Semantizing Complex 3D Scenes using Constrained Attribute Grammars

A. Boulch, S. Houllier, R. Marlet and O. Tournaire

slight variant of the material presented at SGP 2013
Motivation

Semantizing complex objects in 3D scenes

Bare geometry

Semantized geometry
Motivation

Semantizing complex objects in 3D scenes

Bare geometry

Semantized geometry
(rendered)
Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud → building model

*Semantized geometry*
Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud $\rightarrow$ building model
  - for architects
    - building sketch $\rightarrow$ rendering

*Semantized geometry (rendered)*
Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud $\rightarrow$ building model
  - for architects
    - building sketch $\rightarrow$ rendering
- Game industry
  - for graphic designers
    - basic level design $\rightarrow$ rendering

Semantized geometry (rendered)
Motivation

Semantizing complex objects in 3D scenes

- **Building industry**
  - for the renovation market
    - point cloud $\rightarrow$ building model
  - for architects
    - building sketch $\rightarrow$ rendering

- **Game industry**
  - for graphic designers
    - basic level design $\rightarrow$ rendering

- **Object mining in shape databases**
  - for semantic queries
    - 3D object $\rightarrow$ semantic labeling

Semantized geometry
Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments
Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments
Grammars to express hierarchical decomposition

The super computer rules Discovery One

Sentence
  Subject
    Article
      The super
  Verb
    Adjective
    Noun
  Complement

Building
  Facade
    Window
  Roof
    Wall
  Floor
Complex, non-hierarchic relations between components

The super computer rules Discovery One

agreement (singular)
Constrained attribute grammars

\[ G = (N, T, P, S) \]

- **N**: nonterminals (\( \leftrightarrow \) complex forms)  
  e.g., window, wall
- **T**: terminals (\( \leftrightarrow \) geometric primitives)  
  e.g., polygon, cylinder
- **P**: production rules (\( \leftrightarrow \) hierarchical decomposition and constraints)
- **S**: start symbols (\( \leftrightarrow \) root shapes)  
  e.g., building
Basic rules

Decomposition of a complex object $y$ of type $Y$ into its constituents $x_i$ of type $X_i$:

$$Y \ y \rightarrow X_1 \ x_1, \ldots, X_n \ x_n$$

Example:

$$\text{step } s \rightarrow \text{riser } r, \text{tread } t$$

Rule application:

- **top-down view:**
  - $y$ decomposes into $x_1, \ldots, x_n$

- **bottom-up view:**
  - given some $x_1, \ldots, x_n$, create a new object $y$
Basic rules

Decomposition of a complex object $y$ of type $Y$ into its constituents $x_i$ of type $X_i$:

$$Y \ y \rightarrow X_1 \ x_1, \ldots, X_n \ x_n$$

Example:

$$\text{step } s \rightarrow \text{riser } r, \text{tread } t$$

Rule application:

- **top-down view:**
  - $y$ decomposes into $x_1, \ldots, x_n$
- **bottom-up view:**
  - given some $x_1, \ldots, x_n$, create a new object $y$
Constraints

Conditional rule application (conjonction of predicates):

\[ Y \ y \rightarrow X_1 \ x_1, X_2 \ x_2, \ldots \langle \text{cstr}_1(x_1), \text{cstr}_2(x_1, x_2), \ldots \rangle \]

Example:

- \text{riser} \ r \rightarrow \text{polygon} \ p \langle \text{vertical}(p) \rangle
- \text{tread} \ t \rightarrow \text{polygon} \ p \langle \text{horizontal}(p) \rangle
- \text{step} \ s \rightarrow \text{riser} \ r, \text{tread} \ t
  \langle \text{edgeAdj}(r, t), \text{above}(t, r) \rangle
Attributes

Features attached to each grammar element:
- at creation time (primitives)
- at rule application (synthetized attributes)

Examples:
- length, width
- bounding box
- ...

\[ \text{r.length} \quad \text{r.width} \]
Predicates

Predicates on grammar elements:
- `adj`, `edgeAdj`
- `orthogonal`, `parallel`
- `vertical`, `horizontal`
- ...

Predicates on attributes:
- `≥`, `>`, `≤`, ...
- `=`, `≠`
- ...

Example: `r.length == t.length`
Collections of similar elements

Grouping elements via recursion ($Y$ as set of $X$s):

$Y \ y \rightarrow X \ x$

$Y \ y \rightarrow X \ x, Y \ y_2$

Grouping elements via specific collection operators:

$Y \ y \rightarrow \text{coll}(X) \ xs$

Useful operators: maximal collections

- maxconn: maximal set of connected components
- maxseq: maximal sequence
- ...

Example:

$$\textit{stairway sw} \rightarrow \text{maxseq}(\textit{step, adjEdge}) \ ss$$
Collections of similar elements

Grouping elements via recursion ($Y$ as set of $X$s):

$$Y \ y \rightarrow X \ x$$
$$Y \ y \rightarrow X \ x, Y \ y_2$$

Grouping elements via specific collection operators:

$$Y \ y \rightarrow \text{coll}(X) \ xs$$

Useful operators: maximal collections

- maxconn: maximal set of connected components
- maxseq: maximal sequence
- ...

[see complete stairway grammar]

Example:

$$\text{stairway} \ sw \rightarrow \text{maxseq}(\text{step}, \text{adjEdge}) \ ss$$
Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments
Scene interpretation: parse tree

Tree representation of a grammatical analysis of the scene:
- leaves: terminals representing primitives
- non-leaf nodes: instantiations of grammar rules
Scene interpretations: parse forest

Set of parse trees with systematic sharing (DAG)

Compact representation of all possible interpretations
Ambiguity and the exclusivity constraint

Example (assuming no height ordering constraint):

\[ \text{step } s \rightarrow \text{riser } r, \text{tread } t \langle \text{edgeAdj}(r, t) \rangle \]

Exclusivity constraint:

at most 1 occurrence of a grammar element per interpretation
Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments
Parse forest computation

Bottom-up parsing:
construction of the parse forest
from leaves (terminals) to roots (start symbols)

- create one terminal for each primitives
- iteratively create new grammar elements from existing ones
  - e.g., given grammar rule $step \ s \rightarrow riser \ r, tread \ t$
    - given existing instances $riser \ r_{23}, tread \ t_{18}$
    - create new instance $step \ s_5$
- merge identical trees on the fly
- stop iterating when no rule applies
Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality

maxseq
Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality
Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality
  - reverse topological sort of nonterminal dependency graph
Mastering the combinatorial explosion

Usual drawback of bottom-up analysis: combinatorial explosion
- all trees
- all sets, all sequences, ...
- all combinations

Our solution:
- tree sharing: construction of a parse forest (exp. → lin.)
- maximal operators, with efficient implementation (≫ exp. → polyn.)
- constraint propagation: predicate ordering ⇒ early pruning
Maximal operators

$maxseq$ vs $allseq$
Maximal operators

$maxseq$ vs $max(allseq)$
Constraint propagation as predicate ordering
A simple 2D example

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]
Constraint propagation as predicate ordering

Example of poor ordering

\[\text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle\]

1. orthogonality

Complexity:
\[O(\#\text{seg}^2)\]
Constraint propagation as predicate ordering

Example of poor ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, (\text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1)) \]

1. orthogonality

Complexity:
\[ O(\#\text{seg}^2) \]
Constraint propagation as predicate ordering

Example of poor ordering

\[
\text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle
\]

1. orthogonality
2. adjacency

Complexity:

\[
O(\#\text{seg}^2 + \#\text{seg} \times \text{maxDeg}) = O(\#\text{seg}^2)
\]
Constraint propagation as predicate ordering

Example of poor ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]

1. orthogonality
2. adjacency

Complexity:
\[ O(\#\text{seg}^2 + \#\text{seg} \times \text{maxDeg}) = O(\#\text{seg}^2) \]
Constraint propagation as predicate ordering

Example of poor ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]

1. orthogonality
2. adjacency
3. verticality

Complexity:
\[
O(\#\text{seg}^2 + \#\text{seg} \times \text{maxDeg} + \#\text{seg}) = O(\#\text{seg}^2)
\]
Constraint propagation as predicate ordering

Example of poor ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]

1. orthogonality
2. adjacency
3. verticality

Complexity:
\[ O(\#\text{seg}^2 + \#\text{seg} \times \maxDeg + \#\text{seg}) \]
\[ = O(\#\text{seg}^2) \]
Constraint propagation as predicate ordering

Example of good ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]

1. verticality

Complexity :
\[ O(\#\text{seg}) \]
Constraint propagation as predicate ordering

Example of good ordering

\[\text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle\]

1. verticality

Complexity:

\[O(\#\text{seg})\]

satisfied constraints
Constraint propagation as predicate ordering

Example of good ordering

\[\text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle\]

1. verticality
2. adjacency

Complexity:

\[O(#\text{seg} + \#\text{seg} \times \text{maxDeg}) = O(#\text{seg} \times \text{maxDeg})\]
Constraint propagation as predicate ordering

Example of good ordering

\[ \text{pair } p \rightarrow \text{seg } s_1, \text{seg } s_2, \langle \text{orthogonal}(s_1, s_2), \text{adj}(s_1, s_2), \text{vertical}(s_1) \rangle \]

1. verticality
2. adjacency
3. orthogonality

Complexity:
\[ O(\#\text{seg} + \#\text{seg} \times \text{maxDeg} + \#\text{seg} \times \text{maxDeg}) \]
\[ = O(\#\text{seg} \times \text{maxDeg}) \ll O(\#\text{seg}^2) \]

satisfied constraints
Constraint propagation as predicate ordering

1. Unary predicates

\[= \text{constraints implying only 1 element: complexity } O(\#element)\]

Example:

\[\text{riser } r \rightarrow \text{polygon } p \langle \text{vertical}(p) \rangle\]
Constraint propagation as predicate ordering

1. Unary predicates
2. Invertible predicates that are partially instantiated

= constraints with small cardinality when some arguments are fixed

Example:

\[
\text{step } s \rightarrow \text{riser } r, \text{tread } t \langle \text{edgeAdj}(r, t) \rangle
\]
Constraint propagation as predicate ordering

1. Unary predicates
2. Invertible predicates that are partially instantiated
3. General predicates

= remaining relations

Example:

\[ \text{step } s \rightarrow \text{riser } r, \text{tread } t \langle \text{orthogonal}(r, t) \rangle \]
Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments
Semantization pipeline

- CAD Model (triangle soup)
- Point cloud
- Primitive detection
- Polygon extraction
- Parsing
- Grammar
- Semantized model
CAD models

Preprocessing

- Region growing over triangles for polygon creation
- Computation of exact and approximate adjacency graphs
CAD models
Detection of stairs  (more examples in supplem. material)
CAD models
Detection of walls, roofs and openings (more examples in supplem. material)
Real data: photogrammetry

- Preprocessing (point cloud)
  - clustering using RANSAC (or region growing)
  - polygons bounded by alpha shapes

- Problems:
  - missing primitives
  - false primitives
  - wrong adjacencies

- Solution:
  - use of a relaxed grammar
    - looser bounds
    - 1-2 missing items OK
  - 22 openings out of 31
Quasi-real data: simulated LIDAR

- Planes by region growing in depth image
- Polygons as oriented bounding rectangles
- Adjacency based on pixels in depth image
Size and parsing time (CAD models)

<table>
<thead>
<tr>
<th>Name</th>
<th># of triangles</th>
<th># of polygons</th>
<th>Parsing time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>stairs</td>
</tr>
<tr>
<td>LcG</td>
<td>48332</td>
<td>9705</td>
<td>5</td>
</tr>
<tr>
<td>LcA</td>
<td>111979</td>
<td>26585</td>
<td>14</td>
</tr>
<tr>
<td>LcC</td>
<td>385541</td>
<td>111732</td>
<td>33</td>
</tr>
<tr>
<td>LcD</td>
<td>313012</td>
<td>75257</td>
<td>25</td>
</tr>
<tr>
<td>LcF</td>
<td>286996</td>
<td>84347</td>
<td>39</td>
</tr>
</tbody>
</table>
## Precision and recall (%, CAD models)

<table>
<thead>
<tr>
<th>Name</th>
<th># of stairs</th>
<th># of steps</th>
<th>Stairs Prec.</th>
<th>Stairs Rec.</th>
<th>Openings #</th>
<th>Openings Prec.</th>
<th>Openings Rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LcG</td>
<td>3</td>
<td>45</td>
<td>100</td>
<td>93</td>
<td>83</td>
<td>100</td>
<td>90</td>
</tr>
<tr>
<td>LcA</td>
<td>6</td>
<td>84</td>
<td>100</td>
<td>100</td>
<td>62</td>
<td>98</td>
<td>83</td>
</tr>
<tr>
<td>LcC</td>
<td>30</td>
<td>210</td>
<td>100</td>
<td>100</td>
<td>196</td>
<td>100</td>
<td>98</td>
</tr>
<tr>
<td>LcD</td>
<td>5</td>
<td>61</td>
<td>93</td>
<td>100</td>
<td>74</td>
<td>100</td>
<td>93</td>
</tr>
<tr>
<td>LcF</td>
<td>7</td>
<td>98</td>
<td>100</td>
<td>50</td>
<td>99</td>
<td>100</td>
<td>96</td>
</tr>
</tbody>
</table>
Future work

- Principled way to deal with partial or missing primitives
- Exploitation of occlusion/visibility information
- Scoring of interpretations: pick best tree(s)
Conclusion

Constrained attribute grammars:
- appropriate to semantize complex objects
- high-level specification language
  - being expert is enough, computer scientist not required
- efficient even on large models

This work:
- well-delimited first step: perfect data
- extensions required for incomplete/noisy data

On the web
- http://imagine.enpc.fr/
- sites.google.com/site/boulchalexandre/