# Quality enhancement techniques for building models derived from sparse point clouds

Steffen Goebbels[1] and Regina Pohle-Fröhlich[1]

[1] *Institute for Pattern Recognition, Faculty of Electrical Engineering and Computer Science, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany*
*{Steffen.Goebbels, Regina.Pohle}@hsnr.de*

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

Abstract: This paper describes processing steps that improve both geometric consistency and appearance of CityGML models. In addition to footprints from cadastral data and sparse point clouds obtained from airborne laser scanning, we use true orthophotos to better detect and model edges. Also, procedures to heal self-intersection of polygons and non-planarity of roof facets are presented. Additionally, the paper describes an algorithm to cut off invisible parts of walls. We incorporate these processing steps into our data based framework for building model generation from sparse point clouds. Results are presented for German cities of Krefeld and Leverkusen.

## 1 INTRODUCTION

During recent years, 3D city models have become increasingly important for planning, simulation, and marketing, see (Biljecki et al., 2015). CityGML is the XML based standard for exchanging and storing such models, see (Gröger et al., 2012). A common approach to automate the generation of city models is to use cadastral data in combination with point clouds from airborne laser scanning. However, publicly available point clouds of our home country often are sparse and currently consist of less than ten points per square meter. This makes it very difficult to recognize small roof structures which are typical for dormers and tower roofs.

There are two main approaches to roof reconstruction (see (Tarsha-Kurdi et al., 2007), cf. (Haala and Kada, 2010; He, 2015; Henn et al., 2013; Perera and Maas, 2012)): Model driven approaches select standard roofs from a catalogue so that they optimally fit with the points from the cloud. In general, only larger roof structures find their way into these building models. To the contrast, data driven methods detect planes and combine even small roof facets to complete roofs (see e. g. (Elbrink and Vosselman, 2009; Kada and Wichmann, 2013)). Our experiments in (Goebbels and Pohle-Fröhlich, 2016) show that a data driven approach even is able to deal with dormers, but their outline might be distorted due to sparse coverage by laser scanning points. Therefore, one needs additional information to correctly model such small structures. For example, in (Wichmann and Kada, 2016) similar dormers are detected to merge their point clouds. Thus, a prototype dormer model can be generated based on a much higher point density. Also, Demir et. al. describe in (Demir et al., 2015) a procedure to detect repeating structures in point clouds. In this paper

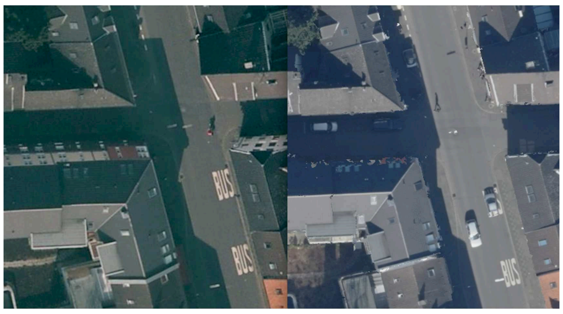
Figure 1: Orthophoto (left) vs. true orthophoto (right)

we use true orthophotos as additional information. An orthophoto is a geometrically corrected areal photo that fits with the coordinate system at ground level. 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.

The given paper is not only concerned with better modeling structures but also with improving geometric consistency, cf. (Gröger and Plümer, 2009; Zhao et al., 2013). 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. propose an approach that corrects all types of geometric error using shrink-wrapping that works on a watertight approximation of the tessellated building. Alam et al. (Alam et al., 2013) describe methods to correct single geometric problems of existing CityGML models. We present complementary and different processing steps to improve such single aspects of geometric quality during model generation. The algorithms deal with self-intersecting polygons, planarity of polygons, and correctness of wall boundaries. Violation of planarity is a common problem of city models. It is regarded as difficult to correct. We fix missing planarity with a surprisingly simple but powerful linear program. The ideas are applied but not limited to CityGML.

## 2 PREVIOUS WORK

We apply the algorithms of this paper in connection with our model generation workflow in (Goebbels and Pohle-Fröhlich, 2016), cf. Figure 2. In this section, we shortly summarize our previous work and explain how the subsequent sections contribute to this automated model generation workflow. However, our workflow only serves as an example and might be replaced by other methods.

Figure 2: City model of our hometown

We detect roof facets on a height map that is interpolated from the sparse point cloud. The height map consists of a raster with $10 \times 10$ cm$^2$ cell size which fits with the accuracy of data. It is partially initialized with the cloud points. To complete the map, we use an interpolation method that maintains step edges.

Before we detect roof planes using a RANSAC algorithm, we first classify regions according to gradient directions. To this end, we handle all regions with gradients shorter than a threshold value as flat roofs. These areas might have arbitrary and noisy gradient angles. For remaining areas we determine minima of a gradient angle histogram. Each interval between two subsequent minima classifies the angles that belong to roof facets with approximately the same gradient direction. On regions of such homogeneous gradient angles we estimate planes and find roof facets using RANSAC. Filtering by gradient angles excludes outliers. It also avoids difficulties with RANSAC. For example, we do not find planes that intersect with many roof facets without themselves being a roof plane. The intermediate result is a 2D raster map in which cells are colored according to the roof facet they belong to.

We complete gaps in the raster map of roof facets using region growing. To maintain the characteristic appearance of the roof, we consider intersection lines between estimated planes (ridge lines) and step edges according to the height map as boundaries for growing. However, small structures like dormers do not show straight and parallel edges after this procedure. To improve model quality, we now additionally consider cadastral building part information and edges that are visible in true orthophotos, see Section 4.

From the raster map we can directly determine inner roof facets that completely lie within a surrounding facet. Such inner facets become openings in the surrounding facet polygon. That is also true for openings of a building's footprint from cadastral data. Border polygons of other openings are outer polygons of inner roof facets (with inverse orientation) so that we only have to detect outer polygons.

We determine each roof facet's outer 2D boundary polygon using Pavlidis algorithm that moves along the facet's contour. Leading to a watertight roof, we determine and merge polygon segments that are shared between adjacent facets. A topologically relevant vertex is a vertex at which more than two roof facets meet. Such vertices are detected. Then 2D roof polygons are simplified using a modified Ramer-Douglas-Peucker algorithm. This algorithm does not allow topologically relevant vertices to be removed from the polygons. Therefore, the roof remains watertight.

To overcome the precision that is limited by the $10 \times 10$ cm$^2$ cell size, we project vertices to edges of the cadastral footprint and to intersection lines (ridge edges) as well as to intersection points of ridge lines. This might lead to self-intersecting polygons. A solution to resolve this problem is described in Section 5.

To generate a 3D model, height values have to be added to the 2D polygons. These heights are computed using estimated plane equations. To avoid small artificial step edges, mean values are used for vertices with same $x$- and $y$-coordinates but only slightly different $z$-coordinates that lie within a threshold distance. This deliberately violates planarity of roof facets. To heal this problem, we use linear optimization in a post-processing step, see Section 6. CityGML requires planarity, see (Gröger et al., 2012, p. 25). To some extend, many city models violate this requirement, cf. (Wagner et al., 2013), and can be corrected using our linear program. However, our workflow can alternatively generate planar roof facets by not using mean values and allowing small step edges. Then linear optimization can be used to eliminate step edges instead of establishing planarity, see Section 8.

Another CityGML requirement is that, together with the ground plane, roof and wall polygons completely belong to a building's outer hull. Therefore, we have to cut away invisible segments of walls. The linear program for planarization might change slopes of roof facets which in turn influences visibility of walls. In Section 7 we finally describe a wall processing step that has to run after planarization.

The complete workflow consists of following steps:

- Computation of a height map that is interpolated from the point cloud
- Estimation of roof facets in 2D raster domain
- Adjusting boundaries, see Section 4
- Contour detection gives 2D boundary polygons
- Simplification of boundary polygons and mapping to ridge lines, step edges, etc. (might lead to additional self-intersections)
- Resolving self-intersections (see Section 5)
- Adding $z$-coordinate values
- Planarization of roof facets (see Section 6)
- Cutting and merging walls to visible segments (see Section 7)

## 3 REFERENCE DATA SET

We apply the presented techniques to a square kilometer of the city center of the German city of Krefeld with 3987 buildings that we process from a sparse point cloud with five to ten points per square meter, see Figure 3. The area corresponds to the UTM interval $[32330000, 32331000] \times [5689000, 5690000]$. Additionally, we use data of the city of Leverkusen.

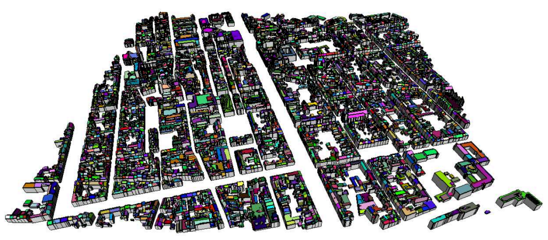
Figure 3: City center as reference data set

## 4 ADJUSTING BOUNDARIES OF ROOF FACETS

In our workflow, we generate a 2D raster map in which pixel colors refer to corresponding roof facets, see Figures 4–6. However, due to the sparse point distribution of the reference laser scanning data, boundary contours of small structures like dormers are not straight and are only approximately correct. There are also gaps in the raster map. We complete the map and optimize boundary contours by applying region growing. The idea is to find candidates of true boundaries that become limits for the growing process. We obtain such candidate lines from the point cloud as well as from additional data sources. For example, we compute intersection lines of estimated plane equations that correspond with ridge lines, see Figure 4. Cadastral footprints of building parts (see

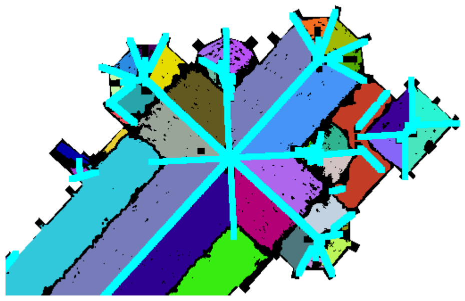
Figure 4: Intersection lines between estimated planes

Figure 5) and edges of true orthophotos are additional information. It is important that additional data are optional and not required. Only a minority of buildings possess building part definitions and such cadastral data often do not cover dormers. If available, 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

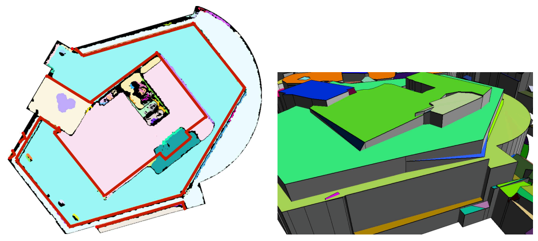
Figure 5: Raster maps of roof facets (left): Red cadastre polygons are borders of building parts, black areas could not be associated with roof facets. The picture on the right hand side shows the derived model.

modeling. For example, Tong et. al. (Tong et al., 2012) 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 (Arefi and Reinartz, 2013) decompose buildings and fit parametric roof models according to ridge lines detected from true orthophotos. In our experiments, we try to utilize all types of edges to adjust roof boundaries that already have been obtained from the point cloud.

We use true othophotos with pixels that represent areas of $10 \times 10$ cm$^2$. Although the photos originate from areal images with only 60% horizontal and 50% vertical overlap, edges of the images fit well with laser scanning data within our computing tolerance of 10 cm$^2$, cf. Figure 6. 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

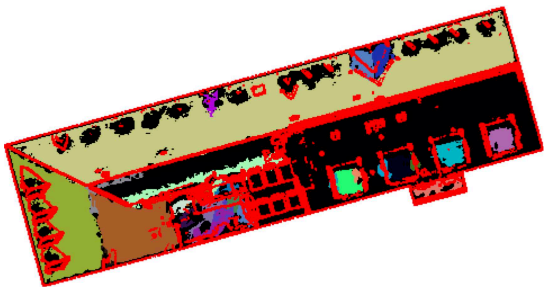
Figure 6: Red pixels belong to edges of a true orthophoto. Colored areas belong to planes that we detect in the laser scanning point cloud. Both data sources match quite well. Small dormers are not sufficiently covered by laser scanning points. Boundaries of larger dormers can be adjusted to the edges.

removal of shadows, cf. (Guo et al., 2011) and the literature cited there. However, our experiments with the algorithm of (Finlayson et al., 2002) did not gave sufficient results. A reason might be the variety of colors of our arial photos. But in our processing pipeline, edges only act as limits for region growing and are not used to detect roof facets. Therefore, wrongly placed edges do not heavily disturb results: If an edge crosses an interior region of a roof facet, then the facet's area grows on both sides of the edge. Thus, the edge becomes no boundary of the facet. However, shadows may hide useful edges.

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 needing to select a threshold value, see Figure 7. 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 a picture $E$ 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 used to filter for edges that are parallel or orthogonal to segments of the building's footprint.

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

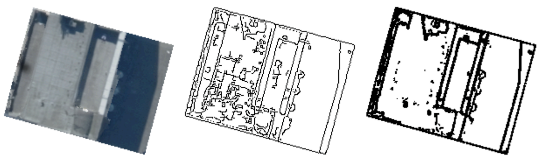
Figure 7: Results of Canny edge detector (middle) vs. our curvature based method (right)

Once we have derived boundaries for region growing, we eliminate noise near computed and detected edges, especially near pixels of $E$. Noise consists of pixels 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 $E$. Additionally, large height jumps (step edges) in the height map limit region growing. Step edges with small height differences can not be detected from the interpolated height map dependably. 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 act as boundaries.

Figure 8 shows excerpts from a city model before planarization is performed (see Section 6). Region growing based on edges of the true orthophoto significantly 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.

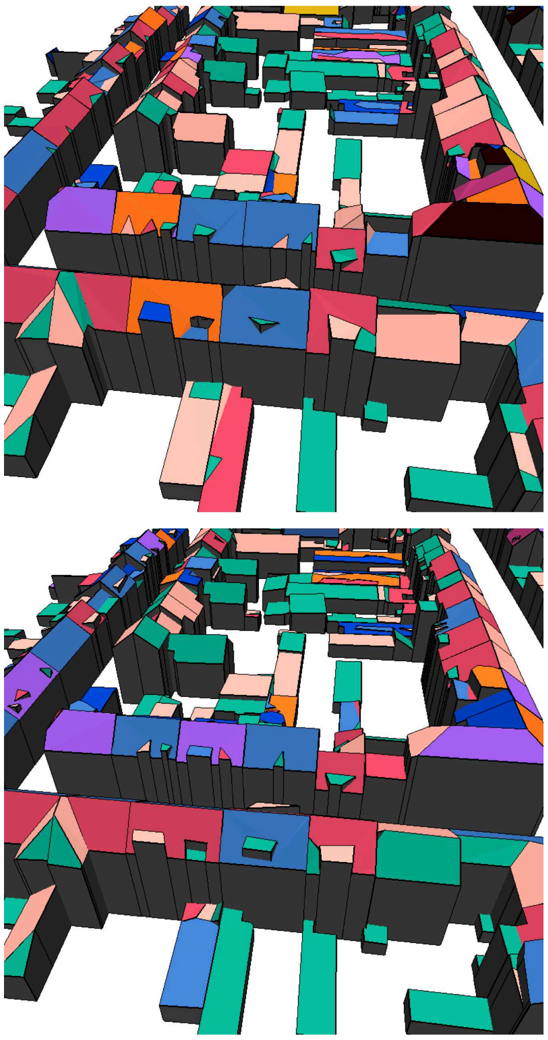

Figure 8: Boundaries of the upper model are adjusted to edges of a true orthophoto. This results in the second picture. Due to the region growing algorithm, some small roof segments vanish, other small facets appear.

## 5 CORRECTING SELF-INTERSECTING ROOF POLYGONS

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 either because of the roof's topology or because of over-simplification. Therefore, we recursively have to split up polygons until all intersections are eliminated. 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. (Bogdahn and Coors, 2010). But we also add it to all other polygons that cross that point. This is necessary to easily decide about wall visibility, see Section 7. 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 fist 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 9a) 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 9b)). All segments with counter-clockwise orientation become new outer polygons. The second example of Figure 9b) is resolved to two outer polygons. All segments with clockwise orientation become additional new inner polygons. This happens in the first example of Figure 9b).

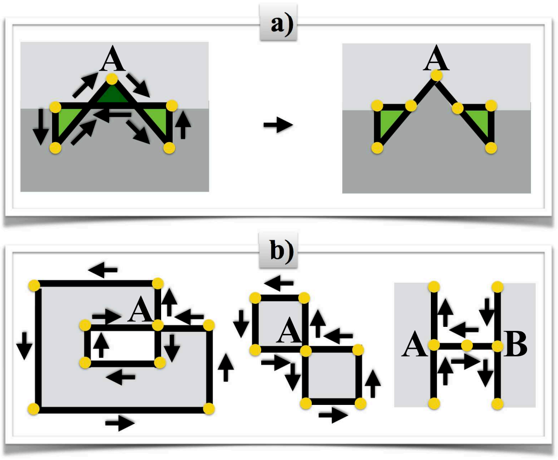

Figure 9: Resolving self-intersections.

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 9b) 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 the unique outer polygon that surrounds it. 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.

## 6 PLANARIZATION OF ROOF FACETS

CityGML only offers primitives to describe planar surfaces. Therefore, e.g., dome roofs or curved roofs have to be approximated by plane fragments. 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 (Boeters et al., 2015). 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 (Alam et al., 2013), 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 appears to be an appropriate means. This approach differs from more general shape preserving algorithms based on non-linear optimization, cf. (Bouaziz et al., 2012; Deng et al., 2015; Tang et al., 2014) and the literature cited there. Non-linear optimization of energy functions in fact is standard in architecture reconstruction. Another example ist the work of Arikan et al. (Arikan et al., 2013). They use a Gauss-Seidel algorithm to snap together polygons.

We execute the linear program with the simplex algorithm that is implemented in the GNU Linear Programming Kit library GLPK (Makhorin, 2009). In brief, the basic idea behind such a linear program is presented in (Goebbels et al., 2017). Here we give an extended description that also covers 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.

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. We do not change $x$- and $y$-coordinates during this processing step. This is important to keep the footprint from cadastral data and to obtain a linear optimization problem. A related approach for quadrangular meshes is presented in (Mesnil et al., 2016).

It propagates $z$-coordinated changes by solving linear equations to keep surface elements planar.

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 the three 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 (SIG3D, 2014, p.5)). But for numerical stability we restrict ourselves to the previously selected vertices with large distances.

We can uniquely write each point $(p_{k,j}.x, p_{k,j}.y)$, $j \in \{1, \ldots, m_k\}$, 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 $r_{k,j}$, $s_{k,j}$. Using constants

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

we introduce auxiliary variables $\alpha_{k,j}$ defined by

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

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:

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

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

$$\delta_k := \mu + \frac{\sqrt{1 - \nu_k.z^2}}{\nu_k.z} \cdot \sqrt{2} \cdot \mu. \qquad (2)$$

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 (2), 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 10.

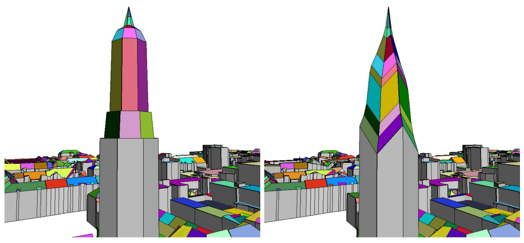
Figure 10: If one chooses $\mu$ as a bound for height changes and does not consider rounding errors of $x$- and $y$-coordinates, then linear optimization changes the shape of the tower (from left to right).

Depending on the model's application, 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 11). 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 $p_{k,j}.h^+$ and $p_{k,j}.h^-$ for $p_{k,j} \in V$, auxiliary variables $\alpha_{k,j}$, $k \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m_k\}$, and weights $\omega(p)$, $p \in V$, see formulas (1, 2):

$$\text{Minimize} \sum_{p \in V} \omega(p)[p.h^+ + p.h^-]$$
$$\text{s.t.} \quad p.h^+, p.h^- \ge 0, \quad -\delta_k \le \alpha_{k,j} \le \delta_k$$

for all $p \in V$, $k \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m_k\}$.

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:

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

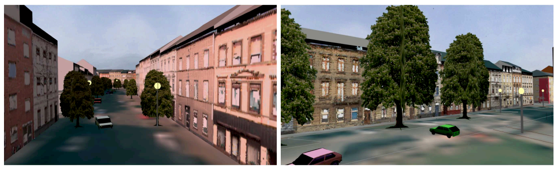

Figure 11: Texturing of facades requires higher precision of $z$-coordinates. Black areas between textures and roofs indicate that walls are too high whereas slopes of roof facets are too small.

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.

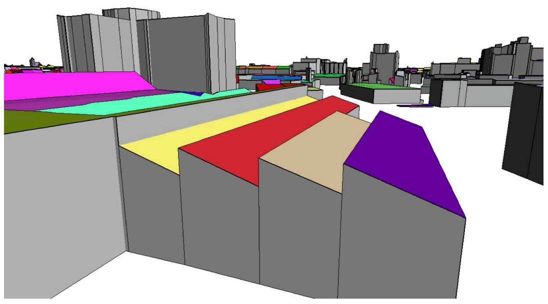

Figure 12: Shed roofs or dormers have walls (step edges) within a roof.

If there are vertices with same $x$- and $y$-coordinates but different $z$-values (see Figure 12), 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

$$v_{i+1}.z + v_{i+1}.h^+ - v_{i+1}.h^- - v_i.z - v_i.h^+ + v_i.h^- \geq 0. \tag{3}$$

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, if the $z$-coordinate of the middle vertex was replaced by a mean value to avoid step edges, we undo that modification.

Nevertheless, the optimization 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 ($1 \leq k \leq n$):

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

Under these restrictions, optimization of some buildings may fail so that roofs keep unevennesses. There is a trade-off between planarity and authenticity of roof slopes.

In our 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 13 shows what happens to the complex roof of a church. Running time for a single-threaded implementation is less than five seconds on an i5 processor.

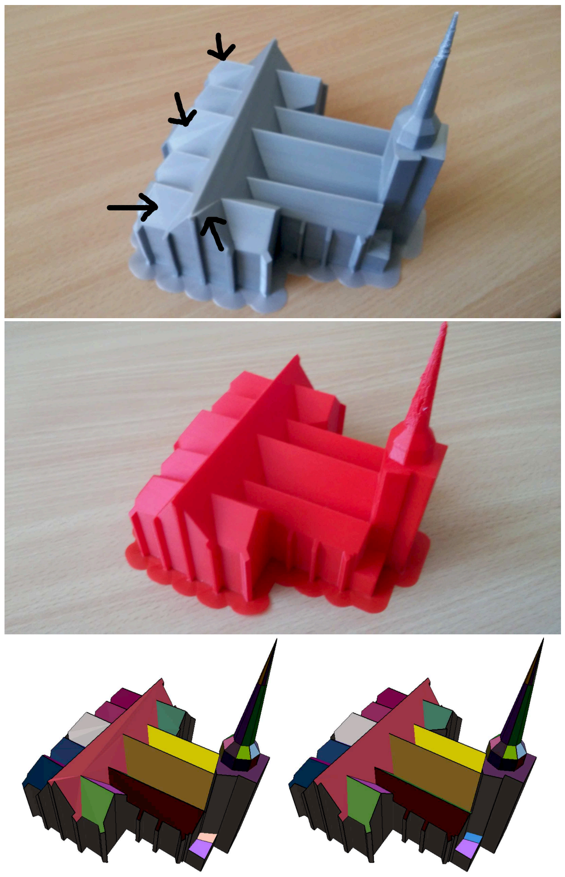

Figure 13: Two 3D prints of a church: The silver version shows the situation before linear optimization. Non-planar roof facets are triangulated, and some triangles show different slopes than others. The red model is the result of planarization. It is free of such artifacts. The figure also shows visualizations of the two models with different colors for each roof facet.

## 7 POST-PROCESSING OF WALLS

After finding planar roof polygons, we generate vertical walls that reach from roof edges down to the ground plane. The height of the ground plane is taken from a digital elevation model. At this point in time, each wall polygon has four different vertices. To generate a well-formed CityGML model, we now have to cut off invisible parts of the walls, see Figure 14. To this end, 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 5 takes care of this.

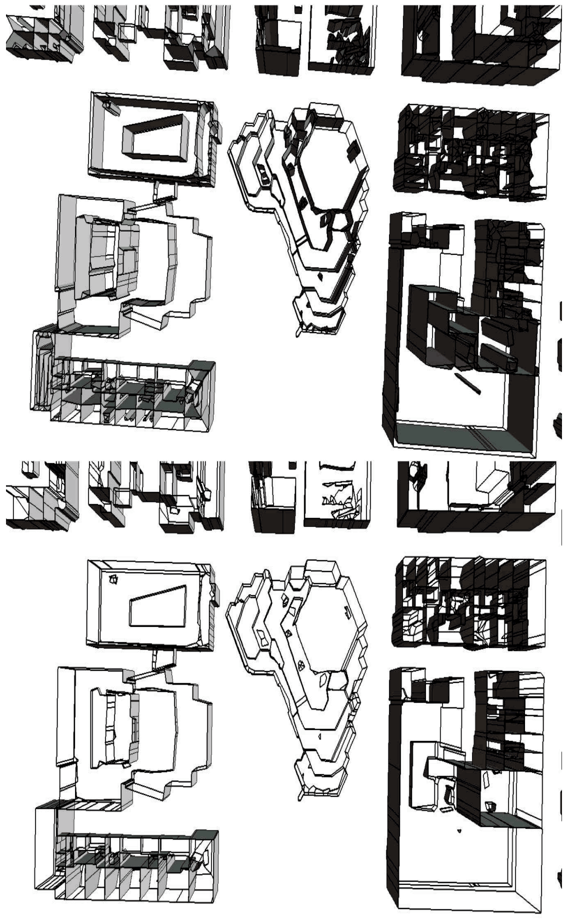

Figure 14: Walls of a conference center and theatre are displayed as wire frames, if their normal vector points away from the view point. Otherwise they are painted grey. The first figure shows the model with uncut walls. The second picture only shows parts of walls that are visible from the outside of each single building (without considering neighbor buildings). The view points through the ground surface to the sky.

Obviously, there has to be at least one (first) roof edge with the same 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 15:

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 linear optimization step. 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.

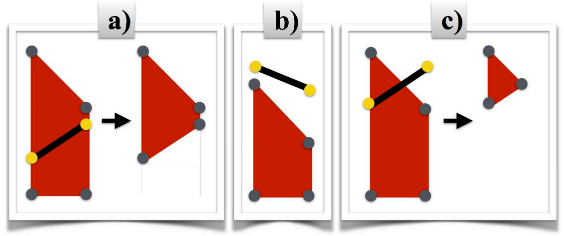

Figure 15: Visibility analysis of walls: Red polygons represent walls and black edges are parts of roof facets.

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.

## 8 CONCLUSIONS AND FURTHER WORK

Presented techniques have been successfully applied to improve city models of the cities of Krefeld and Leverkusen. However, for about 1% of buildings, enforcing planarity leads to visible changes of roofs' appearances ($z$-coordinate errors of more than 2 m). Section 6 describes a general approach to planarization that can be applied to arbitrary CityGML models. But within our workflow we can directly generate planar roof facets if we do not snap together vertices to avoid step edges. Then instead of establishing planarity, linear optimization can be 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 (3) ensure that such differences are non-negative. For such vertices $u$ und $v$, 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 $\varepsilon \max\{\omega(p) : p \in V\}/(\mu/2)$ (see Section 6 for meaning of symbols). By allowing local height changes up to $\varepsilon := \beta$, the linear program of Section 6 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.

Starting from scratch, overall processing time for all 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. With respect to worst-case situations, it could be replaced by an interior point method so that all processing steps run in polynomial time.

So far, our processing workflow is data based. We plan to extend it to a hybrid data and model driven approach. For special structures with small footprints

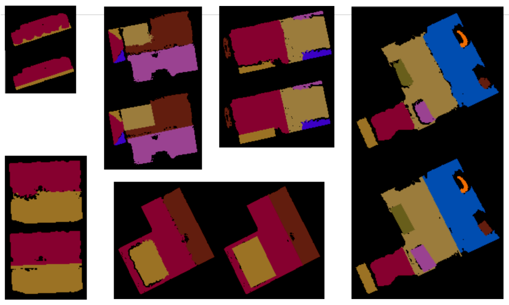
Figure 16: Preserving rectangular shapes: raster representation of roof facets without and with shape estimation

like tower spires, we are experimenting to fit objects from a catalogue. To recognize such structures, we will both use edges from orthophotos as well as the interpolated height map. Since many roof facets have rectangular or triangular shapes, we will detect and maintain such shapes, see Figure 16 for first results. In contrast to (Nan et al., 2010), we will not directly fit geometric objects to the 3D point cloud but use the 2D raster representation.

## 9 ACKNOWLEDGEMENTS

We would like to thank cadastre offices of the cities of Krefeld and Leverkusen as well as the state cadastre office of North Rhine-Westphalia for supporting us with buildings' footprints, laser scanning data and true orthophotos. Especially, Johannes Scharmann and Marco Oestereich gave us valuable support. Cadastral data are protected by copyright (Geobasisdaten der Kommunen und des Landes NRW ©Geobasis NRW 2016).

## REFERENCES

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

Arefi, H. and Reinartz, P. (2013). Building reconstruction using dsm and orthorectified images. *Remote Sens.*, 5(4):1681–1703.

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

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

Boeters, R., Ohori, K. A., Biljecki, F., and Zlatanova, S. (2015). Automatically enhancing CityGML lod2 models with a corresponding indoor geometry. *International Journal of Geographical Information Science*, 29(12):2248–2268.

Bogdahn, J. and Coors, V. (2010). Towards an automated healing of 3d urban models. In Kolbe, T. H., König, G., and Nagel, C., editors, *Proceedings of international conference on 3D geoinformation*, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, pages 13–17.

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

Demir, I., Aliaga, D. G., and Benes, B. (2015). Procedural editing of 3d building point clouds. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2147–2155, Washington, DC. IEEE Computer Society.

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

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

Finlayson, G. D., Hordley, S. D., and Drew, M. S. (2002). Removing shadows from images. In *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ECCV '02, pages 823–836, Berlin. Springer.

Goebbels, S., Pohle-Fröhlich, R., and Rethmann, J. (2017). Planarization of CityGML models using a linear program. In *Operations Research Proceedings (OR 2016 Hamburg)*, pages 1–6, Berlin. Springer, to appear.

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

Gröger, G. and Plümer, L. (2009). How to achieve consistency for 3d city models. *ISPRS Int. J. Geo-Inf.*, 15(1):137–165.

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

Guo, R., Dai, Q., and Hoiem, D. (2011). Single-image shadow detection and removal using paired regions. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2033–2040, Washington, DC. IEEE Computer Society.

Haala, N. and Kada, M. (2010). An update on automatic 3d building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65:570–580.

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

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

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

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

Mesnil, R., Douthe, C., Baverel, O., and Léger, B. (2016). From descriptive geometry to fabrication-aware design. In Adriaenssens, S., Gramazio, F., Kohler, M., Menges, A., and Pauly, M., editors, *Advances in Architectural Geometry*, pages 62–81, Zürich. vdf Hochschulverlag.

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

Perera, S. N. and Maas, N. G. (2012). A topology based approach for the generation and regularization of roof outlines in airborne laser scanning data. In Seyfert, E., editor, *DGPF Tagungsband 21*, pages 400–409, Potsdam. DGPF.

SIG3D (2014). *Handbuch für die Modelierung von 3D Objekten, Teil 1: Grundlagen*. Geodateninfrastruktur Deutschland, 0.7.1 edition.

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

Tarsha-Kurdi, F., Landes, T., Grussenmeyer, P., and Koehl, M. (2007). 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):87–92.

Tong, L., Li, M., Chen, Y., Wang, Y., Zhang, W., and Cheng, L. (2012). 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*, pages 1–5, Washington, DC. IEEE Computer Society.

Wagner, D., Wewetzer, M., Bogdahn, J., Alam, N., Pries, M., and Coors, V. (2013). Geometric-semantical consistency validation of CityGML models. In Pouliot, J. and et. al., editors, *Progress and New Trends in 3D Geoinformation Sciences*, Lecture notes in geoinformation and cartography, pages 171–192, Berlin. Springer.

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

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