Predicates and Directives for a Parametric-associative Matching Mechanism for Shapes and Shape Grammars

Rudi Stouffs[1]
[1]*National University of Singapore*
[1]*stouffs@nus.edu.sg*

Predicates and directives are proposed to extend the versatility and expressivity of a shape rule specification `language'. Specifically, a predicate serves to express a condition on the application of a parametric-associative shape rule that cannot simply be explicated within the left-hand side shape. A directive, on the other hand, is a value specification that is required when applying a parametric-associative shape rule, where this value specification cannot be derived from or expressed within the right-hand-side shape. We discuss the benefits of a set of predicates and directives to address seemingly simple requests that might actually be hard to express otherwise. Specifically, we elaborate on the motivation for introducing predicates and directives, and demonstrate the various predicates and directives in more detail, including their implementation.

Keywords: *shape rule, shape grammar, predicate, directive, parametric-associative rule*

INTRODUCTION

Shape grammars are a formal rewriting system for producing languages of shapes (Stiny 1980). A shape grammar specifies a set of rewriting rules that operate on shapes and, each, replace a matched subshape with another shape under some allowable transformation. Although conceived for producing both existing languages and new languages of design (Stiny and Gips 1972), shape grammars have been used mainly for analytical purposes as a means to understand the rules underlying given design styles. Only in a few cases have they been used as an exploratory tool in the design process. Beirão et al. (2009) suggest distinguishing between 'grammars of designs', being analytical grammars that reflect on a given body of designs, and 'grammars for designing', to denote the progressive development of a new grammar for a new design context. We will adopt the term design grammars to denote 'grammars for designing'.

Design grammars present additional difficulties compared to analytical grammars. Developing an analytical grammar involves systematically determining all possible shape rule variations corresponding to the different designs in the given body, and encoding these into a grammar. Rule variations are necessarily finite and the encoding is done by the developer of the grammar, not by the user. The complexity of the rule is therefore less important. Examples abound (e.g., Stiny and Mitchell 1978; Downing and Flemming 1981; Çagdas 1996). In the case of the Palladian grammar (Stiny and Mitchell 1978), rules develop the grid and its extent, create openings, clean up the line drawings, etc.

For design grammars, however, the designer acts

as both the developer and the user of the shape rules. Rules may be defined from scratch or as alterations of existing rules. The complexity of expressing one's ideas into workable shape rules that can be matched and applied by the shape grammar interpreter, i.e., the rule engine, then becomes an important limiting factor. This complexity may be related to some extent to the user interface but, more importantly, will depend on the versatility and expressivity of the rule 'language'. In principle, a shape rule combines a left-hand-side and a right-hand-side. The left-hand-side specifies the shape that is searched for and matched under an allowable transformation, whereas the right-hand-side specifies the shape that will be used to replace the matched shape under the same transformation. Therefore, the designer needs to identify, at the least, the left-hand-side and right-hand-side shapes as compositions of spatial elements, i.e., as drawings. Then, the rule language minimally depends on the types of spatial elements that are accepted by the rule engine, e.g., points, line segments, plane segments, curves, etc. However, considering shapes of spatial elements only can severely complicate the process of identifying which rule to apply or where to apply the rule to.

All analytical grammars generally consider labels as attributes to points to constrain rule application and to assist in identifying the order in which rules can apply. In the example of the Palladian grammar (Stiny and Mitchell 1978), labelled points serve to constrain which grid cells a particular rule may apply to. For example, a rule to extend the grid with one row should only apply to the last row. Besides labels, other spatial attributes can also be considered to guide rule application and, at the same time, enrich the language of designs the grammar can produce. For example, Stiny (1981) considers a description function to construct verbal descriptions of a design. Where Stiny's (1981) shape descriptions simply reflect on the spatial elements—made up of blocks from Froebel's building gifts—that constitute the design and the way these are combined, other authors describe the adoption of shape descriptions to guide the generative process (Stouffs 2018a). Thus, spatial elements, shape attributes and shape descriptions can together constrain rule application. In addition to the left-hand-side of the shape rule matching a part of the given shape spatially, any shape attributes specified in the left-hand-side must be available in the matched part of the given shape and any shape descriptions specified as part of the left-hand-side must match corresponding shape descriptions accompanying the given shape.

Even then, some conditions cannot simply be expressed as a combination of spatial elements, spatial attributes and/or shape descriptions. Consider the condition that some part of the shape, e.g., the interior of a polygon, should be devoid of any other spatial elements. Shape rules only specify what should be present and matched, not what should be absent. The left-hand-side of a shape rule matches any part of a shape that is equivalent under an allowable transformation. However, the shape can always contain other additional spatial elements that are unaffected by, nor involved in the matching.

In order to be able to express void conditions, Liew (2004) raises the concept of a zone descriptor that can serve to specify a zone to be devoid of any spatial elements. In addition, he extends the concept of the zone descriptor, and of descriptors in general, to other examples. Liew (2004) represents the zone descriptor graphically, as a hatched region. However, representationally, it makes little sense to conflate the spatial representation of a shape in this way as it will necessarily complicate the matching process. Instead, descriptors can be expressed as a separate representational structure, just as shape descriptions are commonly represented separately from the spatial structure of the shape. Note that shape attributes, instead, are necessarily represented as part of the spatial structure they augment.

Liew (2004) uses the term descriptor quite broadly to identify a collection of quite different meta-language elements. Instead, here we adopt the term predicate specifically to express a condition on the application of a rule that cannot simply be expli-

cated within the left-hand side shape. The void predicate is one such example. In addition, we adopt the term directive to indicate a value specification that is required when applying a rule, where this value specification cannot be derived from or expressed within the right-hand-side shape. We will provide examples of both predicates and directives below.

Thus, beyond the types of spatial elements accepted or the various non-spatial attributes allowed for, the expressivity of shape descriptions (Stouffs 2018b, 2018c), on the one hand, and the ability to add various kinds of predicates and directives to a rule, either to constrain the matching (the left-hand-side of the shape rule) or inform the manipulation (the right-hand-side of the shape rule), on the other hand, also impact the expressivity of the rule language. Nevertheless, it is not because a set of rules can be constructed to implement one's idea, that the designer will easily conceive of this rule set or even find it worthwhile to spend the effort in developing such a rule set. In this paper we specifically focus on predicates and directives, and the role predicates and directives can play in translating one's generative idea in a rule or rule set. That is, we discuss the benefits of a set of predicates and directives to address seemingly simple requests that might actually be hard to express otherwise. Specifically, we elaborate on the motivation for introducing predicates and directives, and demonstrate the various predicates and directives in more detail, including their implementation.

PARAMETRIC-ASSOCIATIVE SHAPE RULES

Generally, a shape rule applies to a given shape when it matches some part of the given shape under an allowable transformation. Rule application then involves replacing the matched subshape with another shape under the same transformation. Allowable transformations may be of different kinds (Wortmann and Stouffs 2018). Shape grammars commonly consider transformations of similarity, allowing for translation, rotation, reflection and uniform scaling. Whether these are further constrained to isometric or Euclidean transformations (disallowing scaling) or instead extended to, e.g., affine transformations (allowing for stretching and shearing), these all have in common that they can be represented by a transformation matrix.

Stiny (1977) also suggests a parametric shape grammar, in which parametric shape rules operate on non-parametric shapes. A parametric shape rule embeds (numerical) parameters that govern the position of some spatial elements (or their boundary elements) within the left-hand-side shape (and corresponding elements in the right-hand-side shape). Stiny does not suggest any implementation. Instead, Woodbury (2016) presents the mechanisms of a shape schema grammar, in which parametric shape rules operate on parametric shapes. Here, the matching mechanism is one of constraint satisfaction. However, the algorithm is intractable and no implementation yet exists. Instead, all implementations of parametric-style shape grammars rely on a graph-based matching mechanism (e.g., Wortmann 2013; Grasl and Economou 2013; Stouffs 2018). While the exact graph representation may differ from one implementation to another, commonly, line segments, or the infinite lines carrying these segments, and their intersection points, serve as edges and vertices (or vice versa) of the graph. In addition, one or more kinds of associations between line segments (or their intersection points), such as parallelism and perpendicularity, or equal lengths and distances, are considered as invariants for the matching process. As such, we adopt the term parametric-associative shape rules. From here on, we limit our discussion to parametric-associative shape rules.

PREDICATES AND DIRECTIVES

A predicate serves to express a condition on the application of a parametric-associative shape rule that cannot simply be explicated within the left-hand side shape. The void predicate is one such example, another is the shortest_line predicate. The latter predicates that the specified line segment must be the shortest line segment within the matching shape.

For example, when matching an *n*-sided polygon, there may be *n**2 possible matches to the same polygon, as simple permutations of the cyclically ordered matching of the polygon's sides in one direction, or by reflection in the other direction. Instead, by identifying a shortest line segment, this may be reduced to only two possible matches, if the shortest line segment can be unambiguously identified. Note that the predicate can be assigned to any line segment in the left-hand-side shape and this line segment is not required to actually be the shortest among all line segments within this shape. The condition only applies to any matching (sub)shape.

Directives, on the other hand, are value specifications that are required when applying a parametric-associative shape rule, where this value specification cannot be derived from or expressed within the right-hand-side shape. For example, a new line segment that is added in the right-hand-side shape may be required to be at a certain distance from a given point. If this distance is not already apparent in the left-hand-side shape, e.g., as the distance between two existing points, then it may be impossible to determine this distance unequivocally otherwise.

Note that some predicates (or directives) may be automatically embedded in the rule matching (and application) mechanism and do not need to be explicitly specified. For example, Grasl and Economou (2013) present an implementation of a parametric-associative shape grammar interpreter where the matching mechanism automatically recognizes associations of equal length (or distance) as matching constraints, thus automatically recognizing and matching regular polygons. For example, a square has four sides of equal length and, additionally, two diagonals of equal distance. In our own implementation, the matching mechanism automatically detects parallel and perpendicular lines. As such, without any additional constraints specified, a square will match any rectangle, having twice two parallel segments that are perpendicular with respect to one another (Stouffs 2019).

Predicates

Liew (2004) proposes two types of descriptors for the 'contextual requirements phase', a zone descriptor and a maxline descriptor. As explained before, the first excludes certain elements form a specified area. It actually comes in two flavors. Firstly, the void descriptor excludes any spatial elements (points or line segments in the case of Liew (2004)) from the specified area. Secondly, the exclude descriptor is able to limit the kinds of spatial elements that should be excluded, for example, line segments with a 'gray' attribute.

As discussed before, while Liew (2004) represents the zone descriptor graphically, as a hatched region, instead we choose not to conflate the spatial representation of a shape and adopt a separate representational structure for predicates and directives. Here, we choose to represent predicates and directives in textual form. For example, the specification of the void predicate takes the form of (a list of) a list of coordinate pairs (in 2D) or triples (in 3D) corresponding to the vertices of the polygonal area(s) within which no spatial elements can be present (Table 1). It must be noted that while the vertices are explicated by their coordinates, they must necessarily coincide with any of the line segments in the left-hand-side of the shape in order for the vertices to be recognized via the parametric-associative matching mechanism. As such, while the area must be devoid of any spatial elements, this does not apply to the boundary of the area. Note also that while the void area applies in three dimensions, it is (currently) strictly a two-dimensional area.

It is assumed that the void predicate can be generated for any given polygonal area. For example, in the SortalGI plug-in for Rhino/Grasshopper (Stouffs 2018a), a 'void predicate' component accepts any polygonal area, whether specified as a sequence of points or line segments or, instead, as a closed polyline or surface, and returns the textual specification of the corresponding void predicate.

The void predicate has been extended to accept the type of spatial element that should be ex-

Table 1
Seven predicates. Each predicate accepts a list of arguments in curly brackets.

predicate	elementary argument	example
void	parenthesized list of coordinate tuples ((x1, y1, z1), (x2, y2, z2), …)	void: {((0,0}, (10,0), (10,10), (0,10))}
	parenthesized list of coordinate tuples, with initial list of spatial element types (in curly brackets) ({type, …}, (x1, y1, z1), (x2, y2, z2), …)	void: {({pointP3D, lineSegmentP3D}, (0,0,0), (10,0,0), (10,10,0), (0,10,0))}
maxline	line segment tag #lnTag	maxline: {#ln1, #ln2, #ln3}
bound	line segment tag with preceding '>' and/or succeeding '<'	bound:: {#ln1<, >#ln2}
nolabel	spatial element (point, line or plane segment) tag #tag	nolabel: {#pt, #ln1, #ln2}
embeds	Parenthesized list containing a plane segment tag as container element and one or more point and/or line segment tags as embedded elements (#plnTag, #tag, …)	embeds: {(#plnTag, #ln1, #ln2, #pt)}
shortest_line	line segment tag #lnTag	shortest_line: {#ln1}
longest_line	line segment tag #lnTag	longest_line: {#ln1, #ln2}

cluded, e.g., points, line segments, plane segments, or curves, allowing other spatial elements to be present within the void area. Fig. 1 demonstrates both versions of the void predicate using an example inspired by Stiny's (1977) Chinese ice-ray lattice grammar. Stiny's shape rules iteratively subdivide an initial rectangle into a composition of triangles, quadrilaterals and/or pentagons, each of a minimum size, in each iteration by splitting one polygon into two smaller polygons. Fig. 1 shows two rules that together can identify triangles that are void inside, and thus have not been already subdivided, and have an area larger than a given minimum area. The first rule applies to any void triangle and adds a plane segment that fits the triangle. The second rule follows from the first rule and removes the plane segment again if the area is insufficiently large. The area condition is not implemented as a predicate but instead as a description rule. The description rule contains the minimum area and this value is compared with the area of the plane segment. The value of the description rule remains unaltered at all times. Applying first the first rule exhaustively and subsequently the second rule exhaustively would result in the identification of all triangles that can be further subdivided.

Liew's (2004) zone descriptor is allowed to extend (in principle, infinitesimally) beyond the area's line boundaries. For example, when two line boundaries identifying adjacent sides to a void area only touch, with neither line boundary extending beyond the intersection point, and the zone descriptor engulfs this intersection point, then the respective matching line segments are not allowed to extend beyond their intersection point either. While this ability might be visually attractive, it is a lot more difficult to explicate. Instead, we consider a bound predicate to indicate whether an apparent endpoint to a line segment should, in fact, be an endpoint to the matched line segment or whether, instead, the line segment may extend beyond this endpoint.

The bound predicate accepts any number of line segment tags and requires for each endpoint of each line segment the specification of whether the bounding condition applies (see Table 1 for an example specification). Note that the endpoints are initially ordered as identified when constructing the line segment, but this may change upon manipulating the segment (e.g., through rule application) upon which the endpoints will be ordered corresponding their coordinates (first X, then Y and finally Z).

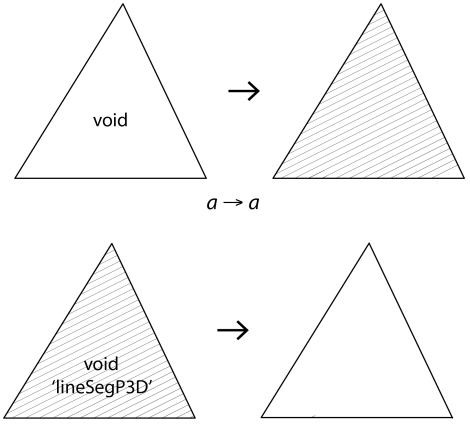

Figure 1
Two rules demonstrating the use of the void predicate. Note that, for visual purposes, the void predicate is not fully explicated but, instead, graphically designated.

If the bounding condition is specified to apply to both endpoints of the same line segment, this becomes an expression of Liew's (2004) maxline descriptor. This descriptor expresses that a matching line segment to the specified line segment must use its full extent to match this line segment, that is, it cannot be a part of a longer line segment. The maxline predicate implements this behavior and can be used as a shorthand for the bound predicate in the case that the bounding condition is specified to apply to both endpoints.

Instead of the exclude predicate, we consider a nolabel predicate (Table 1) which ensures that the specified spatial elements (whether points, line or plane segments) carry no attribute labels or descriptions. Descriptions can be interpreted as semi-structured labels, allowing parametric description rules to operate on labels and, thus, as representational extension of labels. In order to identify the 'no label' spatial element within the left-hand-side shape of the rule, we support the tagging of spatial elements. Spatial element tags can be understood as attributes to the elements, similar to labels (tags are recognized by the '#' symbol preceding the tag identifier). However, different from attributes, tags are particular to the rule in question and only subsist within the rule matching and application process of this rule. As such, tags are not considered attributes; within a predicate (or directive) specification, the tag solely serves to identify the spatial element the predicate (or directive) is referencing.

Note that the nolabel predicate does not entirely match the expressivity of the exclude descriptor, as it neither applies to a 'zone' but instead requires to explicate the 'no label' elements, nor does it allow to specify a specific attribute value to more narrowly limit what is meant to be excluded. As with all predicates and directives, we rely on use cases, observations of user actions and user feedback to provide us with insights into what kind of functionality is beneficial and desirable, and to adjust and extend current functionality where appropriate. One of the benefits noted of the zone descriptor type, is the fact that it allows to affect one type of spatial element with the specification of another type of spatial element, typically of a higher dimensional order. For example, the void predicate adopts the specification of a polygonal area (i.e., a plane segment) to express the absence of spatial elements of lower (or equal) dimension, e.g., points or line segments. This behavior is also present in the embeds predicate (Table 1) which allows one to specify that a point or line segment should be entirely embedded within a given plane segment.

Finally, other predicates considered are shortest_line and longest_line (Table 1), requiring a certain line to be the shortest line, respectively, the longest line, in the matching shape. The shortest_line predicate has been explained above, the longest_line predicate behaves entirely similar.

Directives

Directives, on the other hand, offer an easy way to provide additional information that cannot be expressed spatially in the right-hand-side of the shape. When adding a new line segment at a particular angle with respect to an existing line segment and/or of a particular length, the specific value of that angle or length may not be derivable from the shape as such, and an additional statement may be required to in-

dicate the respective value. For this reason, we consider, among others, an angle directive and a length directive, as well as a distance directive. Rather than only allowing for the specification of a fixed numeric value, we also consider the specification of the value through an expression that itself may refer to properties of other spatial elements, mimicking part of the expressiveness of descriptions.

Considering the example of Stiny's (1977) Chinese ice-ray lattice grammar, each rule allows for a convex polygon, whether a triangle, quadrilateral or pentagon, to be split into two new convex polygons by placing a single line between two of the original polygon's edges. The endpoints of the new line must necessarily lie between the endpoints of the respective original polygon's edges, but it is not specified where the point should be placed. As placing the point close to an existing endpoint may result in a very small polygon, Stiny (1977) offers the specification of a constraint that limits the absolute difference between the two resulting polygonal areas to be below a given value. Unfortunately, this is a constraint on the result from rule application, which cannot be dealt with in the context of predicates and directives, unless as part of a subsequent rule that checks upon the result. Here, we offer a partial workaround in the form of a point_on_line directive that allows to explicate a specific parameter value for the point's position on the line (between 0 and 1, assuming the endpoints have parameter values 0 and 1). The same directive can also be used to specify an allowable interval, by using an expression including the random function to determine the parameter value (Table 2, Fig. 2).

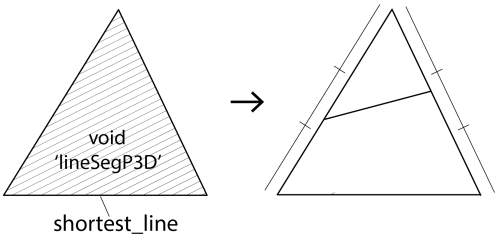

Figure 2
A rule demonstrating the use of the point_on_line directive, as well as the shortest_line predicate. Note that neither the predicate nor directive is fully explicated but, instead, they are both graphically designated.

Other directives mainly support the creation of new line segments, by indicating their length, direction, distance to an existing point or line segment, or angle with respect to an existing line segment. When creating a new line segment, it will only be unequivocal if, for example, it connects two (end)points that are already present or apparent in the left-hand-side, such as points of intersection of line segments and/or the infinite lines carrying these segments. Considering a matching mechanism that automatically recognizes parallel and perpendicular line segments, a new line segment can also be unequivocally determined if specified, for example, as a perpendicular segment to an existing line through an existing (end) point and extending until it touches another line. However, in most (other) cases, some information may be ambiguous, such as its length, direction, distance or angle (Table 2). For example, if we'd like to duplicate a line segment parallel to the original line seg-

Table 2
Five directives. Each directive accepts a list of arguments in curly brackets.

directive	**elementary argument**	**example**
point_on_line	pair of line segment tag and parameter value (#lnTag, parameter)	point_on_line: {(#ln1, 0.5), (#ln2, random(0.3, 0.7)}
distance	triple of new spatial element tag, existing spatial element tag and distance value (#newTag, #tag, value)	distance: {(#newln, #ln1, 10.0), (#newpt, #ln1, `random(0.3, 0.7)`}
direction	pair of line segment tag and direction vector (#lnTag, vector)	direction: {(#newln, (0.0,0.0,1.0})}
angle	triple of new line segment tag, existing line segment tag and angle value (#newLnTag, #lnTag, angle)	angle: {(#newln, #ln1, `pi/4`)}
length	pair of line segment tag and length value (#lnTag, length)	length: {(#newln, `#ln1.length`)}

ment, with the endpoints on a perpendicular to the original endpoints, we may be required to specify the distance between the two line segments (Fig. 3 top). Such distance can be specified as a definite numeric value or as a value derived from an expression, e.g., a random value within a numeric interval or the length (or other numeric property) of a spatial element. Such expressions take the format of a description, or part thereof. The distance directive can also be used to create a point with respect to an existing point or line segment, although the new point would necessarily need to lie on another segment or its infinite carrier line in order to be truly unequivocal. As another example, in the case of extruding a polygon (Fig. 3 bottom), the extrusion rule may need to specify both the direction vector and the length of extrusion.

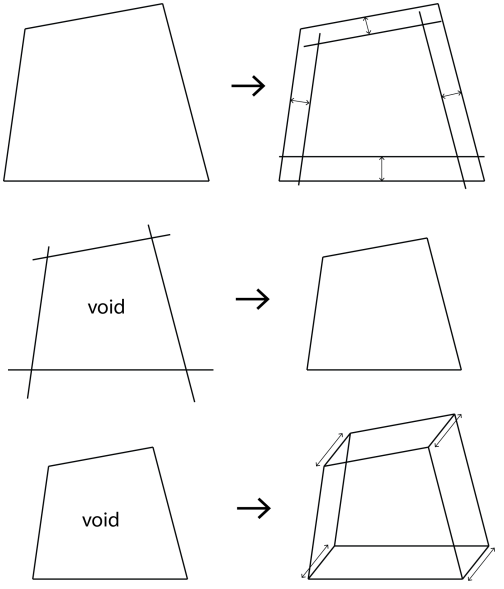

Figure 3
Three rules that extrude a scaled quadrilateral. The distance director is used to scale the quadrilateral (non-uniformly), while the direction and length directors serve to extrude the scaled quadrilateral. Note that the void predicate is additionally used to ensure the rules apply to the appropriate quadrilaterals.

Implementation

The predicates and directives here presented have been implemented as part of the SortalGI shape grammar interpreter [1]. The SortalGI interpreter has been implemented in Python and is accessible, among others, as a Rhino/Grasshopper plug-in (Stouffs 2018a). The plug-in defines both a rule object and a shape object. A Shape component creates a shape object from Rhino geometry (points, lines, polylines, arcs, quadratic Bezier curves, (flat) surfaces); a dShape component additionally accepts any number of descriptions. Additional components allow labels (or descriptions) to be attached to Rhino points, lines and surfaces, before providing these as input to the Shape or dShape component.

A Rule component constructs a non-parametric rule object from a left-hand-side and a right-hand-side shape object; while a pRule component takes two shape objects to create a parametric-associative shape rule. The pRule component also accepts predicates and directives. For each predicate and directive, a component exists to create the predicate/directive in its proper textual format. In addition, a number of other components are available to assist in the construction of descriptions, or parts thereof.

Finally, rules can be applied using a variety of components. An Apply component results in a single rule application; an ApplyAll component yields the output from all possible rule applications; while an ApplyAllTogether component combines the output from all possible rule applications into a single shape object. A Derive component applies a list of rule objects in sequence. Finally, a Matches component does not actually apply the given rule but, instead yields the matching shapes to the left-hand-side of the rule. As such, the Matches component can be used to search for a given shape. Fig. 4 shows the extrusion rules in Fig. 3 implemented in Grasshopper using the SortalGI plug-in. Note that additional rules are used to create plane segments and to determine the height of the extrusion in terms of the desired Gross Floor Area. Note also that the dimensions are not representative and the corresponding floor-to-floor height is assumed to be 0.3.

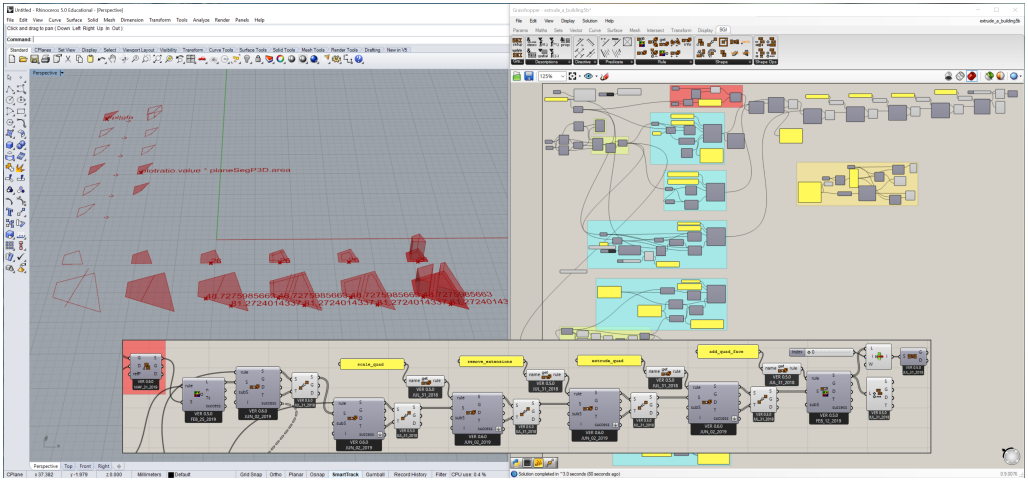

Figure 4
Six rules to scale and extrude quadrilaterals. The scaling adopts a fixed distance of -0.1 on all sides. The extrusion length is based on the ratio between the area of the scaled quadrilateral and the original quadrilateral, mimicking the calculation of a target Gross Floor Area based on a fixed plot ratio and determining a building height from a fixed floor-to-floor height, here 0.3. Note that the dimensions are not representative. The left part shows the visualization of the results in Rhino, the right part shows the entire Grasshopper model and the inset clarifies the sequence of rule applications.

CONCLUSION

We demonstrated the use of predicates and directives in the conception of parametric-associative shape rules in order to address seemingly simple requests that might actually be hard to express otherwise. The selected predicates and directives were partly inspired by Liew's (2004) work on descriptors, although the result here presented draws as much on use cases, observations of user actions and user feedback to provide us with insights into what kind of functionality is beneficial and desirable. We intend to continue this process and adjust and extend the current functionality where appropriate. Our ultimate objective is to make a shape grammar interpreter available that is usable in design practice.

ACKNOWLEDGMENTS

This work received partial funding support from Singapore MOE's AcRF start-up grant, WBS R-295-000-129-133. I would like to thank Bui Do Phuong Tung for his development work on the SortalGI shape grammar interpreter, Bianchi Dy for her development work on a previous version of the SortalGI Grasshopper plug-in, and Dan Hou for her feedback on the use of predicates and directives in her development of a shape grammar.

REFERENCES

Beirão, J, Duarte, J and Stouffs, R 2009 'Grammars of designs and grammars for designing – grammar based patterns for urban design', *Proceedings of CAAD Futures 2009*, Montreal

Downing, F and Flemming, U 1981, 'The bungalows of Buffalo', *Environment and Planning B: Planning and Design*, 8(3), pp. 269-293

Grasl, T and Economou, A 2013, 'From topologies to shapes: parametric shape grammars implemented by graphs', *Environment and Planning B: Planning and Design*, 40(5), pp. 905-922

Liew, H 2004, *SGML: a meta-language for shape grammar*, Ph.D. Thesis, MIT

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

Stiny, G 1980, 'Introduction to shape and shape grammars', *Environment and Planning B: Planning and Design*, 7, pp. 343-351

Stiny, G and Gips, J 1972, 'Shape grammars and the generative specification of painting and sculpture', *Information Processing*, 71, pp. 1460-1465

Stiny, G and Mitchell, WJ 1978, 'The Palladian grammar', *Environment and Planning B: Planning and Design*, 5(1), pp. 5-18

Stouffs, R 2018a 'Where associative and rule-based approaches meet: a shape grammar plug-in for Grasshopper', *Proceedings of CAADRIA 2018*, Vol. 2, Beijing, pp. 453-462
Stouffs, R 2018b, 'Description grammars: a general notation', *Environment and Planning B: Urban Analytics and City Science*, 45(1), pp. 106-123
Stouffs, R 2018c, 'Description grammars: precedents revisited', *Environment and Planning B: Urban Analytics and City Science*, 45(1), pp. 124-144
Stouffs, R 2019 'Shape rule types and spatial search', *Proceedings of CAAD Futures 2019*, Daejeon, South Korea
Woodbury, R 2016, 'An introduction to shape schema grammars', *Environment and Planning B: Planning and Design*, 43(1), pp. 152-183
Wortmann, T 2013, *Representing shapes as graphs*, Master's Thesis, MIT
Wortmann, T and Stouffs, R 2018, 'Algorithmic complexity of shape grammar implementation', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 32(2), pp. 138-146
Çagdas, G 1996, 'A shape grammar: the language of traditional Turkish houses', *Environment and Planning B: Planning and Design*, 23(4), pp. 443-464
[1] http://www.sortal.org/