DGK Deutsche Geodätische Kommission
der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 723

Richard Guercke

Optimization Aspects in the Generalization of 3D Building Models

München 2014

Verlag der Bayerischen Akademie der Wissenschaften
in Kommission beim Verlag C. H. Beck

ISSN 0065-5325 ISBN 978-3-7696-5135-5

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover
ISSN 0174-1454, Nr. 312, Hannover 2014

DGK Deutsche Geodätische Kommission
der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 723

Optimization Aspects in the Generalization of 3D Building Models

Von der Fakultät für Bauingenieurwesen und Geodäsie
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation

von

Dipl.-Ing. Richard Guercke

München 2014

Verlag der Bayerischen Akademie der Wissenschaften
in Kommission bei der C. H. Beck'schen Verlagsbuchhandlung München

ISSN 0065-5325 ISBN 978-3-7696-5135-5

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover
ISSN 0174-1454, Nr. 312, Hannover 2014

Adresse der Deutschen Geodätischen Kommission:

Deutsche Geodätische Kommission

Alfons-Goppel-Straße 11 ● D – 80 539 München

Telefon +49 – 89 – 23 031 1113 ● Telefax +49 – 89 – 23 031 - 1283 / - 1100

e-mail hornik@dgfi.badw.de ● http://www.dgk.badw.de

Prüfungskommission

Vorsitzender:	Univ.Prof. Dr.-Ing. habil. Jürgen Müller
Referentin:	Univ.Prof. Dr.-Ing. habil. Monika Sester
Korreferenten:	Univ.Prof. Dr. rer. nat. Thomas Kolbe
	Univ.Prof. Dr.-Ing. habil. Christian Heipke

Tag der Promotion: 17.10.2013

ISSN 0065-5325 ISBN 978-3-7696-5135-5

Abstract

Different official and commercial providers of geographic information are currently performing extensive surveys to capture 3D building structures. These building models are offered to private users to enable them to get a visual impression of a site from a distant place but also to visualize current plans for changes.

They are also of great interest for scientific and commercial customers because the increasing availability of computing power makes it possible to perform simulations and analysis on increasingly more detailed models. In the context of (3D) city models, this could, for example, mean that a simulation tool that could only handle flat roofs in a previous version may now be able to produce better results using basic roof shapes for each building part.

With an increasing amount of details that have to be incorporated in an analysis, the surveying effort that is needed to capture the features that are needed for the application will increase dramatically. In such a situation, it would be sensible to use an existing highly detailed model and extract the features needed for the application from this model without having to launch another surveying campaign. This is the aim of generalization: *Reduce the complexity of the data set to a minimum amount of data while retaining a maximum amount of information of relevance for the application at a semantic complexity that can be handled by the application.*

This aim inherently includes an optimization problem with constraints: We want to *minimize* the size of the data set and *maximize* the remaining amount of information of importance for the application (conflicting goals) under the constraint that the resulting features can be handled by the application.

It will turn out in the course of this thesis that this problem is NP-hard even for the most simple versions of some very basic problems – the aggregation of simple building models and the simplification of simple facade structures. A powerful tool for finding globally optimal solutions for such complex optimization problems and proving the optimality of the solution is Mixed Integer Programming (MIP). In this thesis, MIP-based approaches for the building aggregation and facade simplification problem are developed; these optimizing approaches may be computationally expensive, but they can be used to benchmark heuristic approaches.

Since the different features (like windows, roofs, stairways etc.) of which 3D building models are composed are often quite different in their shape, structure and function and heuristic approaches will usually perform best if they are as specific as possible, it is a promising approach to try to combine heuristic approaches for the simplification of the individual features to build a high-quality generalized model.

For this reason, an extensible basic infrastructure for combining different generalization modules for different parts of a hierarchical feature model is presented in this thesis. In this infrastructure, generative aspects of the underlying basic model are used to reduce the complexity of conflict resolution in the generalization process.

Besides the formulation of the two sample applications - aggregation of LOD-1 building models and simplifying cell-based facade models - as MIP problems heuristic approaches for the LOD-1 aggregation problem were developed and the optimal result from the solution of the MIP problem evaluated as a reference. It turned out that for this relatively simpleProblem already a simple locally optimized iterative solution produced very good results.

Keywords:
Generalization, 3D Building Model, City Model

Zusammenfassung

In den letzten Jahren führen verschiedene öffentliche und private Anbieter von Geodaten vermehrt Vermessungskampagnen zur Erfassung der dreidimensionalen Struktur von Gebäuden durch. Diese Daten dienen beispielsweise dazu, Nutzern einen visuellen Eindruck von der Situation vor Ort zu vermitteln – etwa Touristen, die eine Stadt bereisen wollen, aber auch Polizisten und Feuerwehrleuten vor einem Einsatz.

Diese Daten sind auch für wissenschaftliche und geschäftliche Abnehmer von zunehmendem Interesse, weil die steigende Leistungsfähigkeit moderner Rechnerhardware Simulationen auf immer detaillierteren Modellen ermöglicht.

Mit zunehmender Menge an Details, die in einer Analyse mit einbezogen werden müssen, steigt der Aufwand für die Erfassung der benötigten Objekte deutlich an. In einer solchen Situation ist es in vielen Fällen sinnvoll, ein bestehendes hochdetailliertes Modell wiederzuverwenden und die für die Anwendung benötigten Modellteile zu extrahieren, ohne eine weitere Erfassungskampagne starten zu müssen. Das ist das Ziel der Generalisierung: *Das Verringern der Komplexität der Daten auf eine minimale Datenmenge unter Erhaltung eines maximalen Informationsumfangs für die Anwendung auf einem semantischen Komplexitätsniveau, das von der Anwendung verarbeitet werden kann.*

Dieses Ziel beschreibt ein Optimierungsproblem mit Nebenbedingungen: Wir wollen die Datenmenge minimieren und den verbleibenden Informationsgehalt maximieren (widersprüchliche Ziele) unter der Nebenbedingung, dass die ausgegebenen Objektklassen von der Anwendung verarbeitet werden können. Im Rahmen dieser Arbeit wird am Beispiel der Aggregation von (LoD-1-)Gebäudemodellen und der Vereinfachung von einfachen Fassadenstrukturen gezeigt, dass bereits einfache Versionen grundlegender Teilprobleme NP-hart sind.

Ein leistungsfähiges Werkzeug für die Suche nach global optimalen Lösungen für komplexe Optimierungsprobleme dieser Art ist *Mixed Integer Programming* (MIP). In dieser Arbeit werden MIP-basierte Ansätze für die genauer untersuchten Teilprobleme – die Aggregation von LoD-1-Modellen und die Vereinfachung von Fassadenstrukturen – entwickelt. Diese optimierungsbasierten Ansätze können rechenintensiv sein, aber sie können als Referenz zur Bewertung heuristischer Verfahren verwendet werden.

Da die verschiedenen Objekte (wie Fenster, Dächer, Treppen usw.), aus denen 3D-Gebäudemodelle zusammengesetzt sind, oft sehr unterschiedlich in ihrer Form, Struktur und Funktion sind und heuristische Ansätze in der Regel zu den besten Ergebnissen führen, wenn sie möglichst spezifisch auf ein Problem zugeschnitten sind, ist es ein vielversprechender Ansatz, heuristische Ansätze zur Vereinfachung der einzelnen Objektklassen zu kombinieren, um qualitativ hochwertige generalisierte Modelle zu erzeugen.

Aus diesem Grund wird im Rahmen dieser Arbeit eine erweiterbare Basis-Infrastruktur für die Kombination verschiedener Module zur Generalisierung der einzelnen Teile eines hierarchischen Modells vorgestellt. In dieser Infrastruktur werden generative Aspekte des zugrundeliegenden Modells genutzt, um die Komplexität des Konfliktsauflösung in der Generalisierung zu reduzieren.

Neben der Formulierung der beiden Beispielanwendungen – Aggregation von LOD-1-Gebäudemodellen und Vereinfachung zellenbasierter Fassadenmodelle – als MIP-Probleme wurden heuristische Ansätze für das Problem der Aggregation von LOD-1-Modellen entwickelt und mit dem optimalen Ergebnis aus der Lösung des MIP-Problems als Referenz evaluiert. Dabei stellte sich heraus, dass für dieses Problem bereits eine einfache lokal optimierende iterative Lösung sehr gute Ergebnisse brachte.

Schlagworte:
Generalisierung, 3D Gebäudemodelle, Stadtmodelle

Inhaltsverzeichnis

1. Introduction

1.1. Motivation

With the growing availability of computing power, increasingly more fine-grained simulations of physical phenomena are conducted, especially in research areas of special economic or social interest. Beyond a certain level of detail in the analysis, results that meet the requirements and merit the additional effort can only be achieved if true three-dimensional building models with detailed facade and roof structures are used.

One example for this trend is the INSPIRE (EU, 2007) directive of the European union that requires all member states to perform noise emission surveys and simulations based on 3D building models and detailed road and traffic models. Further fields of propagation analysis include the calculation of coverage areas for different radio antennas for cellphone or data service base stations and shading analysis for estimating the potential for photovoltaic energy production on roofs.

Because of such special application requirements and the increased 3D surveying activities of both official and commercial data providers, highly detailed 3D building and city models are going to be available in the close future.

In order to make the huge surveying effort accessible to different users, two basic steps are necessary: the augmentation of the data set with application-specific data that will in most cases come from external sources (data integration) and a reduction of the complexity of the data set to meet the requirements of the application (generalization).

There are two main reasons for generalization:

- **Data volume** – If a model is more detailed than necessary for an application, the additional effort for the computations can increase beyond a sensible limit for the given application without yielding better (often even producing worse) results.
- **Semantic complexity** – If a model contains structures that the application is not prepared to deal with, it will fail. This can either happen because an algorithms is faced with a feature class that it does not know and cannot handle or because the algorithms cannot additional detail well and produce worse results than for a simplified model.

The definition of generalization introduced in the abstract *"Reduce the complexity of the data set to the minimum necessary volume at a semantic complexity that can be handled by the application."* implies an optimization problem; this optmization problem is investigated in this thesis for the generalization of 3D building models.

In this problem statement, the second part referring to the problem of semantic complexity is rather a hard constraint: The result of the generalization has to fulfill minimum requirements of the application that make sure that the resulting data set can be used.

The first goal in the definition is a tradeoff between possibly better application results because of the availability of additional information, computational cost and the risk of cluttering described above.

In order to keep the complexity of this optimization problem – especially the risk of silent failures – under control, this problem is usually simplified by using hard constraints (in most cases in the form of minimum size thresholds for features) to ensure that the client application can handle the generalized model.

Note that even in this slightly simplified form, the problem still remains computationally complex: We will see in the course of this thesis that even the most simple forms of some of the most basic sub-tasks in the 3D building generalization problem are NP-hard (i.e. at least as complex as the well-known Traveling Salesman problem and probably not solvable in polynomial time).

From the analysis of the simple problems, we can conclude that most sub-problems in the generalization process are NP-hard because almost all of them will include multiple independent decisions (e.g. between the application of different generalization operators or between assigning features to different groups), leading to a *combinatorial optimization problem* which is often the root cause for the NP-hardness.

In order to tackle this complexity, a hierarchical approach is introduced in this thesis in which different generalization algorithms can be combined to generate a simplified model. For the two sub-problems that are shown explicitly to be NP-hard in this thesis, the aggregation of LoD 1 building models and the simplification of facade structures, approaches based on Mixed Integer Programming (MIP) for finding exact optima for a given objective function are developed.

1.2. Overview

After this introduction, chapter 2 covers some basic concepts in the context generalization and optimization. In the next chapter, different ways to model geometry and 3D building models are discussed; in the context of this thesis, especially the distinction between direct and generative modeling is relevant because generative modeling supports the concept of minimum parameterization that simplifies the generalization process. CityGML is the OGC (Open Geospatial Consortium) standard and the most important current format for the exchange of city models; its underlying model has a large basic feature set from which semantically rich models can be constructed.

Chapter 4 covers different approaches in the field of 3D building generalization. In most publications up to date, algorithms for special aspects of the 3D building generalization problem are introduced. The most frequently investigated fields are LoD 1 (only flat roof structures) building model simplification, simplification of models consisting of collections of wall and roof surfaces, and special aspects of large-scale generalization based on semantic features.

In the next chapter, a hierarchical building model is introduced together with a callback-based infrastructure for combining different simplification approaches to build a generalized model. This modeling and generalization infrastructure is designed to be extensible for different feature types and generalization algorithms.

In chapter 6, an algorithm for the simplification of building footprints based on a full-spectrum Hough Transformation and a flexible least-squares fitting scheme is presented in which relations like parallelity and collinearity are emphsized because they are prominent features in most building footprints. Such a simplified fooprint be used, for example, to provide more meaningful input footprints for generalization algorithms for LoD 1 building models.

In chapter 7, the aggregation of LoD 1 building models is investigated as an optimization problem, and a translation of this application to a Mixed Integer Programming (MIP) problem is introduced. The optimal solutions obtained using MIP software are used to evaluate heuristic approaches that are also presented in this chapter.

In chapter 8, two different MIP-based approaches to the problem of facade structure homogenization are introduced and compared. The first one based on template instancing describes the problem more directly than the second one based on a flow model; the MIP solver could, however, solve considerably more complex instances of the second MIP representation wihtout running out of memory, probably because fast and effective heuristics for finding good solutins for flow problems are integrated in the solver and the first representation contained too many ambiguities.

2. Basics

2.1. Resolution and accuracy

Resolution and *accuracy* are clearly distinguished principles (JCGM, 2008): While the resolution of a measurement is the smallest distinguishable unit, the accuracy is an upper bound to the difference between a measurement and the true value of the phenomenon that was measured.

An additional term that should not be confused with resolution and accuracy is *precision*: The precision of a measurement is the interval in which repeated measurements will fall under equal conditions. A stopwatch may, for example, have a resolution and perhaps even an internal accuracy (reliability of the measurement of the time between the impulses generated by pressing the button(s)) of $1ms$ (3 digits for decimal fractions of a second), but the precision of the system of a person measuring the duration of a given event is worse than $0.1s$ because it is dominated by the time the person needs to react and press the buttons.

In cartography, the concepts of resolution and accuracy are implicitly incorporated in the recommendations and requirements associated with different *map scales*: The scale of a map is the ratio of distances in the real world to distances on the map. In most cases, this ratio is given in the form of $1 : X$ where the *scale denominator* X is a real number – usually a multiple of 1000. Common map scales are $1 : 1,000$ (large scale), $1 : 10,000$, $1 : 20,000$ or $1 : 50,000$ (intermediate scale), $1 : 100,000$, and $1 : 1,000,000$ (small scale).

Resolution and accuracy are related to scale by the constraints of the physical medium of the printed map: An experienced cartographer can, for example, achieve a mapping accuracy of $0.1mm$, so it is impossible to get overall accuracies of more than $0.1mm \times X$ where X is the scale denominator of the map. For a scale of $1 : 50,000$, this means that accuracies better than $0.1mm \times 50,000 = 5m$ are unlikely to be achieved.

As far as resolution is concerned, the minimum distinguishable size for objects in a map varies depending on the geometric type (point, line or area) and the shape of the object. For a point, the minimum diameter may, for example, be $2mm$, so a point feature on a map with a scale of $1 : 50,000$ would cover a circle with a diameter of $2mm \times 50,000 = 20m$ in the real world.

Resolution and accuracy are important in the context of the generalization process because they are well-known concepts that can be used to express goals for the generalization process. In the context of hierarchical building models, the term resolution refers to the minimum size of a feature in the generalized model.

When it comes to accuracy, one has to distinguish between relative and absolute accuracy: If we have feature that is defined in relation to a higher-level feature like a window in a wall, we define the relative accuracy of a feature as the difference between the original and its generalized version in the coordinate system defined by the parent feature while the absolute accuracy is the difference between the original and the resulting feature in world coordinates.

In the case of the window in the wall, this distinction means that if the wall changed its position in the generalization process (for example, due to a displacement operation), then the relative accuracy would measure the change of the window on the wall while the absoute accuracy would also be affected by the displacement of the wall, so the absolute accuracy would be higher if the window remained at its original position in world coordinates which would, in this case, not even be on the wall but "dangling in the air".

For this reason, we will usually refer to the relative accuracy when accuracies are used as quality criteria. For some applications, it may, however, be important to preserve a given global accuracy. In these cases, the positions of all features after the generalization have to be compared to their originals in global coordinates. One way to ensure a given relative or absolute accuracy is to use the Hausdorff property introduced in section 2.1.1.

In this thesis, the concept of a (target) resolution ϱ is used to describe the level of geometric simplification that is intended by the user. This also includes the notion that a relative accuracy in the order of the given target resolution should be preserved.

One way of customizing the generalization process for a given application is to specify different target resolutions for different features or feature types according to spatial criteria: For a flooding scenario, one can, for example,

imagine that buildings up to a given distance from a river should be given at a resolution of $1m$ (but for their roofs, a resolution of $5m$ may be sufficient) while features on hills of sufficient height may not be needed at all ($\varrho = \infty$).

Special parameters, especially parameters concerning semantic aspects like penalties for aggregating buildings with different functions, are strongly application-specific and have to be handled separately: Given similar geometric constellations, for example, the question if it is worse to aggregate a given church with the adjacent supermarket or prison (or not aggregate it with any of them), can only be answered by the application.

This problem is not treated to its full depth in this thesis; one way to handle it without bothering the user with the necessity of trying to develop a deep understanding of each atomic generalization step is to predefine importance levels for different properties that could be transformed into sensible weights for normalized parameter values. The user would then only have to choose the importance levels for different properties.

2.1.1. Hausdorff distance

The Hausdorff distance (Hausdorff, 1914) defines the distance between two sets X and Y as the maximum of the shortest distance from any element of X to Y and the distance from any element of Y to X:

$$d_H(X,Y) = \max\{ \sup_{x \in X}(\inf_{y \in Y} d(x,y)), \sup_{y \in Y}(\inf_{x \in X} d(x,y)) \}$$

where sup means the supremum and inf means the infimum. Since all sets (objects) we will encounter in this context can be considered closed and bounded, the suprema and infima in the definition are true maxima and minima.

Abbildung 2.1.: The Hausdorff distance.

Figure 2.1 shows an example of two sets X and Y; both sets are closed areas in the plane, Y was depicted in a wireframe view in order to avoid occlusions. X and Y may, for example, be the original and a simplified footprint of a building. As the figure shows, the two parts of the Hausdorff distance – the maximum distance $d_{X \to Y} := \sup_{x \in X}(\inf_{y \in Y} d(x,y))$ from any element in X to Y and the maximum distance $d_{Y \to X} := \sup_{y \in Y}(\inf_{x \in X} d(x,y))$ from any element in $Y$to X – are usually not identical. Because the maximum operator is commutative, the Hausdorff distance itself is symmetric: $d_H(X,Y) = d_H(Y,X)$.

Another interpretation of this measure is that each element of X is within a buffer of $d_H(X,Y)$ around Y and each element of Y is within a buffer of $d_H(X,Y)$ around X which makes this measure a sensible candidate to define an important property for a generalization step: The Hausdorff distance between the original and the generalized model should not exceed the target resolution. This is a standard way to control the error produced by a simplification in computer graphics (Luebke et al., 2002).

Figure 2.1 shows, however, that the original definition of the Hausdorff distance may restrict the possibilities for generalization operators too far for our purposes. Imagine that X is some original object – for example, the footprint of a building – and Y is a generalized version of this object: If our target resolution is ϱ, then Y would not be a valid simplification of X although the protrusion of X that causes the high value for $d_H(X,Y)$ is narrow enough to be cut off at resolution ϱ. In an extreme case, an antenna with a height of $20m$ and a diameter of $5cm$ would have to be included in a generalized model with a target height resolution of $10m$ – a resolution level at which even the basic shape of most roofs would be considered irrelevant. This is obviously not what we intend.

For this reason, we use a slightly modified Hausdorff distance to measure the difference e (generalization error) between an original model X and a generalized version Y of X:

$$e(X, Y) = d_H(X, Y \cup (\mathbb{R}^3 \setminus X)).$$

This means that if we restrict $e(X, Y)$ to be less than ϱ, then the generalized version should be contained within a ϱ-buffer of the original: It may not produce new protrusions or isolated areas. The original X, on the other hand, does not need to be covered by the buffer around Y if it can be covered by an inward-facing (from $\mathbb{R}^3 \setminus X$) ϱ-buffer around itself.

Note that, according to this measure, ϱ is the *radius* of (parts of) objects that may be deleted at will in a selection process, not their diameter. A pure selection step is included in this definition by setting $Y = \emptyset$. Obviously, this measure is not symmetric.

This measure defines a sensible property of valid generalization operations with the purpose of controlling the generalization process: If, for example, a feature is enlarged, it should be ensured by the generalization process that it will not exceed the limit defined by the ϱ-buffer around the original feature. On the other hand, a feature X should not be removed if its size according to the modified Hausdorff measure is greater than ϱ, i. e. $e(X, \emptyset) > \varrho$.

Note that conflicts occurring after the generalization of a feature may, for example, trigger the displacement of another feature which could result in other conflicts and lead to different deadlock or livelock situations if the conflicts and resulting generalization operations form a kind of closed loop. To prevent such effects is the responsibility of the generalization process; the locality requirement for a given feature can be used for the detection and as an aid in the prevention of stalling situations. However, it does not, as such, make sure that such situations cannot occur.

Since computing the Hausdorff distance between two objects is computationally quite expensive, this requirement will usually be ensured and tested in practical applications by making sure that the axis-aligned *bounding box* of a given feature (either in global or in feature coordinates) is not extended by more than ϱ in each direction. This approach has the additional benefit of a native support of different resolutions for position and height if the coordinate system in which the bounding box is defined is aligned with the up direction.

In a hierarchical approach and in any context in which different generalization operations can be performed in sequence, it may happen that even if each individual operation preserves the Hausdorff property, the final result may "drift away" and have a Hausdorff distance of more than ϱ form the original. In this context, it is also important if the Hausdorff property is applied with respect to the relative or absolute accuracy: If, for example, a wall surface is displaced in the generalization process, then it makes little sense to apply the Hausdorff property to the absolute position of a window located on that wall because the window should be located within a ϱ-buffer of its original position on the wall than its absolute original position in space,

In the context of features with strongly different extents in different dimensions, the Hausdorff measure can also be misleading: A wall with a thickness of $30cm$ could, for example, be eliminated at a resolution ϱ of $15cm$ which is obviously not the intended result. For this reason, additional rules have to be introduced to preserve relevant features.

2.1.2. (Nested) Earth Mover's Distance

In order to assess the quality of a generalized model, the *nested Earth Mover's Distance* (nEMD) introduced by Kim et al. (2004), a special case of the general *Earth Mover's Distance* (EMD) (Rubner et al., 2000) measure, is an alternative to the Hausdorff metric. The Earth Mover's Distance measures the difference between two sets as an amount of "work" needed to transform one set into the other. An analogy to illustrate this measure is to define the difference between two piles of dirt by the amount of some kind of work necessary to transform the first one into the second by moving a minimumm amount of material.

Similar to the situation in the analogy, different work or cost functions can be imagined. In the analogy, one could, for example, measure the amount of energy or the cost (in money, using a given set of equipment) needed to move the required amount of earth. The two main factors are – in the analogy as well as in the difference measure – the "volume" or "mass" to be moved and the distance over which the material has to be transported.

The nested Earth Mover's Distance is a grah similarity measure that can be tuned to take structures within the model into account in the calculation of the difference between models. It operates on models represented by *attributed relationship graph*s (ARG) in which the features of which is composed are the nodes and relations

between features are modeled as edges. Both nodes and edges have attributes that can be used in the definition of the cost function for the nEMD.

The nEMD offers many degrees of freedom that make it a promising candidate for application-specific similarity measures in the assessment of generalization operators for hierarchical models:

- The construction of the ARG from the feature hierarchy can be defined according to the application, especially the relationships between the features and parameterization of the features and the relations.
- Depending on the application, different strategies can be applied in order to guide the process of determining the correspondences between the nodes in the original and the generalized model.
- Custom cost functions can be applied to determine the differences between the corresponding nodes in the original and the generalized models depending on the parameterizations of the nodes: The cost is determined in parameter space instead of being limited to geometry.

While (especially the general) Hausdorff distance measures the extreme differences between two models, the (n)EMD aims at measuring the accumulated impact of the transformation on the original shape. By assigning higher weights to different parameters, changes in these parameters lead to higher costs for changing these parameters ("moving heavier stuff") in the generalization process, so generalization options in which these parameters are changed significantly are less likely to occur.

In order to achieve this, the (n)EMD offers several degrees of freedom that can help to adjust it to specific feature types and applications. In the context of this thesis, the (n)EMD will not be used directly, but some of the custom difference measures like the (pseudo-) volume change in chapter 7 can be expressed in the form of nEMD terms. One of the most demanding tasks in the development of application-specific generalization processes is the balancing of cost or objective function terms for the evaluation generalization alternatives, for the earth mover's distance as well as in the approches presented in this thesis.

2.2. Cartographic Generalization

The most laborious and expensive part of mapping is data acquisition or surveying. The physical production of tangible maps is also an expensive task – especially the preparation of the actual printing process consisting of involved drawing and the production of the original etchings.

Since many applications of 2D maps share common topics of interest like topography, traffic, water bodies, and settlements, map content tends to be reused for different purposes. The most straightforward way of reusing map content is to use the same map for different applications. With the establishment of national surveying authorities, different map formats (especially topographic and cadastral), were standardized in many countries.

In order to save the effort of having to launch different surveys to acquire the data for maps with similar topics at different scales, smaller scale maps are usually derived from larger scale maps – especially if the producer of both maps is the same; for example, a mapping agency that publishes topographic maps at different scales.

Historically, this classical map generalization process was performed manually by cartographers who drew the shapes of the features in the generalized map on a see-through paper draped over the original map.

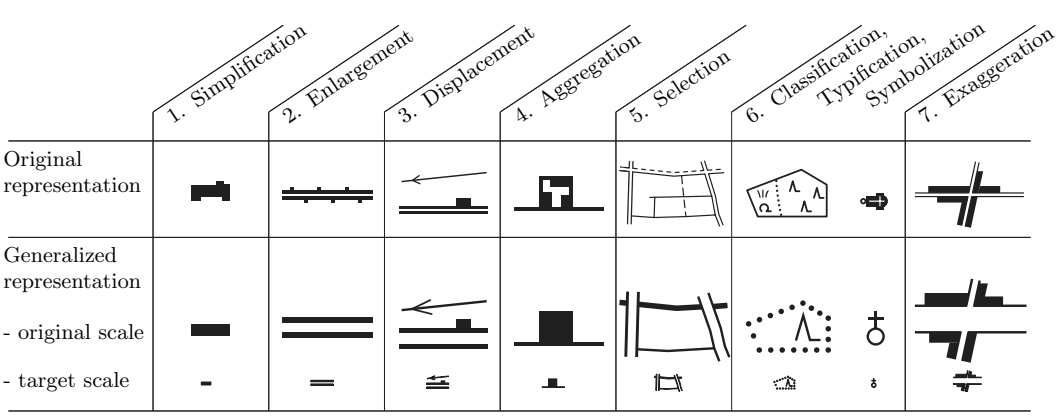


Abbildung 2.2.: Visualization of generalization operators (from Haunert (2009) after Hake et al. (1996)).

Figure 2.2 illustrates the most frequently used generalization operators for cartographic applications:

- **Simplification** – An object is represented by a simplified version of itself.
- **Enlargement, Exaggeration** – In many cases, objects are too small to be recognizable at the target scale andhaveto be enlarged; objects of higher importancemay be exaggerated beyond the necessity of being visible in order to make them more prominent.
- **Displacement** – In order to avoid cluttering the resulting map, objects may have to be shifted away from their original position.
- **Aggregation** – Objects of the same class are merged into one bigger object of the same class.
- **Selection** of relevant or elimination of less relevant features to avoid cluttering the resulting map.
- **Classification, Symbolization** – Objects of more special classes are treated as objects ofmore general classes andmay be represented by a symbol for this more general class.
- **Typification** – n objects of similar classes are represented by $k < n$ objects of the same or a more general class.

Most approaches in the context of 3D buiding model generalization address the transformation of special classes of features between different scales. In most cases, these approaches can be traced back to an adapted application of one or more of the classical generalization operators.

2.3. Building Generalization in 2D and in 3D

In 2D, a building is essentially represented by its footprint with some semantic information attached in the form of a parameter vector that will usually contain values for the function of the building and (optionally) some additional hints as to the shape of the building like its height and the number of floors.

In 3D, a lot of additional entities have to be captured and modeled: A building may consist of different wings with wall and roof structures, there are windows, doors, balconies, intrusions, protrusions, additions, and a vast amount of additional possible features even in quite regularly shaped buildings. For this reason, the generalization of buildings is inherently considerably more complex in 3D than in 2D.

Because of gravity, the third (vertical) dimension is inherently quite different from to the two horizontal dimensions: While the direction of the other two unit vectors is more or less arbitrary, the vertical dimension is more or less unique for local coordinate systems – although there are, of course, some local distortions of the earth's gravity field (that will, however, have no measurable effect in the range of accuracy that we are dealing with in this context).

Since gravity defines the statics of a building, it is one of the most relevant factors in the structure of buildings and therefore also in the structure of building models – severely limiting the feasible size of a building in the vertical direction: While settlement structures may have extents of many kilometers in the horizontal dimensions, their vertical extent will in most cases be below $20m$ with high rise buildings in the downtown areas of some larger cities in order of $100m$ up to $300m$; the currently tallest building in the world, the Burj Khalifa in Dubai, has a height of "just" about $830m$ which is still an order of magnitude smaller than the diameter of an average city.

This means that if we move from larger to smaller scales, the third dimension will decrease in importance much faster than the others: If we assume a resolution of $5m$ (meaning that differences of less than $5m$ can no longer be perceived), then the roof shapes of most buildings will be irrelevant because they could not be distinguished from simple flat-roofed placeholders, which, in turn, are rather 2D entities because they simply consist of a footprint polygon and a uniform height value.

Assuming that the minimum size to represent a roof shape in a visualization scenario would be at least $5mm$, the cartographic scale corresponding to the resolution of $5m$ would be about $1 : 1,000$ which means that the third dimension becomes geometrically more or less irrelevant at scales that are well inside the range that would be referred to as large scale in cartography.

For this reason, the terms *large*, *intermediate* and *small* scale will, in this context, refer to different ranges compared to cartography. Along the lines of the CityGML level-of-detail (LoD) definitions (see section 3.3), a rough classification for buildings can be derived:

- Large scale (LoDs 3 and 4: detailed facade structures): $0.5m$ of resolution or better,
- Intermediate scale (LoD 2: 3D structures like roof shapes preserved): $0.5m$–$5m$ of resolution,
- Small scale (LoD 0 and 1: 3D information more or less reduced to height, 2.5D flat-roofed representatives for buildings): worse than $5m$ of resolution.

The explosion of complexity from 2D to 3D is, in part, caused by these different inherent scales: The majority of the multitude of different features classes (like balconies, windows, etc.) appearing in 3D models are simply too small to be represented in a sensible way at the usual mapping scales of 1:1000 and smaller.

Beyond the issues of scale and detail, an additional problem is occlusion: A 2D (or 2.5D) scene can be represented completely in a single *static* view (map), while a true 3D scene can never be visualized completely in a single static view due to occlusions. Although this is a trivial observation, it is important because it means that there is no canonical representation tool like the map for 3D data.

Over the centuries, best practices and standards have been established for the representation of 2D objects in well-defined multi-purpose map formats like topographic maps. Such a standard defines selection (importance of different objects) and representation rules like minimum sizes of objects (affected by the map scale and mapping accuracy) for a given map format. Due to the manageable number of object classes and the canonical perspective, such a standard format proves sufficient for most applications using 2D data, especially if there is the option of introducing thematic layers.

Note, however, that these standards usually only define *constraints* that a model at a given scale has to satisfy like minimum sizes for objects of a given class. These standard sizes are often the result of graphical constraints defined by the medium of the map printed on paper: At usual viewing distances ($< 1m$), it would be, for example, be difficult for the human eye to perceive lines of a width of less than 0.05 millimeters.

2.4. Generalization and structure recognition

The terms *structure recognition* and *feature extraction* appear in different parts of this thesis referring more or less to the same problem. In a single layer of abstraction, a distinction between the two concepts may be that structure recognition refers to the identification of special patterns (like distributions in a line or regular grid) of known features while feature extraction refers to the identification of special semantic entities (like windows or protrusions) in semantically less structured geometric data like polygon meshes.

Since in a hierarchical modeling framework special distributions of features can be interpreted as special semantic features on a higher level of the modeling hierarchy and the identification of a special structure that is associated with a semantic entity may also be referred to as a structure recognition process, both terms may appear in this thesis for the general problem.

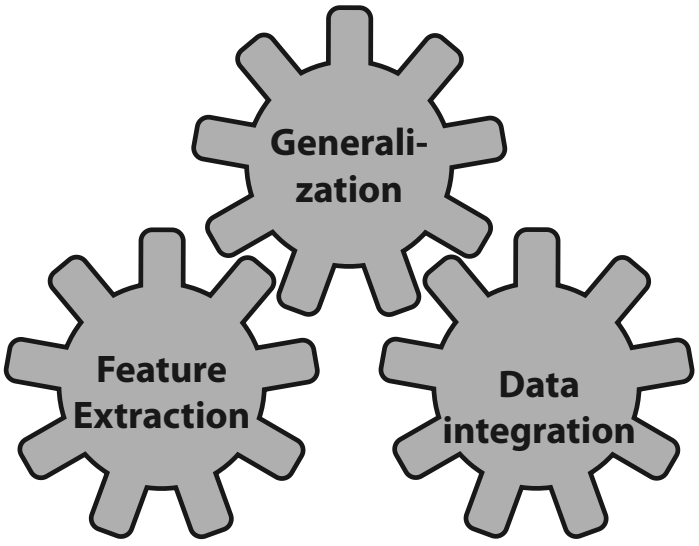


Abbildung 2.3.: Generalization, structure recognition and data integration are closely interdependent.

One crucial problem in the processing of 3D city models is the strong interdependence of generalization, structure recognition and data integration, especially if application-specific data is needed.

Imagine, for example, a scenario in which noise level data has to be processed. Important data for such an application will usually not be part of a general-purpose 3D city model format, e.g. traffic volume on the roads, noise reflectance properties of different building materials, and detailed information about the geometry and materials of crucial features like bridges and noise barriers.

This kind of application data will usually be kept in special databases. In the spirit of generalization as the extraction of relevant information, just these pieces of information are, however, of fundamental importance for the generalization process. For this reason, a data integration or conflation step is necessary to be able to support generalization processes with an integrated view of the structure of the model provided in a basic general-purpose framework and the application data possibly available only from different sources.

In the context of generalization, structure recognition in the narrower sense of identifying special patterns of features is of special importance because information about the presence of these patterns is necessary for the applicability of standard generalization operators like typification.

If specialized generalization procedures are available for special features that are not modeled explicitly in the data, then it can make sense to try to identify them in the data; it may, for example, be possible to identify (patterns of) windows and balconies in raw polygon data in order to be able to apply specific generalization operators for these features.

Especially in a hierarchical model, the entanglement of generalization and structure recognition can make alternating and interdependent generalization and structure recognition steps necessary: At a given resolution, it may be possible to simplify a group of features that was initially quite different in such a way that they are homogeneous after the simplification. Employing typification, this group of homogeneous features could then be replaced by a lower number of similar features.

If the structures are not perfectly homogeneous in the first place and the generalization task is treated as an optimization problem (as outlined in section 2.5), the entanglement becomes even closer because the quality of the final result depends on the combination of the decisions taken in the original simplification, in the homogenization process, and in the final aggregation or typification in our example: The optimality of the result can only be ensured if all steps were executed within the same optimization procedure.

2.5. Optimization and Computational Complexity

2.5.1. Theory of Computational Complexity: the class NP

Decision problems are problems to which the answer is either "yes" or "no", depending on the values of a potentially infinite number of input variables. The complexity class NP (Cook, 1971) is defined as the class of decision problems that can be solved by a **N**ondeterminiistc Turing machine in **P**olynomial time (with respect to the length of the input), so the popular interpretation that NP stands for "non-polynomial" is wrong.

In fact, the question if all problems in NP can be solved by a *deterministic* Turing machine (roughly corresponding to a traditional computer) in polynomial time is one of the most famous open questions in theoretical computer science (the "P vs. NP" problem) and one of the seven Millennium Problems (Jaffe, 2005) for which the Clay Mathematics Institute offers $1,000,000 to the first person to solve it. While the most common guess is that P≠NP (meaning that not all problems in NP can be solved in polynomial time by a deterministic Turing machine), this has not been proved yet.

An alternative definition of the class NP is that it is the class of *polynomially verifiable* decision problems: Given an assignment of values to the input variables, a deterministic Turing machine can verify in polynomial time if the answer to the original decision problem is indeed "yes" for this assignment. The original question if there is an assignment for which the answer is "yes" may, however, be hard and require a super-polynomial number of steps on a deterministic Turing machine (if $P \neq NP$).

Cook (1971) showed that the Satisfiability (Sat) problem that asks if a given Boolean formula is satisfiable is *NP-hard*. This means that every problem A in NP can be *reduced to* Sat: If an efficient (polynomial time) deterministic solution for Sat exists, then this algorithm can be used to solve any other problem A in NP in deterministic polynomial time. Such a reducibility can be proved by giving a polynomial-time algorithm that

transforms an instance a of A to an instance a' of SAT in such a way that the answer to a' is "yes" if and only if the answer to a is "yes".

If a problem is NP-hard and in NP, then it is called *NP-complete*. Many NP-hard problems are not NP-complete because they are not decision problems; a prominent case are optimization problems that ask for the best solution for a given problem and not for a "yes" or "no" decision. The corresponding decision problem that can be defined for an optimization problem A asks if there is a solution for A with a cost lower than a constant k.

In the original proof, Cook had to show that every single problem in NP can be reduced to SAT. Fortunately, the polynomial-time reducibility relation is transitive: If A can be reduced to B and B can be reduced to C, then A can also be reduced to C. This is true because a polynomial runtime for the reductions is required: If the reduction r_1 from A to B and the reduction r_2 from B to C can be achieved in polynomial time, then the sum of the runtimes of the polynomials r_1 and r_2 is, of course, also a polynomial. For this reason, Cook's proof is a most helpful device if we want to prove that a given problem is NP-hard: Because we know that SAT is NP-hard, we only have to show that SAT can be reduced to our given problem.

Taking this idea further, we only have to reduce any problem known to be NP-hard to our problem to show that it is NP-hard. Using this technique, a large number of problems have been shown to be NP-complete, often using reductions to problems that were proved to be NP-hard before. Almost all of these paths of reductions lead back to Cook's original NP-hardness proof for SAT.

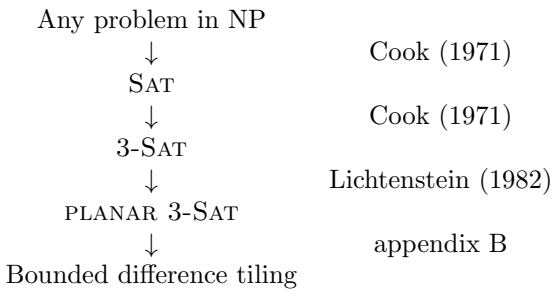


Abbildung 2.4.: NP-hardness proof for the bounded difference tiling problem underlying the facade homogenization problem (section 8): Chain of reductions (Appendix B).

Figure 2.4 illustrates such a reduction chain for the problem of facade homogenization introduced in section 8 (the proof itself is given in Appendix B): Cook (1971) showed that every problem in NP can be reduced to SAT and that each SAT problem can be reduced to 3-SAT, a specialized version of SAT with only 3 literals per clause. Lichtenstein (1982) showed later that even the planar version of 3-SAT is NP-hard by reducing 3-SAT to planar 3-SAT. In Appendix B, we will see that planar 3-SAT can be reduced to the facade homogenization problem, so we know that any problem in NP can be reduced to the facade homogenization problem by applying the four transformations in Figure 2.4.

Having shown that a problem is NP-hard means that it is unlikely that we will find a polynomial-time algorithm that solves it. For this reason, it makes sense to try to divide the problem into as small independent instances as possible and to develop heuristic approaches to find good solutions (instead of the global optimum) if we are faced with an optimization problem.

Because many NP-hard problems are of high economic interest (for example, the well-known Traveling Salesman Problem for the transportation industry), a lot of research effort has been spent to develop strategies for solving as many and as large instances of NP-hard problems as possible. In the context of optimization problems, Mixed Integer Programming (MIP, see section 2.5.3) is an important tool: It offers a model to represent optimization problems in the form of a set of constraints and an objective function and there is high-performance software to solve problems given in this form, so if we can express our problem in the form of a MIP instance, we can profit from the effort of the highly experienced experts who contributed to the development of the software.

One drawback of modeling with MIP is the fact that most solvers can only handle linear and (to a limited degree) quadratic terms in the constraints and the objective function which means that some constraints may need some thought in order to come up with a linear MIP representation and some may be impossible to express in terms of a (linear) MIP model at all. Even though the restriction to linear and quadratic terms may seem very strong, we will see in the course of this thesis that it allows us to model many aspects of the generalization process.

2.5.2. Optimization problems

As we have seen, generalization is inherently an optimization problem: We want to extract a new model that *best* preserves the relevant content of the original one while *minimizing* the complexity of the resulting model. This means that, in most cases, we are faced with the two conflicting goals of preserving as much of the original structure as possible and to simplify the model as far as possible.

In order to deal with these conflicts in a structured way, we can define quality measures (*constraints*) that express desirable or bad properties of a resulting model. We distinguish between hard constraints that have to be satisfied in any case and soft constraints that incur a penalty depending on the degree to which they are violated. The goal of the optimization process is to minimize a cost function derived from the penalties for the violation of the soft constraints while satisfying all hard constraints.

In the context of optimization theory, the soft constraints are usually put into a single cost function that is supposed to be minimized – this is equivalent to maximizing a reward function; the two concepts can easily be converted into each other by multiplying the function with -1 (inverting the sign). In the literature, the minimization version is the standard representation. Most solvers offer both versions for their input and internally convert it to the minimization form if necessary.

While it is not always possible to merge all soft constraints and penalties into a single closed function, this will be the case in all optimization problems we will encounter in the context of this thesis, so we will use the definitions of the optimization community and distinguish between a set of (hard) constraints and an optimization function in which the penalties incurred by the soft constraints are collected: The term *constraint* will only be used for the hard constraints in the following.

Formally, the problem is given by

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0,\ i \in \{0,\dots,k\} \\ & [h_i(x) = 0,\ i \in \{0,\dots,l\}] \end{aligned}$$

where x is a set of variables, $f(x)$ is the objective function to be minimized, and the $g_i(x)$ are a set of (hard) inequality constraints. A constraint of the form $g_i(x) \geq 0$ can easily be transformed to this form by negating the inequality: $g_i'(x) := -g_i \leq 0$. The equality constraints $h_i(x) = 0$ can be expressed in the form of two inequalities: $h_i(x) \leq 0$ and $-h_i(x) \leq 0$. For this reason, they are set in brackets here because it is not necessary to include them in a canonical representation of the problem.

Since we know that it is easy to transform those forms into the canonical representation and software for solving optimization problems offers the option to use them in problem descriptions (converting the problem to the canonical form internally if necessary), we will use "=", "≤", and "≥" constraints throughout this thesis in the way that best supports the explanations given in the text without explicitly converting them to the canonical form.

Note that auxiliary variables can be defined and forced to assume the intended values in (hard) constraints in order to be used in the objective (cost or reward) function. Even if the constraints and the objective function may only consist of linear terms in the variables of the optimization problem, concepts like piecewise linear penalties (approximating, for example, more complex functions) or case distinctions and logical operations on Boolean variables can be expressed.

According to the structure of the constraint and objective functions, there are different classes of optimization problems defined by restrictions on the constraints and the objective functions. While most of those problems are (NP-)hard, there is still a big difference between those problem classes: While most instances of some classes of problems are solvable (for moderate instances) by up-to-date software, there are only theoretical approaches for the solution of others that have not yet found their way into standard software and/or are too specialized to have been streamlined for performance far enough to yield high-quality solutions for most moderate instances in reasonable time.

An important basic distinction concerns the nature of the variables in the constraints and the objective function: If there are discrete (integer) variables in the problem, it becomes much harder than in the all-continuous case:

If, for example, all variables in a linear (all constraints and the objective function are linear in terms of the variables of the problem) optimization problem are continuous, the problem can be solved in polynomial time while the problem becomes NP-hard if (some of) the variables may be be discrete – the special case in which all variables are Boolean (assuming only values 0 or 1) is one of the 21 problems that Karp (1972) proved to be NP-hard.

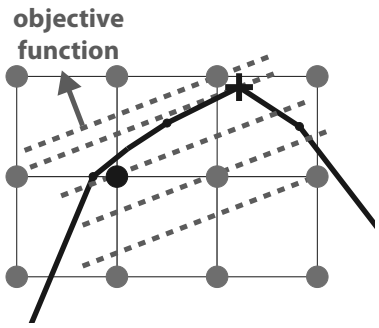


Abbildung 2.5.: Continuous and Integer optimization: A problematic case.

The reason for this effect is that while we can determine the best continuous solution in polynomial time, this will usually not be a feasible solution in which the variables must have integer values. Figure 2.5 illustrates that this can indeed be a serious obstacle to finding the best integer solution: In the drawings, the dotted parallel lines are iso-value lines of the objective function; the arrow points in the direction of the best objective values. The bold black lines illustrate the polygon defined by the constraints. Neither of the neighboring integer solutions (dots on the grid) around the best continuous solution (black cross) is the best feasible integer solution with respect to the objective function, so the intuitive approach of testing all neighbors of the best continuous solution would not yield the right solution in this case even though two of them are feasible – a feasible neighbor of the best solution is not automatically the best feasible solution.

Note that the dimension of this drawing is the number of discrete variables in the problem, so if we had 3 instead of two variables, the feasible region would be a convex polyhedron in 3D, and there would be 8 instead of 4 neighbors of the best solution: The number of neighbors is 2^n where n is the number of dimensions, so an exhaustive search of the neighbors will lead to exponential runtime complexity without providing a guarantee that the optimum will be found.

If some of the variables in an optimization problem are integer and some are continuous, the problem is called a *mixed integer program* where the term "program" dates back to the time of G. Dantzig's (Dantzig, 1951) original paper on the subject and refers to (military) planning or scheduling rather than programming a computer.

Another important distinction between optimization problems concerns the form of the constraints and the objective function: If the constraints and the objective function consist only of linear expressions in the variables of the optimization problem, we have *linear* optimization problem. If there are quadratic terms in the objective function and the constraints are all linear, the problem is referred to as a *quadratic* optimization problem. If there are quadratic terms in the constraints, the optimization problem is called *quadratically constrained* and considerably harder to solve – quadratically constrained problems are NP-complete even if all variables are continuous.

If we are concerned with mixed integer problems, most solvers can handle linear programs up to considerable sizes; some (e.g. the CPLEX software) are able to handle intermediate quadratic and some quadratically constrained problems. There are still comparatively small-sized linear problems left that because of their structure and NP-hardness of the general problem cannot be solved even by advanced solvers.

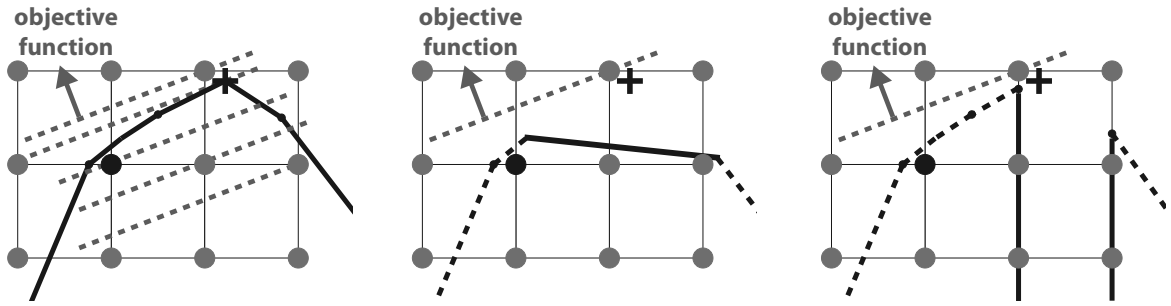
2.5.3. Mixed Integer Programming (MIP) Software

As we have seen in the last section, the term *Mixed Integer Programming* refers to all optimization problems in which non-continuous variables occur. Unfortunately, most available software products for solving such optimization problems are limited to linear and – to a limited extent – quadratic or quadratically constrained problems.

Two of the most prominent software systems for solving MIP problems are the free lp_solve (Berkelaar et al., 2004) software and the commercial solver CPLEX (IBM ILOG, 2011) that is free only for academic use. While

lp_solve handles only linear MIP problems, CPLEX can solve quadratic or quadratically constrained problems as well. For this reason, CPLEX was used in the experiments for this thesis because a least squares adjustment in the computation of intermediate values can only be achieved using quadratic optimization problems.

Most MIP solving software is based on some basic strategies for solving linear MIP problems – the most prominent ones are based on generating cutting planes and the Branch-and-Bound principle or on the Branch-and-Cut approach, a combination of cutting planes and Branch-and-Bound these approaches will be introduced briefly in this section. There are special versions of these approaches for the solution of nonlinear MIP problems, especially for solving quadratic problems. More complex nonlinear MIP problems require more sophisticated approaches with (currently) much lower probabilities of finding an optimum solution with a reasonable amount of resources.



(a) Original situation: Relaxed MIP solution. (b) A cutting plane. (c) A Branch-and-Bound step.

Abbildung 2.6.: Cutting plane and Branch-and-Bound approaches to solving linear MIPs.

Cutting planes were first used in Dantzig et al. (1954) to solve instances of the Traveling Salesman problem. Later, Gomory (1958) introduced a more general approach to solve arbitrary MIP problems. The basic idea (illustrated in figure 2.6(b) for the situation described in section 2.5.2) of the cutting plane approach is to solve the relaxed problem (without the integrality constraint) and, if this solution is not integer, to generate a new constraint (bold line) that separates the non-integer optimum solution (black cross) from the feasible region without rendering any feasible integer solution infeasible. Note that if each cut separates a non-empty part from the interior (not just the boundary) of the feasible region, then a series of cuts will eventually find the best integer solution.

Most of the cutting schemes (including Gomory's) suffer, however, from the problem that it is often necessary to generate a huge number of cuts to find the optimum integer solution and that they suffer from numeric instability

Branch-and-Bound is the second general-purpose approach to solve linear MIP problems. The basic idea was first described by Land and Doig (1960) in the context of operations research. Dakin (1965) refined the approach and made it more efficient by introducing a new way of storing the tree representing the branching history.

As in the cutting plane approach, the first step is to solve the relaxed problem. In the *Branch* step, one of the variables x_i for which the relaxed solution is not integer is chosen as the pivot variable and the original problem is split into two sub-problems: In the first one, a constraint of the form $x_i \leq \lfloor X \rfloor$ is added; in the second one, the constraint $x_i \geq \lceil X \rceil$ is added, where X is the non-integer value of x_i for the non-integer optimum solution. The algorithm is then recursively applied to the two resulting problems, forming a binary search tree.

Once a feasible integer solution has been found (that is not yet known to be the best one), we can compute the corresponding objective value V and use in the *Bound* step: Whenever a new problem is created in a branch step and its relaxation has been solved, we can compare the objective value v of the relaxed solution to the value V. We know that no integer solution in the current branch can be better than the value v of the relaxed solution, so if v is worse than V, we can discard the whole branch at once. Especially if we can discard early branches in the search tree, this can save masses of computations. Once we find an integer solution with an objective value V' that is better than our global value V, we can set $V = V'$ for all succeeding bound steps.

The two basic strategies for evaluating such a search tree are depth-first and breadth-first search: In depth-first search, the first problem is evaluated at once in each branch step; only if all subproblems in the first branch have been solved, the second problem in the current branch will be considered. This strategy is fast in finding integer solutions, but they may be of poor quality. In breadth-first search, the problems are solved as they evolve, so

all nodes on each level of the branch tree are expanded before the algorithm progresses to the next level. This approach may need a long time until it produces an integer solution but once it fins one, it is usually of a high quality.

In most up-to-date solvers, the two approaches are combined and enhanced by heuristics in the choice which node should be expanded next: One idea may be to start with a depth-first search in which the node with the best objective value v of the relaxed solution is expanded first. Once an integer solution is found this way, this can be used in the bound step in a breadth-first search. If parallel threads are available, another approach could be to have some threads perform depth-first searches, supplying parallel breadth-first threads with (hopefully tight) bounds.

Nemhauser and Wolsey (1988) showed that an intelligent combination of cutting planes and the Branch-and-Bound principle can enhance the performance of MIP solvers considerably. In the same paper, they show how to overcome some of the numerical problems in cutting plane approaches. This Branch-and-Cut approach is one of the core elements in most current solvers.

While most solvers are based on these three basic strategies, they are refined and augmented by heuristic elements for special problems like network flow problems. High-performance solvers like CPLEX analyze the problem at hand searching for such special structures and automatically choose an applicable heuristic that can help the program to find and verify optimum solutions much faster than a less flexible standard application of one of the basic strategies.

In the course of this thesis, for example, the runtime for the same building aggregation instance was reduced from about one hour to less than one minute with the change of version from CPLEX 12.0 to CPLEX 12.2, probably because a fitting heuristic was implemented between the two versions. Because of the high econmic interest and the complexity of the Mixed Integer Programming, details about the inner structure of commercial solvers (concerning parameters, heuristics, branching decisions, etc.) are well-kept secrets of the companies that developed the solvers.

The process of finding the optimum solution can be accelerated considerably by supplying the solver with an initial feasible solution, especially if it is as close as possible to the original: The objective value for the initial solution can immediately be used in the *Bound* step, and many branches of the search tree may be cut at very early stages. This is an interesting opportunity to combine problem-specific heuristic and optimizing approaches: A solution obtained by a fast heuristic approach may be used to speed up the MIP solving process used to determine an optimum solution.

2.5.4. Modeling with MIP

Mixed Integer Programming is used to model a wide range of optimization problems, especially in areas where optimum or at least high-quality solutions to difficult (usually, NP-hard) optimization problems are of high economic interest – for example, in the context of route planning in the transportation industry or portfolio optimization in finance.

In this context, MIP is an interface between the user and the theoretical computer scientists and specialized mathematicians who develop sophisticated algorithms to solve as complex instances as possible of those NP-hard problems using a reasonable amount of resources. Even though it may require some analytic effort and creativity to express the problem at hand in the form of a MIP problem, this is often considerably less difficult than trying to implement an application-specific solution scheme which without a deep understanding of the field of optimization will almost certainly be outperformed by a high-quality MIP solver operating on a MIP representation of a reasonable quality.

The performance of a MIP solver for a given problem can suffer considerably due to a problematic MIP representation. With some experience or a new approach, it is, however, possible to avoid the most critical pitfalls and come up with a working solution.

One of the most performance-critical issues is ambiguity: If there are many solutions to the MIP problem that essentially encode the same solution to the original problem in the application, the solver may have to evaluate all of them to prove the optimality of a solution. Especially if there are different independent ambiguities in different parts of the model, the number of solutions that have to be evaluated completely in the worst case is the product of all the independent possibilities.

Sometimes it is difficult to avoid the ambiguities completely. In such a case, it is sometimes more sensible to develop a new approach, especially if this approach is likely to profit from a built-in specialized heuristic

approach provided by the solver. The facade homogenization problem described in section 8 is an example for such a case: The original direct modeling of the problem (section 8.1) showed low performance because of hard-to-resolve ambiguities and the lack of fast heuristics for the structure of the problem. The graph-based approach in section 8.2 describes the problem less directly but most ambiguities can be weeded out quite easily using this approach and it profits from CPLEX's optimized algorithms for handling flow problems.

In order to illustrate the procedure of modeling with (linear) MIPs and some of the special aspects and peculiarities of this approach, we will derive a way to express Boolean operations in the form of linear MIPs. We will use a simple positive representation for Boolean variables: a value of 1 represents the TRUE and 0 represents the FALSE state of a Boolean variable). The mapping of the Boolean expressions to linear programs is similar to the one presented, for example, by FICO Decision Management Community Forum (2009).

The negation

$$X \equiv \neg a$$

can easily be expressed by

$$X = (1 - a) :$$

If a is TRUE (=1), then $X = 1 - 1 = 0$ =FALSE; if a is FALSE, then $X = 1 - 0 = 1$ =TRUE.

Next, we consider the conjunction

$$X \equiv \bigwedge_{a \in A} a$$

of the boolean variables a in a set A. Since we want the result to be a linear program, we can only use $\leq$ and $\geq$ relations of sums (or differences) of variables with constant factors. We do not know how many of the variables in A are going to be TRUE, so we have to split the two implications in the $\equiv$ relation by setting a lower and an upper bound for X in such a way that X is TRUE if and only if all variables in A are TRUE.

We can derive the lower bound by a simple consideration: We want X to be forced to be TRUE (equal 1) if all variables a in A are TRUE. In this case, we have $|A|$ TRUE variables on the right side of our equivalence relation. If we simply sum up the variables a in A, we get $|A|$ if all a are TRUE and any value between 0 and $|A| - 1$ if any of the variables is FALSE, so if we subtract $|A| - 1$ from this sum, we get:

$$X \geq \sum_{a \in A} (a) - (|A| - 1).$$

This ensures that X has to be TRUE if all a are TRUE because in this case, the right hand side of the first constraint is $|A| - (|A| - 1) = 1$. In all other cases, the right side of the first constraint is less than or equal to zero, so the constraint is relaxed.

In order to capture the equivalence relation, we have to set an upper bound for X. Otherwise, we could simply fix X to TRUE and our constraint would always be satisfied. A simple first idea is to introduce constraints that ensure that X cannot be TRUE if any of the variables a in A is not set:

$$\forall a \in A : X \leq a.$$

These constraints ensure that X cannot be 1 if any of the variables a is 0. This would, however, produce $|A|$ constraints which can harm the efficiency of the solver. For this reason, it makes sense to step back and take a slightly less direct look at the problem in order to be able to put the idea into a single constraint.

We want to make sure that X can only be TRUE (equal 1) if all $|A|$ variables in A are TRUE. One way to look at this is to imagine a pair of scales: The left side containing the X variable may only be allowed to rise to the TRUE level if the $|A|$ variables on the right side are also TRUE. For this reason, we scale the X (with natural values of 0 or 1) on the left side by a "weight" of $|A|$:

$$|A|X \leq \sum_{a \in A} (a) \,.$$

If X is TRUE, then the left side is $|A|$. This is only feasible if the right side is $\geq |A|$ which, in turn, is only the case if all variables in A are TRUE.

This example shows some of the basic issues of modeling with MIP:

- In many cases, equalities have to be broken down into two different (sets of) constraints for both directions.
- There are many different ways to transform even a quite basic single constraint into a MIP representation. When it comes to modeling a more complex task like, for example, the generalization scenarios described in sections 7 and 8, there are lots of feasible approaches – though not all of them will be efficient.
- It is often necessary to dismiss a first approach and take new line of thought in order to resolve performance issues.
- Even comparatively easy tasks may need some amount of experience and/or creativity to come up with a high-performance solution. The author does not give any guarantee that the approaches introduced in this section (or anywhere in this thesis) are the best imaginable ways to express a given problem.
- There is often an element of trial-and-error.

For the disjunction

$$X \equiv \bigvee_{a \in A} a$$

we can use the same ideas. For the $\geq$ direction, we use the scales analogy that we used for the $\leq$ direction in the case of the conjunction:

$$|A|X \geq \sum_{a \in A} (a) \,.$$

This means that if any of the variables a in A is TRUE, then the right side is greater than 0 and X has to be TRUE; if all a are FALSE, then the right side is 0 and X may be TRUE or FALSE. Because we have scaled the X on the left side with the factor $|A|$, all variables a in A can be TRUE without rendering the constraint infeasible.

This leads to some more observations concerning the modeling with MIP:

- Reuse successful patterns / ideas / approaches.
- If you want a binary variable to be set to TRUE if a right-hand expression is > 0, scale it with a factor M that is bigger than (or equal to) the maximum possible value of the right side.
- Such so-called "big-M" values appear in several approaches and general tricks, for example, in several places in Haunert (2009). They should be set as tight as possible because otherwise, they may tip the direction of the choices of the solver in the wrong direction and thus make it explore less promising branches of the decision tree before finding the optimum. In the case of the disjunction, setting M is easy because we know that the right side will always be smaller than $|A|$ and that this value can indeed be assumed, so $M = |A|$ is safe and the best choice concerning the optimization process.

For the $\leq$ direction, we invert the "sum-up-the-right-side-and-offset" technique we used for the $\geq$ case in the case of the conjunction:

$$X \leq |A| - \sum_{a \in A} (1 - a) \,.$$

The term $1 - a$ in the sum is 1 if a is the negation of a shown above, so the sum counts the number of FALSE values a in $|A|$. If all a in A are FALSE, then the sum is $|A|$, so the right side is $|A| - \sum_{a \in A} (1 - a) = |A| - |A| = 0$, so X has to be FALSE. In all other cases, the sum is smaller than $|A|$, so the right side is ≥ 1 and X can be TRUE.

Boolean operations were introduced here to give the reader an impression of how some constraints that are not directly linear equations in their nature can be expressed in the language of Mixed Integer (Linear) Programming. In the context of this thesis, several constraints will appear that after having been established have to be translated to become linear expressions. If this happens, there will be an explanation of why the linear forms given in the text have the desired effect.

The rationale behind using MIP to solve optimization problems is that it allows us to transform the problem of finding the optimum solution into the descriptive problem of specifying the properties that a valid solution must have. Sometimes, it is difficult to derive a sensible MIP representation, i. e. one that reflects the original problem well and can be handled by the solver of choice with an acceptable need of computational ressources, for a given application. It is, however, usually easier to do this for someone who is not a dedicated expert in optimization than to implement an own strategy for finding an optimal solution (and proving that it is optimal).

For this reason, MIP is a valuable tool in situations where finding exactly optimal solutions (at the cost of increased comutational cost) is required or desirable. Examples for such situations are the production of fixed LoD representations in a model that are processed once and distributed to many clients, or as a refrence in the evaluation of heuristic approaches. Another application is the evaluation of the objective functions themselves in user surveys: If we present users with results that are provably the best ones for a given objective function, and they reject the results, we can be confident that the objective function was not suitable.

An alternative to MIP is Constraint Logic Programming (CLP), an extension of Logic Programming. Especially in contexts in which many constraints consisting of Boolean operations appear in the model, CLP is an interesting alternative to MIP. In the cases investigated in this thesis, Boolean operations appear in a comparatively small number of constraints (they were only used as a simple example for the introduction in this section); for this reason, CLP-based implementations of the problems were not tested. It remains, however, an interesting topic for further research to investigate the potential of CLP – especially for the template-based facade homogenization approach where many Boolean constraints are involved.

3. Building and City Models

The terms *building* and *city* model appear in different places throughout this thesis. The difference between the two concepts is obvious: While a building model represents only buildings, a city model encompasses any entities that may appear in a city.

In the CityGML city model, for example, there are several different classes of city objects: buildings, transportation, terrain, water bodies and more. In fact, since almost all kinds of objects can appear in cities, a city model comes close to a world model as far as semantic depth is concerned.

On the other hand, dedicated building information models may contain more detailed information on a building than a city model would usually allow to specify. The question up to which semantic depth buildings are represented within a city model depends on the available data, the storage capacity and computing power of the underlying hardware infrastructure and the demands of the users.

In the case of the CityGML data model (section 3.3), for example, there is a limited basic set of supported building features and more complex models like dedicated building information models (section 3.2) can be referenced, so applications that need special information beyond the scope of the specification of CityGML can look them up.

3.1. Modeling geometry

There are different approaches to modeling the geometry in building and city models. In general, we can distinguish between *boundary representations* and *(volumetric)primitive-based descriptions* for modeling solids in 3D. In a *boundary representation*, an object is described by its boundary surface, i.e. the surface in space that separates its interior from its exterior (the rest of the universe). If this boundary surface is not closed, it does notdescribe a proper solid: There are either isolated surfaces or holes in the boundary of the solid. An important aim of *primitive-based descriptions* is to ensure that objects are indeed valid solids in space.

One of the most simple and most common representations of objects in Computer graphics is a boundary representation using a triangular mesh: the representation simply consists of a list of triangles. This approach is motivated by the fact that all continuous surfaces can be approximated by a triangle mesh to an arbitrary degree of accuracy in a Hausdorff (see section 2.1.1) sense.

A simple (not indexed) triangulation simply lists the coordinates of the vertices (corner points) of the triangles. This means that the coordinates of most vertices will be stored multiple times because most vertices will be used by two or more triangles in a closed mesh. In an *indexed* triangulation, the coordinates for each vertex are stored only once and the triangles in the list are represented by the IDs of their corner vertices, so the indexed triangulation saves a considerable amount of memory.

Another advantage of this approach is that due to small numeric inaccuracies it may happen that the coordinates for the same vertex are slightly different in different triangles in a non-indexed representation which would lead to small holes that could break the topology of an otherwise closed surface. Note, however, that there is no inherent simple operator to check the topological consistency of a triangulation: There may be overlaps and gaps between the triangles in the indexed as well as the non-indexed version. Most graphics hardware directly supports indexed and non-indexed triangle meshes.

Some data structures explicitly model topological relationships between parts of an object. An example of such a topological data structure are winged-edge data structures for 2-dimensional objects, for example the *doubly connected edge list* (DCEL) structure in which an object is represented by a list of edges with references to their start and end vertex objects and the face objects on their right and left sides and the succeeding (and preceding) edges in the loops corresponding to the adjacent faces.

For the vertices, a reference to an incident edge is stored. Using this edge, all incident edges can be retrieved following a sequence of successors and predecessors of the edge. The faces store references to one incident edge; following the successors of this edge (with respect to the face), the vertices and edges on the border of the face can be traversed. Through the links to the faces to the right and left of each edge, the topology of the

configuration is explicitly available. In a variation of this representation, the *half-edge* data structure, each edge is split into its two directed half-edges; each half-edge stores a reference to its twin (the half-edge in the opposite direction) and a link to the single face on its left side.

The advantage of such a topological data structure is that topological inconsistencies can be detected more easily than in a data structure that stores the different faces independently. Additionally, geometric operations like the Boolean operators of intersection or union can be implemented more efficiently and more robustly on such a representation; one approach to this problem can, for example, be found in Mäntylä (1986).

A different approach to representing solid objects in 3D is to compose these objects as a combination of simpler primitives. The most basic form of such a *compositional* approach is a *voxel* model: Similar to approximating an area by a raster of pixels, the 3D domain is segmented into an axis-aligned grid of voxels; the object is approximated by the union of all voxels that are intersected or covered by the object.

Such a representation can approximate the object up to the resolution of the underlying grid. Since, however, the number of cells rises cubically with the resolution of the grid, a detailed representation of a complex object will usually need an excessive amount of memory even if compression approaches like octrees are employed. For approximations, this representation can, however, be quite useful and it allows a straight-forward voxel-wise implementation of the Boolean operators.

All voxel models are topologically correct (closed) solids because they are the union of non-overlapping cubes. Especially in the context of visualization, it is sometimes necessary to extract a boundary representation from this voxel data. The most commonly used algorithm for this purpose is the Marching Cubes (Lorensen and Cline, 1987) algorithm. In its standard implementation, this algorithm does not yield a closed surface in many cases which can lead to visible artifacts in visualization scenarios and break algorithms for evaluating boundary representations. For this reason, there are modified versions of this algorithm that produce closed boundary representations.

Another common compositional approach is *Constructive Solid Geometry (CSG)*. A CSG model is defined by Boolean operations on a set of parameterized primitives – usually cuboids, ellipsoids (especially spheres), cylinders, prisms, pyramids, and cones, depending on the software package. The representation of the model is given by the CSG tree in which the leaves are the primitives involved in the construction and the internal nodes are the Boolean operations.

CSG is an extremely powerful tool and frequently used in engineering because through the union and *set difference* operators, it is easy to model a basic shape and fill in details by adding or *removing* parts – especially the second possibility of removing parts is difficult to model in other representations.

Additionally, the basic primitives involved in CSG are usually valid closed solids. This means that their boundary forms a proper closed surface bounding a volume. If regularized Boolean are used, all CSG models are valid solids: There will be no artifacts of two or less dimensions left "hanging in the air" even after set difference operations – at least if we assume infinite precision or apply topologically robust rounding operations.

In *generative modeling*, the object is described as the final result of a sequence of operations. In this general sense, CSG can be interpreted as a special case of a generative modeling approach: The model is the result of applying the Boolean operations to the basic primitives.

In his dissertation, Sven Havemann (2005) proposes a more involved generative modeling approach for meshed surfaces based on Euler operators. Models are created using these operators as parts of a stack-based programming language similar to the PostScript (Adobe, 1999) document description language.

In the framework of this *Generative Modeling Language*, sequences of basic operations like creating vertices, edges or faces can be stored and parameterized to form so-called macros that can be used to describe geometric properties of more complex structural entities.

Beyond the composite primitives of CSG, this framework specifies operators like "push-pull" in which prismatic volumes can be defined by projecting polygons in their normal direction; another important capability is the possibility to "explode" entities to get access to their boundaries: This operator is used to access the faces bordering a solid (for example, the walls in a building model) or the edges bordering an area.

The advantage of such a procedural representation is that it offers the possibility to capture the structural similarities and regularities that appear everywhere in architecture like the arrangement of windows in regular patterns (with possible exceptions like doors or staircase units with shifted windows) and uniform storey heights across otherwise different facade units in an intuitive way.

Shape grammars are another approach to describe a procedural way of creating models. Instead of a stack of objects and operators, the shapes are created by replacing *symbols* defined by the grammar using *productions*: The symbols of the grammar usually correspond to objects and structures supported by the language described by the grammar; the productions define how valid models can be formed using these concepts.

3.2. Building information models (BIM)

Building information models (*BIM*) are used to exchange building information for the construction and maintenance of buildings. One of the most widely used BIM formats is, for example, the *Industry Foundation Classes* (IFC) standard. While geographic information formats for the representation of buildings often have a strong emphasis on the exterior of a building, BIM models explicitly model many additional features like walls as a brickwork structure or the electricity and air conditioning system.

Since a BIM contains almost all information needed for the maintenance of a building, it can grow quite large. For this reason, generalization steps would, in most cases, be needed to import a BIM into a model of a city: Adding an unfiltered BIM including a complete (down to the last electric wall socket) model of a high-rise building to a city model without a generalization step would considerably increase the size and complexity of the model and may even cause a failure of signficant slow-down of the system frontend.

3.3. CityGML

The CityGML (Gröger et al., 2008) model is a standard defined by the Open Geospatial Consortium (OGC) for the exchange of city models. Beyond building models, CityGML covers a wide range of additional thematic layers for transportation, terrain, water bodies and more. Additionally, it is designed to be extensible through *Application Domain Extensions*s (ADEs) for special purposes in which additional attributes can be defined for existing feature types and new feature types can be defined. There are, for example, ADEs for noise simulation models and for the modeling of bridges and tunnels.

	Description	Positional / height accuracy		Minimum footprint diameter
LoD 0	2.5-dimensional terrain model with an optional map or aerial image texture.	–	–	–
LoD 1	Buildings modeled as blocks (flat roofs only).	5m	–	6m x 6m
LoD 2	General roof structures added.	2m	1m	4m x 4m
LoD 3	Detailed roof and facade structures, building installations (balconies, dormers, chimneys, ...).	0.5m	0.5m	2m x 2m
LoD 4	Detailed interior model added.	0.2m	0.2m	–

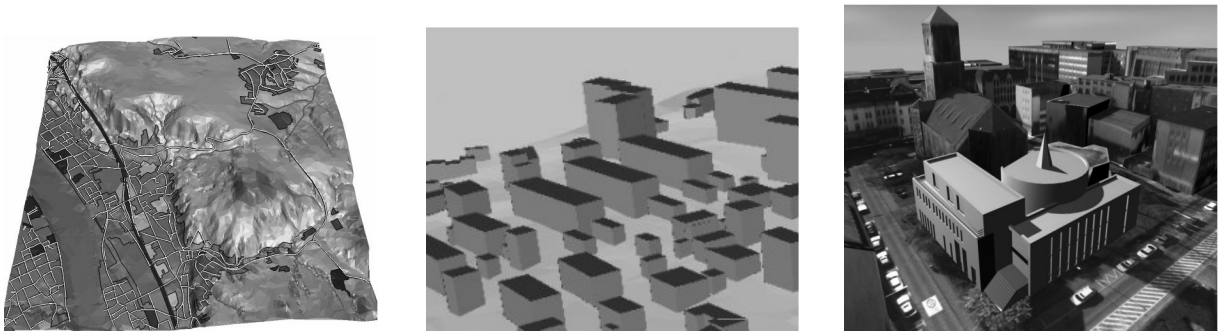
Tabelle 3.1.: CityGML LoD specification with recommended accuracies.

CityGML defines five distinct levels of detail (LoD). For each of these levels, there is a recommendation for the resolution up to which this level is intended to be used.Table 3.1 lists the features that should be added to the model for a given LoD and the recommended minimum accuracies associated with the LoD; a "–" means that the value is not specified or not applicable.

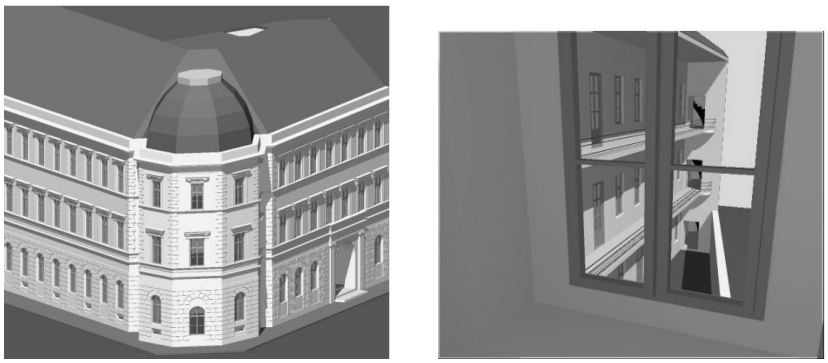
Figure 3.1 shows images of models in the different LoD; the specifications of the LoD and the images are taken from Kolbe et al. (2005). The recommendations for the choice of the different LoD are based on typical sizes for the entities added for a higher LoD: If, for example, we have a target positional resolution of 5m, then most roof structures will be small enough to be dropped according to the Hausdorff concept. For this reason, it makes sense to model roof structures only if the positional resolution is 5m or higher.

In CityGML, city models are represented in a composition tree: the more general entities like building parts maintain references to their smaller parts like roofs or walls which in turn store references to their dependent features like windows or doors.

Figure 3.2 shows some of the concepts related to the modeling of buildings in CityGML: A building is formed by building parts with a specific roof structure. This way, a larger building consisting of several parts (like wings



(a) LoD 0. (b) LoD 1. (c) LoD 2.



(d) LoD 3. (e) LoD 4.

Abbildung 3.1.: CityGML Levels-of-Detail (LoD) (from Kolbe et al. (2005)).

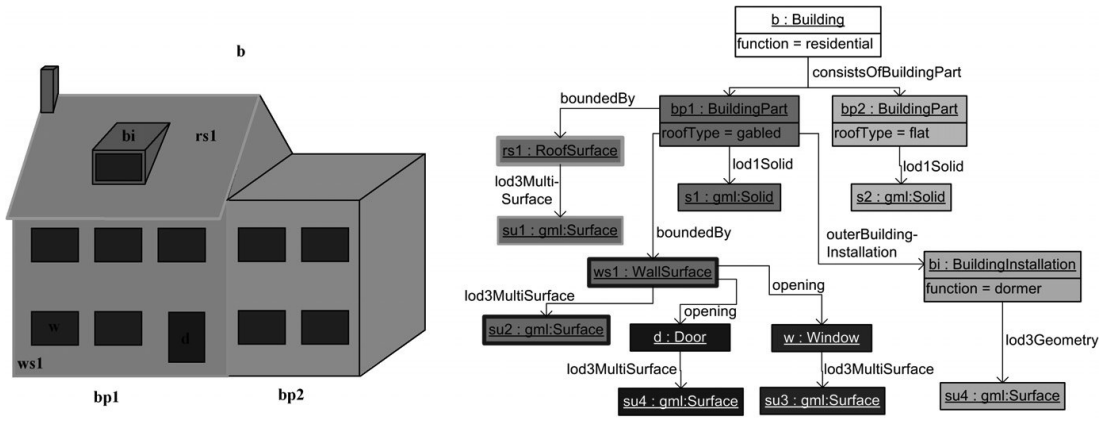


Abbildung 3.2.: Structure of CityGML LoD 3 buildings (UML, right), illustrated by a model (left) (from Gröger and Plümer (2012)).

or sections in a mostly homogeneous building block stretching along a street) can be modeled as a single entity while still preserving the different parts as distinct objects.

CityGML is an application schema of the GML (*Geographic Modeling Language*, Portele (2007)) standard defined by the OGC and the International Standard Organization ISO and inherits its geometry model. This means that the geometry of each object is represented explicitly in the form of geometric primitives like polygons and points.

While this fact gives the user the freedom to model the geometry of the features up to any degree of precision, this concept also introduces unchecked redundancies in the model because the geometry and the semantic annotations may not match. Consider, for example, the building in figure 3.2: If the roofType annotation for building part bp1 had been *flat* instead of *gabled*, the model would still have been valid but semantically wrong.

Trying to automatically derive semantic information from unstructured data – which is essentially the general

feature extraction problem described in section 2.4 – is, however, an extremely complex problem that is far from solved and the topic of a lot of ongoing research as outlined in section 2.4.

Nagel et al. (2009) lists some requirements for input data sets and reconstruction processes for the transformation of unstructured data to CityGML. Especially many of the consistency requirements are often not met, so many owners of 3D models shrink back from the effort of trying to extract a correct semantic structure from their data.

There is no generic way to model patterns of features like regular grids or symmetric groups of features in CityGML. Such structural information can provide valuable hints for generalization operations or may even be a necessary precondition: Operators like typification need, for example, by definition a set of at least two features as their input.

3.4. Grammar-based Modeling of Building Structures

There are several shape grammars that describe different aspects of building and city modeling – Duarte (2002), Downing and Flemming (1981), and Flemming (1987), for example, develop different grammars for buildings designed by Alvaro Siza, for patterns of bungalows, and for Queen Anne style buildings. A grammar for modeling facade structures is a crucial part of the algorithm for the extraction of facade structures from Lidar and image data of Ripperda and Brenner (2009).

The advantage of such highly specialized grammars is that the semantic concepts associated with the symbols of the grammar are clearly defined: A reference to the documentation will tell us that a feature (usually modeled by a symbol) called, for example, a “balcony” does indeed refer to a platform protruding from a wall. While this seems obvious, we will see in a moment that this correspondence may be lost if the modeler is given the degree of freedom to define own symbols and productions.

Abbildung 3.3.: A Model of a building block created using the CityEngine Software (from the CityEngine homepage (ESRI, 2012)).

In the CGA shape grammar, for example, that is used for modeling geometry in the CityEngine (ESRI, 2012) software developed (among others) by Pascal Müller et al. (2006) and Peter Wonka et al. (2003), the user can define new symbols and productions.

Using this feature, the basic infrastructure for modeling geometry can be used to effectively derive special grammars for radically different architectural styles and structures from Ancient Pompeii through the typical 19th century building blocks in Paris (Figure 3.3 shows an example created using the Paris style set of rules) to futuristic Metropolis skyscraper structures within the same framework.

This flexibility means, however, that there is no inherent correspondence between the symbols of the grammar and the semantic entities represented by these symbols. In this context, different languages, sub-type relations and abbreviations can cause problems even if the modeler strived to use readable names: A symbol representing a French window might simply be called “Window” or “Fenster” (German) or more specifically “FrenchWindow” or “FrnWnd” – introducing the problem of resolving abbreviations and sub-type relations: A generalization algorithm would have to be either specifically designed to handle French windows or to know that a French window is a special kind of window.

These problems can make concepts that are intuitively obvious for a human reader incomprehensible for an automated generalization process – although especially this semantic content could provide valuable information for application-specific generalization.

3.5. Direct vs. Generative Modeling

While the structure of a city or building is stored directly in explicit modeling systems like CityGML, it is, in a grammar-based system, usually encoded as the result of a series of productions from the start symbol of the grammar. In this respect, the relationship between explicit and grammar-based modeling systems is similar to the relationship between direct and generative modeling systems for geometry. For this reason, direct modeling systems usually use direct methods to represent the geometry of the features while grammar-based models go more naturally with generative modeling of geometry.

Note that while generative modeling is often associated with grammar-based modeling frameworks, parts of the models may be built directly in a grammar-based modeling environment and there may be generative aspects in otherwise descriptive modeling frameworks like the construction of facade structures from the composition of simple building part models.

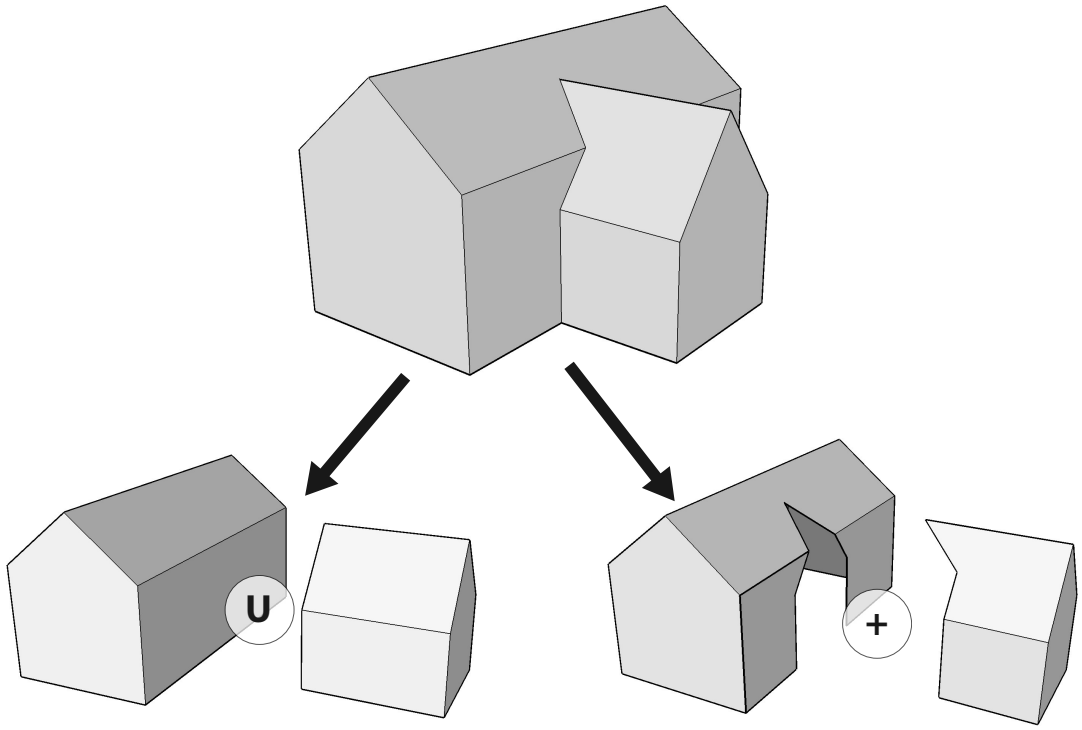


Abbildung 3.4.: Generative vs. direct modeling.

Figure 3.4 illustrates how the (moderately) complex overall shape of a building with a the small extension (top) is split up into its basic parts in two different ways in direct and generative modeling approaches:

- In the generative approach (left), the basic shapes are simple gabled-roofed building parts, but the composing operation is more involved: An underlying engine will have to be able to perform the UNION ($\cup$) operation on the primitives.
- The direct modeling system (right) stores the parts as they appear in the final model. In this case, the geometry of the parts may become arbitrarily complex and – in the case of heavy and complex overlaps – hardly recognizable, especially for automatic analysis tools like generalization processes. The composition algorithm is simple: just collect all surfaces. There is, however, no inherent way to check for (and repair) inconsistencies in the model, especially if parts of the model are changed.

While it is possible (although, due to rounding errors, sliver polygons etc., not as trivial as it may appear on the first glance) to derive a neatly partitioned boundary representation from a generative model (and most generative modeling software provides means to do so), the converse is considerably more difficult, especially if we have to interpret a given model in terms of semantically defined instead of arbitrary CSG primitives. In fact, this is bringing us back to the ubiquitous feature extraction from "polygon-soup" problem outlined in section 2.4.

The hardness of this feature identification process is a huge problem in the context of city models: It is easier to generate sets of polygons from a survey (e.g. a laser scan) than a model composed of semantically meaningful

primitives – especially having to bear in mind that such a model is probably not going to be able to interpret all occurring building structures. Additionally, almost all existing city models consist mainly of more or less structured lists of polygons.

This is probably one of the reasons for the decision of the designers of CityGML to use direct modeling and one of the reasons of the success of CityGML: Existing models can easily be transformed to CityGML. In a quick import routine, the building of figure 3.4 could, for example, be be stored as a single unstructured LOD2 building object with a list of all polygons in the direct model as its geometry. In a slightly more advanced conversion, all vertical polygons would be labeled as Wall and all other ones as Roof surfaces.

This is already a step forward towards automated analysis of city models: Instead of having to deal with more or less arbitrarily labeled ("building", "Bld", "tree") groups (if the polygons are grouped at all) of polygons forming the city model, we can now expect that

- the given group of polygons does indeed exactly (with no missing or excess surfaces) form a building (and not a tree) and
- we can distinguish between roof and wall surfaces.

Generative models are often used in order to automatically generate similar feature hypotheses for automatic model selection processes. For N. Ripperda (2010), for example, the goal is to derive a semantic interpretation of facade structures from laser scanner measurements and images. In the context of the extraction process, different facade structures are generated from the facade grammar and evaluated in a randomized (reversible jump Markov Chain Monte Carlo) framework. Both the goal of extracting a sensibly structured semantic model and the necessity of producing "similar" models favor using generative (grammar-based) models.

4. Related Work: Building model generalization

In this section, several approaches to solve different special problems in the context of 3D city model generalization are presented. The approaches can be classified by the amount of semantic information they take into account and by the levels of detail at which they can be applied.

In the first group of algorithms, building models consisting mainly of wall and roof surfaces are divided by planes into different chunks; the generalization process is applied to a semantic interpretation of the surfaces in the chunks. The simplification of LOD 1 building models through aggregation and simplification of the footprint polygons is also a topic that has been investigated in different publications. In the third section of this chapter, texture generalization and non-photorealistic drawing are introduced.

Section 4.4 introduces some algorithms for the generalization of features that are of special importance at large scales. The final section of this chapter covers approaches in which only the geometry of the surfaces forming the building models is taken into account – in these approaches, semantic knowledge about the structure of buildings appears only in the algorithms but neither in the input nor output models.

All currently available approaches address limited situations concerning feature classes, scale ranges and target application that can be handled. In the following chapters of this thesis, we will see for a few examples that even the most basic and trivial problems occurring in the context of 3D building model generalization tend to turn out to be NP-hard if they are formulated as optimization problems, so it is extremely unlikely that an integrated approach with acceptable runtime behavior for the whole problem can be developed. For this reason, it would be desirable to combine the different existing heuristic approaches.

In his dissertation, Bo Mao (2011), for example, presents a framework in which he integrates different algorithms for the generalization and visualization of 3D city models. Some of these algorithms are presented in earlier publications that form an important part of his compilation thesis. Within his framework, dedicated approaches are applied for the generalization at block, building and facade levels of detail: The framework provides a single visualization and generalization approach for a given situation – it does not directly support the integration of application-specific or other third-party generalization or feature recognition algorithms.

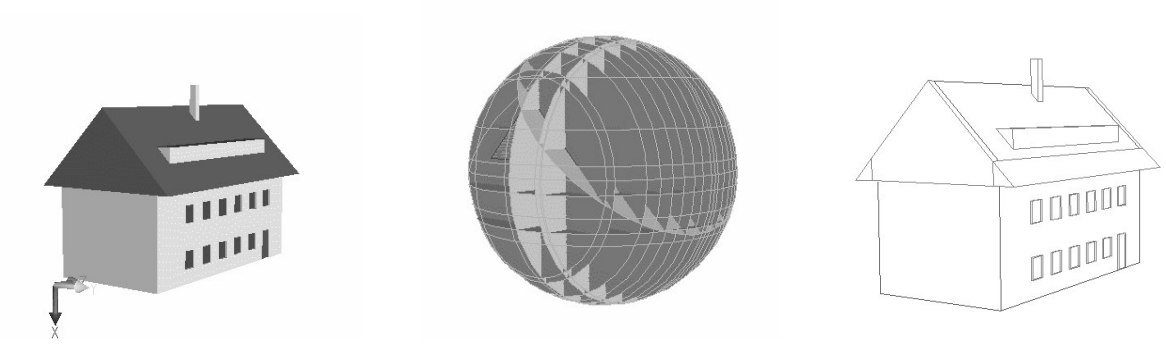
This is a significant problem in the development of the field of generalization of 3D models: There are many powerful approaches for different sub-problems but in order to create a tool that can perform 3D city model generalization for different applications, it would be desirable to have a platform in which different algorithms could be integrated which would enable the developers of simplification approaches to focus on the most relevant feature types for their applications and reuse existing simplification modules for the rest. Such a framework does, however, not exist; in section 5, some basic concepts for such a framework are presented.

4.1. Plane-based segmentation and semantic interpretation

In Thiemann (2002), Frank Thiemann outlines his idea of using CSG models for the representation of building shapes: If a suitable hierarchical representation of a feature is given, this can significantly simplify the generalization process because it may be possible to generalize different parts independently and to identify parts that may be left out for a given level of detail. In CSG models, the shape of an object is modeled by a series of Boolean operations (union or difference) between simple basic primitives; in Thiemann (2002), these primitives are convex polyhedra.

The segmentation process is described in more detail in Thiemann and Sester (2004). Similar to the approach described in Ribelles et al. (2001) for triangle meshes, the CSG model is derived from a traditional boundary surface representation by cutting the shape using the planes of its boundary surfaces as shown in figure 4.1(b); a resulting segmentation is shown in figure 4.1(c). When the building is cut by a plane, it is possible that several different connected components (features) are the result. The roof surface in 4.1(a), for example, separates the chimney and the dormer feature from the rest of the building. In each step, only one feature is separated from the rest of the building, and the process is applied recursively to both resulting parts.

In order to produce a CSG tree that gives a hierarchical description of the building that ranks the features by their geometric impact on its shape, the sequence of the cuts is important. In Thiemann and Sester (2004), the



(a) Original model. (b) Planes. (c) A Segmentation

Abbildung 4.1.: Segmentation process in Thiemann and Sester (2004)

relative enlargement of the affected surfaces $v = A_{new}/A_{old}$ in the cut plane after the cut is used to assess the quality of a feature cut, and the cut with the best (lowest) score is applied in each step. Only if no cut with a single plane yields a feature with a value $v \leq 1$, then combinations of two or more cutting planes are tested. In order to find a global optimum, all combinations of cut planes would have to be tested which would amount to an exponential number of possibilities.

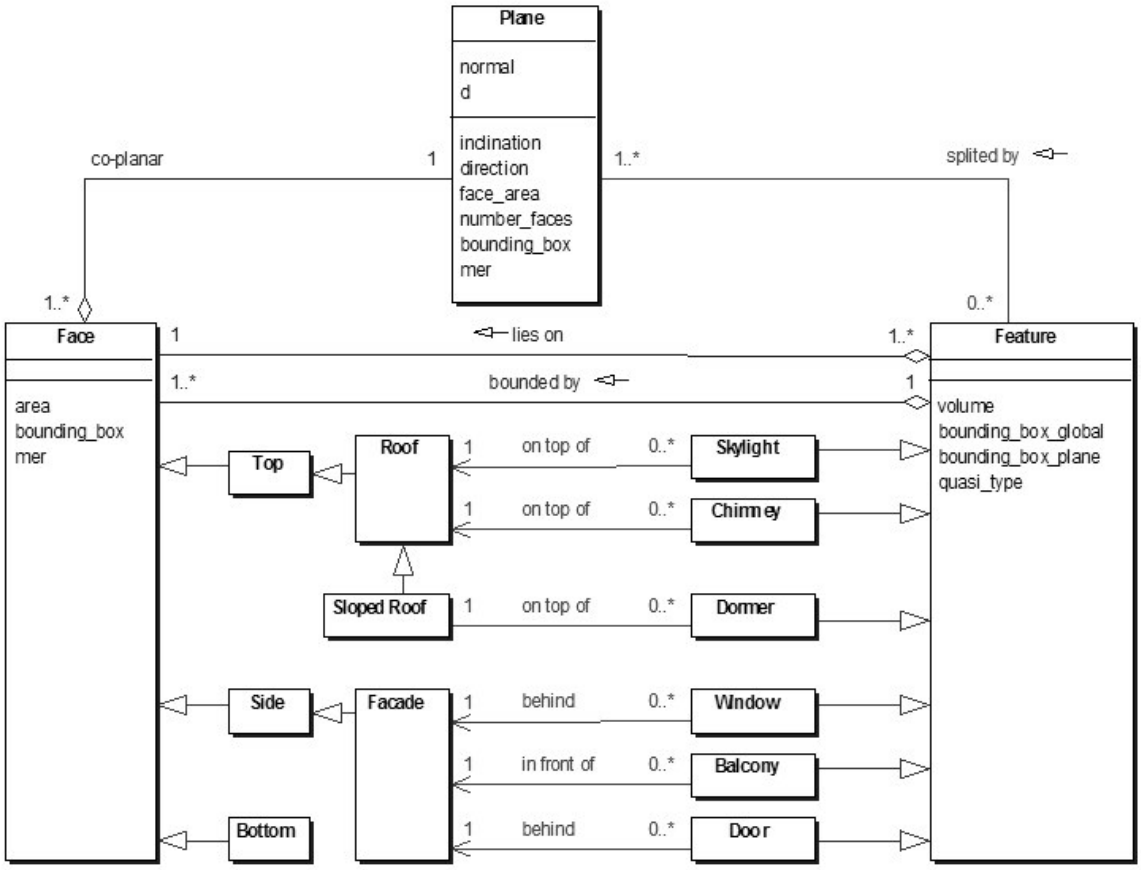


Abbildung 4.2.: Class diagram for building features according to (Thiemann and Sester, 2005)

In (Thiemann and Sester, 2005), a rule-based approach for a semantic interpretation of the CSG nodes extracted by this process is introduced: Depending on different parameters and relations between the nodes, they are classified as different features like walls, windows, doors or balconies. Figures 4.2 and 4.3 show the UML class diagram and the classification rules for different parts of a building.

Using these rules, most primary building features were identified correctly in the data sets used for the survey. Such a classification could either be used directly in semantics-based generalization processes or to identify patterns of features like regular distributions of windows and other higher-level structures in order to support

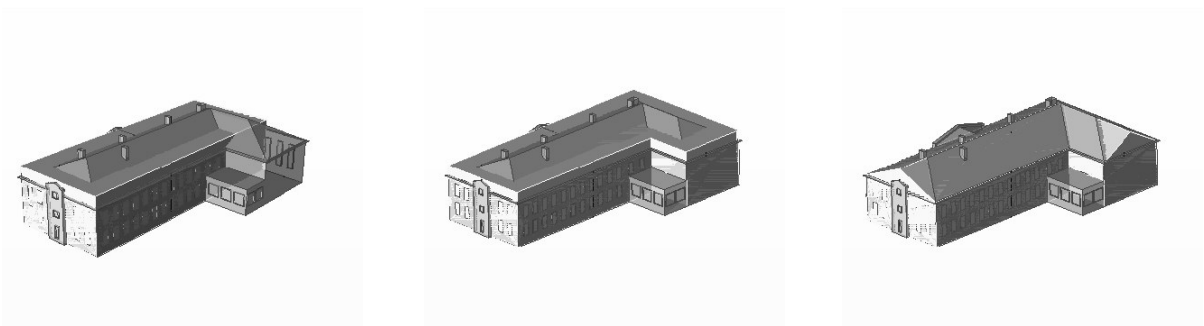
Solid features	Face	Face relation	Height relation	Size limits [m]	Features
dormer	roof	in front		$dz > 0.5$ $b > 1$	windows doors
skylight	roof	parallel in front		$d < 0.2$	none
chimney	roof	in front			none
window	façade	parallel behind	over ground	$d < 0.5$ $dz < dz_{floor}$	none
door	façade	parallel behind	on / over ground	$d < 0.5$ $2 \leq h < dz_{floor}$	none
set-off	façade	behind			any
balcony	façade	in front	over ground	$b > 1, d < 2$ $1 < dz < 1.5$	none
loggia	façade	behind	over ground	$d > 1, b > 1$ $2< dz < dz_{floor}$	none
bay	façade	in front	over ground	$d > 0.5, b > 2$ $dz \geq dz_{floor}$	windows
protrusion	façade	in front	on ground	$dz \geq dz_{floor}$	any
extension	façade	in front	on ground	$dz > 2$ $w > 1, b > 1$	any
roof projection	façade	in front	over façade	$w \geq w_{façade}$	none

Abbildung 4.3.: Classification rules for building features according to (Thiemann and Sester, 2005)

generalization operators like typification that operate on structures of features.

For this reason, the approach is a very promising step in handling the strong interdependence of generalization and feature recognition outlined in section 2.4. Especially the problem of identifying structures of features in the CSG tree remains, however, a challenge that would have to be tackled to take full advantage of the potential of this approach. The ambiguities in the order of the extracted features in the CSG tree and possibly conflicting assignments of features to different structures are the degrees of freedom in the global optimization problem of the interleaved generalization and feature extraction steps introduced in section 2.4.

In Thiemann and Sester (2006), an approach for the simplification of 3D building models is proposed that replaces a more complex building part by a parameterized primitive using least squares adjustment. For this purpose, a set of points is sampled from the surfaces of the original model and in the final adjustment, the parameters of the primitive are determined by a least squares fitting of the primitive to the point cloud. This idea is similar to the approaches used in Brenner (2003) or Brenner (2000) for the extraction of 3D building models from laser scan data.

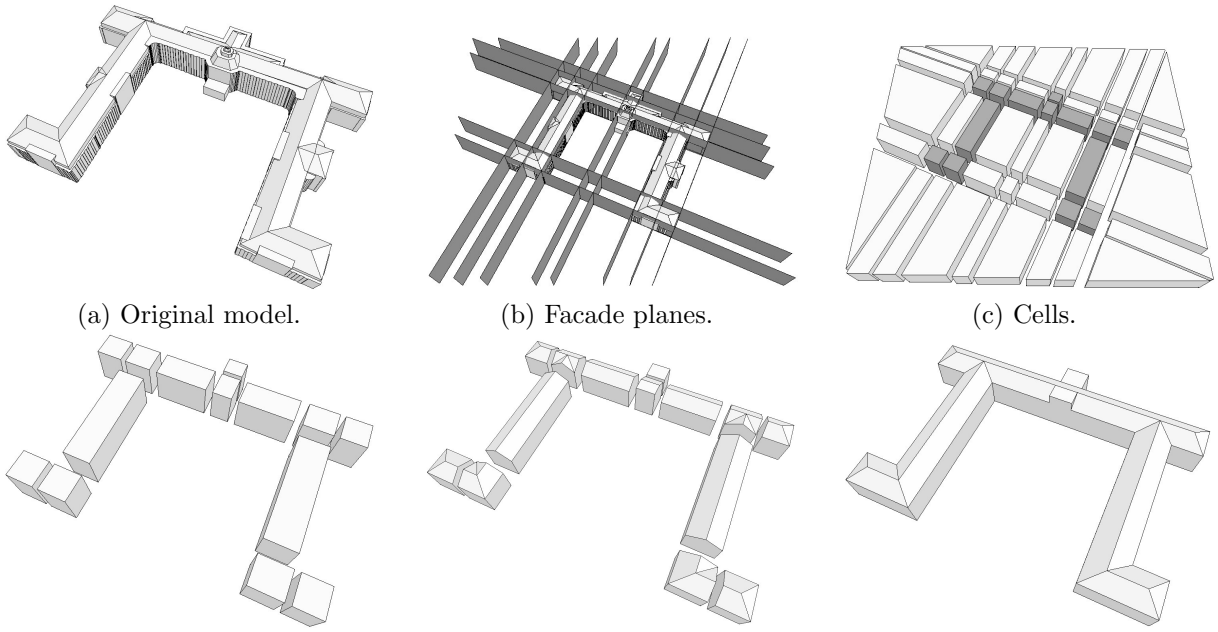


(a) Cuboid. (b) L-shaped flat roof. (c)L-shaped gabled roof.

Abbildung 4.4.: Fitting different primitives to a building model according to Thiemann and Sester (2006) (from Sester (2007))

Figure 4.4 shows the result of fitting different primitives to a more complex L-shaped building. The best primitive for a given building can be determined by an evaluation function that takes the residual error after the least squares fitting and the complexity of the fitted primitive into account.

Martin Kada (Kada, 2007b) modifies and combines the approaches in Thiemann and Sester (2004) and Thiemann and Sester (2006) to build a workflow for the simplification of complex building models.



(a) Original model. (b) Facade planes. (c) Cells.

(d) Occupied cells. (e) Primitive instancing. (f) Final result.

Abbildung 4.5.: Generalization Workflow according to Kada (2007a)

For the initial decomposition, an approach similar to the one presented in Thiemann and Sester (2004) is used. Instead of using all planes in the building model, only the wall surfaces are considered for the split operations. Additionally, all wall surfaces within a given buffer are treated as a single split plane to make sure that only the bigger facade elements are used in the decomposition. The width of the buffer can be used to control the degree of generalization that is applied to the model.

For the resulting cells (see figure 4.5(c)), the best-fitting primitives are determined in a process similar to the one described in Thiemann and Sester (2006) as shown in figure 4.5(e). As an extension to Thiemann and Sester (2006), the shapes in the adjacent cells are also taken into account when the primitives are fitted to the cells. This ensures smooth connections between the parts, resulting in a convincing overall result in many cases as shown, for example, in figure 4.5(f).

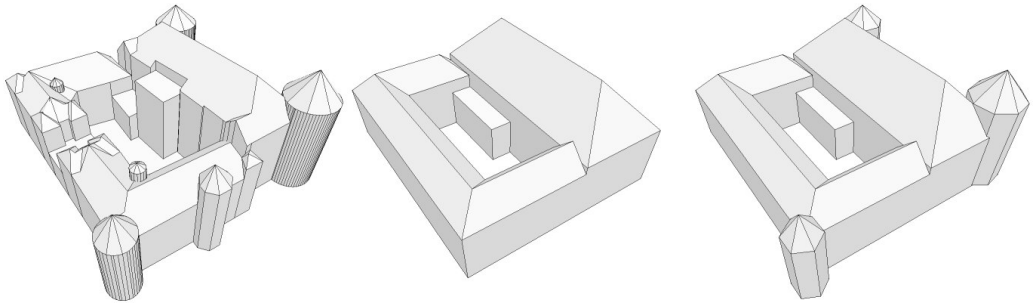


Abbildung 4.6.: Circular turrets are simplified separately in Kada (2007a).

Features with rounded footprints can lead to errors in the cell decomposition of the footprint in the first step. For this reason, they are detected and removed from the model in a preprocessing step, simplified separately, and added to the simplified model in a final postprocessing step. Figure 4.6 illustrates this process.

This basic workflow of separating different parts of the model, independently applying simplification procedures, and merging the results is an important underlying design principle in the development of a framework for the

orchestration of different generalization approaches.

4.2. Small scale simplification approaches (LoD 1)

Glander and Döllner (2007) introduce an aggregation-based generalization approach for LoD 1 (according to the CityGML specification) building models in which the buildings within a block (defined by the cells of the road network graph) are, by default, merged into a single built-up area of an intermediate height. Depending on the distance from the point of view, blocks separated by small streets may be merged across the streets.

As an exception, landmark buildings or building complexes can be retained at higher levels of detail and shown in an emphasized representation. In order to resolve conflicts caused by enlarged landmark buildings, intersecting enlarged landmarks are shifted iteratively until all conflicts are resolved.

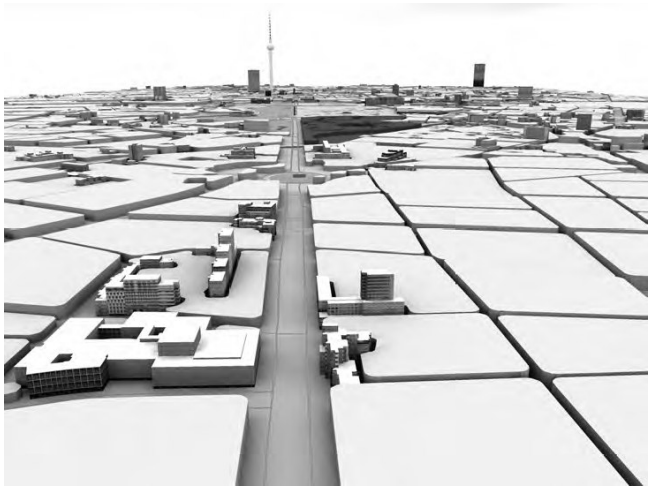


Abbildung 4.7.: Abstract representation of city blocks with landmark buildings (Glander and Döllner (2009)).

Figure 4.7 shows an example of a data set; most blocks are aggregated into built-up areas and some landmark buildings were preserved. while the landmarks in the foreground are mainly emphasized by their higher level of detail (being retained as distinct entities with a close-to-original geometric representation), the landmarks in the distance are enlarged in order to enable the viewer to recognize them.

The approach also defines intervals of distances at which each landmark is displayed in different ways: At close distances, the landmark is displayed in its full geometry without enlargement; at intermediate distances, the feature will always cover the same amount of pixels in the display, at large distances, it disappears. This corresponds to the life span of traditional cartographic point features in scale space: At large scales, they may be polygons, at intermediate scales, they are represented by symbols, at small scales, they may disappear.

Since there is usually only a limited number of landmarks (compared to the total amount of space available) and the landmark buildings take precedence over the road parcels and the rest of the built-up area in the block, deadlock situations in the iterative displacement procedure for enhanced landmarks in which no shift is possible because the space available is not sufficient for all features – for example, because a cell in the road network could not hold all features – are highly unlikely in this case.

Because in real-world data sets there are usually few landmarks and it is usually possible to extend the space covered by the conflicting landmarks, live lock situations in which a series of displacements produces a situation that was encountered before (leading to an infinite loop in the iterative displacement algorithm) are also not relevant for the approach in practice.

Anders (2005) proposes a generalization approach for 3D building models that is based on the combination of the 2D simplification of the outlines of the building ensembles in the main directions of the building.

Figure 4.8 shows the different steps of the algorithm: In the first step, the main directions of a building ensemble are determined, and the buildings are projected parallel to the resulting main directions. These “shadows” of the buildings in the main directions are then simplified using a polygon simplification approach. The simplified shadows are then extruded in the corresponding directions, and the resulting shape is determined as the intersection of these extruded polygons.

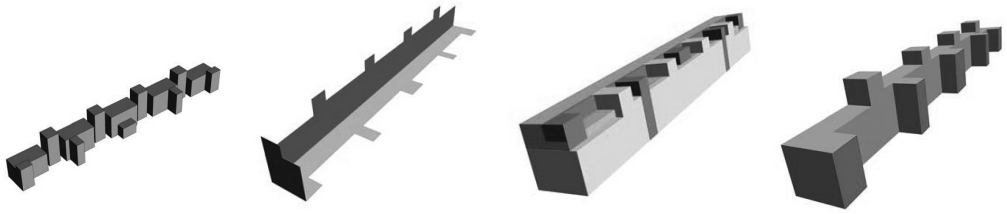


Abbildung 4.8.: Simplification of projections in main directions (Anders (2005)).

The example shows, for example, that the heights of buildings that are shadowed by bigger ones in both horizontal main direction may be enlarged without control (e.g. in Figure 4.8 in the center). This approach is mainly suited for the simplification of LoD 1 buildings because the simplification of the outlines of sloped structures may be a problem.

Additionally, the risk of occlusions increases strongly with the introduction of additional features like dormers that appear at larger scales.

4.3. Texture simplification and non-photorealistic rendering

In Döllner and Buchholz (2005), a building model is introduced that supports different levels of quality for different buildings. These different levels refer mainly to the design process: Details can be added to existing buildings. The system does not support generalization in the sense that less detailed models can be derived from a detailed one according to the needs of an application.

It offers, however, the option of real-time photorealistic and non-photorealistic rendering using textures of different complexity for objects at close and far distances. It is also possible to retain higher levels of detail for landmark buildings at larger distances.

Abbildung 4.9.: Non-photorealistic rendering of a scene from a city model (in Döllner and Buchholz (2005)).

In order to store and retrieve textures at different levels of detail efficiently, they are organized in a tree of texture atlases in which the textures are stored in a quadtree structure defined on the distribution of the buildings in the model. This enables the system to retrieve the textures for the building features at an appropriate level of detail according to their closeness to the camera position.

Especially the feature of non-photorealistic rendering is interesting in the context of generalization because it offers more options for the visualization of application-specific data.

4.4. Feature-specific large-scale simplification approaches

In some highly detailed building models – especially if they are derived from building information models – the walls are often modeled as solid 3D objects. In the CityGML standard, solid wall structures can be found in

models of LoD 3 or 4. In order to reduce the complexity of such a model, Fan et al. (2009) propose to extract the exterior surfaces and use them to represent the building, reducing the solid walls to 2D wall surfaces.

In order to identify the exterior surfaces, a central point for the building is calculated in a first step. The authors report that the centroid of the centroids of all wall surfaces had a higher likelihood of being located inside the building than the centroid of all corner points of the surfaces of the wall solids. For this reason, the centroid of the centroids was chosen. Note that this is still not guaranteed to lie inside of the building.

In the next step, all planes that belong to the same wall are identified in the original model. In CityGML, this correspondence is often modeled explicitly: The geometry of a CityGML WallSurface feature is given by a GML MultiSurface object that includes the surfaces bordering the solid wall element. Then an overall plane equation for the wall is determined by a least squares fitting of a plane to the point cloud defined by the corners of the wall's surfaces.

In the following step, the side surfaces of the wall are eliminated in a selection process: All surfaces that are perpendicular to the wall plane are discarded. Of the remaining surfaces, those with the greatest distance to the building center determined in the first step form the exterior shell that is used as a representative of the building. Finally, the window and door features associated with the wall are projected on the exterior shell.

In this context, Fan (2010) also presents a more sophisticated approach to the typification problem for regular distributions of windows on facades together with a user survey in which the quality of different window typification approaches was rated by students of geodesy.

4.5. Geometry-based simplification of adjacent roof and wall surface combinations

In the 2-dimensional case, the combination of feature extraction from geometric data and the simplification (usually through removal) of the features is a common technique: In the line simplification algorithm presented by Jenks (1989), for example, outliers and small structures are identified and removed in a filtering process in which a window of tree points on the line is examined and the second one is eliminated if it is too close to the first or the first and third point are close (one-point outlier detection) or the two line segments linking the three points are almost collinear.

The algorithm for the simplification of building footprints presented in Sester (2001) allows to identify and simplify more sophisticated patterns formed by consecutive vertices and is better adapted to the special case of building footprints: There are procedures to make sure that the result is still a closed polygon without self-intersections in the implementation; additionally, the orthogonality of the different wall segments is better preserved than in most previous line simplification algorithms.

Forberg (2007) extends this principle of simplifying local geometric patterns to the third dimension. In Forberg (2005) she shows that using vector-based dilation and erosion techniques alone, not all small intrusions and extrusions of sizes below a given resolution can be identified and removed.

She introduces an approach for the simplification of models that are composed of orthogonal surfaces that is based on shifting surfaces in the direction of their normals in order to fill small gaps or to remove small protrusions.

In order to apply these simplifications, all surfaces involved in the local patterns are supposed to be coplanar or perpendicular to each other. Especially sloped roof surfaces are, however, not aligned this way. For this reason, the set of rules illustrated in Figure 4.10 is used to orthogonalize sloped roof surfaces; the applicability of the rules depends on the target resolution and the size of the structures to be simplified. This means that this orthogonalization is a generalization step in its own right.

Fan et al. (2009) extend the building footprint simplification approach described in Sester (2001) by introducing further rules for the identification of further special structures in the footprint polygon in order to simplify LoD 1 building models. Bo Mao (2011) uses this approach to simplify the outlines of LoD 1 buildings in his framework for the generalization of 3D building models.

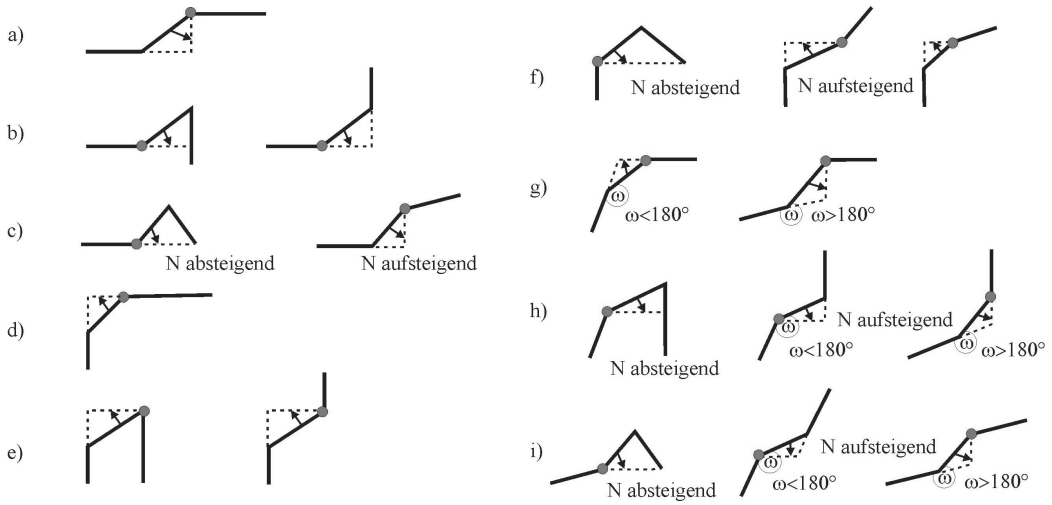


Abbildung 4.10.: Rules for the orthogonalization of adjacent roof surfaces (side views, taken from Forberg (2005)).

5. Towards a flexible infrastructure for orchestrating the generalization and structure recognition cycle

In the course of this thesis, two of the most basic and straightforward parts of the generalization problem for 3D building models – the aggregation of LoD 1 building models and the simplification of facade structures – are going to be shown to be NP-hard, so it is unlikely that a fast or even a polynomial-time algorithm exists that finds an optimum generalization result. For this reason, it makes sense to develop heuristic algorithms to find good solutions using an acceptable amount of resources.

In the face of this high complexity of even the most basic parts, the search space for the overall building generalization problem has to be reduced. In order to produce high-quality solutions with an acceptable need of computing resources, knowledge about semantics of the objects to be simplified is crucial: A specific heuristic approach for the simplification of a given feature (class) should be straightforward, computationally efficient and include options to deal with conflicts arising from the simplification of adjacent features.

In Guercke and Brenner (2009), Guercke et al. (2009a) and Guercke et al. (2009b), some initial considerations are outlined for a framework for the integration of different generalization and pattern identification procedures. In this section, a more comprehensive and structured treatment of this subject will be presented.

5.1. A semantics-based model to support the generalization process

The city model used in the generalization framework is designed with the goals of extensibility and minimum parameterization in mind. Extensibility refers to the possibility of adding new feature types – preferably as subtypes of existing classes – and to add application-specific data to existing features. Minimum parameterization means that the definition (initialization) of a feature should need as few parameters as possible.

In order to achieve such a minimum parameterization, the model combines a direct and a generative representation: The final model contains a full representation of the geometry of the features, but the parameters and structures used for the generative construction of the model are retained and can be used for the analysis of the model and as a basis for the construction of the generalized version.

This model is similar to the one introduced by A. Fischer (2005) for the automatic reconstruction of building models from aerial images. Especially the concepts of a minimum parameterization and the inference of parameter values for dependent features from higher-level features are used in the reconstruction process.

The model presented here is, however, more generic and flexible because it allows arbitrary compositions (CSG operations) of features while the model presented by Fischer (2005) supports only special compositions. The increased felxibility of the model presented here comes, however, at the price of increased computational cost because the intersections between the features involved in the composition have to be calculated explicitly.

Most city models that are currently available are not composed of this kind of parameterized primitives; their shape is usually defined by a set of surfaces in a boundary representation in purely geometric as well as in CityGML models – in CityGML, it is, however, possible to add hints to model entities that indicate a special shape: A roof may, for example, have an annotation stating that it is (supposed to be) a gabled roof. The problem of deriving such a primitive-based representation from a surface-based one is strongly related to the problem of identifying sematic structures in unstructured surface sets outlined by C. Nagel et al. (2009): Parameterized semantic primitives are an abstract representation for parts(sub-trees in the model hierarchy) of the model in which the geometry is defined implicitly by the semantics.

5.1.1. Features as composite entities

The model is organized in a hierarchical way based on the semantic structure of the basic feature set. In this model, there is a strong distinction between the necessary *parts* of a feature and optional *additions*: The types and names of the necessary parts are defined statically for each feature type – a building part, for example, ***consists of*** a body (brickwork structure) and a roof. This ***consists_of*** relation means that a child is a part

of its parent rather than an addition: The parent ***consists of*** its parts, meaning that it is incomplete if one of them is missing.

In the default feature set, the valid types for optional additions are not restricted. An application may, however, introduce restrictions on the types of possible additions for each feature type. In order to formalize the options for creating a valid model for a given application, a grammar can be used in which the productions for creating additions to the features are limited to a sensible subset of the feature types.

While limiting the set of feature types that may be attached as optional additions to a feature of a given type, such a limitation is not included in the basic framework, because constellations that may not make sense on the first glance can prove useful: Adding a roof to a window may not seem logical, but if it is allowed, we can use all the features in our roof modeling feature toolkit to model roof-like constructions on protruding window frames. The same holds, for example, for roof constructions on chimneys.

Allowing to add a building part or even a building object to a wall or another building part makes the whole expressibility of the building model framework available for modeling protruding additions to a building. Due to these reusability considerations, there are no constraints concerning minimum sizes for a given feature type in the basic feature model. This kind of constraints may be introduced by an application in a special grammar specifying constraints on valid models.

The distinction between parts and additions is important especially in the context of generalization. Assume, for example, a simple selection process based on the size of a feature and imagine a feature that, as a whole, is big enough to be retained in the generalization process while one (or some or all) of its necessary parts fall below the size threshold. In such a case, we would, for example, be left with with a building part without a roof or with a huge building complex that was erased completely because all of its parts were just too small to be retained.

For this reason, the default generalization behavior will assume that all parts of a feature have to be retained, even if they would usually be removed in a selection process, if the parent feature should be retained. The child feature will, however, be simplified to the most simple representation available in such a case. As the result of such a valid simplification of the parts, the parent feature may also become irrelevant and be removed.

Additions are optional parts of a feature that may be deleted by generalization processes. Such a removal of additions can cause the parent feature to become irrelevant, especially in geometry-based selection processes.

5.1.2. Abstract view of the model hierarchy

Figure 5.1 shows the most abstract view of the feature composition tree. The underscore "_" in front of the class identifiers in the UML diagrams means that a class is abstract or, in Java terminology, an interface. For the sake of readability and conciseness, these classes have public properties instead of abstract getter and setter methods in the diagrams.

The most important relation is the *part* relationship: A higher-level feature is composed of its *part* features (terminal features have no parts). In the diagram in 5.1, this relation is emphasized by a bold arrow with a filled tip, while the minor importance of the additions is emphasized by a dotted arrow. Although the this is the reverse direction of the standard UML aggregation symbol, the direction of the arrow from the feature to the parts was chosen for two reasons: the feature holds the references to its parts, so it is their "owner" and – in contrast to the difference between aggregation and composition in UML – the difference between a part and an addition is that the parent feature is not complete without its *part*s, while the UML composition is used if the parts depend on the parent feature.

The shape of all features is defined in their own coordinate system: Wherever this is sensible, a coordinate system in which the feature's shape can be represented by a minimum number of parameters is used. For a rectangular building part, for example, such a "natural" coordinate system will have its x and y axes aligned with its sides because in this case, it can be represented completely by its width and depth (and eaves height).

In the case of skewed planes like an inclined rectangular roof plane, for example, a "minimum" coordinate system would be aligned with the rectangle. In this case, however, it would be necessary to keep track of the local UP vector throughout the whole model in order to be able to access it when child features like chimneys and dormers have to be aligned because those features are usually pointing upwards. For this reason, the z axis is always assumed to point in the direction of the UP vector which should, within the bounds of the precision associated with the model, point in the opposite direction of the local gravity vector.

Since the shape of a feature is defined in its own coordinate system, a transform will usually be needed in relations between features in order to perform spatial analysis on them and in order to place the lower-level features in the composition hierarchy; this is the reason why relations between features will usually involve a *_FeatureLink* object that stores the source and target of the relation together with the transform from the source to the target coordinate system.

Application data can be added to all features in the form of (key,value) pairs in which the key is the name of the attribute and the value may be any type; in the case of a Java-based implementation, the type of the values would be *Object*; access to this application data is managed through the getData(..) method of the *_Feature* interface where the *KeyClass* of the object representing the parameter that is accessed may simply be a string or an object of a special parameter key class (which would require a registration procedure for parameter IDs but can speed up the lookup process because the features' attribute maps would be queried by object IDs instead of strings).

Application-specific feature types can be defined as subtypes of the feature classes introduced in the following. All of the feature types introduced in this section are derived from the *_Feature* interface; in order to be used within the generalization framework, a new feature type has to implement this interface.

A new feature type should, however, be derived from the most specific known feature type because only then it could be ensured that a higher-level generalization process could make its decisions based on the type of the feature; a facade layout algorithm would, for example, profit from knowing that a new type *FrenchWindow* is a special kind of window, so it could apply its layout rules for windows to it.

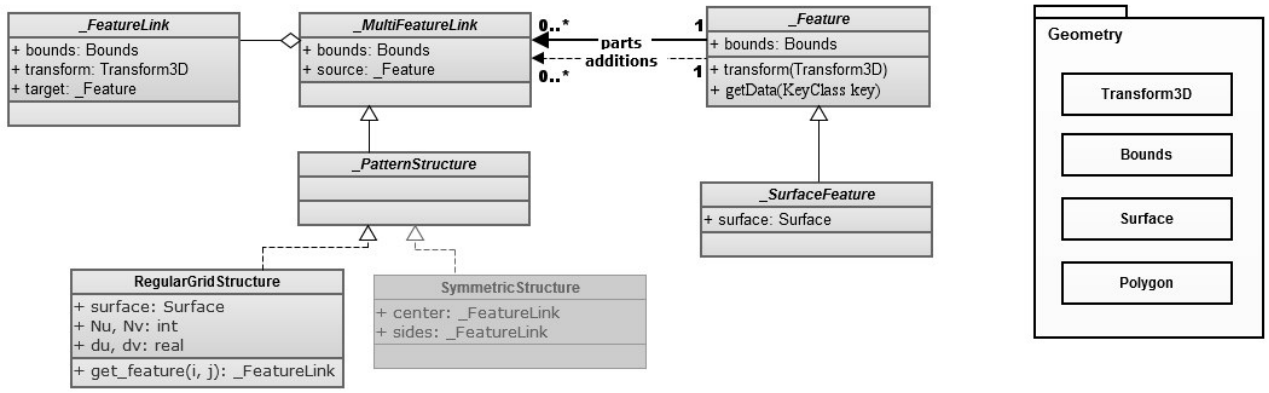


Abbildung 5.1.: Composition of features.

By default, the *part* and *addition* (for the optional additions) relations use *_MultiFeatureLink* objects that pool several *_FeatureLink* objects. This construction allows a straight-forward integration of patterns of features as implementations of the *_MultiFeatureLink* interface; the UML diagram shows the *RegularGridStructure* and the *SymmetricStructure* (not implemented in the current version) as examples.

The *Surface* class in the geometry package defines an abstract functional surface concept that maps each point $U = (u, v)$ in parameter space to a point $X = (x, y, z)$ in the enclosing 3D coordinate system and optionally provides a local normal vector N_0 for each point in parameter space.

Note that the parameters u, v, N_u, and N_v of the *RegularGridStructure* are defined in the parameter space of the underlying *Surface*. This means that distributions along bending lines or curved surfaces can be realized by changing the underlying surface. In the most cases, the surface will, however, simply be a plane, but even in this case, this representation has a great advantage: We do not need different grid structure classes for laying out horizontal (like regular distributions of buildings in the plane), vertical (windows in a wall) or skewed (features on an inclined roof surface) planar grid structures.

In order to lay out features on such inclined surfaces, the $(u, v) \rightarrow (x, y, z)$ mapping from parameter to real-world space in the *Surface* or *Plane* class usually provides the functionality needed in a very intuitive and flexible way without having to change the z direction: The underlying surface defines the position and azimuth for the feature, and the feature can rely on a z axis pointing upwards for its layout. Of course, this can lead to violations of our goal of using minimum parameter sets wherever posiible, but otherwise the UP vector would have to be traced throughout the system.

Each feature stores a reference to its 3D bounding box. In a *_FeatureLink* or *_MultiFeatureLink*, the bounding boxes of the referred features after the *_FeatureLink* transform can be stored. This allows the system to retrieve

the bounding boxes of the parts and additions of a feature and to adjust the parent's bounding box according to changes in its children.

Having the bounding boxes of a feature's children available (even if we have to calculate them on demand) turn our feature hierarchy into a search structure similar to an R tree (Guttman, 1984): If we want to check which parts of a feature intersect a given search shape, we can exclude all children from the search that have bounding boxes that do not intersect the search shape.

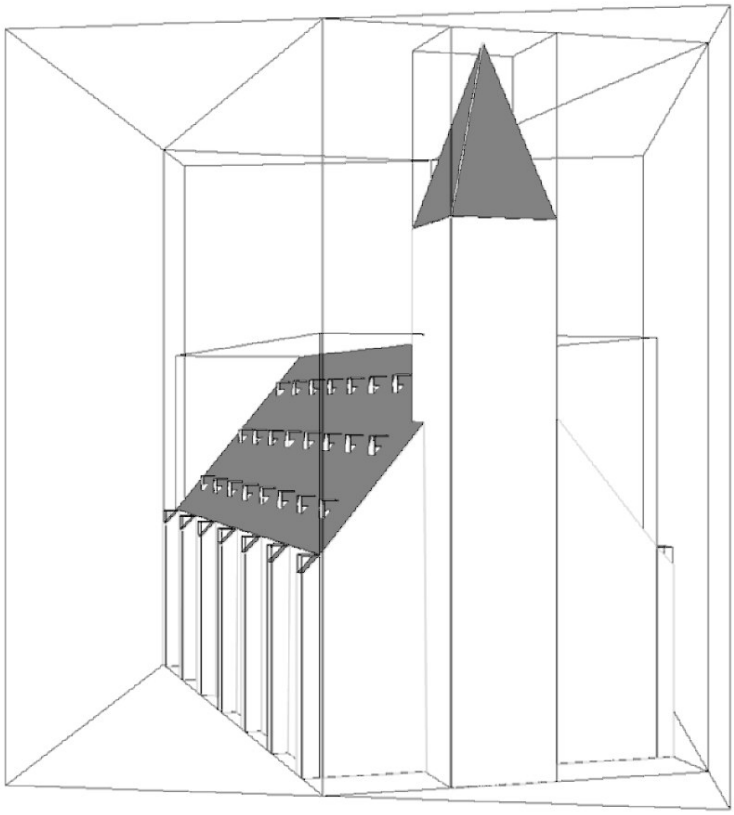


Abbildung 5.2.: Nested structure of a church-like building model.

Figure 5.2 illustrates this nested structure. The main part and the tower are the building parts of which the whole building is composed. Both have a body and a roof – the main part has a gabled roof, the tower has a tent roof. In the figure, the bounding boxes around the parts are shown. Because the z axis is aligned with the UP vector, the inclined roof surfaces need to store more that the minimum parameters of width and depth and their bounding box is larger than the minimal one that wolud have no height for an ideal plane. On the other hand, the alignment of the dormers would have been considerably more complicated if the coordinate system would have followed the direction of the roof surface on which they are defined – and impossible without a tracing of the local coordinates of the UP vector.

In this work, the emphasis is on building models. Note that there are many further thematic layers in a city model; in CityGML, there are, for example, six default thematic layers for terrain, land use, transportation, vegetation, water bodies and sites, in particular buildings. Such a more detailed structure of thematic layers can be achieved by deriving special feature types for the desired thematic view and registering a new thematic layer.

5.1.3. Notation (UML)

The documentation of the structure of the models is based on the UML (Unified Modeling Language) notation described, for example, by Rumbaugh et al. (2004). For reasons of legibility, some short notations were introduced.

In order to avoid cluttering the succeeding diagrams, the *part* relation will be marked by a **[P]** followed by the identifier by which the part can be accessed from the higher-level feature. In figure 5.3, for example, the

arrow between the *_BuildingPart* and *_BuildingBody* types means that there is a field (or rather, in accordance with the Java interface model, an abstract getter method) that refers to an *_MultiFeatureLink* object that pools references to *_FeatureLink* objects pointing to *_BuildingBody* objects.

The *parts* data field or getParts() method in the *_Feature* interface is supposed to give access to the union of all different parts of the feature, either as a mapping from the identifiers to the parts or simply as the list of the parts.

A dashed arrow with a label starting with a [**L**] means a general relation (link) from the source to the target class through a *_MultiFeatureLink*.

5.1.4. The building model

In the default feature set, a *_Building* can be composed of different *_BuildingPart*s which, in turn, can be composites. This offers the possibility of representing a nested composition structure for a building complex. A city block may, for example, consist of several building parts built along its perimeter or forming "letter-like (E, T, F, O) patterns" where each part consists of a row of other, more fine-grained building parts.

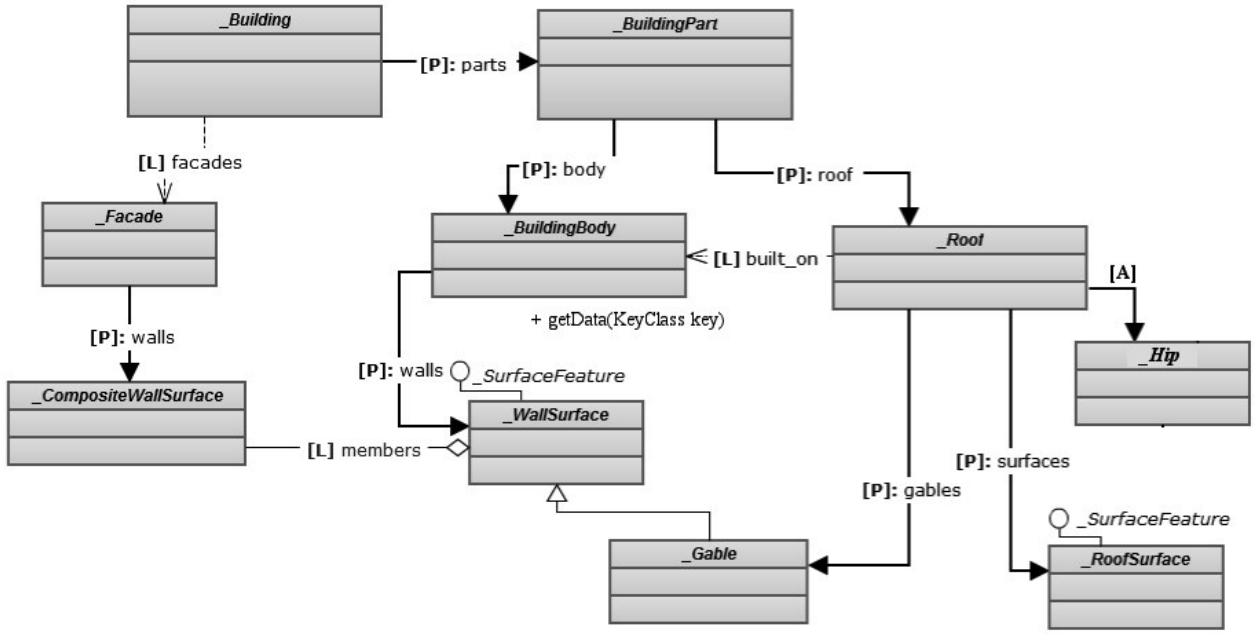


Abbildung 5.3.: Abstract structure of a building in the default feature set.

A *_BuildingPart* consists of a *_Body* and a *_Roof* structure. The *_BuildingBody* is bordered by *_WallSurface*s. The *_Roof* consists of *_RoofSurface*s and *_Gable*s which are wall surfaces that are defined by the roof structure.

In order to model facade structures that are the result of merging wall surfaces and gables from different building parts – for example, the front facades of building complexes with different wings, *_Facade* objects may be added to a building to provide a facade-oriented view of the whole building. A facade consists of *_CompositeWallSurfaces* that are, in turn, composed of different wall surfaces. Since a gable is a wall surface, gables can also be part of a composite wall surface. The [**L**] annotation on an arrow in the diagram means that the relation includes a *_MultiFeatureLink* between the source and the target.

Facades and composite wall surfaces are needed to control the distribution of facade elements like windows and doors because their distribution depends rather on the actual shape of the area on which they are distributed than on the units of which this surface was composed. The regular facade grid structures on which the simplification approaches introduced in chapter 8 operate can be integrated into the framework as *_Facade* structures.

All roofs can be modified by adding hip objects. The *_Hip* is the base class for all classic hip forms and provides a set if cutting planes. The general *_Hip* interface allows using arbitrary cut surfaces which may be needed to model the curved roof and hip surfaces in oriental architecture. In the current design, only *_PlanarHip*s are used that are restricted to planar cutting surfaces.

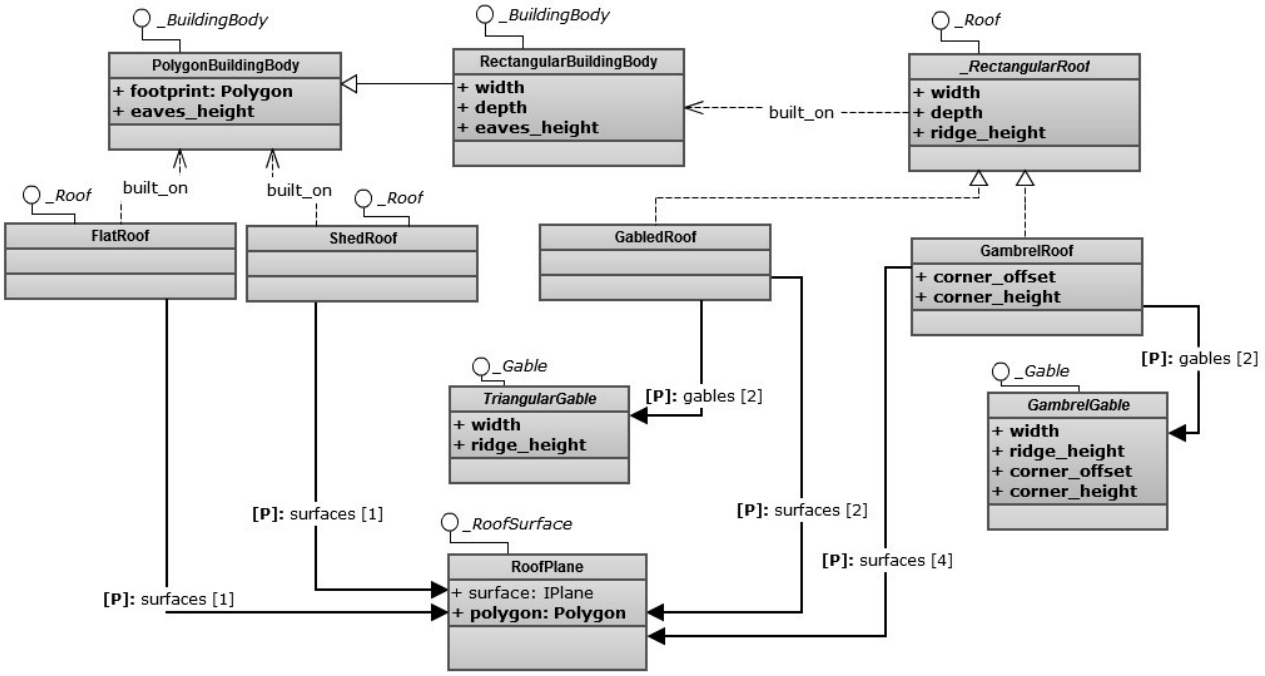


Abbildung 5.4.: Building parts in the default feature set.

Figure 5.4 shows some more specific classes for the composition of building parts. The most basic roof types consisting of a single surface, the flat and the shed roof, can be placed on arbitrary footprints. The gabled and mansard roofs require rectangular footprints.

Note that the default classes assume regular (symmetric) roof structures. In order to model asymmetric structures with shifted ridge or break lines, additional *AsymmetricGabledRoof* or *AsymmetricMansardRoof* types would have to be introduced.

The parameters printed bold in the UML diagram are the minimum parameters needed to describe the feature completely. The set of these parameters is used in the model construction process described in section 5.2. In the process of assembling the model, these parameters can, in many cases, also be derived from the construction parameters of other features.

An example for such an inference of parameters is the construction of a building part with a gabled roof: For the construction of such a building part, we only need to know its width, depth, eaves and ridge height, and optionally a roof overhang width and a Boolean value to specify if the gable is aligned with the width or with the depth of the rectangular body.

Not all building models to be handled in the generalization framework will include the high levels of detail that can be modeled in terms of the feature hierarchy introduced here. Many of the currently available city models are limited to LoD 1 building models in which all buildings are represented only by their footprints and a uniform height.

In order to distinguish between LoD 1 buildings and buildings that are actually known to have a flat roof, we can introduce a dedicated *LoD1Building* class implementing the *_Building* interface. Since roof structures usually have extents of several meters, LoD 1 buildings will also be notable by their low resolution in the z direction.

Building models are often created from cadastre or other high-resolution 2D map data with accuracies of a few centimeters in the footprint. Since information on the more detailed structure of the buildings like roof and facade are often not available, the highly detailed footprints are often combined with an approximate height to produce 3D building models in LoD 1.

If the low resolution resulting from the missing information about the 3D structure of the buildings is indeed sufficient for an intended application, the models can be simplified considerably by aggregating buildings and by simplifying the footprint polygons. In chapter 7, an optimization-based and two heuristic approaches for the aggregation of LoD 1 buildings and a heuristic approach for the simplification of the footprint polygons are introduced.

5.2. Assembling the model

Having to specify all parameters in the construction of all features in the building hierarchy would lead to ambiguities and create numerous opportunities for errors. For this reason, the concrete classes in the feature hierarchy expose a *minimum set of parameters* that completely define a feature of this class.

By default, features are initialized from such a minimum parameter set to the maximum depth in the composition hierarchy: Knowing, for example, the width, depth and eaves heights of a *RectangularBuildingPart*, we can immediately instantiate its *_Body* with the floor and wall surfaces as well as the transformations from the body to the natural coordinate systems of the different surfaces.

The number of necessary parameters can also be reduced if dependent parameters are inferred from existing features: The width and depth of a gabled or Mansard roof are, for example, determined by the width and depth of the building part on which the roof is built – apart from possible roof overhangs that may be defined independently.

This is a great advantage when it comes to storing, transmitting and processing models: Only those features that are actually needed have to be expanded; other features may not be needed to be created at all. Especially if there is an implicit order for the *part* children of of a feature, this property is extremely useful.

A tent roof on footprint defined by a regular footprint polygon, for example, is defined completely by the number n and length of the edges in the footprint and its ridge point height. If we want to visualize such a roof, we do not need to construct the roof surface features along with the appropriate transforms, but we can build the triangles forming the roof directly which will save us a lot of effort.

With a defined ordering of the surfaces, we can preserve this potential for saving resources even if there is an addition to one of the surfaces – a chimney, for example: In the representation and in the construction process we only need to pay special attention to the surface for which the exception occurs; all other surfaces can still share the same model instance or they can be created implicitly as in the visualization scenario.

In their parametric representation, all features are independent and intersections between features are ignored. Due to the composition of the features, however, there may be conflicts and new features may be created.

One of the most prominent cases of composite features are *_CompositeSurface* features that emerge whenever two or more surfaces from different features are joined in a continuous way, forming a single surface. A simple case is, for example, the surface formed by a gable element and the wall surface below it.

While the outer shape of such a composite surface is defined by the surfaces of which it is composed, the distribution of the features on the surfaces is usually independent of the borders of the defining surfaces: a window in the front of a gabled house may, for example, cross the line between the gable and the wall below it; as far as the facade structure is concerned, only the overall shape of the composite wall element matters.

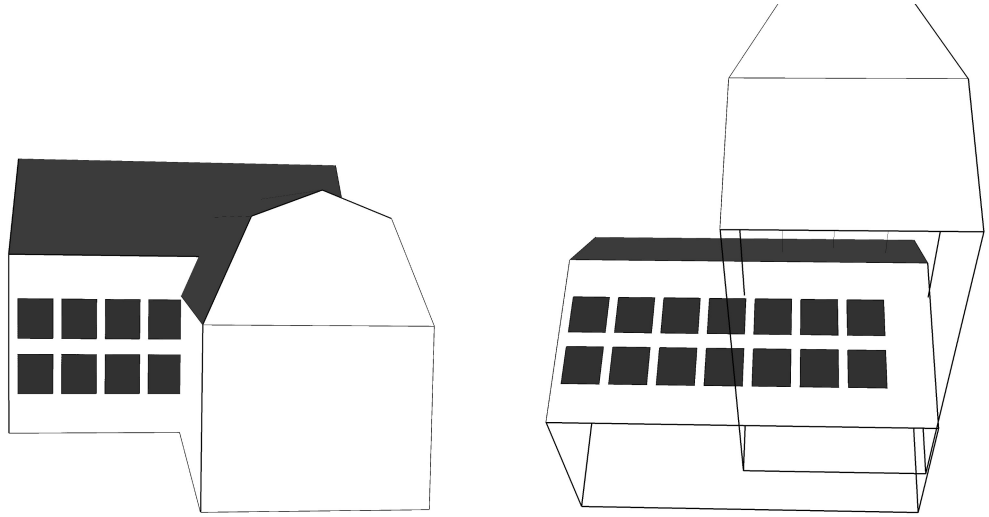
Figure 5.5 shows some features of the composite wall building process. The first row shows a situation in which a wall surface of a building part with a regular grid of window features is partially occluded by another building part if no additional measures are taken: The occluded part of the wall and the occluded windows survive in the interior of the building complex – which is, of course, very unlikely to match the true structure of the interior of the building. Especially if interior features are modeled explicitly, this is not acceptable.

For this reason, the wall composition module calculates the intersections of all other building parts with each wall surface and clips the surfaces accordingly; the result is shown in figure 5.5(b). Using the updated shape of the wall surface, a conflict resolver or the default layout function of the array (just mark the links to the occluded windows and suppress them when the array is asked for its children) may be applied to allow the array to fit to the surface.

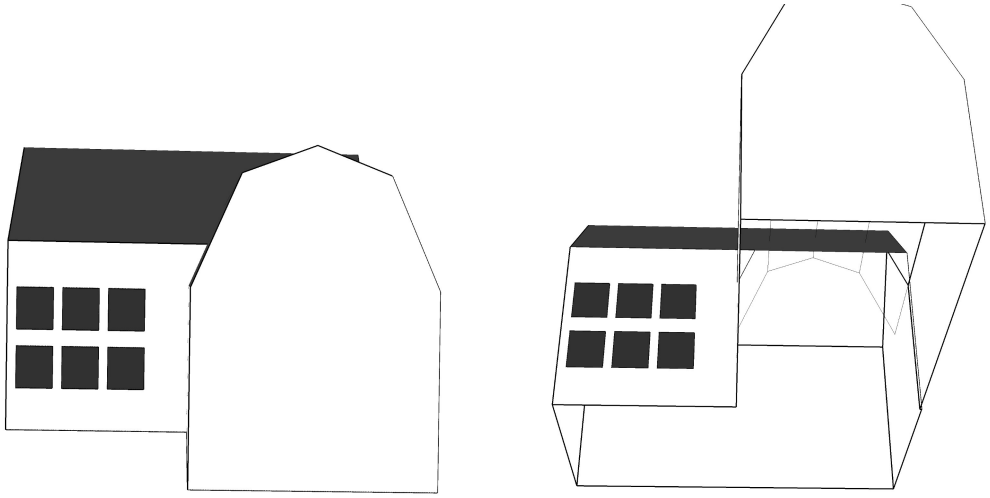
Figure 5.5(c) shows that it makes sense to merge adjacent coplanar wall surface objects into a single composite wall feature: In this case, layout strategies for facade elements have access to the true surface on which they reside and not only to the part for which they happend to have been defined.

This process has not yet been implemented for the roof surfaces; it would, however, use exactly the same principles as the composition and clip algorithms as the algorithm for building the wall surfaces.

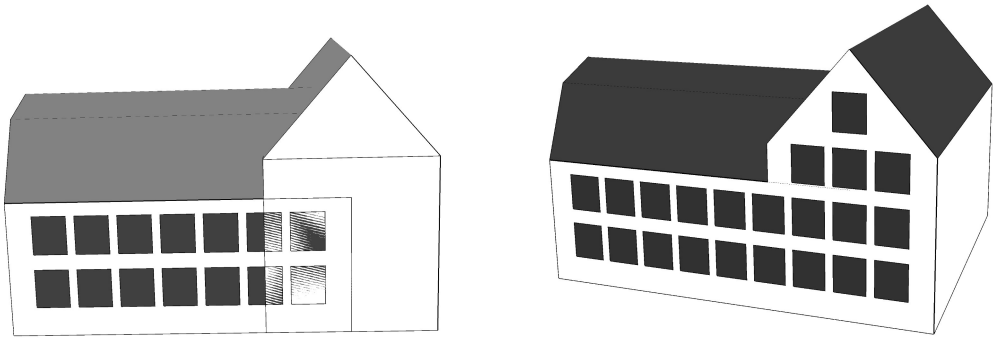
The model building sequence introduced in this section is very similar to the model construction process in the CityEngine software: In a first step, the basic volumes of the building (complex) are constructed. After that, the surfaces forming the boundary of this volume are calculated retaining the semantic classes of the surfaces (wall or roof). After that, facade structure and building installation features are added to the wall and roof surfaces.



(a) Occlusion without clipping (front and bottom front view)



(b) Occlusion with clipping (front and bottom front view)



(c) Composition with and without union operator.

Abbildung 5.5.: Features on composite walls.

5.3. Generalization as a tree traversal

A simplification module is typically designed to handle a given feature type or a special kind of *_(Multi)FeatureLink* representing a structure like regular grids of features or facade part structures similar to those introduced in

(Ripperda and Brenner, 2009). We will only consider simplification approaches that can be associated with a single feature (link) type in this section. In the following, this feature (link) type will be referred to as the *basic* feature type for which the simplification class was defined.

Note that the restriction to simplification modules for single feature (links) is not as strong as it may appear because groups of features are collected in *_(Multi)FeatureLink*s, so structure recognition and simplification can be performed on the level in composition hierarchy where the structures are defined.

The *scope* of a given simplification instance is the set of features or feature types that it can handle by itself. This scope will usually include the basic feature (link) instance with which it is associated and some of its children in the composition hierarchy, especially the *parts* of the basic feature because there is at least some partial knowledge about their feature type and semantic function while an *addition* may be anything.

Due to the consistent use of interfaces that may be realized by different concrete classes in the building model, a single simplification module will in many cases cover only its basic feature (link) for the more generic feature types: A *_BuildingPart*, for example, only knows that it consists of a *_BuildingBody* and a *_Roof*, both of which may have quite complex structures. A generic *_BuildingPart* simplification module will therefore have to rely on external modules for the simplification of the *_BuildingBody* and the *_Roof*, performing only a basic check that there is no gap between the resulting roof and body objects.

A simplification module for a concrete class like a *MansardRoof* can, on the other hand, be much more concrete itself: Depending on the resolution and on the intent of the developer, it will, for example, return a gabled or a flat roof. The height of a resulting flat roof illustrates that there are many options even for a very simple generalization operator: It could be fixed to the average, the maximum (ridge) or the minimum (eaves) height of the original roof or to a height that preserved the volume of the original.

Alternatively, it can be marked as a free variable for simplification processes on higher levels of the composition hierarchy. The aggregation approach for LoD 1 (flat roof) building models described in section 7, for example, uses such ranges for the possible heights of the buildings to determine which buildings may be aggregated and to set the heights of the aggregated buildings.

The key of the modular approach that is introduced in this section is that there is a callback or orchestration module that is available to all simplification modules. Whenever a simplification module instance encounters a child feature that is not within its scope (or that it does not want to handle for some reason), it can ask the callback module to produce a simplification handler for this feature.

It will then ask the handler to produce a simplified version or a set of simplified versions of the client feature for evaluation. The simplification module for a given feature is responsible for the integration of the simplified child features. If the handlers for the child features offer different simplified versions, it can choose the combination of simplified children that scores best with regard to the goal of the generalization process.

The objective of the generalization process depends on the application. For this reason, a single basic implementation can only cover the most basic use cases. A most promising approach to save users from having to implement specific handlers for all combinations of features and use cases is to provide parameterized generalization tools for the standard features that can generate different feature-specific simplification options for the geometry, ensuring that the result is a valid feature (for example, from a Mansard to a gabled or flat roof) but not taking the semantic context of the feature into account.

One of the most crucial problems in the context of generalization is the weighting of structural (geometric) and semantic parameters in the objective. One way to deal with this problem is to provide default values for the parameters and offer different granularities of fine-tuning options like IMPORTANT vs. INSIGNIFICANT (that are transformed to weights internally based on the range of the parameter values) or more elaborate mappings from parameter value differences to penalties for the objective function.

The critical part of this approach is to check if integrity constraints are violated when the simplified features are combined to form the overall simplified feature. At this point, the generative approach to the construction of a building model is an advantage: The generalization process is, in fact, building a model in a very similar way to the original construction, so integrity checks for the model construction can be reused in the generalization. The fact that this approach enables integrity checks to be performed on higher semantic abstraction levels offsets the initial effort of having had to implement clipping algorithms to determine the resulting shapes from the generative description of the model.

Figure 5.6 shows the generalization process for a simple composite facade surface with a regular grid of window objects; the generalization parameter ϱ is the geometric (Hausdorff) difference threshold introduced in section

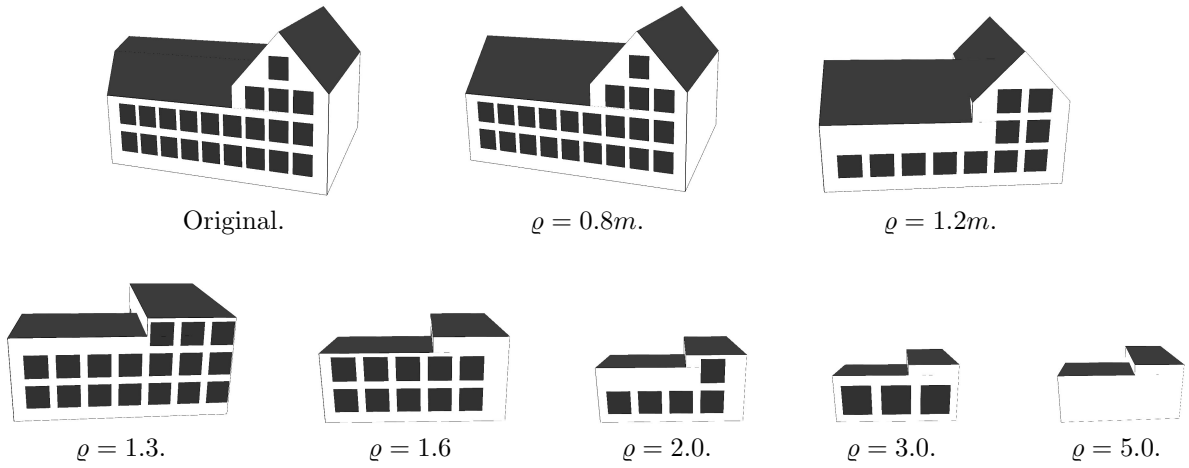


Abbildung 5.6.: Generalization sequence for a simple facade structure at different resolutions

2.1.1. Note that the shape of the surface changes as the shapes of the roofs change from gabled to flat roofs. Using the assembling process described in the previous section, many of the possible conflicts caused by these changes in the supporting surface of an array of holes can be handled gracefully by the conflict resolution of the generating process.

Using the callback mechanism introduced here, algorithms for the simplification of features modeled at higher levels of detail than the standard classes can be integrated in the generalization process as long as suitable integrity concepts exist. An example for such a situation is the approach for the simplification of highly detailed roof models consisting of individual roof tiles modeled as 3D objects introduced in (Guercke et al., 2010): This approach together with the more detailed roof surface model can be integrated in the standard generalization workflow if the supporting beams between roof and wall surfaces are modeled in a consistent way.

5.4. Summary and Discussion

In this chapter, a modular approach for the flexible integration of different simplification operators in a generalization process for 3D building models was presented. In its core, this approach consists of a flexible framework for the management of different generalization handlers that collaborate to construct a resulting generalized building model in a generative way. The generalization handlers can also be seen as agents in an agent-based framework.

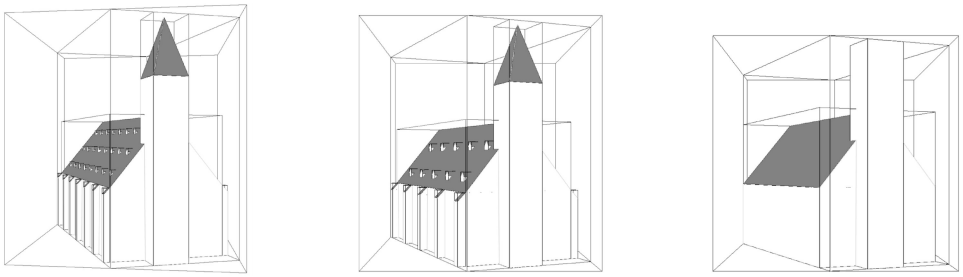


Abbildung 5.7.: Generalization sequence for a church-like building model.

Due to the composition-based strategy, complex building structures can be assembled from simple basic shapes, and the more complex Boolean operations for the construction of the resulting wall and roof surfaces are handled by the underlying geometry engine. Figure 5.7 shows a generalization sequence for a slightly more complex building model that can be handled by the current version of the standard feature set included in the framework.

Note that the bounding boxes of the features in the hierarchy form a nested structure similar to an R-tree (Guttman, 1984) (although they are not aligned with the coordinate axes and explicitly balanced) that can be used for efficient spatial queries.

Abbildung 5.8.: A facade structure covering a part of a block.

The current version of the framework handles only the quite basic feature set introduced in this chapter. In order to process more complex situations, more sophistcated models are going to be helpful or necessary – especially for modeling facade structures.

So far, the resolution of conflicts and measures for the improvement of the quality of the resulting generalized model are currently completely left to the individual generalization handlers; currently, the default composite surface builders and error handlers of the generative modeling infrastructure are used.

In order to account for the combinatoric nature (the global optimum may be a combination of locally sub-optimal solutions) of the building generalization problem (which causes it to be NP-hard in many cases), a sophisticated backtracking system is necessary that enables the generalization handlers to propose alternative generalized features that resolve hard conflicts reliably and lead to an improvement of the global quality of the overall generalized model.

6. Geometric Building Footprint Simplification Using a Modified Hough Transform and Least Squares Adjustment

The first step in the generalization of data sets containing LoD 1 building models is usually the simplification of the footprints. In many cases of data sets currently available – especially if the input is cadastre data –, we have large data sets containing very detailed footprints with accuracies in the range of centimeters, but there is little or no information on the roof shapes or the structure of the facades. If we are interested in a general resolution (not specially focused on the shape of the footprint), this means that these models can only provide an overall accuracy of several meters in 3D because the roof and facade structures can easily have sizes in these ranges. For this reason, it does not make sense to preserve the fine-grained structures of the footprint in a 3D model with a uniform target resolution because this feigns an accuracy that cannot be guaranteed by the model and only creates huge amounts of data.

Several approaches have been proposed to tackle the problem of line simplification. In selection-based approaches like the ones presented by Douglas and Peucker (1973) or Jenks (1989), the vertices of the resulting line are a subset of the vertices of the original line.

Estkowski and Mitchell (2001) showed that the basic problem of selecting a minimum number of vertices from an original line string that form a line string without self-intersections with a Hausdorff distance (see section 2.1.1) of less than ϵ to the original is NP-hard. Haunert and Wolff (2008) introduce a way to include size constraints into a MIP model of the polygon simplification problem.

Most general line simplification algorithms share the problem that they need additional measures to avoid self-intersections of the resulting line segments and that they often do not preserve the characteristic features of building footprints like orthogonality and dominating directions well.

For this reason, special algorithms for the simplification of building footprints have been developed. Staufenbiel (1973) and Sester (2005) present rule-based approaches; an agent-based approach is described in Lamy et al. (1999). In section 4.5, some more recent rule-based approaches for the simplification of building footprints are presented in the context of the simplification of LoD 1 building models.

The approach by Sester and Neidhart (2008) consists of two steps: In a first step, an initial set of line segments that approximate the original footprint is determined using the RANSAC principle. In a second step, the initial line segments are fitted more closely to the original line and parallelity and orthogonality relations between segments are emphasized in a least squares adjustment process.

The approach that is described in this section was first introduced in Guercke and Sester (2011). It is similar to the approach by Sester and Neidhart (2008): It also consists of an initial line segment generation and a refinement step based on least squares adjustment. Instead of RANSAC, the Hough transform is used to generate the initial line segment hypotheses in the new approach.

In order to increase the flexibility of the approach, line segment hypotheses may be generated, dropped or merged in the fitting process, and the assignment of parts of the original polygon to the different extracted lines may be changed. In the least squares adjustment step, the observation equations were also changed slightly compared to Sester and Neidhart (2008).

Note that although the problem of finding an optimal polygon approximating the original footprint – where there may be different definitions of *optimal* – seems to be a continuous optimization problem and therefore less difficult to solve than the obviously discrete problem of choosing the optimum subset of the vertices of the original footprint (which was shown to be NP-complete by Estkowski and Mitchell (2001)), the approximation problem is indeed probably even more complex for most sensible definitions of optimality because a more subtle combinatoric aspect is hidden in the association between the parts of the original polygon (in our case, the sampled points) and the approximating one (the extracted line segments) – especially if the approximating polygon is required to be free of self-intersections.

We leave the NP-hardness proofs and investigation of different quality measures to further research. In the context of this investigation, bad initial segments can be compensated in part by the iterative adjustment process in which the assignments of the sampled points to the extracted line segments and the relationships

between the extracted segments are updated heuristically between the iterations in order to increase the quality of the result.

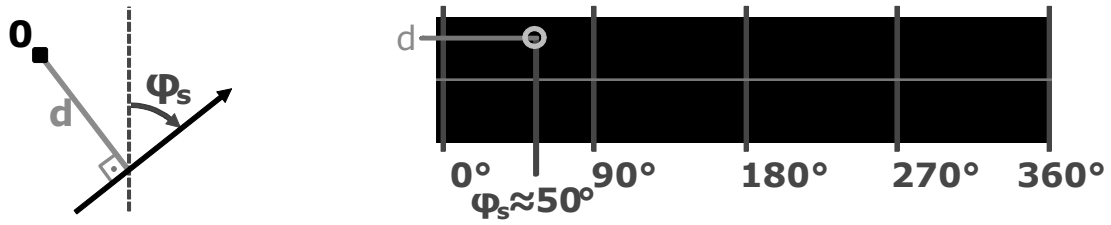
A special treatment of holes is not part of the current implementation, two ways of dealing with them in a topologically clean way are outlined in the results section (6.4).

6.1. Full-spectrum Hough transform

The Hough transform was chosen for generating the initial line hypotheses because it offers a means to identify the dominating directions in a set of points or line segments in the order of their weight – where the weight may be defined by the number of supporting points or the total length of the supporting line segments.

In its general form, the Hough transform can be used to identify shapes by transforming the raw data into the parameter space of the shapes to be detected. In practical implementations, this space is usually discretized, and all raw data points cast a "vote" for the cells representing parameter combinations that are consistent with them. After the voting of all raw data sets, the cells with the highest values are the parameterizations of the shapes with the strongest support in the raw data.

In our example, the shapes to be detected are straight lines in 2D. Such a straight line can be parameterized by an angle φ_s giving its direction and the closest distance d of the line to the origin. In the examples, a geographic definition of the angle φ is used (0° means that the line points in "North" or positive y direction, and the angle is measured in clockwise orientation), and the footprint polygons are given in clockwise order.



(a) Relevant parameters φ_s and d of the line segment.

(b) Resulting point in Hough space.

Abbildung 6.1.: A line segment voting as a single point in Hough space.

Given a footprint polygon, we have different options in our choice of how to treat the raw data. The most simple approach is to let each line segment in the footprint vote for the parameters of its associated straight line with a weight corresponding to its length. This procedure is illustrated in Figure 6.1: The direction φ_s of the line segment in question is about 50°, the distance d is the perpendicular distance of the infinite straight line (dashed line) defined by the segment (arrow) to the origin. Note that the position of the line segment on the line does not affect these parameters, so all line segments lying on the same straight line will vote for the same cell in the Hough accumulator.

In the case of highly detailed footprints from cadastre data in which the main directions are often present in the smaller parts, this approach sometimes succeeds in finding dominant directions. In most cases, however, it will fail – especially in the case of noisy data where the main directions have to be filtered out of jagged outlines like the one shown in figure 6.2.

In such a case, almost none of the original line segments points in the direction of the underlying main directions. Especially if a strong zig-zag pattern overlays the original shape, the directions of the line segments may differ by almost 90° from the main directions.

A more robust approach is to represent the footprint polygon by a set of regularly sampled points as shown in figure 6.2. The main directions of the original footprint are then approximated by the main directions of the point set by the traditional Hough Transform known from image analysis.

Figure 6.3 illustrates this process for a single point: Each straight line passing through the point p is defined by a unique combination of parameters φ and d. The perpendicular distance d of the straight line to the origin can directly be calculated as a function of the direction φ because the point p through which the line has to pass and the direction φ unambiguously define the line.

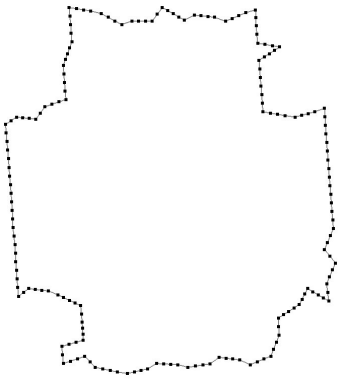
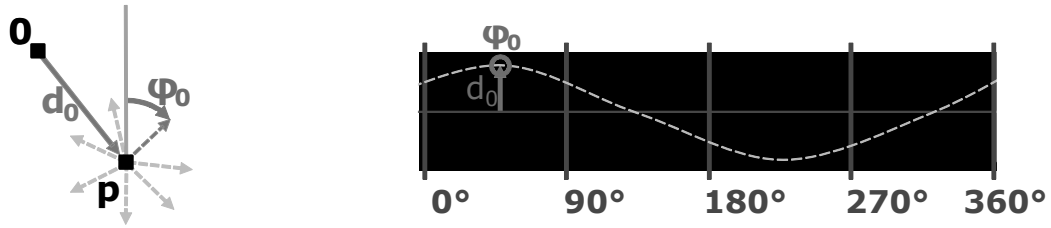


Abbildung 6.2.: Distribution of sampling points on an irregular footprint shape.



(a) A point votes for all lines passing through it.

(b) Resulting sine wave pattern in Hough space.

Abbildung 6.3.: A single point voting for a sine wave pattern in Hough space.

Since our buffer in parameter space is discrete, we can use this functional dependency to calculate the d value of the cell for which the segment will vote. Given the definition of φ_0 in the figure, we get $d(\varphi) = d_0 \cos(\varphi - \varphi_0)$ where d_0 is the distance $|p|$ of p to the origin and φ_0 the angle corresponding to the vector $(-y_p, x_p)^T$ pointing perpendicularly to the left of the position vector $p = (x_p, y_p)$.

The score of the cells in the Hough buffer is then the number of sampled points lying on the straight line parameterized by the φ and d values of the cell.

Note that in this case, we do not use the direction of the line segments but only the sampled points on the footprint polygon. Since the points have no inherent direction, the resulting spectrum will be anti-symmetric: $d(\varphi) = -d(-\varphi)$. In other words, we cannot determine the correct orientation for a straight line connecting a set of points.

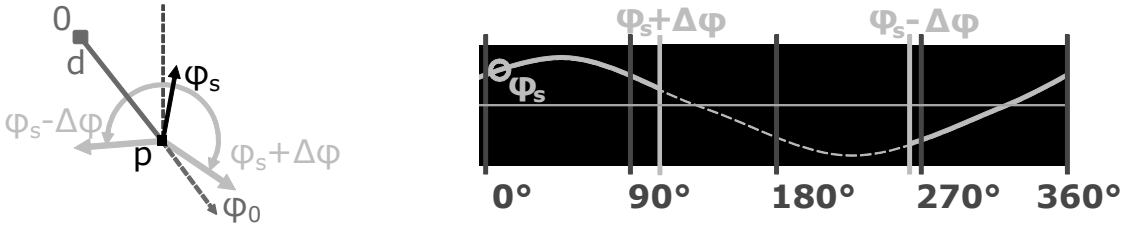
Because of this phenomenon, the spectrum of the angles is reduced to 180° in the traditional Hough Transform for image analysis: The pixels are single points, and no directions for the straight lines to be extracted are known a priori in most image analysis scenarios.

In our case, we do, however, have information on the direction in which a derived straight line may pass through a sampled point on the footprint: While we cannot expect it to be directly aligned with the direction of segment on which the sampled point p is located (in this case, we could simply have let the segment vote as a point in Hough space as described in the first scenario), we may assume that in order for p to vote for a straight line (φ, d), the angle φ should not differ excessively from φ_s, where φ_s is the direction of the segment on which p is located.

We define an angle tolerance Δ_φ and let each sampled point on the footprint vote for all lines with a direction of $(\varphi_s \pm \Delta_\varphi)$ as shown in figure 6.4. As 6.4 shows, calculating the actual values involves a case distinction to perform the periodic wrapping around $0° = 360°$.

In the figure, φ_s is 10° and Δ_φ is 110°. Due to the comparatively large value for Δ_φ, strong zig-zag or stairs patterns will vote for their main common direction as well as for the actual directions of the segments involved. Depending on the resolution (pixel size) R_φ and R_d of the Hough buffer, there will be a single maximum for the main direction of the zig-zag pattern or different maxima for the individual segments that form the pattern: The parameters Δ_φ, R_φ and R_d control the level of generalization employed in the first line extraction step.

In the experiments, the parameters of the Hough buffer were set to conservative values: $\Delta_\varphi = 120°$ ensures that

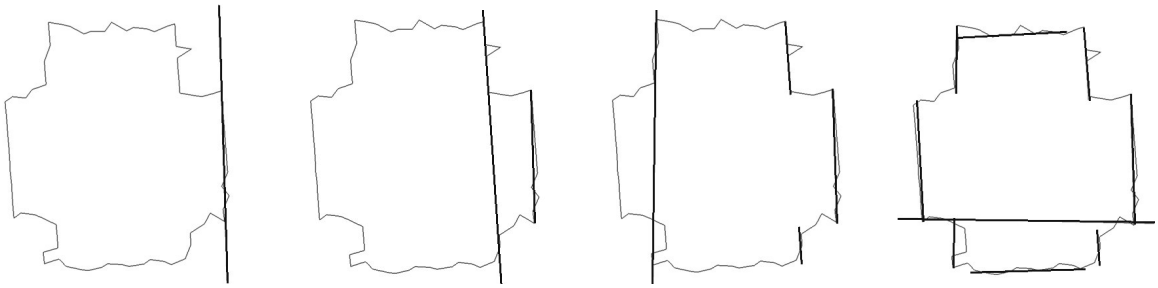


(a) Opening angle Δ_φ

(b) Windowed voting pattern.

Abbildung 6.4.: A short segment voting for a windowed sine wave pattern in Hough space.

zig-zag and stairs patterns can be approximated by their interpolating direction as well as the actual directions of the original edges. R_φ was also set to a small fixed value of 3°, and the distance resolution R_d and the sampling distance ϵ_p between the sampled points on the original outline were set to $0.1 \cdot \varrho$ (but not less that $10cm$ in order to avoid excessive memory demands for large scales).



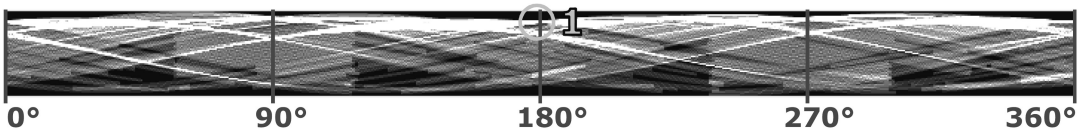
(a) After the first step.

(b) After the second step.

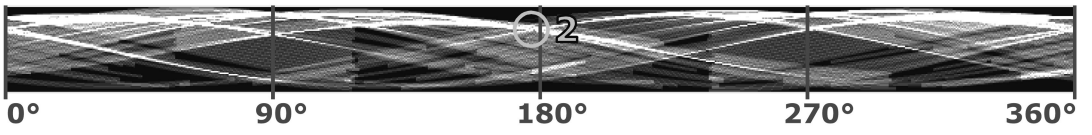
(c) After the third step.

(d) Final set of line segments.

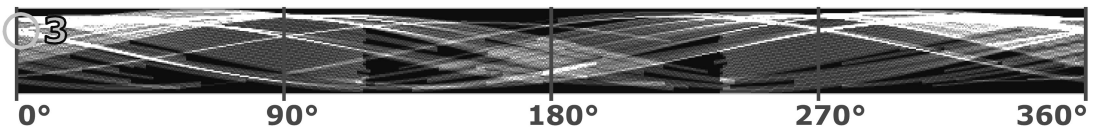
Abbildung 6.5.: Line segments generated in the initial Hough-based extraction process.



(a) Original Hough buffer.



(b) Hough buffer after first step.



(c) Hough buffer after second step.

Abbildung 6.6.: Hough buffer at different stages of the edge extraction process.

Figures 6.5 and 6.6 show the extracted line segments and the Hough buffer after the first line extraction steps. In order to avoid that a sampled point votes for multiple lines, each line extraction consists of the following phases:

1. All (directed) sampled points vote for their associated windowed sine pattern in the Hough accumulator.
2. The line l corresponding to the cell with the maximum value in the Hough buffer is selected as the next line hypothesis.
3. All sampled points within a distance of ϵ_{line} are associated with the line l and removed from the pool of

the sampled points.

4. Repeat until the last extracted line has an insufficient number of supporting points.

In phase 3, the extracted straight line is split into line segments. The straight line shown in figure 6.5(b), for example, was split into two segments as shown in figure 6.5(c). This is achieved by projecting the points associated with the extracted line onto this line in their order on the original polygon. If there is a gap of more than $d_{max,seg}$ between the projections of two consecutive points, then the current segment is closed and a new one is initialized.

Segments with lengths of less than l_{min} are discarded. In order to avoid that a straight line supported by a large number of very short segments stops the extraction process or causes an infinite loop by being extracted several times, the points associated with the extracted line can be left in the sample pool, but for the next run, the parameters of the "corrupt" line have to be added to a blacklist until the next line producing valid segments has been extracted. Since such a problem never occurred for the test data, this procedure was not included in the reference implementation.

For the experiments, the parameters l_{min} and ϵ_{line} are set to the target resolution ϱ, and $d_{max,seg}$ is set to $1.5 \cdot \varrho$ – in a pure Hausdorff interpretation, it could have been set up to $2 \cdot \varrho$ (because the points between the segment end points could be covered from both ends), but the slightly less strict setting reduced overshoots (see section 6.3) considerably.

The computational complexity of this approach is dominated by the size of the Hough buffer which is controlled by the resolution R_φ and R_d in parameter space. The dependency of the parameter R_φ on the global resolution ϱ is quite complex because it depends, among other factors, on the distance of a given line segment from the origin. Since fast computations were not the prominent goal in the tests, these values were fixed generously (high accuracy at the cost of additional computing ressoures needed) for all resolutions at 2° and 0.2m in the tests.

Fine-tuning these parameters may increase the quality of the initial set of segments, but since the results were of sufficient quality in the tests and the segments are adjusted to the data in the next step anyway, this task is left as a subject for further research – especially because there are still far more serious issues left that have to be resolved in order to use this approach in a productive environment.

Note that the size of the Hough buffer also depends on the maximum distance d_{max} of any point p on the outline of the footprint to the *origin of the coordinate system* because the value for the parameter d for which p will vote in the Hough buffer will vary between $+d_{max}$ and $-d_{max}$.

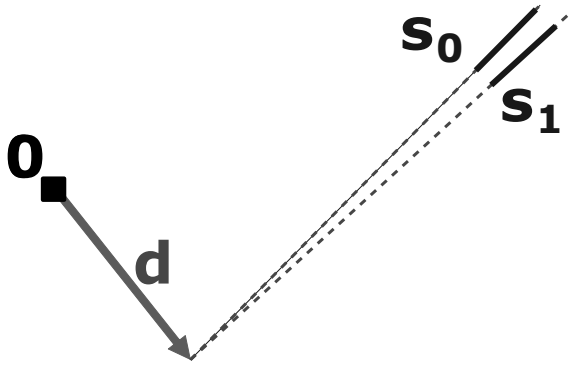


Abbildung 6.7.: Impact of small angle changes for objects far from the origin.

For this reason, it is not wise to use global coordinates directly for Hough analysis: In the case of Gauss-Krueger coordinates, for example, we would have to sample values in the order of millions of meters in a raster of R_d (usually in the order of a few decimeters) which would easily fill all available memory.

Additionally, a small variation of the angle φ has a huge impact on the resulting line in the target area if the target is far from the origin as shown figure 6.7. With an angular resolution of 3°, both segments s_0 and s_1 would, for example, be represented by the same cell in the Hough buffer. Due to this effect, the resolution R_φ would have to be increased as well, increasing the irrational memory requirements even further.

For this reason, the buildings were analyzed in a local metric coordinate system with the origin in the center of their bounding box. This kept the d_{max} values below $200m$ for all investigated buildings and yielded sufficient

angular and distance resolutions for all our tests. After the simplification, the resulting footprints can be transformed back to the original coordinate system.

6.2. Least squares adjustment to refine the line segments

After the initial line segments have been determined, an adjustment process is initialized to increase the quality of the extracted line segments. Since some of the observation equations in this context are not linear, an iterative process is used to fit the segments to the data. After a configurable number of iterations, the correspondences between the extracted line segments and the sampled points on the original segments as well as the relations between the segments are updated.

The current state of the adjustment process is given by a set of line segments s_i with line parameters a_i, b_i and d_i where a_i and b_i are the x and y components of the normalized normal vectors of the line, and d_i is its perpendicular distance to the origin (Hessian normal form of the line in the plane). For each line segment, the corresponding start and end point are are stored as well as a list S_i of the supporting sampled points associated with the line.

The most characteristic relations between segments forming a building footprint are parallelity and orthogonality. If two segments are almost parallel or orthogonal to each other, the algorithm aims to emphasize this relation by trying to force the segments to be exactly parallel or orthogonal. If two segments are (almost) parallel and have similar distances to the origin, then the algorithm will also try to align them to be perfectly collinear.

At each iteration of the least squares adjustment process, the following linearized observation equations are evaluated – the old values that are used as constants because of the linearization are marked by the subindex $_{old}$; the variables of the adjustment problem are bold:

1. The normal vectors have to be normalized:

   $$\forall i \in \text{segs} : \mathbf{a_i} \cdot a_{i,old} + \mathbf{b_i} \cdot b_{i,old} = 1$$

2. Fit the approximating line segments to the sampled points $p \in S_i$ associated with the current segment:

   $$\forall i \in \text{segs}, \forall p \in S_i : \mathbf{a_i} \cdot p_x + \mathbf{b_i} \cdot p_y + \mathbf{d_i} = 0$$

3. Emphasize the relation of perpendicular segments – the dot product of the normal vectors should become 0:

   $$\forall i, j \in S_\perp : \overrightarrow{n_{0,i}} \cdot \overrightarrow{n_{0,j}} = \mathbf{a_i} \cdot \mathbf{a_j} + \mathbf{b_i} \cdot \mathbf{b_j} \stackrel{!}{=} 0$$
   $$\stackrel{\text{linearized}}{\longrightarrow} \mathbf{a_i} \cdot a_{j,old} + \mathbf{a_j} \cdot a_{i,old} + \mathbf{b_i} \cdot b_{j,old} + \mathbf{b_j} \cdot b_{i,old} \stackrel{!}{=} 0 \stackrel{!}{=} 0$$

   where $S_\perp$ is the set of perpendicular segments.

4. Emphasize the relation of parallel segments – the length of the cross product vector of the normal vectors should become 0 (the z component of the normal vectors in the plane is 0):

   $$\forall i, j \in S_\parallel : \|\overrightarrow{n_{0,i}} \times \overrightarrow{n_{0,j}}\|^2 = \mathbf{a_i} \cdot \mathbf{a_j} - \mathbf{b_i} \cdot \mathbf{b_j} \stackrel{!}{=} 0$$
   $$\stackrel{\text{linearized}}{\longrightarrow} \mathbf{a_i} \cdot a_{j,old} + \mathbf{a_j} \cdot a_{i,old} - \mathbf{b_i} \cdot b_{j,old} - \mathbf{b_j} \cdot b_{i,old} \stackrel{!}{=} 0$$

   where $S_\parallel$ is the set of parallel segments. If segments i and j are neighbors and sufficiently close, we introduce an additional observation to bring them even closer in order to merge them if possible:

   $$\forall i, j \in S_{\parallel,close} : \mathbf{d_i} = \mathbf{d_j}$$

   This observation assumes that segments i and j are sufficiently close to parallel and that the segments are located close to the origin.

5. In order to alleviate the conflicts between segment fitting and emphasis of relations, additional observations are added to shift the resulting line segments in order to better fit the points without changing their

direction:

$$\forall i \in \text{segs}, \forall p \in S_i : -\mathbf{d_i} = p_x \cdot a_{i,old} + b_{i,old} \cdot p_y$$

The first observation is rather a hard constraint. For this reason, it receives a high weight of 10^8 in the experiments. With other weights increasing with the number of iteration, the value may have to be increased as well in order to make sure that this observation always takes precedence over the other observations.

In the first phase of the adjustment schedule, the aim is to make the line segments "snuggle" more tightly to the original polygon – due to the discrete nature of the Hough buffer, the extracted line segments may deviate from the best-fitting line by $\pm R_\varphi$ in the angle and by $\pm R_d$ in the orthogonal distance to the origin. For this reason, the set of observations (2) that represents this aim receives a constant weight for all iterations.

In the later stages of the adjustment, the relations between the segments receive increasing weights in order to make sure that they are well preserved in the final result. Therefore, the observations (3-5) that are used to emphasize the orthogonality, parallelity and collinearity relations between segments receive increasing weights with with the number of iterations.

This adjustment towards emphasized relations, however, means that segments have to be rotated which, in turn, means that they will not fit their associated points optimally, and using observation (2) to improve the fitting of the points would mean that the orientation change due to the relationships of the segments would be (partially) undone. For this reason, observation (5) is used to improve the fitting of the segments to the sampled points without changing the direction of the segments: Only the offset is changed in the application of this observation, and the line segment is shifted in the direction of its normal. The weight of observation (5) is increased with number of iterations along with the weight of the relationship observations to support the emphasis on the relations compared to the adjustment of the directions of the segments to the data.

In the split-and-merge step, the correspondences between the sampled points and the approximating line segments and the relations between the segments are updated. This step is conducted after every k^{th} iteration of the adjustment process. In the experiments, k was set to 3 in order to give the adjustment process the opportunity to "settle down" after the possible perturbations caused by the change of the support sets of the line segments and due to the new and deleted line segments.

In order to determine the predecessors and successors of a line segment, the sampled points on the outline are traversed and the line segments with which they are associated are added to a list in the order in which they are encountered. Especially in corners, the resulting list of the segments will often contain alternating sequences of the same pair of segments. Such alternating sequences are removed from the list. This *segment sequence building* process is performed in the course of the split-and-merge steps and before the first adjustment process is started.

The parallel and perpendicular relations between the segments are refreshed after each recalculation of the order of the line segment hypotheses. If two succeeding segments are almost perpendicular or parallel, then the weight of the relation in the adjustment process is increased in order to emphasize relations between adjacent segments. If the distance between two line segments is greater than a threshold, then the relation is not considered in order to avoid that small segments in different parts of the building distort its overall shape. Another useful feature would be to scale the weight of the relations with the length of the segments in order to emphasize relations between long segments. This is, however, not included in the current implementation yet.

The first part of the split-and-merge step is to refresh the mappings from the sampled points on the outline to the line hypotheses. First, the interaction with the pool of the *unclaimed* samples that could not be associated with a line segment in the previous step is performed: If the distance from a point in the support set of the current segment to the segment has increased beyond the distance threshold ϱ, then it is released back to the unclaimed pool. If the distance from an unclaimed sample to the current segment is lower than ϱ, then it is removed from the unclaimed pool and added to the support of the current segment.

Due to the release of sampled points from the support sets of line segments to the pool of unclaimed samples, it may be possible to detect new line segments of sufficient length. For this reason, the Hough-based initial segment extraction step described in the preceding section is now applied again to the samples in the pool of unclaimed samples. If new segments were generated in this process, they are integrated into the segment order by running the *segment sequence building* process described above.

If two succeeding line segments are close enough and almost parallel, they can be merged. The resulting segment is formed by a weighted average of the parameter values of the segments to be merged in which the weight is

the length of the segments. If the parameters were determined by a least squares adjustment of the sampled points associated with the segments to be merged, then the effects of the previous effort to enforce the relations between the segments can be undone. For this reason, no least squares adjustment is performed between the main adjustment steps.

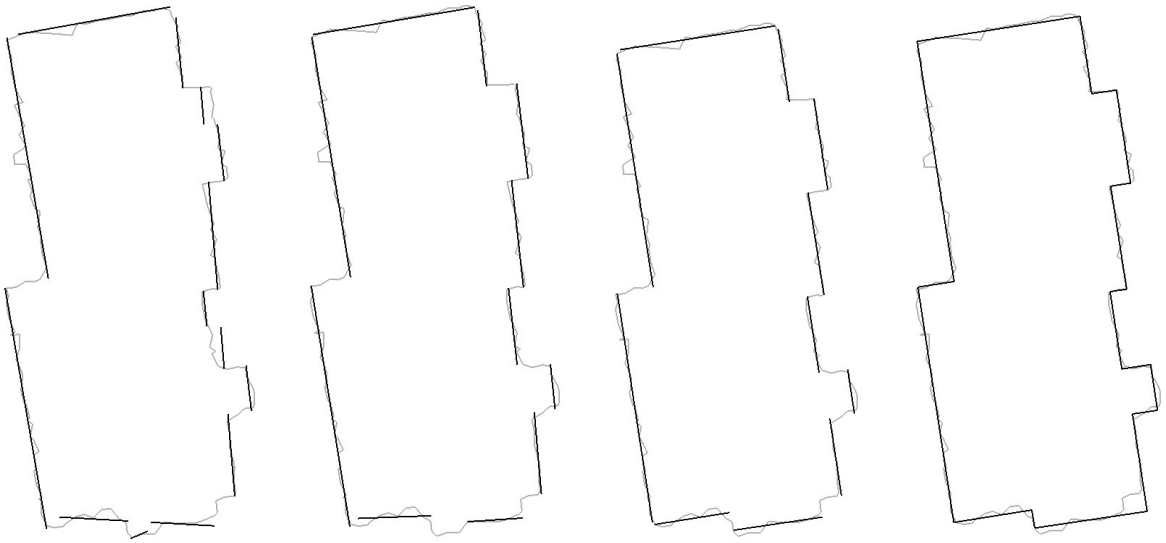
After this interaction with the pool of unclaimed samples has been performed, samples can be reassigned from one segment to another if the distance to the other segment is lower than the distance to its current segment. In order to keep the computational complexity at a reasonable level, the sampled points are tested only in relation to the predecessor and successor of the line with which they were associated before. If the sampled point is closer to the predecessor or successor of the current segment, it is removed from the support set of its current segment and added to that of the segment to which it has the minimum point-line distance.

After this remapping of the samples, the start and end points of the segments are updated by projecting the supporting points on the line associated with the segment. If there are gaps in the projection of the points on the line, the segment will have to be split as in phase 2 of the initial extraction algorithm presented in the preceding section.

If a segment becomes too short in this process, it is deleted and removed from the segment list, and its supporting samples are returned to the pool of unclaimed samples. A shorter segment running close and parallel to a longer one that covers it completely is also deleted, and its supporting samples are transferred to the longer segment if they are close enough to it. The deleted line segments are simply removed from the sequence of the line segments. After segments are merged or deleted, a new support swapping process may be started for the neighbors of the affected segments.

Especially after segments were deleted, new samples will appear in the pool of unclaimed samples. These samples may allow us to detect new line segments in the pool of unclaimed samples, and we can again look for them by applying the Hough-based extraction algorithm and starting a new iteration of the split-and-merge process until no more segments are deleted and no new segments are detected. In the current implementation, only one iteration of the split-and-merge process was performed.

In the experiments, a fixed number of 20 adjustment iterations was performed. This value was determined heuristically by testing the process manually for the most critical (i.e. the largest and most complex) and several randomly chosen buildings. After 15 iterations, no more segments were created or deleted in almost all cases, and the fitting of the segments to the original data did not improve noticeably with further iterations. To be on the safe side, 20 iterations were used in the tests for the whole data sets.



(a) Original strokes. (b) After 3 iterations. (c) After 12 iterations. (d) Final result.

Abbildung 6.8.: Least squares refinement of the segments.

Figure 6.8 illustrates the adjustment process for an example: After the first three adjustment iterations, the segments are better fitted to the data points, and after the first split-and-merge step, three segments could be merged or removed from the list (Figure 6.8(b)). After twelve iterations, we observe that the relations between the segments are more pronounced than after the first iterations, and that the segments still fit well with the

data points. This was achieved in part by introducing the parallel shifting observation (5) and the increasing weight of this observation.

After all adjustment steps have been performed, the line segments will usually not form a closed polygon. For this reason, they have to be "stitched" together in order to form a valid footprint. In the experiments, this was done using simple deterministic set of rules.

(a) Intersection. (b) Parallel segments. (c) Perpendicular segments. (d) Default case.

Abbildung 6.9.: Rules for linking the line segments to form a closed footprint polygon.

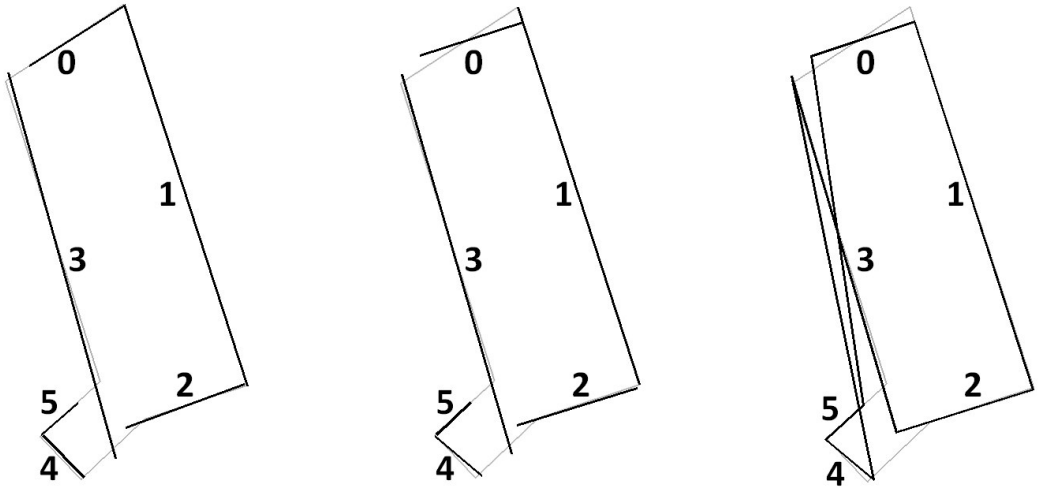
Figure 6.9 illustrates these rules. They are applied to consecutive segments in the segment sequence and tested in the order defined in the figure:

1. If the intersection point of the segments is closer than ϱ to the end point of the first and the starting point of the second segment, the segments are simply extended to meet in their intersection point (Fig 6.9(a)).
2. If the segments are parallel, then a new perpendicular (to both) segment is created halfway between the end point of the first and the starting point of the second segment (Fig 6.9(b)).
3. If the segments are perpendicular, then an inward-facing corner is inserted between the end of the and the start of the second. The rationale between this solution is that if there had been a regular corner, they would have directly and rule (1) would have applied ((Fig 6.9(c)).
4. In all cases (skewed segments) the end of the first and the start of the second segments are simply connected by a new segment (Fig 6.9(d)).

This simple heuristic approach produced valid and sensible results for the vast majority of building footprints in the test datasets. In figure 6.8, we can see that the final stitching produced a convincing closing of the footprint polygon (in this well-behaved example, only rules (1) and (2) were applied).

6.3. Main Issues

Almost all topological errors in the test runs could be traced back to problems in the segment ordering procedure. One of the most prominent of these problems is, especially at small scales, the "overshoot" effect.



(a) Original strokes. (b) After last iteration. (c) Final result.

Abbildung 6.10.: Overshoot

Figure 6.10 illustrates the structure of an overshoot: A segment stretches through the footprint and holds support samples on another segment on the other side of the building. In the original segment ordering list, such a segment will appear twice (or more often due to alternating association of samples, but this case is filtered out effectively).

In the example in figure 6.10, segment 3 obviously is the culprit: Its overshooting part will be registered after segment 2 and its main part after segment 5, so the segment list after the cleaning of alternating segments would be 0,1,2,**3**,4,5,**3**. In the current implementation, the first instance of the two appearances of segment 3 is kept. For this reason, the strange final polygon shown in 6.10(c) is the logical result of stitching the sequence 0,1,2,**3**,4,5: segments 3 and 4 and segments 5 and 0 are stitched together using stitching rule no. 4 (connect end points).

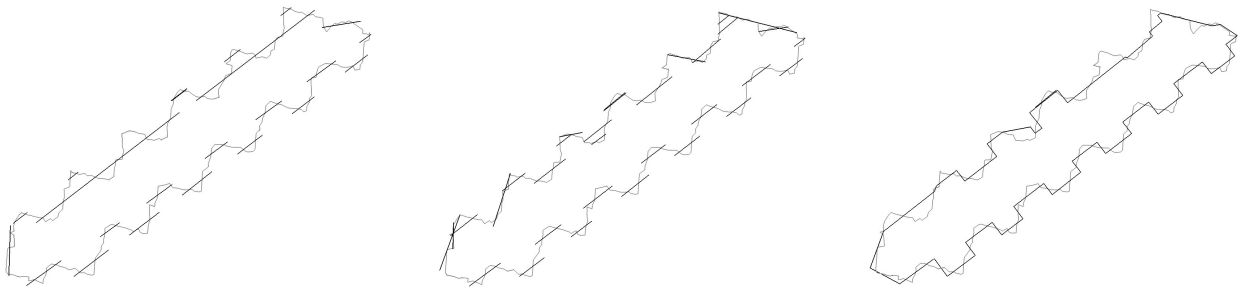
In the case of this example, the reason for the overshoot was that the link of the structure attached to the corner of the building was too narrow. A reason why the segment could find the support on the opposite side of the polygon is the fact that in the current version of the algorithm, the direction of the original polygon in the sampled points is considered only in the filling of the Hough buffer but not in the process of collecting the supporting samples for a line segment.

(a) Original strokes. (b) Final result.

Abbildung 6.11.: A small building at small scales.

The building in figure 6.11 is just "too small to live" at the low resolution of $2.5m$ used in the example: The sides of the polygon are so close that the first line hypothesis covered all samples on both sides for all three main parts of the building. Taking the direction of the original polygon in the samples into account would have solved this problem, but the problem only occurs if the sides are closer than the resolution, and the affected part would be a candidate for being removed.

Note that after the adjustment process, the two segments at the bottom are close to parallel. For this reason, stitching rule no. 2 is applied and the segments are connected by a perpendicular line halfway between the end of the first and the start of the second segment. For this reason, the second (upper) segment is shortened.



(a) Original strokes. (b) After last iteration. (c) Final result.

Abbildung 6.12.: Different mistakes as a result of a problematic stairs pattern interpretation.

Figure 6.12 illustrates that stairs patterns can sometimes cause severe problems for the approach. The root of these problems is the inherent ambiguity of such a stairs pattern: The main directions of the overall footprint may either be aligned with the directions of the edges forming the stairs or with the direction of the regression line of the pattern. The choice of the best-fitting direction depends on the resolution and on the main directions of the rest of the footprint: If we have a very high resolution, then the simplified lines are more likely to follow

the original lines; if significant parts of the pattern can be covered by a single line (with its ϱ-buffer), then this line will have a high score in the Hough accumulator and a high probability of being selected for generating line segment hypotheses.

In the figure, the different parts of the stairs were small enough to be crossed by valid line segments aligned with the regression direction. These segments, however, did not cover the whole of the pattern, so additional parallel segments were generated where enough samples were left uncovered. This leads to a "dangling parallels" effect: There are short segments running parallel to a long one that covers their whole extent in the projection on the main direction and appears before and after the small segment in the segment order.

In the current implementation, these segments are only removed if they are closer that ϱ to the covering long segment. Especially if the covered smaller segments have significant lengths, an alternative approach would be to split the longer segments by cutting out the projection of the shorter one. In this case, one would, however, have to ensure that the remaining parts of the longer segments still remain longer than l_{min}. The situation in the lower part of the polygon is similar the effect of such an approach: The resulting pattern will be a rectangular zig-zag pattern following the regression direction.

In the final set of segments shown in figure 6.12(b), we can also observe mixed interpretations of the stairs pattern: There segments in the simplified version that are aligned with the original ones and segments that are aligned with the main direction of the pattern. In order to avoid such a situation, we can try to identify the stairs patterns as early as possible: In the Hough buffer, for example, the original segments forming the stairs pattern show as two parallel columns of peaks.

6.4. Results

For the evaluation of the simplification approach, the cadastre (ALK) data sets of Dortmund and Hanover introduced for the LoD 1 aggregation problem were used. Another data set consisting of raw footprints derived from laser scanner data from an aerial survey of a part of the city of Hanover was used to test if the approach could handle very irregular footprints. The irregular footprints used in most figures in the preceding section were taken from this data set.

In most cases, the approach could handle both data sets equally well, but some major issues remain that have to be solved in order to consider using the approach in a productive environment. Some of these issues have been discussed in the previous section (6.3).

Figure 6.13 illustrates the result of the algorithm at different scales for a part of the Hanover data set. The buildings were simplified independently without checking for intersections; especially in the 2.5m version, several examples of intersections can be found. In the top right corner, the building used for the "overshoot" example from section 6.3 (figure 6.10) can be found.

Since the algorithm processes the building outlines of the buildings independently, there may be overlaps between buildings in the result and direct (wall-to-wall) adjacency relations may be lost.

Additionally, the quantization error of the Hough transform is sometimes not completely alleviated by the adjustment process: Some generalized buildings are rotated by a few degrees compared to the best orientation one would intuitively have expected and sometimes the generalized segments are shifted a little inwards compared to their expected locations. Again, these effects are more pronounced in the smaller scale (2.5m) version.

The tables in this section illustrate the average data reduction achieved by the algorithm for the different data sets. In order to get a less biased view of the performance of the algorithm, the buildings were classified according to the number of segments in the footprints, and each class was evaluated separately.

Especially in the cadastre data sets, buildings often tend to have regular shapes. If we start with a rectangular building, we can hardly expect a simplification algorithm to come up with a sensible simplified version of such a building containing a lower number of segments. For this reason, these *trivial* buildings were counted but the algorithm was not evaluated for them.

The classification into *small*, *medium* and *large* footprints (concerning the number of segments) shows that with an increasing number of segments, an increasing relative reduction could be achieved. This is a sensible result because an increased geometric complexity is often the result of comparatively small features like additions or steps to the ground floor.

Abbildung 6.13.: Overlay of original ALK footprints and generalized version at $\varrho = 1.0m$ and $\varrho = 2.5m$ for a part of the Hanover data set.

ALK Hanover, 23662 buildings	original buildings	0.7m 187 Errors: errors	0.7m 187 Errors: avg. reduction	1.0m 274 Errors: errors	1.0m 274 Errors: avg. reduction	2.5m 2482 Errors: errors	2.5m 2482 Errors: avg. reduction
trivial (≤ 4)	5877	36	–	165	–	1565	–
simple (≤ 8)	8125	40	21%	56	25%	798	32.5%
intermediate (≤ 15)	6477	52	28%	32	35.5%	85	55%
large (≤ 50)	2643	38	34%	16	42%	14	66%
arc (> 50)	540	21	77.5%	5	81%	20	88%

Tabelle 6.1.: Reduction of line segments achieved at different resolutions for the Hanover ALK data set.

A special case in the cadastre data sets are footprints with *arcs*. These arc sections are often approximated by huge numbers of segments (often about 50 for a quarter of a circle) in the cadastre data, and they can usually be reduced to just a handful (depending on the resolution, but often 5 segments are sufficient) of segments within the desired accuracy.

Since the algorithm will achieve very high compression rates at little cost in such a situation, buildings with more than 50 segments form a separate class. Several milder cases of such an oversampling of arcs will also be found in the *medium* and *large* classes. In order to cleanly separate these instances from the rest, an explicit arc recognition process would be necessary.

		0.7m 155 Errors		1.0m 91 Errors		2.5m 270 Errors	
ALK Dortmund, 56827 buildings	original buildings	errors	avg. reduction	errors	avg. reduction	errors	avg. reduction
trivial (≤ 4)	0	0	–	0	–	0	–
simple (≤ 8)	45713	61	22.5%	46	23%	226	25%
intermediate (≤ 15)	9793	75	25%	35	32%	41	51%
large (≤ 50)	1273	13	27%	8	37%	2	62%
arc (> 50)	47	6	39%	2	49%	1	70%

Tabelle 6.2.: Reduction of line segments achieved at different resolutions for the Dortmund ALK data set.

In the Dortmund data set, the buildings were already preprocessed for a noise simulation application. As table shows, the arc oversampling effect is far less pronounced in the Dortmund than in the "raw" Hanover ALK data set, so we can assume that those oversampled arcs were already thinned out in the preprocessing step or that the arcs were approximated with a lower number of segments in the surveying workflow. This is also an explanation for the generally lower compression rates for the Dortmund data set for buildings with complex footprints.

In some cases, the topological errors were too grave to construct a sensible result. For this reason, the total number of buildings for each class is given in the first column, and the number of errors is included in the statistics for the different resolutions. Most of problems of the algorithm will usually not make the process fail, so several strange results – including (self-) intersections – will be left undetected.

Table 6.1 shows the results for the cadastre data set from the city of Hanover that was also used in section 7.3. We can see that the compression rate increases with the maximum allowed error bound ϱ. At the smallest scale with an allowed error bound of $2.5m$, some of the smaller buildings are sampled in such a coarse way that they almost fall through the grid, leading to an increased number of errors and peculiar results.

Sometimes, the footprints are too small to be processed by the algorithm because even the main sides of the buildings are too short to be recognized in the initial extraction process, or only two sides of the building can be extracted. Especially smaller buildings (which will usually have a low number of segments in their footprint) are affected by this problem; this explains the high error rates for the trivial and small buildings for the small scale ($2.5m$) generalization process.

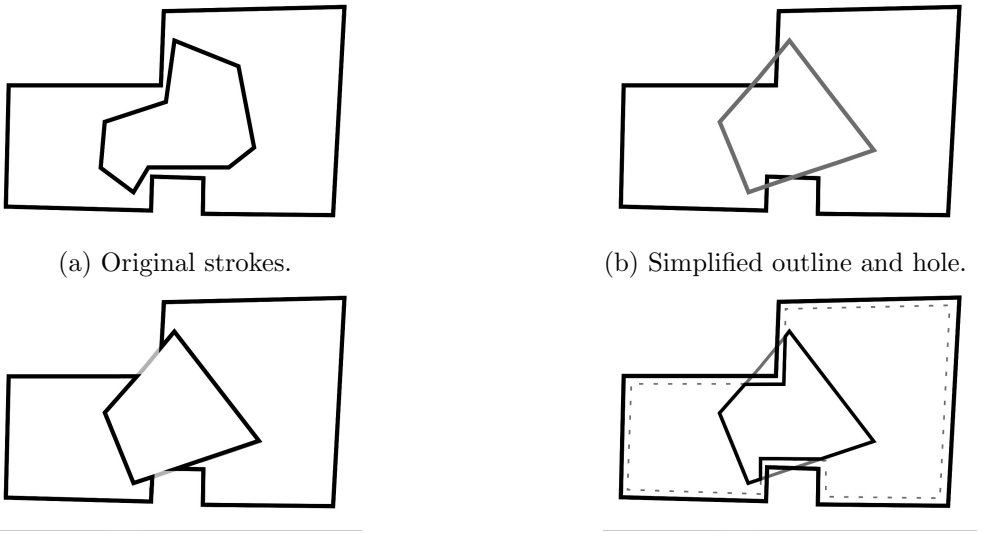
		0.7m 45 Errors		1.0m 28 Errors		2.5m 45 Errors	
	original buildings	errors	avg. reduction	errors	avg. reduction	errors	avg. reduction
large (≤ 50)	545	13	83%	15	86%	39	87%
arc (> 50)	1012	32	90%	13	92%	6	94%

Tabelle 6.3.: Reduction of line segments achieved at different resolutions for the Hanover Lidar data set.

For the buildings in the Hanover Lidar data set, the same classification was used. Due to the irregularity of the footprints, there were no *trivial*, *simple*, or *intermediate* buildings in the data set; all buildings had more than 15 (*large*) or even more than 50 (*arc*) segments. Since the sampled points are distributed in a comparatively even way on the footprint (no oversampling of arcs) in the Lidar data set, especially the labeling of the *arc* classification is misleading in this case.

Table 6.3 shows that most of the redundancies (accounting for more than 80% of the edges) introduced by the

irregular outlines of the footprints extracted from the Lidar data could be removed even at the largest scale ($\varrho = 0.7m$) for which the algorithm was evaluated.



(a) Original strokes.

(b) Simplified outline and hole.

(c) Simple set difference.

(d) Holes are clipped by a buffer around the outline to keep the polygon closed.

Abbildung 6.14.: Intersections of holes and outline.

In the current implementation, holes in the footprint polygons are not covered. A simple but promising approach is to simplify the outline and the holes separately and resolve the possible topologic problems later using set operations. Figure 6.14 shows the two basic options:

- The most straightforward approach is to subtract (set difference) the union (set addition) of the simplified hole outlines from the simplified main polygon.

  $$\text{Result} = P_{\text{smp}} \setminus \bigcup_{h \in \text{H}_{\text{smp}}} h,$$

  where P_{smp} is the simplified main polygon and H_{smp} is the set of the simplified holes. In this case, the original polygon may be split into two or more disconnected regions (fig. 6.14(c)).

- In order to avoid this effect, we can use an alternative approach similar to the morphologic operators (Serra, 1983) of dilation or closing: before the holes are subtracted from the main polygon, they are clipped by a buffer (e.g. of size ϱ) around the boundary of the main polygon. This ensures that the outline of the main simplified polygon remains intact. Formally, we can write this as:

  $$\text{Result} = P_{\text{smp}} \setminus \left(\bigcup_{h \in \text{H}_{\text{smp}}} (h) \setminus \text{Bf}\left(\partial P_{\text{smp}}, \varrho\right) \right),$$

  where ∂P_{smp} is the boundary of the simplified main polygon and $\text{Bf}\left(X, \varrho\right)$ is the polygon defined by a buffer of size ϱ around object X.

While these approaches may not be optimal for all applications, they are guaranteed to produce topologically correct (multi-)polygons without self-intersections if the simplified holes and outer loop are free of self-intersections and the main polygon is bordered by a single closed loop. They are also both (in themselves) valid in the sense of a Hausdorff interpretation because the result is, by definition, included in a ϱ-buffer of the input and vice versa. In combination with the original simplification, differences of $2 \cdot \varrho$ may occur if both the simplification and the hole integration algorithms shift a point in the same direction.

6.5. Summary and Discussion

In this chapter, an iterative approach for the simplification of building footprints was presented that consists of an initial phase of generating a set of line segments and a second phase of fitting the line segments to the original outline.

In the process of generating the initial line segments, a modified version of the Hough transform is used that takes the direction of the segments into account. In order to increase the quality of the resulting line segments, a more detailed analysis of the Hough buffer can be considered in future versions of this step: In order to stress orthogonality and parallelity relations, those relations can be taken into account in the search for the cells with maximum support in the Hough buffer.

Another necessary step towards more reliable results that can be taken in the initial line extraction process is to consider multiple maxima in the Hough buffer when a new line is generated: If the support of the line corresponding to the maximum in the Hough buffer turns out to be very sketchy and a more coherent alternative line exists with a slightly lower Hough score, then this line should be selected. Additionally, the presence of many parallel and perpendicular lines in the data can increase the score of a given line segment candidate.

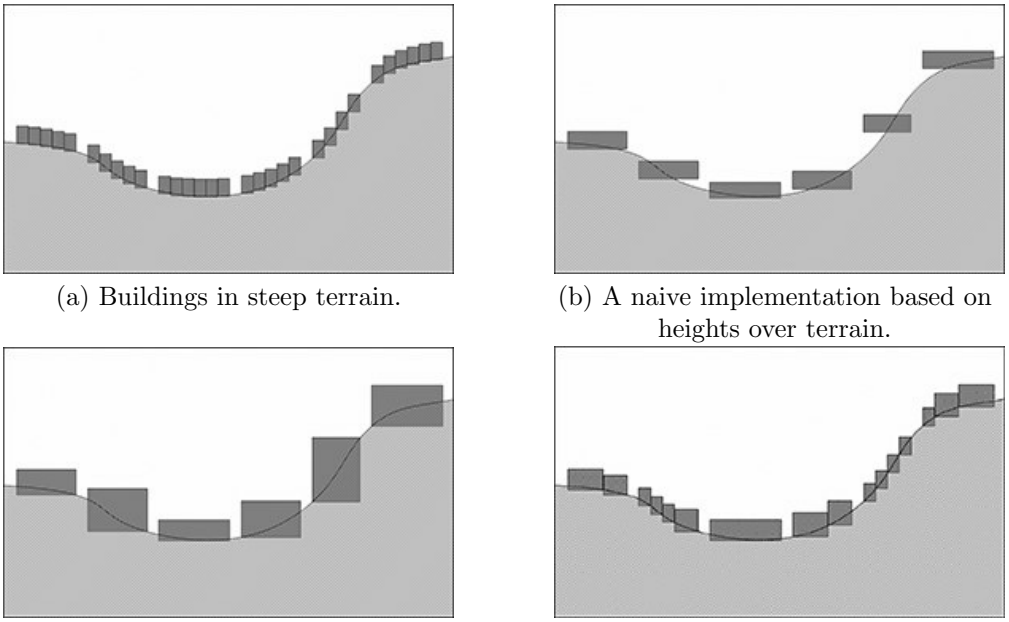
In the second phase, the line segments are aligned with the original outline in an iterative least squares adjustment process. With increasing numbers of iterations, the relations between the segments are weighted more strongly compared to the fitting of the lines to the points on the original outline in the adjustment process in order to emphasize the relations. In order to reduce line segment flipping effects, the segments are fit to the data points by parallel shifts rather than by rotating them to fit the data points with increasing numbers of iterations.

The results obtained using this approach are promising, but in order to use it in a productive environment, it would be necessary to include self-intersection tests in the line fitting process in order to ensure that the result will be a valid polygon without self-intersections.

The quality of the results can also be increased by a more graceful handling of stairs patterns in the original data and in the result – sometimes adjacent parallel lines with small offsets are not merged in the final result which produces two unnecessary line segments.

7. Aggregation of LoD 1 Building Models

If adjacent buildings are sufficiently similar, the complexity of a data set can often be reduced by merging them into a single aggregated building. Especially at small scales, there is a lot of potential for aggregation because with an increasing simplification of the models and the corresponding reduction of potentially conflicting details in adjacent buildings, the probability of a sufficient similarity between adjacent buildings increases dramatically. At LoD 1, finally, the height is almost the only geometric property left that could prevent the aggregation of adjacent buildings.



(a) Buildings in steep terrain.

(b) A naive implementation based on heights over terrain.

(c) Applying corrections for influence of terrain after aggregation.

(d) Aggregation taking terrain into account.

Abbildung 7.1.: Aggregation in steep terrain (from Götzelmann et al. (2009)).

When individual buildings are simplified, it makes sense to measure the height $H_t(b)$ of a building b above the terrain for calculations. When it comes to the aggregation of different buildings, this can be a problem, especially in steep terrain as illustrated in Figure 7.1.

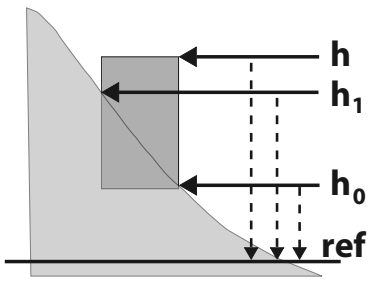


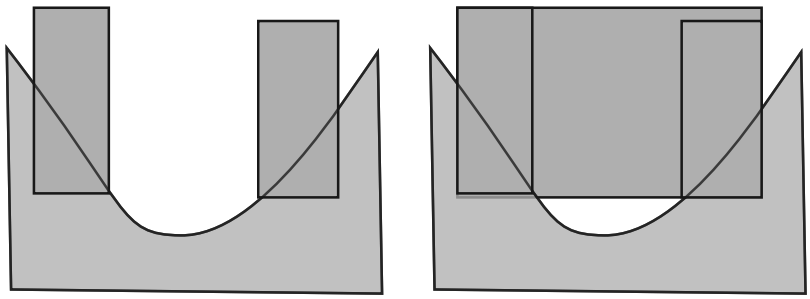
Abbildung 7.2.: Absolute heights.

For this reason, we define the following absolute height levels $h(b)$, $h_0(b)$, and $h_1(b)$ shown in Figure 7.3 for each building b with respect to a reference surface S (e.g. the WGS84 ellipsoid):

- $h(b)$ – the absolute height of the roof surface of b above S,
- $h_0(b)$ – the minimum absolute height of the intersection of b with the terrain relative to S,

- $h_1(b)$ – the maximum absolute height of the intersection of b with the terrain relative to S.

Using absolute heights makes it easy to compare the height levels of adjacent buildings in flat as well as in steep terrain. If we have a target resolution ϱ, a Hausdorff interpretation says that a building may only be part of an aggregated building (also called *cluster* in the following) if the absolute height of the cluster after the aggregation does not differ by more than ϱ from the absolute height $h(b)$ of b. This means that no two buildings can be part of the same cluster that have a height difference of more than $\Delta_h = 2\varrho$.



(a) The original buildings.

(b) Aggregated building: A gap may occur if no additional measures are taken.

Abbildung 7.3.: Aggregation across a hole.

Without taking other possible restrictions into account for the moment, we can define a very basic building aggregation problem: Given a neighborhood graph G, find a minimum set of clusters with $|h(b_i) - h(b_j)| \leq \Delta_h$ for all pairs of buildings b_i and b_j that are part of the same cluster. In appendix A, a detailed proof is presented that this problem is NP-hard even if G is planar and we have only three initial height levels and a fixed Δ_h.

For this reason, it is improbable that there is an algorithm that finds the exact optimum for any input in a reasonable amount of time where a "reasonable amount of time" refers to the notion of theoretical computer science that considers any runtime acceptable that is bounded by a polynomial in the size of the input.

In this section, a MIP-based optimizing and two heuristic approaches are introduced to solve this problem. In order to evaluate the optimizing and the heuristic approaches, they were tested on two data sets consisting of building footprints from the German cadastre data sets (*Automatisierte Liegenschaftskarte*, ALK) with building heights derived from laser scans. The first data set contains buildings from a part of the city of Hanover, Germany, the second one covers a part of the city of Dortmund.

Fortunately, in most cities, the buildings are arranged in blocks separated by streets with a width of some meters. Each block usually consists of a moderate number of buildings – in the Hanover data set, the largest block consists of 52 buildings. The vast majority of the blocks even in densely built-up urban areas has less than 20 buildings.

For those comparatively small numbers of buildings, it is in most cases possible to find an optimum solution for a block within some seconds by translating the problem into a linear (or quadratic) *mixed integer programming (MIP)* problem that is then solved by a special MIP solver software. In section 2.5, a more comprehensive overview of optimization problems and MIP is given. In the experiments, the CPLEX (IBM ILOG, 2011) software was used, a very powerful MIP solver developed by ILOG (now part of IBM) that is available for free for academic use.

The advantage of this approach is that it allows us to benefit from the experience of the developers of the software. Since optimization is a commercially most attractive domain, the pace of innovation and fine-tuning of the software is remarkable: While in the first experiments using CPLEX 10, calculations for the more complex blocks sometimes terminated after several hours because of having used up all available memory, the same calculations were finished within seconds using CPLEX 12.

Even using the most recent version of CPLEX, however, some blocks still need an infeasible amount of time for critical combinations of parameters. For this reason, it still makes sense to develop heuristic approaches to generate acceptable solutions with comparatively little effort. Additionally, the solutions generated by a heuristic approach can be used to "warm-start" a MIP solver like CPLEX which may speed up the optimization process.

Instead of pitting optimizing and heuristic approaches against each other in a competition, both ways to solve the problem can support each other – ideally making use of the advantages of both: A heuristic approach can

usually provide a good solution fast while an optimizing approach will provide the best solution (to the specified problem) but may need an infeasible amount of resources to do so.

Another example of such a combination is to use an optimizing approach to produce a benchmark for a heuristic one: If the heuristic approach produces good results compared to an optimum solution generated by an optimizing approach for a data set containing critical cases, then it can be used with an increased level of confidence for the generalization of a large data set if an optimizing approach would take too many resources for processing the whole data set. In another scenario, a result of a heuristic approach could be used for a block if an optimizing default approach exceeded a certain critical amount of resources.

Additionally, optimizing approaches can be used to evaluate if a certain optimization function adequately represents the intention of the aggregation by producing optimal results for some representative data sets: They show what the most consequent application of a given set of constraints and (weighted) optimization goals would produce. These results could then be used to evaluate different sets of of optimization criteria – for example, in a user survey.

Processing a complete data set at once is neither feasible nor necessary in most cases because, as we have mentioned, almost all cities are divided into blocks of a manageable size – usually containing less than 20 (in the worst case around 60) buildings. In order to identify the blocks in data sets of buildings, either the cells of the road network or a buffer operation may be used. Especially in the context navigation devices, using the road network is a sensible option, because for this application, buildings must not be merged across a road even if it is narrow.

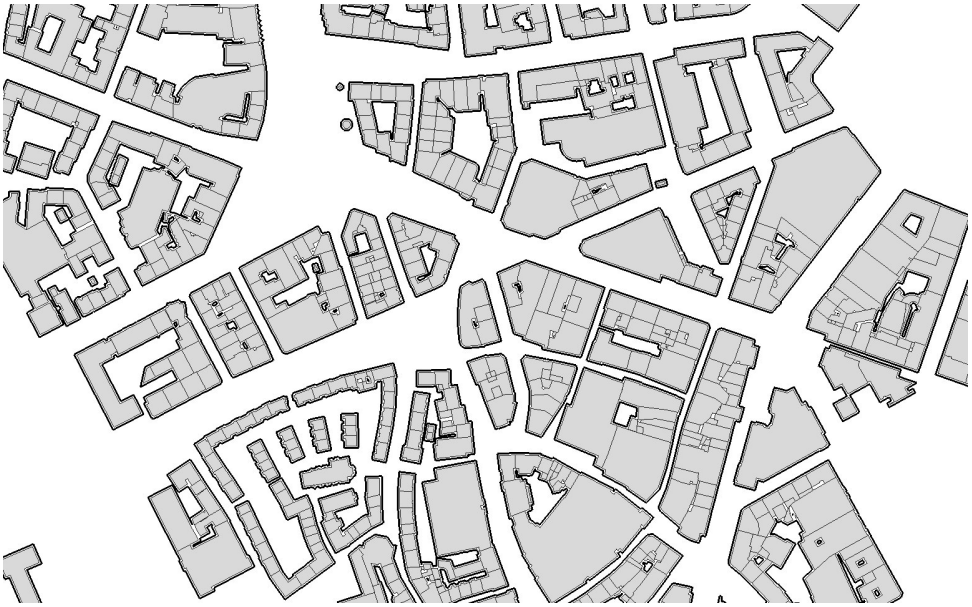


Abbildung 7.4.: Part of the Hanover data set, bold lines: block outlines for $d = 1m$.

In the case of a uniform target resolution ϱ, using a buffer operation is the more natural way to perform a neighborhood analysis: Two buildings b_i and b_j are considered adjacent if a buffer of size ϱ around building b_i intersects building b_j – or, more or less equivalently, if buffers of distance $d := \varrho/2$ around b_i and b_j intersect each other.

This neighborhood relation defines a global neighborhood graph G in which the buildings are the vertices and an edge is inserted if two buildings are adjacent. Usually, this graph will not be connected: the connected components of G form the blocks on which the calculations will be performed. In most cases, the gaps will correspond to the road network, so both approaches for the separation of the blocks should yield similar results for most reasonable values of d. In Figure 7.4, the blocks for $d = 1m$ are surrounded by the bold black outlines.

If d is large, then the blocks can become too large for the optimization-based aggregation process. In this case, the aggregation process can initially be performed for smaller units (defined, for example, by a buffer operation with a smaller value of d). Afterwards, the resulting aggregated blocks can be aggregated in the same way as the smaller units.

At intermediate scales, typification may be employed: groups of buildings of similar height are replaced by other groups consisting of fewer members with emphasized schematic footprint areas and a similar distribution.

At very small scales, the aggregation approach of Glander and Döllner (2009) described in more detail in section 4.2 is promising: A whole block of buildings is replaced by extruding the terrain over a common footprint (defined by the road network or a buffering / closing operation on the footprints of the buildings in the block) by an average building height. If certain buildings (landmarks) are considerably higher than the rest of the block, they are not included in the aggregated block but preserved as individual entities.

Within the blocks, we need a neighborhood graph for the aggregation problem because merging buildings across distances greater than ϱ would violate the Hausdorff criterion introduced in section 2.1.1.

In rural areas or suburbs, the distances between buildings are usually too large for aggregation at scales at which it still makes sense to use 3D building models – at such small scales, the buildings would be aggregated into a large abstract "built-up" area, and in terrain of different elevation, this area should follow the terrain. This would, however, mean that the buildings are no longer LoD 1 buildings in the sense introduced here. For this reason, we only consider aggregation processes here that produce aggregated buildings with flat roof surfaces of uniform absolute height.

In addition to the height differences, further criteria can be used to produce aggregations with a minimum (e.g. visual) impact on the general shape of the block. In order to assess the visual impact of a change of the height of a building, the area of its footprint is important.

Additionally, all the measures one can think of may be used in hard constraints as well as in an objective function. An example of the first case is our threshold for height differences: Putting two buildings with a height difference of more than Δ_h into the same cluster is forbidden for a valid solution. Trying to minimize the total volume change in the course of an aggregation is an example of a measure appearing in the optimization function.

All experiments were conducted using a Core i7 CPU desktop PC with 16GB main memory and could be reproduced on a notebook with a Core2 Duo CPU and 3 GB RAM: All problems that could be solved on the desktop PC could also be solved on the notebook (it may have failed because of missing memory space for the Branching tree of the Branch-and-Bound solving strategy) but needed about twice the time there. All runtime measurements in the text refer to the desktop PC.

7.1. Aggregation of LoD 1 Building Models: A MIP implementation

7.1.1. Assigning buildings to clusters

For the basic MIP representation, the association of a building v to a cluster u is represented by a binary variable $X_{uv} \in \{0, 1\}$. The clusters are defined by the ID of one of their members (the *center* of the cluster); since all buildings may form individual clusters if no aggregation could be performed, the variables X_{uv} are defined for all $u, v \in B$, where B is the set of all building IDs. These definitions are equivalent to the ones described in Haunert (2009) and Guercke et al. (2011).

In order to express this concept in the form of a MIP problem, several constraints have to be introduced.

Constraint 7.1

All buildings are assigned to exactly one cluster:

$$\forall v \in B : \sum_{u \in B} X_{uv} = 1$$

Constraint 7.2

If building v is assigned to building u, then u has to be the center of a cluster (assigned to itself):

$$\forall u, v \in B : X_{uv} = 1 \Rightarrow X_{uu} = 1$$

Since we want a linear representation of this constraint, we write this in a slightly different way:

$$\Leftrightarrow \quad \forall u, v \in B: \; X_{uv} \; \leq \; X_{uu}.$$

If $X_{uv} = 1$, then X_{uu} has also got to be 1 as well to satisfy the constraint; otherwise, the constraint is satisfied in any case.

7.1.2. Connectivity

So far, we have ensured that the clusters form a valid partitioning of the set of buildings in the block: Every building is assigned to exactly one cluster, so the set of clusters covers the set of buildings completely without overlaps of the clusters.

In order to make sure that only adjacent buildings are merged, a flow model is used. The basic idea behind this model is simple: If a virtual commodity can be transported from all vertices (buildings) in the cluster to its center on edges of the neighborhood graph without leaving the cluster, then the cluster is connected, and it is impossible that the cluster consists of unconnected "islands" or that buildings are merged wildly across the block.

In (Haunert and Wolff, 2006), the flow model is used to ensure a minimum size for the resulting partitions in addition to ensuring connectivity. In the case of the building aggregation problem, there are no minimum sizes, so we only want to ensure connectivity, and the constraints can be simplified.

All buildings generate one unit of flow; the centers are the sinks of the network. For each edge a in the set of (directed) edges A of the neighborhood graph, two variables are defined to keep track of the flow: the binary variable $F_a \in 0, 1$ indicates that there is a positive flow on edge a, the continuous variable $f_a \in \mathbb{R}$ represents the (positive) amount of flow on that edge.

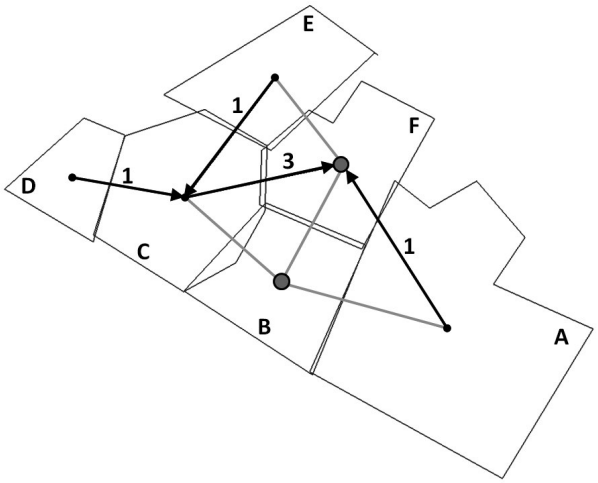


Abbildung 7.5.: Illustration of the flow Model.

Figure 7.5 illustrates the flow model for a simple block: The buildings A and C through F form a cluster C_1; building B could not be merged with the rest. Building F is the center of cluster C_1. Since there are four other buildings in the cluster, it is a sink that consumes a total of four units of flow. Building C receives two units of flow from D and E and generates one itself, so it transfers 3 units to F. Note that the one unit of flow generated by E may also have been tranferred directly to F without the detour via C; in this case, the edge EF would have carried one unit of flow and the edge CF only two units.

Constraint 7.3

First, we make sure that F_a is set if there is a flow on edge a ($f_a \neq 0$) in the neighborhood graph:

$$\forall a \in A: \; M \cdot F_a \; \geq \; f_a.$$

If the number M is greater than any possible value of f_a, then F_a has to be set to 1 if f_a is grater than 0

because $f_a \geq 0$. Since each building generates one unit of flow, the maximum possible flow on an edge is the number N_b of buildings in the block. For this reason, it is safe to set $M = N_b$.

Constraint 7.4

Each node (building) v that is not a center generates a one unit of flow so its net outflow should be 1; if v is a center, it has a negative outflow:

$$\begin{aligned} \forall v \in B: \quad \sum_{a=(v,w)} f_a - \sum_{a=(w,v)} f_a &\geq 1 - X_{vv}(M+1) \\ \sum_{a=(v,w)} f_a - \sum_{a=(w,v)} f_a &\leq 1 - X_{vv} \end{aligned}$$

The left side of both constraints is the net outflow f_v^{out} of vertex v, $a = (v, w)$ means that edge a points from vertex v to vertex w. If vertex v is a center ($X_{vv} = 1$), then the constraints are $f_v^{out} \geq 1 - (M+1) = -M$ and $f_v \leq 1 - 1 = 0$. This means that f_v^{out} is smaller than zero; because we have to define a lower bound for the case that v is not a sink, the term $-X_{vv}(M+1)$ makes sure that a sink can consume any possible amount of flow (which would be $M - 1$, see constraint 7.3).

If v is not a center, then the equations simplify to $f_v^{out} \geq 1$ and $f_v^{out} \leq 1$, which means $f_v^{out} = 1$: A non-center vertex generates one unit of net outgoing flow.

Constraint 7.5

The commodity may flow only between nodes of the same cluster; this means that if an edge $a = (v, w)$ carries positive flow, then v and w must belong to the same cluster:

$$\forall a = (v, w), \forall c \in B: \; X_{cv} \; \geq \; X_{cw} + (F_{vw} - 1).$$

If $F_{vw} = 0$, then this constraint is always relaxed. Otherwise, the set of constraints is simplified to $X_{cv} \geq X_{cw}$ for all v, w; therefore, there is always another constraint $X_{cw} \geq X_{cv}$ in the set, which means that $X_{cv} = X_{cw}$ for all v, w and for all possible centers c – for this reason, X_{cv} and X_{cw} will be 1 for the same center C and 0 for all others: v and w belong to the same cluster defined by C.

Constraint 7.6

The preceding three constraints are sufficient to ensure connectivity. The result may, however, be forking and interleaving paths from the sources to the sink. There could also be edges with a positive flow leaving a sink. Because these different results add a lot of complexity to the computation (severely limiting the size of problems that can be solved within a reasonable amount of time) and can lead to less concise solutions, we introduce a constraint that makes sure that only one edge from a source and none from a center carries a positive flow:

$$\forall v \in B: \; X_{vv} + \sum_{a=(vw)} F_a \; \leq \; 1.$$

If v is a center ($X_{vv} = 1$), then none of the outgoing edges from v can carry a positive flow because the sum ($S_{out} := \sum_{a=(v,w)} F_a$) of the indicator variables for positive flow out of node v is (lower than or equal to) zero, meaning that none of the outgoing edges can carry positive flow. If v is not a center ($X_{vv} = 0$), then $S_{out} \leq 1$, which means that one outgoing edge will carry positive flow – one edge must carry positive flow in order to connect v to its associated center; this is ensured by constraint 7.4.

7.1.3. Constraints to model the basic LoD 1 aggregation problem

The constraints defined so far form a basic frame for any network optimization problem that requires a partitioning of the underlying connectivity graph. They are a simplified version of the constraints defined in Haunert (2009). The constraints in this section introduce concepts related to the aggregation of LoD 1 building models.

The first and most basic constraints concern the height of the individual buildings and of the aggregated buildings. We introduce a variable H_v for each building v that represents its height after the aggregation. The following constraints enforce the requirements for the heights in the aggregation process.

Constraint 7.7

We ensure that all buildings in a cluster have the same height after the aggregation by forcing the heights H_v of all buildings v in the cluster with center u to be equal to the height H_u of u after the aggregation:

$$\begin{aligned} \forall u, v \in B : H_v &\leq H_u + (1 - X_{uv})M_H \\ H_v &\geq H_u - (1 - X_{uv})M_H \end{aligned}$$

This means that $H_v = H_u$ if building v is assigned to the cluster u: In this case, the two constraints become $H_v \leq H_u$ and $H_v \geq H_u$, so putting them together, we get $H_v = H_u$. If v is not assigned to cluster u, we get: $H_v \leq H_u + M_H$ and $H_v \geq H_u - M_H$, so these constraints are always relaxed if we set $M_H := \max\{|h_v - h_u|, u, v \in B\}$.

Constraint 7.8

Using the minimum acceptable absolute height $h_{min}(v)$ defined in chapter **??**, we prevent the buildings from "drowning" in the terrain.

$$\forall v \in B : \; H_v \; \leq \; h_{min}(v) :$$

The absolute level H_v of the roof of building v after the aggregation is not lower than $h_{min}(v)$.

Constraint 7.9

According to the definition of the basic BuildingAggregation problem, no two buildings with a height difference of more than Δ_H can be put in the same cluster:

$$\forall u, v \in B : |h(v) - h(u)| \; \leq \; \Delta_H + (1 - X_{uv})M_H.$$

If buildings u and v are not part of the same cluster, then the constraint becomes $|h(v)-h(u)| \leq \Delta_H + M_H$ which is always true because of the definition of M_H in constraint 7.7. If $X_{uv} = 1$, then $|h(v) - h(u)|$ has to be lower than (or equal to) Δ_H. This matches the definition of the BuildingAggregation problem. Note that h_u and h_v are known in advance and not part of the optimization problem. For this reason, the term $|h(v) - h(u)|$ is also a constant with respect to the optimization problem, so it can be calculated in advance, and only its value is inserted in the MIP formulation.

Using the definitions introduced so far, any building in the cluster may be its center. While this does not change the general result of the optimization process, it can slow down the progress of the MIP solver considerably, because if C_i is the set of nodes in cluster i in the optimal solution, then there are $|C_i|$ equivalent solutions for each cluster i. In the worst case, the solver has to examine every combination of choices of the centers for all blocks in order to determine that they will not lead to better result than the current solution, so the number of possibilities for all clusters is the product of the number of possibilities for all clusters – which can easily make the difference between a runtime of seconds and hours (or running out of resources).

Constraint 7.10

In order to avoid this problem, we simply select the building with the smallest ID as the center of the cluster if we do not introduce additional constraints that can profit by selecting a special node as the center of the block:

$$\forall u, v \in B : X_{uv}(ID(v) - ID(u)) \geq 0.$$

If $ID(v) \geq ID(u)$, then $ID(v) - ID(u) \geq 0$, and therefore $X_{uv}(ID(v) - ID(u)) \geq 0$ no matter if $X_{uv} = 0$ or $X_{uv} = 1$. Otherwise (if $ID(v) < ID(u)$), $ID(v) - ID(u) < 0$, so $X_{uv}(ID(v) - ID(u))$ can only be ≥ 0 if $X_{uv} = 0$. This means that the IDs of all buildings in a center are larger than the ID of the center, so the center is the building with the lowest ID.

Note that this means that we need to define only half of the variables X_{uv}, namely the set of variables $\{X_{uv}|v \geq u\}$. Fortunately, the ID values are constants with respect to the optimization problem, so CPLEX and any other MIP solver will instantly realize that all X_{uv} variables will have to be 0 if $v < u$ and automatically remove these variables from the problem in a matter of seconds.

For this reason, we keep the generality of the representation and save the effort of having to introduce this restriction into any sum and $\forall u, v \in B : \ldots X_{uv} \ldots$ expression by defining all X_{uv} variables. In the context of introducing semantics and different linear height models, we will see that a problem may be simplified if the center of a cluster is not the building with the lowest ID but with the best semantic class or height for the cluster.

7.1.4. Objective functions for the basic problem

So far, we have defined several constraints to model the connectivity of the clusters and to ensure that no buildings with a height difference of more than Δ_h can be part of the same cluster. There is, however, no goal or objective function yet.

In the framework of MIP (and mathematical optimization in general), there can only be a single objective function – in Linear Programs, this function additionally has to be linear in the variables of the optimization problem.

If we want to use different optimization criteria, we can define the corresponding objective functions and compose the global optimization function as a weighted sum of the different objectives:

$$F = \sum W_i O_i$$

where W_i is the weight of the objective function O_i corresponding to goal i. If we want our problem to be linear, all O_i have to be linear. In general, we assume that we want to minimize a cost function, so the overall objective function is minimized. If a partial objective is to maximize a certain function O_i, we have to invert the sign of the corresponding weights W_i in the composition of the global objective function.

Objective 7.1

The most basic objective is to minimize the total number of aggregated buildings (clusters):

$$F_{Clusters} = \min \sum_{v \in B} X_{vv}$$

Constraint 7.11

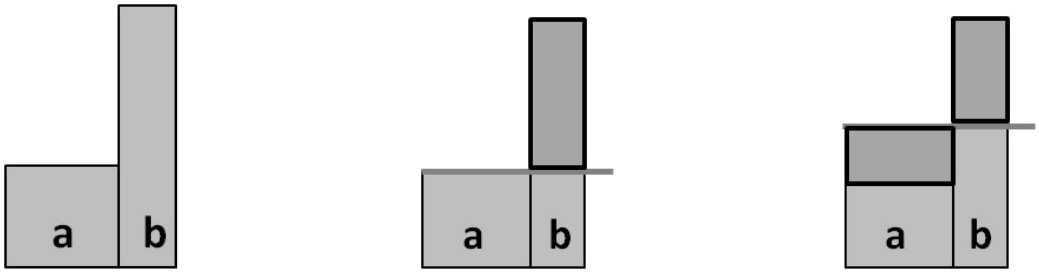
In order to model the absolute value $|(H_v - h_v)|$, we introduce a variable $\Delta_{h,v}$ for each building v and the following constraints:

$$\begin{aligned} \forall v \in B : \Delta_{h,v} &\geq H_v - h_v \\ \Delta_{h,v} &\geq -H_v + h_v \end{aligned}$$

Objective 7.2

The impact of a an aggregation procedure depends strongly on the size of a building: Changing the height of a building with a large footprint has a greater impact than changing the height of a smaller building. Using the real area A_v of the footprint as a scaling factor for the height difference would, however, emphasize the footprint size disproportionately in comparison to the height difference. Using $A'_v = \sqrt{(A_v)}$ yielded reasonable results; choosing a sensible function for deriving A'_v is a degree of freedom for fine-tuning the method to fit special user requirements.

$$F_{Volume} = \min \sum_{v \in B} \Delta_{h,v} A'_v$$



(a) Two buildings.

(b) Minimizing volume differences: Median height.

(c) Minimizing squared volume differences: Intermediate (least squares) height.

Abbildung 7.6.: Minimizing volume differences leads to median instead of intermediate height for the cluster.

This objective will snap the height of the cluster to the height of the building with the median value of $h_i A'_i$ in the cluster: If one considers two buildings, the total change of volume is minimal if the aggregate receives the height of the building with the greater value of A', for example building a in figure 7.6(a). If an intermediate height h was chosen (right drawing), then an additional volume change of $|h_a - h|(A'_a - A'_b)$ compared to choosing the height of the building with the bigger footprint area (7.6(b)) would be the result. In general, choosing the height of the smallest building a for which the sum over the values A' of all buildings with a height of h_a or less is smaller than the sum of the values A' of the buildings higher than (not of equal height) a will minimize the sum of the total pseudo-volume change for the aggregated building.

Objective 7.3

Squaring the values will result in an adjustment of the heights within the cluster in a least squares sense, but the resulting MIP problem is considerably more complex – instead of a *linear* MIP problem, we have a *quadratic* MIP problem:

$$F_{Volume} = \min \sum_{v \in B} (\Delta_{h,v} A'_v)^2$$

Using the CPLEX software, this optimization problem could be solved despite the increased complexity for almost all of the blocks in the test data sets and almost all tested combinations of generalization parameters; there were, however, more combinations of blocks and parameters for which the problem could not be solved than in the linear case. Additionally, runtime went up from several seconds to some minutes on the Core i7 desktop PC for the larger blocks in the usual cases – for the smaller and intermediate-sized blocks, it usually stayed in the order of seconds.

7.1.5. Extensions

Depending on the application, it may be necessary to distinguish between buildings based on semantic classes – for example, if buildings with different functions, risk levels or other properties are going to be represented in, say, different colors in a visualization scenario.

As in Haunert (2009), we define a set Γ of classes; the function γ assigns a class γ_v to each building v. Additionally, there is a function $d : \Gamma^2 \to \mathbb{R}_0^+$ that defines the semantic difference between the classes. For each class $c \in \Gamma$ and for each building $v \in B$, we introduce a binary variable $S_{v,c}$; if $S_{v,c} = 1$, then building v has the class c after the aggregation process.

Constraint 7.12

Each building has exactly one class after the aggregation:

$$\forall v \in B : \sum_{c \in \Gamma} S_{v,c} = 1$$

Constraint 7.13

If building v is assigned to center u, then its class has to match the class of the center:

$$\forall u, v \in B, \forall c \in \Gamma : X_{uv} \Rightarrow S_{v,c} = S_{u,c}$$
$$\Rightarrow S_{v,c} \geq X_{uv} + S_{u,c} - 1$$

The second form of this constraints means that if $X_{uv} = 1$ and $S_{u,c} = 1$, then the right side of the inequality becomes 1 and $S_{v,c}$ must therefore be set to 1. Since only one $S_{v,c}$ can be set for each building v (constraint 7.12), this means that $X_{uv} \Rightarrow S_{v,c} = S_{u,c}$.

Objective 7.4

Our objective is to minimize the total cost for assigning the new classes to the aggregated buildings by adding the term $W_{Sem}O_{Sem}$ to the global objective function where W_{Sem} is the weight of the semantic differences with respect to the other objective values; since we want to minimize O_{Sem}, W_{Sem} will be positive:

$$O_{Sem} = \sum_{v \in B, c \in \Gamma} d(\gamma_v, c) \cdot S_{v,c}$$

Note that d is usually not symmetric: While it may be no problem to make a building of little significance to the application part of a building of special interest, the converse results in the loss of a significant building which would incur high costs or may even be fatal.

An important optimization goal for digital city models is the reduction of the amount of data to be stored, transferred through networks and processed by the application. The number of resulting building objects is one important aspect of the complexity of the resulting data sets, but the complexity of the resulting building footprints also contributes to the complexity of the resulting data set.

For this reason, the number of vertices in the resulting polygons is an additional measurement for the complexity of the result and therefore a quantity to be minimized in the optimization process. In order to include this measure in the aggregation process, we can count the number of edges (or vertices) saved by the aggregation of a pair of buildings and try to maximize the sum of the edges saved in the aggregation process.

If the input building footprints are, for example, cadastre data sets or jagged outlines derived from Lidar interpretation, then artifacts or other features of sizes below the target resolution may unduly influence the aggregation process because there may be an excessively large number of vertices in the area to be filled up in the course of the aggregation.

In figure 7.7, for example, all points of the zig-zag patterns forming the sides of the building the face each other would be counted as successfully eliminated segments in the aggregation although both could have been reduced to a single line within the given resolution.

Because of this effect, it is necessary to simplify the footprints of the original buildings before the aggregation takes place. An overview of existing approaches to achieve this and a sketch of a new approach to solve this problem are given in section 6.

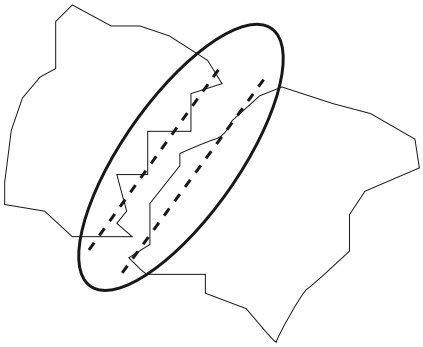


Abbildung 7.7.: Jagged outlines may influence geometric complexity measures for aggregation.

If all individual building footprints have been simplified to the target resolution, then we can calculate the geometric benefit g_a for each edge $a = (u, v)$ of the neighborhood graph as the number of points on the outlines saved by joining u and v.

Constraint 7.14

In order to include this measure into our objective function, we first have to introduce decision variables $J_{a=(u,v),c}$ that indicate if the buildings u and v belong to the same cluster c:

$$\forall a = (u, v) \in A,\ \forall c \in B: \quad J_{a,c} \geq X_{cu} + X_{cv} - 1 \ \wedge\ J_{a,c} \leq (X_{cu} + X_{cv})/2$$

The first term makes sure that $J_{a,c}$ has to be set if u and v belong to center c, the second part forces $J_{a,c}$ to be set to 0 if either u or v does not belong to cluster c. For this reason, $J_{a,c}$ is TRUE if and only if u and v belong to center c.

Objective 7.5

Now we can add the term for the geometric benefit to the global objective:

$$-W_{Geom} \sum_{a=(u,v)\in A, c\in B} g_a \cdot J_{a,c},$$

where W_{Geom} is the (positive) weight of the geometric benefit compared to the other objectives. Because the global objective is to minimize a cost function, the term for the benefit has to be added with a negative sign.

7.1.6. Overview of all constraints and objective function terms

In this section, a list of all constraints and objective function terms in the MIP representation of the LoD 1 building aggregation problem that were introduced in this chapter is given.

The first group of constraints is a common core of most flow-based optimization problem representations and includes constraints that ensure the connectivity of the clusters in the neighborhood graph. The actual constaints may be different in other representations (in Haunert (2009), for example, the constraints are more complex because they are used to ensure minimum sizes for the resulting regions), but the basic structure is similar.

Assignment of buildings to aggregates; connectivity:

C 7.1	Each building is assigned to exactly one center:	$\forall v \in B: \sum_{u \in B} X_{uv} = 1$
C 7.2	Buildings can only be assigned to centers:	$\forall u, v \in B: X_{uv} \leq X_{uu}$
C 7.3	Set F_a if $f_a \geq 0$:	$\forall a \in A: M \cdot F_a \geq f_a.$
C 7.4	All buildings generate 1 unit of net flow (centers are sinks):	$\forall v \in B: \sum_{a=(v,w)} f_a - \sum_{a=(w,v)} f_a \geq 1 - X_{vv}(M+1)$ $\sum_{a=(v,w)} f_a - \sum_{a=(w,v)} f_a \leq 1 - X_{vv}$
C 7.5	No flow across cluster boundaries:	$\forall a = (v,w), \forall c \in B: X_{cv} \geq X_{cw} + (F_{vw} - 1).$
C 7.6	At most one outgoing edge carries positive flow (center: none):	$\forall v \in B: X_{vv} + \sum_{a=(vw)} F_a \leq 1.$
C 7.10	The center of a cluster is the building with the smallest ID:	$\forall u, v \in B: X_{uv}(ID(v) - ID(u)) \geq 0.$

The second group of constraints is used to model the specific aspects of building aggregation: A uniform height for all buildings within each cluster and thresholds for individual and total height and volume changes are enfoced. These height and volume change thresholds control the degree of generalization and can be derived from the global target resolution.

Constraints to model the buiding aggregation problem:

C 7.7	All buildings in a cluster have the same height:	$\forall u, v \in B: H_v \leq H_u + (1 - X_{uv})M_H$ $H_v \geq H_u - (1 - X_{uv})M_H$
C 7.8	The height of a building may not lower than threshold h_{min}:	$\forall v \in B: H_v \leq h_{min}(v):$
C 7.9	Forbid height changes of more than Δ_H:	$\forall u, v \in B: \lvert h(v) - h(u) \rvert \leq \Delta_H + (1 - X_{uv})M_H.$
C 7.11	Set height change variables $\Delta_{h,v}$ for the objective function:	$\forall v \in B: \Delta_{h,v} \geq H_v - h_v$ $\Delta_{h,v} \geq -H_v + h_v$

The goals (objective function terms) are to minimize the number of resulting buildings and the amount of volume change neccessary to perform the aggregations. If the volume change is squared in the objective function, a least squares adjustment of the heights of the buildings within each cluster is performed as a part of the solution of the MIP problem.

In most non-trivial cases, there is a conflict between the two goals because with an increasing number of original buildings in a cluster (aggregated building), the height differences and, as a result, the volume differences are likely to increase. In the setup for the experiments, the objective term weights were chosen to make sure that the first goal of a minimum number of aggregated buildings always took precedence over the second goal in order to make the results of the MIP-based and the heuristic approaches more comparable. The degree of generalization is defined by the threshold consraints for the total and individual height and volume changes.

Components of the objective function:

O 7.1	Minimize the number of aggregated buildings:	$F_{Clusters} = W_{Clusters} \sum_{v \in B} X_{vv}$
O 7.2	Minimize sum of volume changes (absolute values):	$F_{Volume} = W_{Volume} \sum_{v \in B} \Delta_{h,v} A'_v$
O 7.3	Minimize sum of squared volume changes:	$F_{Volume} = W_{Volume} \sum_{v \in B} (\Delta_{h,v} A'_v)^2$

In most cases, either objective 7.2 or objective 7.3 will be used; for this reason, the name of the objective term and the weight are the same for both objectives.

In the following table, MIP constraints and objective function terms for modeling semantic aspects and the geometric complexity of the resulting building shapes are introduced as possible extensions. These extensions are, however, not part of the current implementation.

Extensions for modeling membership of semantic classes:

C 7.12	Each building has exactly one class after the aggregation:	$\forall v \in B : \sum_{c \in \Gamma} S_{v,c} = 1$
C 7.13	If building v is assigned to center u, then its class has to match the class of the center:	$\forall u, v \in B, \forall c \in \Gamma : X_{uv} \Rightarrow S_{v,c} = S_{u,c}$
O 7.4	Minimize sum of class change penalties:	$F_{Sem} = W_{Sem} \sum_{v \in B, c \in \Gamma} d(\gamma_v, c) \cdot S_{v,c}$

Extensions for modeling geometric complexity:

C 7.13	Set $J_{a=u,v,c}$ if u, v belong to cluster c:	$\forall a = (u,v) \in A, \forall c \in B : \quad J_{a,c} \geq X_{cu} + X_{cv} - 1$ $\quad \wedge \quad J_{a,c} \leq (X_{cu} + X_{cv})/2$
O 7.5	Minimize negative sum of geometric simplification benefits:	$F_{Geom} = -W_{Geom} \sum_{a=(u,v) \in A, c \in B} g_a \cdot J_{a,c}$

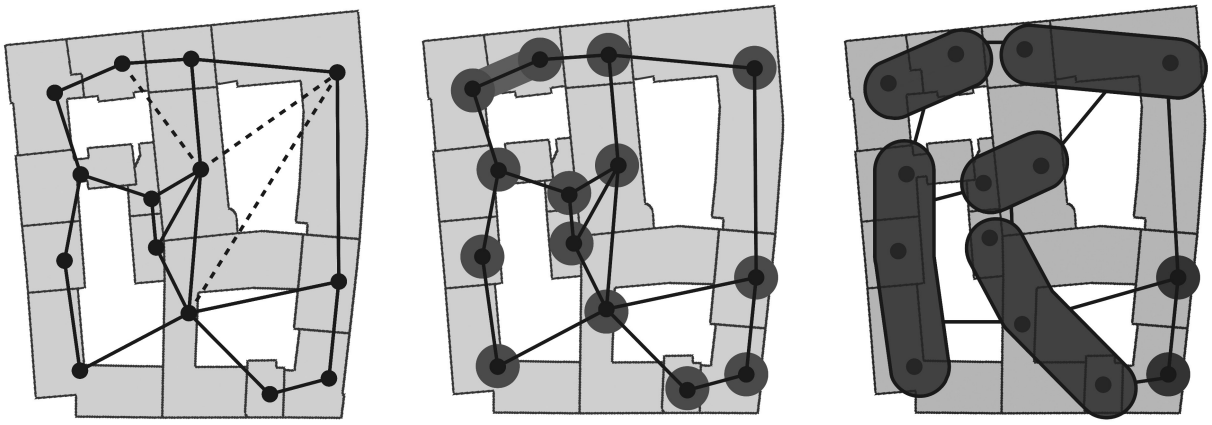
7.2. Heuristic approaches based on region growing

As shown in the introduction to this chapter, approaches to find optimum solutions (with the resulting risk of super-polynomial runtime characteristics) and heuristic approaches that try to find good solutions with a reasonable effort can be used in a complementary fashion in different ways.

In the test scenarios, the objective was to produce a minimum number of buildings without causing height differences of more than the target resolution. Another hard constraint was that only a limited average per-building pseudo-volume change $\overline{\Delta}'_{V,max}$ was allowed for each block, meaning that for a large block of 50 buildings, the sum of the pseudo-volume change for all aggregated buildings must not exceed $50 \times \overline{\Delta}'_{V,max}$. To avoid excessive changes to single buildings, the pseudo-volume change for each building may also be limited.

Two simple heuristic approaches are introduced here. Both are based on the principle of region-growing: In the beginning, each building forms a separate cluster. At each step, two clusters are merged; the process terminates when no further merging operations are possible.

The first one simply chooses an arbitrary possibility when faced with the decision which buildings to merge in a given step (*greedy* region-growing heuristic), the second one chooses the "best" aggregation for a given situation (*best-first* region-growing heuristic). In the case of the basic BuildingAggregation problem, a good heuristic for choosing one of the possible aggregation steps is to perform the aggregation step that causes the smallest change of height or volume.



(a) Neighborhood graph for a block, dashed lines: illegal neighbors.

(b) Heuristic approaches: initial situation.

(c) After 5 aggregation steps.

Abbildung 7.8.: Stages of the region-growing algorithms.

Figure 7.8 illustrates the process for a block of buildings. Figure 7.8(a) shows the initial neighborhood graph obtained by testing the intersection of buffers around the buildings. Although the buffers around the buildings connected by the dashed lines intersect, they should not be considered neighbors because they meet only in a corner. This can be excluded by applying a negative buffer to the union of the two buffered buildings and testing if the resulting region – which is the result of the closing operation performed on the buildings – yields a single shape.

In the beginning, each of the buildings forms a separate cluster; the thin black lines in the figure represent valid aggregation options. In each aggregation step, a pair of clusters is merged into a single cluster. In the greedy heuristic approach, an arbitrary pair of clusters is selected. The best-first approach tentatively merges all pairs of clusters and selects the pair with the minimum impact on the global objective function. If merging the selected pair of buildings causes a global constraint to be violated, a new pair is chosen. The process terminates if no more pairs of clusters can be merged. The situation in figure 7.8(c) is a schematic illustration of the general merging procedure common to both the greedy and best-first heuristic approaches and not an actual result of either the greedy or best-first approach.

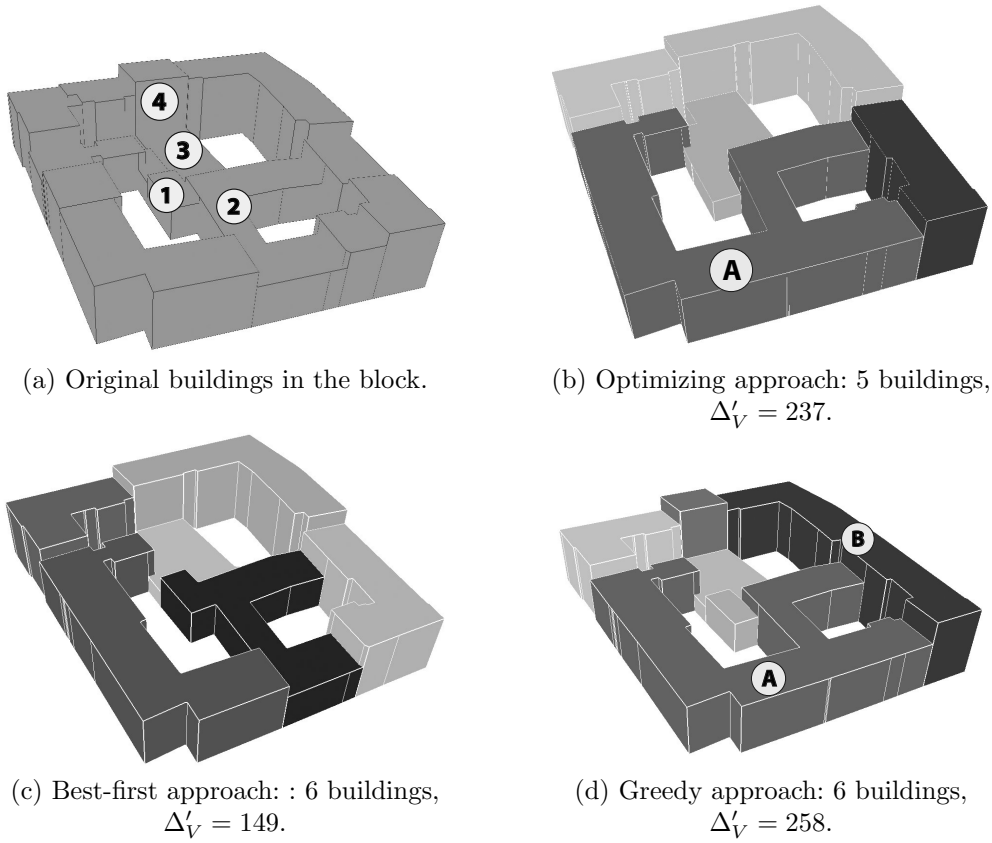
The basic problem of all heuristic approaches is that they have no backtracking options: Once they take the decision to merge two clusters, they will not be able to revise their decision.

Another problem arises if different constraints have to be satisfied. In this case, the best-first approach can run into trouble: Different possible aggregation steps may favor one constraint but be harmful in terms of a different one and vice versa. Since there is no inherent weight between hard constraints, a heuristic approach would have to incorporate strategies for making a sensible choice in such a situation: In the case of accumulating constraints like the restricted total pseudo-volume change, for example, one can assign a weight to each constraint value that increases with the proportion to which the corresponding pool has been used up.

In the case of our examples, the pseudo-volume change is the only part of the objective function – except, of course, for the number of aggregated buildings. For this reason, there is no conflict between choosing the pair of clusters with the best objective value and the smallest pseudo-volume changes (in order to avoid using up the allowed amount of pseudo-volume change): They are the same by the nature of the objective function.

If we had introduced another term rewarding geometric simplifications with a high weight factor, such a conflict may easily occur: Merging one pair of clusters may have a huge geometric simplification benefit but use up more of the volume change pool than a different pair with only a small or no geometric benefit. If the former pair was chosen, then this choice may later prohibit a series of other merges and lead to a worse solution. In the latter case, a sub-optimal solution will be the result if the the pair with the better objective value would not have made other parts of an optimum solution impossible. Trying to keep track of two or more possible pairs in each merging step would lead to an exponential runtime.

In the case of the greedy approach, each additional constraint increases the probability that the approach gets stuck, i.e. it chooses an aggregation step that prevents many other much more favorable aggregation steps from being performed.



(a) Original buildings in the block.

(b) Optimizing approach: 5 buildings, $\Delta'_V = 237$.

(c) Best-first approach: : 6 buildings, $\Delta'_V = 149$.

(d) Greedy approach: 6 buildings, $\Delta'_V = 258$.

Abbildung 7.9.: Optimizing and heuristic approaches: Backtracking.

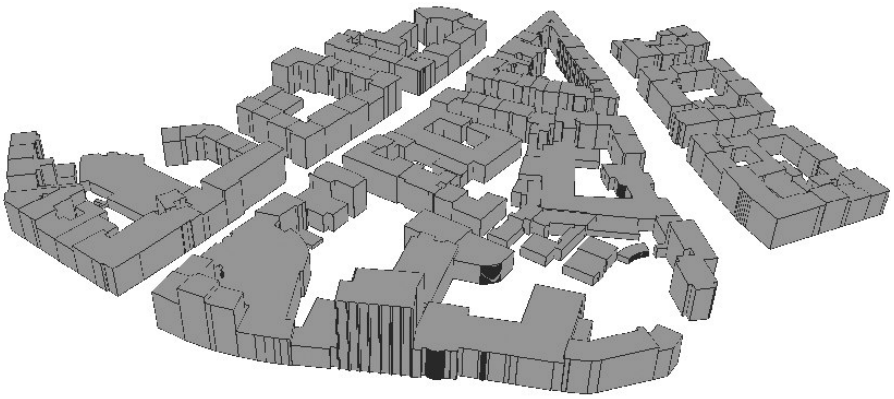
The results obtained with the different approaches are compared using the building block shown in Figure 7.9(a). Figure 7.9(b)–(d) show the resulting aggregated buildings for the optimizing (b), the best-first heuristic (c) and the greedy heuristic (d) approach. The shades are used to distinguish the aggregated buildings; the assignment of the shades to the different aggregated buildings is random.

The optimizing approach yields the smallest number of aggregated buildings (5), at the expense of a higher volume change (237). The best-first heuristic approach merged buildings 1 and 2 at an early stage, and thus could not form the large aggregated building A. For this reason, it produced 6 instead of the optimal number of 5 aggregated buildings. The greedy approach managed to form the aggregated building A, but because it also formed the large and expensive (in terms of pseudo-volume change) aggregated building B, it used up its volume pool early and could not merge any of the buildings 1, 3, or 4 with another building without violating the maximum average volume change constraint (set to 20 units per building meaning a total pool of 14*20=280 units because there are 14 buildings in the block). For this reason, it also produced 6 aggregated buildings and a larger total amount of pseudo-volume difference than the best-first heuristic approach (258 vs. 149 units).

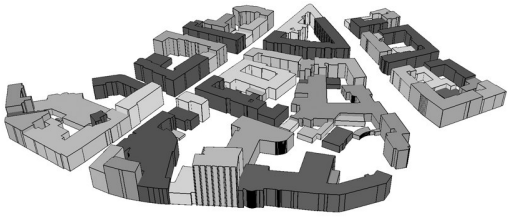
7.3. Results

For the experiments, a simple setup was used to test the heuristic approaches: The objective was to minimize the number of clusters in the aggregated block; if there were different possibilities to achieve the same number of clusters, then the one causing the smallest pseudo-volume change was to be selected. This was achieved in the MIP setup by setting the weight for the number of clusters in the result and the weight for the volume-change to the inverse of the maximum possible total volume change. For the best-first heuristic approach, the target was to minimize the volume change for each merging step.

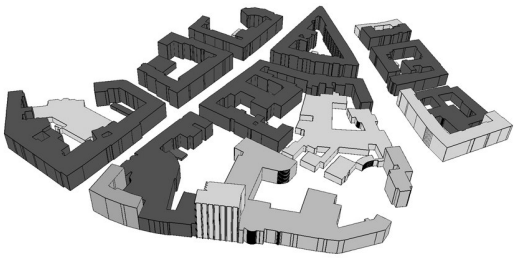
Figure 7.10 shows a part of the Hanover data set at different scales. It illustrates that different height thresholds for the aggregation algorithm presented here indeed form a generalization sequence. As mentioned in the section on 2D and 3D generalization (section 2.3), small and large scales are, in this case, quite different from those usually used in cartography. The scale models were produced with a permissive volume change policy using the optimizing approach; the results from the best-first heuristic approach can, however, not be distinguished from



(a) Original blocks.



(b) An intermediate-scale model of the blocks in (a).



(c) A small-scale model of the blocks in (a).

Abbildung 7.10.: A part of the Hanover data set at different scales. The shades distinguish the aggregated buildings.

the optimum ones in most cases if the resulting clusters are not colored and we do not know for which blocks there are differences. The example was generated using the optimizing approach.

In the tables 7.1 and 7.2, the results of the optimizing and the two heuristic approaches are compared for the Hanover and the Dortmund data set. In each double column section of the table, one approach is compared to another.

In each case, only the instances (blocks) for which algorithm A in the "A vs. B" heading of the double column performed better. In the first double column, for example, the optimizing approach A is compared to the best-first heuristic one B.

The objective function was defined to be dominated by number of resulting buildings: The weight of the pseudo-volume change was chosen to be the inverse of the maximum possible pseudo-volume change; for the secondary *objective value at identical number of buildings* criterion, the actual values of the objective function from the optimization problem were used.

For this reason, the optimizing approach does, by definition, never give worse results than the heuristic ones if it runs successfully. In contrast to the experiments in (Guercke et al., 2011), the quadratic optimization problem incorporating a least squares adjustment of the heights of the buildings within the clusters was used.

Because such a quadratic problem is considerably more difficult to solve, the MIP solver was started with the solution of the more successful heuristic approach as a starting solution and given up to 30 minutes to improve this solution. For this reason, the optimizing approach never produced a worse result than a heuristic one even if it failed to provide a better solution or prove the optimality of the starting solution in the allocated amount of time.

For this reason, there are no "X vs. optimizing" double columns in the tables, because they would all have been empty.

In the first column of each double column, the number of blocks is counted for which algorithm A produced a lower number of buildings than algorithm B. The number in brackets gives the maximum difference for a single block between the results of algorithms A and B.

In the first row in table 7.1, for example, the entry 270(2) in the "optimizing vs. best-first" section means that the optimizing produced a lower number of buildings than the best-first heuristic approach for 270 of the 4095

simple blocks. The (2) annotation means that there was at least one block for which the best-first approach produced two buildings more than the optimizing one.

In the second column of each double column, the blocks are counted for which algorithm A produced the same number of buildings but achieved a better objective value, meaning that the sum of the squared pseudo-volume differences was lower for A than for B: A could produce the same number of buildings as B, but the difference between the generalized and the original block was smaller for A than for B.

		optimizing vs. best-first		optimizing vs. greedy		best-first vs. greedy		greedy vs. best-first	
Dortmund data set	total #blocks	# smaller blocks (max. reduction)	better objective value: #blocks	# smaller blocks (max. reduction)	better objective value: #blocks	# smaller blocks (max. reduction)	better objective value: #blocks	# smaller blocks (max. reduction)	better objective value: #blocks
trivial (≤ 3)	11363	–	–	–	–	–	–	–	–
$\Delta_{h,max} = 1.0m$, $\Delta_{V_{avg},max} = 0.7$ (restrictive)									
simple (≤ 8)	4095	270(2)	524	427(3)	842	177(3)	593	8(1)	49
intermediate (≤ 15)	687	94(1)	153	240(4)	201	182(4)	188	4(1)	14
large (> 15)	134	35(1)	31	83(9)	17	71(9)	18	1(1)	8
$\Delta_{h,max} = 1.0m$, $\Delta_{V_{avg},max} = 5000$ (permissive)									
simple (≤ 8)	4095	3(1)	638	8(2)	728	9(2)	178	3(1)	65
intermediate (≤ 15)	687	1(1)	152	4(1)	176	4(2)	57	0(–)	20
large (> 15)	134	3(1)	39	3(1)	49	2(1)	27	2(1)	8
$\Delta_{h,max} = 1.5m$, $\Delta_{V_{avg},max} = 1.5$ (restrictive)									
simple (≤ 8)	4095	217(2)	597	377(3)	899	174(3)	617	9(1)	66
intermediate (≤ 15)	687	57(1)	136	160(5)	164	119(5)	137	2(1)	16
large (> 15)	134	24(2)	35	74(14)	21	67(14)	21	2(1)	4
$\Delta_{h,max} = 1.5m$, $\Delta_{V_{avg},max} = 5000$ (permissive)									
simple (≤ 8)	4095	11(1)	828	23(1)	1003	22(1)	290	11(1)	67
intermediate (≤ 15)	687	7(2)	200	20(3)	246	19(3)	101	4(2)	26
large (> 15)	134	3(1)	70	3(1)	81	10(2)	33	7(1)	11

Tabelle 7.1.: Data reduction achieved by the optimizing and heuristic approaches at different resolutions for the Dortmund data set.

The results of the approaches were compared for different resolutions where the resolution was mainly defined by the threshold on the maximum height change for a building within a cluster. The data sets were processed with values of $1.0m$ and $2.5m$ for this maximum height difference.

Additionally, there was a limit on the average (scaled) pseudo-volume change for all buildings within each cluster. The algorithms were evaluated with different values for this "global pool" constraint. It turned out that this parameter has a huge impact on the performance of the heuristic approaches: for more restrictive (lower) values, it was considerably more probable that the heuristic approaches got stuck after having taken a series of decisions.

For very permissive values of this parameter, both heuristic algorithms performed almost as well as the optimizing one, especially concerning the number of buildings produced in the output. This is due to the fact that the algorithms have a much higher probability of recovering from a non-optimal merging decision with a good sub-optimum solution if the merging process is not terminated early by having used up a global pool.

The actual values for this overview were chosen to illustrate this effect; the low value for the parameter is due to the fact that it already includes the scaling of the pseudo-volume change in the objective function.

Remember that the NP-hardness of the problem remains and that all heuristic approaches may produce almost arbitrarily bad results even if we relax this bounded pseudo-volume change constraint completely.

If additional, especially conflicting, accumulating hard constraints like bounds on a semantics-based total penalty for the whole block were introduced, the heuristic approaches can be expected to perform considerably worse.

Hanover data set	total #blocks	optimizing vs. best-first: # smaller blocks (max. reduction)	optimizing vs. best-first: better objective value: #blocks	optimizing vs. greedy: # smaller blocks (max. reduction)	optimizing vs. greedy: better objective value: #blocks	best-first vs. greedy: # smaller blocks (max. reduction)	best-first vs. greedy: better objective value: #blocks	greedy vs. best-first: # smaller blocks (max. reduction)	greedy vs. best-first: better objective value: #blocks
trivial (≤ 3)	208	–	–	–	–	–	–	–	–
$\Delta_{h,max} = 1.0m$, $\Delta_{V_{avg},max} = 0.7$ (restrictive)									
simple (≤ 8)	57	5(1)	3	6(1)	4	1(1)	3	0(–)	0
intermediate (≤ 15)	45	5(1)	10	8(4)	13	3(4)	9	0(–)	1
large (> 15)	43	13(1)	8	29(16)	5	24(15)	7	0(–)	0
$\Delta_{h,max} = 1.0m$, $\Delta_{V_{avg},max} = 5000$ (permissive)									
simple (≤ 8)	57	1(1)	19	1(1)	21	0(–)	4	0(–)	0
intermediate (≤ 15)	45	1(2)	20	2(2)	24	1(1)	10	0(–)	0
large (> 15)	43	3(1)	27	3(1)	32	2(1)	19	2(1)	4
$\Delta_{h,max} = 1.5m$, $\Delta_{V_{avg},max} = 1.5$ (restrictive)									
simple (≤ 8)	57	1(1)	6	3(1)	11	2(1)	7	0(–)	0
intermediate (≤ 15)	45	6(1)	9	13(4)	11	8(4)	12	0(–)	1
large (> 15)	43	12(1)	5	34(13)	7	32(13)	8	0(–)	0
$\Delta_{h,max} = 1.5m$, $\Delta_{V_{avg},max} = 5000$ (permissive)									
simple (≤ 8)	57	0(-1)	20	0(–)	21	0(-1)	6	0(–)	1
intermediate (≤ 15)	45	2(2)	26	3(1)	30	2(1)	14	1(2)	2
large (> 15)	43	3(2)	32	4(2)	34	6(1)	23	6(2)	3

Tabelle 7.2.: Data reduction achieved by the optimizing and heuristic approaches at different resolutions for the Hanover data set.

7.4. Summary and Discussion

In this section, the aggregation of LoD 1 buildings has been introduced as an optimization problem. Having established that the problem is NP-hard even in its simplest form, we cannot expect that there is an efficient (polynomial-time) algorithm to find an exact solution for this problem for any input. Fortunately, buildings are arranged in building blocks, providing a natural partitioning scheme that in almost all cases yields units of moderate size – almost never exceeding 60 members.

For these small instances, the problem could be solved exactly in most cases by transforming it to a MIP problem and using special MIP solving software (CPLEX) to determine the optimal solution. Especially if the quadratic MIP resulting in a least squares adjustment of the heights was used, the optimizing approach could sometimes not find an optimal solution in an acceptable amount of time.

The focus of this study was to build a MIP-based framework in which aggregation algorithms can be developed and evaluated rather than to define aggregation algorithms for different purposes. Based on this framework, different algorithms can be evaluated; for example, in the context of a user survey.

In (Guercke et al., 2011), an approach is outlined to extend the MIP-based approach presented here to LoD 2 building models, i.e. building models with simple roof shapes. Implementing and testing such an approach is a direction of research worth further investigation because it would allow a more comprehensive benchmarking of generalization algorithms for LoD 2 buildings that can be expressed in the form of a MIP problem.

The geometric complexity of the individual buildings and the complexity of the aggregation result were not taken into account in the current version of the aggregation algorithms. Such a geometric analysis only makes sense if the individual building footprints are already simplified according to the target resolution before the aggregation is performed. A more detailed description of a possible strategy for the integration of geometric aspects in the aggregation of buildings with footprints simplified to fit the target resolution is given in (Guercke et al., 2011). In the preceding chapter, an approach for such a simplification of the building footprints is introduced.

8. Facade Structure Simplification

Special algorithms for the simplification of complex facade layouts are necessary – especially for rows of townhouses, quite complex decisions may be necessary to determine a sensible simplification across the different parts.

In the generalization framework described in chapter 5, the facade homogenization process operates on the facade structure elements in the building composition hierarchy outlined in section 5.1.4. In the course of the homogenization, it may happen that features have to be merged or deleted – protrusions may, for example, be *pushed back* into the main facade. These procedures are, however, not yet implemented in the current prototype.

In this section, we consider facade structures modeled as a grid of (not necessarily aligned) rectangular facade cells – each cell holds one facade element like a window or a door. For each cell, there is a list of values representing different properties of the facade element within the cell. These properties may be the protrusion or indentation of the cell with respect to the rest of the facade, the width, height and offset (in x and z direction) of a window in the cell, the color of the wall in the background or application-specific data.

Special properties are the width and height of the cells themselves because these should be identical within each column and row, respectively. We will introduce constraints that ensure this and thus make it possible to handle these values in the homogenization process below.

As a preprocessing step for operators like typification and to save memory space, the homogenization operator groups sets of cells into clustered regions in which the parameters of all features are assigned uniform values. In order to take full advantage of the memory saving potential and especially because applying typification after the homogenization procedure is simplified significantly by this property, we require the resulting regions to form rectangles.

Note that aggregation and homogenization are fundamentally different operations: In the homogenization process, the number of features is not changed; only the shapes (parameter values) of the features are made homogeneous across a clustered region. In the case of aggregation, all feaures in the clustered region are merged into a single feature. Both homogenization and aggregation of feature constellations on an facade are simplifications of the structure of the facade. As we will see in the results section of this chapter, there are situations in which there are conflicts between the two operations and situations in which a clever combination of the two generalization operators can provide the best results.

In many cases, there will already be rectangular regions with homogeneous facade elements in the original structure. The procedures given in the following can profit from being supplied with a description in which those regions are specified explicitly, but they give correct results also if all cells are modeled as different regions in the input model. Especially the performance using the first naive MIP description (see section 8.1) uses the fact that the size of the initial partitioning can be used as a lower bound for the size of the number of resulting partitions.

In appendix B, we prove the NP-hardness of even a very trivial case of the homogenization problem in which the goal is simply to minimize the number of resulting tiles and there is only one property and only four possible values for this property. For this reason, it is unlikely that there is an efficient algorithm that finds an exact solution for an arbitrary homogenization problem.

For the comparatively small numbers of cells occurring in the facade homogenization scenario – even in long stretch of facades, there will hardly be more than a few hundred facade element cells (like windows or doors) on the front of a block between two intersections – it is, however promising to try and profit from the experience of the developers of MIP solver software by putting the facade homogenization problem in the form of a MIP instance and feeding it to such a solver.

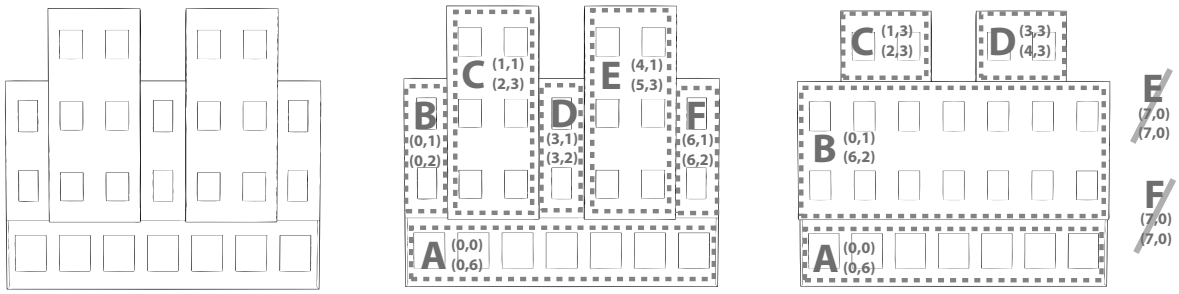
8.1. A direct template-based MIP model for facade homogenization

In this section, we will develop a direct MIP (Mixed Integer Programming, see section 2.5) description of the facade homogenization problem. We will see that although the representation introduced in this section

correctly describes the problem, the graph-based representation given in the next section can be handled far more efficiently by the CPLEX solver: While even a very simple facade structure caused CPLEX to run for more than a day using this simple representation, much more complex facades could be handled within a few minutes using the graph-based representation because for the graph-based version, fast heuristics for flow problems within the CPLEX software were able to speed up the process – there were probably also more ambiguities left in this template-based approach than in the graph-based approach.

The basic idea of this approach is to introduce templates for the tiles in the result. These templates are fitted into the original data set; constraints make sure that the instantiation of the templates does indeed form a valid tiling of the original data set. While we have to *define* tile templates for all possible sensible tilings, one of the main objectives in the optimization will be to actually *use* as few of the templates as possible.

First we define variables that hold the grid addresses of the lower left and upper right (inclusive) corners of the templates. If we know that there were N_F homogeneous facade tiles in the input, then there will be N_F or less tiles in the output, so there are four integer variables for each tile template $f \in \{0, ..., N_f - 1\}$: $X_{min,f}$, $Y_{min,f}$, $X_{max,f}$, $Y_{max,f}$. If we do not know such a lower bound a priori, we have to introduce one template for each cell in the original facade.



(a) Original facade.

(b) Homogeneous tiles in the input.

(c) A tiling with four tiles.

Abbildung 8.1.: Different tilings for a facade

Figure 8.1 illustrates this template-based approach: The facade shown in Figure 8.1(a) is known to consist of the 6 parts (A–F) shown in Figure 8.1(b) that already have homogeneous parameter values. The numbers in the brackets give the indices of the lower left and upper right cells covered by the tile, so the format is $(X_{min,f}, Y_{min,f}), (X_{max,f}, Y_{max,f})$.

Since costs are generated only for adding tiles and for changing parameter values, there will not be more than those 6 tiles in an optimum tiling: the cost for the additional tiles could not be compensated by avoiding parameter changes because there is, of course, no change of parameters in the original model.

For this reason, we only need to define $N_f = 6$ tile templates for this facade. Note that if we set all thresholds to 0, we must use all these tiles, so we have to use all 6 tiles in this case. Figure 8.1(c) shows a possible result of the homogenization process with an intermediate resolution ϱ of $0.5m$ (meaning that the thresholds for the metric parameters are set to ϱ): In this case, for example, the tiles above the ground floor could be covered by a single tile, so two tiles could be saved.

Constraint 8.1

The first constraint ensures that the X and Y values indeed represent the lower left and upper right corners of a valid bounding box:

$$\begin{aligned} \forall f \in \{1, ..., N_f - 1\} : \; X_{min,f} &\leq X_{max,f}, \\ Y_{min,f} &\leq Y_{max,f}. \end{aligned}$$

We define the span of a tile to be the 2D interval covered by the tile:

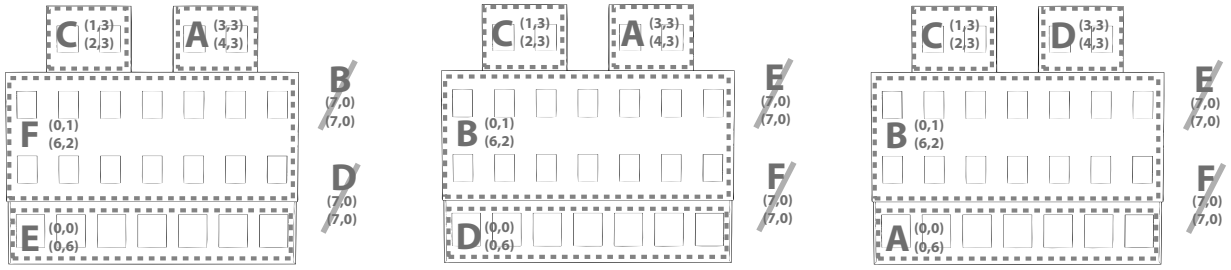
$$Span(f) := \{(i, j) \mid Y_{min,f} \leq i \leq Y_{max,f},\, X_{min,f} \leq j \leq X_{max,f}\}.$$

Each of the X and Y variables may assume values in the range of $\{0, ..., N_C\}$ and $\{0, ..., N_R\}$, respectively. This allows us to place tile templates outside the scope of the original facade array in order to dispose of the templates that were not needed in the tiling: While we have to assign valid X and Y for these templates as well, they must not interfere with the rest of the tiling of the facade.

We can see this in figure 8.1(c): the two tile templates E and F were not needed for the tiling, but their corner coordinates still have to be assigned valid values. This is achieved by allowing the X values to be set to the additional value N_C (7 in this case); the two unused templates E and F will have corner indices that keep them outside the original cell array.

In the example, both templates were placed to cover only the cell $(7, 0)$. Note that any placement of the tiles outside the original array would have been valid. This introduces ambiguities for valid solutions, which may be one of the reasons for the poor performance of the template-based approach.

In order to keep track of the facade tiles we did actually use, we define binary variables $A_f, f \in \{0, ..., N_f - 1\}$ that are 1 if tile f is used in the final tiling and 0 if it is not used.



(a) A tiling violating constraint 8.2: Template C is instantiated, but not template B.

(b) A tiling satisfying constraint 8.2 but violating constraint 8.3: Template A is located to the right of template C.

(c) A valid optimum tiling satisfying both constraints.

Abbildung 8.2.: Partial disambiguation of the optimum tiling.

Constraint 8.2

In order to avoid ambiguities that would unnecessarily slow down the solver and in order to obtain more concise solutions, we want to use only the first k tile placeholders if there are k tiles in the solution:

$$\forall f \in \{1, ..., N_f - 1\} : \; A_f \leq A_{f-1},$$

meaning that a template f can only be used if its predecessor $f - 1$ has been used as well. If we assume the templates in our example in figure 8.2 to be ordered in alphabetical order, this means that the tiling in 8.2(a) violates this constraint in multiple cases: A tiling satisfying these constraints would only use the templates A–D like the one in figure 8.2(b). There is, however, still a lot of ambiguity left because any permutation of the different template assignments to the resulting tiles would be valid. In our example, this means that there are $4! = 24$ valid assignments left.

Constraint 8.3

For this reason, we define constraints that order the resulting tiles from left to right:

$$\forall f \in \{1, ..., N_f - 1\} : \; X_{min,f} \geq X_{min,f-1},$$

so the left side of tile f may not be located on the right of tile $f - 1$. Note that there is still an ambiguity because the order of two tiles that are located on top of each other is still undefined. This may be another reason why the performance of this approach is inferior to that of the graph-based approach given below.

A true ordering of the lower left indices of the tiles could have solved this issue; this is, however, not included in the current system. One way to easily achive such an ordering is an „x before y“ ordering in

which an additional constraint set $\forall f \in \{1, ..., N_f - 1\} : (X_{min,f} = X_{min,f-1}) \Rightarrow (Y_{min,f} \geq Y_{min,f-1})$ is added as an extension of the constraints.

Using this constraint leaves only two valid assignments of the instantiated tiles in our example in figure 8.2: The combination given in figure 8.2(c) and the same combination with templates A and B swapped.

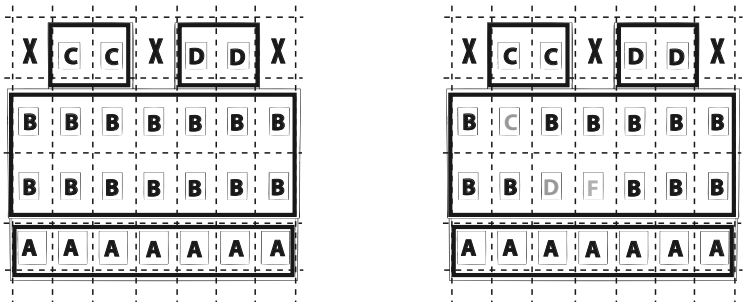
Objective 8.1

The primary objective in our optimization process is to produce a minimum number of tiles. This is easily encoded in an objective term because we explicitly keep track of the number of resulting tiles:

$$\text{MIN} \quad \left(W_{\#f} \sum_{f=0}^{N_f-1} A_f \right),$$

where $W_{\#f}$ is a constant factor that expresses the relative weight of the number of tiles compared to the other objective terms.

Now we have to make sure that no tiles overlap: For each grid cell (i, j) of our facade array, we define the Boolean variables $F_{(i,j),f}$ that are TRUE (have value 1) if the cell (i, j) is covered by tile f and FALSE otherwise. In order to avoid overlaps, we introduce the following group of constraints:



(a) A valid distribution of TRUE $F_{(i,j),f}$ assignments.

(b) A distribution violating constraints 8.5 and 8.6.

Abbildung 8.3.: Distribution of TRUE *values of $F_{(i,j),f}$ variable assignments for the facade of figure 8.1.*

Constraint 8.4

$$\forall (i, j) \in \text{Cells} :$$
$$\text{BG}(i, j) + \sum_{f=0}^{N_f-1} F_{(i,j),f} = 1,$$

where the short form $\forall (i, j) \in \text{Cells}$ stands for $i \in \{0, ..., N_R - 1\} \forall j \in \{0, ..., N_C - 1\}$, so the index i iterates over the rows while the index j iterates over the columns of the facade array. This convention will be used for the rest of this section.

The term $\text{BG}(i, j)$ is TRUE (adds 1 to the left hand side) if cell (i, j) is a background cell, so in this case, the cell must not be covered by a tile; otherwise, it must be covered by exactly one tile.

For the example in figure 8.3, this means that there are 6 $F_{(i,j),f}$ ($F_{(i,j),A}$–$F_{(i,j),F}$) variables for each cell (i, j) of which exactly one has to be TRUE if (i, j) is part of the facade and none is TRUE if (i, j) is a background cell (denoted by the black crosses in the figure). Note that under this constraint, the assignment shown in figure 8.3(b) is valid.

Constraint 8.5

Additionally, only active templates may cover a cell:

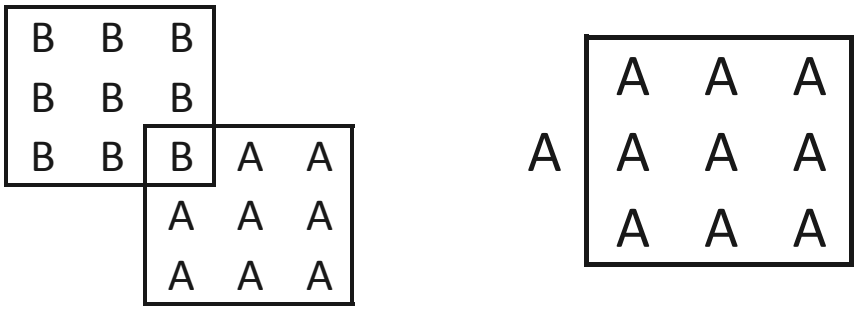
$$\forall (i,j) \in \text{Cells}, f \in \{0,...,N_f - 1\} : \\ F_{(i,j),f} \leq A_f.$$

This constraint is violated by the assignment of cell $(3,1)$ in figure 8.3(b): Template F is not in use. The other assignments in figure 8.3(b) are still valid.

Constraint 8.6

In order to establish the link between the span of the different tiles and the cells they cover, we have to make sure that all cells (i,j) within the span of a given tile f are covered by that tile ($F_{(i,j),f} = 1$) and that if $F_{(i,j),f} = 1$ for a given cell (i,j), then the span of f must include this cell:

$$\forall (i,j) \in \text{Cells}, f \in \{0,...,N_f - 1\} : \\ F_{(i,j),f} \Leftrightarrow (i,j) \in Span(f).$$



(a) A valid constellation if only $F_{(i,j),f} \Rightarrow (i,j) \in Span(f)$ is enforced.

(b) A valid constellation if only $F_{(i,j),f} \Leftarrow (i,j) \in Span(f)$ is enforced.

Abbildung 8.4.: Why both directions of $F_{(i,j),f} \Leftrightarrow (i,j) \in Span(f)$ are relevant in constraint 8.6.

Figure 8.4 illustrates why both directions in the equivalence relation in this constraint are relevant: If we only enforce $F_{(i,j),f} \Rightarrow (i,j) \in Span(f)$, then we make sure that no cells outside the span of a template may be associated with this template, but we do not prevent the spans of the facade templates from overlapping as shown figure 8.4(a). If we only $F_{(i,j),f} \Leftarrow (i,j) \in Span(f)$, then all cells within the span of a template have to be assigned to this template (preventing overlapping cells), but cells outside the span of the template may be associated with the template as well as figure 8.4(b) shows.

In order to feed this constraint to a MIP solver, we first convert the expression $(i,j) \in Span(f)$ into a set of linear constraints. In a first step, we define four binary indicator variables (a–d) for the four parts of the definition of $Span(f)$:

$$\begin{aligned} a &:= (i \geq Y_{min,f}) : && \text{cell } (i,j) \text{ is not located } \textit{below} \text{ Span}(f). \\ b &:= (i \leq Y_{max,f}) : && \text{cell } (i,j) \text{ is not located } \textit{above} \text{ Span}(f). \\ c &:= (j \geq X_{min,f}) : && \text{cell } (i,j) \text{ is not located to the } \textit{left} \text{ of Span}(f). \\ d &:= (j \leq X_{max,f}) : && \text{cell } (i,j) \text{ is not located to the } \textit{right} \text{ of Span}(f). \end{aligned}$$

The four indicator variables a–d are all TRUE if and only if cell (i,j) is contained in Span(f). Figure 8.5 illustrates this fact: If the cell is located neither below or above nor to the left or to the right of the

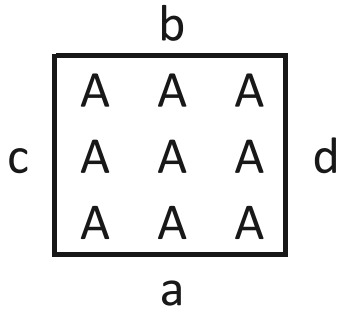


Abbildung 8.5.: Illustration of the indicator variables a–d: The letters indicate positions prevented by the indicators.

interval defined by Span(f), then it must be within this area and vice versa. More formally, we can write: $(i, j) \in \text{Span}(f) \Leftrightarrow a \wedge b \wedge c \wedge d$.

For the auxiliary variable a, we will explain the conversion of the definition given above to a set of linear constraints in more detail. The definition $a := (i \geq Y_{min,f})$ of a can be broken down into the following two constraints:

$$
\begin{aligned}
a \Leftrightarrow (i \geq Y_{min,f}) : & \\
& (i \geq Y_{min,f}) \Rightarrow a \;\rightarrow\; a\,M_i \geq i - Y_{min,f} + 0.1 \\
& a \Rightarrow (i \geq Y_{min,f}) \;\rightarrow\; i + M_i \geq Y_{min,f} + a\,M_i,
\end{aligned}
$$

where M_i is a number that is larger than any difference between row indices in our facade array, so we set $M_i := N_R + 1$. Note that any larger value for M_i would lead to the same result, but the smaller a "big-M" value is chosen, the more stable and the faster most MIP solvers will find a solution.

In the first line, we make sure that a must be TRUE if $i \geq Y_{min,f}$: In this case, the right side of the first line is larger than 0 (if $i = Y_{min,f}$, then it is 0.1) but smaller than M_i, so if $a = 1$, then the constraint is fulfilled. If $i < Y_{min,f}$, then the right side of the first row is smaller than 0 (-0.9 if $i = Y_{min,f} - 1$), and the constraint is relaxed no matter if a is TRUE ($a = 1$) or FALSE.

So in order to complete the definition of a, we have to enforce the opposite direction as well. This is done in the second line: If a is TRUE, then the terms M_i on the left side and $a\,M_i$ on the right side cancel, and the equation becomes $i \geq Y_{min,f}$. If a is FALSE, then we obtain $i + M_i \geq Y_{min,f}$, so the constraint is relaxed. Putting both lines together, we obtain $a \Leftrightarrow (i \geq Y_{min,f})$.

The rest of the conversions work in similar way:

$$
\begin{aligned}
b \Leftrightarrow (i \leq Y_{max,f}) : & \\
& (i \leq Y_{max,f}) \Rightarrow b \;\rightarrow\; b\,M_i \geq Y_{max,f} - i + 0.1 \\
& b \Rightarrow (i \leq Y_{max,f}) \;\rightarrow\; i + b\,M_i \leq Y_{max,f} + M_i \\
c \Leftrightarrow (j \geq X_{min,f}) : & \\
& (j \geq X_{min,f}) \Rightarrow c \;\rightarrow\; c\,M_j \geq j - X_{min,f} + 0.1 \\
& c \Rightarrow (j \geq X_{min,f}) \;\rightarrow\; j + M_j \geq X_{min,f} + c\,M_j \\
d \Leftrightarrow (j \leq X_{max,f}) : & \\
& (j \leq X_{max,f}) \Rightarrow d \;\rightarrow\; d\,M_j \geq X_{max,f} - j + 0.1 \\
& d \Rightarrow (j \leq X_{max,f}) \;\rightarrow\; j + d\,M_j \leq X_{max,f} + M_j,
\end{aligned}
$$

where $M_j := N_C + 1$ is a "big-M" value for maximum column value differences similar to the value M_i for the rows.

The constraint 8.6 is then equivalent to

$$F_{(i,j),f} \Leftrightarrow (i,j) \in Span(f) \Leftrightarrow a \wedge b \wedge c \wedge d.$$

which can be expressed in the form of a linear set of constraints:

$$\begin{aligned} F_{(i,j),f} \Leftrightarrow (a \wedge b \wedge c \wedge d): \\ (a \wedge b \wedge c \wedge d) \Rightarrow F_{(i,j),f} \ &\rightarrow \ F_{(i,j),f} \geq (a+b+c+d) - 3.5 \\ F_{(i,j),f} \Rightarrow (a \wedge b \wedge c \wedge d) \ &\rightarrow \ 3.5 \cdot F_{(i,j),f} \leq (a+b+c+d). \end{aligned}$$

If the variables a through d are TRUE, then the right side in the first line evaluates to 0.5, so $F_{(i,j),f}$ has to be TRUE to satisfy this constraint. Otherwise the left side is smaller than 0 and the constraint is relaxed. Conversely, if $F_{(i,j),f}$ is TRUE, the left side in the second line is 3.5, so all variables a through d have to be TRUE to satisfy the constraint. If $F_{(i,j),f}$ is FALSE, the left side of the line is 0 and the constraint is relaxed.

The constraints given above allow us to model the structure of the tiling. In the following constraints and objective terms, the parts of the MIP model are introduced that are defined by the data. For each facade grid cell, the facade piece represented by the cell is modeled by a list of parameter values. These values may describe arbitrary properties of the object contained in the cell. Common parameters are, for example, the width and height of features like doors or windows in the cell, also the color (red, green and blue values of the color) of the background wall may be part of the parameter list.

For each generic parameter p in the set P of parameters, a difference threshold Δ_p and a weight W_p are given. For special parameters like the width and height of the cell itself, there are additional constraints: The width of all cells in a column and the height of all cells in a row should, for example, be identical. The features within a cell should also not protrude across the border of the cell: The width plus the horizontal offset of a feature should, for example, not exceed the width of the cell holding it. These special parameters and constraints are not modeled in this first approach but they are added in the more efficient approach described in the next section.

For each tile placeholder f, there are continuous variables $V_{f,p}$ that store the values of the parameters $p \in P$. The difference between this value $V_{f,p}$ to the original value $v_{(i,j),p}$ in each cell (i,j) covered by tile f must not be larger than Δ_p for each property p.

In order to check this constraint, we must establish the absolute value of this difference. A problem in this context is that we do not know by which tile a given cell will be covered. For this reason, we have to introduce variables for the differences $D_{(i,j),f,p}$ for all tile placeholders f in each cell (i,j) and to set these differences to 0 if cell (i,j) is not covered by tile f because otherwise there would be conflicts in the "big-M" terms in the filter of the active tile.

Constraint 8.7

We define the difference $D_{(i,j),f,p}$ to be the absolute value of the difference between $v_{(i,j),p}$ and $V_{f,p}$ if cell (i,j) is covered by f and 0 otherwise:

$$D_{(i,j),f,p} := \begin{cases} |v_{(i,j),p} - V_{f,p}|, & \text{if } (i,j) \text{ is covered by } f \ (F_{(i,j),f} = 1) \\ 0, & \text{otherwise.} \end{cases}$$

This can be written as the following set of linear constraints:

$$\begin{aligned} \forall (i,j) \in \text{Cells}, \forall f \in \{0,...,N_f - 1\}, \forall p \in P: \\ D_{(i,j),f,p} \geq v_{(i,j),p} - V_{f,p} - (1 - F_{(i,j),f}) M_p \\ D_{(i,j),f,p} \geq -v_{(i,j),p} + V_{f,p} - (1 - F_{(i,j),f}) M_p, \end{aligned}$$

where M_p is a big-M constant greater than the greatest possible difference between values of the parameter p. Since we do not want the parameters of the new features to be outside the range of the original parameters (the optimization is designed to give intermediate values resulting from least squares adjustment), M_p can easily be calculated as the difference between the biggest and smallest values of parameter p in the original data set.

The two lines in the linear form represent the absolute value of the difference $|v_{(i,j),p} - V_{f,p}|$ in the original formula. By default, all variables have a lower bound of 0 if not stated otherwise for most MIP solvers. This is assumed here as well, so $D_{(i,j),f,p}$ will never be less than 0 even if the right side of both lines is smaller than 0. This will happen if $F_{(i,j),f}$=FALSE because in this case, $(1 - F_{(i,j),f})M_p$ equals M_p which is, by definition, bigger than $|v_{(i,j),p} - V_{f,p}|$. Otherwise (if $F_{(i,j),f}$=TRUE), the term $(1 - F_{(i,j),f})M_p$ evaluates to 0 and disappears. In this case, the two lines together form the constraint $D_{(i,j),f,p} \geq |v_{(i,j),p} - V_{f,p}|$ because one line will evaluate to $D_{(i,j),f,p} \geq |v_{(i,j),p} - V_{f,p}|$ and the other one to $D_{(i,j),f,p} \geq -|v_{(i,j),p} - V_{f,p}|$.

Note that we only defined lower bounds for $D_{(i,j),f,p}$. Because this variable is only going to be used in contexts where a minimum value is desired (positive occurrence in a MIN objective term and upper bounds in constraints), this is sufficient. Otherwise, we would have had to define constraints for lower bounds in a similar way.

Constraint 8.8

We want to ensure that the (absolute value of the) difference between the original value and the value in the new tile is never larger than Δ_p for any parameter p. Since we already have this value, this constraint is easily established:

$$\forall (i,j) \in \text{Cells}, \forall f \in \{0, ..., N_f - 1\}, \forall p \in P : \\ D_{(i,j),f,p} \leq \Delta_p.$$

Note that this will also check the values for $D_{(i,j),f,p}$ if cell (i,j) is not covered by tile f, but since these values are 0 in this case, this is not a problem. Because we do not know in advance which one of the values $D_{(i,j),f,p}$ will be the one carrying the important value, we have to check them all.

Objective 8.2

Different parameters may be of different importance to a given application. For this reason, it makes sense to set weights W_p that regulate how strongly differences in the parameters will be penalized in relation the other parameters. Since we want the resulting parameter values to be determined by a least squares adjustment, we will minimize the sum of the squares of the differences:

$$\text{MIN} \left(\sum_{(i,j) \in Cells} \sum_{f=0}^{N_f - 1} \sum_{p \in P} W_p \, D^2_{(i,j),f,p}. \right)$$

In order to wrap up this section, all constraints and objective function terms introduced in this section are listed in the following table. These constrains define a proper tiling and homogenization of the facade structure because they enforce that all cells belong to a tile, that exactly the cells within the span of a tile belong to the tile, and that the parameters of all cells within the tiles are identical after the process.

The set of valid solutions and the evaluation of these solution are identical to those of the graph-based MIP representation introduced in the next section, but the CPLEX solver needed a significantly smaller amount of resources for the graph-based version. In the results section of this chapter, it is shown for a sufficiently example that the constraints for the homogenization of the cell parameters lead to a sensible partitioning of the facade.

Constraints:

C 8.1	Variables $X_{min\|max}$ and $Y_{min\|max}$ define a valid interval:	$\forall f \in \{1, ..., N_f - 1\} : X_{min,f} \leq X_{max,f}, \quad Y_{min,f} \leq Y_{max,f}.$
C 8.2	Use only the first k tiles:	$\forall f \in \{1, ..., N_f - 1\} : A_f \leq A_{f-1}$
C 8.3	Order tiles from left to right:	$\forall f \in \{1, ..., N_f - 1\} : X_{min,f} \geq X_{min,f-1}$
C 8.4	A cell cannot be covered by more than one template:	$\forall (i,j) \in \text{Cells} : \text{BG}(i,j) + \sum_{f=0}^{N_f-1} F_{(i,j),f} = 1,$
C 8.5	Only active temapltes may cover a cell:	$\forall (i,j) \in \text{Cells}, f \in \{0, ..., N_f - 1\} : F_{(i,j),f} \leq A_f.$
C 8.6	A covered cell must be contained in the span of the covering template:	$F_{(i,j),f} \Leftrightarrow (i,j) \in Span(f) \Leftrightarrow (i \geq Y_{min,f}) \wedge (i \leq Y_{max,f}) \wedge (j \geq X_{min,f}) \wedge (j \leq X_{max,f}).$
C 8.7	Define variable $D_{(i,j),f,p}$ for change of parameter value due to homogenization:	$D_{(i,j),f,p} := \begin{cases} \lvert v_{(i,j),p} - V_{f,p} \rvert, & \text{if } (i,j) \text{ is covered by } f \ (F_{(i,j),f} = 1) \\ 0, & \text{otherwise.} \end{cases}$
C 8.8	Threshold for $D_{(i,j),f,p}$:	$\forall (i,j) \in \text{Cells}, \forall f \in \{0, ..., N_f - 1\}, \forall p \in P : D_{(i,j),f,p} \leq \Delta_p.$

Objective function terms:

O 8.1	Minimize the number of active templates:	MIN $\left(W_{\#f} \sum_{f=0}^{N_f-1} A_f \right)$
O 8.2	Minimize the weighted sum of squared parameter changes:	MIN $\left(\sum_{(i,j) \in Cells} \sum_{f=0}^{N_f-1} \sum_{p \in P} W_p \, D^2_{(i,j),f,p} \right)$

8.2. A graph-based MIP model for facade homogenization

The approach introduced in this section addresses the same facade homogenization problem as the direct approach described above (section 8.1), but the representation is slightly different; it is rather similar to the graph-based approaches used in Haunert (2009) for the aggregation of land parcels in map generalization and in chapter 7.1 for the aggregation of LoD 1 building models.

As in these cases, one member is defined as the representative (also referred to as *center* in Haunert (2009) and chapter 7.1) of a unit in the result. In the case of our facade homogenization problem, this means that one cell will be designated as the representative of a given tile. Note that this representative does not have to be a kind of geometric center of the tile as the term *center* suggests – in fact, we will define the center to be the lower left corner of each tile to avoid ambiguities.

Similar to the representation of cluster membership in section 7.1, We define a set of variables $X_{(a,b),(i,j)}$ that are TRUE if cell (i,j) is assigned to center (a,b). In our case, the underlying neighborhood graph is implicitly given by the logical grid structure of the facade.

Because the center has no other function than to represent the tile, the constraints defined in the rest of this section are designed in such a way that the center of each tile is the lower left cell in the tile. This definition has the advantage that it reduces the ambiguity of a valid solution considerably because there is only a single possible center for a given tile and that it simplifies the constraints for ensuring the rectangular shape of the clusters significantly. Additionally, it saves variables because the variable $X_{(a,b),(i,j)}$ only has to be defined for $i \geq a$ and $j \geq b$.

For this reason, iterating over all variables would be denoted in the form: $\forall(a,b) \in \text{Cells}, \forall i \in \{a, \ldots, N_R - 1\}, \forall j \in \{b, \ldots, N_C - 1\}$). As a short form for this expression we write: $\forall(a,b), (i \geq a, j \geq b) \in \text{Cells}$, meaning that i and j will assume all valid values in the given array greater than a or b, respectively.

Note that this implementation is a completely independent alternative to the approach introduced in section 8.1 despite the continued numbering of the constraints.

Constraint 8.9

As in Haunert (2009) and chapter 7.1, the first constraint states that cells can only be assigned to centers where a center is a cell that is assigned to itself:

$$\forall(a,b), (i \geq a, j \geq b) \in \text{Cells} : X_{(a,b),(i,j)} \leq X_{(a,b),(a,b)}.$$

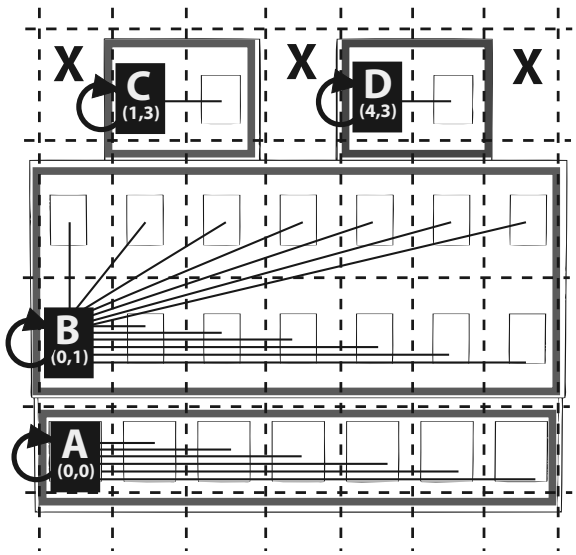


Abbildung 8.6.: Visualization of the TRUE $X_{(a,b)(i,j)}$ *variables for a tiling of a facade.*

Figure 8.6 shows the $X_{(a,b)(i,j)}$ variables that were assigned the value TRUE for a tiling of our illustration facade. The black lines connect the cells in the tiles to their centers; the circular lines at the centers illustrate that for each center $c = (a,b)$, the variable $X_{c,c}$ is TRUE. Constraint 8.9 ensures that a cell can only point to a center: Each black line terminates in a center in figure 8.6.

The letters in the centers are of no consequence for the graph-based approach presented in this section; they illustrate that the tiling shown in this figure is equivalent to the one used in the example in figure 8.2 in the illustration of the template-based approach introduced in the last section (section 8.1).

Constraint 8.10

Additionally, all cells will belong to exactly one tile (center) if (i,j) is part of the facade and to none if it is a background cell:

$$\forall(i,j) \in \text{Cells} : \sum_{a=0}^{i} \sum_{b=0}^{j} X_{(a,b),(i,j)} = 1 - \text{BG}(i,j),$$

where $\text{BG}(i,j)$ is TRUE(=1) if (i,j) is a background cell and FALSE(=0) if not. The black crosses in the

unoccupied cells in figure 8.6 illustrate that none of the X variables is set to TRUE for the background cells. For the other cells, this constraint makes sure that only one of the outgoing X variables is set: there is only one black line from each occupied cell in figure 8.6.

Objective 8.3

Now we can define the objective of using a minimum number of tiles:

$$\text{MIN} \quad \left(W_{\#f} \sum_{(a,b)\in \text{Cells}} X_{(a,b),(a,b)} \right),$$

where $W_{\#f}$ is a constant factor that expresses the relative weight of the number of tiles compared to the other objective terms. The sum counts the number of centers because a center is defined as a cell assigned to itself.

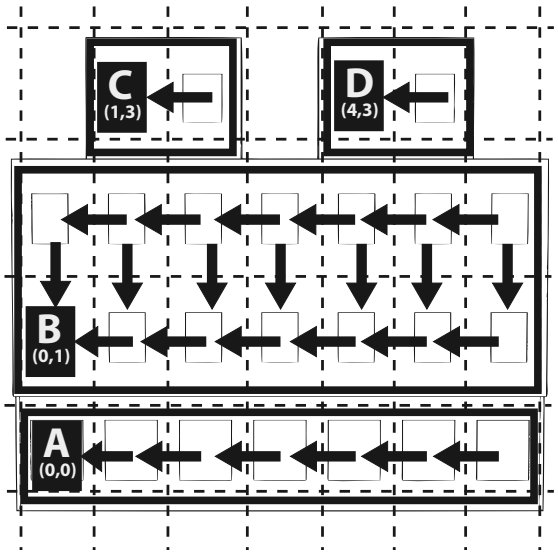


Abbildung 8.7.: Constraint set 8.11 for the tiling in figure 8.6.

Constraint 8.11

The following constraint set ensures that the center of a tile will be its lower left corner and that the rectangle between a given tile and its associated center must be part of the tile defined by the center:

$$\begin{aligned} \forall (a,b), (i \geq a, j \geq b) \in \text{Cells}: & \\ X_{(a,b),(i,j)} &\leq X_{(a,b),(i-1,j)} \quad \text{if } i > a \\ X_{(a,b),(i,j)} &\leq X_{(a,b),(i,j-1)} \quad \text{if } j > b. \end{aligned}$$

A cell (i, j) can only be assigned to center (a, b) if its predecessors in x (second constraint) and y (first constraint) direction are also assigned to the same center. Note that if cell (i, j) is already in the same row or column as the center (a, b), the corresponding predecessor must not be forced to be assigned to the same center. For this reason, the constraints for the predecessors in y direction are only defined if $i > a$ and the constraints for the predecessors in x direction are only defined if $j > b$.

The transitive effect of these constraints on the predecessors ensures that for all tiles (i, j) all cells in the rectangle spanned by (a, b) and (i, j) must be assigned to (a, b) if (i, j) is assigned to (a, b). Figure 8.7 illustrates this effect for a valid tiling: The horizontal arrows show the implications enforced by the first constraint (in x direction), the vertical arrows the implications in the y direction. Since the center is the bottom left cell in the tile, all arrows have to converge towards the center and no predecessor of a cell in the tile can be left out.

This does, however, not yet ensure that each tile will be a rectangle.

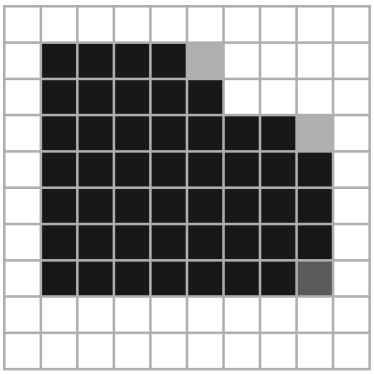


Abbildung 8.8.: Valid non-rectangular tile under constraint 8.11.

Figure 8.8 shows that in constraint 8.11, there is a "loophole" that allows non-rectangular tiles: In the upper right corners, fragments may be cut out of the rectangle without violating the constraint. The light gray cells that define the two basic rectangles of the tile shown in the figure may also be left out; the dark gray cell in the bottom line, for example, must, however, be part of the tile because it is "covered" by the cell above it.

Constraint 8.12

In oder to ensure that the tiles are rectangles, we define a constraint that works in the opposite direction compared to constraint 8.11:

$$\begin{aligned}\forall (a,b), (i > a, j > b) \in \text{Cells}: \\ X_{(a,b),(i-1,j)} \wedge X_{(a,b),(i,j-1)} \Rightarrow X_{(a,b),(i,j)} \\ \longrightarrow X_{(a,b),(i,j)} \geq X_{(a,b),(i-1,j)} + X_{(a,b),(i,j-1)} - 1.\end{aligned}$$

This closes the rectangle in the direction of its upper right corner: If both predecessors of a cell in x and y direction point to the same center, the cell itself has to point to this center as well. Note the ">" in the $\forall$-clause: This constraint will only concern cells that are located neither on the same row nor on the same column of a given possible center. This constraint only works in context with constraint 8.11 because it only performs the closing in the upper right direction. A more generic implementation in which the center may be an arbitrary member of the cluster of the cluster would require a more sophisticated set of constraints.

In the linear form given in the second row, the right side of the constraint is 1 if both predecessors point to the same center (a, b), so $X_{(a,b),(i,j)}$ has to be TRUE as well in this case. Otherwise, the right side is 0 or -1, so the constraint is relaxed.

The parameters are handled by assigning tile parameter values to the centers of the tiles. Since we do not know which cells are going to be centers, there have to be parameter placeholders $V_{(i,j),p}$ for each parameter p for all cells (i, j). For cells that are not centers, we force this variable to equal the value of the center and use it to calculate the difference $D_{(i,j),p}$ between the original value $v_{(i,j),p}$ of parameter p and the new value $V_{(i,j),p}$ defined by the covering tile for each tile (i, j).

Constraint 8.13

First, we make sure that all cells in a tile have the same value for each parameter:

$$\begin{aligned}\forall(a,b),(i>a,j>b) \in \text{Cells}, \forall p \in P: \\ V_{(i,j),p} \geq V_{(a,b),p} - M_p \left(1 - X_{(a,b),(i,j)}\right) \\ V_{(i,j),p} \leq V_{(a,b),p} + M_p \left(1 - X_{(a,b),(i,j)}\right),\end{aligned}$$

where M_p is an adequate "big-M" value for parameter p (see section 8.1), for example the maximum difference between two values of p in the original facade data set.

If (i,j) is assigned to (a,b), then the term $M_p \left(1 - X_{(a,b),(i,j)}\right)$ becomes 0 and the two inequalities can be merged to the equation $V_{(i,j),p} = V_{(a,b),p}$. For all other possible centers (a,b), the constraints are relaxed because M_p is, by definition, greater than any possible difference between $V_{(i,j),p}$ and $V_{(a,b),p}$, and $M_p \left(1 - X_{(a,b),(i,j)}\right)$ evaluates to M_p in this case.

So all cells within a tile have the same values for each parameter as the center which means, by the transitivity of the "=" relation, that all cells have the same value.

Constraint 8.14

Now we set the variables $D_{(i,j),p}$ to hold the (absolute value of the) difference between the value $V_{(i,j),p}$ of parameter p in the tile and the original value $v_{(i,j),p}$ of p in cell (i,j):

$$\begin{aligned}\forall(i,j) \in \text{Cells}, \forall p \in P: D_{(i,j),p} &\geq \left|V_{(i,j),p} - v_{(i,j),p}\right| \\ \longrightarrow D_{(i,j),p} &\geq V_{(i,j),p} - v_{(i,j),p} \\ D_{(i,j),p} &\geq -V_{(i,j),p} + v_{(i,j),p},\end{aligned}$$

Since we will use the variables $D_{(i,j),p}$ only in contexts where a minimum value is desired, we do not need to define constraints that enforce an upper bound on $D_{(i,j),p}$.

Constraint 8.15

Now we can easily implement the constraint of bounding the homogenization error for each parameter in all cells:

$$\forall(i,j) \in \text{Cells}, \forall p \in P: D_{(i,j),p} \leq \Delta_p$$

Objective 8.4

Finally, we add the penalties for the parameter errors to the objective function:

$$\text{MIN} \left(\sum_{(i,j) \in Cells} \sum_{p \in P} W_p \, D^2_{(i,j),p}. \right)$$

This final objective term ensures that the resulting value for each parameter p for a given tile is the average of the values in the cells of the tile – if the difference threshold allows the parameter to assume this value.

In order to wrap up this section, all constraints and objective function terms introduced in this section are listed in the following table.

Constraints:

C 8.9	Cells can only be assigned to centers:	$\forall (a,b), (i \geq a, j \geq b) \in \text{Cells} : X_{(a,b),(i,j)} \leq X_{(a,b),(a,b)}$
C 8.10	A cell is assigned to exactly one center:	$\forall (i,j) \in \text{Cells} : \sum_{a=0}^{i} \sum_{b=0}^{j} X_{(a,b),(i,j)} = 1 - \text{BG}(i,j)$
C 8.11	Cells in rectangle between a cell and its associated center must also be assigned to the center:	$\forall (a,b), (i \geq a, j \geq b) \in \text{Cells} :$ $X_{(a,b),(i,j)} \leq X_{(a,b),(i-1,j)}$ if $i > a$ $X_{(a,b),(i,j)} \leq X_{(a,b),(i,j-1)}$ if $j > b$.
C 8.12	Close the rectangle to upper right:	$\forall (a,b), (i > a, j > b) \in \text{Cells} :$ $X_{(a,b),(i-1,j)} \wedge X_{(a,b),(i,j-1)} \Rightarrow X_{(a,b),(i,j)}$ $\longrightarrow X_{(a,b),(i,j)} \geq X_{(a,b),(i-1,j)} + X_{(a,b),(i,j-1)} - 1.$
C 8.13	Ensure homogeneous parameter values within the clusters:	$\forall (a,b), (i > a, j > b) \in \text{Cells}, \forall p \in P :$ $V_{(i,j),p} \geq V_{(a,b),p} - M_p \left(1 - X_{(a,b),(i,j)}\right)$ $V_{(i,j),p} \leq V_{(a,b),p} + M_p \left(1 - X_{(a,b),(i,j)}\right),$
C 8.14	Define variable $D_{(i,j),p}$ for change of parameter value due to homogenization:	$\forall (i,j) \in \text{Cells}, \forall p \in P :$ $D_{(i,j),p} \geq \left\lvert V_{(i,j),p} - v_{(i,j),p} \right\rvert$
C 8.15	Threshold for $D_{(i,j),p}$:	$\forall (i,j) \in \text{Cells}, \forall p \in P : D_{(i,j),p} \leq \Delta_p$

Objective function terms:

O 8.3	Minimize the number of clusters (centers):	MIN $\left(W_{\#f} \sum_{(a,b) \in \text{Cells}} X_{(a,b),(a,b)}\right)$
O 8.4	Minimize the weighted sum of squared parameter changes:	MIN $\left(\sum_{(i,j) \in Cells} \sum_{p \in P} W_p \, D^2_{(i,j),p}.\right)$

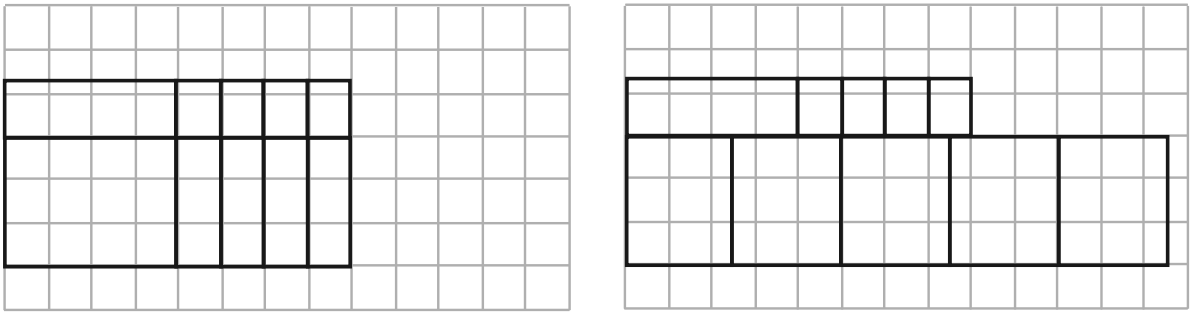
The objectives in this section are the same as in the last section, but the corresponding objective function terms are different because of the different MIP models.

8.3. Extensions

In the current version, the width and height of the cells and the features are treated like all other parameters for the cells. This can lead to cells with different widths in the same column or cells with different heights in the same row when the cells are assigned to different tiles.

Figure 8.9 illustrates this problem with the help of an example in which the effect is very strong: Assuming that the threshold for width changes is 1.5 units in the raster and all cells in the bottom row can be merged while those in the top row remain separated.

In this case, all five cells in the bottom row will have a width of 2.5 units after the homogenization while those in the top row still have their original widths of 4 units for the left and 1 unit for the other cells. The figure shows that this completely break the integrity of the grid structure because the cells in the columns cannot be aligned any more and because the overall with of the facade may change considerably – in the example, it is increased by 4.5 units.



(a) Original facade. (b) After homogenization.

Abbildung 8.9.: Different cell widths in the same column through homogenization.

The most straightforward approach to avoid this problem is to introduce additional constraints to ensure that all cells in a column have equal widths and all cells in a row have equal heights:

1. Equal column widths:

   $$\forall (i,j) \in \text{Cells}: \; V_{(i,j),w} = V_{(i,j+1),w}$$

   if (i,j) and $(i,j+1)$ are cells for which valid width values are defined; $V_{(a,b),w}$ stands for the value of the width parameter of the cell (a,b) in the optimization problem.

2. Equal row heights:

   $$\forall (i,j) \in \text{Cells}: \; V_{(i,j),h} = V_{(i+1,j),h}$$

   if (i,j) and $(i+1,j)$ are cells for which valid width values are defined; $V_{(i+1,j),h}$ is the value of the height parameter.

3. The total width of the facade across all columns may not change by more than ρ compared to the original situation:

   $$\forall i \in \text{Rows}: \left| \sum_{j \in \text{Cols}} CW_j - \sum_{j \in \text{Cols}} V_{(i,j),w} \right| \leq \rho,$$

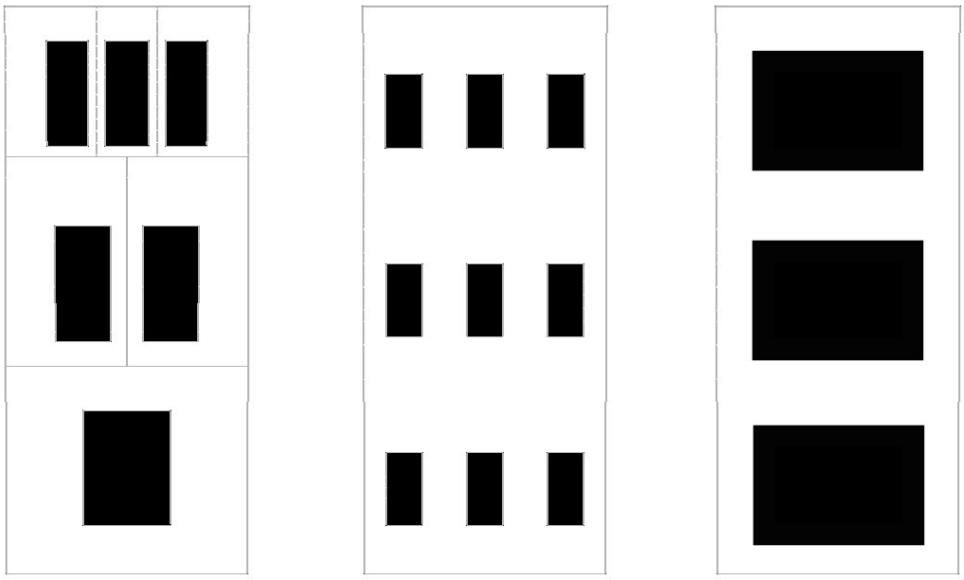
   where CW_j is the width of column j in the original data.

The small facade structure in figure 8.10 illustrates some of the shortcomings of this strict approach: It can only handle regular grids of cells. Obviously, such a situation is not given here because the features in the lower cells overlap features in the cells above them. If a model at a high resolution is requested, the strict constraint would be impossible to satisfy without changing the width of one of the cells by a value larger than ϱ which would mean a conflict between hard constraints.

In order to avoid this problem, only those cell borders that are already aligned are required to be aligned after the homogenization step. This is achieved by inserting dummy cell rows or columns that have a smaller number of cells than their neighbors. For the example, this means that one dummy cell has to be added in the middle and two cells have to the bottom row of windows.

After homogenization with suitably high value for ϱ, the cells in the aligned area can be merged, and features with parameters equal to those in the surrounding cells will appear as shown in figure 8.10(b). Such a homogenized grid can be used for typification in a next step.

The example also shows that homogenization and aggregation may, in combination, yield better results than applying just one of the two. In the example, the homogenization could only be applied because of the preceding aggregation of the windows in the rows.

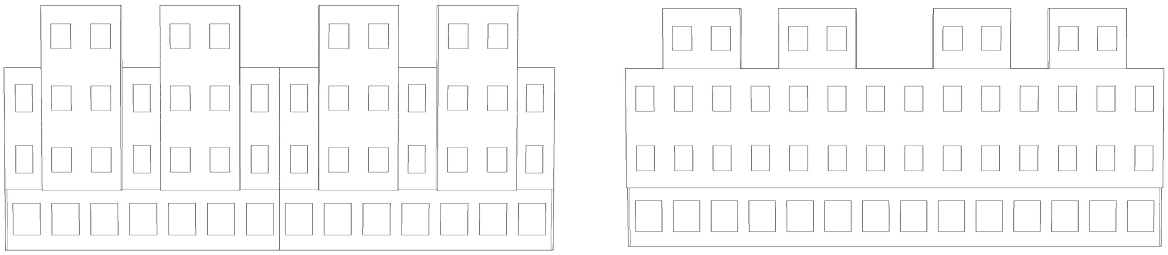


(a) Original facade. (b) After homogenization alone (current status). (c) After aggregation and homogenization.

Abbildung 8.10.: Aggregation and homogenization.

8.4. Examples and results

The effects of the homogenization process are illustrated with the help of examples in this section. We can see that the generalization process provides visually convincing results for simple as well as non-trivial setups like the string of adjacent townhouse facades shown in figure 8.15. In order to resolve issues that are identified in the course of this analysis, extensions to the method are proposed that are, however, not yet part of the current implementation.



(a) Original facade. (b) After homogenization.

Abbildung 8.11.: A typical townhouse facade.

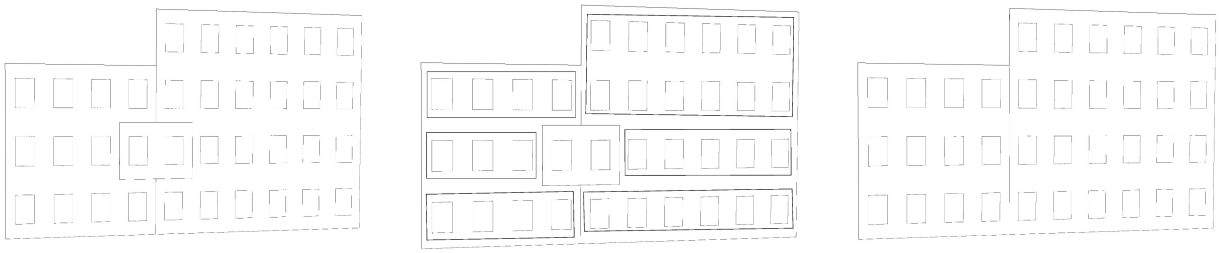
Figure 8.12 shows an example of a small facade part modeled after a facade from a typical townhouse. In this simple case, the protrusions were pushed back into the main facade plane and previously slightly different measures of the windows were set to identical values.

Because tiles are supposed to be rectangular, the windows in the gables of the dormers on top of the protrusions are not part of the main tile covering the upper floors. For this reason, they are left in their initial protruding position because changing the depth of a tile incurs a penalty in the optimization function.

In order to align as many tiles as possible with the main facade plane even if the facade elements could not be included in a tile on the main plane, we can introduce a term to the objective function that assigns a high benefit if the depth of a tile can be set to the same value as all its neighbors. This implies that all neighbors have to have the same depth for this benefit. In the case of our dangling protrusions, the only neighbor is the main plane, so this will have the desired effect.

Another option would be to allow tiles to cover empty cells that do not belong to the facade. In this case, the protrusion could have been merged with the main facade tile, and their depth would have been adjusted

together with the rest of the cells in the tile. Because the homogenization process is also a preparation step for typification, this would cause new problems because the windows in a typified layout of the facade might not fit into the gables. For this reason, holes in the tiles are not modeled in the homogenization process.



(a) Original facade. (b) A valid tiling of the facade. (c) After homogenization.

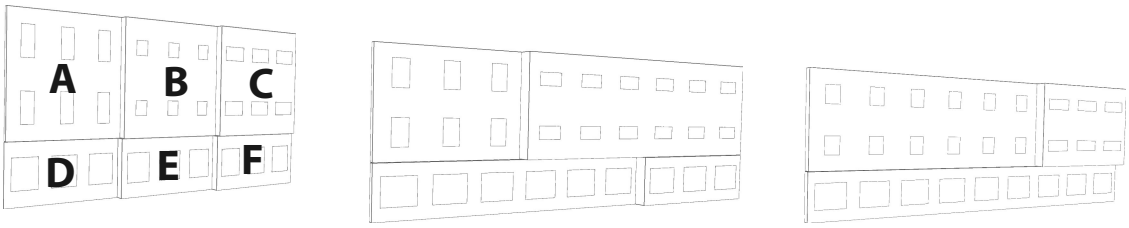
Abbildung 8.12.: Split of an area with identical parameter values.

The simple example in figure 8.12 shows that it is necessary that the homogenization algorithm has the flexibility to split areas with homogeneous parameters. Had this not been possible, the result would have had to consist of at least seven tiles – the tiles needed to form the original as shown in figure 8.12(b) – instead of the two that can be formed by splitting the homogeneous region in the center.

Note that in the figure, the original situation as shown in figure 8.12(a) is not a valid tiling because the central tile overlaps the others. For the graph-based approach, this is not a problem because it uses only the map containing the parameter values. The direct MIP approach also relies only on the map for its decisions how to distribute the samples.

It could, however, run into trouble because the number of templates to be distributed would be limited to three if the overlapping initial tiles would be given to it as input. In this special case, it would have worked because only two tiles are needed to cover the facade.

Had it not been possible to split the central initial tile, the seven templates needed to cover the facade would not have been available. The unavailability of the templates would have been due to the illegal input; the fact that they would have been necessary would have been due to the impossibility to split the central tile.



(a) Original facade. (b) Facade after first iteration. (c) Facade after second iteration (final).

Abbildung 8.13.: Homogenization sequence for LOW *penalty for the width and* HIGH *for the other variables.*

In appendix C, the impact of different weights for different parameters is discussed at length for an example. For this example, the homogenization process was repeated until no more simplification could be achieved.

Figure 8.13 shows the homogenization sequence for the case in which the differences of the width of a facade part received a LOW penalty (in relation to the others) while the other parameters (width and height of the windows) had a HIGH penalty for changes in the homogenization process.

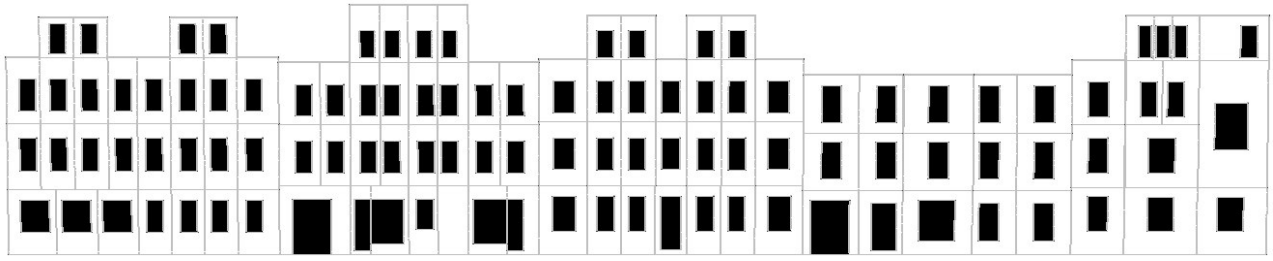
The example was constructed in order to show that different weights for the parameters can be used to guide the generalization process. In the case shown in figure 8.13, for example, the LOW penalty for width and the the HIGH penalty for height changes means that regions B and C are merged first.

Note that the simplification of the top and bottom rows are independent in this example, and the width and height parameters as well as the depth differences between regions D and E and regions E and F are equal. For this reason, it can happen that the mergings of the regions in the top and bottom rows are not aligned. In the case shown in figure 8.13, one would, for example, intuitively have merged regions E and F rather than regions D and E in order to achieve a more homogeneous appearance.

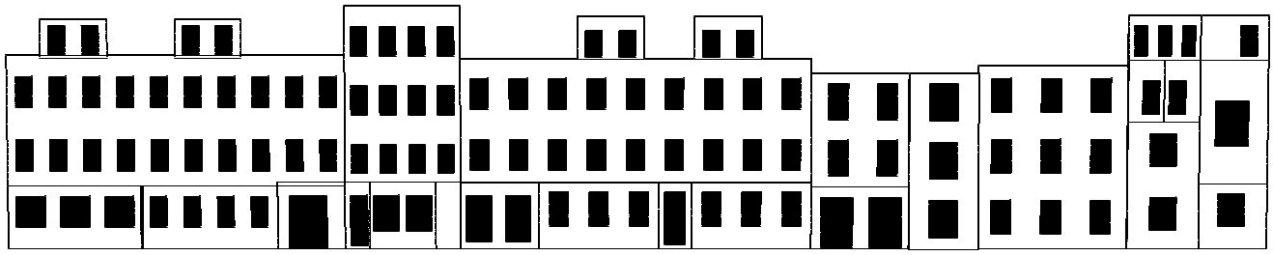
Another issue is the fact that small differences in the depth of cells that are not part of the same homogenization tile are preserved. If we brought them to the same depth, we could save the wall surfaces that are needed to connect the tiles in order to close the facade. For this reason, it makes sense to add a term to the objective function that rewards bringing adjacent tiles to the same depth.

Abbildung 8.14.: A facade structure covering a part of a block.

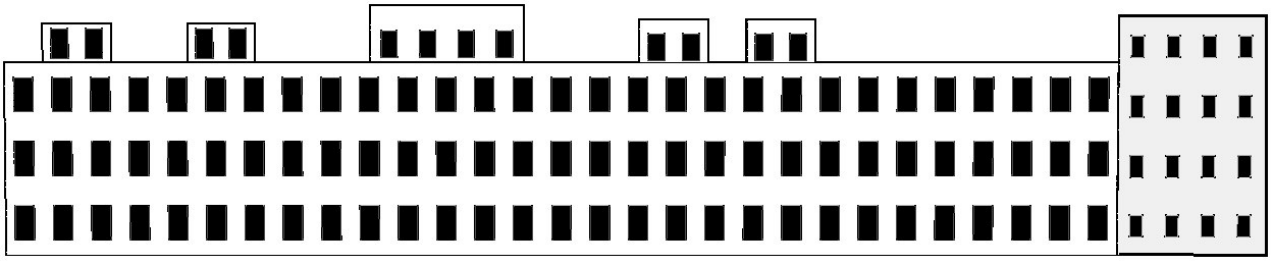
Figure 8.14 shows a facade structure covering one side of a block of townhouses in Hanover (Schneiderberg). The figure shows a screenshot of a part of a point cloud acquired using the Riegl VMX 250 (Riegl, 2013) mobile laser scanning device available at the Institute of Cartography and Geoinformatics (IKG).



(a) Original



(b) $\varrho = 0.3$



(b) $\varrho = 5.0m$

Abbildung 8.15.: Cells for the facade structure in figure 8.14.

In Figure 8.15, an abstract representation of the facade structure is shown in the first row. In this representation, ornaments were ignored and windows and doors were treated in the same way. The lines dividing the facade are the boundaries of the original tiles for the homogenization.

In the following rows, results of the homogenization process are shown for a large scale ($\varrho = 0.3m$) and a small scale ($\varrho = 5.0m$) generalization request; the gray lines in the figure mark the boundaries of the homogeneous

regions. Note that even for the large scale scenario, many large homogenization groups could be formed because the parameters of the openings in the tiles were very similar – especially in the upper floors.

Especially the slightly generalized version in the second row shows that in the current version, an alignment of the heights is only performed within the tiles, not along a whole row of windows – although the windows were almost aligned in the original model.

The rightmost building in the facade string (shaded in the bottom row of the figure) is similar in its structure to the example illustrating possible advantages of combinations of aggregation and homogenization. Because in the current version, only homogenization is applied, the maximum number of four columns of windows (from the four windows in the top row) are generated in the simplified facade.

8.5. Summary and Discussion

In this chapter, the facade homogenization problem was introduced. It was established that, for data compression, homogenization in itself is a means of generalization if efficient structures representing grids of identical features are available.

Two different optimizing approaches to solve the problem were introduced and evaluated. While the first one describes the problem directly, the second one uses a flow model similar to the one used for the aggregation of LoD 1 building models described in chapter 7. Note that in both cases, the number of variables, constraints, and objective function terms grows in the order of $O(N_{cells}^2 N_{parameters})$. There were, for example, 175,428 variables, 178,057 constraints, and 40,290 objective function terms for the Schneiderberg facade example (without the alignment constraints) for the graph-based model; the template-based approach.

The fact that the change from the first to the second version dramatically improved the runtime and resource characteristics shows that a less direct approach to modeling a MIP problem can help to reduce ambiguities and that MIP solver performance can be increased if heuristics for special cases like flow problems can be used.

In this chapter, the facade homogenization problem was examined in an isolated fashion. Even though facade homogenization is in itself a generalization step, it is, however, strongly related to aggregation and typification; it is, in fact, a degenerated case of a typification in which a set of k features is replaced by a set of k' identical symbolic features where, for the homogenization, $k' = k$ and for a typification, $k' < k$. As we have seen, performing an aggregation step before a homogenization step can increase the number of options for the homogenization.

The decision which of the generalization operators are best applied to which parts of a facade or if an optimal result can be achieved by different sequences of the operators for the different parts of a facade is a most involved optimization problem (model selection) on top of the optimization of the homogenization process introduced in this section.

An important aspect to be integrated in future versions of the facade simplification is the special role of the (relative) depth of the facade tiles: Even if they do not have anything else in common, we can save several wall, roof and floor features if two adjacent tiles have the same depth. In the current implementation, this aspect is not yet considered, so in some cases, adjacent wall tiles with only very small offsets were not merged to form a single wall surface because the features in the cells of the tiles were not similar enough.

9. Conclusions and Outlook

9.1. Summary of the MIP-based Optimizing Approaches

Although they are both NP-hard, the two problems of LoD1 building aggregation and facade structure homogenization that were investigated as optimization problems turn out to be different in practice because they represent different classes concerning the applicability of simple heuristic approaches: A simple region-growing based heuristic approach can be used to find good solutions for the first problem, while finding a promising approach for the second problem of facade homogenization is considerably more involved.

The main reason for this is the fact that small local changes of a solution can have strong global implications due to the rectangularity constraint for the clusters: There is no "neighborhood" of other valid solutions around a valid solution that would allow a "walk" towards a good solution from a bad one, and it is hard to "repair" a non-valid solution.

This is also a serious problem concerning the applicability of randomized approximation schemes like simulated annealing, genetic algorithms or Monte Carlo methods (N. Ripperda (2010), for example, used reversible jump Markov Chain Monte Carlo successfully for facade pattern recognition) to the facade homogenization problem and the reason why these approaches are not discussed at length in this thesis: For the facade homogenization problem, these approaches are problematic because of the lack of valid solutions in the neighborhood of a given solution, and for the LoD 1 building aggregation problem, the simple region-growing heuristic provided almost optimal results even for the most complex cases, so more complex Monte Carlo approaches were not investigated in this context for this problem.

For both of these problems, Mixed Integer Programming (MIP) representations were developed; MIP solver software (CPLEX) was used to obtain optimal solutions for real-world data sets. The advantage of using MIP (software) is that if our problem can be represented by a set of linear constraints and a linear objective function, the software can find an optimum solution (and even prove it to be optimal) in many cases although the general problem is NP-hard.

One reason for using MIP is to find out if a given combination of **linear** constraint set and **linear** or **quadratic** objective function captures the intent of the generalization well. For both scenarios, the MIP-based generalization operators provided visually convincing solutions despite the limitation to linear (and quadratic) terms in the constraints and objective function.

In order to achieve a thorough evaluation of the quality of the generalization operators, a large-scale user survey would be necessary; the advantage of using MIP solutions in such a survey is that these solutions are optimal, i.e. they are the result of the most rigorous application of the constraints and objective. Because the details of the generalization operators are not the focus of this thesis, no extensive user survey was conducted.

For the LoD 1 aggregation problem, a heuristic approach based on region-growing was introduced that uses criteria comparable to the ones employed in the constraints and objective function of the MIP problem. This means that we can use the results of the MIP solution as a reference to evaluate the quality of the solutions from the heuristic approach. In a large-scale application scenario, we can increase our confidence in a heuristic algorithm by evaluating its results using the MIP results for a representative test data set before the heuristic algorithm is applied to the whole data set.

Using the facade homogenization problem as an example, it was shown that the same problem can be mapped to different MIP representations by providing two possible representations based on template instancing and a flow model. It turned out that the CPLEX solver software could handle only very small problems in the first representation and considerably more complex ones in flow representation. The most probable reasons for this result are the larger potential for ambiguities in the first representation and more advanced heuristics for the flow problem structure in the CPLEX software.

9.2. Building Footprint Simplification

An important aspect in the aggregation of LoD 1 building models is the degree of geometric simplification that can be achieved in an aggregation step. This simplification can be measured by counting the total number of line segments in all building footprints before and after the aggregation. Such a measure is, however, only meaningful if all individual building footprints have already been simplified to the target scale.

Most LoD 1 building data sets are, however, derived from cadastre data sets, and these cadastre data sets usually contain very detailed and precise building footprints (in Germany, for example, accuracies of 10cm are not uncommon). Especially in the modeling of arcs, there is often a large number of edges: Many arcs are modeled by hundreds of edges where less than 30 would have been sufficient even with accuracy requirements in the range of decimeters.

For such detailed footprints, an aggregation algorithm would be biased to perform building matches in which such detailed feature parts are eliminated even though a further total simplification could have been achieved if the buildings had been simplified before the aggregation.

For this reason, an algorithm for the simplification of buildings is presented in this thesis. In contrast to existing approaches, this algorithm uses the direction of the edges in the footprint in the determination of the initial line segment hypotheses; modifications in the following adjustment process include a dynamic reassignment of parts of the original footprint to line segments in the simplified models in each step.

The results of this approach are promising but some stability issues remain, especially concerning the algorithm's ability to produce valid (free of self intersections) output polygons for small input footprint structures. For this reason, the algorithm was not yet used to produce the input data set to test if the incorporation of constraints and objective terms for the reduction of geometric detail could improve the results of the LoD 1 aggregation approaches.

9.3. Orchestration of Specialized Feature Generalization Operators in a Hierarchical Model

Due to the complexity of even the most basic sub-problems, an infrastructure for combining different approaches for individual features was proposed. The basic idea to tackle the complexity of the problem is to use generative modeling and minimum parameterizations to reduce the complexity of conflict resolution when independently simplified parts are put together again.

In doing this, we have, however, to be aware that if we do not include backtracking strategies in this approach, we cannot be sure to get the best or even very good results, because the quality of the generalization of a part of a model depends on the context: An, in itself, strongly sub-optimal generalization of one feature may, for example, make it fit perfectly with a surrounding pattern that allows a very effective generalization strategy to be applied.

To develop the ideas sketched in this context into a framework with a wide choice of features and generalization approaches is one of the most prominent topics for further research that can be derived from this thesis. The first step in this direction is to integrate the different special approaches derived in the context of this thesis into the framework. Another promising line of research is to include simplification approaches for structural features like symmetries or special facade structures.

9.4. Final Summary and Outlook

As we have seen, 3D building generalization is a complex optimization problem in different ways: In the first place, a generalization approach has to fit the intent of the user, resp. meet the requirements of the given application if they are (can be) specified exactly. These requirements are often hard to capture analytically, especially if human factors like aesthetic aspects in map-like representations are part of the optimization problem.

Additionally, resulting analytic optimization problems tend to be NP-hard because in most cases, a large number of independent decisions have to be taken in the generalization process (which generalization operator to apply, which features to aggregate etc.), leading to a combinatorial optimization problem.

Using LoD1 building aggregation and facade homogenization as examples, it was shown that for elementary generalization problems, analytic representations that produce convincing results can be derived in the form of MIP problems. Unfortunately (but not surprisingly), it turned out that solving even these elementary basic problems for small instances, up-to-date hardware was pushed to its limits even using sophisticated commercial MIP solvers.

For this reason, a closed in-depth MIP-based representation of the complete building generalization problem will probably be infeasible to compute because the computational complexity of the whole problem is a product of the complexity of the sub-problems (for each higher level decision, all lower level decisions may have to be reconsidered).

This means that at least at higher levels, heuristic approaches are likely to be necessary to handle the complexity of the problem. In many cases, a combination of specific simplification algorithms for the most relevant features and standard generalization operators for the rest will be the most effective way of designing a heuristic approach for the complete generalization problem for a given application without having to reinvent the wheel. In order to support this concept of reusability of generalization algorithms for different features in a hierarchical model, a structure for a framework for the flexible combination of generalization approaches was sketched in this thesis.

Considering the applicability of exact optimization-based approaches to large real-world data sets, an encouraging aspect is that for almost all feature classes and scales, there are natural opportunities for partitioning in the structure of city models that usually yield sufficiently small units: For the facade homogenization problem, for example, the natural units to be considered are the facades around a building complex; for the aggregation of LoD 1 building models, the natural units are building blocks or, at very small scales, super-blocks defined by higher-level roads, rivers, or larger areas with few buildings like forests or fields.

If we can split out data set into independent parts, the complexity of the whole calculation is no longer the product but the sum of the complexity of the partial problems, so if we can identify a maximum complexity for the single units, the total complexity scales linearly in terms of this complexity: even if the maximum complexity can already occur if each part contains only one feature, the total complexity for the whole data set of N features would be bounded by N times the maximum complexity.

Due to the NP-completeness of the problem, the solving software may still need an excessive amount of resources for finding an optimal solution for some of the parts and proving the optimality of this solution. In these cases, the best solution that was found by the solver in a given amount of time or an alternative heuristic approach may be used.

In cases of more complex generalization strategies that introduce far-reaching dependencies (like the requirement that identical window styles, floor heights or shapes of balconies across block / building complex boundaries along a street still have to be identical after the generalization), optimization-based methods may still be used on a global level if approaches for the different parts can be defined that provide a sensibly small number of alternative solutions (or can be re-evaluated quickly for changing external constraints) and measures for the quality of the overall solution can be evaluated.

In addition to these considerations, the concept of homogenization without an immediate application of a further simplification of the enclosing pattern is a new approach in a cartographic context; for the production of a map, it offers no direct advantage – except, perhaps, for a smoother appearance.

For this reason, it does not appear in the classical list of generalization operators; it is neither a kind of symbolization because it changes only values of parameters and pattern group memberships but not the structure of the feature, nor is it aggregation or typification because the number of features is unchanged.

In the context of a digital model in which uniform patterns can be represented in a compressed form, it is, however, a generalization operator in its own right because it allows a reduction of the size of the data set. Especially if patterns with exceptions can be modeled, the concept of a minimum description length can be introduced as a generalization objective that can be balanced against the amount of changes necessary to achieve the simplification of the representation. An alternative term for this operator may be *pattern regularization*.

A. LoD 1 Building Aggregation is NP-hard

The complexity class NP ("nondeterministic polynomial time") refers to problems that can be solved by a nondeterministic Turing machine in a number of steps that is bounded by a polynomial function of the size of the input. NP-hard problems (like the Traveling Salesman Problem) are "at least as complex as the most complex" problems in NP: every problem in NP can be converted into such a problem in polynomial time. It is generally assumed that NP-hard problems cannot be solved by traditional (non-quantum) computers in polynomial time. A more detailed introduction to computational complexity and NP-hardness is provided, for example, in Garey and Johnson (1979).

Unfortunately, the reduction from the Vertex Cover problem to the AreaAggregation Problem given in (Haunert, 2009) cannot be used to prove the NP-hardness of BuildingAggregation because it relies on postulating a minimum size for each cluster which is something that cannot be assumed in BuildingAggregation because the buildings do not cover the underlying plane completely and, for this reason, introducing a minimum area for aggregated buildings may lead to infeasible problems.

For this reason, we have to find a new approach to establish the computational complexity of the problem. This section introduces a proof that BuildingAggregation is NP-hard even if we restrict the initial heights of the buildings to three levels, Δ_h is fixed, and the neighborhood graph is planar.

Theorem 1.1

> BuildingAggregation is NP-hard even if only three original height levels and an appropriate value of Δ_h are used, the neighborhood graph is planar, and all possible additional constraints are relaxed.

An instance of the BuildingAggregation is given by a neighborhood graph $G = (V, E)$, a function $h : V \to \mathbb{R}$ assigning a height to each building, and a maximum height difference Δ_h. The corresponding decision problem asks: Given a set of buildings arranged in a neighborhood graph, is it possible to partition this block into k or less connected clusters in such a way that $|h(v_i) - h(v_j)| \leq \Delta_h$ if vertices v_i and v_j are assigned to the same cluster?

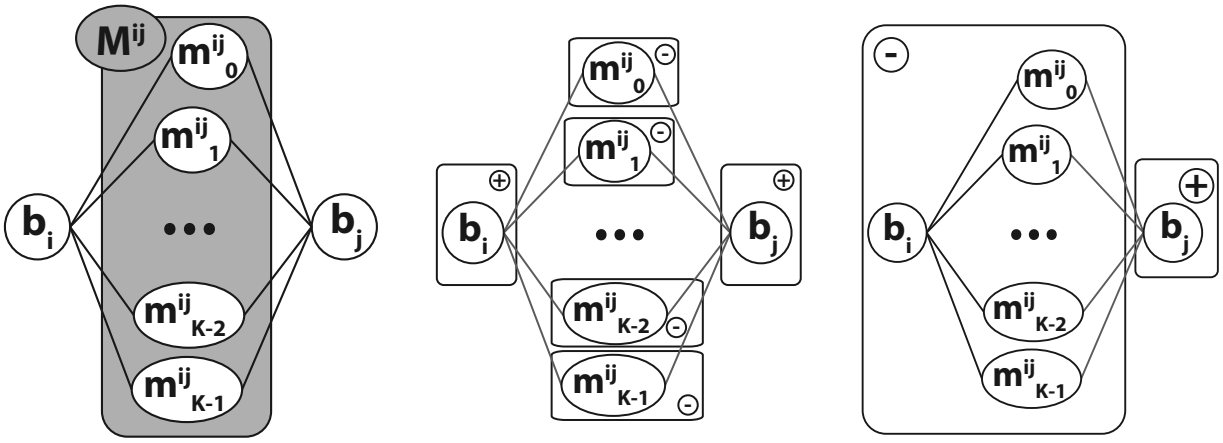
It is known (see e.g. Garey and Johnson (1979)) that an optimization problem is NP-hard if its corresponding decision problem is NP-hard. For this reason, BuildingAggregation is NP-hard if its decision problem is NP-hard. We prove this by giving a polynomial-time reduction from the decision problem corresponding to the maximum independent set problem which is known to be NP-complete (Karp, 1972) even for planar graphs (Berman and Fujito, 1995) to the decision problem of BuildingAggregation.

An independent set of a graph $G = (V, E)$ is a subset V' of the vertices of G that does not contain any adjacent vertices in G. The maximum independent set problem asks for the largest independent set in a given graph, the corresponding decision problem is to determine if a given graph contains an independent set of a size $\geq k$.

We reduce this decision problem to an instance $X := \{\tilde{G} = (\tilde{V}, \tilde{E}), \Delta_h, h, \tilde{k}\}$ of the BuildingAggregation decision problem. The basic idea of the reduction is to have a building b_i for each of the vertices v_i of the input graph with a neutral initial height – for example, we can set $h(b_i) = h_0 = 0$ for all b_i, so we have a set $B = \{b_i | v_i \in V\}$ with $h(b_i) = h_0 \; \forall 0 \geq i < |V|$.

Setting, for example, $\Delta_h = 1$, we define two additional height levels h_- and h_+ that are mutually incompatible but consistent with h_0, for example $h_- = -1$ and $h_+ = 1$. If a building b_i shares a cluster with another building of height h_+, this means that v_i is part of the independent set V', if it is associated with a building of height h_-, then it does not belong to V'. The height levels h_- and h_+ together with the value of Δ_h make sure that each original vertex is either assigned to V' or the rest $V \setminus V'$ of V.

In order to make sure that no pair of connected vertices in G can both be part of an independent set, we insert a large number of buildings ($K > \tilde{k}$, e.g. $K = \tilde{k} + 1$) with height h_- between the buildings representing adjacent vertices in G as shown in Fig. A.1(a), so if the adjacent vertices v_i and v_j were both selected for the independent set (share a cluster with a building of height h_+), then those K buildings would each form an individual cluster (see Fig. A.1(b)), and the overall limit for the number of clusters $\tilde{k}$ would be exceeded (we will show that we can define $\tilde{k}$ independent of K). This means that at least one of the vertices in each



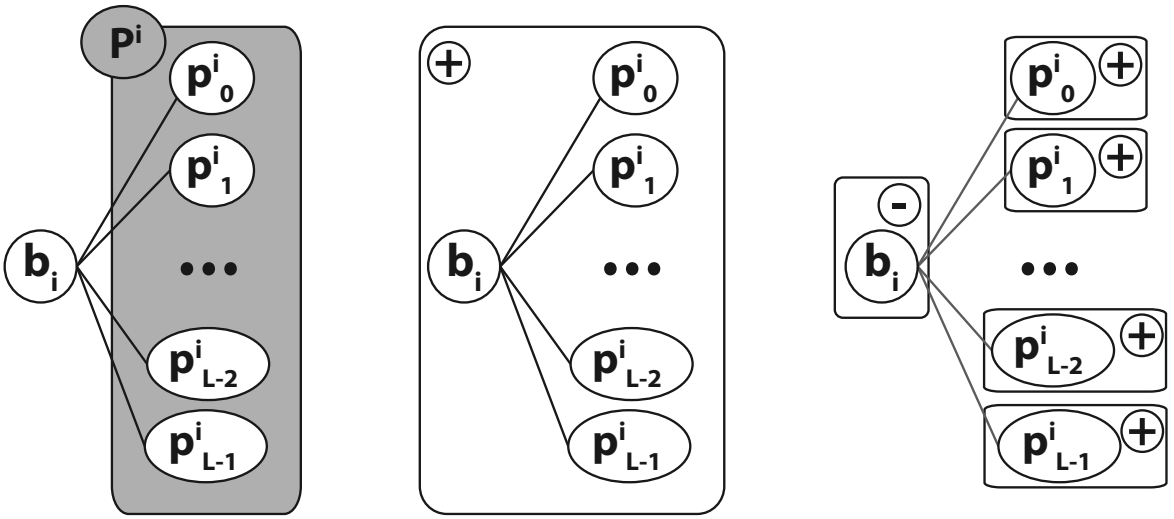
(a) Arrangement of the buildings m_r^{ij} between buildings b_i and b_j.

(b) Buildings b_i and b_j are both selected (height level h_+): Each m_r^{ij} forms a separate cluster.

(c) Building b_i is not selected (height level h_-) and forms one cluster containing all m_r^{ij} as well.

Abbildung A.1.: Reduction from Independent Set to BuildingAggregation: Modeling the independent set property.

pair of adjacent vertices in G is not part of the independent set: in this case, all vertices m_r^{ij} can be put into the same cluster as the unselected vertex as shown in Figure A.1(c). More formally, we define a set of buildings $M := \{m_r^{ij} | (v_i, v_j) \in E, 0 \leq r < K\}$ with $h(m_r^{ij}) = h_- \; \forall (v_i, v_j) \in E, 0 \leq r < K$ and a set of edges $E_M := \{(b_i, m_r^{ij}), (m_r^{ij}, b_j) | (v_i, v_j) \in E, 0 \leq r < K\}$. The set $M^{ij} = \{m_r^{ij} | 0 \leq r < K\}$ is the subset of M that corresponds to the edge (v_i, v_j) in G; we call this set M^{ij} the *M-group* corresponding to the edge (v_i, v_j).



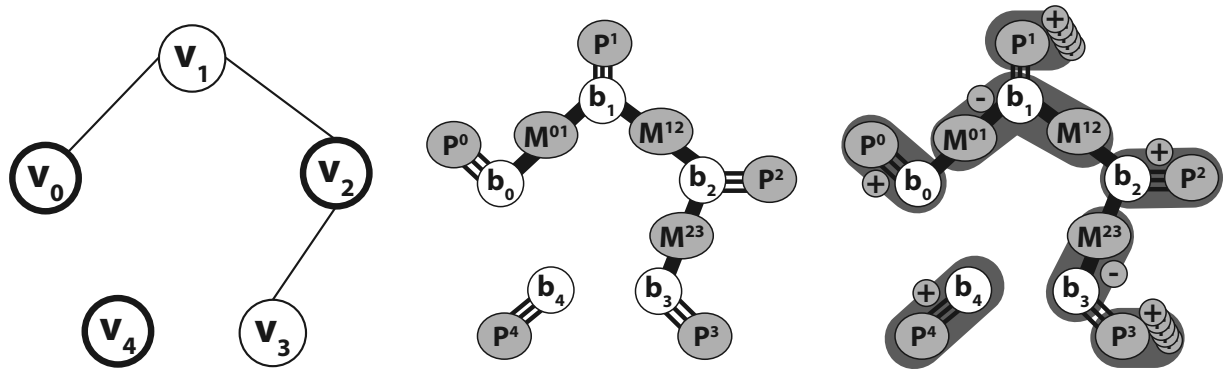
(a) Buildings p_j^i are attached to building b_i.

(b) Buildings b_i is selected: buildings p_j^i can share a cluster with b_i.

(c) Buildings b_i is not selected: each building p_j^i forms a separate cluster.

Abbildung A.2.: Reduction from Independent Set to BuildingAggregation: Maximizing the number of vertices in the independent set.

With this setup, there is a simple solution for the aggregation problem: Just select no building for the independent set and put everything into one cluster. For this reason, we introduce a penalty for each vertex that is not part of the independent set by attaching a set of L buildings with a height of h_+ to each building b_i, introducing the set $P = \{p_j^i | 0 \leq i < |V|, 0 \leq j < L\}$ of buildings and the set $E_p = \{(b_i, p_j^i) | 0 \leq i < |V|, 0 \leq j < L\}$ of edges shown in Figure A.2(a). If v_i is not part of the independent set (b_i is clustered with a building of height h_-), then each building p_j^i forms an isolated cluster as shown in Figure A.2(c), otherwise the vertices can be merged with the cluster formed by b_i (illustrated in Figure A.2(b)). The set $P^i := \{p_j^i | 0 \leq j < L\}$ (the *P-group* of b_i or v_i) is the subset of P that is attached to building b_i.



(a) Example: A simple graph; bold nodes: A maximum independent set.

(b) Building configuration corresponding to the graph in (a).

(c) A Minimal number of clusters matching the independent set marked in (a).

Abbildung A.3.: Reduction from Independent Set to BuildingAggregation: A simple example with $k = 3$.

The basic idea is to choose L sufficiently large to keep the maximum number of clusters that can occur in the case of $\geq k$ vertices in the independent set smaller than the minimum number of clusters that is possible if there are $< k$ vertices in the independent set.

Figure A.3 shows the worst-case situation for a simple example. Note that apart from the set $V_1' = \{v_0, v_2, v_4\}$ shown in the figure, there is another maximum independent set $V_2' = \{v_1, v_3, v_4\}$. According to the definition of an independent set, every unconnected vertex is automatically part of a maximum independent set because it has no adjacent vertices and therefore adding it to an independent set cannot create any conflicts. For this reason, it is no surprise that vertex v_4 is part of both possible maximum independent sets in the example.

If there is an independent set of size $\geq k$, then we can form k clusters on a positive height level containing the k buildings v_i in the independent set V' and their associated P-groups. In the worst case (if k is the maximum number of independent vertices), each vertex v_j of the $n-k$ ($n := |V|$) vertices not in V' must have at least one adjacent selected vertex v_i. This means that in order to "satisfy" the M-group M^{ij} associated with the edge (v_i, v_j), the building b_j must share a cluster with M^{ij} – in the worst case, these clusters cannot be merged, so there are $n - k$ clusters for the buildings not in V'. Having had to put the buildings corresponding to the vertices v_j not in V' on a negative height level, all L buildings in the associated P-groups P^j form an isolated cluster on a positive level, so we have another $(n - k) \cdot L$ clusters.

Summing up, we have at most the following number of clusters for an independent set of size $\geq k$:

- k clusters containing the buildings $v_i \in V'$ and their P-groups (level h_+)
- $n - k$ clusters containing the buildings $b_j, v_j \in V \setminus V'$ and their adjacent M-groups (level h_-)
- $(n - k) \cdot L$ clusters formed by the isolated buildings $p_r^j, v_j \in V \setminus V', 0 \leq r < L$ (level h_+)

So, in total, there are at most $k + (n - k) + (n - k) \cdot L = n + (n - k) \cdot L$ clusters in a minimum partition of $\tilde{G}$ if G has an independent set of size k. This means that if we set $\tilde{k} := n + (n - k) \cdot L$, the first (forward) part of the reduction is true: G contains an independent set of size $\geq k \Rightarrow \tilde{G}$ can be partitioned into $\tilde{k}$ or less clusters.

We now have to choose a value for L that makes sure that if $\tilde{G}$ can be partitioned into $\tilde{k}$ clusters, then G must contain an independent set of size $\geq k$. In order to do this, we determine the minimum number of clusters for an independent set of size $k - 1$:

- $k - 1$ clusters containing the buildings $v_i \in V'$ and their P-groups (level h_+)
- 1 cluster containing the buildings $b_j, v_j \in V \setminus V'$ and their adjacent M-groups (level h_-)
- $(n - (k - 1)) \cdot L$ clusters formed by the isolated buildings $p_r^j, v_j \in V \setminus V', 0 \leq r < L$ (level h_+)

So, in total, there are at least $(k - 1) + 1 + (n - (k - 1)) \cdot L = k + (n - k) \cdot L + L$ clusters if there is an independent set of size $k - 1$. For an illustration of item 2 in this list, imagine that vertex v_2 was not selected for the independent set in the example in Figure A.3: In this case, all buildings corresponding to vertices not in V' and their M-groups could be merged into a single cluster.

In order to establish that G has an independent set of size $\geq k$ if and only if $\tilde{G}$ can be clustered into $\leq \tilde{k}$, we have to select the values of L, K, and $\tilde{k}$ in such a way that the two worst cases described above can be separated by $\tilde{k}$ – so we have choose a value for L that makes sure that the worst-case minimum number of clusters for $|V'| = k$ is always lower than the absolute minimum number of clusters for $|V'| = k - 1$:

$$\begin{aligned} n + (n-k) \cdot L &< k + (n-k) \cdot L + L \\ \Leftrightarrow L &> n - k \end{aligned}$$

So choosing $L := n - k + 1$ ($\Rightarrow\ L \in O(n)$) separates the worst cases. This means that if we can partition the neighborhood graph $\tilde{G}$ corresponding to the graph G into less than

$$\tilde{k} := n + (n-k)L = n + (n-k)(n-k+1) = n + (n^2 - nk + n - nk + k^2 - k) = n^2 - 2nk + 2n + k^2 - k$$

clusters, then G must have an independent set containing at least k vertices because if the maximum independent set of G consisted of only $k-1$ or less vertices, then $\tilde{G}$ could not have been covered by less than $k+(n-k)L+L = k+(n-k)(n-k+1)+(n-k+1) = k+(n^2-nk+n-nk+k^2-k)+(n-k+1) = n^2-2nk+2n+k^2-k+1 = \tilde{k}+1$ legal clusters which is a contradiction to the assumption that we had covered $\tilde{G}$ with $\leq \tilde{k}$ clusters. This proves the second ("backward") part of the equivalence needed for a valid reduction: G contains an independent set of size $\geq k \Leftarrow \tilde{G}$ can be partitioned into $\tilde{k}$ clusters. Setting $K := \tilde{k} + 1 = n^2 - 2nk + 2n + k^2 - k + 1\ \in\ O(n^2)$ makes sure that selecting any two adjacent nodes of G for the independent set would result in at least $\tilde{k} + 1$ clusters even if the rest of the buildings is neglected, so this is no option. Any clustering not corresponding to assigning the vertices of G to the independent set or not would also only increase the number of clusters.

Wrapping up the reduction, we collect the components of X and their sizes in order to establish the complexity of the reduction:

- $\tilde{V} = B \cup M \cup P \rightarrow O(n^4)$ vertices:
  - $B = \{b_i | v_i \in V\} \rightarrow |B| \in O(n)$
  - $M := \{m_r^{ij} | (v_i, v_j) \in E, 0 \leq r < K\} \rightarrow |M| \in O(|E| \cdot K) = O(n^2 n^2) = O(n^4)$ (there are at most $O(n^2)$ edges in a graph with n vertices)
  - $P = \{p_j^i | 0 \leq i < |V|, 0 \leq j < L\} \rightarrow |P| \in O(n \cdot L) = O(n \cdot n) = O(n^2)$
- $\tilde{E} = E_P \cup E_M \rightarrow O(n^4)$ edges:
  - $E_M := \{(b_i, m_r^{ij}), (m_r^{ij}, b_j) | (v_i, v_j) \in E, 0 \leq r < K\} \rightarrow |E_M| = 2|M| \in O(n^4)$
  - $E_p = \{(b_i, p_j^i) | 0 \leq i < |V|, 0 \leq j < L\} \rightarrow |E_p| = |P| \in O(n^2)$
- $\Delta_h = 1$.
- $h = \{h(b) = 0\ \forall b \in B,\ \ h(m) = -1\ \forall m \in M,\ \ h(p) = +1\ \forall p \in P\} \rightarrow$ at most $O(|\tilde{V}|) = O(n^4)$ elements.
- $\tilde{k} = n + (n-k)(n-k+1)$.

So, in total, we have $O(n^4)$ elements (vertices with heights and edges) to create in the reduction and each individual element can be created in constant time. For this reason, our reduction works in polynomial time.

Since inserting the M-groups and the P-groups does not destroy the planarity of the graph G, $\tilde{G}$ is also planar if G was planar. Reducing from the independent set problem for planar graphs, we therefore get an instance of BuildingAggregation with a planar neighborhood graph. For this reason, BuildingAggregation is NP-hard even if the neighborhood graph is planar, because we can reduce an instance of the planar independent set decision problem – which we know to be NP-complete – to a planar BuildingAggregation problem.

So using the reduction introduced in this section starting with the planar independent set problem, we have shown that BuildingAggregation is NP-hard even if only three original height levels (e.g. $h_+ = 1$, $h_- = -1$ and $h_0 = 0$) and an appropriate Δ_h ($= 1$) are used, the neighborhood graph is planar, and all possible additional constraints are relaxed.

■

B. Facade Homogenization is NP-hard

In this section, we consider the problem of finding a minimum number of rectangular tiles that cover a given schematic facade representation with a bound ρ on the difference in a single parameter within each cluster. It will be shown that this problem is NP-hard even if only four different values occur for the property in the array (if these values are chosen appropriately with respect to the error threshold ρ).

Different problems concerning minimum rectangular tilings of matrices (2D grid structures) are important in many fields of research, including data base management, electronic circuit design, and many others. The RTile problem consists of finding the minimum number of rectangles that cover a given matrix if the sum of of the entries within each tiles is bounded by a given constant value. This problem has been shown to be NP-hard in Khanna et al. (1998). Different variations of it like its dual, the drtile problem, have been studied in various papers, e.g. Grigni and Manne (1996), Berman et al. (2001), Nichterlein et al. (2011). In these papers, variations of the problem and different approximation approaches are introduced.

Unfortunately, the different communities often use different terminology. Dividing a structure by rectangles is, for example, referred to as tiling, partitioning, or decomposition. If the covering shapes may overlap, the process is called covering. For this reason, it is difficult to find the appropriate literature. Although many similar problems are covered, the exact bounded difference rectangle tiling problem does not appear in the literature to the best knowledge of the author.

Muthukrishnan et al. (1999) generalizes the RTile problem to different classes of metrics and aggregation functions. Our bounded differences problem, for example, is closely related to the MAX-MAX-DIFF-AVG problem described there: the goal in MAX-MAX-DIFF-AVG is to find a minimum tiling given a upper bound for the maximum (first MAX, giving the global aggregation function over the accumulated values in the clusters) of the maximum (second MAX, referring to the aggregation function in each cluster) of the difference of the value within a cell to the average value of all cells in the cluster (DIFF-AVG).

0	**0**
1	**0**

Abbildung B.1.: DIFF-AVG and bounded difference: a pathological case.

Our case is a little different to the DIFF-AVG case because we want to restrict the maximum difference of the values of any pair of cells within a cluster instead of the distance of the value of a cell to the average value of the cluster. In many cases, these restrictions will lead to similar results, but the DIFF-AVG constraint is slightly harder because it takes all members in the cluster into account. Figure B.1 shows a pathological case illustrating this difference: In the case of DIFF-AVG, the four cells cannot be clustered for $\rho = 0.5$ because the average value in the tile would be 0.25; in the case of the bounded difference with $\rho = 1.0$, all cells could be merged.

Because of the limited space in the publication, Muthukrishnan et al. (1999) do not give a full NP-hardness proof for the problem. Instead, they refer to Khanna et al. (1998), claiming that a construction similar to the one used there to prove the NP-hardness of the RTile problem could also be used to prove the NP-hardness of the MAX-MAX-DIFF-AVG problem.

For this reason and because of the slightly different nature of the MAX-MAX-DIFF-AVG and bounded difference tiling problems, we will prove that the bounded difference tiling problem is NP-hard in the following.

This proof is based on the approach taken by Moore and Robson (2001) to prove that tiling a given grid with different simple shapes is NP-hard and the approach by Khanna et al. (1998) to prove the NP-hardness of the RTile problem. The basic idea is to construct a number of "gadgets" that simulate variables and logic gates or clauses in a satisfiability problem.

We consider the decision problem of the bounded difference tiling problem: Can a given array be divided into k tiles if the maximum difference between any pair of entries within a tile is smaller than a constant ρ?

In our proof, we will reduce the problem of planar 3-satisfiability (3-SAT) to the bounded difference tiling problem, showing that if we could solve the tiling problem efficiently (in polynomial time), we could also solve planar 3-SAT efficiently. Since planar 3-SAT is known to be NP-hard (Lichtenstein, 1982), this means that the bounded difference tiling problem is also NP-hard.

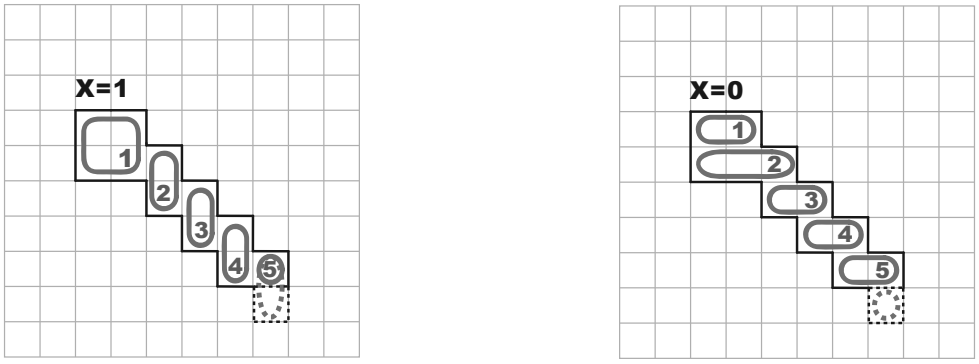
The 3-SAT problem asks if there is a satisfying assignment (meaning that the formula evaluates to TRUE) of values to the variables in a formula in conjunctive normal form (CNF, meaning a conjunction (AND) of clauses) in which each clause is a disjunction (OR) of 3 literals (positive or negative occurrence of a variable). In *planar* 3-SAT, the graph $G(F)$ in which the variables and the clauses of the formula are the vertices and an edge links a clause to a variable if the variable occurs in the clause (as a positive or negative literal) is planar.

This graph can easily be embedded in the rastered plane in polynomial time: Hopcroft and Tarjan (1974) give an efficient algorithm for embedding a planar graph in the plane, and Chrobak and Payne (1989) show that this planar graph can be drawn on a planar grid of dimension $(2n-4)x(n-2) = O(n^2)$. The size of our gadgets will increase this number but since each gadget has a constant size, the overall number of cells needed to represent G will still be in a low polynomial order ($O(n^2)$).

In our construction, we will develop "gadgets" that represent variables, edges, and clauses. The basic idea is to build the gadgets in such a way that there are basically two minimum tilings for each variable, representing positive and negative truth values. In a clause gadget, there is one cell that can only be covered without producing an additional tile if at least one of the literals (a positive or negative occurrence of a variable) in the clause is TRUE.

The parameter values in the cells within the black border lines in the figures are set to 0 if not marked otherwise, the ones outside are set to 5ρ, so it is impossible for a tile to be extended across a border. In the following, only the tiling of the interior area is considered – a minimum tiling of the exterior background cells can be determined effectively with polynomial-time algorithms for unconstrained rectangle tilings like the one described in Eppstein (2009); the size of such a tiling is treated as a constant K_{out}.

The "+" and "-" signs in some cells represent values of $+0.75\rho$ and -0.75ρ, respectively, so they can be merged with the neutral cells and with cells of the same sign, but cells of opposite signs cannot be part of the same tile.



(a) Tiling corresponding to TRUE value. (b) Tiling corresponding to FALSE value.

Abbildung B.2.: Variable gadget with part of an edge.

All gadgets except the clause gadget are constructed in such a way that the tilings for TRUE and FALSE are *balanced*, meaning that both tilings have the same size. In the example of the variable gadget shown in Figure B.2, both tilings have the size 6. It is easy to see that no tilings with fewer tiles are possible.

The variable gadget is the square in the top left corner; we define a tiling to represent the TRUE value for a variable if this square is covered by a single tile (Figure B.2(a)); in the other case, the variable has the value FALSE (Figure B.2(b)). The stairs pattern attached to the variable gadget represents an edge; the terminating cell (5) ensures that the tiling is balanced.

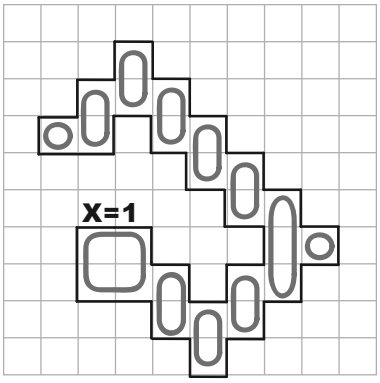
Note that if we add a cell (6) in vertical direction, the TRUE tiling can be extended in vertical direction without using an additional tile (dotted cell) while the FALSE tiling would need an additional tile to cover cell (6). Without taking additional measures, there is no such extension for which the FALSE tiling can be extended while the TRUE one cannot.

In this situation, we say that the TRUE value of the variable is *leading* on the edge. Because the TRUE tiling can be extended only in vertical direction, the *orientation* of the edge is vertical.

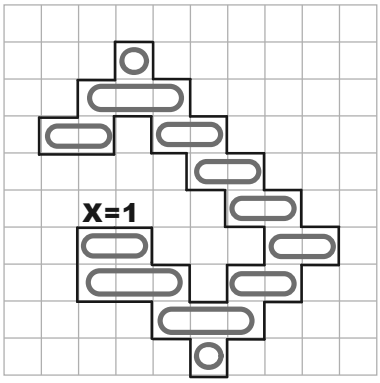
The clause gadgets are designed in such a way that their central tiles are connected to edges on which the values of the variables are leading for which the corresponding clause is satisfied (evaluates to TRUE); this may be the TRUE or FALSE tiling for the variable. For this reason, we need an inverter gadget that changes the leading truth value.

Since we are considering the 3-SAT problem, each clause gadgets has three input connector points (cells) to which edges can be connected. Depending on the connector, the orientation of the incoming edge has to be horizaontal or vertical. For this reason, we need an orientation change gadget.

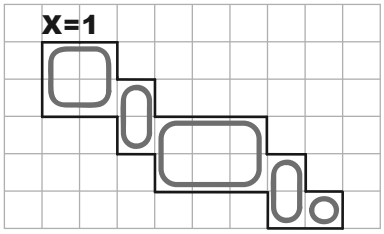
Since the graph representing the formula is planar, we do not need gadgets for managing intersections of edges.



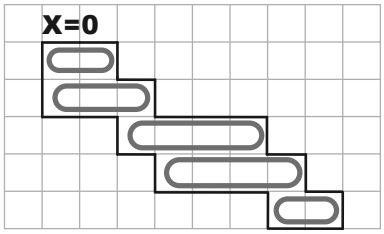
(a) Bending gadget: TRUE tiling.



(b) Bending gadget: FALSE tiling.



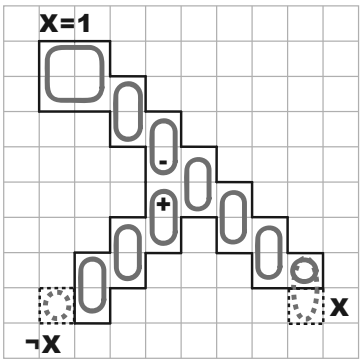
(c) Horizontal stretching gadget: TRUE tiling.



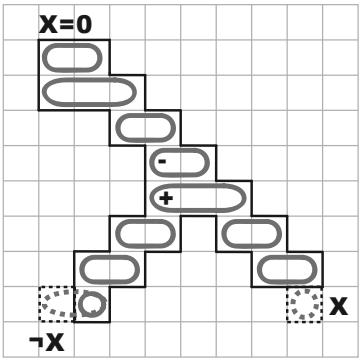
(d) Horizontal stretching gadget: FALSE tiling.

Abbildung B.3.: Edge direction adjustment: Bending and stretching edges.

Ordinary edges can only run diagonally in the plane; by using the bending and stretching gadgets shown in Figure B.3, the direction and slope of the edges can be adjusted as needed without changing the leading truth value or the orientation of the edge: Throughout the three bends in Figure B.3(a) and (b) and the in the stretching in B.3(c) and (d), the TRUE value is leading and has a vertical orientation.



(a) Inverter gadget: TRUE tiling.



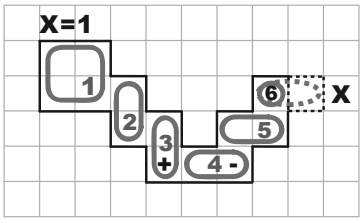
(b) Inverter gadget: FALSE tiling.

Abbildung B.4.: Inverter gadget

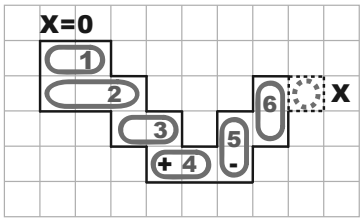
Since the clauses may also contain negative literals (inverted variables), it is necessary to implement a gadget

that allows extending a tile if a variable is FALSE. Such a gadget is shown in Figure B.4. Note that the ¬X branch is extensible in horizontal direction while the X branch is extensible in vertical direction. Using the orientation change gadget given below, this can be changed as necessary.

If it is not needed, the X branch may be left out or cut off with an appropriate terminator that retains the balance between TRUE and FALSE tilings.



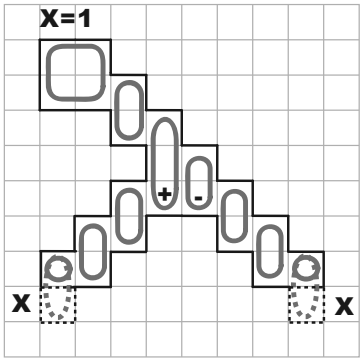
(a) Orientation change: TRUE tiling.



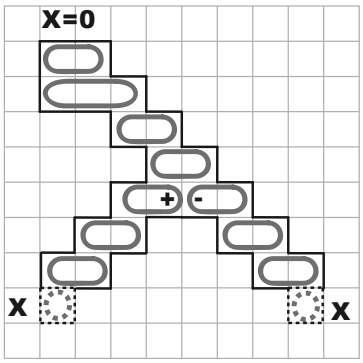
(b) Orientation change: FALSE tiling.

Abbildung B.5.: Orientation change gadget

The orientation change gadget shown in B.5 changes the direction in which an input tiling may be extended from vertical to horizontal. Rotating this gadget by 90 degrees allows direction changes from horizontal to vertical. Note that still the same truth value will be extensible as in the input. In our example, this is the TRUE value of a variable; it may, however, also be the FALSE value.



(a) Split gadget: TRUE tiling.



(b) Split gadget: FALSE tiling.

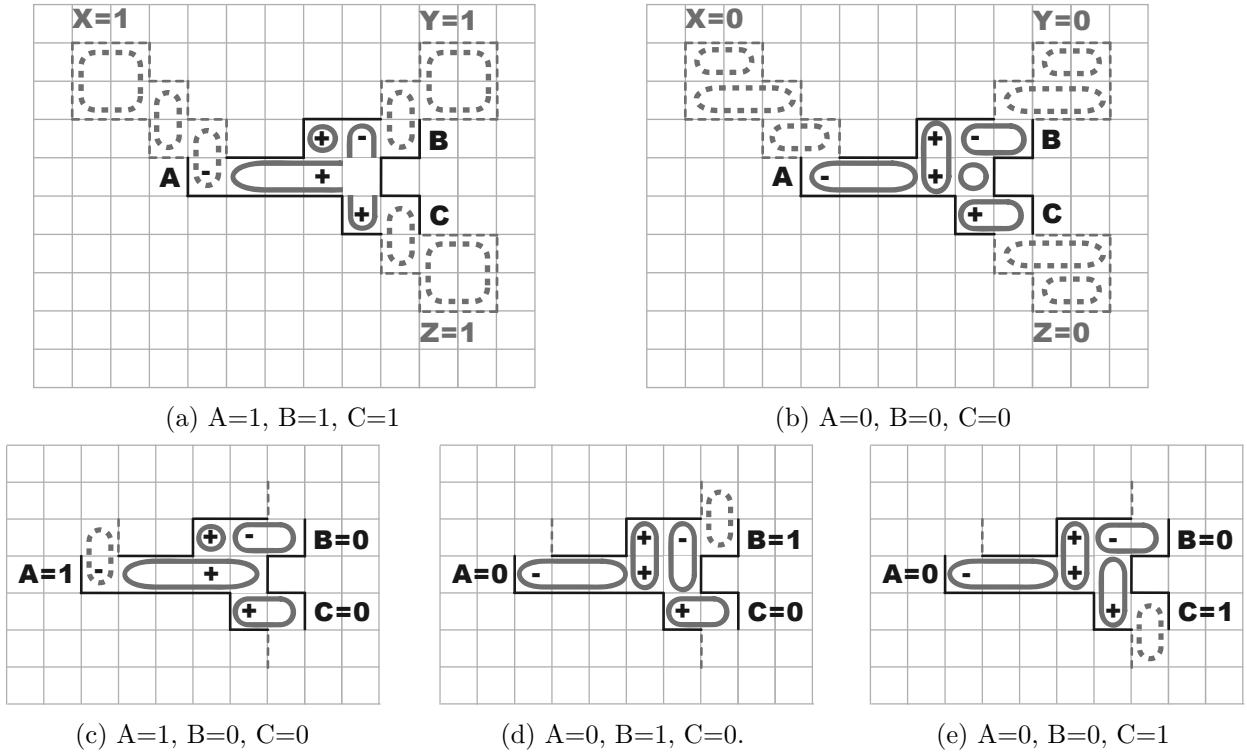
Abbildung B.6.: Split gadget

Figure B.6 shows that it is also possible to split an edge; this allows us to let different edges emerge from a given variable vertex. Note that for both output edges, the input truth value is preserved, and the orientation in which the truth value can be extended remains vertical. By rotating the gadget, edges of different orientation and direction may be split.

Using the gadgets introduced so far, we can make sure that edges can be made to arrive at a clause gadget with any truth value leading in any given orientation and direction. Each of the literals x_1, x_2, x_3 of the clause is mapped to one of the inputs A, B, C of the clause gadget. The assignment depends on the layout of the planar embedding of the graph G representing the formula, not on the order of the literals in the clause. Due to the planarity of G, there is always one such mapping that causes no topological obstacles.

Without loss of generality, we expect the appropriate truth value to be leading with a vertical orientation, so if there is a negative literal in a clause, there has to be an inverter and an orientation change gadget somewhere between the variable gadget and the clause gadget. For input A, the edge arrives from the upper or lower left, for input B from the upper right, and for input C from the lower right direction. Figure B.7 shows minimum tilings of the clause gadget for different truth levels at the input A, B and C. The variable gadgets directly attached to the inputs in Figures B.7(a) and (b) were added only for illustration purposes in order to support the reader in verifying that the gadget is indeed working as intended; in any non-trivial cases, there will be split and inverter gadgets between the variables and the clauses.

Figure B.7(a) shows that if all inputs have the right truth value, any of them can be extended to cover the central clause cell, but it is impossible to merge tiles that belong to different variable domains across the clause



(a) A=1, B=1, C=1

(b) A=0, B=0, C=0

(c) A=1, B=0, C=0

(d) A=0, B=1, C=0.

(e) A=0, B=0, C=1

Abbildung B.7.: Clause gadget

cell because of the "-" and "+" above and below the cell. It is left to the reader to verify that there are different possibilities to tile the gadget with four tiles in this case but that there is no tiling that uses less than four tiles.

If all inputs have the wrong truth value (Figure B.7(b)), then none of the variable tilings can be extended to cover the central cell. For this reason, the clause gadget cannot be covered by less than five tiles. Figures B.7(c)-(e) show that even if only one of the inputs is TRUE, there is a tiling with four tiles for the clause gadget because the tiling of the TRUE input can be extended to cover the central cell and the rest of the gadget can always be covered by three additional tiles.

It is left to the reader to verify that the gadget can be covered by four (but not less than four) tiles if any pair of inputs evaluates to TRUE. In order to check this, imagine one of the FALSE inputs being TRUE in Figures B.7(c), (d), or (e) and check that this does not require using additional tiles. Referring to Figure B.7(a), it is obvious that this will not offer any opportunities for covering the gadget with only three tiles.

So each clause gadget can be covered by four tiles if and only if the clause evaluates to TRUE; otherwise, five tiles would be needed. Since all other gadgets are designed to be balanced and there is no way to tile the clause gadget using three or less tiles, we can now wrap up our reduction:

- Given an instance of planar 3-SAT, i.e. a formula F in CNF with three literals in each clause and a planar associated graph $G(F)$ defined as above, embed $G(F)$ in the rastered plane, leaving space for the the different gadgets. Since all gadgets have a constant maximum extent c in each direction, such an embedding will use only $c(2n-4)(n-2) = O(n^2)$ cells according to Chrobak and Payne (1989) and can be computed in polynomial time (even linear, if only the vertices are placed, and no array is initialized).
- Replace the variable and clause vertices in $G(F)$ by the appropriate gadgets. This is done by filling the array A of size $O(n^2)$, so the runtime will be $O(n^2)$. Let $|A|$ be the number of cells in A.
- Count the number K_{out} of tiles needed to cover the background cells of A, using e.g. the algorithm given in Eppstein (2009). This can be done in $O(|A|^{3/2} \log |A|)$ time which is obviously polynomial.
- Count the number K_{in} of tiles needed to cover the cells in the gadgets except the central cells of the clause gadgets. This can easily be done by choosing arbitrary truth values for all variables (for example, all TRUE) and greedily tiling the gadgets. Due to the design of the gadgets, this can be done in linear time in the number of cells in the gadgets, so $O(|A|)$ is an upper bound for this operation, so this polynomial as well.

If and only if A can be tiled by $K := K_{in} + K_{out}$ tiles, then there is a satisfying assignment for F: As we have seen above, the clause gadget excluding its central cell can always be covered by four tiles, so it will count as four cells in the calculation of K_{in}. If the clause is satisfied by an assignment (one of the inputs is has the appropriate truth value), then no additional tiles are needed because the tiling of the TRUE input can be extended to cover the central cell. For this reason, it always possible to tile A with K tiles if F is satisfiable.

If a clause is not satisfied, then an additional fifth tile is needed to cover its central cell. At this point, it is important that there is no way to tile a satisfied clause gadget with three or fewer tiles because this could allow a non-satisfied clause to "slip through". Since this is not the case, it is impossible to tile A with K tiles if F is not satisfiable.

This reduction from planar 3-SAT shows that covering an array with a minimum number of rectangles with bounded differences between the cell values within each tile is indeed NP-hard. Since we used only the values for background, neutral, "+" and "-" cells are used in the reduction, this hardness result holds even if only four different parameter values are used in the cells.

■

C. Detailed discussion of the influence of different parameter weights in the facade homogenization problem

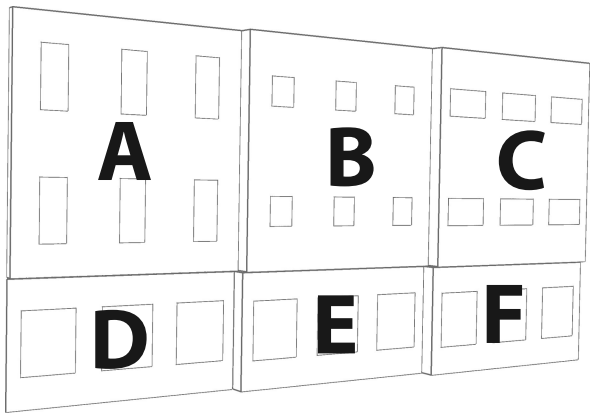


Abbildung C.1.: Original

In this section, the influence of the different weights for the parameters is discussed using the simple artificial example shown in figure C.1. In this example, we consider three different parameters: the width and height of the windows within a cell and the depth of the cell, meaning its shift with respect to the main plane of the facade: a higher depth value means a protrusion, a lower one means an indentation.

The widths and heights of the windows in the example are set to $1m \times 2m$, $1m \times 1m$ and $1m \times 1m$ in the upper floors (from left tom right) and to $2.1m \times 2.1m$ for the ground floor. The depths are (from left to right) 0, 0.6 and $1.2m$ for the ground floor and 0.3, 0.9 and $1.5m$ for the upper floors. The threshold Δ_p for all the three parameters is set to equal a resolution ϱ of $0,51m$. These values were chosen to illustrate different effects that occur in the homogenization process.

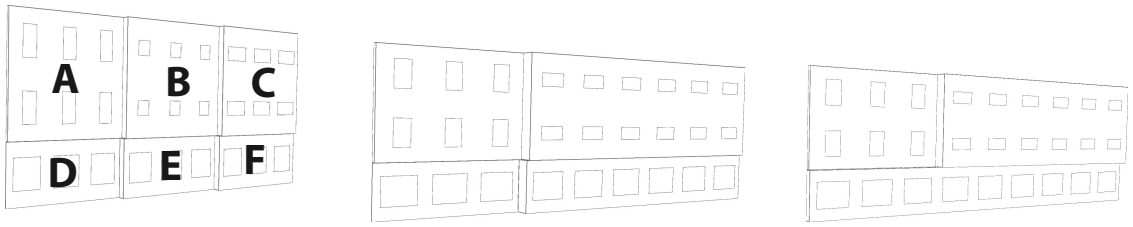
In order to avoid that there are small differences (below the resolution) in adjacent tiles in the result, the homogenization process is repeated using the result from the last iteration as the input for the next until no further simplification occurs. This may, however, cause violations of the Hausdorff property described in section 2.1.1 – for a single run, it is ensured by the Δ_p bound on the parameter difference. The setting of the width and height parameters forbids that cells form the upper and lower floors can be merged in the first run because either the width or the the depth (or both) are $1.0m$ in the upper floors while it is $2.1m$ in the ground floor, there is no possible intermediate value with a difference of less than $\varrho = 0.51m$ to $1m$ and $2.1m$.

For the experiment, the value $W_{\#f}$ was fixed at 100 and the value W_p was set to simulate a different impact for the different parameters: If a variable has a high importance – so changes to it should therefore be penalized more heavily, the value W_p was set to 20 for it; otherwise, it was set to 1. We will refer to these values as HIGH and LOW importance levels.

Note that the different penalties also define the relation of the parameter difference penalties to the penalty $W_{\#f}$ for the number of resulting tiles, so there is a difference between the version in which all parameters are HIGH and the one in which all parameters are LOW.

Figure C.2 shows the result of the optimization runs with high penalties for all parameters. Because the weights for the width and height of the windows are the same, it does not make a difference if the areas A and B or the areas B and C are merged. In our test, CPLEX chose to merge areas B and C. Note that although in the case of this example, all cells within the areas are part of the same tile in the result for all combinations of weights, this is not always the case.

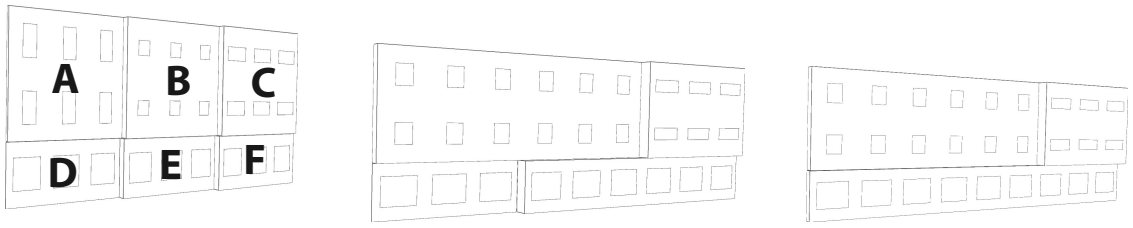
Given the difference of $1.5m$ in the depth between areas A and C, it is impossible to merge the three cells $A-C$ in the first step. The same holds for lower part (areas $D-F$). In the second run, however, the the area resulting from E and F has a depth of 0.9, so it can be merged with area D which still has a depth of 0. According to the



(a) Original facade. (b) Facade after first iteration. (c) Facade after second iteration (final).

Abbildung C.2.: Homogenization sequence for high penalties for all variables.

objective of minimizing the sum of the squared differences between the original values and the resulting values in all cell, the new depth value for the merged cells in the ground floor should be 0.6 because there are 6 cells with a value of 0.9 and three ones with a value of 0. This would, however, violate the constraint enforcing the Δ_p ($= 0.51$ in the example) bound on the parameter differences. For this reason, the resulting depth is chosen as close as possible to the optimal value of 0.6 without violating the Δ_p constraint, so because the optimum value is 0.6 and the critical value is 0, the resulting value for the depth is $0 + \Delta_p = \Delta_p = 0.51$ in this case.

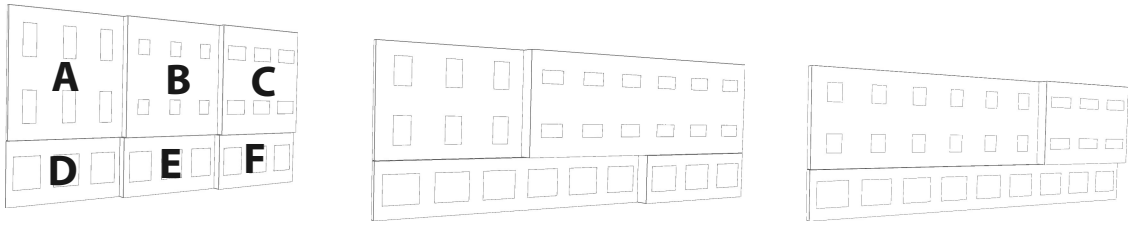


(a) Original facade. (b) Facade after first iteration. (c) Facade after second iteration (final).

Abbildung C.3.: Homogenization sequence for LOW *penalty for the height and* HIGH *for the other variables.*

If we have a LOW importance for the window height (with the rest of the parameters set to HIGH importance; see figure C.3), then the areas A and B will have to be merged in the first run because the penalty for this homogenization is lower than it would have been for merging B and C.

Since the depth differences and the parameters of the windows are identical for all cells in the ground floor, the choice whether the areas D and E or the areas E and F are merged is completely arbitrary in the basic set of constraints. In the previous setting, a sensible combination was chosen – the tilings of the ground floor and the upper ones were aligned. This was, however, pure chance. In this second example, the different parts are not aligned correctly. In order to prefer sensible alignments, additional constraints or objective terms would be needed.



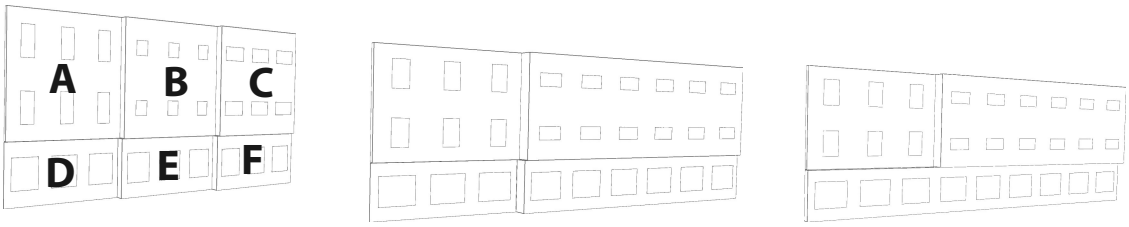
(a) Original facade. (b) Facade after first iteration. (c) Facade after second iteration (final).

Abbildung C.4.: Homogenization sequence for LOW *penalty for the width and* HIGH *for the other variables.*

If the penalty for the differences in the width of the windows is LOW (and HIGH for the other parameters), then the areas B and C will have to be merged in the first run of the optimization process; merging A and B would cause a higher penalty. This is shown in figure C.4.

In this case, the areas D and E were merged in the first run, so the resulting depths will be $0.3m$ for the tile resulting from D and E and $1.2m$ for the cells in area E. The optimum value is, again, $0.6m$:

$(6 \times 0.3m + 3 \times 1.2m)\,/9 = 0.6m$. This would, however, violate the Δ_p bound on the parameter difference for the cells in area E, because $1.2m - 0.6m = 0.6m$ which is greater than $\varrho = 0.51m$. For this reason, the resulting depth for the ground floor after the second run will be $1.2m - 0.51m = 0.69m$ instead of $0.51m$ which was the result of merging D and E first (see above). This shows that different orders of merging in different runs can cause parameters to "drift" in different directions.

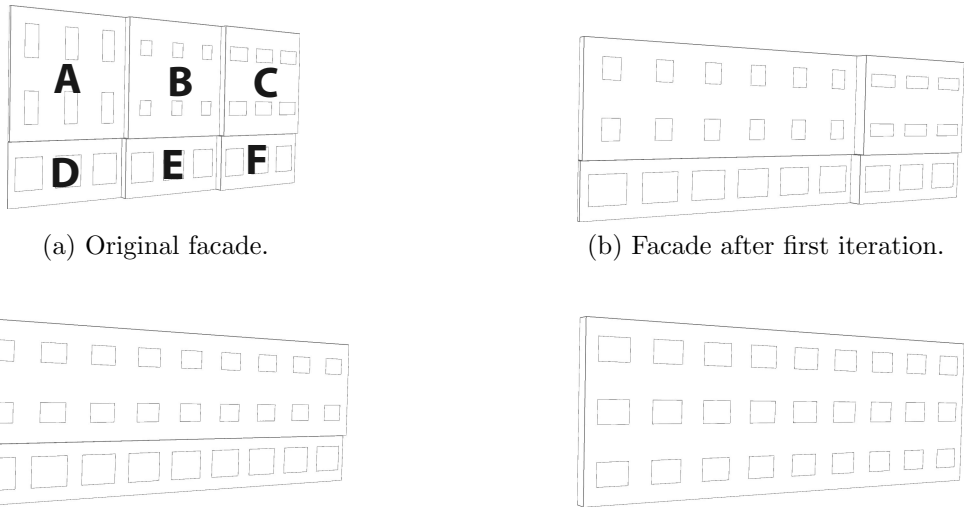


(a) Original facade. (b) Facade after first iteration. (c) Facade after second iteration (final).

Abbildung C.5.: Homogenization sequence for LOW *penalty for the depth and* HIGH *for the other variables.*

As we have established above, it is not possible to merge cells from the upper and lower floors in the first run because of the values of $1.0m$ and $2.1m$ for one of the width or height parameters. Because of this fact, no cells from the upper and lower floor are merged (A and D, for example) although the weight for the depth is LOW in figure C.5 while it is HIGH for the other parameters.

Fore this reason, the upper and lower floors are treated independently and there is no preference for merging A and B or merging B and C first in the upper part nor for preferring the merging of either D and E or merging E and F.



(a) Original facade. (b) Facade after first iteration.

(c) Facade after second iteration. (c) Facade after third iteration (final).

Abbildung C.6.: Homogenization sequence for HIGH *penalty for the width and* LOW *for the other variables.*

So far, we considered cases in which the majority of the parameter weights was HIGH. The values of the parameters were chosen in such a way that, in this case, the cost for changing the parameters would have exceeded the gain of saving another tile if the cells of the upper floors would have been merged in the second run. With the majority of the weights being LOW, this is now different.

In order to convince ourselves of the fact that after the first run, the cells of the upper floors can indeed be merged, we take a closer look at the situation shown in figure C.6. Because the weight for the width is high, the areas A and B have to be merged in the first run because areas B and C have different window widths, so merging them would cause a higher penalty.

In order to simplify the descriptions in the following, we will refer to the area resulting from the homogenization of X and Y as area XY for the next run; if area Z was merged to this area in a following run, the result will be called XYZ. Note that XYZ and YZX have different histories and, as we will see in the following, may have different parameter values.

After the first run, the windows in area AB will have a width of $1m$ and a height $1.5m$. The cells in area C keep their width of $2.0m$ and their height of $1m$. This means that it would be just possible to merge areas AB and C in the second run. The optimum values for the width and height of the windows would, in this case, be $(12 \times 1.0m + 6 \times 2.0m)/18 = 1.\bar{3}m$ for the width and $(12 \times 1.5m + 6 \times 1.0m)/18 = 1.\bar{3}m$ for the width of the windows. We round this value to $1.33m$.

Due to the Δ_p bound on the parameter differences, not all parameters can be set to this value. In the case of the width of the windows, the value of $2.0m$ in area C has a difference of $2.0m - 1.33m = 0.67m$ which is more than the threshold δ_p if $\varrho = 0.51m$. For this reason, the width will be set to the valid value that is closest to the optimum; in this case, this will be $2.0m - 0.51m = 1.49m$.

As far as the height is concerned, we have values of $1.5m$ for the area AB and $1.0m$ for area C, so we can set the height to the optimum value of $1.33m$. The depth of area AB is $(6 \times 0.3m + 6 \times 0.9m)/12 = 0.6m$; the depth of area C remains at $1.5m$. The optimum value for the depth in ABC is therefore $(12 \times 0.6m + 6 \times 1.5m)/18 = 0.9m$. With the Δ_p constraint, we have to set the depth to $1.5m - 0.51m = 0.99m$.

In total, we notice that the values are shifted towards those of the area that was not part of the homogenization in the first run if they cannot assume the optimum value – in the given example, the height could be set to the optimum value. This shows that there is a problem in the multi-run strategy: The parameter with the higher importance value (in this case, the width) finishes farther from the optimum than the one with the lower importance.

On the other hand, if there is (a set of) cell(s) that have considerably different values for an important parameter compared to the rest of the facade, this effect reduces the likelihood of merging such a special cell with the rest of the facade. Additionally, if such a special cell is merged with other cells, the special property is pronounced. Only if a bigger area is merged as in the example, this is a problem; otherwise, it is a feature rather than a bug.

In the bottom row, CPLEX chose to merge areas D and E in the first and DE and F in the second run. Since all parameter differences are the same for the bottom row, this is completely arbitrary. As we have seen above, this leads a depth of $0.69m$ for the area DEF. Had we merged E and F first, the resulting depth for the area EFD would have been $0.51m$. For this reason, the depth of the final facade would have been slightly different: In the DEF case, it would have been $(18 \times 0.99m + 9 \times 0.69m)/27 = 0.89m$ and $(18 \times 0.99m + 9 \times 0.51m)/27 = 0.83m$ in this example (with the ABC merging sequence in the upper part). Similar effects with slightly different values will occur if the merging sequence had been BCA in the upper part.

Now we can go back to see why the areas in the upper floors were not merged in the previous examples. After the first run, we have values of $0.6m$ for the depth, $1.5m$ for the height and $1m$ for the width of the windows in area AB. If area C was merged with area AB, the resulting values would be $0.99m$ for the depth, $1.33m$ for the height and $1.49m$ for the width as explained above.

So the basic (unweighted) cost for this merge is $12 \times (0.6 - 0.99)^2 \approx 1.83$ for the depth, $12 \times (1.5 - 1.33)^2 \approx 0.35$ for the height and $12 \times (1.0 - 1.49)^2 \approx 2.88$ for the width in area AB and $6 \times (1.5 - 0.99)^2 \approx 1.56$ for the depth, $6 \times (1.0 - 1.33)^2 \approx 0.65$ for the height and $6 \times (2.0 - 1.49)^2 \approx 1.56$ for the width in area C.

Sorted by parameters, we get total unweighted costs of $1.83 + 1.56 = 3.39$ for the depth, $0.35 + 0.65 = 1.0$ for the height and $2.88 + 1.56 = 4.44$ for the width. Due to the effects analyzed above, the greater unweighted difference will always occur for the parameter with the higher weight if width and height are weighted differently. For this reason, we have a total cost of $W_{width}C_{width} = 20 \times 4.44 = 88.8$ for the width.

If all other weights are LOW(=1), then we get a total cost of $88.8 + 3.39 + 1.0 = 93.2$ for merging AB with C. This cost is just lower than the benefit of $W_{\#f} = 100$ that we get for having saved one facade tile by merging AB and C. For this reason, area AB is merged with area C in this case. Now we also have the explanation why this did not occur in the examples we encountered before: In these cases, at least one of the other parameters had a HIGH weight, so the cost for merging AB and C was greater than the benefit of saving the additional tile.

Due to the symmetry of the example, the explanations and calculations given above for the case of a HIGH weight for the width can be used for the case of a HIGH weight for the height of the windows . Note that, in this case, areas B and C will have to be merged first and the values and costs for the widths and heights have to be swapped appropriately in the assessment of the option of merging area BC with area A. The basic arguments and values are, however, the same, so it is left to the reader to verify the figures.

Abbildungsverzeichnis

Literaturverzeichnis

Adobe, 1999. Postscript language reference manual. Adobe Systems, Addison-Wesley.

Anders, K.-H., 2005. Level of Detail Generation of 3D Building Groups by Aggregation and Typification. In: Proceedings of the XXII International Cartographic Conference, La Coruna.

Berkelaar, M., Eikland, K., Notebaert, P., 2004. lp_solve 5.5, Open source (Mixed-Integer) Linear Programming system. Software, available at http://lpsolve.sourceforge.net/5.5/. Last accessed Aug. 30, 2012.
URL `http://lpsolve.sourceforge.net/5.5/`

Berman, P., Dasgupta, B., Muthukrishnan, S., Ramaswami, S., 2001. Improved Approximation Algorithms for Rectangle Tiling and Packing (Extended Abstract). In: Proc. 12th ACM-SIAM Symp. on Disc. Alg. pp. 427–436.

Berman, P., Fujito, T., 1995. On approximation properties of the Independent set problem for degree 3 graphs. In: Akl, S., Dehne, F., Sack, J.-R., Santoro, N. (Eds.), Algorithms and Data Structures. Vol. 955 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 449–460.

Brenner, C., 2000. Dreidimensionale Gebäuderekonstruktion aus digitalen Oberflächenmodellen und Grundrissen. Dissertation, Universität Stuttgart, Deutsche Geodätische Kommission, Reihe C, Heft Nr. 530.

Brenner, C., 2003. Building Reconstruction from Laser Scanning and Images. In: Proceedings of the ITC Earth Observation Science Department Workshop on Data Quality.

Chrobak, M., Payne, T., 1989. A Linear-time Algorithm for Drawing a Planar Graph on a Grid. Information Processing Letters 54, 241–246.

Cook, S. A., 1971. The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing. STOC '71. ACM, New York, NY, USA, pp. 151–158.

Dakin, R., 1965. A Tree Search Algorithm for Mixed Integer Programming Problems. The Computer Journal 8, 250–255.

Dantzig, G., Fulkerson, D. R., Johnson, S., 1954. Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America 2 (4), 393–410.

Dantzig, G. B., 1951. Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation. Wiley, New York, Ch. XXI.

Döllner, J., Buchholz, H., 2005. Continuous level-of-detail modeling of buildings in 3D city models. In: Proceedings of the 13th annual ACM international workshop on Geographic information systems. GIS '05. ACM, New York, NY, USA, pp. 173–181.

Döllner, J., Buchholz, H., 2005. Expressive Virtual 3D City Models. In: International Cartographic Conference. CD proceedings.

Douglas, D. H., Peucker, T. K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization 10 (2), 112–122.

Downing, F., Flemming, U., 1981. The Bungalows of Buffalo. Environment and Planning B 8, 269–293.

Duarte, J., 2002. Malagueira Grammar – towards a tool for customizing Alvaro Siza's mass houses at Malagueira. Dissertation, MIT School of Architecture and Planning.

Eppstein, D., 2009. Graph-Theoretic Solutions to Computational Geometry Problems. CoRR abs/0908.3916.

ESRI, 2012. Esri CityEngine — 3D Modeling Software for Urban Environments. Software, URL: `http://www.esri.com/software/cityengine`, last accessed Aug. 31, 2012.

Estkowski, R., Mitchell, J. S. B., 2001. Simplifying a polygonal subdivision while keeping it simple. In: Symposium on Computational Geometry. pp. 40–49.

EU, 2007. Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE).

Fan, H., 2010. Integration of time-dependent features within 3D city model. Dissertation, Technische Universität München.

Fan, H., Meng, L., Jahnke, M., 2009. Generalization of 3D Buildings Modelled by CityGML. In: Advances in GIScience: Proceedings of 12th AGILE Conference on GIScience. Lecture Notes in Geoinformation and Cartography. Springer, pp. 387–405.

FICO Decision Management Community Forum, 2009. Xpress Optimization Suite. MIP formulations and linearization quick reference. Online Quick reference, URL: `http://brblog.typepad.com/files/mipformref-1.pdf`.

Fischer, A., 2005. Automatische Gebäuderekonstruktion mittels parametrisierter Komponenten. Dissertation, University of Bonn.
URL `urn:nbn:de:hbz:5-04965`

Flemming, U., 1987. More than the sum of its parts: the grammar of queen anne houses. Environment and Planning B 14, 323–350.

Forberg, A., 2005. Generalisierung dreidimensionaler Gebäudedaten auf der Basis von Maßstabsräumen. Dissertation, Universität der Bundeswehr, Munich, Germany.

Forberg, A., 2007. Generalization of 3D building data based on a scale-space approach. ISPRS Journal of Photogrammetry and Remote Sensing 62 (2), 104 – 111.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

Glander, T., Döllner, J., 2009. Abstract representations for interactive visualization of virtual 3D city models. Computers, Environment and Urban Systems 33 (5), 375 – 387.

Glander, T., Döllner, J., 2007. Cell-based generalization of 3D building groups with outlier management. In: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems. GIS '07. ACM, New York, NY, USA, pp. 54:1–54:4.

Gomory, R. E., 1958. Outline of an Algorithm for Integer Solutions to Linear Programs. Bulletin of the American Mathematical Society 64 (5), 275–278.

Gröger, G., Kolbe, T. H., Czerwinski, A., Nagel, C., 2008. OpenGIS City Geography Markup Language (CityGML) Encoding Standard, Version 1.0.0. Tech. Rep. Doc. No. 08-007r1, OGC, Wayland (MA), USA.

Gröger, G., Plümer, L., 2012. CityGML – Interoperable semantic 3D city models. ISPRS Journal of Photogrammetry and Remote Sensing 71 (0), 12–33.

Grigni, M., Manne, F., 1996. On the complexity of the generalized block distribution. In: Ferreira, A., Rolim, J., Saad, Y., Yang, T. (Eds.), Parallel Algorithms for Irregularly Structured Problems. Vol. 1117 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 319–326.

Götzelmann, T., Guercke, R., Brenner, C., Sester, M., 2009. Terrain-Dependent Aggregation of 3D City Models. In: Workshop on quality, scale and analysis aspects of city models, Lund, Sweden, International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences, Volume XXXVIII-2/W11. pp. 75–82.

Guercke, R., Brenner, C., 2009. A Framework for the Generalization of 3D City Models. In: Proceedings of 12th AGILE Conference on GIScience. Hannover, Germany, pp. CD–ROM.

Guercke, R., Brenner, C., Sester, M., 2009a. Generalization of 3d City Models as a Service. In: Proceedings of the ISPRS/COST Workshop on Quality, Scale and Analysis Aspects of City Models, Lund, Sweden, ISPRS XXXVIII-2/W11. pp. CD–ROM.

Guercke, R., Brenner, C., Sester, M., 2009b. Generalization of Semantically Enhanced 3d City Models. In: Proceedings of the GeoWeb 2009 Conference, Vancouver, Canada. pp. 28–34.

Guercke, R., Götzelmann, T., Brenner, C., Sester, M., 2011. Aggregation of LoD 1 building models as an optimization problem. ISPRS Journal of Photogrammetry and Remote Sensing 66 (2), 209 – 222.

Guercke, R., Sester, M., 2011. Building Footprint Simplification Based on Hough Transform and Least Squares Adjustment. In: Proceedings of the Joint Workshop Geographic Information on Demand. Joint ICA/ISPRS Workshop Generalization and Multiple Representation, pp. CD–ROM.

Guercke, R., Zhao, J., Brenner, C., Zhu, Q., 2010. Generalization of Tiled Models with Curved Surfaces Using Typification. In: Proceedings of the Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science. Vol. 38. ISPRS Technical Commission II Symposium IGU International Symposium on Spatial Data Handling IGU International Conference on Modelling Geographical Systems, Hongkong, pp. 39–44.

Guttman, A., Jun. 1984. R-trees: a dynamic index structure for spatial searching. SIGMOD Rec. 14 (2), 47–57. URL http://doi.acm.org/10.1145/971697.602266

Hake, G., Grünreich, D., Meng, L., 1996. Kartographie. Visualisierung raum-zeitlicher Informationen. Walter de Gruyter & Co., Berlin, Germany.

Haunert, J.-H., 2009. Aggregation in Map Generalization by Combinatorial Optimization. Dissertation, Leibniz Universität Hannover. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften. Reihe C. Dissertationen. Heft Nr. 626.

Haunert, J.-H., Wolff, A., 2006. Generalization of Land Cover Maps by Mixed Integer Programming. In: Proceedings of the 14th Int. ACM Symposiun: Advances in Geographic Information Systems (ACM-GIS '06). pp. 75–82.

Haunert, J.-H., Wolff, A., 2008. Optimal Simplification of Building Ground Plans. In: Proc. 21st Congress Internat. Society Photogrammetry Remote Sensing (ISPRS'08), Technical Commision II/3. Vol. XXXVII, Part B2 of Internat. Archives of Photogrammetry, Remote Sensing and Spatial Informat. Sci. Beijing, pp. 373–378.

Hausdorff, F., 1914. Grundzüge der Mengenlehre. Von Veit.

Havemann, S., 2005. Generative Mesh Modeling. Dissertation, Institute of Computer Graphics, Braunschweig Technical University, Germany.

Hopcroft, J., Tarjan, R., Oct. 1974. Efficient Planarity Testing. J. ACM 21 (4), 549–568.

IBM ILOG, 2011. CPLEX, Optimization, Operations Research, Mathematical Programming, Linear Programming, Integer Programming, Mixed Integer Programming, Quadratic Programming, Modeling and solving optimization problems. Website, URL: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Jaffe, A., 2005. The millennium grand challenge in mathematics. Harvard University.

JCGM, 2008. International vocabulary of metrology - Basic and general concepts and associated terms (VIM) Vocabulaire international de metrologie - Concepts fondamentaux et generaux et termes associes (VIM).

Jenks, G., 1989. Geographic logic in line generalisation. Cartographica 26 (1), 27–42.

Kada, M., 2007a. Scale-Dependent Simplification of 3D Building Models Based on Cell Decomposition and Primitive Instancing. In: Winter, S., Duckham, M., Kulik, L., Kuipers, B. (Eds.), Spatial Information Theory. Vol. 4736 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 222–237.

Kada, M., 2007b. Zur maßstabsabhängigen Erzeugung von 3D-Stadtmodellen. Dissertation, Universität Stuttgart, Germany.

Karp, R. M., 1972. Reducibility Among Combinatorial Problems. In: Miller, R. E., Thatcher, J. W. (Eds.), Complexity of Computer Computations. Plenum Press, pp. 85–103.

Khanna, S., Muthukrishnan, S., Paterson, M., 1998. On approximating rectangle tiling and packing. In: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms. SODA '98. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 384–393.

Kim, D. H., Yun, I. D., Lee, S. U., 2004. A new attributed relational graph matching algorithm using the nested structure of earth mover's distance. In: Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04). Vol. 1. pp. 48 – 51.

Kolbe, T. H., Gröger, G., Plümer, L., 2005. CityGML – Interoperable Access to 3D City Models. In: Proceedings of the first International Symposium on Geo-Information for Disaster Management. Springer Verlag, pp. 21–23.

Lamy, S., Ruas, A., Demazeu, Y., Jackson, M., Mackaness, W., 1999. The Application of Agents in Automated Map Generalisation. In: Proceedings of the 19th International Cartographic Conference of the ICA, Ottawa, Canada.

Land, A. H., Doig, A. G., 1960. An Automatic Method of Solving Discrete Programming Problems. Econometrica 28 (3), 497–520.

Lichtenstein, D., 1982. Planar Formulae and Their Uses. SIAM Journal on Computing 11 (2), 329–343.

Lorensen, W. E., Cline, H. E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques. SIGGRAPH '87. ACM, New York, NY, USA, pp. 163–169.

Luebke, D., Watson, B., Cohen, J. D., Reddy, M., Varshney, A., 2002. Level of Detail for 3D Graphics. Elsevier Science Inc., New York, NY, USA.

Mäntylä, M., Jan. 1986. Boolean operations of 2-manifolds through vertex neighborhood classification. ACM Trans. Graph. 5 (1), 1–29.

Mao, B., 2011. Visualisation and Generalisation of 3D City Models. Dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., Gool, L. V., 2006. Procedural Modeling of Buildings. Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics 25 (3), 614–623.

Moore, C., Robson, J., 2001. Hard Tiling Problems with Simple Tiles. Discrete and Computational Geometry 26 (4), 573–590.

Muthukrishnan, S., Poosala, V., Suel, T., 1999. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In: Proceedings of the 7th International Conference on Database Theory. ICDT '99. Springer-Verlag, London, UK, UK, pp. 236–256.

Nagel, C., Stadler, A., Kolbe, T. H., 2009. Conceptual Requirements for the Automatic Reconstruction of Building Information Models from Uninterpreted 3D Models. In: Academic Track of the Geoweb 2009 Conference. Vol. XXXVIII, 3–4 / C3 of the ISPRS Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences.

Nemhauser, G., Wolsey, L., 1988. Integer and combinatorial optimization. Wiley-Interscience series in discrete mathematics and optimization. Wiley.

Nichterlein, A., Dom, M., Niedermeier, R., 2011. Aspects of a multivariate complexity analysis for Rectangle Tiling. Oper. Res. Lett. 39 (5), 346–351.

Portele, C., 2007. OpenGIS Geography Markup Language (GML) Encoding Standard (OGC 07-036). Tech. rep., OGC.

Ribelles, J., p., H., Garland, M., Stahovich, T., Srivastava, V., 2001. Finding and Removing Features from Polyhedra. In: American Association of Mechanical Engineers (ASME) Design Automation Conference.

Riegl, Jan. 2013. RIEGL VMX-250.
URL `http://www.riegl.com/nc/products/mobile-scanning/produktdetail/product/scannersystem/6/`

Ripperda, N., 2010. Rekonstruktion von Fassadenstrukturen mittels formaler Grammatiken und Reversible Jump Markov Chain Monte Carlo Sampling. Dissertation, Leibniz Universität Hannover. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften. Reihe C. Dissertationen. Heft Nr. 649.

Ripperda, N., Brenner, C., 2009. Application of a Formal Grammar to Facade Reconstruction in Semiautomatic and Automatic Environments. In: Proceedings of 12th AGILE Conference on GIScience. Hannover, Germany.

Rubner, Y., Tomasi, C., Guibas, L. J., Nov. 2000. The Earth Mover's Distance as a Metric for Image Retrieval. International Journal of Computer Vision 40 (2), 99–121.

Rumbaugh, J., Jacobson, I., Booch, G., 2004. Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education.

Serra, J., 1983. Image Analysis and Mathematical Morphology. Academic Press, Inc., Orlando, FL, USA.

Sester, M., 2001. Maßstabsabhängige Darstellung in digitalen räumlichen Datenbeständen. Habilitation, Universität Stuttgart. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften. Reihe C. Dissertationen. Heft Nr. 544.

Sester, M., 2005. Optimizing Approaches for Generalization and Data Abstraction. International Journal of Geographic Information Science 19 (8-9), 871–897.

Sester, M., 2007. 3D Visualization and Generalization. In: 51st Photogrammetric Week. Stuttgart, Germany, pp. 285–295.

Sester, M., Neidhart, H., 2008. Reconstruction of Building Ground Plans from Laser Scanner Data. In: Proceedings of the 11^{th} AGILE conference. Girona, Spain, pp. CD–ROM.

Staufenbiel, W., 1973. Zur Automation der Generalisierung topographischer Karten mit besonderer Berücksichtigung großmaßstäbiger Gebäudedarstellungen. Dissertation, Wissenschaftliche Arbeiten der Lehrstühle für Geodäsie, Photogrammetrie und Kartographie an der Technischen Universität Hannover Nr. 51.

Thiemann, F., 2002. Generalization of 3D Building Data. Geospatial Theory, Processing and Applications 34 (4).

Thiemann, F., Sester, M., 2004. Segmentation of Buildings for 3D-Generalisation. In: Proceedings of the ICA Workshop on Generalization and Multiple Representation, Leicester, UK.

Thiemann, F., Sester, M., 2005. Interpretation of building parts from Boundary Representation. In: Proceedings of ISPRS Workshop Next Generation 3D City Models. Bonn, Germany, pp. pp. 29–34.

Thiemann, F., Sester, M., 2006. 3D-Symbolization Using Adaptive Templates. In: Proceedings of ISPRS Technical Commission II Symposium 2006. pp. 49–54.

Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W., 2003. Instant Architecture. ACM Transaction on Graphics 22 (3), 669–677, proceedings ACM SIGGRAPH 2003.

Acknowledgments / Danksagung

Diese Arbeit entstand im Rahmen meiner Tätigkeit als Wissenschaftlicher Mitarbeiter am Institut für Kartographie und Geoinformatik (IKG) der Leibniz Universität Hannover im Rahmen des Teilprojekts GDI-Grid im Rahmen der vom Bundesministerium für Bildung und Forschung (BMBF) geförderten D-Grid-Initiative.

Ich danke Frau Prof. Dr. Monika Sester dafür, dass sie mir die Mitarbeit auch über den Ablauf des Projektes hinaus ermöglicht hat und für die vielen wertvollen Anregungen. Sie hat es auf ihre stets freundliche Art immer wieder verstanden, mich wieder „einzufangen“ und auf einen zielführenden Weg zurückzuführen, wenn ich mich wieder einmal in dem Versuch verstiegen hatte, ein generisch erweiterbares und generalisierbares Weltmodell zu entwickeln. Ohne sie wäre diese Arbeit wohl niemals zu einem Ende gekommen.

Ich danke auch Herrn apl. Prof. Dr. Claus Brenner, der als stets sprudelnder Quell neuer Ideen mit seinem fundierten und interdisziplinären Fachwissen diese Arbeit bereichert hat, und allen anderen Kollegen am IKG für die stets angenehme Arbeitsatmosphäre und viele fruchtbare Diskussionen.

Darüber hinaus danke ich Herrn Prof. Dr. Kolbe für die Übernahme des Korreferats und seine wertvollen Anregungen, insbesondere zum Thema Stadtmodellierung. Herrn Prof. Dr. Christian Heipke danke ich für die Übernahme des zweiten Korreferats und seine vielfältigen Denkanstöße, die deutlich zur Steigerung der Qualität dieser Arbeit beigetragen haben. Darüber hinaus danke ich ihm besonders für die Möglichkeit, meine wissenschaftliche Karriere am IPI fortsetzen zu können.

Vor allem aber gilt mein Dank meiner Familie, insbesondere meiner Frau, für das Durchstehen der harten Zeiten bei der Entstehung dieser Arbeit, insbesondere der langen, unberechenbaren Abwesenheiten und der Nachtschichten während der Phase des Fernpendelns im letzten Jahr. An dieser Stelle sei auch unseren beiden kleinen Damen Julia und Annika meine Anerkennung ausgesprochen, die trotz des permanenten Ausnahmezustandes (mal mehr, mal weniger) fröhlich gedeihen.

Lebenslauf

Persönliche Daten

Name	Richard Guercke
Geburtstag und -ort	20.10.1978, Hannover
Staatsbürgerschaft	Deutsch
Familienstand	Verheiratet, 2 Töchter

Schulbildung

1985 - 1989	Grundschule Wasserkampstraße, Hannover
1989 - 1991	Orientierungsstufe Lüerstraße, Hannover
1991 - 1998	Kaiser-Wilhelm- und Ratsgymnasium, Hannover

Wehrdienst

1998 - 2000	Ausbildung zum Reserveoffizier (Minden)

Studium

2000 - 2002	Elektro- und Informationstechnik, Technische Universität München
2002 - 2005	Angewandte Informatik (Abschluss: B.Sc.), Leibniz Universität Hannover
2005 - 2007	Informatik (Abschluss: M.Sc.), Leibniz Universität Hannover
2007 - 2014	Promotionsstudium Geodäsie und Geoinformatik, Leibniz Universität Hannover

Beruflicher Werdegang

2000 - 2007	Studentische Hilfskraft an verschiedenen Instituten der Leibniz Universität Hannover
2007 - 2012	Wissenschaftlicher Mitarbeiter am Institut für Kartographie und Geoinformatik, Leibniz Universität Hannover
2013 - 2014	Volkswagen AG Konzernforschung, Wolfsburg
2014 -	Wissenschaftlicher Mitarbeiter am Institut für Photogrammetrie und GeoInformation, Leibniz Universität Hannover