Semantizing Complex 3D Scenes using Constrained Attribute Grammars

A. Boulch[1], S. Houllier[1] R. Marlet[1] and, O. Tournaire[1,2]
[1] Université Paris-Est, LIGM (UMR CNRS), Center for Visual Computing, Ecole des Ponts ParisTech
6-8 av. Blaise Pascal, 77455 Marne-la-Vallée, France [2] CSTB

1 Parsing stairs

Grammar for stairs

Figure 1 shows a pretty-printed version of the grammar used for extracting stairways. There are two types of risers, to cope with different situations occurring in CAD models. Figure 2 and 3 pictures typical steps extracted using *riserBase* and *riserBottomTreadBase*.

```
// A riser base is a vertical polygon of constrained size.
                 $riserBase\ b \rightarrow$ polygon $p$
                                  $\langle$ vertical($p$),
                                    $0.1 <= p$.breadth $<= 0.3$,
                                    $0.5 <= p$.length $<= 2\ \rangle$
// A nosing element is a thin polygon whose length direction is horizontal.
              $nosingElement\ e \rightarrow$ polygon $p$
                                  $\langle$ horizontal($p$.lengthVector),
                                    $p$.breadth $<= 0.08$,
                                    $0.5 <= p$.length $<= 2\ \rangle$
// A riser with optional nose is a riser base topped with a possibly empty sequence
// of nosing elements (see Figure 2).
              $riserOptNose\ ron \rightarrow riserBase\ b$, maxseq($nosingElement$, edgeAdj) $n$
                                  $\langle$ edgeAdj($b, n$),
                                    above($b, n$) $\rangle$
// A tread is a horizontal polygon of constrained size.
                      $tread\ t \rightarrow$ polygon $p$
                                  $\langle$ horizontal($p$),
                                    $0.1 <= p$.breadth $<= 2$,
                                    $0.5 <= p$.length $<= 2\ \rangle$
// A riser topped with a tread underside and nose (see Figure 3) — useful for CAD models.
     $riserBottomTreadBase\ bt \rightarrow riserBase\ b,\ tread\ t$
                                  $\langle$ adj($b, t$),
                                    above($b, t$) $\rangle$
 $riserBottomTreadNose\ rbtn \rightarrow riserBottomTreadBase\ bt$,
                              maxseq($nosingElement$, edgeAdj) $n$
                                  $\langle$ $1 <= n$.size,
                                    edgeAdj($bt, n$), above($bt, n$) $\rangle$
// A riser is any kind of riser.
                      $riser\ r \rightarrow riserOptNose\ ron$
                                $|$ $riserBottomTreadNose\ rbtn$
// A step is a riser topped by a tread.
                       $step\ s \rightarrow riser\ r,\ tread\ t$
                                  $\langle$ edgeAdj($r, t$),
                                    above($r, t$) $\rangle$
// A stairway is a sequence of steps topped by an optional riser.
                  $stairway\ w \rightarrow$ maxseq($step$, edgeAdj) $s$, optional $riser\ r$
                                  $\langle$ edgeAdj($s, r$),
                                    above($s, r$) $\rangle$
```

Figure 1: Grammar for a stairway (units in meters)

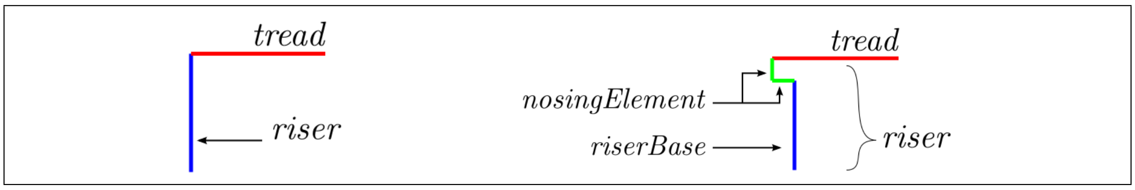


Figure 2: Example of steps extracted with *riserBase*.

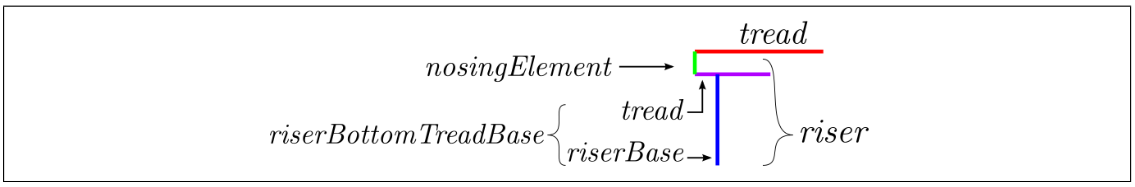


Figure 3: Example of step extracted with *riserBottomTreadBase*.

Parse tree of a stairway model

To illustrate a parse tree example, we use the simple synthetic model pictured on Figure 4. It is composed of four steps with nose.

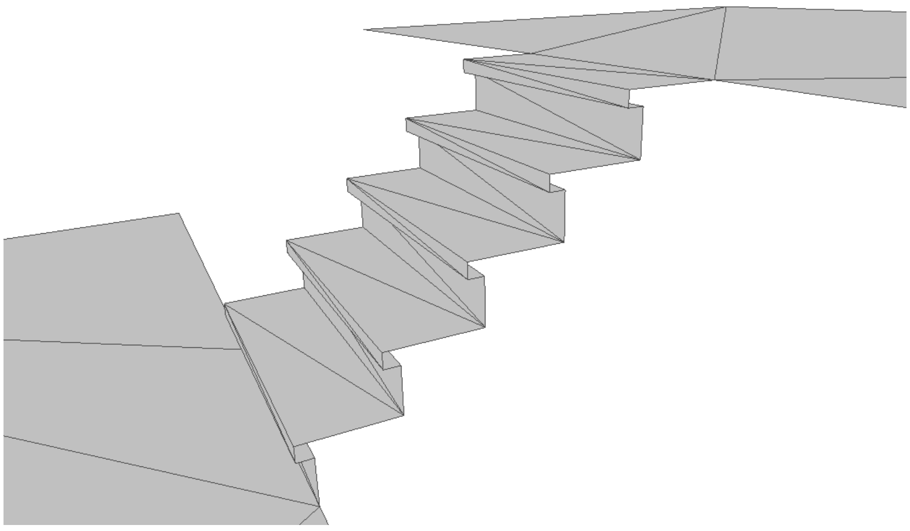


Figure 4: A model to illustrate stairway parsing.

The corresponding parse tree is represented on Figure 5. Nodes with label *OPE* are intermediate nodes generated by our prototype for maximal operators (here a maximal sequence). There is only one interpretation and the parse forest we construct actually is a parse tree.

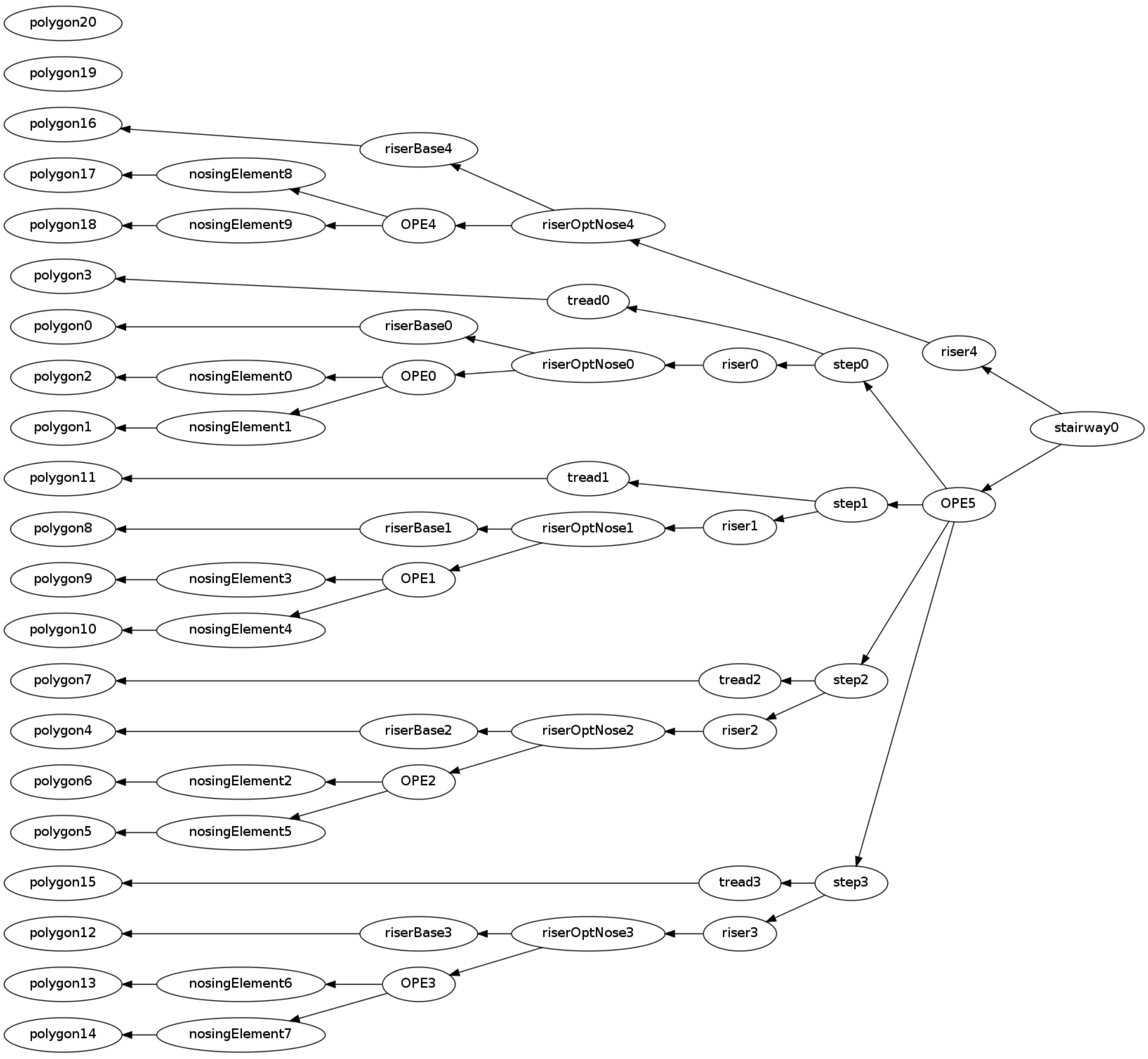


Figure 5: Parse forest (actually tree) of the model pictured on Figure 4, using stairway grammar of Figure 1.

Stair parsing examples on CAD models

The paper describes some results of stairway parsing on CAD models. They are illustrated here on Figures 6, 7, 8, 9 and 10. We show risers in red and treads in orange. We also added basic detections of walls in gray (large vertical polygons) and of floors in brown (large horizontal polygons). As we can see, steps in LcF (Figure 9) are not well detected due to the peculiar modeled geometry of the stairway: risers have twice the usual height.

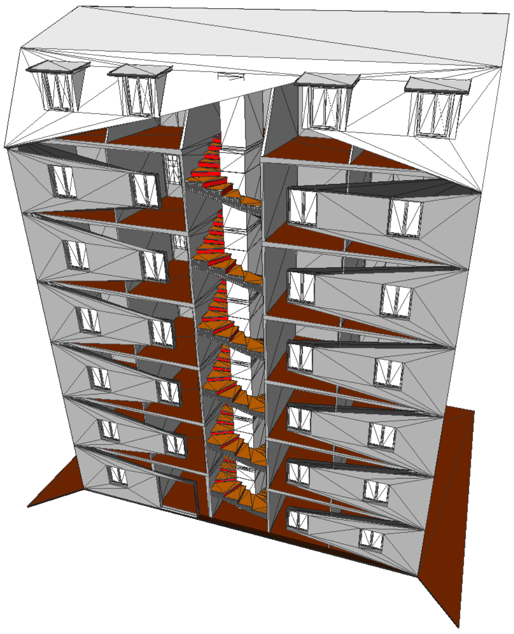
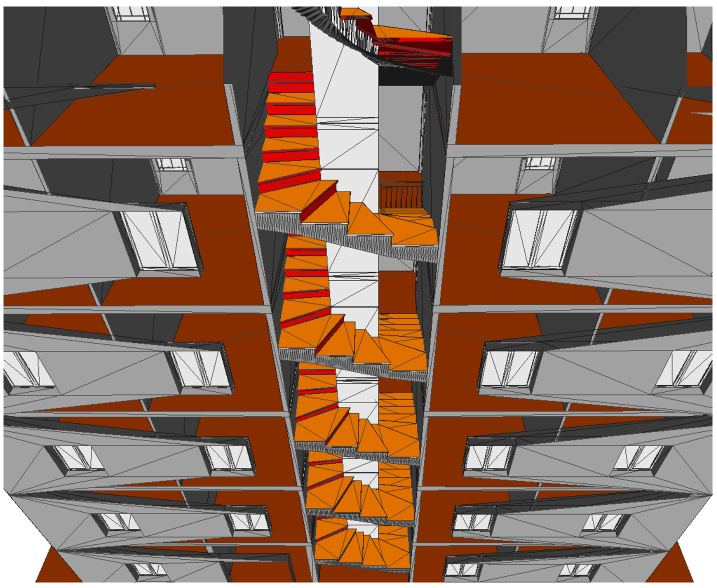



Figure 6: Stairs of the LcA model.

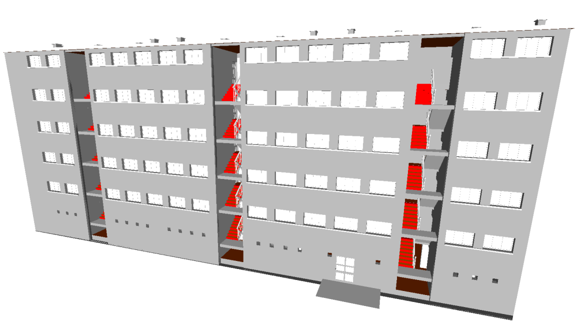
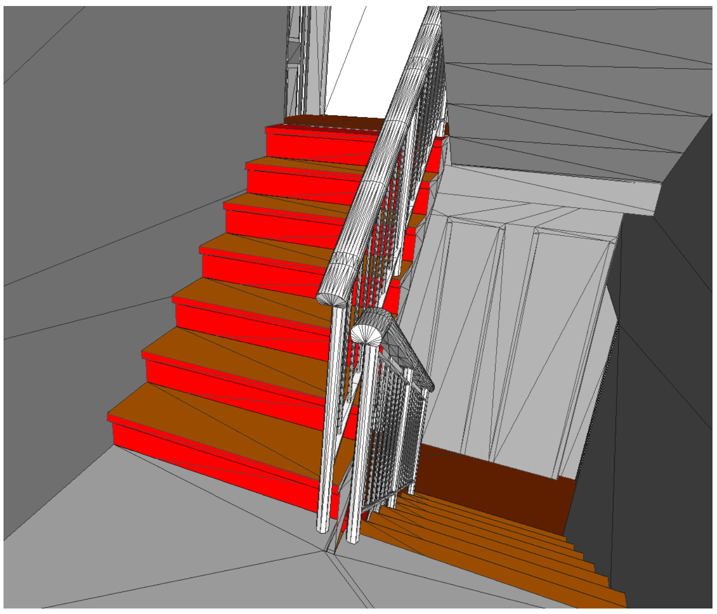



Figure 7: Stairs of the LcC model.

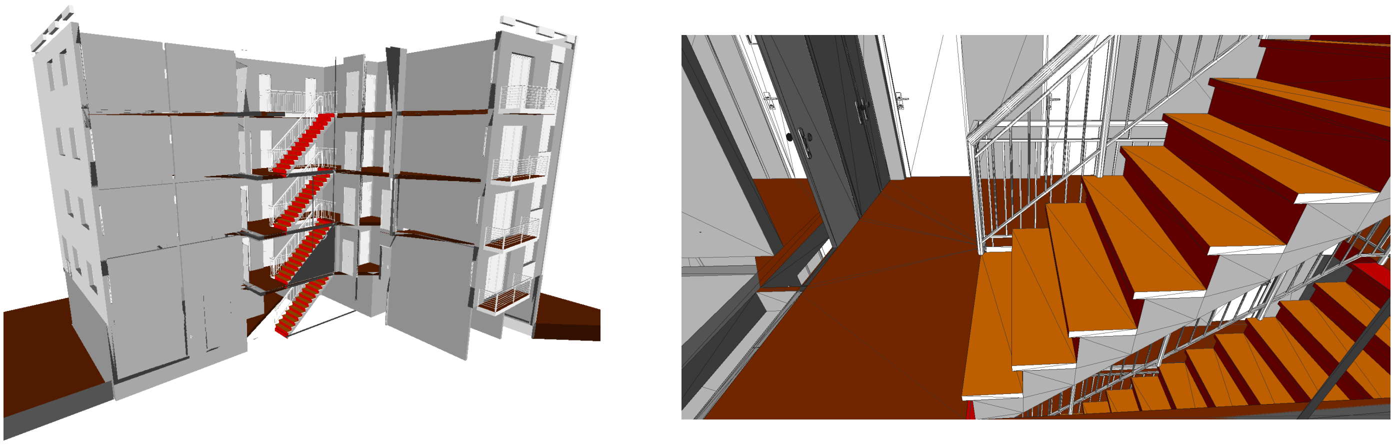


Figure 8: Stairs of the LcD model.

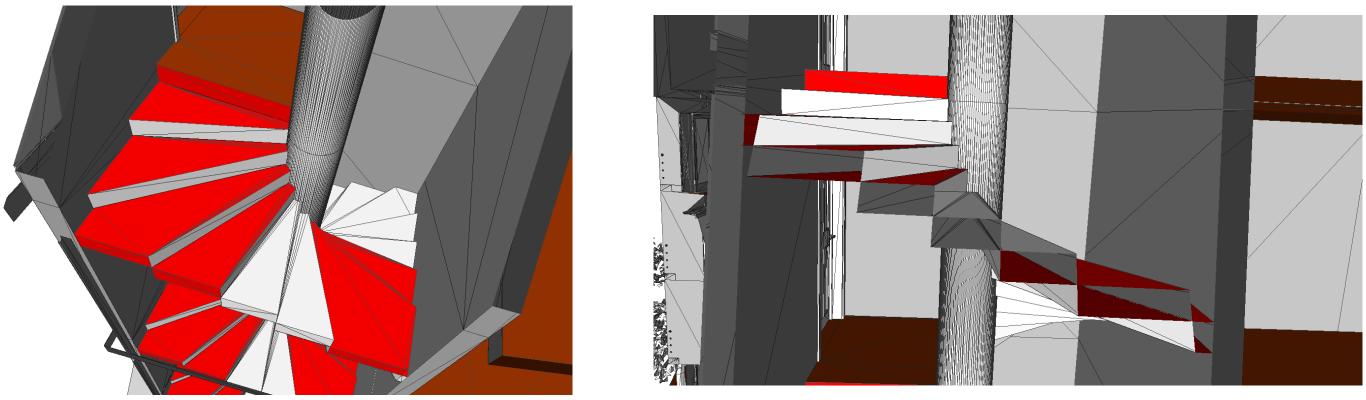


Figure 9: Stairs of the LcF model.

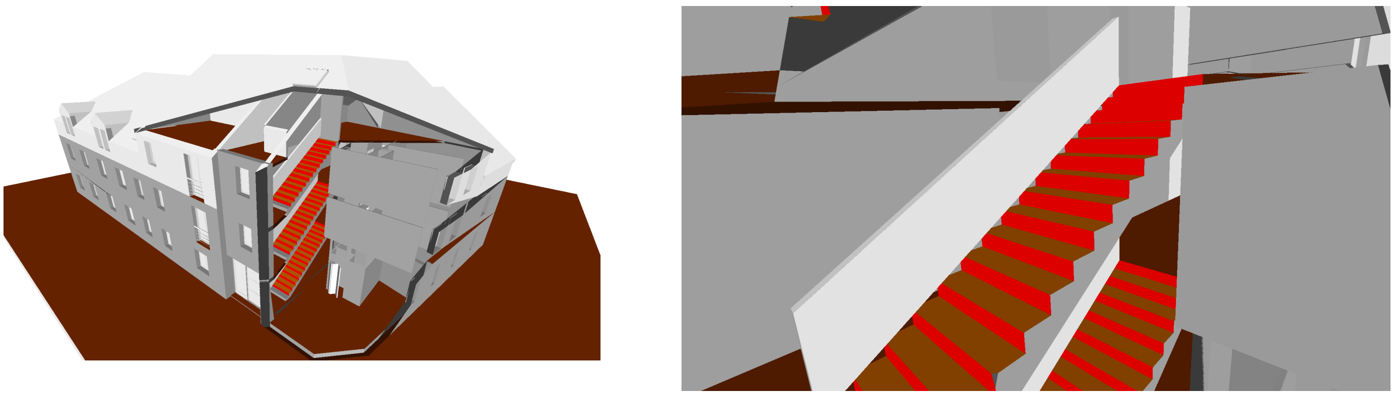


Figure 10: Stairs of the LcG model.

2 Parsing openings

Figure 11 shows a model with a wall, two floors and two openings: a window (hole opening) and a door (border opening). As it is a surface model, there are two polygons associated to a panel (windowpane or door panel), corresponding to each face, and likewise for a main wall. Note that we differentiate wall edges, around an opening, from frame edges, inside an opening.

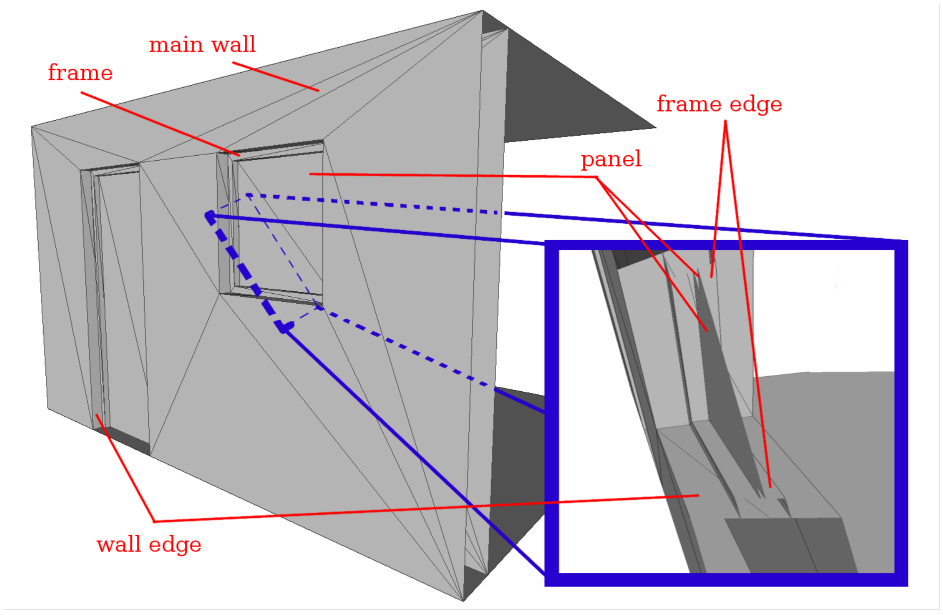


Figure 11: A model to illustrate opening parsing, with a cross-section for the window.

Grammar for openings

The pretty-printed version of the grammar we used for the detection of openings is shown on Figure 12. It actually relies on a few straightforward features that were not presented in the paper:

- p.area is the area of polygon p.
- p.nbHoles is number of holes in polygon p.
- f.bbz is the height, along the vertical axis, of the bounding box of form f.
- adj(f_1, f_2) checks any kind of adjacency (by a vertex or an edge) between forms f_1 and f_2.
- or introduces disjunction in a condition.
- context introduces a *contextual* grammar element: it is not considered a component of the constructed left-hand side and it is not taken into account when checking exclusivity.
- maxconn(X, c, c', x) is as maxconn(X, c, c') with the extra constraint that x should be among any resulting set $\{x_1, \dots, x_m\}$ of connected instances of X. Providing a "first" x allows pruning the search space by just growing a strongly connected component from x.

The use of maxconn here, as opposed to other operators, was motivated by complex geometries with nested frames as well as for sliding doors and windows.

Parse forest of a model with openings

We use the simple synthetic model pictured on Figure 11 to illustrate a parse tree example (see Figure 13). As for stairs, intermediate nodes labeled *OPE* are introduced by operators, here by maxconn. Due to a current limitation in the implementation of maxconn(X, c, c', x), the prototype introduces two parses (with sharing) for each opening, one for each frame side (inside and outside).

```
// A floor is a large enough horizontal polygon.
              floor f  →  polygon p
                            ⟨ horizontal(p),
                              3 <= p.breadth, 3 <= p.length, 7 <= p.area ⟩
// A main wall surface is a high and large enough vertical polygon, adjacent to two floors.
         main_wall w  →  polygon p, context floor f_1, context floor f_2
                            ⟨ vertical(p),
                              1 <= p.bbz, 0.4 <= p.breadth,
                              adj(p, f_1), adj(p, f_2) ⟩
// A wall edge (i.e., a small wall surface around openings) is a thin polygon without holes,
// adjacent to two walls (inside wall and outside wall).
         wall_edge we  →  polygon p, context main_wall w_1, context main_wall w_2
                            ⟨ horizontal(p) or vertical(p),
                              p.length <= 2.7,
                              0.05 <= p.breadth <= 0.5,
                              p.nbHoles == 0,
                              adj(p, w_1), adj(p, w_2) ⟩
// A pre-opening (i.e., a candidate for a main opening component) is a polygon the size of
// an opening.
       pre_opening po  →  polygon p
                            ⟨ vertical(p), 0.5 <= p.bbz,
                              0.2 <= p.length <= 2.7, 0.2 <= p.breadth <= 2.7 ⟩
// A pre-frame (i.e., a candidate for an opening frame) is a pre-opening with either one or
// more large holes (typically for a windowpane), or no hole and a small area compare to
// its bounding rectangle area (typically for a door).
         pre_frame pf  →  pre_opening po
                            ⟨ po.nbHoles >= 1,
                              po.area <= 0.5 * po.length * po.breadth ⟩
                       |  pre_opening po
                            ⟨ po.nbHoles == 0,
                              po.area <= 0.25 * po.length * po.breadth ⟩
// A panel (windowpane or door panel) is a pre-opening with no hole and an area similar to
// that of its 2D bounding box.
             panel pn  →  pre_opening po
                            ⟨ po.nbHoles == 0,
                              0.95 * po.area <= po.length * po.breadth <= 1.05 * po.area ⟩
// A frame edge is a thin polygon whose length is up to that of an opening.
        frame_edge fe  →  polygon p
                            ⟨ horizontal(p) or vertical(p),
                              horizontal(p.lengthVector) or vertical(p.lengthVector),
                              p.nbHoles == 0,
                              p.length <= 2.7, 0.05 <= p.breadth <= 0.5 ⟩
// An opening element is either a panel, a pre-frame or a frame edge
opening_element oe  →  panel pn | pre_frame pf | frame_edge fe
// A frame is pre-frame adjacent to a wall edge.
              frame f  →  pre_frame pf, context wall_edge we
                            ⟨ adj(pf, we) ⟩
// An opening is a connected set of opening elements adjacent to a wall, implemented here
// as a connected set of opening elements based on a frame.
            opening o  →  frame f, maxconn(opening_element, true, adj, f) soe
```

Figure 12: Grammar for openings (units in meters)

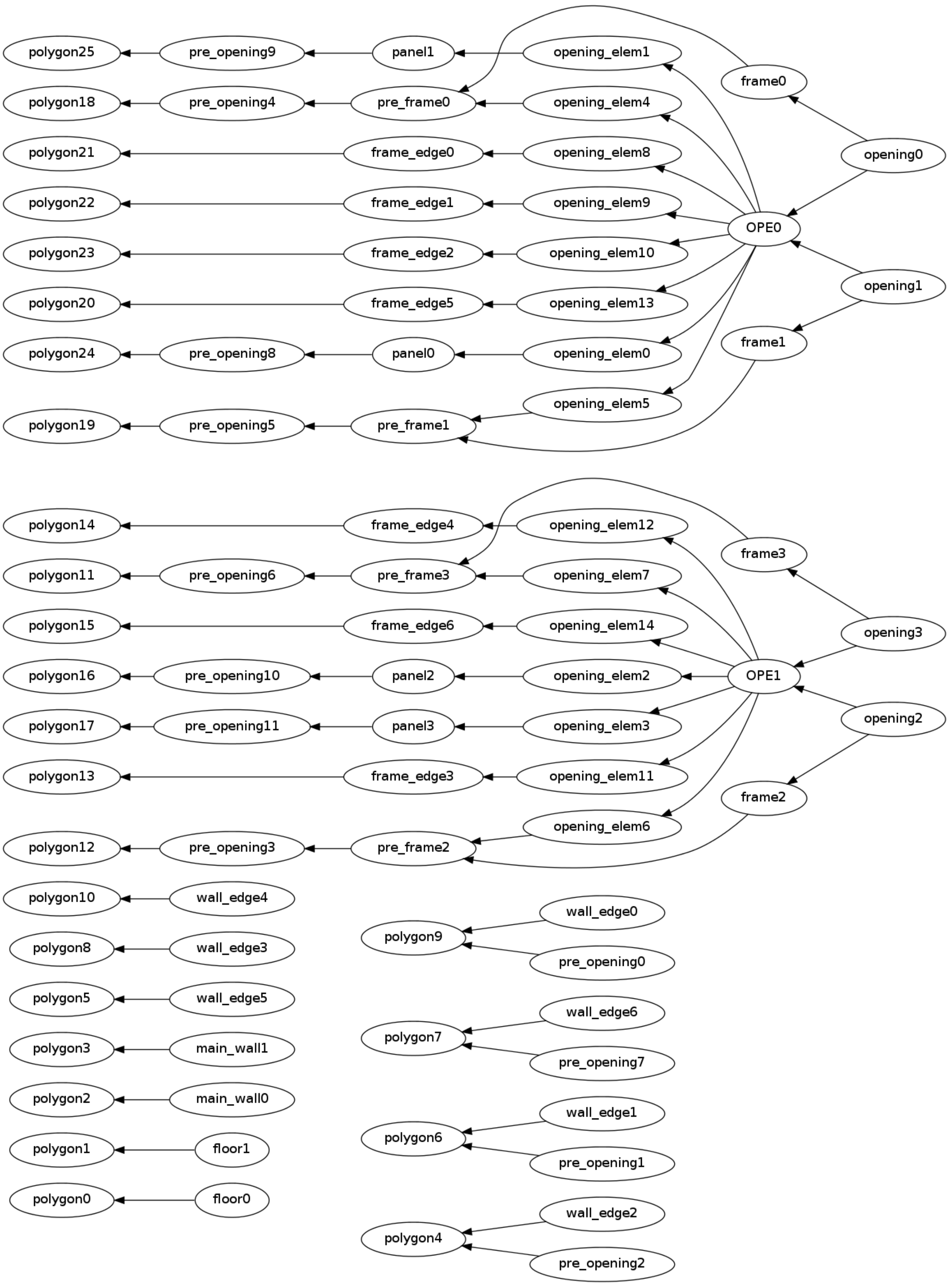


Figure 13: Parse forest of the model pictured on Figure 11, using opening grammar of Figure 12.

Opening parsing examples on CAD models

The paper describes some results of opening parsing on CAD models. They are illustrated here on Figures 2, 2, 2, 18 and 2. See Table 2 in the paper for the color code.

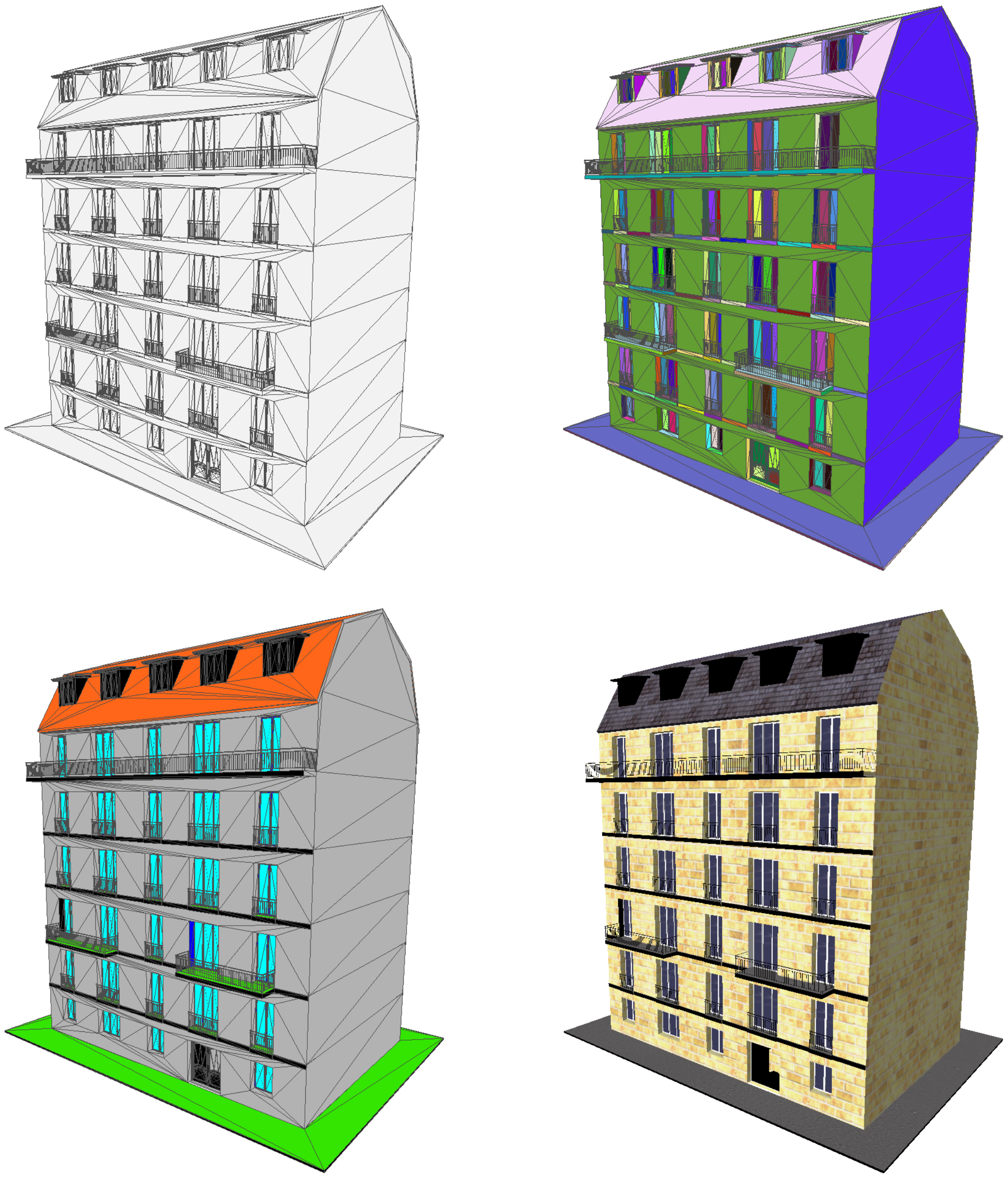


Figure 14: LcA building with, from upper left to lower right, model without semantization, after polygon extraction, after semantization and with texture.

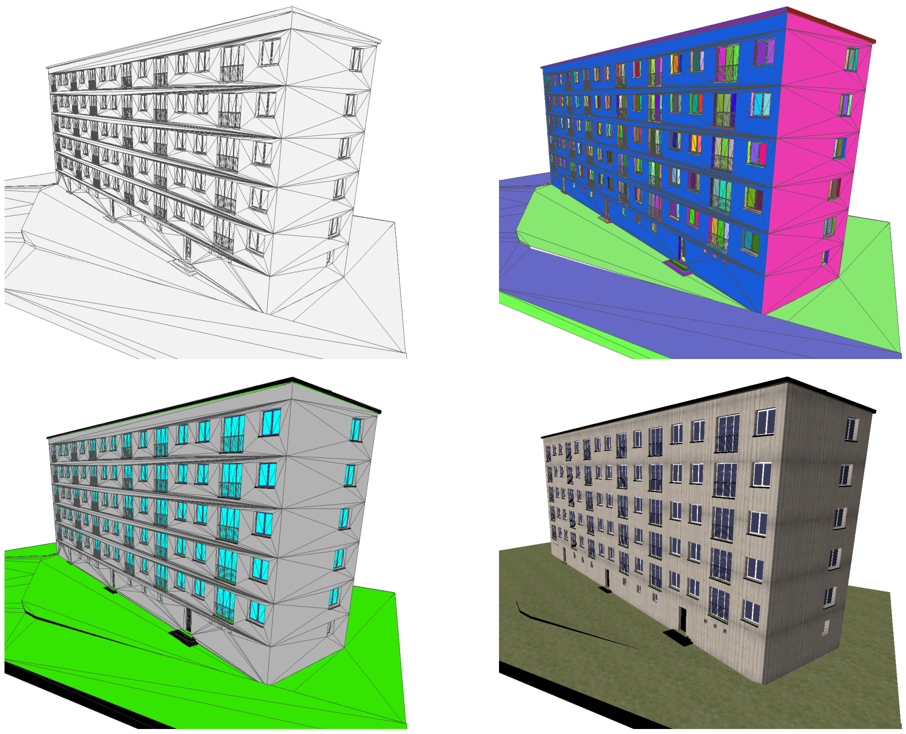


Figure 15: LcC building with, from upper left to lower right, model without semantization, after polygon extraction, after semantization and with texture.

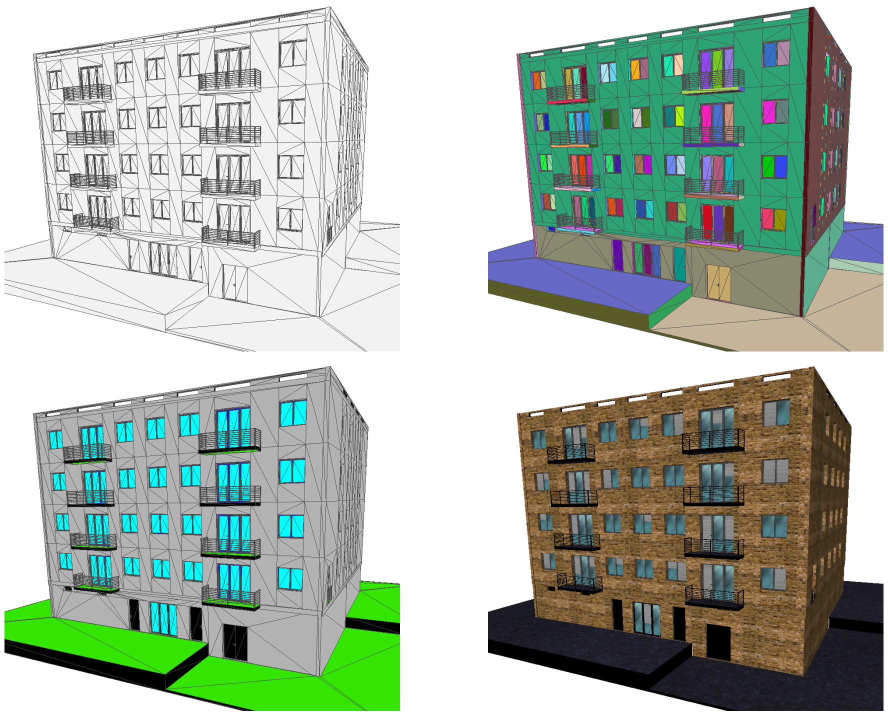


Figure 16: LcD building with, from upper left to lower right, model without semantization, after polygon extraction, after semantization and with texture.

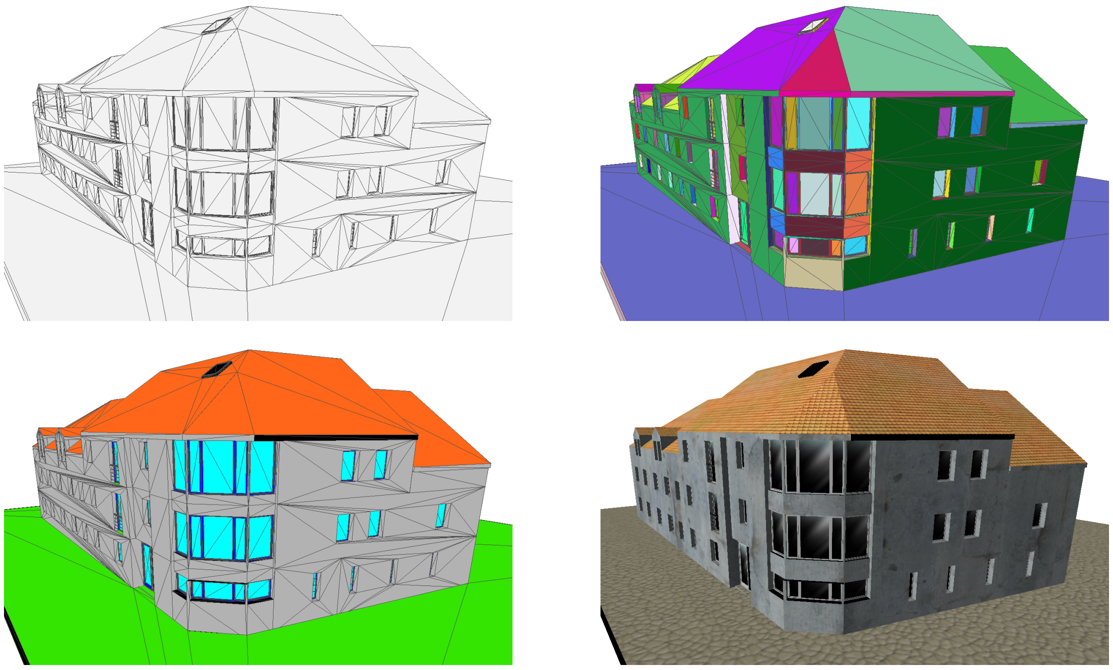


Figure 17: LcG building with, from upper left to lower right, model without semantization, after polygon extraction, after semantization and with texture.

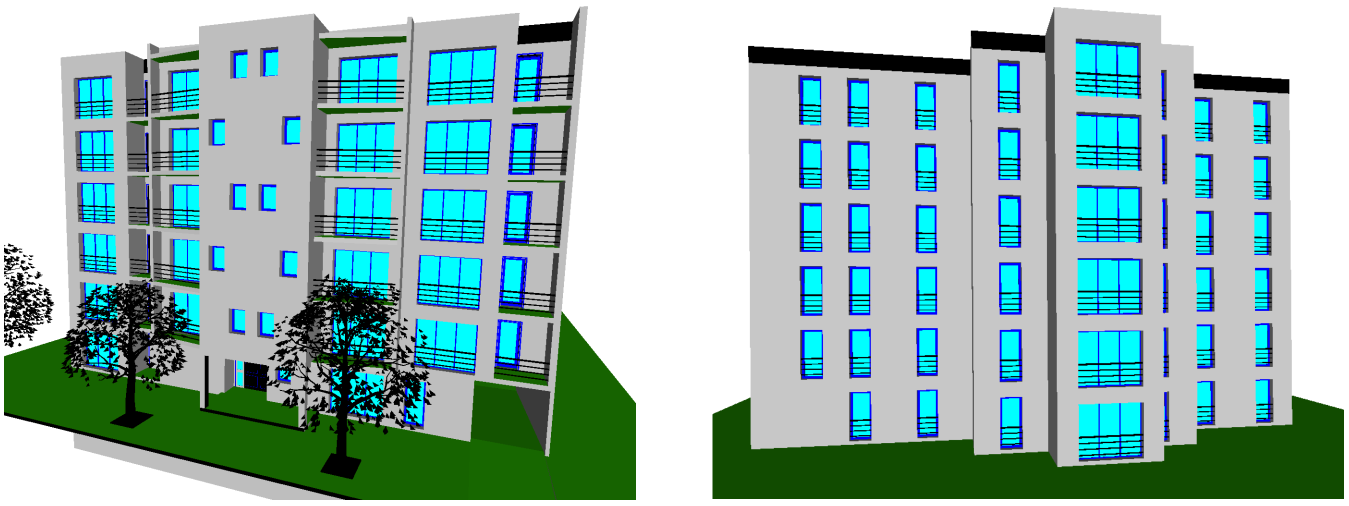


Figure 18: Openings of the LcF model.

3 Issues with the geometry of CAD models

CAD models are not perfect. They may contain errors such as spaces between walls and floors (Figure 19, left) or doors intersecting floors (Figure 19, right). Because of this kind of inconsistencies, adjacencies may be missing or spurious, and the parsing may be wrong or not optimal.

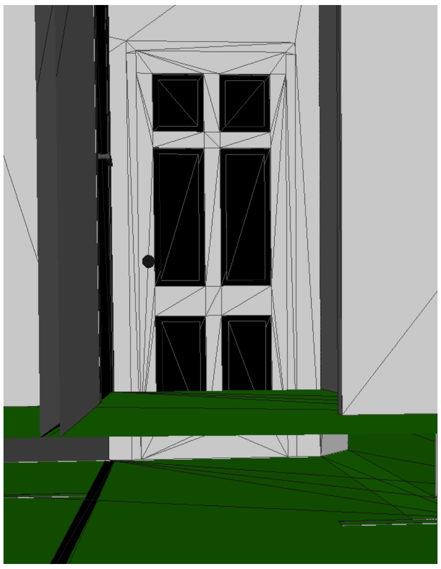


Figure 19: Door going through floor (LcC).

For instance, on Figure 20, the stairway is partly included in the floor, leading to a large polygon for the bottom riser, which is not detected as such.

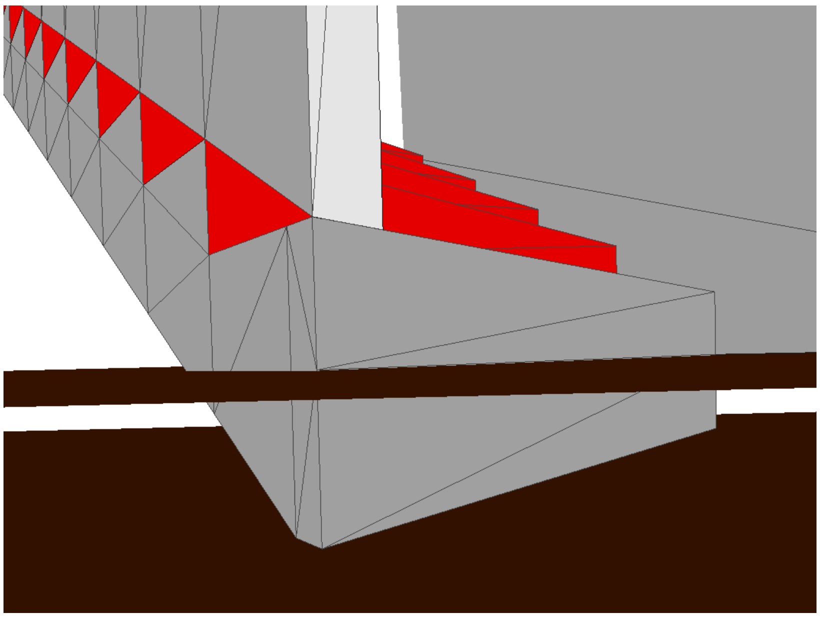
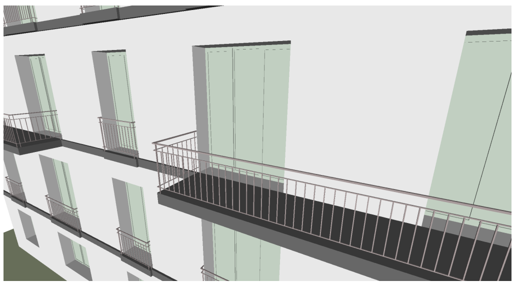
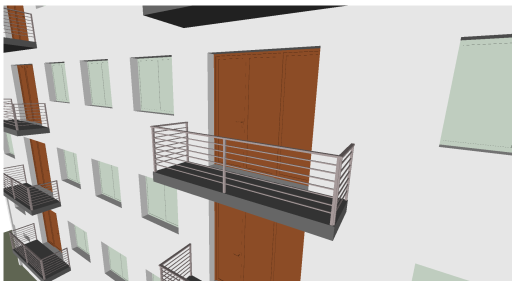


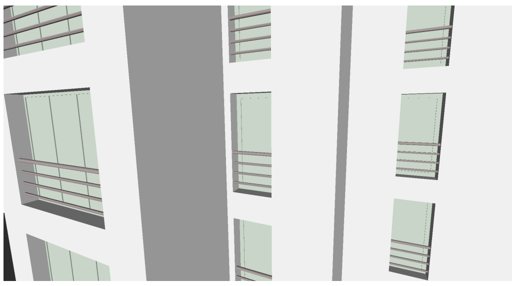
Figure 20: Stairway going through the floor (LcG).

As said in the paper, these geometric issues do not occur with data originating from existing objects as only their surface is visible.

Besides, the building models we used were available in IFC format (Industry Foundation Classes). The IFC label and relate each geometric component of the model. However, there were labeling inconsistencies between different models. For instance, balcony openings (French doors/windows) were not interpreted in the same way by the creator of LcA and the creator of LcC: the former considers them as windows, the latter as doors; the situation is similar for openings with guardrails in models LcF and LcG (see Figure 21).


(a) LcA, window and balcony.


(b) LcC, door and balcony.


(c) LcF, window and guardrail.

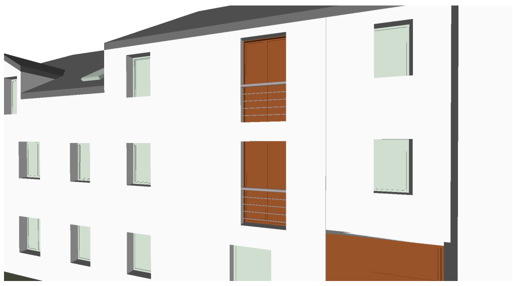

(d) LcG, door and guardrail.

Figure 21: Similar openings with different labels in the original IFC models.

For that reason, we do not try to differentiate doors from windows, and compare with the "ground truth". (Door adjacency to a floor should be enough to differentiate them from windows most of the time.)