CREATING PROCEDURAL WINDOW BUILDING BLOCKS USING THE GENERATIVE FACT LABELING METHOD

W. Thaller[a], R. Zmugg[a], U. Krispel[a], M. Posch[a], S. Havemann[a], D.W. Fellner [a,b]

[a] Institute of ComputerGraphics & KnowledgeVisualization (CGV), TU Graz, Austria
[b] TU Darmstadt & Fraunhofer IGD, Germany

KEY WORDS: Procedural Modeling, Neo-Classical Architecture, 3D-Reconstruction, Shape Grammars, Fact Labeling Method

ABSTRACT:

The generative surface reconstruction problem can be stated like this: Given a finite collection of 3D shapes, create a small set of functions that can be combined to generate the given shapes procedurally. We propose generative fact labeling (GFL) as an attempt to organize the iterative process of shape analysis and shape synthesis in a systematic way. We present our results for the reconstruction of complex windows of neo-classical buildings in Graz, followed by a critical discussion of the limitations of the approach.

Figure 1: Reasons for the complexity of window modeling. Intricacies of facade composition (left), vertical coherence (middle), and horizontal coherence (right) between ajacent windows.

1 INTRODUCTION

Most digital urban reconstructions today suffer from bad windows. There are two main sources of inaccuracy: Either the window is well modeled but does not match the original (because it is selected from a set of pre-modeled assets), or the window matches but is badly modeled (window texture). How can this situation be improved? We opt for geometric reasoning.

Windows are among the most salient features of façades. In most classical styles of architecture a window is not just a rectangular hole in a wall, but rather a combination of different inter-related design elements. They may derive from a long-standing architectural tradition. Thus, when creating a 3D model of a façade, a substantial part of the effort will be spent on modeling the windows and the decorative elements that go with them. A library of common pre-modeled windows can be used only for superficial reconstructions. When more accuracy is required, windows from an asset library can at most be used as a starting point for further manual modeling, or they must be camouflaged by photo-texturing. Windows in different buildings are often similar but hardly ever identical. We must better understand this phenomenon (Figures 1, 2) to produce more accurate reconstructions.

In this paper we propose a methodological approach to deal with situations where a large number of highly structured, similar but not identical shapes must be captured. Our *generative fact labeling* (GFL) method has three phases:

- **Analysis phase**: We have gathered a collection of 150 photographs of complex windows. We have structured them into elements by assigning *fact labels* (see Section 3).
- **Synthesis phase**: We have produced a library of combineable *procedural assets* corresponding to the elements identified in the analysis phase (see Section 4).
- **Verification phase**: We have 3D-reconstructed several well-chosen windows from this collection in order to assess the usefulness of our procedural library (see Section 5).

We discuss strengths, weaknesses and limitations in Section 6.

1.1 Contribution

We introduce the mentioned concepts (fact label, attribute, element, procedural asset, exemplar) as part of our generative fact labeling (GFL) method, a simple conceptual framework to deal with families of complex structured shapes. The goal of this method is *generative shape reconstruction*, i.e., to produce a library of functions that allow not only reproducing the limited number of given exemplars, but also the *design space* that is spanned by them. Whenever factoring shapes into procedural elements or components, one must be aware that this factorization is only an interpretation (speculation); there is no such thing as the "best" procedural description. Elegance is related to simplicity, but one can never be sure, e.g., to have found the shortest procedural description[1]. Our method is a guideline how to find at least a reasonable procedural explanation of a complex shape class.

1.2 Benefit

Procedural models have striking advantages over other types of 3D models (compact, editable, re-usable, scalable, meaningful semantics), especially over 3D scans (sampling approach). Generating 3D models by varying a few high-level parameters is nice, but it can be difficult to determine the parameters of a given real-world shape (procedural shape fitting). Even more challenging, and far from solved, is the aforementioned problem to determine a suitable set of procedures for a given set of shape examplars (also called inverse procedural modeling). The use case in this paper shows how to approach such a complex problem.

The second benefit is that we invite all knowledgeable specialists in the field of architecture to refine and specialize our imperfect taxonomy (discussion in Section 6). In contrast to other taxonomies with mainly academic value, the purpose of ours is to actually reproduce the shapes, i.e., it is a *generative* taxonomy.

[1] The Kolmogorov complexity $\mathrm{KC}(b)$ of a bit sequence b, the length of the shortest computer program that (re-)produces b, is not computable.

Figure 2: Window exemplars. The full set contains 150 images of windows from neo-classical buildings erected in Graz, Austria, in 1860-1890 (*Gründerzeit*). The complexity and visual dominance of the windows pose challenges to any digital urban reconstruction.

2 RELATED WORK

Digital reconstruction of buildings and monuments was a prominent topic throughout the years, and generative approaches on architecture are a hype today. Already the first shape grammars (Stiny and Gips, 1972, Stiny, 1980) were very useful for understanding the patterns of classical architecture. Hierarchical structures, such as façade layouts, are an ideal use case for shape grammars; but it turns out that their value for windows is only limited.

One of the first applications of shape grammars not just to understand, but also to generate complex architecture was (Wonka et al., 2003). Recent approaches on shape grammars for procedural modeling of architecture (Müller et al., 2006, Hohmann et al., 2010, Krecklau et al., 2010) usually focus on scripting as the main method to achieve their stunning results. As an alternative we have presented in (Zmugg et al., 2012) an approach where scripted procedural assets and shape functions can be applied and assembled interactively in order to reconstruct complicated architecture (e.g. the façade of the Louvre in Paris). The system presented in this paper follows the same approach. The result of the modeling process can be described procedurally, but not as a box grammar since we can perform more general shape operations.

For the analysis phase in our method (see Section 3) it is useful to study the appropriate literature in the field of architecture; we chose (Chitham, 2005, Mitchell, 1992, Schulze, 2008, Davies and Jokiniemi, 2008). These books provide comprehensive information about possible window element configurations, naming of the individual parts, and how they were composed. A very important reference, in particular for windows in Graz (Austria), is the work of August Ortwein (Ortwein and Scheffers, 1893). His work on the German renaissance, consisting of nine volumes, is still used as seminal compendium in building restoration due to the high accuracy in the description of details. Ortwein had much influence in Graz during the 19th century; he designed buildings in the famous ”Sporgasse” as well as several churches.

Our interactive 3D modeling approach for the synthesis phase (Section 5) is based on (Thaller et al., 2011). All 3D-models shown in this paper are internally represented as a collection of convex polyhedra. They support modeling operations that are very useful for architecture, in particular cut operations and CSG. These are the low level shape operations that were used to realize the procedural window elements presented in Section 4.

3 WINDOW ANALYSIS

In the beginning we are confronted with an unordered set of about 150 *exemplars* of complex windows, a selection of which is depicted in Figure 2. The question now is, how do we synthesize a function library that reproduces these windows? As outlined before, the first step is the analysis described in the following.

3.1 The Generative Fact Labeling Method

The process of generative shape reconstruction starts generally with a finite collection of exemplars, which are undisputable *facts*. Each exemplar is then associated with a number (set) of *observations*, which are (human) interpretations of the facts. Observations from different exemplars are then grouped together, thereby exploiting the structure of the observations. A first observation could be that every window consists of a hole in a wall, and that this hole can have different shapes. Hole shapes are mutually exclusive, a hole is either round or rectangular, but not both. A set of mutually exclusive observations leads to a set of alternatives which are grouped together in a *label group*. A label group is assigned a *group label*, for simplicity we use A, B, C, etc. Every alternative is enumerated, leading to labels A1, A2, A3 etc. for the label group A. We call the ensemble of observations, labels, and label groups a *classification* of the exemplars.

Unfortunately, we cannot assume that the set of labels in each group, nor the set of groups, will ever be exhaustive. We follow the *open world assumption* that our set of exemplars may always grow. Therefore we add two special labels for each group, namely *not applicable* (A-), and *other* (A*) indicating that “something A-ish is there” but none of the available alternatives apply.

The fact labeling process typically proceeds in a coarse-to-fine manner. We first determine rough structural units (window layout) and then look at the parts in order to differentiate alternatives more locally. This produces new label groups, which we still keep in a flat list. Note that although we seek to decouple the labels we do not assume any hierarchy in the labels; the relation between the label groups (and the labels) can be quite complex, in fact.

The fact labeling approach as described so far is very generic and applies to *any* effort to create a classification scheme. The distinguishing property of *generative* fact labeling is the procedural view: We seek to group our observations in such a way

Figure 3: Various fact labels applied to five example windows. Refer to Table 1 and to Section 3.2 for explanations of the labels.

that they can be mapped to procedures that (re-)produce the observed shapes (Sections 4, 5). This allows introducing a metric on the produced classification: The better the procedures (software engineering) and the better the results, the better is the classification. This distinguishes our classifications from those produced for academic reasons, e.g., in art history or history of architecture.

3.2 Generative fact labeling applied to windows

Consulting the relevant architectural literature was helpful in the communication with experts who helped us making more relevant observations. Broad prior knowledge is not mandatory, however, and may even be distracting since the GFL method focuses exclusively on the geometric aspects of architecture.

We started with more or less obvious observations such as “this window has a sill” or “next to the window are pilasters”. After some iterative refinement we arrived at the fact labels and label groups shown in Table 1; labels *not applicable* (-) and *other* (*) are omitted. Figure 3 shows some fact labels on example windows. Note that each label group, i.e. each line in the table, can be interpreted as a question that can be asked about a window:

A. count How many windows are there?
B. side Is the window framed at the side by columns or pilasters? Alternatively, the decoration above the windows can be symbolically supported by brackets at the *side* of the window.
C. sill Is there a sill below the window, or is there a sill with additional decorations below it?
D. above Is there a cornice above the window, or a pediment, or a combination of the two?
E. frieze Is there additional space, a frieze, or an architrave below that cornice or pediment (Figure 4(a))?
F. layout The interaction between pillars at the side and the frieze or architrave between the cornice and the opening.
G. shape The shape of the window opening itself.
H. frame Is there is an added frame around the opening? Does that frame have a visible keystone at the top?
I. pediment The basic shape of the pediment.
J. pediment2 A systematic variation of pediment shape.
K. pediment3 Is there a open pediment, or a keystone?
L. cornice Is the cornice broken in the center?
M. below-cornice Are there brackets that symbolically support the cornice? This does not include the ”side” brackets (B2).
N. below-sill Are there brackets that symbolically support the window sill?

(a) (b) (c) (d)

Figure 4: Pilasters, brackets, friezes and architraves, and their possible arrangements. A window with no side decoration but with a frieze (yellow) is shown in (a). Pilasters (red) bypass the frieze in (b). Pilasters can be reduced to smaller brackets that support the cornice (c). Finally, pilasters end at the top of the window opening and support an architrave (d).

Some of these questions depend directly on the answers to other questions; if a window has been labeled as not having a cornice or pediment (D-), all questions about the shape of the pediment and cornice will have to be answered with *not applicable* as well. Note that this hierarchical relation is not imposed from the outside; such dependencies emerge from the observations. Dependencies between label provide only a hint for a possible hierarchical structure to be used in the generative reconstruction later on; however, there need not be a simple one-to-one correspondence since the fact labels do *not* form a strict hierarchy naturally.

4 WINDOW ELEMENTS

The window analysis from Section 3 must eventually lead to the synthesis of three-dimensional window shapes described in Section 5. The bridge in between is actually a software engineering task, namely to factor the shapes to be produced into re-usable procedures. We have identified, for example, in many parts the necessity to apply mouldings; technically, this is a profile sweep along certain edges of a shape. Other examples of re-usable patterns that can be mapped to functions are circular partitions, circle segments, and linear repetitions. These are the functions that are then used to quickly obtain scripted building blocks for interactive procedural modeling, in this case window elements.

A selection of the realized procedural window elements is illustrated in Figure 5, and some of the elements are described in some detail in the following; an index like (a1) refers to an image in the table of images in this Figure.

Cornice and Pediments We offer two basic shapes of pediments - triangular and round - with the possibility of adding some customizations, including extended end parts (a1), open top sections (h1), the addition of a keystone (g1), or stepped designs (e1). To realize open pediments an additional CSG-difference operation is used. Keystones are inserted by extruding a part of the pediment to the front as well as up and down. Stepped designs for pediments, cornices and keystones can be achieved by a separate extrusion step. Broken cornices (a1, c1) are supported besides the regular ones. Mouldings can be applied to further enhance the appearance of pediment and cornice (see below).

Window Shapes and Crossbars The most common window shapes are supported in our system. These include the common rectangular shape (d2), round shape (c2), as well as several arches; among these are the round arch (e2), segmental arch (a2) and lancet arch (b2).

Crossbar assets (f2-h2) are realized independently of the shape of the window itself. The crossbar rules adapt automatically to the (convex) space that is provided for them.

L.	LABEL GROUP	1	X	2	X	3	X
A	**count**	single window		double window		triple window	
B	**side**	pilaster	f3	big bracket			
C	**sill**	simple sill	a4	sill and decoration below			
D	**above**	cornice	e1	pediment	a1	cornice and pediment	b1
E	**frieze**	frieze/architrave					
F	**layout**	pilasters/brackets beside frieze		pilasters end below architrave		crossing	
G	**shape**	rectangular opening	d2	round arch	e2	segmental arch	a2
H	**frame**	frame		frame with keystone			
I	**pediment**	triangle pediment	b1	round arch pediment		segmental arch pediment	
J	**pediment2**	horizontal cornice at the sides	a1				
K	**pediment3**	open	h1	keystone	g1	stepped	
L	**cornice**	broken	c1	stepped	e1		
M	**below-cornice**	brackets at side	a3	many brackets	c3	centered brackets	b2
N	**below-sill**	brackets at side	c4	balustrade			

Table 1: This labeling table is the result of the analysis phase (Section 3). Every label, e.g. A1, is associated with a set of observations on the given facts (exemplars). Entries in the X-columns refer to the table of images of procedural assets in Figure 5.

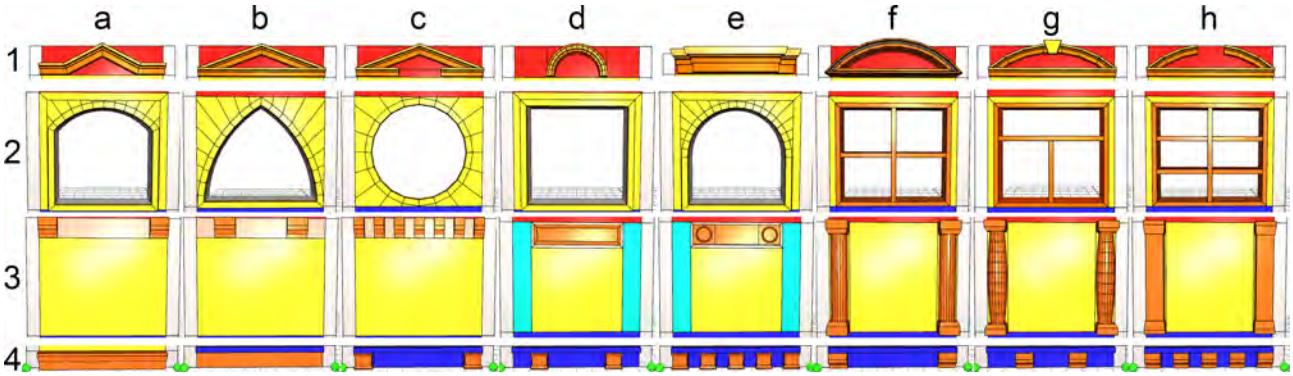

Figure 5: Samples of procedural assets from the window part library: Cornices and pediments (first row, a1-h1), window shapes with borders and crossbars (second row, a2-h2), friezes, panels and pilaster (third row, a3-h3), and window sill and decorations (bottom row, a4-h4). All these assets adapt their size to the space they are inserted to.

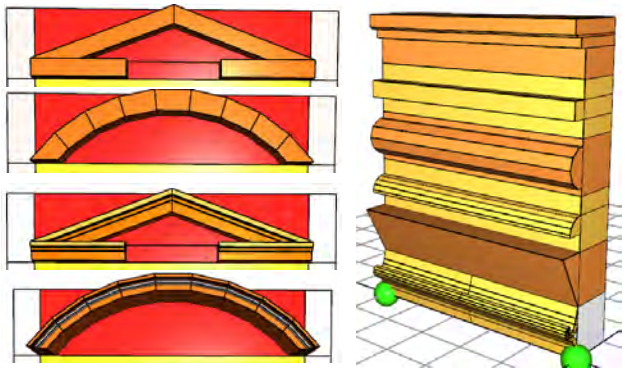

Figure 6: Different mouldings on assets created by the frame operation. Pediment and cornice are generated in a generic way (upper left), the moulding is then applied using a specific profile (lower left). The differnt mouldings (right) are defined independently from the assets that they are applied to.

Frames and Mouldings Our system provides a frame operation that generates a frame for arbitrary convex shapes. This operation is used for frames of the window pane (a2-e2) and to generate the shape of the pediment (a1-d1, f1-h1). Additionally we can create geometry along a poly-line to achieve certain pediment shapes (a1).

The output of this operations can be refined further by applying mouldings (see Figure 6). Moudlings apply an extrusion profile along the course of the frame. This extrusion profile for the moulding is generated independently from the asset it is applied to. The cornice (e1), the architrave or the window sill (a4) can also be refined by adding mouldings to them.

Pilaster and Brackets We have a round (f3-g3) and a rectangular pilaster (h3), equipped only with a basic capital and pedestal. However, their usage can be very versatile - from being the essential part of the balustrade, to various uses in friezes and other decorative elements.

In most cases, pilasters can be exchanged with brackets. Since their appearance can be quite diverse, we only provide a crude approximation to give the basic idea of the real shape. Especially friezes are often decorated with a multitude of brackets (a3-c3). Two fundamental types can be identified, pillar-shaped brackets (c4-e4) for which we use our pilaster assets, and brackets with a slanted bottom part (a3-c3, f4-h4).

Frieze and Architrave Several decorative elements are placed symmetrically in the frieze (a3-e3). We provide supporting elements like brackets and decorative panels (d3-e3). These panel operations use the window shape and frame operations together with extrude and bevel operations to obtain their look. Brackets and panels are placed with a repeat operation that subdivides the available space to place repetitive elements (a3-c3).

Architraves are often just decorated with a moulding that runs across the width of the window. All the supported profiles for the mouldings can be used to decorate the architrave.

Window Sill and Decoration beneath A window sill is an extruded part that has often a moulding applied to it (a4). The space beneath the window sill can be decorated like a frieze (c4-h4). Brackets then support the sill instead of the cornice. This space can also feature a balustrade. The same pillars as described before, just smaller, appear equally spaced below the sill. The repeat operation ensures that the number of pillars is adjusted according to the available space.

5 WINDOW SYNTHESIS

To assess the usefulness of our approach we have reconstructed a selection of window exemplars by combing the procedural assets described in the previous section.

5.1 Interactive step-by-step example

A step-by-step illustration of a window reconstruction is shown in Figure 7. The modeling process mostly follows the labeling process. It can be understood to some extent as a shape grammar since one part is selected and then replaced by one or more new parts. However, with a shape grammar it is not possible to realize crossings of vertical and horizontal structures and to snap to other parts in different branches of the hierarchy, as we sometimes do.

Modeling a window proceeds by combining discrete assets; even when this is done in a graphical user interface, it can still be seen as interactive scripting. A script is generated in background that will reproduce the model when executed. The main difference to conventional interactive free-form modeling is that the mouse is in fact only used as a selection tool in our system. Therefore, the recorded script does not have to contain any tracked mouse coordinates, only references to high-level assets.

The first step in the window creation process is to decide what kind of window is to be created. Then different layouts are chosen for the window elements, for example the window part is partitioned into a center and two pilaster parts to support the frieze. All measurements and sizes of specific parts can be modified by parameters, i.e., the vertical alignment of pilaster and brackets can be adjusted manually. Asset insertion steps can be executed in any order since each asset operates on a single selected part.

Although the number of different procedural assets appears to be fairly limited, quite a variety of shapes can be achieved since elements can be combined, nested and repeated sometimes in surprising ways. The versatility of the procedural modeling tools encourages us to believe that it will indeed be possible to achieve eventually a good coverage of the architectural variety of the given examplars by further extending the toolset.

5.2 Example reconstructions

Figure 8 shows reconstructions of five different windows from exemplars, all with different layouts. Some of the decorations where only approximated by manually placing variations of other assets at certain positions. By manually adjusting certain dimensions and exchanging a few assets several other (similar) windows can be realized quickly. The benefit of our procedural approach is that the parts can adapt flexibly to a wide variety of surroundings. It is therefore in most cases much more efficient to adapt an existing window than to create a window from scratch; so the re-use of models is encouraged by the system.

Figure 9 shows some variations of the two windows in red and blue frames from Figure 8. Assets can be nested and combined to realize also more complicated configurations (third variation of the blue window). In terms of operations these windows are 'close', which suggests that *procedural distance* could be a useful shape similarity measure.

6 DISCUSSION

We discuss the limitations of our method on three levels. We first present some challenging window exemplars that are hard to synthesize with our current procedural library, then some shortcomings of the classification, and then elaborate on whether or not this invalidates the fact labeling approach. Finally, we discuss the relation of our approach to shape grammars.

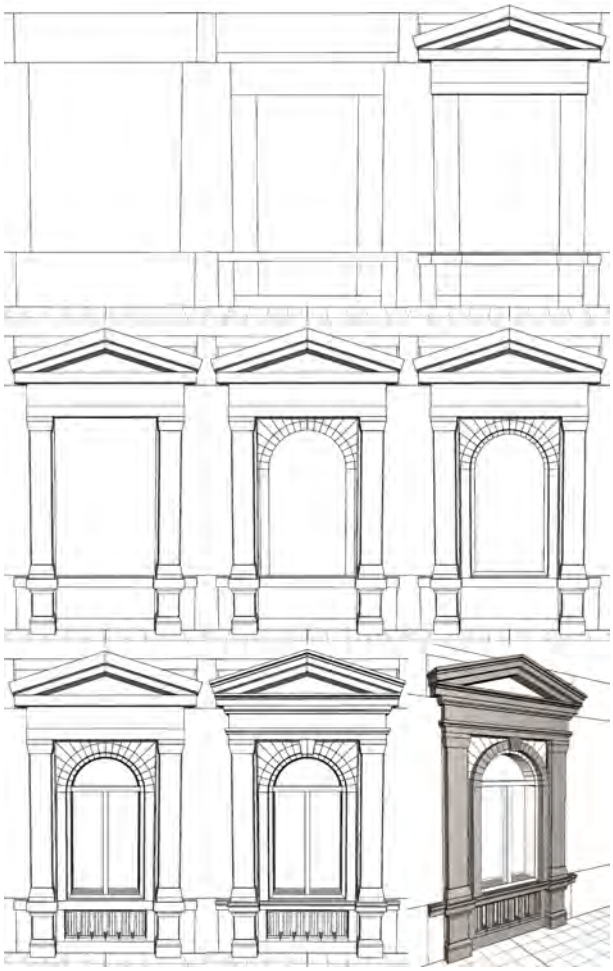

Figure 7: Interactive window modeling using procedural assets. First, the layouts for the individual sections are chosen (row 1, images (a), (b)), in this case a single window with decoration above and below the hole. The order in the asset insertion steps (row 1(c) to row 3(a)) is not important since assets can be inserted independently from each other. The last step is to apply detail mouldings to the window elements and to add the keystone (3(b)).

6.1 Challenging window exemplars

We have grouped the problematic windows (Fig. 10) into difficult but feasible (left) and more fundamental problems (right). Windows in the left group exhibit a feature that is not yet supported but has a well-defined place in the classification of Table 1. They can be synthesized when specific assets are added (*missing feature issue*), e.g., new opening shapes for windows (c), (d) and (j) and new frame decoration styles for window (b). The intricate decorations around windows (a) and (i) are out of scope since we have limited ourselves to convex partitions; general ornamentary requires a different shape modeling approach. The work-around is to allow importing decorations as non-procedural pre-modeled 3D assets like in box grammar systems (Müller et al., 2006).

The windows to the right in Figure 10 reveal problems of a slightly more fundamental nature. Window (e) exhibits delicate tracery in the top, leading to bar and hole shapes that require specific geometric constructions which are not found elsewhere; the same applies to window (k) (*general construction issue*). Window (e) also features a ledge that runs along the façade and crosses the window on a horizontal bar, interacting with various structures of the window along the way (*cross feature issue*). Even more drastic is the *cross hierarchy issue*; the seemingly innocent example is the top

Figure 8: Synthesis of example windows. The first exercise was to reconstruct five window exemplars with sufficiently different layouts and decorations. The second task was the variation of the two windows in the red and blue frames (see Figure 9).

Figure 9: Variations of the two windows in red and blue frames from Figure 8. The variations are derived only by replacing assets and manually adjusting proportions. Assets can be combined to realize also more challenging configurations (right group, third variation).

Figure 10: Windows that cannot be handled properly by our current system. The left group of windows could be handled by special-purpose assets, the right group reveals more fundamental issues and problems. See Section 6.1 for an explanation.

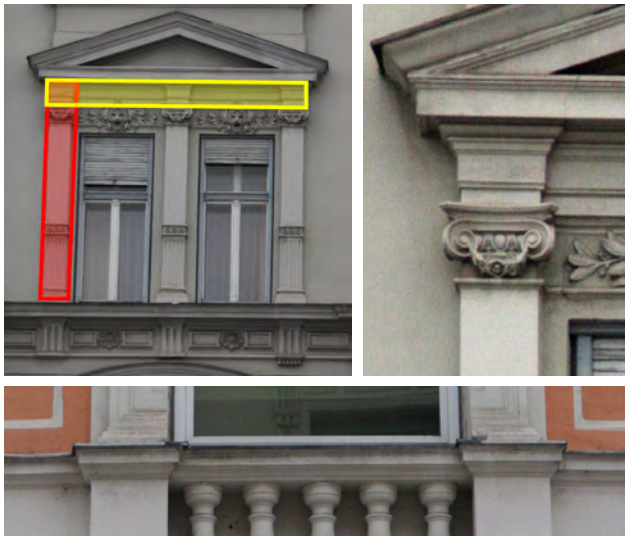

Figure 11: Examples of a window with overlapping structures. The pilaster (red) overlaps the architrave (yellow). In the overlapping area, the moulding of the architrave runs across a continuation of the capital of the pilaster. In the window at the bottom, a ledge that runs across the whole façade also serves the purpose of a window sill.

of window (m) where the keystone not just protrudes downwards and upwards, but actually bridges and breaks the circular profile, then the horizontal frame, and finally becomes part of the frieze in the top. Window (g) violates one of our implicit assumptions, namely that window decorations are applied to planar façades. The pediment and the structures at the sides of this circular window – which are vaguely reminiscent of ionic columns – are part of the three-dimensional structure of this particular building (*planarity issue*).

The holes of the multi-windows (h) and (n) are so tightly coupled that the window classification is ambiguous. Apparently a larger hole was partitioned by bars, so it makes no sense to treat the sub-windows separately; but the bars are also so prominent the windows are clearly separated. We have found so many examples of this *ambiguity issue* that we suspect that ambiguities are introduced intentionally by architects. The issue is similar to the *repurpose issue* of window (l), where the pediment is not on top of the window, but the window is *inside* a triangle pediment.

6.2 Classification problems

The fact labeling approach allows making interdependencies explicit by assigning additional labels. We can, for instance, easily state that in a given window, there are (i) pilasters that support the cornice and pediment, and (ii) there is an architrave below this cornice. We have then introduced an extra label group F that describes how these two structures interact. The window in Figure 11 is labeled D3, as it has a cornice and pediment, and E1 because it has an architrave and frieze. Additionally, it receives the label F3 because both structures overlap.

To generate geometry for this fact label F3, the assets must be able to deal with the interaction of the architrave and the pilaster. However, this interaction depends on the specific assets that are used for the pilaster and the architrave; in the worst case, one asset has to be defined for every possible combination of a pilaster asset and an architrave/frieze asset. This is clearly not scalable.

Another example of problematic overlaps are ledges running horizontally over the whole façade. Sometimes they do not interfere with the windows, for example between storeys; in other cases they are interrupted by a window. Sometimes, such a ledge is re-used as a window sill. The most difficult case arises when the ledge is modified slightly to accommodate the role of a window sill. The corresponding procedural asset for the window sill would have to describe a window sill created by modifying an existing ledge in a certain way. By limiting our focus to individual windows and their immediate surroundings, we have side-stepped this *feature in disguise* problem for now.

6.3 Feasibility of the GFL approach

We have mentioned now so many issues and problems that we need to examine the feasibility of the overall approach. The question is, will the GFL method ever converge, and is there a realistic chance to obtain in the end a reasonably small procedural function library that can synthesize for *all* examplars a 3D-model with satisfactory detail resolution?

GFL produces a flat list of labels without any hierarchy or other additional structure because the expectation is that such a structure shall in fact emerge during the exercise. However, since the method makes no a priori assumptions whatsoever about any potential structure or relation between the labels, the resulting labels may be arbitrarily unorganized. A great danger of the GFL method is over-specialization when more detailed analysis of the exemplars produces an ever-growing number of observations, labels and label groups. The only effective counter-measure is continuous re-iteration in order to identify similarities between labels that can be merged and mapped to the same generative procedure. We call this inductive reasoning process *label reduction*.

To illustrate the label reduction process, consider for example the labels M (below-cornice) and N (below-sill) from Table 1. They reveal striking similarities when comparing images (a3)-(c3) and (c4)-(h4) in Figure 5. Consequently, both labels can certainly share most of their procedures, and maybe even merge.

The main difficulty in label reduction is to 'factor out' structural ambiguities in a correct way. We have realized that it is in fact a common situation that some architectural rule A allows or requires an element X, and another architectural rule B allows or requires a different element Y in approximately the same place. One way for an architect to merge A and B is to make X resemble Y in an creative way; and this is very difficult to schematize and to map to common procedures. We suspect that this is an important reason for the abundant variety of windows shapes.

6.4 Relation to Shape Grammars

It is interesting to note that "overdoing" the label reduction results in a procedural library that is indeed very simple, but captures shape interdependencies only insufficiently. The discussion in the previous section has shown that although the list of label groups A,B,C,... produced by the GFL method remains flat, the relation between the individual labels can in fact become very intricate and complicated. In order to reduce this complexity we have intuitively taken the approach to decouple as much as possible the different labels. Label dependency is minimal when every label depends only on a single other label. This leads to a linear label refinement process and thus, eventually to a (context-free) parametric shape grammar.

We had to realize that the result of our experiment is a procedural function library that is effectively such a shape grammar. The implications are revealed by revisiting the step-by-step example from Figure 7: The vertical separation (image 1(a)) nicely decouples the three window parts (top, mid, bottom) that can then

be independently refined into sub- and sub-sub-parts without any side effects; but note that the window sill and the column bases are contained in the bottom part, separated from the column tops. Although the resulting columns look like coherent vertical structures (image 3(c)), they are composed of three independent parts; and the middle part, the window sill, is in fact even a separating horizontal structure cutting through both columns.

This model captures interdependencies insufficently because the vertical integrity of the column cannot be preserved when the three parts forming each column are varied horizontally. Consequently, the apparent coherence is only a coincidence.

7 CONCLUSION AND FUTURE WORK

We have presented a fully manual method to obtain a library of functions that is capable of re-generating procedurally a given collection of non-trivial input shapes, as well as many variations of these shapes. We propose an iterative process of grouping, labeling and re-grouping of shape observations, followed by reflection and inductive reasoning to discover structures and similarities that can then be exploited in the software development process to produce a concise but powerful set of shape generating functions. We have integrated these functions as 3D modeling tools in a simple graphical user interface, so that the actual shape composition, which requires no programming but only straightforward function composition, can be carried out interactively.

As confirmed by our experiment with 150 windows of neo-classical buildings, several passes of this potentially very tedious process are necessary in order to obtain satisfactory results. We acknowledge the fact that procedural 3D modeling ultimately implies shape programming, and our generative fact labeling method is a first attempt to guide the development process in a systematic way. Since the method is extremely generic, it can be applied in any domain of man-made shape, from castles over buildings and furniture to engineering and automotive design. Our conjecture is that in fact *any* attempt to produce a library of shape generating functions for a certain domain must apply this method, or at least a variant of it. We admit, though, that this generality is also a weakness since although the method is effective, much sophisitication is required for it to be also efficient.

We see mainly three avenues for further research. First, it is interesting to note that the labeling method gives rise to a very straightforward string encoding of the exemplars. Mentioning only relevant labels, the windows from Figure 3 can be encoded as follows using our very simple labeling from Table 1:

```
A1_C2_D1_E1_G2_H1_N2
A1_B1_C2_D1_E1_F2_G2_H2_N1
A1_B2_C2_D3_E1_F1_G1_I1
A1_C2_D3_E1_G1_H2_I3_K2_M1_N*
A2_B1_C2_D3_E1_F2_G1_H1_I1_J1_N1
```

The great value of such a shape *formalization* is that it provides an interface to bridge the gap between shape analysis and shape synthesis. Although this is only a very rough encoding, it could already be useful, e.g., as feature vectors for training shape analysis and machine learning algorithms. And even without any dimensions, more faithful window geometry can be produced already from this encoding. An obvious area for further research is how to extend this encoding when more structure has been found.

The second research question is whether the tedious label production and reduction process can be supported with algorithms, e.g., to check an observation for consistency with the exemplars ("how many windows have a triangle pediment?"). Such checks are obviously required very often; but manual checking is very tedious, which limits both label production and reduction. There are many interesting quantifiable measures to assess observation and label quality, such as relevance, coverage, conflicts, etc.

The third research challenge is whether this method might eventually lead, after experiments in various different shape domains, to a more general understanding of *shape ontologies*. Our conjecture is that the observations on how shapes interact in one shape domain will not be fundamentally different from the observations in other domains. If this is true, then it should be possible to identify a few central concepts of a "*Conceptual Reference Model*" for shape, that can then be mapped to specific shape domains via subclassing (specialization). The proof that such an endeavour can eventually lead to success was given by CIDOC-CRM for the equally complex problem of creating a small vocabulary for encoding facts in cultural heritage (Crofts et al., 2005).

REFERENCES

Chitham, R., 2005. The Classical Orders of Architecture. second edn, Architectural Press.

Crofts, N., Doerr, M., Gill, T., Stead, S. and Stiff, M., 2005. Definition of the CIDOC Conceptual Reference Model. Version 4.2 edn, CIDOC Documentation Standards Working Group. (ISO 21127).

Davies, N. and Jokiniemi, E., 2008. Dictionary of Architecture and Building Construction. Architectural Press.

Hohmann, B., Havemann, S., Krispel, U. and Fellner, D., 2010. A gml shape grammar for semantically enriched 3d building models. Computers & Graphics 34(4), pp. 322 – 334.

Krecklau, L., Pavic, D. and Kobbelt, L., 2010. Generalized use of non-terminal symbols for procedural modeling. Computer Graphics Forum 29(8), pp. 2291–2303.

Mitchell, W.-J., 1992. The Logic of Architecture. The MIT Press.

Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L., 2006. Procedural modeling of buildings. ACM Trans. Graph. 25(3), pp. 614–623.

Ortwein, A. and Scheffers, A., 1893. Deutsche Renaissance Band 1,. 2nd edn, Seemann Verlag.

Schulze, J., 2008. Wie man um 1874 Fenster baute. In: Fenster im Baudenkmal, Lukas Verlag.

Stiny, G., 1980. Introduction to shape and shape grammars. Environment and Planning B 7(3), pp. 343–351.

Stiny, G. and Gips, J., 1972. Shape Grammars and the Generative Specification of Painting and Sculpture. In: C. V. Friedman (ed.), Information Processing '71, Amsterdam, pp. 1460–1465.

Thaller, W., Krispel, U., Havemann, S., Redi, I., Redi, A. and Fellner, D., 2011. Developing parametric building models - the GANDIS use case. In: F. Remondino and S. El-Hakim (eds), Proc. ISPRS Workshop 3D-ARCH 2011, ISPRS.

Wonka, P., Wimmer, M., Sillion, F. and Ribarsky, W., 2003. Instant architecture. ACM Trans. Graph. 22(3), pp. 669–677.

Zmugg, R., Krispel, U., Thaller, W., Havemann, S., Pszeida, M. and Fellner, D. W., 2012. A new approach for interactive procedural modelling in cultural heritage. In: Proc. Computer Applications & Quantitative Methods in Archaeology (CAA 2012).