms and not general quadrilaterals, there will still be eight matches per parallelogram. These eight matches can again be interpreted as the automorphism group D_4 of the graph including the constraints. Since we are dealing with parallelograms this time it must be dissected using the respective symmetry group, the cyclic group C_2. The results are the four parallelograms *P1-P4* or the cosets of C_2 in D_4.

$D_4 = \{C_2 = P1 = \{1234, 3412\}, P2 = \{2341, 4123\}, P3 = \{2143, 4321\}, P4 = \{1432, 3214\}\}$

It might be somewhat counter intuitive that eight matches are returned for a parallelogram since its symmetry group C_2 is only of order 2. This behavior is due to the ‘elastic’ nature of parametric rules. We are not looking for a parallelogram with specific metric features, but for all parallelograms at once. It becomes obvious if one thinks of the parametric rule shown in

Figure 22. This rule can be applied to a parallelogram in eight different ways.

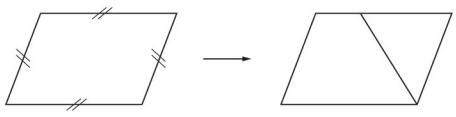

Figure 22. A parametric rule which can be applied to a parallelogram in eight different ways.

Visually this might still not be intuitive since one is used to reading rules as being non-parametric. Sometimes verbal instructions prove to be more flexible. Consider for example, the algorithm:

"Take any three corner points P1, P2, P3 from a parallelogram so that P1 is connected to P2 by an edge and P2 is connected to P3 by an edge. Find a point P4 one-third along the line P3P2. Draw a line from P4 to P1."

Continuing towards finding squares one could proceed by constraining adjacent sides to be of equal length.

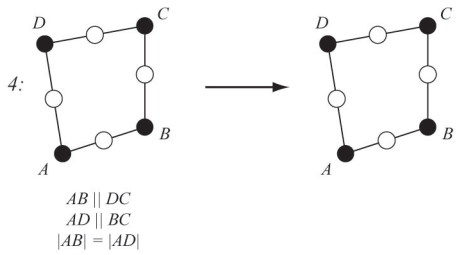

Figure 23. Rule 4 will find all rhombs.

Rule 4 shown in Figure 23 will return eight isomorphisms for each rhomb. So for each rhomb two of the parallelograms found before will turn out to be the same. In symmetry terms a reflection was added to the cyclic group C_2, that captures the symmetry structure of the parallelogram, to produce the dihedral group D_2, which in turn is the symmetry group of the rhomb. Thus, these matches can again be interpreted as two rhombs *R1* and *R2* or the cosets of D_2 in D_4. It might seem as though little progress is being made since there are still eight matches. However these matches are now guaranteed to be at least a rhomb, if not a square, whereas beforehand 'only' a parallelogram was ascertained.

$D_4 = \{D_2 = R1 = P1 \cup P3 = \{1234, 2143, 3412, 4321\}, R2 = P2 \cup P4 = \{1432, 2341, 3214, 4123\}\}$

Finally we must ask that the diagonals be of equal length.

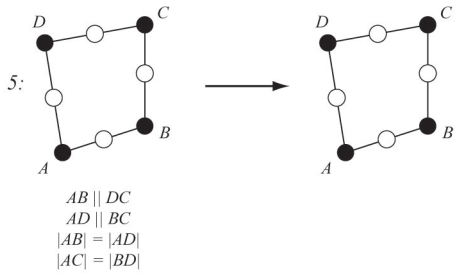

Figure 24. Rule 5 will find all squares.

Rule 5 shown in Figure 24 will return eight isomorphisms for each square. Again this is made possible because two of the rhombs will turn out to be same and the symmetry group of the square is D_4. In this case all eight isomorphisms can be interpreted to be instances of the same square *S1* under the operations of its symmetry group. Here the automorphism group of the graph is equal to the symmetry group of the square.

$$D_4 = S1 = R1 \cup R2 = \{1234, 1432, 2143, 2341, 3214, 3412, 4123, 4321\}$$

3.2 Incorporating the R

If not all eight isomorphisms are needed because, for example, the rule itself exhibits some symmetry and therefore cancels out some of the possibilities, additional constraints are required. It is best to incorporate one's knowledge of the right hand side in the search pattern of the left hand side. This will prevent having to deal with unnecessary matches later on. For example, Figure 25 shows a rule, which exhibits some symmetry. Because the square is moved along a diagonal there are only four possible ways of applying the rule, if it were moved along an arbitrary vector, not along an axis of symmetry, there would be eight possible applications.

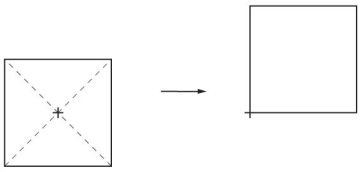

Figure 25. Rule to move a square along its diagonal.

In this case it is necessary to distinguish between the left and the right shapes. Defining a sense of rotation can filter out reflections. Hence for rule 6 (Figure 26) a constraint on the cross product of two vectors is added. Such a constraint can be used to return either left-turning or right-turning shapes.

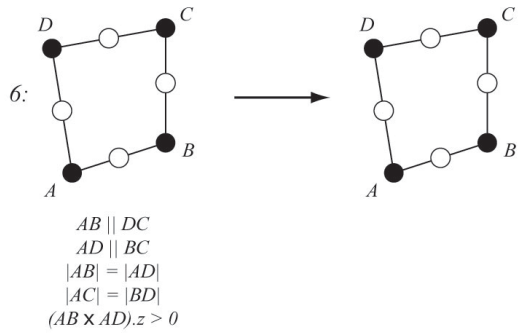

Figure 26. Rule 6 will return the 4 isomporphisms of a square that can be generated by rotation.

Here the group C_4, covering the four rotations of the square, is returned.

$C_4 = \{1234, 4123, 3412, 2341\}$

A single isomorphism is returned if a specific point is requested to be the bottom-left most point. Such a constraint is shown in rule 7 (Figure 27).

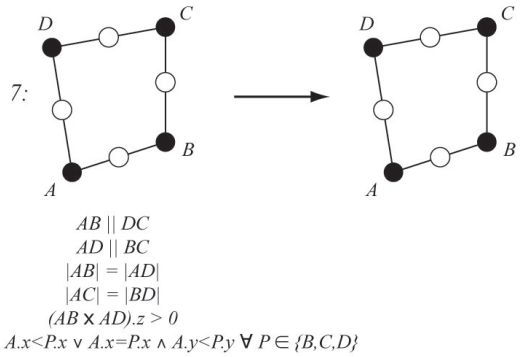

Figure 27. Rule 7 will return only one isomorphism per square.

Only the identity element C_1 is returned from the query. It is still necessary to keep the constraint on reflections, lest two matches are returned.

$C_1 = \{1234\}$

4 Implementation

Several interfaces to the GRAPE library have been implemented. Most are based upon commercial CAD packages, so the user can act in a familiar environment and manually process the result once the derivation has finished. One interface is implemented as a web application. The interfaces differ based on the possibilities offered by the host application. The web interface does not offer a visual editor.

The editor is a key component of the application for its successful and seamless integration in design practice. Marc Tapia's seminal paper on shape grammar implementations (Tapia, 1999) has highlighted many of the difficulties faced while designing a shape grammar editor. The greatest difficulty is of course the ambiguity in the definition of the rules that often

needs be constrained. Filling in the gaps is something humans are so good at they tend to do it without noticing, so as intuitive as some shape grammar rules may seem while viewing them and reading through their description, they will often prove to be too ambiguous to hand to a computer without further editing.

Even simple rules as the one shown in Figure 28(a) cannot be used without additional detail. The rule is taken from the Palladian grammar (Stiny and Mitchell, 1978) and shows the concatenation of spaces. Reading the accompanying description and looking at example derivations it becomes clear that the rule is parametric. It cannot only be applied to the configuration depicted on the left hand side of Figure 28(a), it will also match horizontally elongated spaces as shown in Figure 28(b). Graphically there then seems no reason why it couldn't also match vertically elongated spaces as shown in Figure 28(c), however this would allow configurations with several non-rectangular spaces to be created and hence must be prevented in order to stay within the body of Palladian villas. To model this rule unambiguously for the automation would require adding several constraints to the pattern.

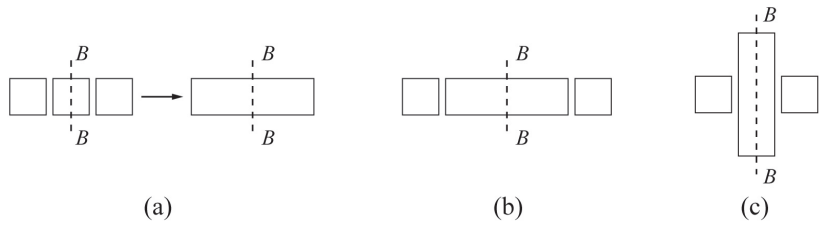

Figure 28. From its depiction it is not clear that rule 12 from "The Palladian grammar" (a) is applicable to horizontally elongated spaces (b) but not to vertically elongated ones(c).

4.1 Editor

So implementing an editor is mainly a matter of establishing conventions of how to unambiguously describe a rule. Since the library is built upon graph grammars the most straightforward way of defining a parametric shape rule, is to directly define its graph equivalent. In any case all graphically defined rules will have to be parsed into graph rules in order to be applied, and since the graphic rules are a source of ambiguity avoiding it can be beneficial. Listing 1 shows the graph grammar definition for the rule from Figure 22 in the GrGen.NET modeling language (Blomer et al., 2012).

```
rule split_parallelogram {
```

```
        A:PointNode -:RealPartEdge- :LineNode -:RealPartEdge-
        B:PointNode -:RealPartEdge- :LineNode -:RealPartEdge-
        C:PointNode -:RealPartEdge- :LineNode -:RealPartEdge-
        D:PointNode -:RealPartEdge- :LineNode -:RealPartEdge- A;

        if {ABparallelToCD(A, B, C, D);}
        if {ABparallelToCD(A, D, B, C);}

        modify {
                D -:BoundaryEdge- :LineNode -:BoundaryEdge-
                E:PointNode;

                eval {
                        E.x = B.x + (A.x - B.x)/3;
                        E.y = B.y + (A.y - B.y)/3;
                        E.z = B.z + (A.z - B.z)/3;
                }
        }
}
```

Listing 1. The definition for the rule in Figure 22.

While we join Mitchell (1990) in arguing that there is merit in verbally expressing ones intentions, at times it is more abstract than one would hope for from a shape grammar interpreter. So in order to lower the access threshold and to address the visual reasoning promoted by shape grammars, additional modes of defining rules will be explored.

The approach implemented for the visual editor is based on constructive, parametric geometry to formulate unrestricted rules. To start at the beginning the left hand side of a rule will be dealt with first. Figure 29 shows some examples. A shape is generally taken to be parametric, unless indicated otherwise. Thus a square will actually be interpreted as a general quadrilateral, unless constraints are placed on the geometry showing the specific properties of the square actually matter. Coincidence of points or intersections is the only constraint applied automatically. Thus if two lines meet in a point, this is assumed to be important. External intersections on the other hand have to be modeled explicitly.

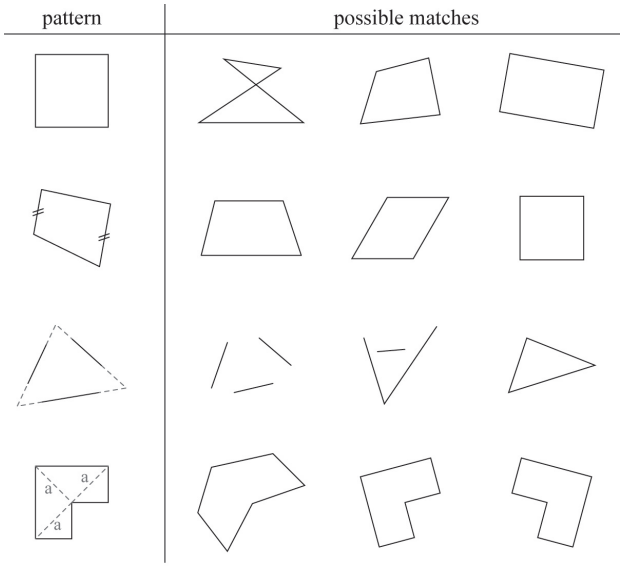

Figure 29. Several patterns and some possible matches.

Construction lines may be used to place constraints on relations not otherwise available. These lines are placed on a separate layer, here they are shown as grey dashed lines. They are not part of the derivation, rather they are only a vehicle to help describe the geometry. For example the diagonals of a parallelogram might be constrained to have equal length, resulting in a rectangle. Hereby the diagonals do not have to explicitly be present in the derivation to find a match, nor will they be added once the rule is applied.

Once the pattern on the left hand side of the rule has been sufficiently described, the right hand side is added. Hereby it is important to note that the description of the right hand side must be sufficiently anchored in the description of the pattern on the left, lest unpredictable results may occur due to indeterminacy In fact the left and right hand sides are not divided by an arrow as is customary, rather they are drawn on top of each other. It is also interesting to note that by default the left hand side is kept, that is x → x + y, if the whole or parts of the left hand side are to be deleted it has to be marked appropriately.

Figure 30 shows a rule to embed one quadrilateral within another. The circles are construction lines, their radii are determined by constraints. Thin lines are part of the left hand side of the rule, thick lines are added on the right hand side.

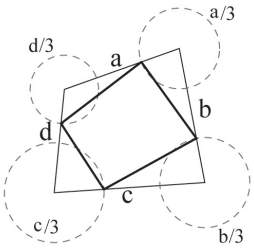

Figure 30. A rule to embed a quadrilateral within another.

4.2 Parallel descriptions

Chase (2002) describes a user interaction model for shape grammar applications. According to this model the implementation described in this section is a semi-automatic shape grammar system. The user selects a rule, the computer searches for matches, the user picks one and continues to select the next rule.

So far the graph data-structure used for the matching process was shown and the method of defining parametric rules has been explained. However, once a rule is applied to the graph, the connection between the graph representation and the shape representation deteriorates. If new subshapes have emerged during the rule application, they will go undetected by the graph.

An important aspect of this implementation is the notion of parallel description; one that accounts for what we see, a visual description, and another that accounts for the indexing, bookkeeping and actual implementation of what (in theory) we see, a symbolic description. The visual description consists of lines that merge and fuse at any time of the computation to create any shape conceivable. The symbolic description consists of a graph that captures at any time of the computation the relations of these lines one to another. Any execution of a shape rule requires its translation in graph form, its computation within the graph structure of the design, and the retranslation of the graph in a pictorial setting.

Switching back and forth between these descriptions not only necessary for the obvious reason of offering a user interface based on shapes rather than on graphs. It is also important when dealing with emergent behavior since intersections are detected in the shape description only.

4.3 Complexity

Since parametric subshape recognition is NP-hard (Yue et al., 2009) using subgraph matching, which is NP-complete, for the computation does not add to the complexity. Moreover, there is an extensive body of work in the field of subgraph matching trying to evade as often as possible the theoretic worst case and make subgraph matching feasible in practice. This can for example be done by taking the structure of the graph into account and devising an optimized matching strategy (Batz, 2006).

Since no benchmarking system for parametric shape grammars has been devised yet, the following examples are an attempt to provide some reproducible results for future reference. Furthermore it should be shown that the given approach can return results within practical time and resource limits. All tests were executed on the web version of GRAPE, running on a server with 613MB memory and two 1,2 GHz 2007 Opteron processors.

The Palladian grammar has been partially implemented using the graph grammar modeling language. The derivation of the Villa Malconenta as shown by Stiny and Mitchell (1978) is generated [Figure 31(a)] in order to use it as a benchmark. The entire derivation, starting from an empty graph

and consisting of 47 rule applications, can be executed in well under half a second. Thereby the graph grows to at most 456 nodes and 408 edges. Stiny and Mitchell originally use 58 rule applications; the discrepancy can be explained by minor differences in the rule set.

While generating the Villa Malcontenta there are no rules that return a high number of matches because the rules are well sequenced and the derivation contains only little repetition. For contrast two additional examples were executed, both are applied to a 10x10 square grid. The first rule could also be implemented by non-parametric shape grammar applications, it searches for squares without isomorphisms in order to rotate them by 45°. The rule returns 385 matches [Figure 31(b)]. The second rule is parametric, it searches for rectangles and rotates them by 90°, again without isomorphisms. The rule returns 3025 matches [Figure 31(c)]. Both rules return results in about half a second.

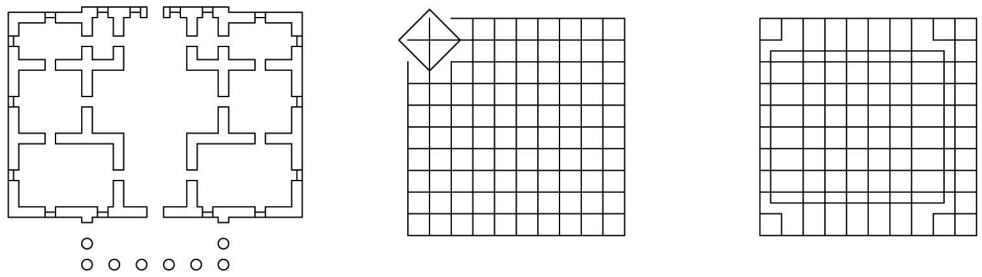

Figure 31. The generated plan of the Villa Malcontenta (a); In a 10x10 square grid 385 squares (b) and 3025 rectangles are found (c) to rotate.

5 Conclusion

The GRAPE library enables shape grammar implementations that support both emergent subshapes and parametric rules. It is based upon graph grammars and while the examples discussed here were all from the algebra U_{12}, the library currently supports grammars up to $U_{13} \times V_{03}$. Since the method described above can also handle additional dimensions and object types expanding the software to other classes of algebras is possible, including the algebras of higher dimensional shapes, curves and weights. First promising results of implementing grammars in $U_{23} \times V_{03}$ can be reported. Figure 32 shows the eight results for one spatial relation under varying labeling strategies using the Fröbel building blocks as discussed by Stiny (1980b). The results were generated using the GRAPE web interface and by modeling the blocks as six individual faces each.

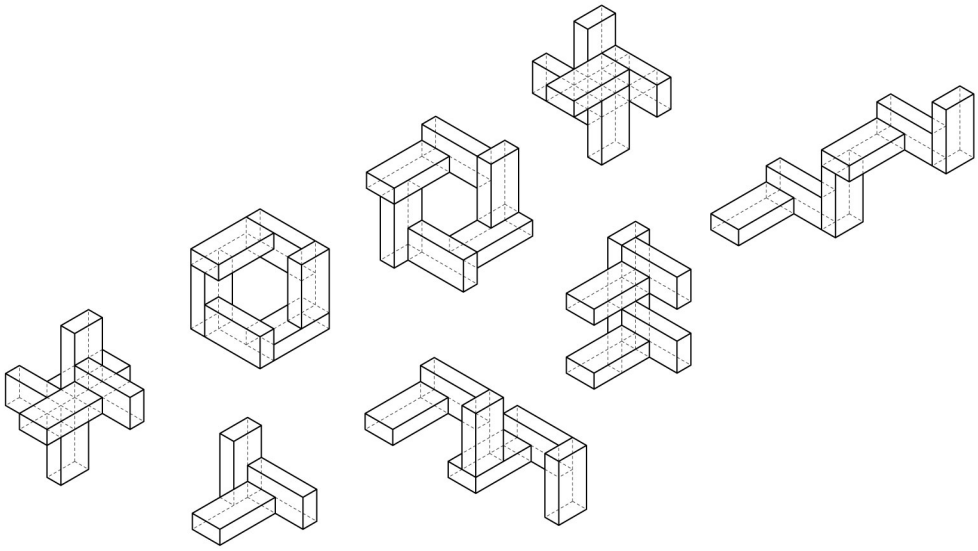

Figure 32. Derivations examining one spatial relation of Fröbel's building blocks

Acknowledgements
We would like to thank George Stiny for his support and encouragement and especially for his suggestions on the general solution of line intersections and symmetry subgroup detections.

References

Batz G V, 2006, "An Optimization Technique for Subgraph Matching Strategies" Internal Report, Universität Karlsruhe

Blomer J, Geiß R, Jakumeit E, 2012, *The GrGen.NET User Manual*, Universität Karlsruhe

Chase S C, 2002, "A model for userinteraction in grammar-based design systems" *Automation in Construction* **11**(2) 161 – 172

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

Duarte J P, Correia R, 2006, "Implementing a Description Grammar for Generating Housing Programs Online" *Construction Innovation* **6** 203-216

Drewes F, Kreowski H-J and Hable A, 1997, "Hyperedge Replacement Graph Grammars" in *Handbook of Graph Grammars* Ed. G Rozenberg Volume 1

Economou A, 1999, "The symmetry lessons from Froebel building gifts" *Environment and Planning B: Planning and Design* **26**(1) 75 – 90

Economou A, 2006, "Tracing Axes of Growth" in Visual thought: The Depictive Space of Perception, Advances in Consciousness Research 67 Ed. L Albertazzi 351 – 365

Economou A and Grasl T, 2009, "Point Worlds. Computation: The New Realm of Architectural Design" in *27th eCAADe Conference Proceedings* pp. 221 – 228.

Ertelt C and Shea K, 2009, "An application of shape grammars to planning for CNC machining" in *Proceedings of the ASME 2009 IDETC/CIE Conference*

Flemming U, 1987, "More than the sum of parts: the grammar of Queen Anne houses" *Environment and Planning B: Planning and Design* **14**(3) 323 – 350

Gips J, 1999, "Computer Implementation of Shape Grammars" *NSF/MIT Workshop on Shape Computation*

Grasl T, 2012, "Transformational Palladians" *Environment and Planning B: Planning and Design* **39**(1) 83 – 95

Heisserman J, 1991, *Generative Geometric Design and Boundary Solid Grammars* PhD dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA

Heisserman J, 1994, "Generative Geometric Design" *IEEE Computer Graphics and Applications* **14**(2) 37 – 45

Hoisl F and Shea K, 2011, "An interactive, visual approach to developing and applying parametric 3D spatial grammars" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (Accepted for publication).

Jowers I, Earl C, 2010, "The construction of curved shapes" *Environment and Planning B: Planning and Design* **37**(1) 42 – 58

Jowers I, Hogg D, McKay A, Chau H H, de Pennington A, 2010, "Shape Detection with Vision: Implementing Shape Grammars in Conceptual Design" *Research in Engineering Design* **21**(4) 235 – 247

Jowers I, Earl C, 2011, "Implementation of curved shape grammars" *Environment and Planning B: Planning and Design* **38**(4) 616 – 635

Keles H Y, Özkar M, Tari S, 2010, "Embedding shapes without predefined parts" *Environment and Planning B: Planning and Design* **37**(4) 664 – 681

Keles H Y, Özkar M, Tari S, 2012, "Weighted shapes for embedding perceived wholes" *Environment and Planning B: Planning and Design* **39**(2) 360 – 375

Knight T, 2003, "Computing with emergence" *Environment and Planning B: Planning and Design* **30**(1) 125 – 155

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

Krishnamurti R, Earl C F, 1992, "Shape recognition in three dimensions" *Environment and Planning B: Planning and Design* **19**(5) 585 – 603

March L, Steadman P, 1974, The geometry of environment: an introduction to spatial organization in design (MIT Press)

McCormack J P, Cagan J, 2002, "Supporting designers' hierarchies through parametric shape recognition" *Environment and Planning B: Planning and Design* **29**(6) 913 – 931

McCormack J P, Cagan J, 2006, "Curve-based shape matching: supporting designers' hierarchies through parametric shape recognition of arbitrary geometry" *Environment and Planning B: Planning and Design* **33**(4) 523 – 540

Michell W J, 1990, The Logic of Architecture: Design, Computation, and Cognition (MIT Press)

Read R C, Corneil D G, 1977, "The graph isomorphism disease" *Journal of Graph Theory* **1**(4) 339 – 363

Ržiha, F, 1883, *Studien über Steinmetz-Zeichen* (K.k. Hof- und Staatsdruckerei)

Steadman P, 1976, "Graph-theoretic representation of architectural arrangement" in *The Architecture of Form* Ed. L March (Cambridge University Press)

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

Stiny G, 1980b, "Kindergarten grammars: designing with Froebel's building gifts" *Environment and Planning B* **7**(4) 409 – 462

Stiny G, 1991, "The algebras of design" *Research in Engineering Design* **2**(3) 171 - 181

Stiny G, 1992, "Weights" Environment and Planning B: Planning and Design **19**(4) 413 – 430

Stiny G, 2006, Shape: Talking about Seeing and Doing (MIT Press)

Stiny G and Gips J, 1972, "Shape Grammars and the Generative Specification of Painting and Sculpture" in *Information Processing 71* Ed. C V Freiman

Stiny G, Mitchell W J, 1978, "The Palladian grammar" *Environment and Planning B* **5**(1) 5 – 18

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

Trescak T, Esteva M and Rodriguez I, 2009, "General shape grammar interpreter for intelligent designs generations" in *Proceedings of the Computer Graphics, Imaging and Visualization 2009* Ed. B Werner pp 235–240

Yue K, Krishnamurti R and Grobler F, 2009, "Computation-friendly shape grammars: Detailed by a sub-framework over parametric 2D rectangular shapes" in *Joining Languages, Cultures and Visions: CAADFutures 2009* Eds. T Tidafi and T Dorta pp. 757- 770