Semantizing Complex 3D Scenes using Constrained Attribute Grammars

Alexandre Boulch, Simon Houllier, Renaud Marlet, Olivier Tournaire

Semantizing Complex 3D Scenes using Constrained Attribute Grammars

A. Boulch[1], S. Houllier[1], R. Marlet[1] and O. Tournaire[2]

[1]Université Paris-Est, LIGM (UMR CNRS), Center for Visual Computing, ENPC, F-77455 Marne-la-Vallée [2]CSTB, F-77447 Marne-la-Vallée

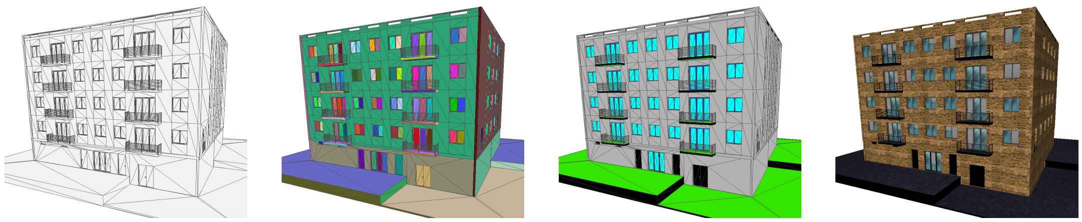

Figure 1: *Automatic analysis. Left to right: input CAD model (triangle soup), polygon reconstruction, semantization, rendering.*

Abstract

We propose a new approach to automatically semantize complex objects in a 3D scene. For this, we define an expressive formalism combining the power of both attribute grammars and constraint. It offers a practical conceptual interface, which is crucial to write large maintainable specifications. As recursion is inadequate to express large collections of items, we introduce maximal operators, that are essential to reduce the parsing search space. Given a grammar in this formalism and a 3D scene, we show how to automatically compute a shared parse forest of all interpretations — in practice, only a few, thanks to relevant constraints. We evaluate this technique for building model semantization using CAD model examples as well as photogrammetric and simulated LiDAR data.

Categories and Subject Descriptors (according to ACM CCS): I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—3D/stereo scene analysis I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Object hierarchies I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Object recognition I.5.4 [Pattern Recognition]: Applications—Computer vision

1. Introduction

We consider here the problem of semantizing complex objects in a 3D scene. The need for interpreting such scenes is an old problem but still largely unsolved. The recent availability of inexpensive 3D data has stimulated progress on this topic. It is now cheap and easy to make real world acquisition, e.g., using photogrammetry, time-of-flight cameras or active stereovision systems such as the Kinect. Purely synthetic 3D data has become mainstream too, with 3D modeling software such as SketchUp, making it possible for anybody to easily create 3D models.

Scene semantization has many uses. In large 3D repositories (such as Aim@Shape or Trimble 3D Warehouse), semantic information is needed to perform relevant and efficient queries. Applications include, e.g., virtual sets for games or movies, interior design, product design for e-manufacturing [ARSF09], etc. However, manually annotating geometry with semantics is a time-consuming and tedious task, which is error prone and expensive. Besides, the size of some repositories is such that a systematic manual treatment is unfeasible in practice. Automation is required.

Businesses such as the building industry are also rapidly moving toward relying heavily on high-level semantized 3D models for the whole lifecycle of their products, from sketch and design to construction, operation and dismantlement. These industries typically work from data acquired using devices such as laser scanners, whose expensive price has also greatly reduced lately. The stakes are high as semantized 3D models enable much quicker, safer and cheaper developments and decisions. However, while 3D models tend now to be created for new products, they are generally not available for existing ones. In particular, old buildings — i.e., by

Figure 2: *Automatic semantization of LcF building.*

far, most buildings — do not enjoy digital 3D information whereas they would benefit the most from it, e.g., to plan cost-effective renovation that achieves good thermal performance. Actually, as building rehabilitation becomes more complex due to the evolution of techniques and regulations, and as customers now require not only low-priced solutions but also performance guarantees, rules of thumb and cross-multiplication do not scale. Simulations based on complete, accurate and semantized data are needed.

The challenge is thus to efficiently produce 3D models that are not limited to geometry but that are also structured with respect to their semantics, e.g., building elements: floor, roof, wall, windows, stairs, etc. What we propose in this paper is a new technique to semantize an existing high-level geometry. We assume we are given a geometric model as a set of 3D polygons and we produce labels for these polygons identifying the semantic part of the model they belong to.

Interestingly, architects can also make use of such a semantizer after they sketch a building for presentation to bidders or customers, to get approval before producing detailed design and plans. Although CAD systems do offer rendering features, quality is often poor and inadequate for ambitious projects. Architects have to go to a graphic designer to obtain spectacular views of their proposal. For this, they have to communicate the model with information about the nature of building elements. They usually do so by assigning different colors to geometric components and telling the graphic designer how to interpret and replace them by relevant textures for quality rendering. We can do this labeling automatically (see Fig. 1). A similar issue arises at a large scale in the game industry. Level designers tend to focus on the geometry and structure of what they create, providing little information to graphic artists, that often have to "paint" each scenic element separately (architecture, objects, etc.) [CLS10].

Related work. There is a large amount of work regarding semantic segmentation, classification, matching, retrieval, fitting and reconstruction of 3D shapes. Most methods concerning complex objects rely on graph-based analysis. The graphs may originate from volume skeletons [BMMP03] or surface decomposition, either from "perfect" geometry [EMM03], point clouds [SWK07] or meshes [PSBM07]. They can be automatically learned from examples [TV08] or designed "by hand" [SWWK08]. Graph matching algorithms are then used to retrieve given objects in the scene, possibly allowing a few mismatches: maximal common subgraph, matching with edit distance [GXTL10], etc. Although these matching problems are NP-complete, basic algorithms and heuristics are usable if the graph is not too large.

However, we are interested here in objects, e.g., buildings, that are complex in the sense that (1) they have a high degree of compositionality, i.e., they can be broken down into a deep hierarchy of components, (2) the number of parts is unknown and potentially highly variable, and (3) the relations between the different parts can be sophisticated, not limited to adjacency and relative position or orientation. For this kind of objects, the relevant underlying structure seems to be a grammar: (1) each production rule represents an alternative decomposition level in the hierarchy, (2) recursive rules can express an arbitrary number of items, and (3) the grammar can express constraints to cope with complex relations between parts. Furthermore, grammars are naturally modular to some extent, enabling the writing and maintenance of large specifications, contrary to hard-coded approaches relying on weak, hard-coded domain knowledge [MMWVG12].

Shape grammars have been popular for the procedural *generation* of model instances [MWH*06], but hardly reversed to perform *analysis*: such uses are scarce, with simplified grammars (e.g., split grammars) and reduced to fronto-parallel image parsing, i.e., 2D [MZWVG07, RB07, TKS*11]. Generative models in 3D have little been considered for analysis and are then restricted to specific, hard-coded grammars such as roofs [HBS11] and Manhattan-world buildings [VAB10, VAB12]. Only very recently has procedural modeling been associated to multi-view 3D for parsing [STK*12]. However, it reduces to the incorporation of a depth term in a basically 2D energy; it does not compute 3D from parsing. Moreover, the search space is so huge that its requires a heavy optimization machinery, and it is not clear how it can scale to complete buildings and full 3D.

As opposed to the above top-down parsing techniques, bottom-up approaches have also been proposed, with interleaved top-down predictions of missing or occluded components [HZ09, WZ11]. However, they are currently restricted to images and compositions of basic primitives, e.g., projections of 3D rectangles or circles. A multi-view variant with a 3D grammar interpreter that hard-codes some operations and exhaustively checks shape composability (without constraint propagation) has been used for reconstructing Doric temples [MMWG11]. In 3D, a grammatical approach has been used to construct building shapes from airborne LiDAR data [TMT10], but the parsing is hard-coded, e.g., rules are applied in a specific order and specific structures are sought "outside the grammar" such as maximum spanning trees; it cannot be generalized to any set of grammar rules.

Our approach. We present here a purely bottom-up approach, although it can be complemented by top-down predictions to treat incomplete data (see Section 6). It relies on a constrained attribute grammar with geometry-specific predicates. Terminals are detected 3D primitives that are combined using production rules to construct higher-level nonterminals. This could normally lead to a combinatorial explosion. But one of the key to the practicality of our approach is the use of constraint propagation to reduce the search space incrementally as much as possible, in particular with the use of invertible predicates, which drastically reduce the involved domains as soon as part of their arguments are known. "Constrained" attribute grammars have been proposed in the past but in much more restricted settings, e.g., without constraint propagation and with a single greedy interpretation, to parse the layout of mathematical formulas [Pag98]. Our contributions are as follows:

- We propose a constrained attribute grammar formalism for specifying complex objects, including 3D objects.
- It provides a practical conceptual interface as opposed, e.g., to low-level graph node descriptions, which is arguably crucial to write large maintainable specifications.
- As complement to recursion, we introduce maximal operators, that are essential for search space reduction.
- Given a grammar in this formalism and a scene, we show how to efficiently compute a shared parse forest.
- We evaluate our approach both on synthetic and real data (CAD models, photogrammetry and simulated LiDAR).

Though we illustrate here building semantization, our technique is general and could be applied to other domains too.

The rest of the paper is organized as follows. Section 2 presents the formalism. Section 3 describes what a corresponding parse tree is. Section 4 explains how to efficiently compute a share parse forest. Section 5 shows experimental results with building grammars. Section 6 studies the advantages and perspectives of grammars and Section 7 concludes.

2. Constrained attribute grammar

We consider a scene containing objects that have a hierachical decomposition into parts having complex relations. This can be modeled using a constrained attribute grammar.

Basic grammar ingredients. We consider a *grammar* $G = (N,T,P,S)$ consisting of a set N of nonterminals, a set T of terminals disjoint from N, a set P of production rules and a set $S \subset N$ of start symbols. A *terminal* corresponds to a *geometric primitive* in the scene, e.g., polygon with holes, cylinder. A *nonterminal* corresponds to a complex *form* in the scene; it can be decomposed via G into other grammar elements. A *production rule* $r \in P$ is of the form $r = Y \to X_1,\ldots,X_k$ where the rule's left-hand side (LHS) is $Y \in N$ and the right-hand side (RHS) is $(X_i)_{1\le i\le k} \in (N \cup T)^*$, e.g.,

$$step \to riser, tread$$

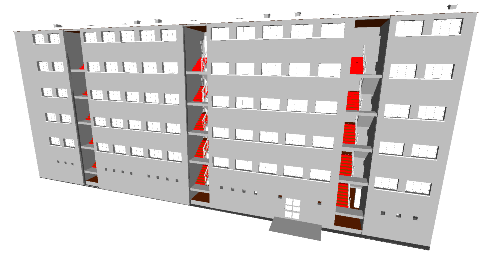

Figure 3: *Automatic stairs detection in LcC building.*

Some nonterminals may be introduced in a grammar as auxiliary constructs, that are meaningless for end users. We use *start symbols* to identify parses only of relevant objects.

We use the vertical bar for disjunction: $Y \to W \mid W'$ is equivalent to the two rules $Y \to W$ and $Y \to W'$. And $Y \to X_1, \mathsf{optional}\ X_2$ is equivalent to $Y \to X_1 \mid X_1, X_2$, although with a semantic nuance (see "Collections" below). This definition of a grammar is refined in the following.

Constraints. Contrary to 1D grammars and split grammars, a rule $r = Y \to X_1,\ldots,X_k$ only expresses dominance of Y over X_i, not precedence of X_i over X_{i+1}. To cope with 3D, complex association constraints between $X_1,\ldots,X_k$ are shifted into a separate *condition* C attached to the rule, which we note $r\ \langle C\rangle$. (See below for predicates usable in conditions.) Additionally, as there might be several occurrences of the same grammar element in a rule, we also introduce the possibility to name each occurrence with a corresponding *variable*: $Y\ y \to X_1\ x_1,\ldots,X_k\ x_k$, e.g.,

$$step\ s \to riser\ r,\ tread\ t\ \langle\, \mathsf{edgeAdj}(r,t)\,\rangle$$

Attributes. Moreover, we consider that each grammar element has *attributes*, describing features of the underlying geometric primitive or complex form. An attribute can be of a primitive type (e.g., Boolean, integer, float, 3D-vector) or correspond to a grammar element. We thus actually consider an *attribute grammar* $G = (N,T,P,S,A)$ where A is a set of attribute names and each rule in P is of the form $Y\ y \to X_1\ x_1,\ldots,X_k\ x_k\ \langle C\rangle\ \{E\}$ where the condition C may refer both to rule variables $(x_i)_{1\le i\le k}$ and to associated attributes $(x_i.a)_{1\le i\le k, a\in A}$, and where E is a set of *evaluation rules* defining attributes of y among $(y.a)_{a\in A}$, e.g.,

$$step\ s \to riser\ r, tread\ t\ \langle\, \mathsf{edgeAdj}(r,t)\,\rangle\ \{s.\mathsf{len}\} = \{r.\mathsf{len}\}$$

We currently consider only inherited attributes: in a rule $Y \to X_1,\ldots,X_k$, the attributes of Y are determined by the attributes of $X_1,\ldots,X_k$, but the attributes of X_i do not depend on the attributes of Y nor on other X_j with $j \neq i$. Some attributes are also predefined for every form. For example, each nonterminal has a smallest bounding box and provides breadth, length, breadth vector and length vector.

Predicates. A condition C is a conjunction of *predicates* applying to rule variables x_i, possibly via attributes $x_i.a$. Predicates primarily express geometric requirements but can more generally represent any constraint on the underlying object(s). Most predicates apply both to terminals and non-terminals. The main available predicates are:

- $\mathsf{edgeAdj}(x,y)$, $\mathsf{intEdgeAdj}(x,y)$, $\mathsf{extEdgeAdj}(x,y)$, $\mathsf{vertexAdj}(x,y)$: x is adjacent to y via an edge (resp. interior hole edge, exterior contour edge, vertex), as primitive or via any underlying primitive,
- $\mathsf{horizontal}(x)$, $\mathsf{vertical}(x)$: x is horizontal (resp. vertical), as a primitive or all of its underlying primitives,
- $\mathsf{parallel}(x,y)$, $\mathsf{orthog}(x,y)$, $\mathsf{above}(x,y)$, $\mathsf{under}(x,y)$: x is parallel to y (resp. orthogonal, above, under), as primitives or all of their underlying primitives,
- $x\,{=}{=}\,y$, $x\,!{=}\,y$, $x < y$, etc.: x is equal to y (resp. different from, less than, etc.). y can be a literal constant.

Exact predicates are actually relaxed to allow approximate satisfaction: adjacency and relative height position is up to a given distance; horizontality, verticality, parallelism and orthogonality is up to some angle error. Predicates can also be user-defined in the implementation language of our prototype for more complex checking but then, they are not specifically optimized (see below).

Collections. A collection Y of items of the same kind X can be expressed with a recursive rule, e.g., $Y \to X, Y \mid \varnothing$. However, if there are m instances of X, there can be up to 2^m different groups Y. This is useless in practice for semantizing 3D models. Indeed, we are generally interested in the largest group, e.g., all windows in a wall, all steps of a stair, not every subset of windows or steps. For this, we introduce *collection operators* (default value in brackets):

- $\mathsf{maxset}(X, c, [c' = true])$: maximal set of $m \geq 0$ instances $x_1, \ldots, x_m$ of X such that for all $1 \leq i \leq m$, condition $c(x_i)$ holds, and for all $1 \leq i < j \leq m$, both conditions $c'(x_i, x_j)$ and $c'(x_j, x_i)$ hold.
- $\mathsf{maxconn}(X, [c = true], c')$: maximal set of $m \geq 0$ instances $x_1, \ldots, x_m$ of X such that for all $1 \leq i \leq m$, condition $c(x_i)$ holds, and for all $1 \leq i < j \leq m$, there exists a sequence of indices $1 \leq e_1 \ldots e_l \leq m$ such that $i = e_1$, $e_l = j$ and for each $1 \leq k < l$, both conditions $c'(x_{e_k}, x_{e_{k+1}})$ and $c'(x_{e_{k+1}}, x_{e_k})$ hold.
- $\mathsf{maxseq}(X, [c = true], c')$: maximal sequence of $m \geq 0$ instances $x_1, \ldots, x_m$ of X such that for all $1 \leq i \leq m$, condition $c(x_i)$ holds, and for all $1 \leq i < m$, condition $c'(x_i, x_{i+1})$ holds.
- $\mathsf{cycle}(X, [c = true], c')$: collection of $m \geq 0$ instances $x_1, \ldots, x_m$ of X such that for all $1 \leq i \leq m$, conditions $c(x_i)$ and $c'(x_i, x_{i+1})$ hold, noting $x_{m+1} \stackrel{\text{def}}{=} x_1$.

Depending on expected arity, arguments c and c' can be unary or binary predefined predicates, e.g., $\mathsf{edgeAdj}$, or "lambda-terms" such as $x \mapsto c$ or $(x, x') \mapsto c'$.

A operator $oper(X, x \mapsto c, (x, x') \mapsto c')$ is *context-free* iff the only variables that c and c' refer to, if any, are the *non-contextual variables* x and x'. Otherwise, it is *context-sensitive*: it refers to at least one *contextual variable*, i.e., a rule variable or attribute in the RHS different from x and x'. All operators define an attribute "size" that can be used to constrain in C the number of elements in the collection. Note that the $\mathsf{optional}$ statement actually is a maximal operator too. $Y \to X_1, \mathsf{optional}\ X_2$ means that X_1 is enough to build a Y, but X_2 has to be associated if present. It is equivalent to $Y \to X_1, \mathsf{maxset}(X_2)\ s_2\ \langle s_2.\mathsf{size} <= 1 \rangle$, which has a single interpretation, whereas $Y \to X_1 \mid X_1, X_2$ has two. A simple grammar example for a stairway is given in Figure 4. Mentioned grammar elements are pictured in Figure 5.

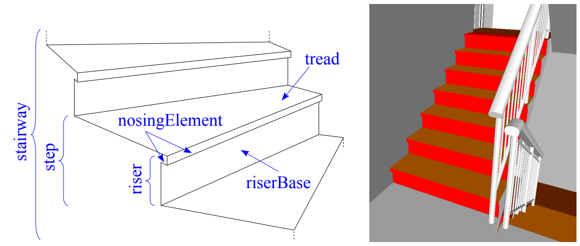

Figure 5: *Elements of the stairway grammar (left), and example of automatic recognition on a CAD model (right).*

3. Scene interpretation

Interpreting a scene, given primitives and a grammar, consists in producing one or several parse trees. A parse tree reflects the structure of the scene w.r.t. the grammar. It provides primitives with semantic labels and relations.

Parse tree. A *parse tree* is a tree t anchored on primitives that reflects the instantiation and combination of grammar rules. Each node n in t corresponds to a terminal or non-terminal $\tau(n) \in T \cup N$, that is called the *type* of n. A terminal node is associated to each geometric primitive in the scene according to its type: polygon, cylinder, etc. Each nonterminal node n corresponds to the instantiation of a rule $Y\ y \to X_1\ x_1, \ldots, X_k\ x_k\ \langle C \rangle\ \{E\}$ in the sense that:

- node n has type $\tau(n) = Y$,
- the sons of n are nodes $(n_i)_{1 \leq i \leq k}$, of type $(X_i)_{1 \leq i \leq k}$,
- assigning $(n_i)_{1 \leq i \leq k}$ to variables $(x_i)_{1 \leq i \leq k}$ satisfies condition C and defines via E the attributes $y.a$ of n.

The set of all terminals under node n if called a *form*, of type $\tau(n)$. (We detail in Sect. 4 the case of collection operators.) With a 1D grammar, the ordered sequence of terminals in a parse tree is equal to the input string, each terminal being counted once and only once by construction. In our relaxed grammatical setting, we have to explicitly require an *exclusivity constraint* to prevent the multiple occurrence of the same terminal in a parse tree. E.g., if a step belongs to a stair, it cannot be used in another stair in the same building.

Parse forest. In 1D, parsing generally has to derive the *whole* input string from the start symbol. In contrast, for 3D scene interpretation, we are interested in locating as many objects as possible among possibly uninterpreted data. In

tread t	$\rightarrow$	polygon p	$\langle$horizontal(p), p.breadth <=2.0$\rangle$
riserBase b	$\rightarrow$	polygon p	$\langle$vertical(p.breadthVector), 0.05 <=p.breadth, p.breadth <=0.25$\rangle$
nosingElement e	$\rightarrow$	polygon p	$\langle$horizontal(p.lengthVector), p.breadth <=0.05$\rangle$
riser r	$\rightarrow$	*riserBase b*, maxseq(*nosingElement*, edgeAdj) n	$\langle$edgeAdj(b,n), above(b,n)$\rangle$
step s	$\rightarrow$	*riser r*, *tread t*	$\langle$edgeAdj(r,t), above(r,t)$\rangle$
stairway w	$\rightarrow$	maxseq(*step*, edgeAdj) s, optional *riser r*	$\langle$edgeAdj(s,r), above(s,r)$\rangle$

Figure 4: *Grammar example for a stairway (units in meter).*

fact, contrary to much other work that produce a single, best parse tree, we construct a shared forest F representing *all* possible parse trees. Consequently, we identify by nature all occurrences of a grammatical element in the scene. However, grammatical ambiguity can occur and different, mutually exclusive analyses can be obtained. One of the key to our approach is to constrain the grammar enough so that only relevant analyses are defined. In most cases in our experiments, we actually obtain scenes with one or a few interpretations. It is future work to estimate the likelihood of rule application to prune ambiguous analyses and select only the most pertinent parse(s). The shared parse forest F is a directed acyclic graph (DAG): a node n in F can belong to several trees. This is a compact representation: a forest of size m can denote a number of parse trees exponential in m.

Node sharing is not incompatible with the exclusivity constraint. The actual requirement is that two forms sharing a terminal are not associated into a new form. E.g., if a step belongs to a stair, it cannot be used *simultaneously* in another stair in the same building. But it can be shared by two stairs and thus belong to both of them as long as they do not "live" in the same interpretation: they are then mutually exclusive. Owing to this constraint, the different parses in a shared forest F correspond exactly to all maximal subsets of root nodes in F such that their tree are pairwise disjoint.

4. Bottom-up parsing

We now describe how to efficiently parse a scene, given geometric primitives and a grammar as described above. This procedure is fully automatic.

Parse forest computation. Our parsing algorithm operates bottom-up, starting with a forest made of all geometric primitives as terminal nodes, and iteratively applying all possible grammar rules to create new nonterminals nodes in the forest. Applying a rule (see below) consists in looking in the forest for nodes of type as specified in the RHS, checking that the rule condition holds, and generating a new node corresponding to the LHS, with associated attributes.

To enforce sharing and guarantee termination, (1) we merge all identical trees as they are constructed, (2) we reject a rule application that constructs a new node of type X if it already contains a node of type X as a succession of only-child descendants, and (3) we stop iterating when no rule can be applied anymore. The exclusivity constraint guarantees that all rule application eventually fail as the number of primitives involved in a nonterminal instance is bounded by the number of primitives in the scene.

Besides, building a maximal collection $oper(X,\ldots)$ practically requires that all instances of X are known. For this, we partition the grammar rules into layers that are fully processed in an order that guarantees this requirement. We consider the graph of the nonterminal dependency relation defined as follows: given a rule $Y \rightarrow X_1,\ldots,X_k$, then Y depends on each nonterminal X_i, or on X_i' if $X_i = oper(X_i',\ldots)$. We then construct the graph strongly connected components. If any component includes an edge corresponding to a maximal operator dependency, e.g., Y depends on $oper(X,\ldots)$ and X depends on Y, then the grammar is deemed unusable for parsing. Last, the condensation of the dependency graph, i.e., the contraction of each component to a single vertex, is a DAG that we order using a reverse topological sort, yielding an ordered partition $((Y_{i,j})_{j\in J_i})_{1\leq i\leq d}$ of the nonterminals. The layers $(R_i)_{1\leq i\leq d}$ that we consider consist of the rules $r_{i,j}$ that have $Y_{i,j}$ as LHS. All this computation of ordered layers is linear in the number of nonterminals.

This iterative scheme is summarized in Algorithm 1. Note that terminals are encapsulated primitives, predefining extra information, such as the bounding box, in the form of grammar attributes. In the following, we consider the case of a rule $r = Y\ y \rightarrow X_1\ x_1,\ldots,X_k\ x_k\ \langle C\rangle\ \{E\}$. We explain how to efficiently look for nodes in F that match types $\tau(X_i)_{1\leq i\leq k}$ and that satisfy C and the exclusivity constraint. We first describe the case of simple grammar elements and general constraints, then extended it to reversible predicates as well as context-free and context-sensitive collection operators.

Algorithm 1 Parse forest computation

```
F ← {terminal(p) | p geometric primitive}
(R_i)_{1≤i≤d} ← ordered partition of P (see text)
for i = 1 to d do
    repeat
        unchanged ← true
        for each rule r ∈ R_i do
            F' ← applyRule(r, F)
            if F' ≠ ∅ then
                unchanged ← false
                F ← F ∪ F'
    until unchanged
```

Operator	Graph components	Algorithm
maxseq	maximal acyclic paths	depth-first search (DFS) to find extremal vertices, then DFS to find paths
cycle	elementary cycles	Tarjan's algorithm
maxset	maximal cliques	Bron-Kerbosch's algorithm
maxconn	strongly connected components	DFS (as the graph is undirected owing to the symmetry of edges)

Table 1: *Maximal collection operators and corresponding graph algorithms to enumerate all instances.*

Basic rule application. The basic idea consists in assembling the sets $D(x_i)$ of nodes in F whose type is X_i, i.e., $D(x_i) = \{n \in F \mid \tau(n) = \tau(x_i)\}$. This can be computed for the whole rule in a single pass on F, or maintained in a separate data structure. If any domain $D(x_i)$ is empty, the rule does not apply. Then all node combinations $(n_i)_{1 \leq i \leq k} \in \prod_{1 \leq i \leq k} D(x_i)$ can be checked to see if condition $C(n_i)_{1 \leq i \leq k}$ is satisfied, before a corresponding new Y node is constructed, with specific attributes $E(n_i)_{1 \leq i \leq k}$.

For efficiency, it is critical to order variable instantiations and constraints checking so that impossible combinations are discovered as soon as possible to prune the search space. The general principle consists in instantiating the most constrained variables first, as it may fail early. All *unary* predicates $p(x_i)$ in C are thus checked before a new node n_i is inserted into $D(x_i)$. Besides, without prior knowledge, variables x_i with the smallest domains $D(x_i)$ are more likely to participate in failing predicates $p(\dots n_i \dots)$ and should be instantiated first. Similarly, variables involved in many predicates are more likely to fail before variable involved in few or no predicate, and should also be instantiated first. As often in constraint programming, we heuristically stipulate a constraint degree to order variables (in decreasing degree):

$$deg(x) = \frac{\text{number of predicates on } x \text{ in } C}{\mid D(\tau(x)) \mid}$$

Invertible predicates. This simple scheme can be made more efficient using specific predicate knowledge. Considering a binary predicate $p(z_i, z_j)$ where $z_l = x_l$ or $x_l.a_l$, we say it is *invertible* iff, given a value v_i for argument z_i (and conversely for z_j), the set N_j of values n_j satisfying $p(v_i, n_j)$, or respectively $p(v_i, v_j.a_j)$ if $z_j = x_j.a_j$, is small and can be efficiently enumerated. It is used to narrow the domains: $D_j \leftarrow D_j \cap N_j$. If D_j becomes empty, variable assignments backtracks and next possible node for x_i is considered.

Adjacency predicates are invertible: given a primitive or form n_i, it is in general adjacent to just a few other primitives or forms n_j. For primitives, a rich adjacency graph is computed before rules are processed, yielding immediate adjacency answers. For forms, to list all nodes of a type τ adjacent to a node n, we go down node n to its primitives, get adjacent primitives, and go up the forest from these adjacent primitives to look for nodes of type τ. A cache of already visited nodes prevents redundant traversals.

Equality $z_i == x_j$ is invertible too: given a node or value v_i for argument z_i, there are in general few nodes n_j such that $v_i = n_j$. This is also true for $z_i == x_j.a$ and $v_i = n_j.a$. (Note that strict equality only makes sense for discrete domains.) Efficient retrieval of such nodes can be achieved by hashing: before processing a rule, a map is built for each argument of such an equality, associating to each possible value v_i of argument z_i the set of nodes n_j such that $v_i = n_j$ and likewise symmetrically. Again, this is also true for $v_i = n_j.a$. These maps can be constructed with a single, linear pass on all nodes of F.

On the contrary, predicates such as above or orthog are not considered invertible: many primitives or forms can be above another one and, in a man-made environment such as a building, many primitives are orthogonal to each others.

Last, as variables occurring in an invertible predicate are by definition more constrained, they should be instantiated first. For this, we introduce a constraint degree $deg_{\text{inv}}(x)$ defined as follows and list variables lexicographically in descending order, first according to deg_{inv}, then deg.

$$deg_{\text{inv}}(x) = \frac{\text{number of invertible predicates on } x \text{ in } C}{\mid D(\tau(x)) \mid}$$

Context-free operators. Collection operators $oper(X, x \mapsto c, (x, x') \mapsto c')$ are treated as any terminal or nonterminal X_i. The only difference is in the computation of domain $D(x_i)$, that is not built via a linear traversal of F.

The exclusivity constraint is a major issue for such operators. In general, the set of *valid* collections (i.e., that satisfy exclusivity) that are maximal cannot be easily deduced from the set of maximal collections. Our approach is to approximate the set of maximal valid collections, staying on the safe side: we may underestimate it, i.e., miss some collections, but not overestimate it, i.e., build non-valid or non-maximal valid collections. Besides, if there is no exclusivity between elements of a given type, maximal collections also are maximal valid collections. Users can thus be told if a parsing result is complete or if solutions could be missing. In practice, grammars can often be written to prevent incompleteness.

To build $D(x_i)$ if the operator is context-free, i.e., if variables in c and c' are only x and x', we construct a graph $G = (V, E)$ where $V = \{n \in F \mid \tau(n) = X \wedge c(n)\}$ and $E = \{(n, n') \mid c'(n, n') \wedge \neg excl(n, n')\}$, where *excl* checks exclusivity. E is symmetrized for maxset and maxconn. Maximal collection instances are then computed as described in Table 1. The exhaustivity of maximal valid collections is always satisfied for maxset as there is no edge between exclusive nodes. It is not for other operators. Besides, exclu-

Element	Color	Element	Color
Wall	gray	Opening	cyan, blue
Roof	terracotta	Stair	red, orange
Floor	green, brown	Not assigned	black

Table 2: *Color assignments for semantized elements.*

sivity has to be enforced at collection level. For maxseq and maxconn, the constraint is checked as the graph is traversed; for cycle, it is checked after cycle construction. In the worst case, despite maximality, the number of such collections can be exponential in the number of primitives. But in practice, with the low connectivity of our graphs, the number of collections remains low, and enumeration is fast.

Context-sensitive operators. Semantic and complexity issues arise when context is introduced in the conditions of maximal collection operators. For this reason, we impose that all contextual variable in c and c' are instantiated before the operator is considered for instantiation.

It excludes rules with circular dependencies between operators. It does not seem a practical limitation: this was in fact never needed in our experiments. Also, the prior instantiation condition never was a sacrifice to efficiency. On the contrary, collections are more efficiently grown from known anchors, rather than without constraints and later discarded. If a valid variable instantiation ordering exists (no cycle), it is given by a topological sort on the variable dependency relation. This order is partial and previously mentioned criteria concerning constraint degrees deg and deg_{inv} are then used for optimizing backtracking efficiency. Last, variables with identical constraints for instantiation ordering are kept sorted as written in the rule to let the writer of the grammar possibly specify his/her own heuristics.

5. Experiments

We have implemented a parser prototype that supports the above features (except the syntactic sugar). We present here results obtained both on CAD models with a relatively clean geometry, and on real or realistic data with incomplete and partly wrong geometry. In all experiments, we have used grammars where terminals are surfaces rather than volumes. It is consistent with a semantization task from surfacic data obtained, e.g., with a laser scan or photogrammetry. Our approach is not restricted to surface parsing though.

Results on CAD models. We processed 5 building information models (BIM) created with CAD software, in IFC format. Using IfcObj, we extracted their geometry, i.e., a soup of 3D triangles, which we merged into 3D polygons with holes with proper adjacency, and feeded to our parser. The buildings are listed in Table 3, with size information.

In theory, one could construct polygons by just merging co-planar triangles sharing two vertices. In practice, triangles are not exactly co-planar and what we actually construct as "3D polygons" are sets of almost co-planar 3D triangles (angular difference less than 3°). More importantly, CAD models do not feature proper triangulations. The reason is they are in general constructed mostly by replicating, stucking and stacking already meshed components (2D or 3D), rather than by appropriately meshing the surface of a well-defined volume. Besides, some points or surfaces are not always properly snapped one to another. The actual task to form polygons is thus to merge possibly overlapping triangles with approximate adjacency, rather than just connect them via simple and exact adjacency (sharing two vertices). This a difficult problem, for which we have used a simple heuritics that works well on our examples. We first establish the inclusion relation or approximate adjacency of triangle edges in other triangles. To that end, for each vertex of a triangle T, we look for neighboring triangles T' (at a small distance, typically 1-10 mm), using an AABB tree. If two vertices v_1 and v_2 of T are inside or nearly adjacent to the same triangle T', then we merge the two triangles. There are actually two cases. If triangles T and T' share both vertices v_1 and v_2, this is a simple ordinary fusion. If not, we create a new vertex v' at the centroid of T' as well as a new triangle $T'' = (v', v_1, v_2)$, and then merge the three triangles. The process is iterated from a given triangle in a region-growing fashion: merging it with nearly adjacent or overlapping triangles, and putting them into a worklist from which new triangles are drawn to continue growing the region. The actual fusion of triangles consists in projecting them into the plane of the seed triangle and merging the projected triangles (with the 2D-polygon tools of the CGAL library). This defines a 2D polygon with holes, whose edges are back-projected to the corresponding 3D triangles, forming a 3D polygon.

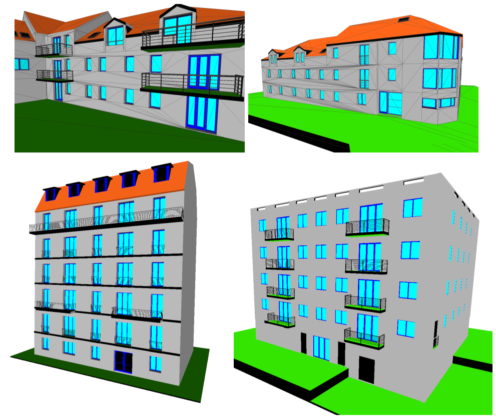

Figure 6: *Automatic semantization of roofs, walls, openings in CAD models. Top: LcG. Bottom: LcA (left), LcD (right).*

We ran the parser with a general building grammar (not particular to a specific kind of architecture) to recover building elements: walls, roofs, floors, stairs, openings (windows and doors). Some results are pictured in Figures 2, 3, 6 and 7. The color code is given in Table 2. Table 4 provides quantitative evaluation for stairs and outside building elements.

Name	# of triangles	# of polygons	Parsing time (s) stairs	openings
LcG	48332	9705	5	15
LcA	111979	26585	14	42
LcC	385541	111732	33	306
LcD	313012	75257	25	111
LcF	286996	84347	39	322

Table 3: *Models and parsing time (w/o polygon processing).*

Name	# of stairs	# of steps	Stairs (%) Prec.	Rec.	Openings #	Prec.	Rec.
LcG	3	45	100	93	83	100	90
LcA	6	84	100	100	62	98	83
LcC	30	210	100	100	196	100	98
LcD	5	61	93	100	74	100	93
LcF	7	98	100	50	99	100	96

Table 4: *Evaluation of stair & opening semantization: number of items (#), precision (Prec., %) and recall (Rec., %).*

As can be visible from this table as well as illustrations, we properly recognize most elements. We only have a few false positives (stairs with a spurious riser, openings including a spurious window ledge) and some false negatives (missed stairs and openings). We actually discovered that the geometry originating from CAD models was not as clean as expected. Triangles happened to reflect the history of building block composition, with meaningless interpenetrating volumes causing spurious polygon adjacencies and bogus sizes. For instance, some steps are missed because some risers go deep inside the slab or the preceding step, leading to sizes larger than expected and wrong adjacencies. Missed steps in LcF are due to the strange geometry of the stairs, that looks like two interlaced stairs with twice as large risers and treads. There are missed adjacencies too due to bad block snapping. Those were by far the major cause of missed and spurious detections. In fact we would not have such problems with (clean) real data, e.g., laser scans, as only the visible geometry would be modeled. Concerning false negatives, a few frameless doors are geometrically part of a wall plane in LcC, LCD and LcF, and thus included in wall polygons before having a chance to be parsed (see Section 6 though). Missed openings in LcA and LcG are due to limitations in our grammar, which does not take into account dormer windows (with small walls sticking out of the roof) nor skylights (non vertical windows). There are also many French windows/doors in these buildings because of balconies, but model designers disagree: some tag them as doors in the original model, others as windows. This is why we do not report here separate figures for doors and windows; we just labeled them as openings.

Running times are given in Table 3, with a prototype implementation that is far from being optimized. The grammar has been split here in two parts to measure the relative contribution of the two main complex objects: stairs and openings.

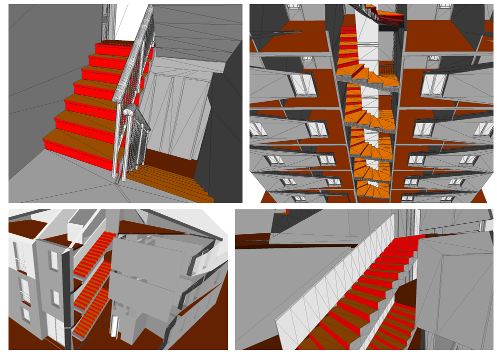

Figure 7: *Semantization of slabs and stairs in CAD models.*

(It has little impact on the overall parsing time.) We can observe good asymptotic properties. In particular, even though we use a bottom-up approach, it is *not* exponential, which is a key feature and major originality of our parser. The whole point of our constraint-processing approach is precisely to prune the search space to prevent exponential exploration. For instance, when parsing stairways, a first *step* is drawn as a potential element of a "maxseq(*step*)". Being a maxseq, a stairway is grown with more steps in two directions starting from this initial step, based on the adjacency graph, which is basically linear. The steps belonging to the resulting stairway are recorded as such and when they are later considered as other potential seed steps to find other stairways, no growing is attempted in the same directions as it has already been performed. This mostly linear behavior is reflected in the time measurements of Table 3, w.r.t. the number and size of stairs (Table 4). Considering the most complex model (LcC), an exponential approach would not be able to determine in 33 s that the 210 detected steps exactly form 30 different stairways. This generalizes to more complex situations. In practice, the running time depends mostly on constraint rigidity and on the average vertex degree in the adjacency graph, little on the number of rules or the number of variables in a rule (assuming rule conditions constrain rule application enough). A combinatorial explosion is theoretically possible, but does not seem to happen on realistic cases.

In the current implementation of the algorithm, rules are applied sequentially. But they could be applied in parallel too, in particular independant rules. Rule dependency has to be obeyed though (see Section 4). For instance, the computation of walls and stairs are independent, but it is not the case for walls and floors or stairs and floors as walls must be adjacent to floors, so as stairs; floors must be detected first.

Results on real data. To evaluate our algorithm on real data, we consider a multi-view photogrammetric reconstruction of a castle courtyard using images in Strecha's dataset [SvHVG*08]. This is a kind of worst case scenario as noise here is typically 5-10 cm, which is huge compared to window ledges. Yet, the 3D mesh is precise enough to display a number of edges of windows looking onto the yard. To construct polygons, we segment the underlying point cloud

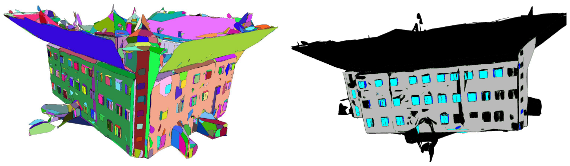

Figure 8: *Castle courtyard, seen "from the outside". Left: 1817 extracted primitives. Right: automatic semantization.*

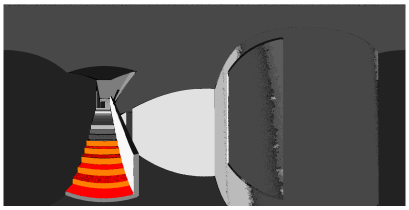

Figure 9: *Starway detection on simulated LiDAR data.*

into planar clusters [SWK07], then compute an alpha shape for each cluster to build the 1817 corresponding polygons (see Fig. 8, left). Some polygons corresponding to edges of windows are not found at this point; conversely, some shapes are split into separate primitives or wrongly estimated, yielding spurious or abnormal polygons. Finally, an approximate adjacency graph is computed, to be given as input with polygons to our parser. To accomodate the noise in the data, we use a simplified grammar for windows. We look only for windowpanes surrounded by a large enough (≥ 2) sequence of edges (using maxseq) rather than a complete frame (using cycle). Besides adjacency, we also weaken size and orientation constraints. Although a general methodology for such a grammar relaxation is needed (see Section 6), the result is encouraging. Indeed, walls and many windows are reasonably well recognized (see Fig. 8): we detect 22 windows out of 31 on the 3D facade. These are decent results given the level of noise and the lack of photometric information. Window recognition actually is difficult. It is not considered a solved problem in 2D [TKS*11, STK*12], let alone in 3D.

We also experimented with CAD-based, simulated LiDAR scans, including effects due to visibility, anisotropic sampling and slight noise. After normal estimation [BM12], we produced segments as regions grown on depth images and polygons as bounding rectangles. Figure 9 illustrates stairs detection on such data: visible steps are well detected, but invisible treads prevent the full straiway to be recovered.

6. A case for grammars

Results on CAD models are extremely promising, both regarding accuracy and practical complexity. However, there is still room for improvement concerning real data. The main issues are noise and incomplete data.

Because of noise, geometric primitives can be missed, wrongly estimated, or split into separate components. As future work, to make our apporach more robust, we are considering introducing simultaneous alternative detections (the most probable ones), rather than a single one. This would fit nicely with the excluvity constraint: at most one of the detections could participate in an actual form. This would be compatible with split detections too: subsets of the point cloud could be interpreted alternatively as several primitives or a single one. Additionnaly, rather than following sharp binary decisions to allow (or not) component composition via grammar rules and constraint satisfaction, we are considering continuously relaxing some constraints to provide approximate matching. Of course, detection alternatives and contraint relaxation would make the search space much larger. But this could be adressed by taking into account detection likelihoods and fitness measures for relaxed constraints, which could be combined with rule probabilities too. As a result, infered nonterminals could be given scores and the shared parse forest would then be pruned according to this score to allow exploring large spaces while keeping a reasonable size. Along this line, a useful feature would be to possibly split (as alternatives) a shape into several pieces, e.g., to address the detection of doors that are merged into wall too early in the process. This could either be supported by photometric information at primitive detection level for a better segmentation, or by late splitting during parsing if shape cues are provided (e.g., the recognition of the other side of the door). Photometric cues could also be used to assign labeling likelihoods, as in top-down approaches [TSKP10], to later participate in shape scoring.

Besides, we believe the strength of our approach will be fully distinctive with a proper treatment of partial information, e.g., due to occlusions or fragmentary acquisitions. The fact that all the structure and regularity is expressed formally in the grammar makes it easier to reason on it, to complete existing elements and to hypothesized missing ones [HZ09]. Our preliminary results on this kind of top-down completion are encouraging, including when several elements are missing, such as a few steps between two floors. To infer likely objets like this, comprehensive grammar constraints are essential. An open issue is to find the right level to enforce general regularity such as dimension repetition and symmetry [PMW*08], whether by explicit operators or meta rules.

Actually, we argue that formulating structure, including regularity, as expressed in a grammar, is crucial for semantic analysis. With real (noisy, incomplete) data, structure is essential to prevent spurious and missed detections. For instance, grid alignment constraints would be crucial to recover missed windows in Figure 8. Even on perfect CAD data, spurious step objects are detected in the polygon soup, and regularity is required to compose individual steps into consistent stairways, filtering out false detections. One of the nice thing about grammars is that each rule is simple; complexity only originates from their composition. For example, *all* 9 rules in the actual stairs grammar are indispensable to obtain the accurate recognition of Table 4.

Structure can be hard-wired in code performing semantic analysis. But such programming would not scale to complex objects and scenes. A formalism to represent structure is required, both to allow systematic optimization (pruning via constraints) and to facilitate the expression of rules by humans. We see a grammar formalism as a domain-specific language to express regularity. In fact, rules here are assumed written by experts, e.g., architects, not computer scientists. A separate issue is the automatic inference of grammar rules from a database of annotated examples. Another one is shape approximation, to prevent exhaustive grammatical descriptions and gain robustness. For instance, although our actual grammar for stairs easily accomodates complex (but regular) noses, it would be better to specify a simple "generic" nose and a basic shape matching approximation. Anyway, structure and regularity have to be made explicit.

7. Conclusion

We have presented a high-level grammar formalism to specify complex objects, and a practical parsing procedure. Although it relies on efficient graph algorithms, the low-level node-and-edge machinery remains hidden, allowing large and maintainable specifications, to be written by non computer scientists. Thanks to alternatives, maximality operators, and recursion, the expressive power our grammars is larger than that of graph pattern matching. Owing to maximality and to the mixing of adjacency conditions with other kinds of constraints, it is superior to graph grammars too.

Although we have good results for CAD models, concerning both accuracy and running times, work clearly remains to properly handle noise and incomplete data. We actually consider that this paper constitutes a well-delimited first step towards more general scene parsing. We enumerated (Section 6) a number of sensible and promising research directions to address more complex issues arising with real data.

References

[ARSF09] ATTENE M., ROBBIANO F., SPAGNUOLO M., FALCIDIENO B.: Characterization of 3D shape parts for semantic annotation. *Computer-Aided Design 41*, 10 (2009), 756 – 763. 1

[BM12] BOULCH A., MARLET R.: Fast and robust normal estimation for point clouds with sharp features. *Comp. Graph. Forum 31*, 5 (Aug. 2012), 1765–1774. 9

[BMMP03] BIASOTTI S., MARINI S., MORTARA M., PATANÉ G.: An overview on properties and efficacy of topological skeletons in shape modeling. In *Shape Modeling Int'l* (2003). 2

[CLS10] CHAJDAS M. G., LEFEBVRE S., STAMMINGER M.: Assisted texture assignment. In *SI3D* (2010), ACM. 2

[EMM03] EL-MEHALAWI M., MILLER R. A.: A database system of mechanical components based on geometric and topological similarity. *Computer-Aided Design 35*, 1 (2003). 2

[GXTL10] GAO X., XIAO B., TAO D., LI X.: A survey of graph edit distance. *Pattern Anal. Appl. 13*, 1 (Jan. 2010), 113–129. 2

[HBS11] HUANG H., BRENNER C., SESTER M.: 3D building roof reconstruction from point clouds via generative models. In *Advances in Geographic Information Systems* (2011), ACM. 2

[HZ09] HAN F., ZHU S.: Bottom-up/top-down image parsing with attribute grammar. *Tr. PAMI 31*, 1 (2009), 59–73. 2, 9

[MMWG11] MATHIAS M., MARTINOVIC A., WEISSENBERG J., GOOL L. V.: Procedural 3D building reconstruction using shape grammars and detectors. In *IEEE International Conference 3DIMPVT* (2011), pp. 304–311. 2

[MMWVG12] MARTINOVIĆ A., MATHIAS M., WEISSENBERG J., VAN GOOL L.: A three-layered approach to facade parsing. In *ECCV* (2012), LNCS 7578, Springer, pp. 416–429. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (2006), 614–623. 2

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3 (2007). 2

[Pag98] PAGALLO G. M.: Constrained attribute grammars for recognition of multi-dimensional objects. In *Advances in Pattern Recognition*, LNCS 1451. Springer, 1998, pp. 359–365. 3

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3 (2008), 1–11. 9

[PSBM07] PASCUCCI V., SCORZELLI G., BREMER P.-T., MASCARENHAS A.: Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph. 26*, 3 (2007), 58. 2

[RB07] RIPPERDA N., BRENNER C.: Data driven rule proposal for grammar based facade reconstruction. In *Photogrammetric Image Analysis (PIA)* (2007), pp. 1–6. 2

[STK*12] SIMON L., TEBOUL O., KOUTSOURAKIS P., VAN GOOL L., PARAGIOS N.: Parameter-free/Pareto-driven procedural 3D reconstruction of buildings from ground-level sequences. In *CVPR* (2012), pp. 518–525. 2, 9

[SvHVG*08] STRECHA C., VON HANSEN W., VAN GOOL L., FUA P., THOENNESSEN U.: On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR* (2008). 8

[SWK07] SCHNABEL R., WAHL R., KLEIN R.: Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (June 2007), 214–226. 2, 9

[SWWK08] SCHNABEL R., WESSEL R., WAHL R., KLEIN R.: Shape recognition in 3D point-clouds. In *WSCG* (2008). 2

[TKS*11] TEBOUL O., KOKKINOS I., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Shape grammar parsing via reinforcement learning. In *CVPR* (2011). 2, 9

[TMT10] TOSHEV A., MORDOHAI P., TASKAR B.: Detecting & parsing architecture at city scale from range data. In *CVPR* (2010). 2

[TSKP10] TEBOUL O., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Segmentation of building facades using procedural shape priors. In *CVPR* (2010). 9

[TV08] TANGELDER J. W., VELTKAMP R. C.: A survey of content based 3D shape retrieval methods. *Multimedia Tools Appl. 39*, 3 (Sept. 2008), 441–471. 2

[VAB10] VANEGAS C., ALIAGA D., BENES B.: Building reconstruction using Manhattan-world grammars. In *CVPR* (2010). 2

[VAB12] VANEGAS C. A., ALIAGA D. G., BENES B.: Automatic extraction of Manhattan-world building masses from 3D laser range scans. *IEEE Trans. Vis. Comput. Graph. 18*, 10 (2012), 1627–1637. 2

[WZ11] WU T., ZHU S.: A numeric study of the bottom-up and top-down inference processes in and-or graphs. *IJCV 93*, 2 (June 2011), 226–252. 2