# Techniques for improved CityGML models[☆]

Steffen Goebbels and Regina Pohle-Fröhlich[a]

[a] *iPattern Institute, Niederrhein University of Applied Sciences, 47805 Krefeld, Germany*
*{steffen.goebbels, regina.pohle}@hs-niederrhein.de*

**Abstract**

This paper deals with some techniques that can be used to improve quality and geometric consistency of data-driven 3D city models. It focuses on two practical tasks with respect to roof polygons: finding better edges and planarization. To increase the quality of edges, models are merged with true orthophotos and cadastral data. With mixed integer linear programs, the number of right angles and symmetry are maximized. Linear optimization problems also help to correct non-planar roof facets and to avoid artificial step edges within roofs. Since these methods change the positions of vertices, self-intersections of polygons have to be healed, and building walls have to be re-arranged. The techniques focus on roof geometries that can be represented in 2.5D. In general, this is true for CityGML models in level of detail 2 that do not have detailed facades or overhangs. However, techniques can also be adapted to other 2.5D modeling tasks.

*Keywords:* Point clouds, building reconstruction, CityGML, linear optimization, true orthophotos

## 1. Introduction

Cities are often visualized with a textured surface triangulation. However, there is no semantics associated with triangles. To the contrary, CityGML [2] is an XML-based description language for semantic city models. In CityGML, each wall, roof facet, window etc. is defined as an individual planar polygon. This is required for cadastral, planning, simulation, and marketing purposes, see [3]. Typically, CityGML models are generated by using a model or a data driven approach, or by combining both methods (see [4], cf. [5, 6, 7, 8]). Model driven approaches use a library of parameterized standard roofs. To this end, a building is segmented into small atomic parts, for example by extending footprint edges. Then, for each part, the parameterization of a standard roof is selected which fits best to the point cloud. This approach might fail for complex buildings like churches, and small structures like dormers and chimneys do not find their way into these building models but might lead to a wrong choice of parameters like roof slopes.

Data driven methods detect planes or other geometric primitives and combine even small roof facets to complete roofs (see e. g. [9, 10]). Due to low resolution of airborne laser scanning or artifacts in photogrammetric point clouds, data-driven CityGML models often violate planarity requirements and might have noisy step edges. Orthogonality of real building edges might not be correctly modeled.

The appearance of models can be improved either during the original model creation workflow or later based on existing models. Examples for improving model quality within the creation workflow are given by Wichmann and Kada in [11] and by Demir et al. in [12]. Wichmann and Kada segment similar dormers in a point cloud. Then

[☆]A short version of some parts of this paper has already been published in the proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP), Porto, 2017, see [1]. This article has been accepted for publication in Graphical Models Journal, see https://doi.org/10.1016/j.gmod.2019.101044 corresponding point cloud regions are merged to enhance density. A prototype dormer model can be generated on the dense cloud. Demir et. al. describe a procedure to detect repeating structures in point clouds.

This paper's techniques can be applied both during model creation and on existing models. They adjust roof edges and establish planarity of roof facets. Edges are re-positioned based on domain knowledge, cadastral data and true orthophotos as additional (optional) information, see Section 3. Domain knowledge, for example, is that the most likely angle between two building edges is the right angle. Often structures are rectangular or show symmetry.

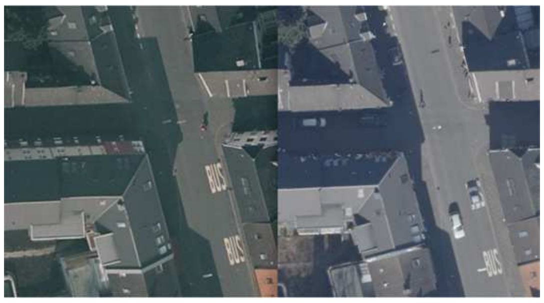


Figure 1: Orthophoto (left) vs. true orthophoto (right) [1]

Throughout this paper we are concerned with CityGML models that are given in a level of detail (LoD) 2, i.e., they consist of somewhat detailed roof structures but only straight walls without any facade information like windows or doors. Such 2.5D models can be represented by roof facets and a ground plane, wall facets can be automatically added.

A geometrically corrected areal photo, fitting with the coordinate system at ground level, is called orthophoto. In a true orthophoto, not only the ground level fits with the coordinate system but also all levels above the ground including the roof, see Figure 1. We detect edges of such photos and adjust models according to these edges as well as to footprint edges from cadastral data, see Section 3.1.

To establish right angles, we detect rectangles (Section 3.2) or solve an optimization problem with a mixed integer linear program. This technique can also be used to re-arrange edges in order to increase a roof's symmetry, see Section 3.3.

Changes to the roof layout might result in self-intersecting polygons. Such error situations have to be healed afterwards, see Section 3.4.

The given paper is not only concerned with better modeling structures but also with improving geometric consistency, cf. [13, 14]. It focuses on planarity of polygons. One can apply general methods for repairing arbitrary 3D polygon models. But most building models have vertical walls, so that one only has to deal with roof structures that can be projected to 2D. Zhao et al. [14] propose an approach that corrects multiple types of geometric error using shrink-wrapping on a watertight approximation of the tessellated building. Alam et al. [15] describe methods to correct single geometric problems of existing CityGML models.

We present a new algorithm to heal non-planarity of roof facets and to avoid artificial step edges. CityGML requires planarity, see [2, p. 25]. To some extend, many city models violate this requirement, cf. [16, 17]. This is regarded as difficult to correct. We fix missing planarity with a surprisingly simple but powerful linear program in Section 4. Missing planarity often occurs because artificial step edges should be avoided in roofs. Models with artificial step edges can also be improved. Often, the edges can be removed by solving a similar linear optimization task, see Section 4. The algorithms of this section only adjust $z$-coordinates (height values). Therefore, no further self-intersections of polygons can occur. But the algorithms might change slopes of roof facets. This in turn influences visibility of walls. Also, rearrangement of roof edges (Section 3) leads new wall positions. In Section 5 we describe a wall processing step that has to run after all these modifications. By utilizing the 2.5D structure of models, it fulfills the CityGML requirement that roof and wall polygons completely belong to a building's outer hull.

## 2. Reference data set

We applied presented techniques to a square kilometer of the city center of the German city of Krefeld with 3987 buildings that we generated with our data-driven workflow [18] from a sparse point cloud with five to ten points per square meter, see Figure 2. The area corresponds to the UTM interval

$$[32330000, 32331000] \times [5689000, 5690000].$$

If nothing else is said, results were obtained for this data set.

To test our planarization approach (Section 4), we also consider the larger 16 square kilometer interval $[32329000, 32333000] \times [5687000, 5691000]$ and also use a model-driven CityGML model of North Rhine Westphalia that is freely available [19]. For Krefeld's city center square kilometer, this model contains 41496 different LoD 2 roof vertices that define 4102 buildings. Despite its model-driven origin, it also shows (few) non planar roof facets, probably because of rounding errors.

Additionally, we worked with a city model of the city of Leverkusen, generated with the same data-driven algorithm that in fact requires quality improvement post-processing.

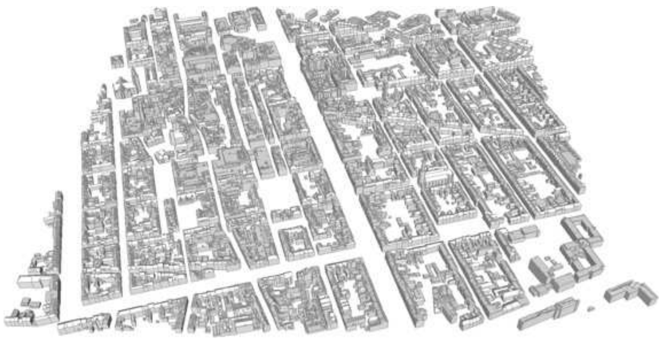


Figure 2: City center as reference data set

## 3. Adjusting roof edges

### *3.1. Merging edge data from different sources*

Either from an existing 3D model or based on point clouds during initial model generation, we compute a 2D raster map in which pixel colors refer to corresponding roof facets, see Figures 3–5. One pixel of the raster image corresponds with an area of $10 \times 10$ cm$^2$. This fits with typical data precision. When not dealing with existing models, there might also be gaps in the raster map. The idea is to improve a roof's layout by fitting border contours of roof facets to auxiliary lines. Some auxiliary lines can be computed directly from a given model or from detected planes. Intersection lines of roof planes correspond with ridge lines, see Figure 3. Cadastral footprints of

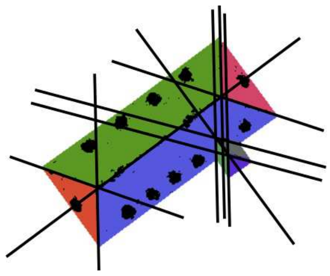


Figure 3: Intersection lines between adjacent roof planes

building parts (see Figure 4) and edges of true orthophotos are additional information. If available,

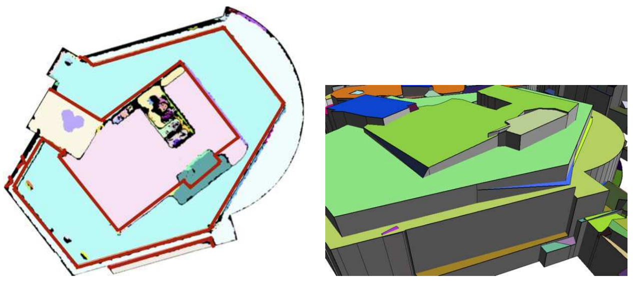


Figure 4: Raster maps of roof facets (left): Red cadastre polygons are borders of building parts, black areas could not be associated with roof facets. The picture on the right hand side shows the derived model [1].

one can generate a true orthophoto from overlapping areal images - for example using the Structure from Motion algorithm. Our pictures were computed from such overlapping photos. An alternative method is to improve orthophotos with the help of terrain and 3D building models. Vice versa, there also exist various approaches to combine LIDAR point clouds with information from areal images to improve 3D building modeling. For example, Tong et. al. [20] use edges of (true) orthophotos to adjust step edges of flat roofs in a city model that is based on a Delauny triangulation. Arefi and Reinartz [21] decompose buildings and fit parametric roof models according to ridge lines detected from true orthophotos.

We use true othophotos that have the same resolution as the roof's 2D raster map. Although the tested photos originate from areal images with only 60% horizontal and 50% vertical overlap, the images fit well with models, cf. Figure 5. However, edges tend to be not exactly straight. Also, a minor problem is that roofs may have textures showing tiles or tar paper with many edges. Unfortunately, shadows lead to more significant additional edges. A lot of research deals with re-

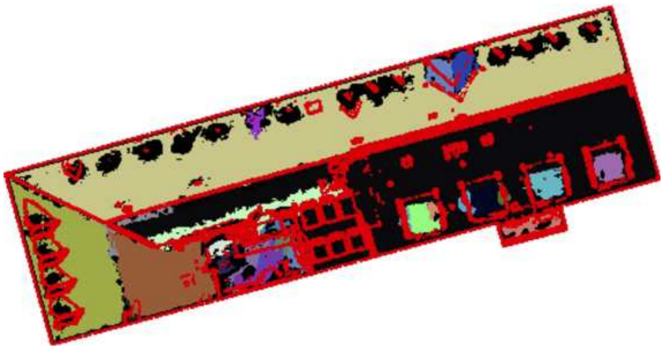


Figure 5: Red pixels belong to edges of a true orthophoto. Colored areas belong to planes that were detected in a model generation workflow. Both data sources match quite well. Boundaries of larger dormers can be adjusted to the edges [1].

moval of shadows, cf. [22] and the literature cited there. However, our experiments with the algorithm of [23] did not lead to sufficient results. A reason might be the variety of colors of our arial photos.

To find relevant edges with Canny edge detector, one has to individually find a threshold that excludes irrelevant edges from textures. Instead, we generate a kind of principal curvature image without the need to select a threshold value, see Figure 6. After anisotropic filtering, we convert the orthophoto to a grey image and compute second partial derivatives for each pixel position in terms of a Hessian matrix, a symmetric matrix with real eigenvalues. Then we filter for pixel positions for which the matrix has a locally largest absolute eigenvalue, i.e., in terms of absolute values, the eigenvalue has to be larger than the eigenvalues of Hessian matrices of the two horizontal, vertical or diagonal neighbors. At such positions, an eigenvector $\vec{d_\lambda}$ of the eigenvalue $\lambda$ with largest absolute value $|\lambda|$ points into the direction of largest absolute curvature. It is orthogonal to an eigenvector $\vec{d_\mu}$ of the eigenvalue $\mu$, $|\mu| \leq |\lambda|$. If additionally $|\lambda| > 0$ and $|\mu| \approx 0$, then curvature is large in one direction only and $\vec{d_\mu}$ is orthogonal to that direction. Thus, $\vec{d_\mu}$ is parallel to an edge. We mark the corresponding position in an image $I_e$ of edges. Typically, the Hessian matrix is used to find positions like corners, where local geometry changes in two directions. We use it differently to detect changes that only occur in one direction. As a side effect, this direction can be taken to filter for edges that are parallel or orthogonal to segments of the building's footprint.

Edges might be not connected in $I_e$. We connect them by computing the principal curvature image in a higher resolution. A reduction of the resolution (supersampling) then closes most gaps.

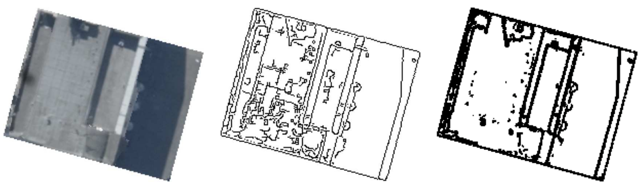


Figure 6: Results of Canny edge detector (middle) vs. our curvature based method (right) [1]

Once we have derived auxiliary lines, we eliminate noise near computed and detected edges, especially near pixels of $I_e$. Noise consists of raster map pixels indicating a roof facet at the wrong side of a line. To this end, we remove pixels that represent points closer than 30 cm to these lines, depending on the type of line. Then, in a first pass of region growing, we expand each colored area into the free space that has been created by removing pixels of the area's color. In doing so, we do not cross the lines under consideration. The outcome is that pixels on wrong sides of lines are deleted. This gives straight boundaries at ridge lines, at edges of building parts and at edges of the true orthophoto. However, by deleting pixels, facets might become unconnected. Such facets have to be split up into separate connected components.

In a second pass of region growing, we let the colored regions grow until they collide or reach one of the previously discussed lines or pixels of $I_e$. Additionally, large height jumps (step edges) limit region growing. There might be step edges with small height differences in a given model. Such edges might be either accurate or an artifact. Here, the edges from orthophotos come into play.

A third pass of region growing is needed to take care of areas that are still not colored. In this pass, only other regions and the footprint from cadastral data or a given CityGML model act as boundaries. Since we use region growing, wrongly placed auxiliary lines do not do much harm. This especially is true for shadow edges.

From the new raster image of the roof, a 3D building model can be easily reconstructed. Plane equations are known. Edges between roof facets correspond with contours in the raster image, and they can be adjusted by snapping to auxiliary lines (cf. [18]).

Figure 7 shows excerpts from a city model before planarization is performed (see Section 4). Region growing bounded by edges of the true orthophoto improves the original model. However, due to the resolution of data and quality of true orthophotos, the outcome is far from being perfect. But it is a good basis to apply heuristics that lead to parallel or orthogonal edges, see next Sections 3.2 and 3.3.

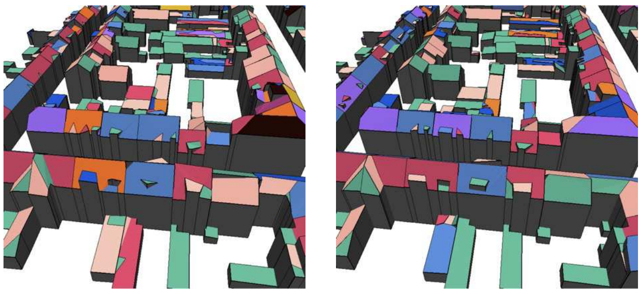


Figure 7: Boundaries of the left model are adjusted to edges of a true orthophoto. This results in the second picture. Due to the region growing algorithm, some small roof segments vanish, other small facets appear [1].

## 3.2. *Rectangle estimation*

Since buildings typically are composed from rectangular structures, we detect and maintain such rectangular shapes in the fashion of a hybrid data and model driven approach. Figure 10 shows first results. In contrast to [24], we do not directly fit geometric objects to the 3D point cloud but use the 2D raster representation in two ways. We search for roof shapes that are approximately rectangular, but we also look for rectangular regions that are bounded by step edges. Such areas might be footprints of building parts like towers or courtyards that rise higher or lower than the surrounding roofs.

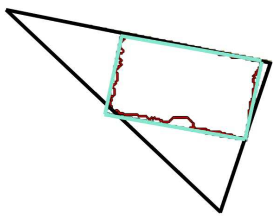


Figure 8: Detection of rectangular shapes by comparing contours (red) with enclosing triangles and rectangles.

To find 2D rectangular structures, we compute convex hulls of boundary contours and compare their area with the area of their smallest enclosing triangles, circles and rectangles, see Figure 8. If the rectangle area is closest to the convex hull's area and the difference is below a threshold value, then the contour might be rectangular. Due to experimental results, we also consider the intersections between these candidates to select rectangle candidates that are suitable for alignment. As a heuristics, we consider a candidate contour to be a suitable alignment rectangle if and only if one of the following two cases is at hand, see Figure 9:

- The candidate's enclosing rectangle intersects with the enclosing rectangle of another candidate so that the intersection region is a polygon with four vertices and has an area larger than a threshold size, see middle case in Figure 9. Then both candidates are considered. They are modified by using a common edge that avoids the intersection. This edge lies on the intersection line between the roof planes of the two regions. By using this rule, gable roofs get precise ridge lines.

- The candidate's enclosing rectangle is completely included in another enclosing rectangle. This is the case with dormers. Both rectangles might or might not have a common edge section, see right case in Figure 9.

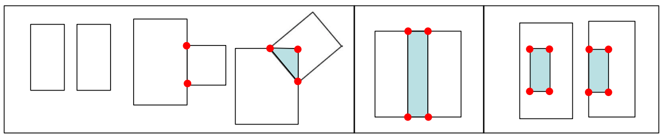


Figure 9: Selection of rectangle candidates: We do not consider rectangles that do not have a significantly large intersection area with another candidate rectangle. The left part of the figure shows three such situations. We do consider rectangles with intersection areas as shown in the middle and right part of the figure.

We slightly adjust the orientation of alignment rectangles to match with long footprint edges. For each alignment rectangle, the boundary polygon of its corresponding roof facet is replaced by the rectangle (Figure 10). One method to integrate the rectangles into the graph of roof edges is to use a raster representation as described in the previous section. All further model repair steps should preserve these rectangular facets.

Figure 11 shows typical outcomes for dormers.

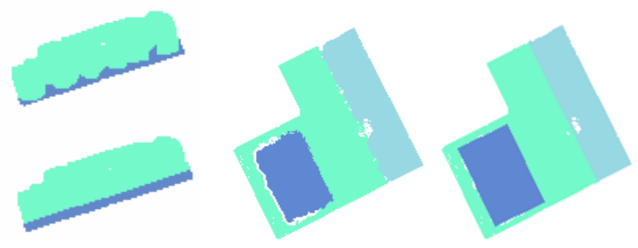


Figure 10: Preserving rectangular shapes: raster representation of roof facets without and with shape estimation.

### 3.3. Optimization of angles

An alternative approach to fitting geometric structures is to arrange planar wall and roof segments so that many become pairwise co-planar or orthogonal (cf. [25]). This corresponds with modifying detected 2D raster map contours so that as many edges become orthogonal or parallel to footprint edges as possible. In CityGML, a buildings's footprint is defined by its ground plane or terrain intersection curve. The task can be formulated as a mixed integer linear optimization problem with restrictions that maintain the buildings footprint and roof topology, cf. [26]. We give a short outline of such a linear program and start with some notations that are summarized in Figure 12. Let $D$ be a set that contains normalized direction vectors $d = (d.x, d.y)$ of footprint edges that are longer than a threshold length of 2 m, $\sqrt{(d.x)^2 + (d.y)^2} = 1$. $D$ also contains directions that are orthogonal to such long footprint edges.

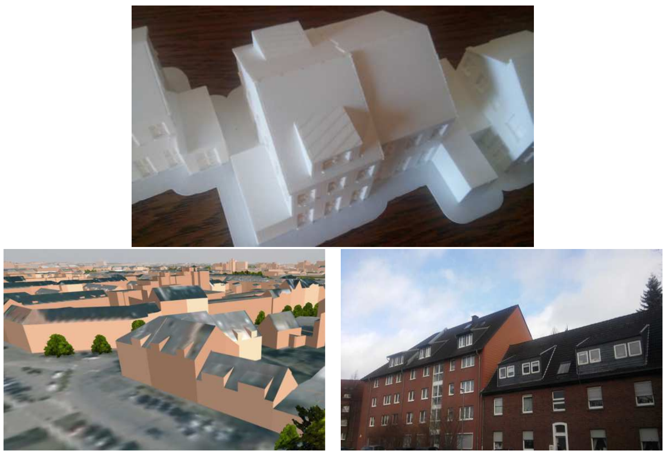


Figure 11: Top to bottom: 3D print of a building with rectangular dormers, a model with corrected rectangular dormers and the original buildings

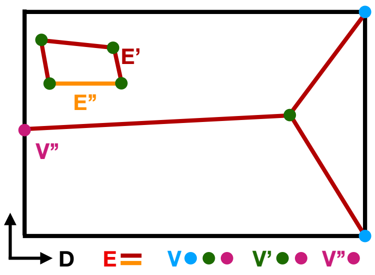


Figure 12: Notations for optimization problem: The roof graph consists of two connected components that are processed separately. $E = E' \cup E''$ ist the set of edges, $V'' \subset V' \subset V$ are sets of vertices, respectively. The building's footprint leads to two direction vectors in $D$.

We obtain a planar graph by projecting roof edges to the $x$-$y$-ground plane. Then we compute connected components of this graph. We will change $x$-$y$-coordinates slightly to get orthogonal edges. But such changes might propagate through

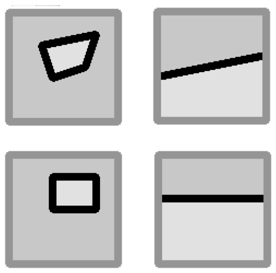


Figure 13: Two scenarios for orthogonalization: The upper row shows the initial layout. The second row visualizes the outcome of optimization.

the corresponding connected component. Therefore, we formulate separate optimization problems for separate components. With $E$ we denote the set of all 2D edges

$$e = (e_1, e_2) = ((e_1.x, e_1.y), (e_2.x, e_2.y))$$

with non-zero length of a connected component that additionally fulfill restrictions:

- An edge has to possess at least one vertex that does not also belong to the footprint.
- An edge is not completely covered by edges of inner or outer footprint polygons.

We use a partitioning $E = E' \cup E''$. The sets $E'$ and $E''$ are disjoint, $E''$ is the set of all edges that are a-priori orthogonal to a vector of $D$. In what follows we allow edges in $E'$ to change their direction. Edges in $E''$ must keep their direction.

Let $V$ be the set of vertices belonging to edges in $E$ and let $V' \subset V$ be the set of vertices that are not used in footprint polygons and that are no intersection points of estimated roof's ridge lines. We allow position changes only for vertices in $V'$. But there still might exist vertices in $V'$ that are positioned on the interior of a footprint edge. We have to keep them on this edge. $V'' \subset V'$ is the set of these vertices.

We model vertex changes with float variables. For each $v \in V$, variables $x_v^+$, $x_v^-$ and $y_v^+$, $y_v^-$ represent non-negative coordinate changes. Since a change for $v \in V \backslash V'$ is not allowed, corresponding variables are fixed set to zero.

We introduce binary variables $x_{e,d}$ that indicate, if an edge $e \in E'$ becomes orthogonal to a direction vector $d \in D$:

$$x_{e,d} = \begin{cases} 1 & : \ e \text{ becomes orthogonal to } d \\ 0 & : \ \text{else} \end{cases} .$$

The proposed mixed integer linear program maximizes an objective function that implements the primary goal to find a maximum number of orthogonal edges. As a secondary aim, coordinate changes have to be minimal:

$$\text{max.} \left( \sum_{e \in E'} \sum_{d \in D} x_{e,d} \right) - \frac{1}{8 \cdot |V'| \cdot \varepsilon} \sum_{v \in V'} \left( x_v^+ + x_v^- + y_v^+ + y_v^- \right). \quad (1)$$

$|V'|$ denotes the number of elements of $V'$. Factor $\frac{1}{8 \cdot |V'| \cdot \varepsilon}$ is used to separate primary and secondary goal. It ensures that coordinate changes contribute significantly less than one binary variable does.

We seek a maximum subject to following restrictions:

If an edge $e \in E'$ is marked as modified by $x_{e,d} = 1$, it must be orthogonal to $d \in D$:

$$\begin{aligned} & -M \cdot (1 - x_{e,d}) \\ \leq \ & (e_2.x + x_{e_2}^+ - x_{e_2}^- - e_1.x - x_{e_1}^+ + x_{e_1}^-) \cdot d.x \\ & + (e_2.y + y_{e_2}^+ - y_{e_2}^- - e_1.y - y_{e_1}^+ + y_{e_1}^-) \cdot d.y \\ \leq \ & M \cdot (1 - x_{e,d}), \end{aligned}$$

where constant $M$ is larger than the longest occurring edge.

We need further restrictions to avoid changes of vertices on the footprint and of intersection points of ridge lines. For all $v \in V \setminus V'$ we require

$$x_v^+ = x_v^- = y_v^+ = y_v^- = 0. \quad (2)$$

Other vertices $v \in V'$ are allowed to move within a threshold distance $\varepsilon > 0$:

$$0 \leq x_v^+, x_v^-, y_v^+, y_v^- < \varepsilon. \quad (3)$$

However, vertices $v \in V''$ have to stay on their footprint edges with 2D vertices $a_v$, $b_v$. This is the case for the right example in Figure 13. We model this with a parameter variable $0 \leq r_v \leq 1$:

$$x_v^+ - x_v^- - (b_v.x - a_v.x) \cdot r_v \ = \ a_v.x - v.x$$

$$y_v^+ - y_v^- - (b_v.y - a_v.y) \cdot r_v \;=\; a_v.y - v.y. \tag{4}$$

Finally, we also keep the direction of edges that already are orthogonal to a $d \in D$: For all $e \in E''$ we require

$$\begin{aligned}(x_{e_2}^+ - x_{e_2}^- - x_{e_1}^+ + x_{e_1}^-) \cdot (e_1.y - e_2.y) & \\ +(y_{e_2}^+ - y_{e_2}^- - y_{e_1}^+ + y_{e_1}^-) \cdot (e_2.x - e_1.x) &= 0.\end{aligned} \tag{5}$$

In general, mixed integer linear programs are NP-hard. However, we can reduce the degrees of freedom, if we only allow $x_{e,d}$ to be one for edges $e$ that are roughly orthogonal to CityGML footprint directions. Then for our test data set, the number of non-fixed binary variables is bounded by 1500. Also, we can define a time limit for processing. More than 95 % of buildings can be optimized within two seconds with the GNU Linear Programming Kit library GLPK [27] on one kernel of a 2.4 GHz i5 processor[1]. If applied to the result of rectangle estimation (Section 3.2), the optimization algorithm still changes a median value of three vertices per building. Thus it improves the results of simpler rectangle estimation slightly because additionally corners are considered that do not belong to rectangles. It also corrects quantization errors that result from working on a raster map. The rectangle estimation procedure can be omitted if using the optimization step.

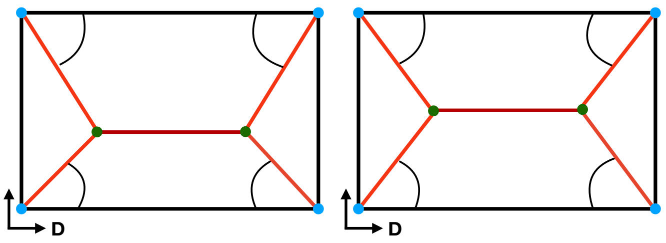


Figure 14: Improving symmetry by considering red edges with similar length and angle.

We focus on orthogonality, but it is also easy to use classical linear programming to improve symmetry of roof models. If edges $e = (e_1, e_2)$, $f = (f_1, f_2) \in E'$ with vertices $e_1, e_2, f_1, f_2 \in V$ have lengths

$$\|e\| = \sqrt{(e_2.x - e_1.x)^2 + (e_2.y - e_1.y)^2}$$

and $\|f\|$ within a threshold distance and if there exists a footprint direction vector $d$ so that both edges have nearly the same angles $\beta_1$ and $\beta_2$ with $d_1 = \pm d$ and $d_2 = \pm d$, respectively, then lengths as well as angles should become equal. Figure 14 shows a graph with four edges that pairwise fulfill such conditions. For each such pair of edges $e$ and $f$ the difference

$$\begin{aligned}&\cos(\beta_1)\|e\| - \cos(\beta_2)\|f\| = \\ &\quad (e_2.x - e_1.x) \cdot d_1.x + (e_2.y - e_1.y) \cdot d_1.y \\ &\quad -(f_2.x - f_1.x) \cdot d_2.x + (f_2.y - f_1.y) \cdot d_2.y\end{aligned}$$

should be zero. To obtain a linear minimization problem, we introduce non-negative variables $\delta^+ \geq 0$ and $\delta^- \geq 0$ such that

$$\begin{aligned}&\delta^+ - \delta^- = \\ &\quad (e_2.x + x_{e_2}^+ - x_{e_2}^- - e_1.x - x_{e_1}^+ + x_{e_1}^-) \cdot d_1.x \\ &\quad +(e_2.y + y_{e_2}^+ - y_{e_2}^- - e_1.y - y_{e_1}^+ + y_{e_1}^-) \cdot d_1.y \\ &\quad -(f_2.x + x_{f_2}^+ - x_{f_2}^- - f_1.x - x_{f_1}^+ + x_{f_1}^-) \cdot d_2.x \\ &\quad +(f_2.y + y_{f_2}^+ - y_{f_2}^- - f_1.y - y_{f_1}^+ + y_{f_1}^-) \cdot d_2.y.\end{aligned}$$

Now one can minimize the sum of expressions $\delta^+ + \delta^-$ belonging to all suitable pairs of edges. To maintain the overall shape of the roof, restrictions (2)–(5) are required as before. Prior to solving this problem (for example with the simplex algorithm as implemented in GLPK), one can first improve orthogonal structures by computing (1). Condition (5) maintains these structures.

While we have been concerned with aligning roof edges of LoD 2 building models, these techniques can also be applied to beautification of LoD 3 models. A deep learning based method for adding facade details based on optimized LoD 2 models is described in [28]. To cluster and align windows and doors, the method also uses a mixed integer linear program.

Coordinate changes due to the solutions of optimization problems might lead to self-intersections of polygons. Before we can apply further

[1]It is well-known that for mixed integer problems commercial solvers like IBM's ILOG CPLEX or Gurobi and the non-commercial solver SCIP outperform GLPK.

linear programs to correct roof facets' planarity, we eliminate such errors as described in the next paragraph.

### *3.4. Correcting self-intersections*

Each roof facet consists of exactly one outer polygon and zero or more inner polygons that define openings within the roof. The outer polygon is oriented counter-clockwise, the inner polygons are oriented clockwise. CityGML does not allow polygons to have self-intersections. However, such intersections often do occur because of the roof's topology or because of over-simplification. Alignment to auxiliary lines in Section 3.1 and the linear programs in Section 3.3 might lead to self-intersections. These algorithms do not process single polygons but the graph of all roof edges.

Automatic repair of single 2D polygons is a general task not restricted to building modeling. Ledoux et. al describe a general method based on triangulation of the interior in [29] and give some literature overview, see also [30]. We use a simple, specialized method for polygons that work on a watertight 2.5D roof polygon mesh. The method recursively splits up polygons to new polygons until all intersections are eliminated. Then it removes intersection-free new polygons that are not needed for a watertight roof. We do this in the $x$-$y$-plane and add height values ($z$-coordinates) later. To begin with, we compute all points of self-intersection. If an intersection point is no known vertex, then we add it as a new vertex to the polygon, cf. [31]. But we also add it to all other polygons that cross that point. This is necessary to easily decide about wall visibility, see Section 5. Now, we have to split up each polygon at vertices that occur more than once within the polygon.

To simplify the outer polygon, we have to distinguish between two general cases of self-intersection at a vertex: The polygon might really cross itself (case a) ) or it is tangent to itself (case b) ). We resolve intersections using two passes. The first pass handles case a), the second one deals with b):

a) This is an error situation that might occur if one snaps vertices to other places like ridge lines. For example, vertex $A$ in Figure 15a) might have been moved upwards. After dividing the polygon into non-self-intersecting segments, all segments with clockwise orientation (like the dark green triangle in the figure) have to be removed. The space covered by this segment might also be shared with multiple adjacent roof facets. To heal this error in practice, it is often sufficient to snap the wrong segment's vertices to the nearest vertex of the segment's longest edge.

b) This is a regular situation. As in a), we split up into non-self-intersecting polygons. We discard polygons with zero area that occur if there are edges that are used twice (different directions of traverse, see edges between vertices $A$ and $B$ in the second example of Figure 15b)). All segments with counter-clockwise orientation become new outer polygons. The second example of Figure 15b) is resolved to two outer polygons. All segments with clockwise orientation become additional new inner polygons. This happens in the first example of Figure 15b).

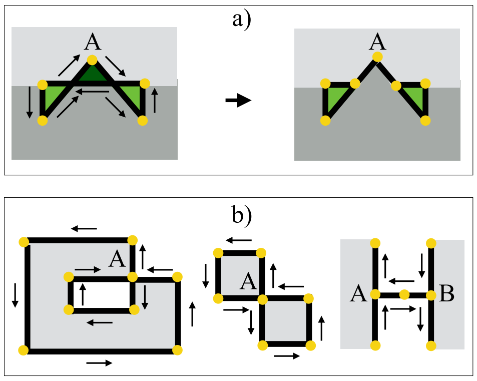


Figure 15: Resolving self-intersections [1].

To simplify previously existing inner polygons, we first merge them if possible: We compute a list of all edges of all inner polygons. Then we remove edges that occur twice but with different direction of traverse. We put together remaining edges to new inner polygons. Then we split them up in the same manner as the outer polygon with one exception: Under the preliminary that openings like the one in the first example of 15b) are not part of the outer surrounding facet, openings of inner polygons are not relevant, we do not have to recursively generate inner polygons of inner polygons. Thus, such polygons are discarded.

Now, the roof facet is decomposed into a list of one or more non-self-intersecting outer polygons and a list of zero or more non-self-intersecting inner polygons. For each inner polygon, we have to find a unique outer polygon that surrounds it. In rare cases of damaged roof topologies, such a polygon might not exist. In such a case, the inner polygon and all included polygons get discarded. Then we can write a separate roof facet for each outer polygon with its corresponding openings, i.e., inner polygons.

For the square kilometer of the city center, the algorithm has to resolve 174 points of self-intersection with crossing. These points result from merging vertices and snapping vertices to ridge lines. Also, polygons are separated at 450 tangent points.

## 4. Planarization of roof facets

CityGML offers primitives to describe planar surfaces only. For example, dome roofs or curved roofs have to be approximated by fragments of planes. However, often CityGML models violate planarity to some extend. The easiest way to heal non-planar surfaces is to triangulate them. This actually is done in practice, see [32]. More sophisticated algorithms optimize roof structures. By locally adjusting a single roof facet to become planar, adjacent facets might loose this property. Therefore, in contrast to the algorithm in [15], we solve a global optimization problem to establish planarity of all roof facet's at the same time. Due to the linear nature of roof planes, linear programming (cf. Section 3.3) appears to be an appropriate means to also deal with this problem. The approach differs from more general shape preserving algorithms based on non-linear optimization, cf. [33, 34, 35] and the literature cited there. Nonlinear optimization of energy functions in fact is standard in architecture reconstruction. Another example ist the work of Arikan et al. [36]. They use a Gauss-Seidel algorithm to snap together polygons.

In brief, the basic idea behind our linear program is presented in [37]. Here we give an extended description covering additional rules and refinements that are necessary to make the solution work in practice.

$P_k$, $k \in \{1, 2, \dots, n\}$, denotes a roof polygon with vertices $p_{k,1}, \dots, p_{k,m_k} \in V$,

$$p_{k,j} = (p_{k,j}.x, p_{k,j}.y, p_{k,j}.z),$$

where $V \subset \mathbb{R}^3$ is the roof's finite vertex set. Via $V$, different polygons might share vertices.

The task is to replace $z$-coordinates $p_{k,j}.z$ with new height values

$$p_{k,j}.h = p_{k,j}.z + p_{k,j}.h^+ - p_{k,j}.h^-,$$

$p_{k,j}.h^+, p_{k,j}.h^- \geq 0$, such that the polygons become planar. Since polygons are coupled via $z$-coordinates of shared vertices, this is a global optimization task. In contrast to the mixed integer linear program in Section 3.2, we do not change $x$- and $y$-coordinates during this processing step. This is important to keep both given footprints and rectangular structures as well as to obtain a linear optimization problem. A related approach for quadrangular meshes is presented in [38]. It propagates $z$-coordinate changes by solving linear equations to keep surface elements planar.

By changing $z$-coordinates, visibility of outer walls do change. Since we deal with LoD 2, we can completely replace all wall polygons by new ones only depending on the planarized roof facets, see Section 5.

For each polygon $P_k$, we determine three non-collinear vertices $p_{k,u}$, $p_{k,v}$, and $p_{k,w}$ such that, if projected to the $x$-$y$-plane, $p_{k,u}$ and $p_{k,v}$ have a largest distance. Then $p_{k,w}$ is selected such that the sum of $x$-$y$-distances to $p_{k,u}$ and $p_{k,v}$ becomes maximal and vectors $\vec{a}_k := (p_{k,u}.x - p_{k,v}.x, p_{k,u}.y - p_{k,v}.y)$ and $\vec{b}_k := (p_{k,w}.x - p_{k,v}.x, p_{k,w}.y - p_{k,v}.y)$ become linear independent.

We compare all vertices of $P_k$ with the reference plane defined by points $(p_{k,u}.x, p_{k,u}.y, p_{k,u}.h)$, $(p_{k,v}.x, p_{k,v}.y, p_{k,v}.h)$, and $(p_{k,w}.x, p_{k,w}.y, p_{k,w}.h)$.

According to CityGML guidelines, one has to consider all combinations of three non-collinear vertices and compare against the corresponding planes (see [39, p.5]). But for numerical stability, we restrict ourselves to the previously selected vertices with large distances.

Each point $(p_{k,j}.x, p_{k,j}.y)$, $j \in \{1, \dots, m_k\}$, can be uniquely written as linear combination

$$(p_{k,j}.x, p_{k,j}.y) = (p_{k,v}.x, p_{k,v}.y) + r_{k,j}\vec{a}_k + s_{k,j}\vec{b}_k$$

with scalars $r_{k,j}$, $s_{k,j}$.

$$\begin{aligned} c_{k,j} \quad := \quad & -p_{k,j}.z + (1 - r_{k,j} - s_{k,j})p_{k,v}.z \\ & +r_{k,j}p_{k,u}.z + s_{k,j}p_{k,w}.z, \end{aligned}$$

we introduce auxiliary variables $\alpha_{k,j}$ via (see [1])

$$\begin{aligned} \alpha_{k,j} := & -p_{k,j}.h^+ + p_{k,j}.h^- \\ & +(1 - r_{k,j} - s_{k,j})(p_{k,v}.h^+ - p_{k,v}.h^-) \\ & +r_{k,j}(p_{k,u}.h^+ - p_{k,u}.h^-) \\ & +s_{k,j}(p_{k,w}.h^+ - p_{k,w}.h^-) + c_{k,j}. \end{aligned} \tag{6}$$

If the facet $P_k$ is planar then $\alpha_{k,j} = 0$ for all $j$. But because coordinates typically are rounded to three decimals in CityGML model files, we have to allow $\alpha_{k,j}$ to be within certain bounds:

$$-\delta_k \leq \alpha_{k,j} \leq \delta_k.$$

To define bound $\delta_k$, let $\nu_k$ be the reference plane's normal vector. Since the plane belongs to a roof facet, we can assume $\nu_k.z > 0$. Let $\mu$ be an error threshold (for example $\mu := 0.001$ m if coordinates are given in millimeters), then [1]

$$\delta_k := \mu + \frac{\sqrt{1 - \nu_k.z^2}}{\nu_k.z} \cdot \sqrt{2} \cdot \mu. \tag{7}$$

This means that each vertex $p_{k,j}$ has to be closer to the reference plane than

$$\nu_k.z \cdot \delta_k = \mu \left( \nu_k.z + \sqrt{2}\sqrt{1 - \nu_k.z^2} \right).$$

The right side reaches a maximum for $\nu_k.z = 1/\sqrt{3}$ so that the maximum distance to the plane is less or equal $\sqrt{3}\mu$.

In bound (7), the first summand $\mu$ allows $z$-coordinates to vary up to $\mu$. The second expression is constructed to take care of deviations into $x$- and $y$-directions up to $\mu$. Depending on the normal $\nu_k$, such deviations result in height changes on the reference plane that are up to a magnitude of $\frac{\sqrt{1-\nu_k.z^2}}{\nu_k.z} \cdot \sqrt{2} \cdot \mu$. If the roof facet has large slope and appears to be nearly vertical, then $\nu_k$ is close to zero and we allow large deviations of $z$ coordinates. When processing the reference data, such larger bounds in fact are required to maintain the shape of some tower roofs, see Figure 16.

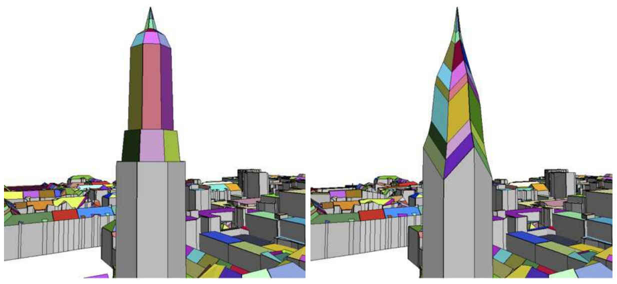


Figure 16: If one chooses $\mu$ as a bound for height changes and does not consider rounding errors of $x$- and $y$-coordinates, then linear optimization changes the shape of the tower (from left to right) [1].

It might be necessary to make changes to certain roof vertices more expensive. For example, this is the case if vertices belong to facades that should be textured with photos (cf. Figure 17). Such vertices often have $x$- and $y$-coordinates that also occur within the terrain intersection curve of a building. We introduce costs to the objective function in terms of the weights $\omega(p)$. Such weights do not influence the existence of solutions.

Summing up, we use a linear program with structural variables $p.h^+$ and $p.h^-$ for $p \in V$, auxiliary variables $\alpha_{k,j}$, $k \in \{1, \dots, n\}$, $j \in \{1, \dots, m_k\}$, and weights $\omega(p)$, $p \in V$, see formulas (6, 7):

$$\text{minimize} \sum_{p \in V} \omega(p)[p.h^+ + p.h^-]$$

s.t. $p.h^+, p.h^- \geq 0$, $-\delta_k \leq \alpha_{k,j} \leq \delta_k$ for all $p \in V$, $k \in \{1, \dots, n\}$, $j \in \{1, \dots, m_k\}$.

We are looking for a global minimum that always exists because each flat roof is a feasible solution. But the global minimum might require large height changes for single vertices. To this end, we also introduce a bound $\varepsilon > 0$ for height changes at each vertex [1]:

$$p.h^+ \leq \varepsilon, \; p.h^- \leq \varepsilon \text{ for all } p \in V.$$

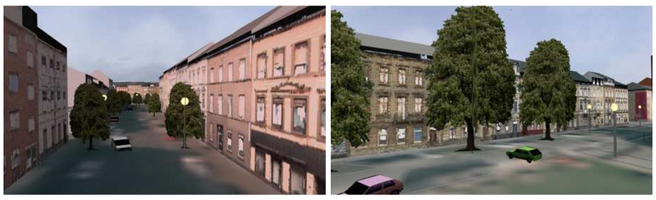


Figure 17: Texturing of facades requires higher precision of $z$-coordinates. Black areas between textures and roofs indicate that walls are too high whereas slopes of roof facets are too small [1].

If one chooses $\varepsilon$ too small, then there might be no feasible solution. Therefore, we start planarization of a building with a small value ($\varepsilon = 0.1$ m) and then iteratively double it until a solution is found.

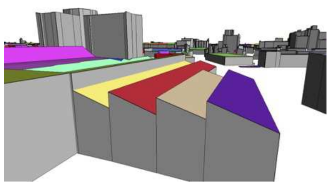


Figure 18: Shed roofs (or dormers etc.) have walls (step edges) within a roof [1].

If a vertex in $V$ belongs to two polygons then both polygons keep the same $z$-value at this position. Different $z$-values require different vertices. If there are vertices with same $x$- and $y$-coordinates but different $z$-values (see Figure 18), then in an optimal solution the order of their $z$-coordinates might be different. Then the set of visible walls and the appearance of the roof might also change. To avoid this, we sort the vertices by increasing $z$-coordinates. Let $v_1, \cdots, v_l \in V$ be vertices with $v_i.x = v_j.x$, $v_i.y = v_j.y$, $1 \leq i, j \leq l$ and $v_1.z \leq v_2.z \leq \cdots \leq v_l.z$. For $1 \leq i < l$ we add constraints [1]

$$v_{i+1}.z + v_{i+1}.h^+ - v_{i+1}.h^- - v_i.z - v_i.h^+ + v_i.h^- \geq 0. \tag{8}$$

Although we use weights to make changes to facades more expensive, experiments show that vertices of upper wall edges might change their heights independently. Vertices with the same original height then might get different $z$-coordinates so that corresponding facade edges get a wrong slope. Therefore, we collect all pairs of vertices of upper facade edges for which the vertices have approximately the same $z$-coordinates. Then we add rules that ensure that for each pair the new height values do not differ to much from each other.

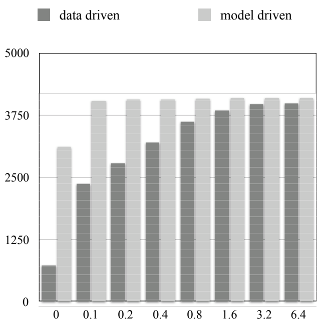
Also, one has to add lower bounds $v.z + v.h^+ - v.h^- \geq b$ to ensure that changed height values are not below the ground plane $z$-coordinate $b$ of each building.

If two roof polygons share three or more non-collinear vertices then optimization can't find different planes for these polygons. Therefore, prior to running the linear program, we analyze sequences of three consecutive non-collinear vertices that are shared with same $z$-coordinates between each two roof facets. If one roof facet is an inner polygon of the other facet then we merge both roof segments. This reduces the number of roof facets and simplifies the model. Otherwise, we split up middle vertices into two separate vertices. Formally this requires $V$ to be a multi-set instead of the set used here. In such situations, optimization might lead to artificial step edges.

The linear program changes normal vectors of roof segments. We allow that. But if one wants to exclude such solutions that involve larger changes of normal vectors, one can add further restrictions to the linear program that bound $x$- and $y$-coordinates of normal vectors only to vary in a certain range defined by $\kappa \geq 0$. In order to keep constraints linear in $p.h = p.h^+ - p.h^-$, we do not normalize cross products of vectors that contain structural variables. Let $\tilde{\nu} := (p_{k,v} - p_{k,u}) \times (p_{k,w} - p_{k,u})$ then conditions read ($1 \leq k \leq n$, [1]):

$$\begin{gathered}
|(p_{k,v}.y - p_{k,u}.y)(p_{k,w}.h - p_{k,u}.h) \\
-(p_{k,v}.h - p_{k,u}.h)(p_{k,w}.y - p_{k,u}.y) - \tilde{\nu}.x| \\
\leq \kappa|\tilde{\nu}|, \\
|(p_{k,v}.h - p_{k,u}.h)(p_{k,w}.x - p_{k,u}.x) \\
-(p_{k,v}.x - p_{k,u}.x)(p_{k,w}.h - p_{k,u}.h) - \tilde{\nu}.y| \\
\leq \kappa|\tilde{\nu}|.
\end{gathered}$$

Due to the strong restrictions, optimization may fail for some buildings so that their roofs keep unevennesses. There is a trade-off between planarity and authenticity of roof slopes.



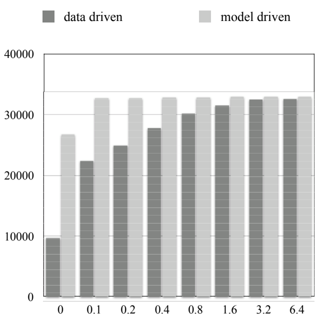


Figure 19: Numbers of planarized buildings depending on the choice of $\varepsilon$ with regard to the one square kilometer of reference data (left) and a larger area of 16 square kilometers covering the city center of Krefeld (right). Numbers are given both for our data driven model as well as for the model-driven North Rhine Westphalian model.

In our data-driven reference data set, 59% of buildings become 0.001-approximate planar if we set threshold value $\varepsilon$ to 0.1. We do not use GLPK's exact rational arithmetic because of performance issues. The standard float arithmetic causes some rounding errors. If we apply the algorithm to its own output, then we see that still not all buildings have approximate planar roof facets. But then all buildings become approximate planar with $\varepsilon = 0.1$. Thus, the true share of buildings that are already given with approximate planar roof facets is up to 59%.

If we successively double $\varepsilon$ to 0.2, 0.4, 0.8, 1.6, and 3.2, an additional number of 11%, 10%, 10%, 6%, and 3% of buildings become approximate planar through optimization. All buildings become approximate planar by choosing $\varepsilon = 6.4$. Figure 20 shows what happens to the complex roof of a church. Running time for processing all buildings of the square kilometer with a single-threaded implementation is less than five seconds on an i5 processor. For about 1% of buildings, enforcing planarity leads to visible changes of roofs' appearances ($z$-coordinate errors of more than 2 m).

We also applied the algorithm to the corresponding model-driven North Rhine Westphalian model. With $\varepsilon = 0.1$, 98,7% of buildings are 0.001-approximate planar after optimization, i.e., they can be considered as correctly modeled. By successively doubling $\varepsilon$, an additional number of 0.4%, 0.3%, 0.3%, and 0.2% of buildings become approximate planar. All buildings become approximate planar with $\varepsilon = 3.2$. Figure 19 summarizes these results. It also shows outcomes for a larger area covering 16 square kilometers.

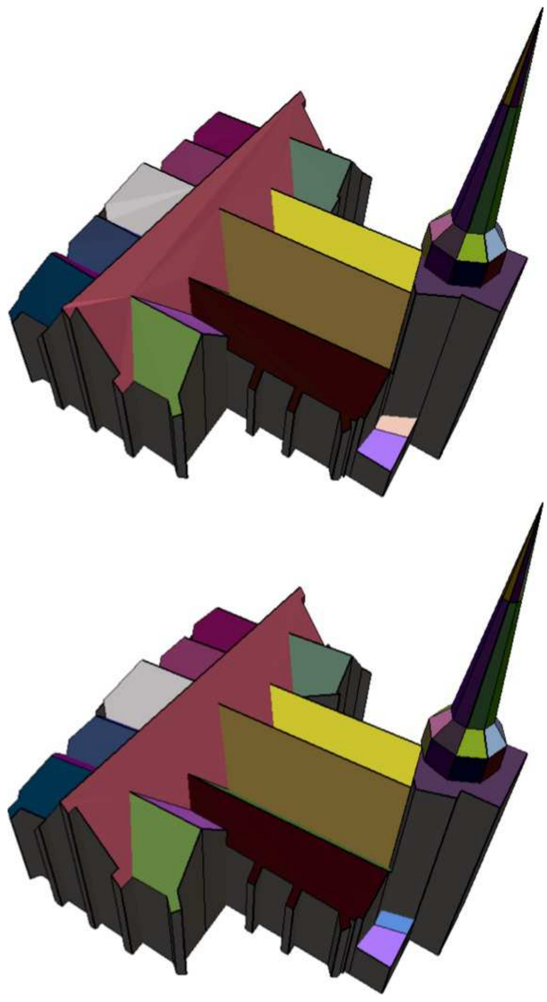


Figure 20: The upper model shows triangulated, non-planar roof facets. Different colors indicate different facets. Within a facet, roof triangles have different slopes. The second image is the result of planarization. It is free of such artifacts [1].

If one deals with a city model that already has planar roof facets but shows artificial step edges to allow for planarity, linear optimization can be also used to eliminate step edges while at the same time planarity is maintained. To this end, height differences

$$v.z + v.h^+ - v.h^- - u.z - u.h^+ + u.h^-$$

have to be minimized for each pair of vertices $u, v \in V$ with $u.x = v.x$, $u.y = v.y$, and $u.z \leq v.z$, where $v.z - u.z$ is less than a threshold bound $\beta$ for unwished step heights like $\beta = 0.5$ m. Please note that constraints (8) ensure that such differences are non-negative. For such vertices $u$ and $v$, we add summands $w\cdot(v.h^+ - v.h^- - u.h^+ + u.h^-)$ to the original objective function $\sum_{p\in V} \omega(p)[p.h^+ + p.h^-]$, where $w$ is a large weight that forces height differences to be smaller than a given precision (if compatible with constraints) whereas the original objective function avoids unnecessary height changes. For example we set $w$ to the overall number of variables times $\varepsilon \max\{\omega(p) : p \in V\}/(\mu/2)$. By allowing local height changes up to $\varepsilon := \beta$, the linear program of Section 4 then minimizes step edges with height differences below $\varepsilon$. If compliant with auxiliary conditions, height differences become smaller than precision so that step edges vanish. Unfortunately, it turns out that in practice the optimization of step edges leads to visible roof changes similar to the changes caused by planarization.

We developed a workflow to generate CityGML models from cadastral footprints and airborne laser scanning data, see [18]. Within that workflow, we use GLPK's simplex algorithm for planarization. Overall processing time for the entire reconstruction of Leverkusen's 66,400 buildings on 79 square kilometers is about four hours on two cores of an i5 processor with 4 GB of RAM. The simplex algorithm theoretically has an exponential worst-case running time but turns out to be extremely fast for our small optimization problems. It's share of processing time for the Leverkusen data set only is about five minutes. With respect to worst-case situations, the simplex algorithm could be replaced by an interior point method so that the optimization step runs in polynomial time.

## 5. Post-processing of walls

Coordinate changes of roof vertices do change the set of a building's outer walls. Since all walls are determined by roof edges and the height value of the building's ground plane, we can discard given, no longer fitting walls. For each oriented edge of each roof polygon, we generate a new vertical wall that reaches from the roof edge down to the ground plane of the building. At this point in time, each generated wall polygon has four different vertices, and each roof facet owns a complete set of surrounding walls. To generate a well-formed CityGML model, we now have to cut off invisible parts of the walls for each single building, see Figure 21. We do not remove walls between different buildings. Visible walls do not only occur in connection with the building's footprint. Step edges also occur within the interior of a roof, see Figure 18.

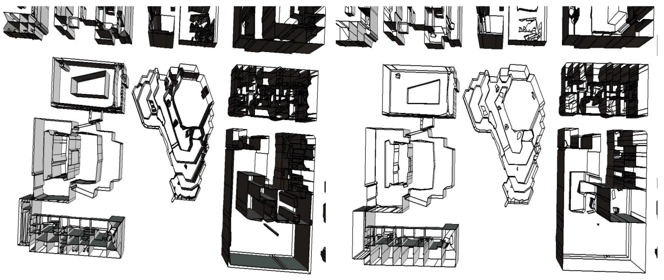


Figure 21: Walls of a conference center and theatre are displayed as wire frames, if their normal vector points away from the view point. Otherwise they are painted grey. The first figure shows the model with uncut walls. The second picture only shows parts of walls that are visible from the outside of each single building (without considering neighbor buildings). The view points through the ground surface to the sky [1].

For cutting, we use the precondition that no vertex of a roof or wall polygon lies on the inner part of any roof or wall edge. Thus, any vertex also is a vertex of all adjacent polygons. Therefore, we can determine visibility by comparing the pair of upper vertices of each wall with all roof edges between pairs of vertices that have the same $x$- and $y$-coordinates as the two wall vertices. With one exception, there can be at most two roof edges that fulfill this condition. The exception is the existence of openings that are modeled using inner polygons. Other roof facets can be positioned within openings. If two inner polygons share a common edge, then there might be up to four roof edges that have to be considered. Thus we exclude walls that belong to edges shared between inner polygons from further considerations since they do not belong to the building. The procedure in Section 3.4 takes care of this.

Obviously, there has to be at least one roof edge with the same vertex coordinates as the upper wall vertices. If there is a second roof edge with exactly these coordinates (including $z$-coordinates), then the edge is a ridge line and no wall is needed. If there is no second other roof edge with same $x$- and $y$-coordinates (and arbitrary $z$-coordinates), then we deal with an outer wall belonging to the building's footprint. This wall is completely visible. There are three remaining cases, see Figure 22:

a) $z$-coordinates of the upper wall vertices are both greater than the corresponding $z$-coordinates of the second roof edge: In this case, the lower edge of the wall has to be replaced by the roof edge.

b) $z$-coordinates of the upper wall vertices are both less or equal to the corresponding $z$-coordinates of the second roof edge: The wall completely is invisible because it is adjacent to a higher segment of the building.

c) One $z$-coordinate of the upper wall vertices is greater or equal and one $z$-coordinate is less or equal to the corresponding $z$-coordinate of the second roof edge: Only a triangle part of the wall is visible. A new vertex has to be introduced at the intersection point between the two edges. The wall polygon is replaced by a triangle, and the new vertex also has to be inserted to affected roof polygons.

Case c) is the reason why we cut off invisible parts of walls only after the planarization step (Section 4). Optimization leads to different slopes of roof edges, so that intersection points might change $x$- and $y$-coordinates. If we would add these intersection points before optimization takes place, then we would pin together both roof facets and do not allow $x$- and $y$-coordinate changes. This reduces the number of feasible solutions and might change the appearance of the roof.

For the tested square kilometer, 11,015 $z$-coordinates of wall polygons are modified according to case a), 36,619 wall polygons are removed along case b), and 8,394 walls are replaced by a triangle as described in case c).

Finally, adjacent walls with equal orientation can be merged to larger polygons. This happens 15,416 times.

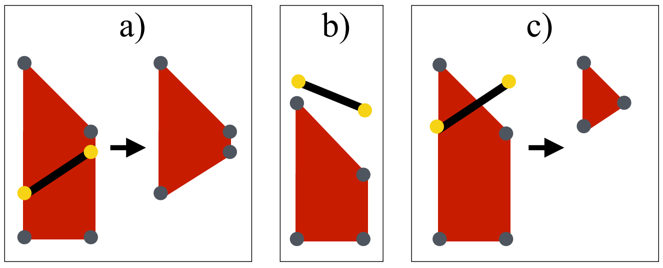


Figure 22: Visibility analysis of walls: Red polygons represent walls and black edges are parts of roof facets [1].

## 6. Conclusions and further work

We applied the presented techniques to improve data-based city models of the cities of Krefeld, Leverkusen, and Dortmund. According to cadastral requirements, their building models do not overlap cadastral footprints. But in reality, there are roof overhangs. Also, cadastral footprints might not exactly match the outer hull of a building. Both aspects in connection with merging data from different sources have impact on precision. Figure 23 shows buildings that are textured with oblique arial photos based on a plane's

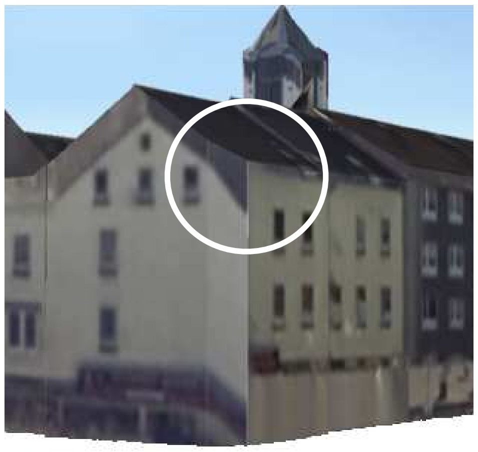


Figure 23: Shift of texture (left facades do not reach to the roof, right facades do match) indicate model's precision.

position data. One can clearly see that wall textures do not fit precisely. Because of the airborne camera's outer orientation, a horizontal shift by the thickness of a wall directly leads to a vertical shift of texture by the same magnitude. Buildings in the figure exactly match with cadastral data but do not match with oblique areal images. Based on such images, not only further improvements of CityGML models seem possible but also generated recommendations for cadastral footprint corrections.

## Acknowledgements

We would like to thank cadastre offices of the cities of Krefeld, Leverkusen, and Dortmund as well as GeoBasis NRW for supporting us with buildings' footprints, laser scanning data and true orthophotos. Especially, Marco Oestereich and Johannes Scharmann gave us valuable support.

## References

[1] S. Goebbels, R. Pohle-Fröhlich, Quality enhancement techniques for building models derived from sparse point clouds, in: Proceedings of International Conference on Computer Graphics Theory and Applications (GRAPP), Scitepress, Porto, 2017, pp. 93–104.

[2] G. Gröger, T. H. Kolbe, C. Nagel, K. H. Häfele, OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0, Open Geospatial Consortium, 2012.

[3] F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, A. Çöltekin, Applications of 3D city models: State of the art review, ISPRS Int. J. Geo-Inf. 4 (2015) 2842–2889.

[4] F. Tarsha-Kurdi, T. Landes, P. Grussenmeyer, M. Koehl, Model-driven and data-driven approaches using LIDAR data: Analysis and comparison, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 36 (3/W49A) (2007) 87–92.

[5] N. Haala, M. Kada, An update on automatic 3D building reconstruction, ISPRS Journal of Photogrammetry and Remote Sensing 65 (2010) 570–580.

[6] Y. He, Automated 3D Building Modeling from Airborne LiDAR Data (PhD thesis), University of Melbourne, Melbourne, 2015.

[7] A. Henn, G. Gröger, V. Stroh, L. Plümer, Model driven reconstruction of roofs from sparse LIDAR point clouds, ISPRS Journal of Photogrammetry and Remote Sensing 76 (2013) 17–29.

[8] S. N. Perera, N. G. Maas, A topology based approach for the generation and regularization of roof outlines in airborne laser scanning data, in: E. Seyfert (Ed.), DGPF Tagungsband 21, DGPF, Potsdam, 2012, pp. 400–409.

[9] S. O. Elbrink, G. Vosselman, Building reconstruction by target based graph matching on incomplete laser data: analysis and limitations, Sensors 9 (8) (2009) 6101–6118.

[10] M. Kada, A. Wichmann, Feature-driven 3D building modeling using planar halfspaces, ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci. II-3/W3 (2013) 37–42.

[11] A. Wichmann, M. Kada, Joint simultaneous reconstruction of regularized building superstructures from low-density LIDAR data using ICP, ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci. III-3 (2016) 371–378.

[12] I. Demir, D. G. Aliaga, B. Benes, Procedural editing of 3D building point clouds, in: 2015 IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society, Washington, DC, 2015, pp. 2147–2155.

[13] G. Gröger, L. Plümer, How to achieve consistency for 3D city models, ISPRS Int. J. Geo-Inf. 15 (1) (2009) 137–165.

[14] J. Zhao, J. Stoter, H. Ledoux, A framework for the automatic geometric repair of CityGML models, in: M. Buchroithner, N. Prechtel, D. Burghardt (Eds.), Cartography from Pole to Pole, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, 2013, pp. 187–202.

[15] N. Alam, D. Wagner, M. Wewetzer, J. von Falkenhausen, V. Coors, M. Pries, Towards automatic validation and healing of CityGML models for geometric and semantic consistency, ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci. II-2/W1 (2013) 1–6.

[16] D. Wagner, M. Wewetzer, J. Bogdahn, N. Alam, M. Pries, V. Coors, Geometric-semantical consistency validation of CityGML models, in: J. Pouliot, et. al. (Eds.), Progress and New Trends in 3D Geoinformation Sciences, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, 2013, pp. 171–192.

[17] H. Ledoux, val3dity: validation of 3D GIS primitives according to the international standards, Open Geospatial Data, Software and Standards. 3 (1) (2018) 1–12.

[18] S. Goebbels, R. Pohle-Fröhlich, Roof reconstruction from airborne laser scanning data based on image processing methods, ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci. III-3 (2016) 407–414.

[19] M. Oestereich, Das 3D-Gebäudemodell im Level of Detail 2 des Landes NRW, Nachrichten aus dem öffentlichen Vermessungswesen Nordrhein-Westfalen 47 (1) (2014) 7–13.

[20] L. Tong, M. Li, Y. Chen, Y. Wang, W. Zhang, L. Cheng, A research on 3D reconstruction of building rooftop models from LiDAR data and orthophoto, in: Proceedings of the 20th International Conference on Geoinformatics (GEOINFORMATICS), Hong Kong, IEEE Computer Society, Washington, DC, 2012, pp. 1–5.

[21] H. Arefi, P. Reinartz, Building reconstruction using DSM and orthorectified images, Remote Sens. 5 (4) (2013) 1681–1703.

[22] R. Guo, Q. Dai, D. Hoiem, Single-image shadow detection and removal using paired regions, in: Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '11, IEEE Computer Society, Washington, DC, 2011, pp. 2033–2040.

[23] G. D. Finlayson, S. D. Hordley, M. S. Drew, Removing shadows from images, in: Proceedings of the 7th European Conference on Computer Vision-Part IV, ECCV '02, Springer, Berlin, 2002, pp. 823–836.

[24] L. Nan, A. Sharf, H. Zhang, D. Cohen-Or, B. Chen, Smartboxes for interactive urban reconstruction, ACM Trans. Graph. 29 (4) (2010) 93:1–93:10.

[25] Y. Verdie, F. Lafarge, P. Alliez, LOD generation for urban scenes, ACM Trans. Graph. 34 (3) (2015) 30:1–30:14.

[26] S. Goebbels, R. Pohle-Fröhlich, Beautification of city models based on mixed integer linear programming, in: Proceedings Operations Research 2018 (OR 2018), Springer, Brussels, 2019, pp. 119–125.

[27] A. Makhorin, The GNU Linear Programming Kit (GLPK), Free Software Foundation, Boston, MA, 2009.

[28] S. Hensel, S. Goebbels, M. Kada, Facade reconstruction for textured LoD2 CityGML models based on deep learning and mixed integer linear programming, ISPRS Ann. Photogramm. Remote Sens. and Spatial Inf. Sci. IV-2/W5 (2019) 37–44.

[29] H. Ledoux, K. Arroyo Ohori, M. Meijers, A triangulation-based approach to automatically repair GIS polygons, Computers & Geosciences 66 (2014) 121–131.

[30] M. Attene, M. Campen, L. Kobbelt, Polygon mesh repairing: An application perspectice, ACM Computing Surveys 45 (2), article 15.

[31] J. Bogdahn, V. Coors, Towards an automated healing of 3D urban models, in: T. H. Kolbe, G. König, C. Nagel (Eds.), Proceedings of international conference on 3D geoinformation, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 2010, pp. 13–17.

[32] R. Boeters, K. A. Ohori, F. Biljecki, S. Zlatanova, Automatically enhancing CityGML LOD2 models with a corresponding indoor geometry, International Journal of Geographical Information Science 29 (12) (2015) 2248–2268.

[33] S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, M. Pauly, Shape-up: Shaping discrete geometry with projections, Computer Graphics Forum 31 (5) (2012) 1657–1667.

[34] B. Deng, S. Bouaziz, M. Deuss, A. Kaspar, Y. Schwartzburg, M. Pauly, Interactive design exploration for constrained meshes, Comput. Aided Des. 61 (C) (2015) 13–23.

[35] C. Tang, X. Sun, A. Gomes, J. Wallner, H. Pottmann, Form-finding with polyhedral meshes made simple, ACM Trans. Graph. 33 (4) (2014) 70:1–70:9.

[36] M. Arikan, M. Schwärzler, S. Flöry, M. Wimmer, S. Maierhofer, O-snap: Optimization-based snapping for modeling architecture, ACM Trans. Graph. 32 (1) (2013) 6:1–6:15.

[37] S. Goebbels, R. Pohle-Fröhlich, J. Rethmann, Planarization of CityGML models using a linear program, in: Operations Research Proceedings (OR 2016 Hamburg), Springer, Berlin, 2016, pp. 591–597.

[38] R. Mesnil, C. Douthe, O. Baverel, B. Léger, From descriptive geometry to fabrication-aware design, in: S. Adriaenssens, F. Gramazio, M. Kohler, A. Menges, M. Pauly (Eds.), Advances in Architectural Geometry, vdf Hochschulverlag, Zürich, 2016, pp. 62–81.

[39] SIG3D, Handbuch für die Modelierung von 3D Objekten, Teil 1: Grundlagen, 0th Edition, Geodateninfrastruktur Deutschland, 2014.