Polyline Simplification with Predefined Edge Directions by Mixed Integer Linear Programs

Steffen Goebbels[1][a] and Jochen Rethmann[1]
{*Steffen.Goebbels, Jochen.Rethmann*}*@hsnr.de*

[1]*Institute for Pattern Recognition, Faculty of Electrical Engineering and Computer Science, Niederrhein University of Applied Sciences, Reinarzstr. 49, 47805 Krefeld, Germany*

Keywords: Contour Simplification, Polyline, Mixed Integer Linear Program, 3D City Model

Abstract: Mixed integer linear programs are presented that simplify polylines such that edges follow only some predefined directions from a given set. Under this constraint, solutions are computed that are closest to the given vertices, or only close to the original data, but with a minimum number of edges, or with a minimum length. The algorithms are applied to 3D building modeling from point clouds. Boundaries of roof facets (roof polygons) are simplified by considering directions of roof plane gradients and intersection lines between roof planes.

1 INTRODUCTION

Many polyline or contour simplification procedures map a long list of points to a small sub-list consisting of the vertices of the simplified polyline. In general, there are three strategies for condensing a given sequence of points into vertices of a simplified polyline: One can iteratively select points to become vertices of the simplified polyline, one can iteratively remove points until only relevant vertices remain, or one can specify the vertices of the simplified polyline by feature values that only need to be computed once.

A popular example of iterative vertex selection is the Ramer-Douglas-Peucker algorithm, see (Ramer, 1972) and (Douglas and Peucker, 1973). It connects the end vertices of the given input polyline with an edge and determines a vertex that is furthest away from that edge. If the distance exceeds a threshold value, the vertex is included in the simplified polyline, and the algorithm is applied recursively to both parts of the given polyline that are separated by the selected vertex. Other algorithms, that iteratively select vertices based on distances between vertices and straight lines through vertices are the Reumann-Witkam algorithm (Reumann and Witkam, 1973), its variant, the Opheim routine, the Lang algorithm, see (Lang, 1969), and many more.

Visvalingam's algorithm is a method of iterative vertex removal, see (Visvalingam and Whyatt, 1992): Three successive vertices define a triangle. The triangle with a smallest area is determined. If the area is below a threshold value, the middle vertex is removed. The procedure is then repeated until only the vertices of the simplified polyline remain. As no endpoints are required, this method can be applied directly to closed contours and will then produce a closed polyline (polygon). The Zhao-Saalfeld sleeve-fitting algorithm deletes vertices that lie within locally computed angle tolerances (Zhao and Saalfeld, 1997).

Reducing a polyline to every n-th vertex (nth point algorithm) is a naive example of non-iterative vertex selection. More sophisticated is the use of dominant corners, e.g. by applying curvature methods, see (Pinheiro and Ghanbari, 2010). Another example is to compute a sub-list of vertices such that the resulting polyline consists of a minimum number of edges (shortcuts) under distance constraints. Such an algorithm is presented in (Funke et al., 2017). It is based on mixed integer linear programming.

However, all these methods use vertices that are present in the given polyline. But it may be necessary to introduce different vertices if the original polyline is only a coarse approximation of a real curve and the simplified polyline has to fulfill some restrictions imposed by the underlying application scenario. The Imai-Iri algorithm (Imai and Iri, 1986) computes a polyline that is close to the given one but has a minimum number of (new) vertices. In (Aronov et al., 2005), a theoretical discussion of polyline approximation in terms of complexity is given.

[a] https://orcid.org/0000-0003-4313-9101

Our contribution consists of mixed integer linear programs (MIPs) that compute a simplified polyline with edges that run only in given directions. Whereas coordinates are represented as floating point numbers, binary decision variables are required to assign edges to given directions and vertices. The programs are allowed to introduce vertices other than the given ones, but they ensure that the simplified polyline is close to the original curve and satisfies certain criteria for optimality. The approach makes it easy to add other application specific features, such as requiring an endpoint to be on a particular line.

The algorithms are motivated by determining the boundary polygons of roof facets when deriving 3D building models from airborne laser scanning (ALS) point clouds. We assume that each roof facet is (approximately) planar, i.e., it lies on a plane. That excludes cupolas from our considerations but fits with the CityGML description standard (Gröger et al., 2012) of city models.

3D reconstruction of roofs from ALS data can be achieved by a combination of model-based and data-based approaches. Model-based methods fit parameterized standard roofs to point clouds. Data-based methods detect individual plane segments and combine them into a watertight roof. While the model-based approach results in well-structured roof topologies, they may differ significantly from reality. On the other hand, sparse ALS point clouds make it difficult to correctly locate the boundaries of individual plane segments. While ridge lines can be easily calculated by intersecting planes, step edges between plane segments of building parts with different numbers of floors or between dormers and the surrounding roof are more difficult to locate accurately.

Roof planes can be easily found by applying the RANSAC algorithm to a point cloud or by normal-based region growing. This results in a 2D map of a roof with regions representing roof facets. One may need additional region growing to complete the map, and the boundary of each region is only a somewhat noisy approximation. But typically roof facet polylines have edges that are perpendicular or parallel to the gradient of the roof plane or its neighbor, or edges that follow intersection lines between roof planes. This limits the possible directions to a few that can be defined by normal vectors, so that the MIPs can be applied to the roof facet boundaries. The performance and the usability of the approach are evaluated within a workflow for 3D city model generation.

There are many alternative solutions for obtaining roof facet contours. Recently, deep learning has been applied in a number of papers. For example, in (Nauata and Furukawa, 2020), a convolutional neural network (CNN) is applied to aerial images. But it is also combined with mixed integer linear programming to fuse geometric primitives. Deep neural networks that find edges in unstructured point clouds are described in (Bode et al., 2022) and (Årøe, 2022). See (Bode et al., 2022) for a literature review. Even complete graphs of intersection edges of roofs can be obtained using deep point features obtained with PointNet++, see (Li et al., 2022). However, many of these techniques also require some sort of post-processing and polyline simplification.

In the next section, we construct MIPs that simplify polylines under several optimization objectives. We then describe their application to the creation of 3D roof models. Finally, we evaluate the results.

2 MIXED INTEGER LINEAR PROGRAMS

Let $\vec{v}_1 = (\vec{v}_1.x, \vec{v}_1.y), \dots, \vec{v}_S = (\vec{v}_S.x, \vec{v}_S.y)$ be the vertices of a given polyline in the order of its traversal. The range of vertex coordinates can be limited to non-negative intervals. We consider the given polyline to be a representation of a contour. Since solving MIPs in general is an NP-hard task, it may be necessary to sample long contours at fewer vertices so that the number of variables can be reduced. The selection of sample points can be done with the polyline simplification algorithms described earlier. Then, thresholds must be chosen so that not too much information is lost.

The simplified polyline consists of the vertices $\vec{p}_1 = (\vec{p}_1.x, \vec{p}_1.y), \dots, \vec{p}_M = (\vec{p}_M.x, \vec{p}_M.y)$ that are connected by edges. Here, M is the maximum number of vertices to consider. Since less than M vertices may be sufficient to represent the polyline, we allow subsequent vertices to be equal. A minimum number of relevant edges with non-zero length then corresponds to a maximum number of consecutive equal vertices. If we do not minimize the number of edges, it may be possible to merge adjacent edges into larger edges. Thus, vertices can be intermediate points of these larger edges. With respect to running times of linear programs, the number M should be chosen as small as possible. Again, the polyline simplification algorithms described earlier can help to obtain an initial estimate of M. If no feasible solution exists, then one may wish to increase M.

We only allow edges that are perpendicular to one normal vector from a given list $\vec{r}_1, \dots, \vec{r}_N$. All these normal vectors are normalized to have length one: $|\vec{r}_l| = 1$ for $l \in [N] := \{1, 2, \dots, N\}$. We add vectors pointing in opposite directions: $\vec{r}_l := -\vec{r}_{l-N}$ for

$l \in \{N+1,\dots,2N\}$. These additional vectors are needed later to measure the length of the edges between points $\vec{p}_k$ and $\vec{p}_{k+1}$ with a linear constraint.

If a closed polyline (a polygon) is given, let $\vec{p}_1 = \vec{p}_M$. Otherwise and with respect to our application, each endpoint of the polyline is either assigned to an endpoint $\vec{v}_1$ or $\vec{v}_S$ (so that it coincides with it or so that their coordinates differ from each other by a maximum of ε) or it must be placed on an edge, e.g. defined by vertices Q_1, Q_2 via the linear constraint

$$\vec{p}_1 = Q_1 + \lambda(Q_2 - Q_1) \tag{1}$$

with a variable $0 \le \lambda \le 1$.

We discuss three optimization objectives. Most of the constraints are used with all objectives. We first state the common constraints and explain them afterwards. Let $C > 0$ be a large number (greater than a longest edge) and

$$a_{k,l} \in \{0,1\} \text{ for } (k,l) \in [M-1] \times [2N],$$

$$\forall_{k\in[M-1]} \sum_{l=1}^{2N} a_{k,l} = 1, \tag{2}$$

$$\forall_{(k,l)\in[M-1]\times[2N]}$$
$$-(1-a_{k,l})C \le (\vec{p}_{k+1} - \vec{p}_k)\cdot\vec{r}_l \le (1-a_{k,l})C, \tag{3}$$

$$b_{k,s} \in \{0,1\} \text{ for } (k,s) \in [M-1] \times [S],$$

$$\forall_{s\in[S]} \sum_{k=1}^{M-1} b_{k,s} = 1, \tag{4}$$

$$d_s^+, d_s^-, \delta_s^+, \delta_s^- \in \mathbb{R},\ d_s^+, d_s^-, \delta_s^+, \delta_s^- \ge 0, \text{ for } s \in [S],$$
$$\lambda_s \in \mathbb{R},\ \lambda_s \ge 0, \text{ for } s \in [S],$$

$$\begin{aligned}&\forall_{(k,s,l)\in[M-1]\times[S]\times[2N]}\\ &-C(2-a_{k,l}-b_{k,s})\\ &\quad\le (\vec{p}_k.x + (\lambda_s + \delta_s^+ - \delta_s^-)(-\vec{r}_l.y))\\ &\qquad - (\vec{v}_s.x + (d_s^+ - d_s^-)\vec{r}_l.x)\\ &\qquad\quad \le C(2-a_{k,l}-b_{k,s}),\end{aligned} \tag{5}$$

$$\begin{aligned}&\forall_{(k,s,l)\in[M-1]\times[S]\times[2N]}\\ &-C(2-a_{k,l}-b_{k,s})\\ &\quad\le (\vec{p}_k.y + (\lambda_s + \delta_s^+ - \delta_s^-)\vec{r}_l.x)\\ &\qquad - (\vec{v}_s.y + (d_s^+ - d_s^-)\vec{r}_l.y)\\ &\qquad\quad \le C(2-a_{k,l}-b_{k,s}),\end{aligned} \tag{6}$$

$$L_k \in \mathbb{R},\ L_k \ge 0 \text{ for all } k \in [M-1],$$

$$\begin{aligned}&\forall_{(k,l)\in[M-1]\times[2N]}\\ &-C(1-a_{k,l}) \le L_k - (\vec{p}_{k+1} - \vec{p}_k)\cdot(-\vec{r}_l.y, \vec{r}_l.x)\\ &\qquad \le C(1-a_{k,l})\end{aligned} \tag{7}$$

$$\forall_{(k,s)\in[M-1]\times[S]}\ \lambda_s \le L_k + C(1-b_{k,s}). \tag{8}$$

The conditions (2) and (3) deal with feasible directions for edges. If the inner product (marked with a dot) between $\vec{p}_{k+1} - \vec{p}_k$ and $\vec{r}_l$ is zero, i.e.,

$$(\vec{p}_{k+1} - \vec{p}_k)\cdot\vec{r}_l = 0, \tag{9}$$

then the edge between $\vec{p}_k$ and $\vec{p}_{k+1}$ is perpendicular to $\vec{r}_l$. Binary variables $a_{k,l}$ are used to assign a normal vector to each edge of the simplified polyline. Thus, $a_{k,l} = 1$ is equivalent to assigning $\vec{r}_l$ to the edge between $\vec{p}_k$ and $\vec{p}_{k+1}$. The condition (2) requires that exactly one normal vector is assigned to each edge. If $a_{k,l} = 1$, the condition (3) implies (9). If $a_{k,l} = 0$, then the inner product in (3) has to be within the interval $[-C, C]$. In MIPs, it is a standard trick to model conditions by using large constants C so that being an element of $[-C, C]$ does not represent a real restriction. Note that, independent of $a_{k,l}$, (3) is also not a restriction if the length of the edge is zero.

Within a certain distance, the edges of the simplified polyline must coincide with the vertices of the given polyline. We measure the distance between each given vertex $\vec{v}_s$ and exactly one edge of the simplified curve. This edge is indicated by the binary variable $b_{k,s}$, which is one. Due to (4), exactly one edge is assigned to each given vertex.

We compute the absolute value of the distance between $\vec{v}_s$ and the straight line through the associated edge but also consider the endpoints of the edge. Let $\vec{r}_l$ be the normal with length one, assigned to the edge such that $(-\vec{r}_l.y, \vec{r}_l.x)$ points into the same direction as the vector $\vec{p}_{k+1} - \vec{p}_k$ with length L_k if $L_k > 0$. Then we solve

$$\vec{p}_k + \tilde{\lambda}_s(-\vec{r}_l.y, \vec{r}_l.x) = \vec{v}_s + d_s\vec{r}_l$$

to obtain $\tilde{\lambda}_s$ and d_s. The distance of $\vec{v}_s$ to the line through the edge is $|d_s|$, and the nearest point on the straight line to $\vec{v}_s$ is $\vec{p}_k + \tilde{\lambda}_s(-\vec{r}_l.y, \vec{r}_l.x)$. This is the orthogonal projection of $\vec{v}_s$ to the line. If this point lies inside the edge, i.e., $\tilde{\lambda}_s \in [0, L_k]$, we measure the distance of $\vec{v}_s$ to the edge with the value $|d_s|$. Otherwise, we consider the distance between this point and $\vec{p}_k$, which is $|\tilde{\lambda}_s|$, and the distance to $\vec{p}_{k+1}$, which is $|\tilde{\lambda}_s - L_k|$. Then, we measure the distance of $\vec{v}_s$ and the edge as $|d_s| + \min\{|\tilde{\lambda}_s|, |\tilde{\lambda}_s - L_k|\}$. We translate this into linear constraints.

Since the computation of absolute values is nonlinear, another standard trick of linear programming is to write potentially negative numbers as the difference of two non-negative numbers. In the context of using these variables in an objective function where the sum of the variables must be minimal, one of these variables will be zero and the other variable will contain the absolute value which is also the sum of both variables. For each point $\vec{v}_s$, we introduce two pairs of non-negative variables d_s^+, d_s^- and δ_s^+, δ_s^-, which appear in a sum that is minimized by the

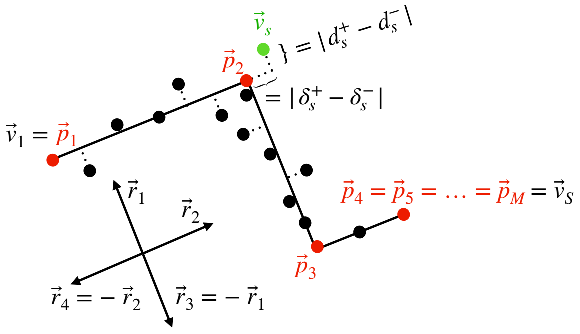


Figure 1: The dotted lines represent the shortest distances of points to polyline edges.

objective functions. These non-negative variables are used to express the distance of $\vec{v}_s$ to the corresponding edge, $d_s = d_s^+ - d_s^-$, and for an optimal solution: $\delta_s^+ = \delta_s^- = 0$ if the orthogonal projection lies inside the edge, and $\delta_s^+ + \delta_s^- = \min\{|\tilde{\lambda}_s|, |\tilde{\lambda}_s - L_k|\}$ otherwise.

The conditions (5) and (6) only become relevant, if $b_{k,s} = a_{k,l} = 1$. Then normal $\vec{r}_l$ will be assigned to the edge between $\vec{p}_k$ and $\vec{p}_{k+1}$, and the direction of the edge is given by $(-\vec{r}_l.y, \vec{r}_l.x)$. The given vertex $\vec{v}_s$ is also assigned to the edge. The conditions (5) and (6) are used to compute the point $\vec{p}_k + \tilde{\lambda}_s(-\vec{r}_l.y, \vec{r}_l.x)$ that is closest to $\vec{v}_s$ and lies on the straight line. Since $\vec{r}_l$ is perpendicular to the line, this point also has a representation of $\vec{v}_s + (d_s^+ - d_s^-)\vec{r}_l$, and by solving these two linear equations (5) and (6) one obtains $\tilde{\lambda}_s$ and $d_s = d_s^+ - d_s^-$. The distance between $\vec{v}_s$ and the straight line is $|d_s^+ - d_s^-|$ and will equal $d_s^+ + d_s^-$ for an optimal solution.

Note that we cannot use the direction vector $\vec{p}_{k+1} - \vec{p}_k$ instead of $(-\vec{r}_l.y, \vec{r}_l.x)$ because we would multiply the structure variables $\vec{p}_{k+1}$ and $\vec{p}_k$ with $\tilde{\lambda}_s$. Thus, we would loose linearity.

The normal $\vec{r}_l$ assigned to the edge between $\vec{p}_k$ and $\vec{p}_{k+1}$ can be chosen so that vectors $(-\vec{r}_l.y, \vec{r}_l.x)$ and $\vec{p}_{k+1} - \vec{p}_k$ point in the same direction (constraints (7) and $L_k \geq 0$), because we have added negative normals to the initial list of normal vectors. The inner product between vectors pointing in the same direction is non-negative, and the length of the edge is given by (7):

$$\begin{aligned} L_k &:= (\vec{p}_{k+1} - \vec{p}_k) \cdot (-\vec{r}_l.y, \vec{r}_l.x) \\ &= |\vec{p}_{k+1} - \vec{p}_k||(-\vec{r}_l.y, \vec{r}_l.x)|\cos(0) = |\vec{p}_{k+1} - \vec{p}_k|. \end{aligned}$$

The factor $\tilde{\lambda}_s$ is decomposed into the sum $\lambda_s + \delta_s^+ - \delta_s^-$. With condition (8), there is $\lambda_s \in [0, L_k]$. If and only if the nearest point lies within the edge between $\vec{p}_k$ and $\vec{p}_{k+1}$, the variables δ_s^+ and δ_s^- become zero. Otherwise, in conjunction with the objective functions, $\delta_s^+ + \delta_s^-$ is the absolute distance from the closest point to $\vec{v}_s$ on the straight line through the edge to the nearest endpoint of the edge, see Figure 1. Thus,

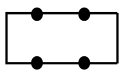


Figure 2: Four vertices are connected with a minimum number of horizontal and vertical edges of a closed polyline.

for an optimal solution, each point $\vec{v}_s$ is closer to its associated edge than

$$\sqrt{(d_s^+ - d_s^-)^2 + (\delta_s^+ - \delta_s^-)^2} \leq \sqrt{2}\max\{d_s^+, d_s^-, \delta_s^+, \delta_s^-\},$$

because two of the four variables become zero.

For performance reasons, we define SOS 1 sets of binary variables that sum up to at most one. For each $k \in [M-1]$ variables $a_{k,l}$, $l \in [2N]$, and for each $s \in [S]$ variables $b_{k,s}$, $k \in [M-1]$, constitute such sets.

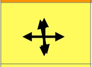
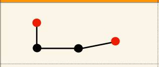
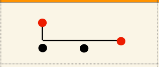
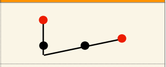
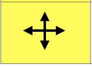
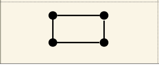
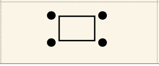
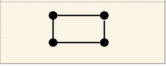
normals	min. dist.	min. length	min. edges
			
			

Figure 3: The three objective functions lead to different results: Red vertices are fixed endpoints.

We discuss the following optimization goals for which Figure 3 shows the principal differences:

1. [**min. dist.**] Find a polyline (with a limited number of vertices) that minimizes a linear combination of the distances to the given points and its length. The coefficients $\alpha > \beta > 0$ of the linear combination can be chosen so that the focus is on the minimization of the distances. However, the length of the polyline must also be considered, so that the graph in Figure 2 will not be an optimal solution.

   minimize
   $$\alpha \sum_{s=1}^{S}(d_s^+ + d_s^- + \delta_s^+ + \delta_s^-) + \beta\left(\sum_{k=1}^{M-1} L_k\right). \tag{10}$$

2. [**min. length**] Find a polyline of minimum length so that all given points are within a threshold distance defined by $\varepsilon > 0$. To this end, let $d_s^+, d_s^-, \delta_s^+, \delta_s^- \leq \varepsilon$. Then
   $$\text{minimize} \left(\sum_{k=1}^{M-1} L_k\right).$$
   However, we also want to minimize the distances to given points. With (cf. (10))

   minimize
   $$\left(\sum_{k=1}^{M-1} L_k\right) + \frac{\mu}{S \cdot 2\varepsilon}\sum_{s=1}^{S}(d_s^+ + d_s^- + \delta_s^+ + \delta_s^-) \tag{11}$$

we find polylines that may slightly exceed the min length up to $\mu > 0$. But they are generally closer to the given points. If one deals with closed contours, then the objective will result in a polygon that may be slightly smaller than the given polyline, see Figure 3.

3. [**min. edges**] Find a polyline with a minimum number of edges such that all given points are within a threshold distance defined by ε. Similar to before, among all polylines that meet these conditions, we select one that is closest to the given points. To also count edges, we add constraints $d_s^+, d_s^-, \delta_s^+, \delta_s^- \leq \varepsilon$ and

$$c_k \in \{0,1\} \text{ for } k \in [M-1]$$

$$\forall_{k\in[M-1]}$$

$$-C(1-c_k) \leq \vec{p}_k.x - \vec{p}_{k+1}.x \leq C(1-c_k) \quad (12)$$

$$-C(1-c_k) \leq \vec{p}_k.y - \vec{p}_{k+1}.y \leq C(1-c_k) \quad (13)$$

$$c_k + \sum_{s=1}^{S} b_{k,s} \geq 1. \quad (14)$$

Then the task is to

$$\text{maximize} \left(\sum_{k=1}^{M-1} c_k\right) - \frac{1}{4S\varepsilon} \sum_{s=1}^{S} (d_s^+ + d_s^- + \delta_s^+ + \delta_s^-). \quad (15)$$

If and only if the consecutive points $\vec{p}_k$ and $\vec{p}_{k+1}$ are equal, $\vec{p}_k = \vec{p}_{k+1}$, the binary variable c_k can and will be set to one, see (12) and (13) in conjunction with the objective function. The primary optimization goal in (15) is to maximize the number of equal points, i.e., to minimize the number of vertices of the simplified polyline. As a secondary objective in (15), the sum of the distances between the given vertices and the edges of the simplified polyline is minimized. Each of the S summands is bounded by ε. Thus, the sum does not exceed $S\varepsilon$ and its factor limits the size at 0.5. As a consequence, it is more important to save a vertex then to have edges that are closer to given vertices. However, among all solutions with a minimum number of vertices, a solution with edges closest to the given vertices is chosen (in the sense of the l_1-norm realized by the secondary objective). Without the condition (14), the polygon in Figure 2 would be optimal since it is allowed to use edges that are far away from all sample points. To avoid this, (14) prescribes every edge of positive length to be assigned to a given vertex. In some tests with the solver GLPK[1], the running time was reduced significantly by an additional constraint to avoid the assignment of given points with edges of zero length:

$$\forall_{(k,s)\in[M-1]\times[S]}\ b_{k,s} \leq 1 - c_k.$$

If one uses too few sample points, an optimal solution may not be the intended one, see Figure 4.

In general, our goal is to map simple polylines to simple polylines. However, the MIPs do not check for self-intersections, so complex polylines may occur. Self-intersection must be corrected in a post-processing step.

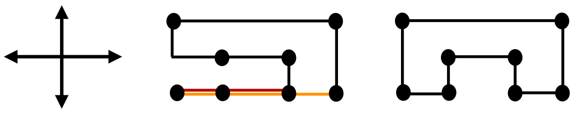


Figure 4: The first polygon has seven but the second polygon has eight edges. Thus the second layout is not optimal with respect to (15). However, the second polyline is shorter than the first one.

3 APPLICATION TO ROOF FACET BOUNDARIES

We detect roof planes using RANSAC on ALS points of roof segments with homogeneous gradient direction. Then we generate a 2D map showing regions of points belonging to individual roof facets and fill the map with region growing. Each pixel represents a square of $10\,\text{cm} \times 10\,\text{cm}$. Then we detect the contours of the regions and mark the critical points where more than two facets are adjacent (counting the outside of the building footprint as a facet). All non-black points are critical in Figure 5. Points near the intersection points of three or more adjacent roof planes are identified with the intersection points. Now, contour segments between critical points are simplified using the MIPs. In what follows, we describe how the parameters and normals are chosen, see Figure 6 for some results.

A simpler but somewhat similar MIP approach to ours is used in (Goebbels and Pohle-Fröhlich, 2017), where the edges are adjusted to the cadastral footprint directions they are already close to. However, only existing vertices are moved slightly within given bounds and the number of edges basically remains fixed. Here, we also consider a longest footprint direction and its perpendicular direction but only when a polyline separates two flat roofs. Otherwise we use the gradients of non-zero length of the two roof facets

[1]GLPK LP/MIP Solver 5.0, `http://www.gnu.org/software/glpk/glpk.html` (all websites accessed: September 14, 2023)

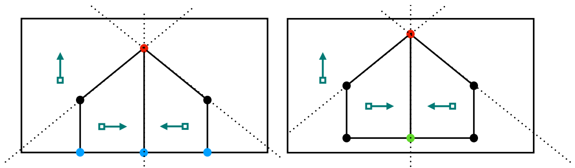


Figure 5: Typical roof, projected onto the 2D ground plane: A dormer is placed in a surrounding roof facet. The red vertex is an intersection point of three roof planes. Their facets are adjacent to the vertex. Blue vertices lie on a cadastral footprint edge. Green is used for other critical points that are adjacent to more than two facets. Dotted lines correspond with intersection lines between roof planes. Arrows indicate gradient directions.

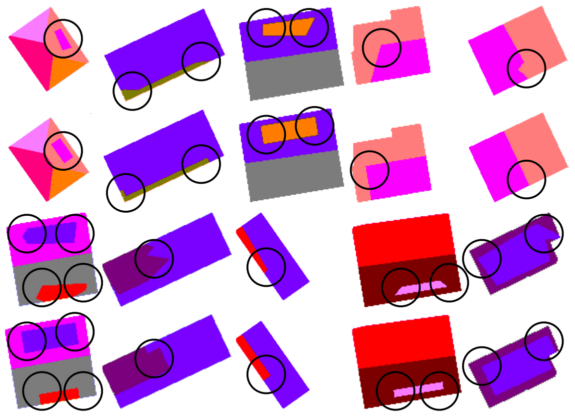


Figure 6: Examples for roof layout simplification with $\varepsilon = 6$ pixels. Closed contours were optimized with the min. edges goal (15) and all other contours were treated with the min. length objective (11).

that are separated by the polyline to be simplified. We also consider their perpendicular directions and the direction of an intersection line of the two facet planes, if it exists. This results in a list of normals $\vec{r}_l$.

The roof facet boundaries form a graph. Before applying optimization procedures, we reduce the number of edges with the Ramer-Douglas-Peucker algorithm while keeping in particular the critical points as vertices. We discuss several cases:

The first case deals with a closed polyline that separates exactly two roof facets. Often, such a polygon is a rectangle defining a dormer. This polygon is either simplified by the min. edges or by the min. dist. objective. Minimizing the length of the polyline would result in polygons that are slightly too small, see Figure 3. The other cases deal with open polylines that have critical points as endpoints.

Open polylines can be simplified with any of the three objective functions. We need to determine how to handle their endpoints.

- An endpoint must remain in place, if it is an intersection point of three or more adjacent roof planes or a vertex of the cadastral footprint.

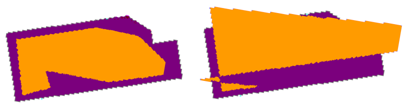


Figure 7: An example with self-intersection and intersection with the enclosing building footprint.

- For each other endpoint that lies on a footprint edge, we add a constraint that the simplified polyline must end on that line, cf. (1). In Figure 5, the blue critical points have to stay on the footprint.
- In our test data (see next section), the previous two cases already cover over 90% of the contour segments that have more than one edge. In principle, the remaining endpoints can be moved. However, since we are optimizing contour segments iteratively, we must avoid undoing improvements that have already been made. If such a remaining endpoint is used for the first time in an optimization problem, its coordinates are allowed to vary by ε. However, if it has already been used, it may be moved by a maximum of ε on all the straight lines that lie on the edges where it was the endpoint in a result of a previous optimization problem. If an endpoint already lies on two straight lines with linearly independent directions, it is fixed.

The position of an endpoint of an open polyline may depend on two optimization problems that are solved sequentially. This is much faster than solving combined problems that deal with multiple polylines.

The resulting polylines may have self-intersections. Also intersections with boundaries of enclosing roof facets and the cadastral footprint are possible, see Figure 7. Such intersections become unlikely if small distance bounds ε are used, contours are sampled with sufficiently many points, and normal vectors fit with the contour. We generally remove intersections in a post-processing step.

4 EXPERIMENTS

We evaluated the approach for one square kilometer of the city center of Krefeld with 5,467 buildings shown in Figure 8 on a computer with a 2.3 GHz dual-core Intel Core i5 with 16 GB of RAM. The corresponding ALS point cloud was provided by GeoBasis NRW[2]. The parameters are set as follows: $C = 1000$, $\alpha = C$, $\beta = \varepsilon$, $\mu = 1$, $\varepsilon := 3$ pixels. Since we want to reduce complexity, the maximum number M of polyline vertices is chosen to be equal to the number S of

[2]`https://www.bezreg-koeln.nrw.de/geobasis-nrw`

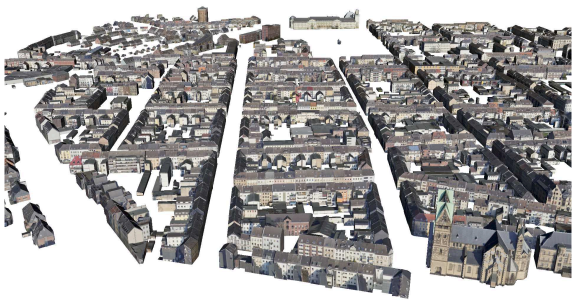


Figure 8: The final 3D model of one square kilometer.

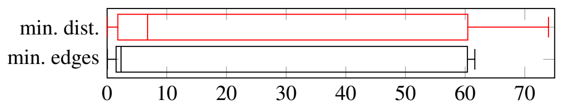


Figure 9: Running times for 111 closed contours with existing solutions in seconds, red: 111 solutions for the minimum distances goal (10), black: 95 solutions of the min. edges objective (15)

given vertices if the contour is not closed. Otherwise we set $M := S+1$ to ensure that the startpoint and endpoint of the simplified polyline are the same. We have integrated the MIPs into a workflow for creating 3D city models using the C-API of the IBM CPLEX 12.8.0. optimizer[3]. Instead of working with optimal solutions, we are satisfied with a best solution found within a time-limit of 60 seconds. This time-limit is checked with a callback function that is invoked by CPLEX at irregular time steps. Thus, running times may exceed 60 seconds slightly. The optimization problems had between 21 and 2,118 variables including 6 to 1,813 binary variables. We used up to 60,796 constraints.

We simplified 111 closed polylines with the two objective functions (10) and (15). In 33 and 34 problems, respectively, the time-limit was reached. Nevertheless, we obtained a feasible solution for all of the problems when minimizing distances with the min. dist. objective (10), while no feasible solution was obtained for 16 instances with the min. edges goal (15). The reasons for this behavior are the same as for open polylines and are discussed below. The running times are compared in Figure 9.

Open polylines had an average of 18,352 fixed endpoints, 38,280 endpoints that were restricted to vary on edges, and 1,343 endpoints that were allowed to move freely within the coordinate-wise tolerance ε. Figure 10 compares the running times due to the different objectives. Problems with the min. dist. condition (10) took slightly longer to solve than problems with the other two objectives. However, more feasible

[3]`https://www.ibm.com/de-de/analytics/cplex-optimizer`, the newer CPLEX version 22.1.1 showed almost exactly the same running times

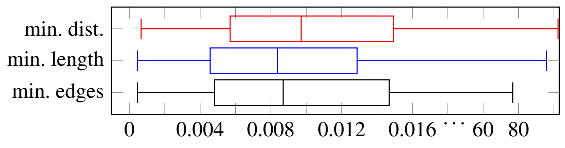


Figure 10: Running times for open contours with existing solutions in seconds, red: minimum distances optimization with (10), blue: minimum length objective (11), black: minimum number of edges goal (15).

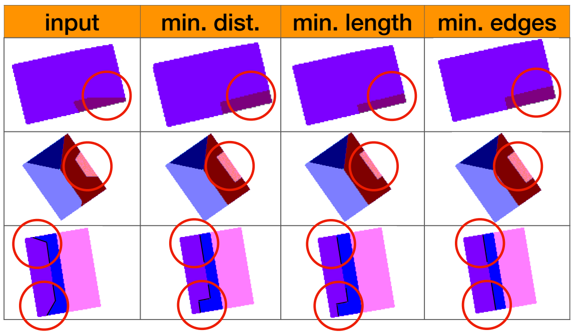


Figure 11: The three optimization goals may result in slightly different roof layouts.

solutions were found for this objective (10) than for the other goals because distances were not bounded by ε, see Table 1. However, in some examples this resulted in larger changes which can be avoided by also requiring $d_s^+, d_s^-, \delta_s^+, \delta_s^- \leq \varepsilon$. In general, the qualitative results differ only slightly between the chosen objective functions, see Figure 11. If, unlike in our test scenario, models are created interactively, all three optimization goals can be offered as tools so that the best fitting result can be selected.

Table 1: Feasible solutions for polylines with critical endpoints, found with the three objectives; problem instances that have exceeded the time-limit also contribute to the rows “feasible solutions” and “no solutions found”. However, apart from the small number of aborted problems with reached time-limit, feasible solutions are optimal and “no solution found” means that the associated problem really has no solution.

	min. dist.	min. length	min. edges
feasible solutions	18,331	14,978	15,218
no solution found	10,591	13,665	14,180
time-limit reached	328	295	256

One reason for the significant number of problem instances without feasible solutions is that the contours do not sufficiently match with the prescribed directions defined by the roof plane gradients and their intersections. This can be an effect of the region growing method applied previously. Especially, it occurs when small roof facets are not detected by RANSAC, so that corresponding regions must be filled with adjacent facets. The maximum tolerance ε was chosen to be small enough to avoid inconsistencies, but the behavior does not change if the tolerance is moderately increased from three to six pixels without increasing the number of vertices M.

5 CONCLUSIONS

We have introduced MIPs that modify polylines under directional constraints. The applicability of the programs has been demonstrated in the context of 3D modeling of building roofs. In this scenario, we had to deal with a large number of contours. Therefore, short running times of the individual MIPs were important. Each contour was already simplified so that it could be described with a few sampled polyline vertices. In most cases, the MIPs did not reduce the number of vertices. The maximum reduction was 21 vertices. This resulted in MIP running times of a few milliseconds. When applying the MIPs to polylines with more vertices, longer running times can be expected.

Polyline simplification based on normals is not limited to 3D building reconstruction. An everyday example is public transport maps that show generalized paths instead of exact ones.

Acknowledgements

The authors are grateful to Dagmar Schumacher for proof-reading and to Udo Hannok and Philipp Blumenkamp from the Krefeld land registry office for providing us with oblique aerial images.

REFERENCES

Årøe, A. L. (2022). *Detection of Edge Points of Building Roofs from ALS Point Clouds*. Norwegian University of Science and Technology (PhD thesis), Trondheim.

Aronov, B., Asano, T., Katoh, N., Mehlhorn, K., and Tokuyama, T. (2005). Polyline fitting of planar points under min-sum criteria. In Fleischer, R. and Trippen, G., editors, *Proc. ISAAC 2004: Algorithms and Computation*, volume 3341 of *LNCS*, pages 77–88, Berlin, Heidelberg. Springer.

Bode, L., Weinmann, M., and Klein, R. (2022). BoundED: Neural boundary and edge detection in 3D point clouds via local neighborhood statistics. *arXiv*, arXiv.2210.13305:1–20.

Douglas, D. and Peucker, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122.

Funke, S., Mendel, T., Miller, A., Storandt, S., and Wiebe, M. (2017). Map simplification with topology constraints: Exactly and in practice. In Fekete, S. and Ramachandran, V., editors, *Proc. 19th Workshop on Algorithm Engineering and Experiments 2017 (ALENEX17)*, pages 185–196, Red Hook, NY. Curran Associates.

Goebbels, S. and Pohle-Fröhlich, R. (2017). Quality enhancement techniques for building models derived from sparse point clouds. In *Proc. 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 1: GRAPP, (VISIGRAPP 2017)*, pages 93–104. INSTICC, SciTePress.

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.

Imai, H. and Iri, M. (1986). An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162.

Lang, T. (1969). Rules for robot draughtsmen. *The Geographical Magazine*, 42(1):50–51.

Li, L., Songa, N., Sun, F., Liu, X., Wang, R., Yaoa, J., and Cao, S. (2022). Point2roof: End-to-end 3D building roof modeling from airborne LiDAR point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 193:17–28.

Nauata, N. and Furukawa, Y. (2020). Vectorizing world buildings: Planar graph reconstruction by primitive detection and relationship inference. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J., editors, *Proc. Computer Vision–ECCV 2020: 16th European Conference, Part VIII*, number 12353 in LNCS, Cham. Springer.

Pinheiro, A. M. G. and Ghanbari, M. (2010). Piecewise approximation of contours through scale-space selection of dominant points. *IEEE Transactions on Image Processing*, 19(6):1442–1450.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256.

Reumann, K. and Witkam, A. (1973). Optimizing curve segmentation in computer graphics. In Gunther, A., Levrat, B., and Lipps, H., editors, *Proc. International Computing Symposium, Davos*, pages 467–472, New York, NY. Elsevier.

Visvalingam, M. and Whyatt, J. D. (1992). Line generalisation by repeated elimination of the smallest area. *Cartographic Information Systems Research Group, University of Hull.*

Zhao, Z. and Saalfeld, A. (1997). Linear-time sleeve-fitting polyline simplification algorithms. In *Proc. AutoCarto 13, Seattle, WA*, pages 214–223, Maryland. American Congress on Surveying and Mapping & American Society for Photogrammetry and Remote Sensing.