Dipl.-Ing. Ulrich Krispel

Generative Methods for Data Completion in Shape Driven Systems

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. techn. Dieter W. Fellner

Institute of Computer Graphics and Knowledge Visualization (CGV)

Graz, August 2018

Abstract

In many application domains, such as building planning, construction, or documentation, it is of high importance to acquire a digital representation of the shape of real world objects, e.g. for visualization or documentation purposes. Such objects are often part of a class or domain of similarly structured objects; and often complex objects, such as houses, are composed by simpler objects, such as walls, doors and windows. Especially man-made objects exhibit such structure, mostly due to manufacturability and design reasons.

A rich digital representation of a complex object consists not only of its shape, but also its structure, i.e. the composition hierarchy of simpler objects. A more general way to represent such a composition hierarchy is a generative model, that generates the structure upon evaluation; a parametric generative model can generate a whole class of similarly structured objects.

In this thesis, I review shape-based methods for generative creation of models, and present a novel system for generative forward modeling based on shape grammars. Furthermore, I present two methods for solving the inverse problem: acquiring a rich digital representation of real-world objects from measurements and utilizing a generative model of prior domain knowledge. Using this prior knowledge, it is now possible to complete missing features, or reduce measurement errors. The first method parses the hierarchical structure of a building façade, given an ortho photo and a grammar that describes architectural constraints. The second method yields a hypothesis of electrical wiring inside walls, given optical measurements (point clouds and photographs), and a grammar that describes the technical standards.

Kurzfassung

In vielen Anwendungsdomänen wie zum Beispiel Visualisierung oder Dokumentation für Gebäudeplanung und -konstruktion, ist es von großer Wichtigkeit, eine digitale Repräsentation von real existierenden Objekten zu erfassen. Solche Objekte sind oft Teil einer Gruppe oder Klasse (die Domäne) von ähnlich strukturierten Objekten, und komplexere Objekte, wie etwa Häuser, sind oft aus simpleren Objekten zusammengesetzt, wie etwa Wände, Türen und Fenster. Dies ist besonders oft bei vom Menschen geschaffenen Objekten der Fall, aufgrund von Gestaltung und Fertigbarkeit.

Eine reichhaltige digital Beschreibung eines komplexen Objekts besteht daher nicht ausschliesslich aus seiner Form oder äußeren Hülle, sondern auch aus seiner Struktur beziehungsweise der Kompositionshierarchie von simpleren Objekten. Eine allgemeinere Möglichkeit um solche Kompositionshierarchien zu beschreiben ist ein generatives Modell. Ein solches Modell generiert die Hierarchie oder Struktur bei seiner Auswertung. Ein parametrisches generatives Modell kann somit eine ganze Klasse von ähnlich strukturierten Objekten generieren.

In dieser Arbeit bespreche ich formbasierte Methoden zur generativen Erzeugung von Kompositionshierarchien, und präsentiere eine neuartige Shape-Grammatik für generative Vorwärtsmodellierung. Weiters stelle ich zwei Methoden vor, um das inverse Problem zu lösen: Die Akquise einer reichhaltigen digitalen Beschreibung von real existierenden Objekten durch die Weiterverarbeitung von Meßdaten. Zur Erzeugung dieser Struktur werden generativen Modellen miteinbezogen, welche das Vorwissen über die Objektklasse repräsentieren. Damit wird es möglich fehlende Meßdaten zu ergänzen oder Meßfehler zu dezimieren. Die erste Methode parst die hierarchische Struktur einer Gebäudefassade, ausgehend von einem Orthofoto der Fassade und einer Grammatik welche die architektonischen Richtlinien beschreibt. Die zweite Methode liefert eine Hypothese von elektrischen Leitungen in Wänden, anhand von Distanzmessungen, optischen Meßdaten (Fotos), und einer Grammatik welche die technischen Normen zur Leitungsverlegung beschreibt.

Acknowledgment

This work builds upon many discussions with my supervisors and colleagues; my thanks go to Mr. Fellner for always providing me with helpful comments end encouragement. Furthermore, I want to thank Torsten Ullrich and Sven Havemann for the many interesting discussions and guidelines. I also want to thank Wolfgang Thaller, Bernhard Hohmann, and Christoph Schinko – with whom I shared offices, worked on collaborative projects and had many insightful discussions.

The methods and results presented in this work were partly funded by the following authorities and projects: The Austrian Research Promotion Agency (FFG) and Microsoft Photogrammetry for funding the research project CITYFIT ("High-Quality Urban Reconstructions by Fitting Shape Grammars to Images and derived Textured Point Clouds"), grant number 815971/14472. The European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 600908 ("DURAARK - Durable Architectural Knowledge"), 2013-2016 and the Fraunhofer Discover project GENMOD - "Generative, Rule Based Modeling of the Real World for Virtual Purposes". The Austrian Research Promotion Agency (FFG) for the research project AEDA (K-Projekt "Advanced Engineering Design Automation").

This thesis would not be possible without the work of many people I had the joy of working with in the last years; First, the CITYFIT project: the raw scan data (Lidar measurements and photographs) was acquired and kindly provided by Microsoft Photogrammetry, thanks go to Konrad Karner and his team; The data preprocessing, façade ortho photo generation (Figure 4.1) and initial semantic façade classification was done by Hayko Riemenschneider; my project co-workers Bernhard Hohmann and Wolfgang Thaller. I also want to thank my colleagues from the DURAARK project: Stefan Dietze, Jakob Beetz and Martin Tamke as project leads; Robert Viehauser from ICG for the implementation of the socket detection pipeline for *ElecDetect*; Henrik Leander Evers, Per-Kristian Hanson and Marcus August Frølich Innvær from CITA for the acquisition of the 3D scan data set, panoramic images and ground truth for the electrical line data set described in Section 4.2; the method for extracting an augmented floor plan representation was kindly provided by Sebastian Ochmann from UBO; Martin Hecher implemented the DURAARK service platform and the DURAARK workbench. Thanks also go to the architecture office ORTLOS engineering for their models in the GANDIS project.

Finally, I would like to thank my family, friends, and colleagues that supported me throughout the development of this thesis: my parents Theresia and Gottfried, Helmut Fresl Sensei, Matzi (TTD!), Erich, Andy, walx, – and Nicoletta for her unconditional support. Thank You.

Contents

1 Introduction

Digital methodologies have become ubiquitous in today's world, particularly in all domains that are concerned with a representation of the physical appearance of an object – the *shape*. Due to the growing availability of efficient 3D scanning methods, the costs of acquiring measurements of an object have decreased, and more scanned shape data is available. These measurements consist typically of large sets of individual 3D scan points of the measured surface, which are called point clouds. A connected surface can be obtained from these point clouds using a surface reconstruction algorithm – but an underlying abstraction, or semantic information about the shape is not directly accessible from this data. Therefore, extracting geometric and semantic information from such unstructured data is a very wide, and very active area of research. In this thesis, I review and analyze shape-based methods that utilize prior knowledge, a "vocabulary" of a specific domain such as architecture. This knowledge is utilized to extract domain information from measured data, e.g. the structure of a building façade from a photograph. I present contributions in the areas of generating shapes using such domain-specific vocabularies and extracting geometric and semantic information for two specific applications in the domain of building modeling.

1.1 Generative Modeling

Connecting geometric shape representation with corresponding semantic information is getting more prominent, as these metadata are instrumental for many further processing tasks. Application domains that utilize enriched shape data include computer-aided design (CAD), computer-aided engineering (CAE) and computer-aided manufacturing (CAM), or the digital representation of all building aspects – building information modeling (BIM) – to name a few. Each of these domains uses a digital representation of shape, the *model*. The creation of such models is often a resource demanding process, in which *generative methods* – algorithms that generate a model – can help to improve the quality of a model and the speed of its construction process. Such methods have become popular in several domains where it is desirable to create specific instances of an object, the *model*, which belongs to a family of objects, the *domain*. To name a few examples, in visualization and the movie and games industry such methods have been used to create instances of plants and buildings, up to whole cities. Generative techniques are used in games to create varied instances of levels or game-play. In Computer Aided Design (CAD), parametric modeling is used to describe real-world behavior of components, a designer can alter a few parameters to adjust the model and dependent shapes are updated accordingly.

The procedure of obtaining a specific instance of an object, which is part of a family of objects, is a frequent use scenario. A practical example is the creation of a detailed model of a building façade from an observation, e.g. a photograph, as seen in Figure 1.1. It depicts a façade in the city center of Graz, Austria. Creating a digital representation is helpful in a variety of scenarios: Visualization for navigation, documentation or architectural planning. The digital version can also be used to display enhanced or altered variants of the object. A classical manual reconstruction workflow consists of a human using a 3D modeling suite to create the digital 3D model. The suite provides a set of tools for constructing basic shapes such as cubes or spheres, or more complex tools, like directly modifying the surface mesh representation, e.g. adding or modifying vertices and facets. Modeling the façade could proceed in the following fashion: Starting with a large rectangular block that represents the complete façade, holes would be cut into the place where windows will be placed, facade elements, like windows, doors and ornaments would be modeled in detail afterwards. Coding this construction process for a model, e.g. using a scripting language, is called forward generative modeling, and obtaining a generative model from an real-world observation or a coarse specification of high-level parameters is called inverse generative modeling.

Forward Generative Modeling

The architectural components of the example façade from Figure 1.1 show several symmetries: There is a slightly protruding middle column of the façade, with columns on the left and right that use a symmetric layout. All windows use similar wood frames and glass configurations, the keystone ornaments above the windows are the same for columns and floors. A human modeler could reduce the modeling effort by making use of these symmetries: modeling the inner window details (glass and frame) only once, and placing it in each of the different frames with keystone ornaments using copy and paste. Furthermore, only the middle column and one side column need to be modeled, because the second side column can be copied and pasted, and mirrored if necessary, as well.

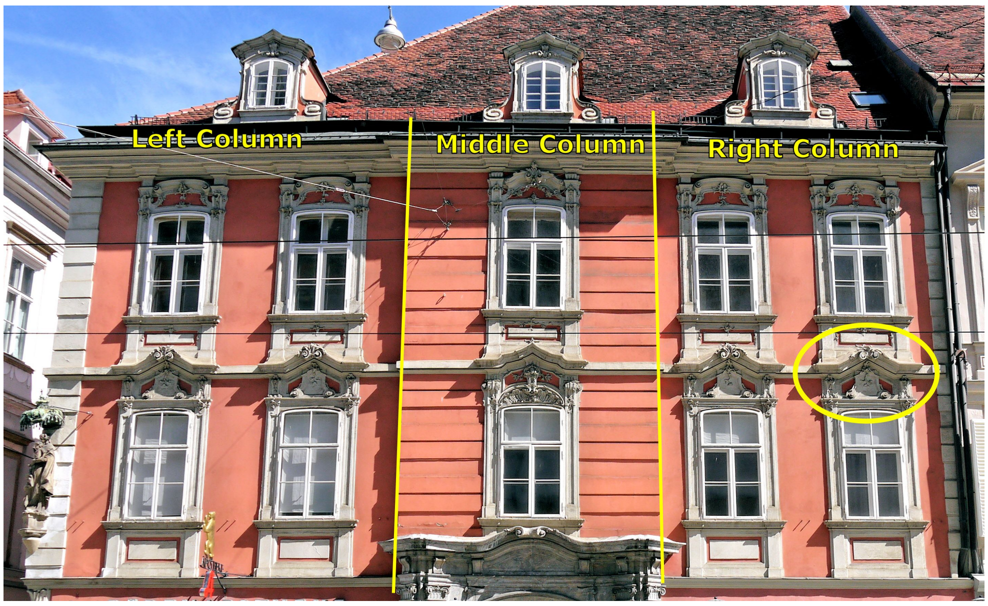
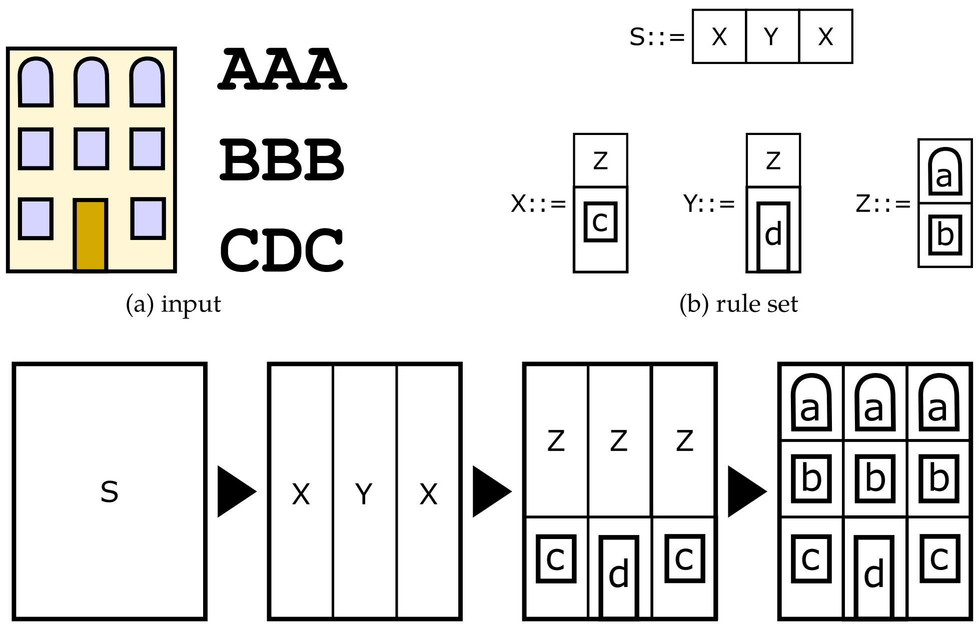


Figure 1.1: An example of a façade in the city of Graz, Austria. We can observe several symmetries in the architectural design: similar windows with different ornaments above – see also the encircled example – a middle column with symmetric parts left and right, and similarly structured elements like columns and ornaments (photo by author).

The steps carried out when modeling the façade can be seen as an algorithm that utilizes tools of the 3D modeling suite – which is the *generative model* of this façade. Such a generative representation can be desirable in several scenarios:

- A generative model can encode a class of shapes using parameters – various shapes can be created by re-evaluation using different parameter values.
- Machine code evaluation is less error prone than manual modeling.
- The generative model is often extremely small, compared to the evaluated geometry and therefore a compact representation.

Creating such representations is also associated with increased initial costs – creating a generative model of a single façade may be costly, but when modeling many façades (e.g., a city), the effort will soon pay off. Still, just having an algorithmic description does not automatically reduce modeling effort, therefore, it is desirable to have a set of composable – and hence reusable – generative components. Examples of such generative components would be windows that automatically adapt to a given space, while maintaining a fixed width for its framing – a feature that cannot be accomplished by copy and paste with rigid transformations.



(a) input (b) rule set

(c) parse tree

Figure 1.2: Reconstructing structure: First, an input observation is preprocessed using a classifier for façade elements such as windows and doors (a). Given a set of rules that describes valid façades (b) the inverse problem retrieves a concrete rule application hierarchy (c) that describes the observation.

Inverse Generative Modeling

The inverse approaches described in this thesis obtain a concrete generative model, i.e. rules and their parameters, given an observation (e.g., a photograph such as in Figure 1.1) and prior knowledge about the model structure. This prior knowledge is described generatively, and constrains the solution search space. For instance, a class of façades is described by a set of rules, a formal grammar system. This model is composed using structural information such as floor partitioning or symmetric subdivision of the façade, like repetition of similar elements. Similar to syntax parsing in compiler construction, the inverse modeling approach will yield the concrete steps of rule application from the given grammar, that will produce a façade most similar to the observation. This is demonstrated in Figure 1.2. This generative representation has several advantages: it can be transformed to a generative forward model using code transformation tools; this generative model, when evaluated, will produce a 3D model which is always valid. Furthermore, the parse tree can be analyzed to gain insight into the observation, e.g. extract the number of floors or examine found symmetries.

To summarize, the main problem is to extract geometric and semantic shape information from real-world measurements. For example, the geometric information may be a 3D model of a façade, the semantic information may be the number of floors and the symmetries of the façade structure. Now, the idea is to utilize specific domain knowledge to reduce the search space. Typical approaches in this problem field encode the domain knowledge directly in the reconstruction or post-processing algorithms. The approaches presented in this thesis are based on the idea to utilize a more general description to express the domain knowledge. This is done with a set of rules that encodes "good" or "possible" (valid) configurations. Creating a reconstruction algorithm that utilizes such a generative description is a powerful tool, as it decouples the information extraction step from the description of prior knowledge. This allows to easily exchange the domain knowledge without the need to modify the recognition algorithm. Thus, the proposed method is to describe domain knowledge by generative modeling and fit to measured data.

1.2 Contribution

This thesis deals with the application of generative methods for shape creation in the context of data completion, mainly in the domain of building modeling. It contains three main contributions

- As first contribution, I describe a new generative forward modeling system that is flexible, fast and robust. It uses a novel geometric representation (convex polyhedra) in a shape grammar approach.

- In the second contribution, I utilize a two-dimensional split grammar to describe a façade style prior and to parse the structure of a given façade photo. Then I describe an automatic conversion of these parse trees to a generative forward model using the modeling approach shown in the first contribution.

- In the third contribution, I derive a hypothesis of the layout of electrical wiring in indoor situations from preprocessed terrestrial laser scans, photographs, and a generative specification: a 2D grammar of technical standards. Augmented floor plans are extracted from the measurement data, and the positions of observable endpoints (sockets and switches) are detected using a pre-trained classifier. From these input data, a hypothesis for power wiring connecting the observable endpoints is deduced.

1.3 Overview

This thesis is structured as follows: Chapter 1 motivates and introduces the methods developed in this work, and gives an overview of the main contributions. Chapter 2 summarizes important techniques and related work that have been instrumental for the development of the proposed methods. Chapter 3 is divided in two parts; in the first part, I give a thorough description of the developed generative forward modeling system, and discuss the robust and fast implementation of surface evaluation for the proposed system. The second part of Chapter 3 is concerned with an application of this shape grammar to the domain of building exterior modeling. Chapter 4 summarizes my research on inverse generative modeling. First, a method for exterior façade reconstruction by parsing shape grammars on façade orthophoto classifications is presented; the second method describes the reconstruction of electrical wiring in office spaces using a generative representation of technical standards and preprocessed measurements from terrestrial laser scanning. In Chapter 5, I show examples of collaborative works that utilize the proposed methods. The results of this thesis are discussed in Chapter 6, which is concluded with an outlook on future work and some open questions.

2 Background

This chapter summarizes important techniques and foundations that have been relevant for this thesis. Most of this chapter appeared as *"The rules behind - Tutorial on Generative Modeling"* at the Symposium for Geometry Processing, 2014 [108] and *"A Survey of Algorithmic Shapes"* [109].

2.1 Introduction to the Generative Paradigm

In the context of computer-aided design (CAD) and shape description, the digital creation of a shape is called modeling. The most common representation of a shape is a composition of elementary objects. However, a shape can also be described by its generating process. In this case, the description is called a **generative model**. Note that there is also a paper titled *Generative Modeling: a symbolic system for geometric modeling* by Snyder and Kajiya [172] which describes an early version of a symbolic shape design system; shapes are created by transformation of another shape called the generator shape. The term *generative model* in this thesis is used in a broader context and stands for any generating process. A generative model does not describe a shape by the parts it consists of, but by the operations and steps needed to be performed in order to create it; in other words, a generative model is an algorithm. Its implementation is an algorithmic description written in a programming language. Depending on the used software engineering paradigm, a generative model may also be called a **procedural model** or a **functional model**, if the algorithm is implemented procedurally, respectively functionally. For many purposes in CAD, the mightiness of a Turing-complete programming language may lead to potential problems, such as the halting problem. In order to avoid these problems, CAD frameworks often offer a language that is not Turing-complete; i.e., the set of language features is reduced to **parametric modeling**. Note that in the literature the term **procedural modeling** is often used ambiguously, describing all kinds of generative models.

In generative modeling, the object is not just the end result of applied operations, as this paradigm describes a shape by a sequence of processing steps. The result is a paradigm shift from shape design to rule design. This approach is very general and can be applied to any domain, as well as using any shape representation that provides a set of generating functions.

In many cases, it is desirable not only to automate the shape construction, but to create an abstraction that allows expressing semantics within the construction, or as E. W. Dijkstra put it [43]:

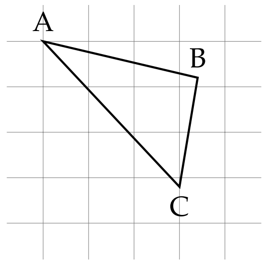
> *In this connection it might be worth-while to point out that the purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.*

I follow with some examples of domains with established practice of generative design.

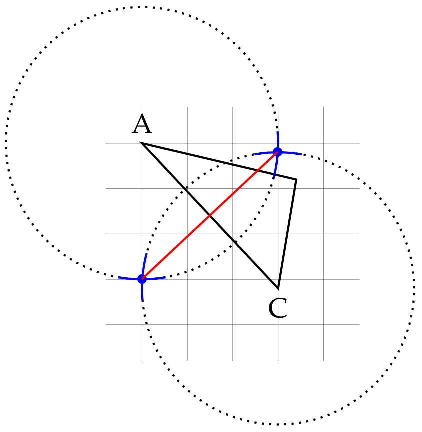
Compass and Ruler

The ruler-and-compass construction is the construction of lengths, angles, and other geometric figures using only an idealized ruler and compass. Geometry from the days of the ancient Greeks placed great emphasis on problems of constructing various geometric figures using only a ruler without markings (to draw lines) and a compass (to draw circles). All ruler-and-compass constructions consist of repeated application of five basic constructions based on Euclid's axioms [74] using the points, lines and circles that have already been constructed. It turns out that all constructions possible with a compass and straightedge can be done with a compass alone, as long as a line is considered constructed when its two endpoints are located [209]. The reverse is also true, since Jacob Steiner showed that all constructions possible with straightedge and compass can be done using only a straightedge, as long as a fixed circle and its center have been drawn beforehand. Such a construction is known as a Steiner construction. Based on these geometric primitives and a fixed set of operations, the ruler-and-compass constructions – such as illustrated in Figure 2.1 – have been the first algorithmic descriptions of generative models. This kind of constructions were made available on computers to teach geometry and trigonometry, one of the first being *Cabri Geometry*[1]. A free version of such a construction program is included in the open source *K Education Project* under the name *Kig*[2].
The long history of geometric constructions [125] is also reflected in the history of civil engineering and architecture [132]. For example, Gothic architecture and especially window tracery exhibits quite complex geometric shape configurations. But this complexity is achieved by combining only a few basic geometric patterns. Sven Havemann and Dieter W. Fellner present some principles of this long-standing domain, together with some delicate details, and show how the constructions of some prototypical Gothic windows can be formalized using their

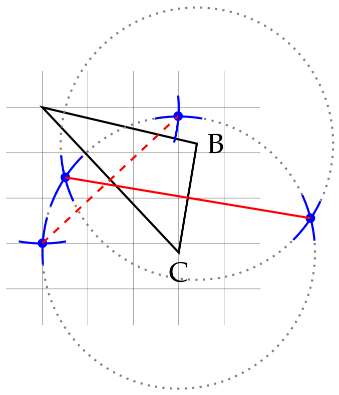
[1] `http://www.cabri.com/`
[2] `https://edu.kde.org/applications/mathematics/kig/`



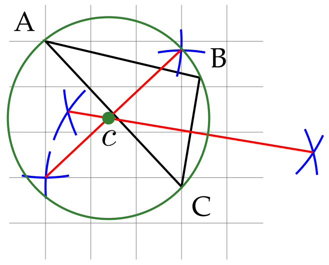
(a) Triangle ABC



(b) Construct perpendicular to AC



(c) Construct perpendicular to BC



(d) Circumcircle center at intersection

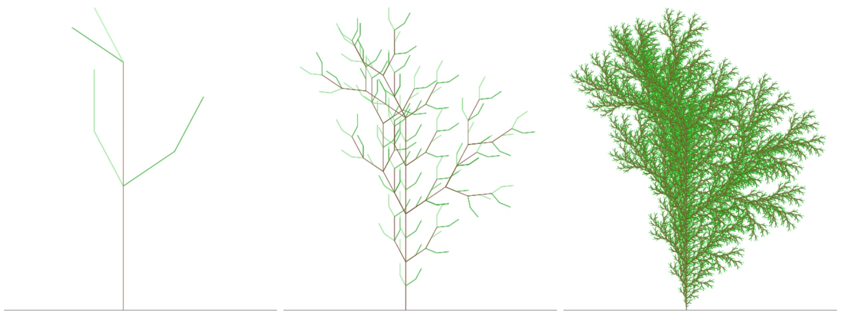
Figure 2.1: A simple example of a generative model using compass and ruler constructions: The circumcircle of any non-degenerate Triangle ABC is constructed by creating perpendicular lines to any two triangle edges, in this case AC and BC. The intersection of these lines is the center c of the circumcircle of ABC, the radius corresponds to the distance between c and any triangle vertex.

generative modeling techniques [71]. Using modularization, so that complex configurations can be obtained from combining elementary constructions, different combinations of specific parametric features can be grouped together, which leads to the concept of styles. They permit differentiating between the basic shape and its appearance, i.e., in a particular ornamental decoration [189]. This leads to an extremely compact representation for a whole class of shapes [21].

Natural Patterns: Fractals and L-Systems

Benoit B. Mandelbrot of "The Fractal Geometry of Nature" [122] developed a theory of self-similar sets, the fractals, that exhibit similarities to natural phenomena. Such sets may be constructed by iterated function systems (IFS) which were developed by John Hutchinson [87]. An extensive overview of fractal geometry and iterated function systems is given in the book "Fractals Everywhere" of Michael F. Barnsley [14].

In today's generative modeling systems, scripting languages and grammars are often used as a set of rules to achieve a description. Early systems based on grammars were Lindenmayer systems [155] (L-systems) named after the biologist Aristid Lindenmayer. They were successfully applied to model plants. Given a set of string rewriting rules, complex strings are created by applying these rules



The L-System is described by:

- The axiom: FX
- The angle: $28°$
- Two rules:
  $F \mapsto C_0FF - [C_1 - F + F] + [C_2 + F - F]$
  $X \mapsto C_0FF + [C_1 + F] + [C_3 - F]$

whereas F denotes "draw forward" and $+/-$ denote "turn left"/"turn right". The square bracket $[$ corresponds to saving the current values for position and angle, which are restored when the corresponding square bracket $]$ is executed. C_0, C_1, C_2 switch colors and X does not correspond to any drawing action. This example can be executed online by Kevin Roast's L-Systems-Demo:
`http://www.kevs3d.co.uk/dev/lsystems/`

Figure 2.2: Lindenmayer systems are a simple but elegant "turtle rendering" platform. The recursive nature of L-system rules lead to self-similarity and thereby fractal-like forms. Plant models and natural-looking organic forms "grow" and become more complex by increasing the iteration level – i.e., the number of substitutions.

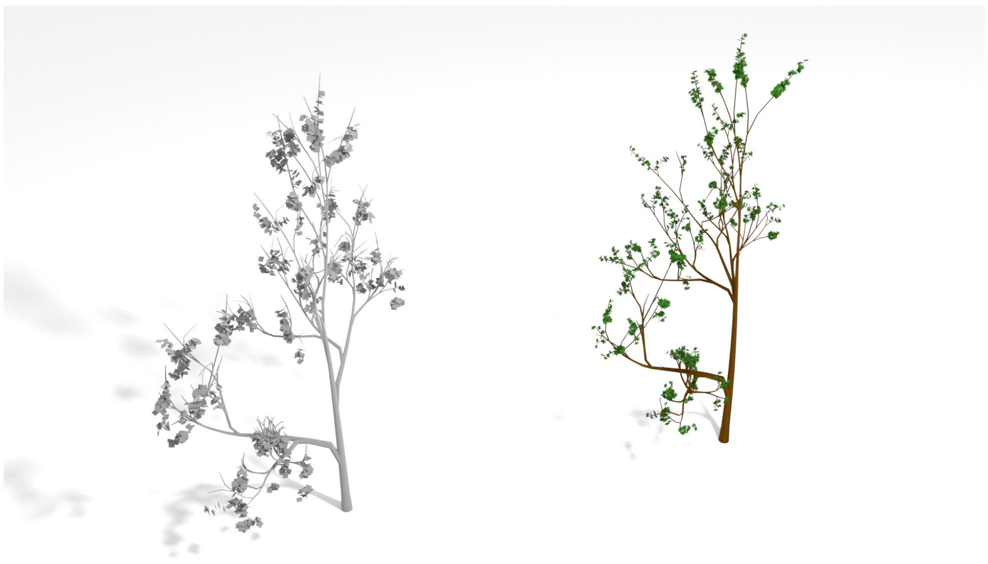


Figure 2.3: L-Systems may also be used to grow 3D geometry, as shown in this figure. It first grows the trunk and branches of the tree and instantiates quads for leaves (left). Assigning materials to branches and leaves yields a more realistic rendering (right). This example was generated using the *VegGen* plugin for Blender from Tom Trval.

to simpler strings. Starting with an initial string the predefined set of rules form a new, possibly larger string. The L-systems approach reflects a biological motivation. In order to use L-systems to model geometry an interpretation of the generated strings is necessary.

The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures (see Figure 2.2). This lead to the introduction of parametric L-systems. The idea is to associate numerical parameters with L-system symbols to address continuous phenomena which were not covered satisfactorily by L-systems alone.

Combined with additional 3D modeling techniques, Lindenmayer systems can be used to generate complex geometry [193], [194]. In order to generate models of plants, terrains, and other natural phenomena that are convincing at all different scales, Robert F. Tobler et al. introduce a combination of subdivision surfaces, fractal surfaces, and parameterized L-systems, which makes it possible to choose which of them should be used at each level of resolution. Since the whole description of such multi-resolution models is procedural, their representation is very compact and can be exploited by level-of-detail renderers that only generate surface details that are visible.

This kind of data amplification can be found in various fields of computer graphics. E.g. curved surfaces specified by a few control points are tessellated directly on the GPU. This results in low storage costs and allows generating the complex model only when needed, while also reducing memory transfer overheads. Although L-systems are parallel rewriting systems, derivation through rewriting leads to uneven workloads. Furthermore, the interpretation of an L-system is an inherently serial process. Thus, L-systems are not straightforwardly amenable to parallel implementation. In 2010, Markus Lipp et al. presented a solution to this algorithmic challenge [120].

Art and Entertainment

George Stiny coined the term **shape grammar**, when he presented a formal approach to capture the design of two-dimensional paintings in his seminal work *"Shape grammars and the generative specification of painting and sculpture"* [178]. An example for another method of formalizing art is presented by Henderson [75], where he deconstructs the woodcut *Square Limit* of M.C. Escher using a functional approach. The annual *Generative Art Conference* [173] presents artworks and life performances in addition to conference talks (papers).

The *demoscene* is a computer related subculture that specializes in producing *demos*, which are programs that perform audio-visual presentations. This scene has been active since the advent of personal computers, for a brief history I refer to the book of Tomas Polgar [153]. The creation of such programs is often tied to various constraints, as memory and general hardware capabilities of the early personal computers were quite limited. Nowadays, these constraints are mostly of self-imposed nature, e.g. creating a demo using only a file size of 4096 bytes (or 4kB). Naturally, these constraints facilitate the usage of procedural methods for any type of presentation content: 3D geometry (meshes), textures [49], sound, and so on.

Even without such limitations, various forms of content creation [197] have always been the main field of application for procedural techniques: from game design [9], [38], [93], [151], [196], and virtual worlds [163], to non-geometry aspects such as story-telling and drama management [139], camera movements [98] and player [195] as well as artificial intelligence modeling [12], [85].

Although procedural methods arose in the field of early video game development, from a historical point of view their use was not common in the special effects and feature animation community.

While computer-generated content made first appearance in movies in the late 70s, it was not until the 90s, most notably with the movies *Terminator 2* and *Jurassic Park*, that the film industry started using 3D computer graphics for content authoring and animation. Nowadays, procedural effects for 3D games are similar to procedural effects for movie productions. Differences can be found in the degree of visual fidelity, but these are mainly caused by the real-time demands of 3D games.

Visual artists use 3D animation tools to create procedural effects. Some recent examples are the approach presented by Daniel Heckenberg et al.[73] to create, animate and render repetitive geometric features such as scales or cobblestones developed for the movie *Walking With Dinosaurs 3D*. These tools were used to realize hundreds of different dinosaur characters from nine species. Another example is an algorithm to create procedural tentacle bundles for an alien creature in the movie *Edge of Tomorrow* that was presented by Dan Sheerin [168]. It allows an entire tentacle bundle to be defined by a base curve and a list of parameters.

Generative Architectural Design

In many cases, there is a kind of structural interrelation with a design idea of an architect, for example, in classical architectural design [34] there are fixed ratios for lengths in the design of columns; classical architecture exhibits much regularity, e.g. repetitions or symmetries. Utilizing such regularities makes the domain of design and planning of buildings much suited for a generative description. The term *generative design* is used architecture to describe algorithm-based design techniques.

The usage of generative modeling techniques in architecture is not limited to buildings of the past [133], [134]. Over the last few decades, architects have used a new class of design tools that support generative design. Generative modeling software extends the design abilities of architects and may even help to reduce costs by harnessing computing power in new ways. Computers, of course, have long been used to capture and implement the design ideas of architects by means of CAD and 3D modeling. Generative design actually helps architects design by using computers to extend human abilities [82].

An impressive example is the Helix Bridge in Singapore (see Figure 2.4). This 280 m bridge is made up of three 65 m spans and two 45 m end spans. The major and minor helices, which spiral in opposite directions, have an overall diameter of 10.8 m and 9.4 m respectively. The outer helix is formed from six tubes which are set equidistant from one another, whereas the inner helix consists of five tubes. This bridge design is the product of inseparable collaboration between architects (Cox Architecture and Architects 61) and civil engineers (Arup Consultant). For its 280 m length, the dual helix structure of the bridge utilizes five times less steel than a conventional box girder bridge. This fact enabled the client to direct the structure to be constructed entirely of stainless steel for its longevity.

Another example of generative, architectural design has been presented by Torsten Ullrich et al. [202]. They interpret a generative script as a function, which is nested into an objective function. Thus, the script's parameters can be optimized according to an objective. They demonstrate this approach using architectural examples: each generative script creates a building with several free parameters. The objective function is an energy-efficiency-simulation that approxima-

Figure 2.4: The Helix Bridge is a pedestrian bridge in the Marina Bay area in Singapore. Its generative design has been optimized numerically. Furthermore, the bridge was fully modeled in order to visualize its form and geometric compatibility, as well as to visualize the pedestrian experience on the bridge. Photo kindly provided by Volker Settgast.

tes a building's annual energy consumption. Consequently, the nested objective function reads a set of building parameters and returns the energy needs for the corresponding building. This nested function is passed to a minimization and optimization process. Outcome is the best building (within the family of buildings described by its script) concerning energy-efficiency. The contribution is a new way of modeling: The generative approach separates design and engineering. The complete design is encoded in a script in a way that ensures that all parameter combinations (within a fixed range) generate a valid design. Then the design can be optimized numerically.

The adjustment of architectural forms to local and specific conditions is a fundamental study. When discussing energy consumption and solar power harness in buildings, important aspects have to be taken into account, e.g., the relation between a building form and its energy behavior, and the local weather conditions on an all-year basis. Several studies were published so far, trying to answer these questions. "Form follows energy" has become an omnipresent dogma in architecture, but its realization is difficult. The manual analysis of the various relations between form, volume, and energy consumption has to face many – not only numerical – problems.

The new approach by Torsten Ullrich et al. [202] for architectural design is opening the door to new possibilities for the user. It relieves the user from additional, interdisciplinary burdens: the designer can concentrate on the design, while the civil engineer can focus on engineering aspects. This new approach based on procedural modeling can be used in various fields of product design.

2.2 Languages and Grammars

Originally, scripting languages have been designed for a special purpose, e.g., to be used for client-side scripting in a web browser. Nowadays, the applications of scripting languages are manifold. JavaScript, for example, is used to animate 2D and 3D graphics in VRML [29] and X3D [17] files. It checks user forms in PDF files [27], controls game engines [42], configures applications, defines 3D shapes [165], and performs many more tasks. According to John K. Ousterhout scripting languages use a higher level of abstraction compared to system programming languages as they are often type-less and interpreted to emphasize the rapid application development purpose [145]. Whereas system programming languages are designed for creating algorithms and data structures based on low-level data types and memory operations. As a consequence, low-level graphics libraries [143], graphics shaders [140] and scene graph systems [159], [206] are usually still written in C/C++ dialects [50], and procedural modeling frameworks use scripting languages such as Lua, JavaScript, C#, etc.

Language Processing & Compiler Construction

The evaluation of procedural descriptions typically utilizes techniques used for description of formal languages and compiler construction [148]. There exists a wide range of different language concepts to describe a shape, which also comprehends all kinds of linguistic concepts [35]. The main categories to describe a shape are

- rule-based: using sets of substitutions and substitution rules to build complex structures out of simple starting structures [146], [103], [135], [172].
- imperative and scripting-based: using a scripting engine and techniques used in predominant programming languages [69], [165], [102], or
- GUI and dataflow-based: using new graphical user interfaces (GUI) and intelligent GUIs to detect structures in modeling tasks, which can be mapped onto formal descriptions [119], [187].

Nevertheless, the general principles of formal descriptions and compiler construction are in all cases the same – independent of ahead-of-time compilation, just-in-time compilation or interpretation [166].
From a historical point of view, the first procedural or generative modeling systems were Lindenmayer systems [155], or L-systems for short. These early systems, based on grammars, provided the means for modeling plants. The idea behind it is to start with simple strings and create more complex strings by using

a set of string rewriting rules. The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures.

Later on, L-systems are used in combination with shape grammars to model cities [147]. Yogi Parish and Pascal Müller presented a system that generates a street map including geometry for buildings given a number of image maps as input. The resulting framework is known as *CityEngine* – a modeling environment for *CGA Shape*. Also, based on *CGA Shape*, Markus Lipp et al. presented another modeling approach [119] following the notation of Pascal Müller [135]. It deals with the aspects of more direct local control of the underlying grammar by introducing visual editing. Principles of semantic and geometric selection are combined as well as functionality to store local changes persistently over global modifications.

Sven Havemann takes a different approach to generative modeling. He proposes a stack based language called *Generative Modeling Language* (GML) [69]. The postfix notation of the language is similar to that of *Adobe Postscript*. Havemann calls this approach generative mesh modeling; a shape is constructed by a small set of operations: the *euler operators*. These operators are a closed and complete set of five local mesh modification operations that preserve the Euler-Poincaré characteristic of a shape.

Generative modeling inherits methodologies of 3D modeling and programming [200], which leads to drawbacks in usability and productivity. The need to learn and use a programming language is a significant inhibition threshold especially for archaeologists, cultural heritage experts, etc., who are seldom experts in computer science and programming. The choice of the scripting language has a huge influence on how easy it is to get along with procedural modeling. *Processing* is a good example of how an interactive, easy to use, yet powerful, development environment can open up new user groups. It has been initially created to serve as a software sketchbook and to teach students fundamentals of computer programming. It quickly developed into a tool that is used for creating visual arts [157].

Processing[3] is basically a Java-like interpreter offering new graphics and utility functions together with some usability simplifications. A large community behind the tool produced libraries to facilitate computer vision, data visualization, music, networking, and electronics. Offering easy access to programming languages that are difficult to approach directly reduces the inhibition threshold dramatically. Especially in non-computer science contexts, easy-to-use scripting languages are more preferable than complex programming paradigms that need profound knowledge of computer science. The success of *Processing* is based on two factors: the simplicity of the programming language on the one hand and the interactive experience on the other hand. The instant feedback of scripting

[3] `https://processing.org/`

environments allow the user to program via “trial and error”. In order to offer our users this kind of experience, we enhanced our already existing compiler to an interactive environment for rapid application development.

Scripting Languages for Generative Modeling

There exists a broad variety of tools and techniques for procedural modeling. We provide an overview of a collection of generative modeling techniques (see Table 2.1 and 2.2) under the following aspects:

application domain: Generative modeling tools often incorporate prior knowledge of a specific application domain, e.g. generative modeling of architecture [178], or modeling of organic structures [122], [155], which is reflected in this aspect.

programming category: Some methods are building on conventional programming languages, or scripting languages. On the contrary, some techniques are built using proprietary languages, such as rule-based systems for buildings [213], or [135] for urban modeling. Some systems can be used even without any scripting, e.g. graph-based languages [189], or the visual interactive editing of split grammars [119].

environment: This aspect covers the tool set that provides geometric entities and operations, for example the geometry kernel of a 3d modeling software, e.g. the open source modeling suite blender or a proprietary system such as shape grammars on convex polyhedra [190].

There exist various programming paradigms in software development. Therefore, they also apply to the field of generative modeling, where some paradigms emerged to be useful for specific domains.

imperative: In many cases, generative modeling is carried out using classical programming paradigms: A programming language is used to issue the commands that generate a specific object using a library that utilizes some sort of geometry representation and operations to perform changes. An example are compass and ruler systems used by an imperative language. Thus, any modeling software that is scriptable by an imperative language, or provides some sort of API, falls into this category. Note that the resulting geometry is often produced as side effects.

dataflow based: The program is represented as a directed graph of the data flowing between operations [86]. The graph representation also allows for a graphical representation; Visual Programming Languages (VPL) allow to create a program by linking and modifying visual elements, many VPL’s are based on the dataflow paradigm. Examples in the domain of generative modeling are the Grasshopper3D plug-in for the Rhinoceros3D modeling suite, or the work of Gustova Patow et al. [150] built on top of the procedural modeler Houdini.

rule based systems: Another different representation that proved useful for generative modeling are rule-based systems. Such systems provide a declarative description of the construction behavior of a model by a set of rules. An example are L-Systems, as described in the Introduction. Furthermore, the seminal work of George Stiny and James Gips [178] introduced shape grammars, as a formal description of capturing the design of paintings and sculptures, in the sense of "design is calculating". Similar to formal grammars, shape grammars are based on rule replacement.

shape grammars: In its classical definition [178], a shape grammar is the 4-tuple $SG = (V_T, V_M, R, I)$, where V_T a set of shapes, ${V_T}^*$ denotes the set of the shapes of V_T with any scale or rotation. V_M is a finite set of *non-terminal* shapes (markers) such that ${V_T}^* \cap V_M = \varnothing$. R denotes the set of rules, which consists of pairs (u, v), such that $u = (s, m)$ consists of a shape $s \in {V_T}^*$ combined with a marker of $m \in V_M$, and v is a shape consisting of either

- $v = s$
- $v = (s, \tilde{m})$ with $\tilde{m} \in V_M$
- $v = (s \cup \tilde{s}, \tilde{m}$ with $\tilde{s} \in {V_T}^*$ and $\tilde{m} \in V_M$

Elements of the set ${V_T}^*$ that appear in and rules of R are called *terminal shapes*. I is called the *initial shape*, and typically contains an $u \in (u, v) \in R$. The final shape is generated from the shape grammar by starting with the initial shape and applying matching rules from R: for an input shape and a rule (u, v) whose u matches a subset of the input, the resulting shape is another shape that consists of the input shape with the right side of the rule substituted in the matching subset of the input. The matching identifies a geometric transformation (scale, translation, rotation, mirror) such that u matches the subset of the input shape and applies it to the right side of the rule. The *language* defined by a shape grammar SG is the set of shapes that will be generated by SG that do not contain any elements of V_M.

split grammars: The work of Peter Wonka et al. [213] applied the concepts of shape grammars to derive a system for generative modeling of architectural models. This system uses a combination of a spatial grammar system (split grammar) to control the spatial design and a control grammar, which distributes the design ideas spatially (e.g. set different attributes for the first floor of a building). Both of these grammars consist of rules with attributes that steer the derivation process. The grammar consists of two types of rules: *split* and *convert*. *split* is a partition operation which replaces a shape by an arrangement of smaller shapes that fit in the boundary of the original shape. The *convert* rule replaces a shape by a different shape that also fits in the boundary of the original shape. A simple example is shown in Figure 2.5.

This system has further been extended by the work of Pascal Müller et al. [135], which introduced a *component split* to extend the split paradigm to arbitrary 3d meshes, as well as occlusion queries and snap lines to model non-local influences of rules. For example, two wall segments that intersect each other should not produce windows such that the window of one wall coincides with the other wall, therefore occlusion queries are used to decide if a window should be placed or not.

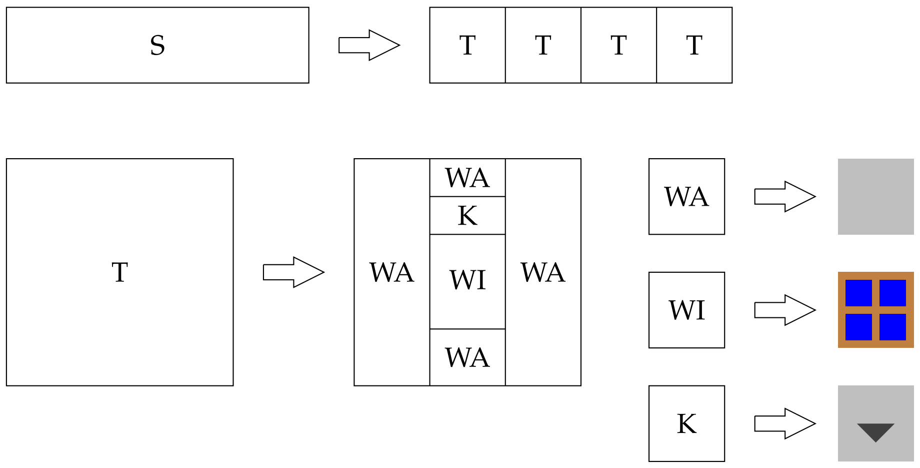
The evaluation of a split grammar, starting from an initial shape, yields a tree structure, which suggests that the evaluation can be speed up by a parallel implementation, which has been shown by Jean-Eudes Marvie et al.[126]. Parallel generation is especially useful in an urban context, with scenes with high complexity and detail. The work of Lars Krecklau et al. [101] used gpu accelerated generation in the context of generating and rendering high detailed façades; the work of Zhengzheng Kuang et al. [112] proposes a memory-efficient procedural representation of urban buildings for real-time visualization.

With more advanced shape grammar systems, non-local influences are a problem because they introduce dependencies between arbitrary nodes of the derivation tree. Recent work by Markus Steinberger et al. [176] shows how to overcome this problem in an GPU implementation. Furthermore, the same authors presented methods to interactively generate and render only the visible part of a procedural scene using procedural occlusion culling and level of detail [177]

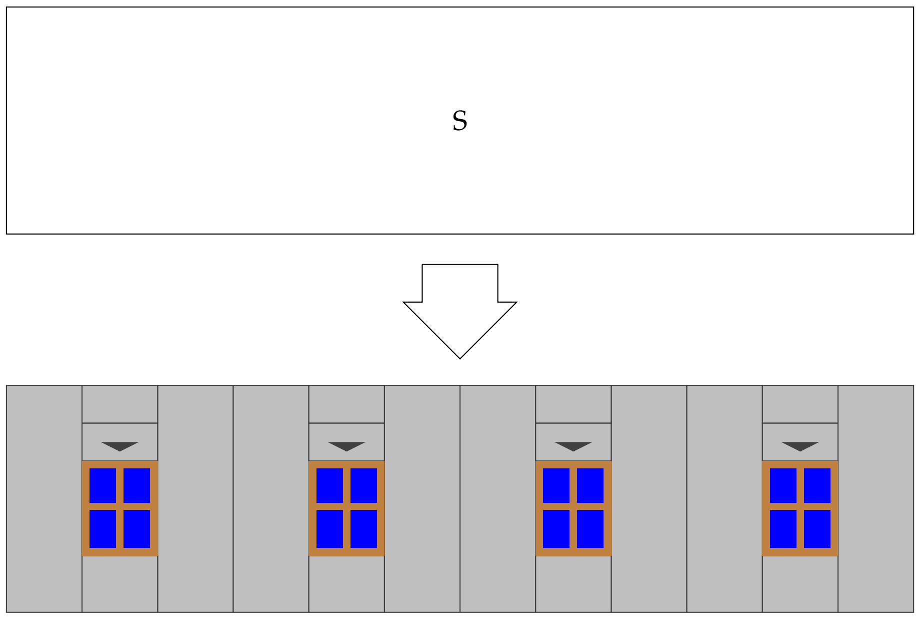
2.3 Shape Representations and Building Blocks

This section is concerned with mathematical representations of shapes. As this thesis mainly concerns with man-made objects such as buildings, I restrict the shape representations to three-dimensional and two-dimensional shapes.

The description will start at a quite low level with reference to unstructured representations, continuing with a short introduction to *topology* and the resulting common surface representations within computers and algorithms. Furthermore,



(a)



(b)

Figure 2.5: A split grammar is suitable for representing repetitive structures, like building windows with decorative elements are generated (right) if the set of rules (left) is applied to the start image. The split grammar derivation process introduced by Wonka et al. [213] is guided by an additional control grammar, that decides e.g. which keystone is selected.

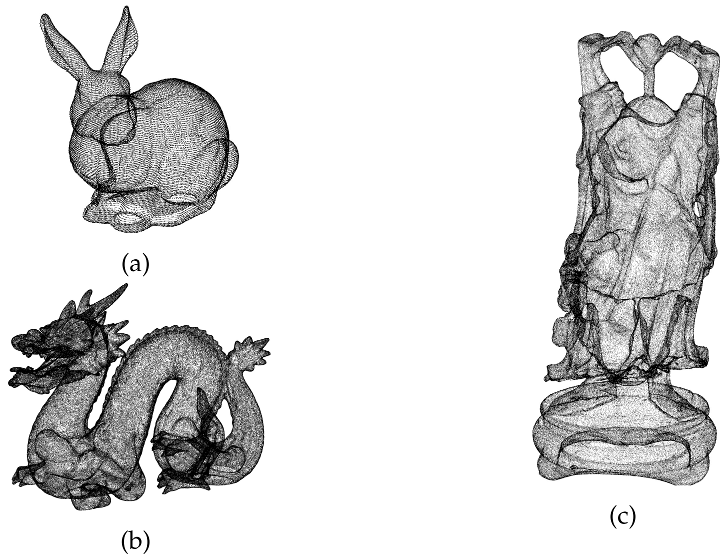


Figure 2.6: One of the simplest geometry representations are point clouds; the surface of an object is described by an unstructured set of points. This representation is of importance because it is the natural representation of several surface measurement techniques, e.g. laser scanning methods. In these examples, the bunny (a) consists of 35.947 points. The dragon (b) is made of 437.645 points, and the happy buddha (c) is composed of 543.652 points. Data provided by the Stanford 3D Scanning Repository. [4]

related work concerning the building blocks for the next chapter, half space modeling, is introduced.

The surface of a shape may be represented by a set of points, a so-called *point set*. A point set is a list of points defined in a coordinate system. Due to the lack of connectivity information between points, such a representation is also called unstructured. While such a representation is not preferred for modeling, or changing a shape, it is a natural representation for scanned objects. Commercial 3D scanning system such as laser scanning systems (called LiDAR - Light Detection And Ranging) survey an object by performing many distance measurements to an objects' surface, which maps directly to a point set. These devices are typically placed on the ground, in which case they are also called Terrestrial Laser Scanning (TLS) devices. Such devices that do not move acquire a point cloud in a spherical area around the measuring device per scan. Indoor situations or larger areas may require multiple scans to cover the whole area of interest. Multiple scans have to be aligned to a common coordinate system before further processing.

Various approaches for rendering point sets are available, see the literature survey of Markus Gross and Hanspeter Pfister for an extensive explanation [64].

[4] `https://graphics.stanford.edu/data/3Dscanrep/`

Basic Topology Facts

The mathematical branch *topology* concerns itself with the study of geometric objects, respectively topological spaces, and their invariant properties under deformations. I give an informal overview on some aspects of topology that are important for this thesis in this subsection, for a formal overview I refer to the books of Erich Ossa [144] and John Lee [116]. Note also that topology, while being the theoretical basis for the representation of shape within computers, is a much wider and abstract field that is applied to many problems outside of shape representation and processing.

The majority of shape representations that are of interest for shape modeling using computers are based on some core topological principles. Topology introduces *cell complexes*, a method to construct topological spaces from elementary building blocks, the *cells* [68]: In general, we start with a discrete set X^0, whose points are regarded as 0-cells. Then, form the n-skeleton X^n from X^{n-1} by attaching n-cells. This means that X^n is the quotient space of the disjoint union of X^{n-1}, with a collection of n-disks. Thus, any X^n is a set of the union of d-disks with d ranging from 0 to n. In computer graphics, the 0-cells are often referred to as *vertices*, the 1-cells as *edges* and the 2-cells are called *faces*.

Surface Representation

The simplest surface element is a triangle: three coordinates define via their barycentric parametrization a linear surface representation. Any surface may be piece-wise linearly approximated by a set of such triangles.

An unstructured set of triangles may also represent a surface, but, similar to unstructured point representations, such a representation is not so suited for modeling. For such applications, a topological representation that describes neighborhood relation of geometric elements is desirable.

Therefore, the common definition of a surface in the context of computer graphics applications is that of an *orientable continuous two-dimensional manifold embedded in* $\mathbb{R}^3$ [24]. A manifold of dimension d is a topological space in which the neighborhood of each point is homeomorphic to the d-dimensional euclidean space. In other words, the intersection of an infinitesimal small sphere with a surface point of such an embedding is homeomorphic to a disk.

In many cases, such surfaces are represented by triangle meshes, which consist of a geometric and a topological component. The latter can be represented by a graph structure, a complex, that consists of a set V of vertices v and a set F of triangular faces f that connects the vertices. Sometimes, the connectivity may also be represented by V and the set of edges E where each edge connects exactly two vertices. The geometric embedding of a triangle mesh into $\mathbb{R}^3$ is specified by associating a 3D coordinate to each vertex v.

An important topological quality of a triangle mesh is if it is a two-manifold or not, as mentioned before. A triangle mesh is not two-manifold if it contains a *non-manifold* edge or a *non-manifold vertex*. If it contains neither, it is two-manifold. Such non-manifold edges and vertices violate the condition that the manifold is locally similar to two-dimensional euclidean space: a non-manifold edge is adjacent to more than two triangles, and a non-manifold vertex can be generated by connecting the tip of two cones, each represented by a fan of triangles. An example of non-manifold configurations is shown in Figure 2.7. To evaluate a general surface, composed of triangles, we utilize the surface evaluation of a triangle: any point p in the interior of a triangle, defined by the points a, b and c, can be written as a barycentric combination:

$$p = \lambda_0 \cdot a + \lambda_1 \cdot b + \lambda_2 \cdot c \tag{2.1}$$

with

$$\lambda_0 + \lambda_1 + \lambda_2 = 1 \tag{2.2}$$

Two-manifolds can describe the surface of non-degenerate three-dimensional solids; non-degenerate means in this context that the solid does not have any infinitely thin parts, in which case the surface properly separates the solid into an *interior* and *exterior* part. Continuous is intuitively understood such that the surface does not have any holes (in which the surface is called a surface with boundaries, and can be transformed into a proper boundary surface by filling the holes).

Orientability can be understood as the possibility to consistently assign an orientation to a manifold surface. In the case of two-manifolds this corresponds to consistently assign surface normals. In other words, on a non-orientable two-manifold, there exists a path that brings a surface normal back to its starting position, pointing in the opposite direction. Non-orientable two-manifolds do not correspond to solids, as there is no interior or exterior surface. A prominent example of a non-orientable two-manifold is the *moebius strip*, see Figure 2.8.

A d-dimensional *simplex* is the generalization of a tetrahedral region of space to d dimensions. With $d = 1$, the simplex is a line segment. With $d = 2$, the simplex is the convex hull of three points, a triangle. In $d = 3$, the simplex is the convex hull of 4 points, a tetrahedron, etc. Barycentric coordinates were first introduced by AUGUST FERDINAND MÖBIUS in 1827, as a means of describing points with respect to the simplex.

Representation using computers

There exist many variations of surface representations, which representation to choose depends on the intended purpose – a geometry processing algorithm may need to utilize the topological structure, for data exchange a simpler description of smaller size might be sufficient.

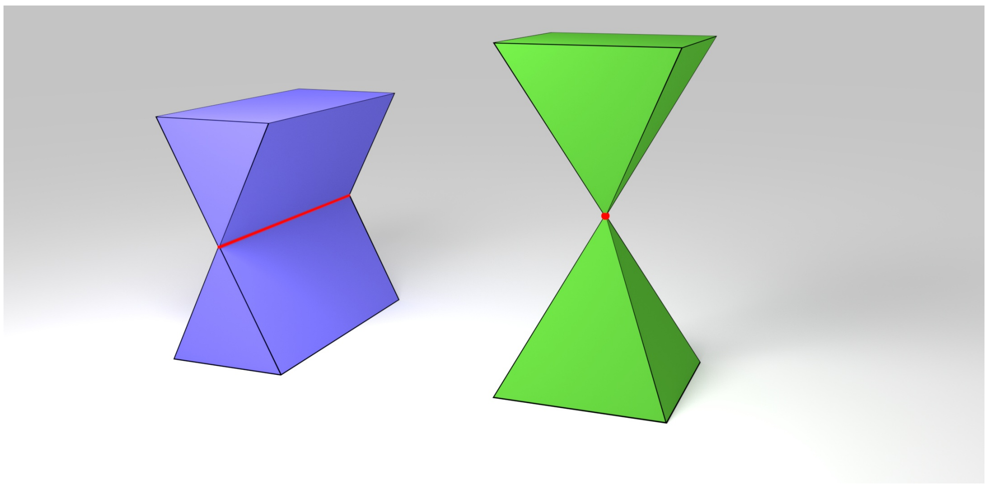


Figure 2.7: Two-manifold meshes are an important representation for geometry processing algorithms. A mesh is not two-manifold if it contains a non-manifold edge, or a non-manifold vertex. The left object shows a non-manifold edge configuration, marked by the red line, and the right object contains a non-manifold vertex, marked by the red sphere.

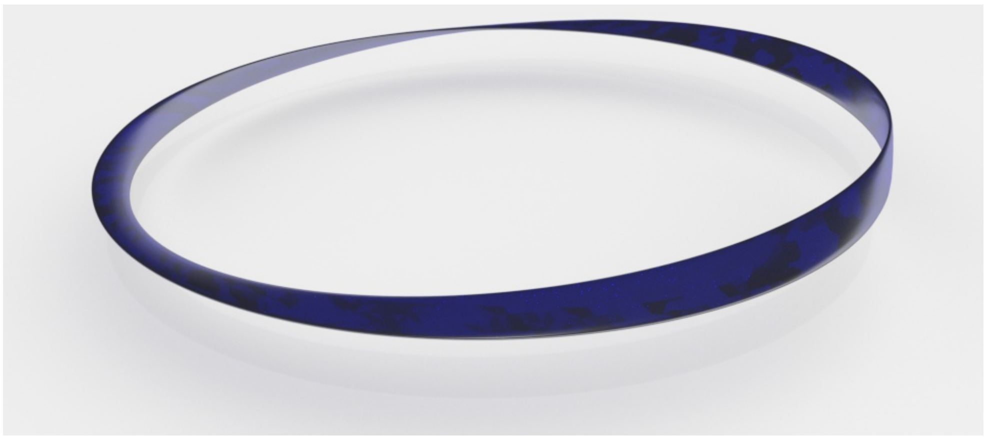


Figure 2.8: The moebius strip is a non-orientable two-manifold - a single surface that cannot be coherently oriented. It can be constructed by taking a rectangular strip of paper, and gluing together the shorter sides, twisting one side by 180° before connecting.

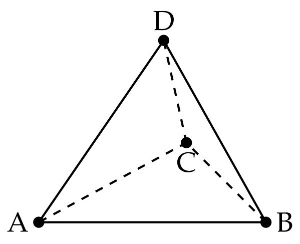
In many cases, the surface of a shape is represented by a set of polygonal faces. Often, two-dimensional simplices – triangles – are used for this representation. For data exchange, these representations are often unstructured lists of faces, in the case of triangles also called *triangle soups*. The de-facto standard data interface between CAD software and machines, such as milling machines or 3D printers, is the *STL* file format which encodes a triangle soup.

While unstructured data is sufficient for manufacturing or rendering purposes, it is often not suitable for further processing, as there is no exploitable topological information, e.g. for traversing the local neighborhood of a triangle.

The *indexed face set* (IFS) is a commonly used representation of surfaces for computer graphics and data exchange. For example, the OpenGL graphics application programming interface (API) defines the concept of vertex and index buffers. A common file format for computer graphics applications is the Wavefront OBJ format, which utilizes an IFS based description. An IFS defines a list of vertices V, where the geometric coordinates for each vertex are specified. Furthermore, it contains a set of faces F; each face is defined by a list of *vertex indices*, where each index points to the position of a vertex in V. Thus, different faces may reference the same vertex by using the same index, which corresponds to neighborhood, or topological information. See also Figure 2.9 for a simple example.

While IFS representations are sufficient for rendering and data exchange, richer topology information is desired for shape modeling and modification operations. An important method for representing exploitable topology information uses an edge based representation, an early version is the winged edge data structure, presented by Bruce G. Baumgart in 1972 [15]; more modern variants are known in slight variations as half-edges [123], [19], directed edges [33], doubly connected edge list (DCEL) [40], or FE-structure [208] which can represent orientable two-manifolds. An example is shown in Figure 2.10. Each edge of manifold is represented by two directional half-edges with opposite orientations. Half edges are linked to adjacent half edges in the face cycle, each face and each vertex are linked to at least one adjacent half-edge. This data structure is also used to represent surface descriptions in the computational geometry algorithms library (CGAL) [97].

These edge based data structures are used to represent two-dimensional manifolds, which also is the representation used in this thesis. Although outside the scope of this thesis, arbitrary dimensional manifolds can be represented with a generalized data structure - the combinatorial map [53] [198]. Non-orientable surfaces can be represented by generalized maps [117].

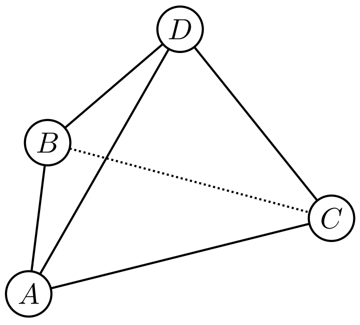


(a) A tetrahedron

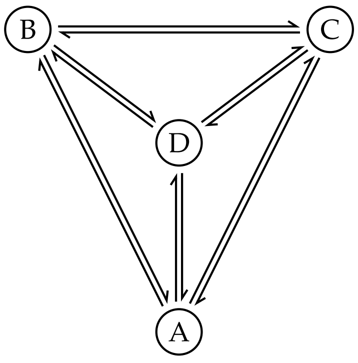
```
# WaveFront OBJ comment
# starts with '#'

# list of vertices

v -1 -1 0      # A (1)
v 1 -1 0       # B (2)
v 0 1 0        # C (3)
v 0 0 1        # D (4)

# triangular face cycles

f 1 3 2        # A - C - B
f 1 2 4        # A - B - D
f 2 3 4        # B - C - D
f 1 4 3        # A - D - C
```

(b) Wavefront OBJ file

Figure 2.9: The Wavefront OBJ file format is a text based file format. This example shows the OBJ representation (right) of a simple tetrahedron (left). The format uses an indexed face set to describe the surface of a shape: First, a list of vertices is defined, corresponding to A, B, C and D. Each face is defined by a sequence of vertex indices, with 1 pointing to the first vertex in the list. Note that comments at the end of lines with vertex or face definitions are here displayed for brevity and may not be supported by a software that reads OBJ.

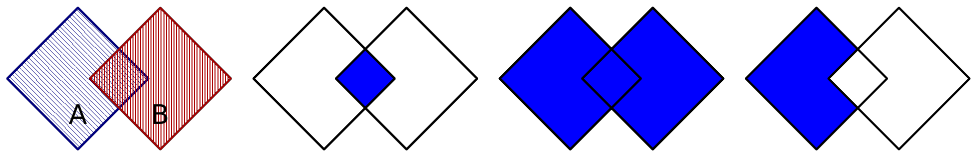


(a)



(b)

Figure 2.10: The surface of a tetrahedron (a) is a closed orientable two-manifold and can be represented by a directed-edge data structure; each edge is represented by two oriented half-edges, each face is composed of an consistently oriented loop of half-edges, as can be seen on the right (b).



(a) Input Shapes (b) Intersection (c) Union (d) Difference

Figure 2.11: Constructive solid geometry describes a shape by combination of shapes using Boolean set operations. This example shows two input shapes A and B (a), and the result of the corresponding Boolean operation as filled area. Common operations are *intersection* $\bigcap$ (b), *union* $\bigcup$ (c) and *difference* $-$ (d).

Solid Modeling - Constructive Solid Geometry - Half Space Modeling

The principles for modeling three-dimensional solid objects are summarized by the term *solid modeling*. A solid can be represented by a closed orientable two-manifold surface. Typically, these principles describe the composition of complex shapes by a combination of simpler shapes. Mathematically, this is represented by associating the shapes to sets, and describing composition using operations of elementary set theory [10], i.e. operations such as *intersection* or *union*, as can be seen in Figure 2.11. These operations are also referred to as *Boolean* operations. The term *Constructive Solid Geometry* denotes the methodology to create complex shapes by a Boolean combination of primitive shapes. This combination is often represented by a binary tree, the *CSG Tree* where the leaves consist of primitive shapes, and each node contains a corresponding Boolean operation. The root of the tree corresponds to the final shape.

The term *half space modeling* denotes a modeling paradigm that allows creating complex shapes by the combination of simpler basic shapes, namely half spaces. A half space is a region of d-dimensional space that lies on either side of a d-dimensional hyperplane. A more thorough definition of the half space modeling methods used by the proposed method are given in the beginning of the next chapter. A basic form of this modeling paradigm dates to Nef polygons [138], which introduced modeling of polygonal structures in 2D using the combination of half spaces with the operations complement and intersection.

An euclidean half space is a convex set, as it satisfies the convexity criterion: Given any two distinct points in the set, their shortest connection corresponds to a straight line segment. A set is considered convex if every point on the connecting line of any point pair is also in the set. It can be trivially shown that this is the case for any point pair on one side of a d-dimensional hyperplane, therefore a half space is also convex. The intersection of convex sets is also convex: For any point pair that is contained in each set, their connection is also contained in each set. Therefore, the connection is also contained in the intersection of all sets. An intersection of half spaces is also called a convex polyhedron.

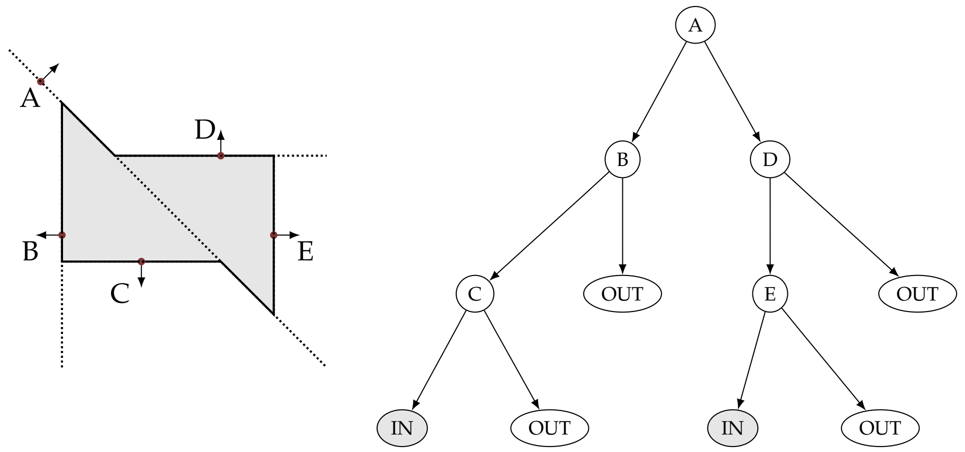


Figure 2.12: A BSP tree is a binary partition of space into convex cells using subsequent hyperplane splits. In this example, a simple two-dimensional shape as seen on the left is described by the BSP tree shown on the right. Each node corresponds to a directed hyperplane (A - E), the left child of a node corresponds to the space below the hyperplane, the right child to the space above. Leaves are labeled ether as IN or OUT, whether they belong to the inside or the outside of the shape.

A related technique is the representation of solid shapes using *binary space partitioning* (BSP) trees. A BSP tree is a binary tree that corresponds to a successive subdivision of space using half spaces. Each node corresponds to a volume, a convex polyhedron, with the root being the whole space. A hyperplane divides this space into two non-overlapping volumes, the intersections with the half spaces on either side of the hyperplane, which correspond to the children of a node. Thus, the leafs correspond to convex polyhedra that are composed by the intersection of all hyperplanes along the path from the leaf to the root. Thus, a BSP tree corresponds to a complete, non-overlapping partition of space into convex cells, its leaves. A shape is now described by a union of such leaf cells, which are labeled as *inside*, all other leaves are labeled as *outside*. See also Figure 2.12 for a two-dimensional example. BSP trees also generalize to any dimension d, where d-dimensional space is partitioned by d-dimensional hyperplanes.

The BSP tree is popular in computer graphics as it is a fast and versatile data structure. It is often used for visibility calculations, e.g. back to front sorting for rendering transparencies, occlusion culling, or visibility culling [5]. But there is a drawback: constructing a BSP tree for a given shape is prone to numerical instabilities, as cutting hyperplanes might also split the shape's geometry - which means intersections have to be calculated. For a more extensive description of the causes of such problems see also Section 2.6 "The Robustness Problem". This is often not an issue because the construction of a BSP tree is a pre-processing step, and encountered problems can be identified and fixed at this stage. Furthermore, building the smallest possible BSP tree is NP-complete [89], so any real world application has to use approximation techniques.

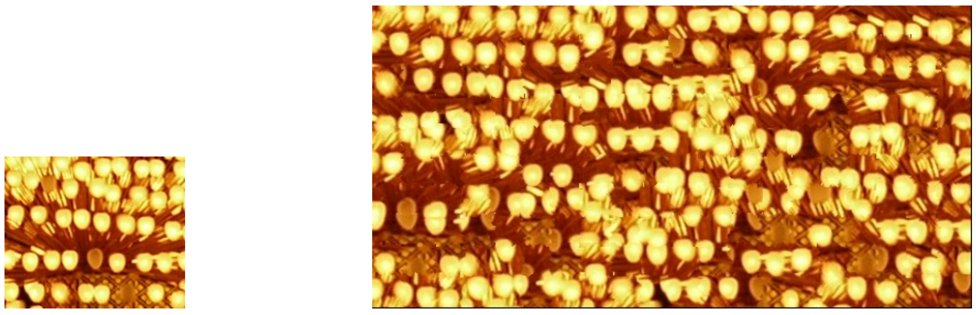
While BSP trees are popular for rendering, the data structure is also used in other contexts: The seminal work of Bruce Naylor et al. [137] presented an algorithm for combining BSP trees using Boolean set operations. In contrast to Booleans on boundary representations (BReps), which require complicated and error-prone modifications of the boundary representation, the BSP method ends up *merging* binary tree structures and intersecting polygons, a much simpler operation. This algorithm has been improved by Mikola Lysenko et al. [121] by replacing the polygon intersection step with linear programming. Furthermore, they presented a compression schema for a reduced memory footprint of BSP trees by reusing identical sub trees. The surface of all IN-leaves of a BSP tree can be recovered from a binary tree that contains only hyperplane definitions [37] - a more robust variant has been proposed in 2012 by Wang et al. [207]. For use cases that require frequent conversion between a BSP tree and a corresponding BRep, mixed representations that contain both representations have been proposed, such as the *BRep-index* by George Vaněček [204], or the topological bsp tree by Comba et al. [37]. While half spaces are mostly represented by planar (linear) hyperplanes, extensions to BSP representations have been proposed that use non-planar (quadric) partitions, and methods to convert between boundary representations and constructive solid geometry (CSG) trees [31]. Additionally, BSP trees are a suitable method for an implicit representation of polygonal objects: signed distance fields, or *BSP-fields* as presented by Fryazinov et al. [59]

2.4 Syntactic Pattern Recognition and Inverse Generative Modeling

In the seminal work *Pattern Recognition and Image Processing* of King-Sun Fu and Azriel Rosenfeld, pattern recognition methods are grouped into two main categories: **decision-theoretic** (or discriminating) approaches and **syntactic** (or structural) approaches. The first category, a set of characteristic measurements, the *features*, are extracted into a feature vector, and recognition is done by building a classification algorithm that associates a given feature to a pattern class – which is a classical machine learning task.

The syntactic approach draws an analogy between the structure of patterns and the syntax of a language. Pattern recognition is carried out by parsing the structure using a given set of syntax rules. In this sense, the approaches I present in this thesis belong to the syntactic pattern recognition family.

As has been explained in the foregoing sections, forward generative modeling describes a shape or object using an algorithm – a set of parameters will yield a



(a) Input Image (b) Synthesized Image

Figure 2.13: Texture synthesis approaches synthesize an image (b) that is locally similar to a given input image (a). The above image was created using the *Image Quilting* algorithm from Alexei A. Efros and William T. Freeman [55].

model instance after evaluation. The inverse problem is now to find the model parameters, given a model or an observation.

When talking about creating mathematical models of physical phenomena, an important property of such problems is the so-called *well-posedness* of a problem, as defined by Jacques Hadamard [67]. A problem is well-posed in the mathematical sense if

- a solution to the problem exists
- the solution is unique for any given input
- the solution depends continuously on data and parameters

The third point entails that a small change in the input results in small changes in the solution, and a large change in the input will lead to a proportionally larger change in the solution. In general, inverse problems are often ill-posed, which makes them hard to solve robustly.

I will now consider several previous works, grouped into similarity of the approach taken, that concern themselves with the inverse problem using generative methods. Generally, I will call a generative Model M, that, when evaluated with parameters $x = (x_0, \cdots, x_n)$, yields a model instance I. An observation or measurement is called O.

Similar Model Synthesis. One family of approaches does not use a generative description language, but will produce a similar object, given an exemplary 3D model. The *Model Synthesis* algorithm [130], as presented by Paul Merell and Dinesh Manocha, is inspired by well-known Texture Synthesis approaches. Texture Synthesis is a method to create larger 2D images from a smaller sample image, often used for texture mapping in computer graphics, see also Figure 2.13 for a texture synthesis example using the Image Quilting algorithm [55]. Merell describes in his thesis [129] discrete model synthesis, where a model is composed of transformed base model pieces, the degrees of freedom are reduced using several types of constraints. He further describes continuous model synthesis, where an example model does not have to be decomposed into pieces: the face planes of the input model are shifted parallel in discrete steps, the arrangement of all such planes in space forms a discrete structure from which a locally similar shape can be derived. Again, the search space is reduced using several types of constraints: dimensional constraints, which e.g. fix the width of an object, algebraic constraints, which e.g. fix the aspect ratio of an object, incidence constraints, which maintain neighborhood structure and large scale constraints, which specify the rough structure of the generated object.

Grammar Inference from Symmetry. A related, more general approach was presented by Martin Bokeloh et al. in the paper *A Connection between Partial Symmetry and Inverse Procedural Modeling*. In this work, they describe the identification of symmetric boundaries, that separate the dissimilar parts of a model. These boundaries are called *docking sites*, and the parts are called *dockers*. The docking sites are automatically identified in an input model via symmetry detection. Symmetric parts (e.g. one part that is symmetric under translation) can be used to construct docking sites by cutting through them and recombine the model together using cut lines from different symmetric parts. As not all cuts through symmetric parts are valid docking sites, an algorithm is presented to extract valid docking sites from an input model. From the hierarchical dependencies of extracted valid docking sites, a context free grammar is extracted. Random productions from this grammar will yield models that are locally similar to the input model. A variant to analyze 2D vector images was presented by Ondrej Stava et al. [175]: *Inverse Procedural Modeling by automatic generation of L-Systems* automatically generates a generative representation from a given image. The method analyzes instances of terminal symbols and clusters similarities in placement to create non-terminal symbols and L-System rules.

The second family will find a "best" approximation to a given high-level description, while using a generative description to constrain the search or result space. These problems are generally hard to solve, as the search space is too vast to explore exhaustively. Torsten Ullrich presented in his thesis *Reconstructive Geometry* [199] a method to semantically recognize 3D objects. This solution utilizes a generative description to describe an object class and its variability. A distance based minimization procedure is utilized to find the best model parameters P. The optimization utilizes a compiler tool-chain to analyze the generative description, the *Euclides* framework. The description itself is a scripting language similar to JavaScript. The observation of the input shape is given as a point cloud O, the algorithm answers the questions whether O can be described by the generative Model M and if so, what are the input parameters x such that $M(x)$ is a good description of O. The latter is found using a distance based, weighted error function

$$f(x) = d_\psi(O, M(x)) \tag{2.3}$$

which should be minimal. Summarizing, the result is constrained to the family of shapes described by the generative description. The similarity is measured using a distance metric based on point distributions in space, the solution is a local optimum, i.e. it depends on the initialization values.

Stochastic Approaches. Due to the vastness of the search space, stochastic methods are often used. The family of Markov Chain Monte Carlo (MCMC) methods samples from an unknown probability distribution by constructing a Markov Chain with the desired distribution as equilibrium, each step in the chain depends on the foregoing step only. In the context of recognition of façade structure from images, several works have been concerned using variants of syntactic pattern recognition. Nora Ripperda et al. presented an approach [162] to reconstruct a façade structure to a given grammar using a reversible jump Markov Chain Monte Carlo (rjMCMC) process to explore the solution space. Similarly, Olivier Teboul et al. use a random walk MCMC to derive a parse tree from a 2D split grammar to obtain a semantic labeling of an ortho photo of a façade. A more general variant has been presented by Jerry O. Talton et al. - in their work *Metropolis Procedural Modeling*, they present a method to automatically create a derivation, given a grammar and a high-level specification of the desired production [183]. The approach is also based on MCMC inference using the Metropolis Hastings algorithm, and the authors demonstrate its application on generative models of trees, cities buildings and Mondrian paintings. Similarly, Ondrej Stava et al. [174] presented a method to fully automatic find the parameters of a general procedural tree model. The general model covers several species and is based on recent advances in plant biology and computer graphics. It grows branches from active so-called *buds* per growth cycle. The model contains 24 parameters in total and the authors define a novel similarity measure to determine the parameters to a given representation that has been obtained from point clouds; From this representation they are able to create similar instances of the same plant or tree family, or replace all plants of a family with another. The optimization procedure also utilizes the Metropolis-Hastings algorithm to obtain the parameters of the generative representation.

Similarities to Parsing. As many of these syntactic pattern recognition approaches model structural relations with formal grammars, parsing methods from compiler construction may also be applied to derive a derivation tree from an observation. The main problem with such methods is that classical parsing methods are used to analyze text, and not observations such as images, that may contain uncertainties or measurement errors. One such algorithm, the well-known Cocke-Younger-Kasami (CYK) algorithm is a dynamic programming algorithm that iterates over all substrings of the input and all non-terminals of the grammar. Michail I. Schlesinger et al. [167] adapted this algorithm for two-dimensional split grammars to parse the structure of musical note sheets. Such classical parsing algorithms fall into the category of discrete optimization, many problems can be modeled using graph theoretic approaches.

Graph theory and discrete optimization. A graph consists of a set of elements (nodes) together with a binary relation that is defined on the set [211]. Graphs can be visually represented by diagrams in which the elements are shown as points and binary relations as lines (edges) joining pairs of points. Discrete or combinatorial optimization [99] concerns itself with finding optimal solutions in discrete problem domains. Such problems often exhibit a seemingly simple structure, but finding the optimal solution leads to an exhaustive search of the solution space, which is often not feasible (e.g. the famous traveling salesman problem). Therefore, approximative algorithms are used in practice that may not find the global optimum but terminate in an acceptable amount of time.

Further Literature. For the interested reader I refer to the excellent course on inverse procedural modeling for virtual worlds that was given in SIGGRAPH 2016 by Daniel Aliaga et al. [7] for additional information.

2.5 Building Information Modeling

Most of the generative approaches are used in a presentation or visualization context, e.g. in virtual worlds. Nevertheless, the demand of the application of such methods in a real world context, e.g. in the domains of Architecture, Engineering and Construction (AEC) or Facility Management (FM), is increasing. Therefore, generative modeling can also be helpful in the context of building information modeling (BIM), the new paradigm of today's building industry [2]. The American National Building Information Model Standard (NBIMS-US) project committee defines BIM as "a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life cycle; defined as existing from the earliest conception to demolition" [136]. Other definitions are summarized in a literature review by Abbasnejad and Moud, who conclude that a generally accepted comprehensive definition of BIM has not been established yet, and different stakeholders (architects, builders, owners, *etc.*) have mixed expectations towards BIM [1]. In contrast to established computer-aided design (CAD), a building information model does not just store the geometry of a building, but includes semantic data about the functions of the buildings and its elements. Furthermore, BIM is intended to be used throughout the building's life cycle, containing information for planning, design, construction, operation and maintenance. That is, a model is not only used by architects, contractors and suppliers, but by all kinds other users, e.g., government agencies, owners, real estate agents, facility managers, *etc*.

Eastman *et al.* help to understand BIM by describing examples that are not BIM technology. As already mentioned, models without object attributes, but only 3D data, are not considered BIM. Furthermore, models composed of multiple 2D drawings that have to be combined or models that do not automatically reflect changes made in one view in other views are not building information models. Moreover, Eastman *et al.* consider parametric object capabilities as essential for BIM. Parametric objects in BIM can include rules to automatically modify associated objects (e.g., a wall is changed when a door is placed in it) and for ensuring feasibility (e.g., regarding size and manufacturability) [48]. Such intelligent objects are similar to the idea of generative modeling.

One use-case is documenting a building "as-built BIM" [77], to aid, amongst others, in the application scenarios of restoration, documentation and maintenance. Such a model is built from measured data, which is typically acquired by terrestrial laser scanning (TLS) or image-based approaches (photogrammetry or structure from motion techniques), which yields point positions in 3D. From these point clouds, a mesh can be created using 3D surface reconstruction techniques, e.g., Poisson surface reconstruction [95]. Furthermore, the surface appearance has to be acquired [210]. Such semantic relationships have to be acquired and represented within the model [184]; see also the foregoing section about semantic enrichment. A recent example of the usage of parametric elements for the reconstruction and documentation of complex architecture is the case of a reactor building, as shown by Jean-François Hullo *et al.* [84].

For historic building information modeling, or HBIM, procedural methods have been used to aid the reconstruction and documentation process. In the work of Conor Dore *et al.*, a shape grammar approach was applied to model classical building facades for historical BIM [46] and reconstructed the Four Courts, a historic classical building in Dublin City [47] using rule-based modeling in ArchiCAD. Another recent example of creating a HBIM model with rich semantics from terrestrial laser scanning data has been shown by Ramona Quattrini *et al.* in the case study of the Church of Santa Maria at Portonovo [156] using Autodesk Revit.

In the context of functional building information modeling (FBIM), generative techniques can be used to semantically filter a CAD data set of a building. A major future challenge in the building industry is to reduce primary energy use of buildings. Hence, energy performance simulation becomes an increasingly important topic. Accurate, yet efficient simulation depends on simple building models. Most of the required data can be found in BIMs. However, typical BIM data contains a lot of irrelevant data, in particular geometric representations, which are too detailed for energy performance simulation. Using generative modeling techniques, Daniel Ladenhauf *et al.* [113, 114] show an approach of geometry simplification subject to semantic and functional groups. These simplified models are sufficiently accurate for energy calculations and small enough so that they do not flood simulation software with unnecessary details. As these semantically-filtered models are generated automatically, they simplify the design process significantly and offer an energy calculation, even at early design stages.

2.6 The Robustness Problem

This section is concerned with some obstacles that arise when using generative shapes in the real world, i.e. problems that occur within implementations of geometric algorithms.

In general, implementations of geometric algorithms are often susceptible to being *non-robust*, due to numerical errors that arise when working with limited-precision floating-point representation – which is the default data type for numerical operations in most programming languages. As has been said by Yap [214]:

> "Non-robustness refers to qualitative or catastrophic failures in geometric algorithms arising from numerical errors."

Why do geometric algorithms seem to be especially error-prone? There is often a gap between the theoretical design of a geometric algorithm and its practical implementation: geometric algorithms are usually designed and proven using real numbers, while fixed-size floating-point numbers, *floats*, are used for implementation. The IEEE Standard for Floating-Point Arithmetic (IEEE 754 [90]) is one of the most widely used technical standards, available in hardware floating-point units; it defines e.g. binary and decimal arithmetic formats, interchange formats, special values, rounding rules etc. In general, a floating-point number in this format is represented by a sign bit s, and the integer mantissa m, or significant, and an integer exponent e with respect to a base b:

$$(-1)^s \cdot m \cdot b^e \tag{2.4}$$

For example, the IEEE 754 *double precision* format defines a mantissa size of 52 bits, and an exponent size of 11 bits, which allows it to represent an absolute numeric range between $\approx 5 \cdot 10^{-324}$ and $\approx 1.798 \cdot 10^{308}$. The intermediate result of any numerical operation using such numbers will be rounded to this representation, introducing small errors. Furthermore, the representable values are not evenly spaced on the real number line; the gap between two representable values increases with the distance from the origin.

The rounding error introduced by using floats is negligible in many applications. However, when used in geometric algorithms, the consequences can be quite severe. Typically, the observed behavior is undesired or strange output for some input configurations, in some cases the program may even crash. In many cases, this leads to a tedious cycle of problem identification and local repairing, typically by introducing small tolerances, also often referred to as epsilon values. While this approach reduces problems in most cases, it is commonly agreed on that this approach does not lead to a completely robust algorithm [80]. The problems can also be hard to track down, as these errors typically arise after applying specific operations in a specific order, which can be hard to reproduce.

To find out why and where these algorithms fail we take a look at the representation of shape in a program, and the way of applying modification to this representation: Typically, the data representing a shape is stored using some sort of topological information, as well as geometric information. The topological information is discrete, which naturally maps to an exact digital representation. The positional information, however, often corresponds to real values and is represented using floating-point. Depending on the application, the represented shape may be assumed to comply with specific properties. An algorithm that modifies such a representation might violate these properties inadvertently as a side effect, due to rounding errors in the positional information.

As an example, imagine a line pair of intersecting lines which are almost identical. Rounding the intersection point to a representable value may introduce a deviation from the real intersection, in the case of almost identical lines the deviation may even get relatively large, as a small deviation of the input yields a large deviation in the output.

So, what constitutes a *robust* algorithm? The answer mostly depends on the needs of the person being asked this question, but in most cases a robustly implemented algorithm means it will not fail (in the sense of a program error), regardless of the input given to the program. A detailed answer depends on the use case; In some cases, exact evaluation might be necessary, in other cases it is acceptable to tolerate rounding errors in favor of evaluation speed.

The following paradigms on how to treat this situation can be identified:

The standard paradigm In many cases the fact that the algorithm is not robust is simply overlooked or ignored. Tolerances are used to handle uncertainties, but it is commonly agreed that this method will not be robust [79], and one will most likely end up in the aforementioned cycle of problem track-down and local repair.

The exact paradigm Conceptually, the simplest solution is to remove the discrepancy between the theoretical algorithm and its implementation. This is achieved by changing the representation of numerical values in the system from floats to a suitable representation of arbitrary precision. Therefore, no rounding error is made; the implementation directly reflects the theoretical algorithm description. However, the simplicity at implementation level comes with a performance cost: arbitrary sized numbers are typically slower by several orders of magnitude, and as numerical values grow in precision, the memory cost can be high as well.

Depending on the nature of the problem, the result of a geometric algorithm is sometimes bounded in precision. For example, Boolean operations of solids bounded by planar regions: the result of such an operation on two polyhedra will either contain boundary elements of the input polyhedra and optionally new elements introduced by the intersection of the two objects. The input polyhedra can uniquely be described just by their topological information and the plane equations of their faces. Vertex positions correspond to intersections of adjacent planes. Therefore, there is no growth of precision as newly created intersection vertices also correspond to intersections of planes of the two input objects. Thus, any intermediate calculations in the algorithm (the predicates) are also bounded in precision, as they depend on the input planes (and vertices expressed through intersection of such planes).

These predicates typically give the answer to a question with a few discrete answers (e.g. *inside* or *outside*), such an answer can be obtained using interval arithmetic [78] to decide if the calculation can be carried out efficiently using an imprecise data type, or if higher precision is needed to obtain the correct result. Thus, the calculation is carried out using normal floats, and a rounding error interval (dependent on the actual arithmetic tree) is propagated with the calculation. After evaluation, the answer may be given directly if the interval suggests that the answer is unambiguous. If not, a higher precision (and slower) numerical representation is used to obtain the true answer. Various types of such predicates exist; they however all guarantee the correctness of their result. Their implementation can be quite involved, therefore research has been carried out into systems that automatically generate filter code, given a predicate algorithm [128].

Another recent approach proposes a new floating-point based format called *UNUM* [66] that utilizes one bit of the representation to distinguish between exact, i.e. a floating-point value exactly represents a real value, and inexact, i.e. the value represents the interval between two consecutive representable floating-point values. Therefore, no rounding will occur if calculations are carried out exactly - which may require interval arithmetic techniques.

The rounding paradigm Another method to tackle this problem is to calculate one step of the algorithm precisely, and rounding the result back to a representation using a limited precision, while maintaining a topology as close as possible to the exact result. The predicates therefore operate on bounded precision input, and therefore require only bounded precision as well. Their evaluation however still has to be exact.

An example is the method of *snap rounding* an arrangement of line segments in low dimensions [62]. The input consists of a set of lines with integral start and end coordinates. The algorithm calculates an arrangement with intersection vertices at integral positions. Different variations of this rounding techniques exist, each with different properties on the number of newly introduced segments or bounds of the distance of intersection vertices to original line segments.

Unfortunately, these algorithms are available for a limited number of geometric algorithms only, and some are too involved (regarding implementation or run-time cost) to be considered for practical implementation.

The symbolical reasoning / topological consistency paradigm It was observed that geometric algorithms can also be made robust (in they sense that they always produce a result that conforms to given specifications, regardless of the input) if the topological structure is always held consistent with respect to the aforementioned requirements to a geometric algorithm. Let's look again at a clipping problem, where this approach was proposed by SUGIHARA [180]. When clipping a convex polyhedron with a plane, the vertices of the polyhedron classify into the two groups inside and outside the clipping region. When we look at the neighborhood graph of the polyhedron, due to convexity, each vertex group is one connected subset, and intersection vertices are placed on edges between the two groups. An algorithm can now be made robust if the topological consistency is enforced before updating the topological structure: In this example, it corresponds to re-classify vertices until the aforementioned requirement holds. The algorithm will always produce some output and terminate; however the resulting geometric information may not correspond to the desired output. Consider an example where the convexity requirement is violated, due to round-off errors. The classification now gives two connected regions that classify as outside, and one region that classifies as inside. Enforcing topological consistency, we relabel one outside region to inside; the topological structure now corresponds to the requirement. The relabeled region is however ignored for clipping and may introduce a geometric error (e.g. a bulk that protrudes into the clipping region). Cleverly designed algorithms however run nearly as fast as when ignoring the problem while introducing only small deviations from an exact solution. State-of-the-art methods perform e.g. a direct repairing after a non-robust operation to guarantee the robustness [11].

Micro Fragments. Nearly all these methods share a common problem: as the evaluation is carried out exact, small errors - that stem from any problems in the input - are represented exactly as well. For example, using a plane-based representation to intersect planes that form a cone, their intersection will almost never meet in a common point. This may lead to geometrically small fragments (*micro-fragments*) in the topological representation, which decrease performance. A geometric rounding strategy or a post processing step might be able to remove some micro-fragments, but it is not clear if rounding is always possible without compromising geometric and topological constraints, e.g. that all vertices of a face lie exactly in one plane.

2.7 Summary

In this Chapter, i first gave an extensive overview of the generative modeling paradigm (Section 2.1) and discuss several existing methodologies to approach generative modeling (Section 2.2), in particular various languages and shape grammars. Then, I described the basic building blocks for generative modeling: the mathematical description of shapes and their surfaces (Section 2.3) and some modification paradigms. Furthermore, I gave an introduction to established methods for inverse generative modeling in Section 2.4, which builds the foundation for the inverse approaches described in Chapter 4. Specifically, Section Section 2.5 discusses the application scenario for the inverse approach presented in Section 4.2.

Finally, I discuss the most important robustness problems that arise frequently within generative forward modeling Section 2.6. The next chapter is concerned with my proposed generative forward modeling system based on convex polyhedra, and its robust evaluation.

Tool Name	Application Domain	Programming Category	Environment
Blender Scripting	general purpose modeling	python scripting	open source modeling software *blender*
CGAL - The Computational Geometry Algorithms Library[192]	general purpose modeling	C++	CGAL open source project
CityEngine [135]	urban modeling	CGA shape	commercial integrated development environment *CityEngine*
Generalized Grammar G^2 [103]	scientific	python scripting	commercial modeling software Houdini
Generative Modeling Language (GML) [69]	CAD	postscript dialect	proprietary, integrated development environment for polygonal and subdivision modeling
Grasshopper 3D	visual arts, rapid prototyping, architecture	visual programming based on dataflow graphs, Microsoft .NET family of languages	commercial modeling software *Rhinoceros3D*
HyperFun [149]	scientific	specialized high-level programming language	proprietary geometry kernel FRep (Function Representation)
Maya Scripting	general purpose modeling	Maya Embedded Language (MEL) and python scripting	commercial modeling software *Autodesk Maya*
OpenSCAD	CAD	OpenSCAD language	open source, based on CGAL geometry kernel

Table 2.1: Overview on generative 3D modeling tools and approaches (part 1).

Tool Name	Application Domain	Programming Category	Environment
PLaSM	scientific	python scripting, Function Level scripting	integrated development environment *Xplode*
Processing	visual arts, rapid prototyping	Java dialect	open source, integrated development environment *Processing*
PythonOCC	general purpose modeling and CAD	python scripting	Open CASCADE Technology
Revit Scripting	architecture	Microsoft .NET family of languages	commercial modeling software *Autodesk Revit*
siteplan [96]	rapid prototyping, architecture	interactive GUI-based modeler	open source, integrated development environment *siteplan*
Sketchup Scripting	architecture, urban modeling and CAD	Ruby scripting	commercial modeling software *SketchUp*
Skyline Engine [150]	urban modeling	visual programming based on dataflow graphs, python scripting	commercial modeling software Houdini
speedtree	plants/trees	interactive GUI-based modeler, SDK for C++	standalone modeler and integration into various game engines
Terragen	landscape modeling	interactive GUI-based modeler	free and commercial, integrated development environment *Terragen*
XFrog [41]	plants/trees	interactive GUI-based modeler	integrated development environment, standalone and plugins for *Maya* and *Cinema4D*

Table 2.2: Overview on generative 3D modeling tools and approaches (part 2)

3 Robust Generative Shape Composition using Convex Polyhedra

In this chapter, I give a short review of the characteristics of rule-based procedural modeling systems and give some design rationale, followed by the description of the proposed system for a shape grammar on convex polyhedra. Parts of this system have been published in *"Shape Grammars on Convex Polyhedra"* [190] and *"Fast and Exact Plane-based Representation for Polygonal Meshes"* [111]. First, I start by describing the basic structure and a low level interface for shape construction via intersection of half spaces, then I discuss a robust implementation of the surface evaluation. Afterwards, I review some higher level operations for modeling of architecture and showcase a standalone application that allows direct interactive modeling of generative façades.

3.1 Basic Structure

As described in Section 2.3, there exist various shape representations and building blocks for shape creation and modification. A generative modeling system utilizes generating functions to create such a shape representation. Naturally, these generating functions are tightly coupled to the chosen representation of geometric entities and their adjacency relations. A desired property is that rules can be generalized, i.e. applied to a wider range of situations. For example, if a rule can not only be applied to a specific shape (e.g. a box), but on families of shapes (e.g. convex polyhedra).

The goal of the proposed rule based modeling system is to obtain a concise and generalized generative description of a desired model (or model class). The system reaches these goals by describing the model using a combination of hierarchical refinement and growing metaphors: *Refinement* is carried out by partitioning shapes into smaller shapes (e.g. a split) as can be seen in Figure 3.1. *Growing* (see Figure 3.2) is specified by constructing new shapes using other shapes (e.g. an extrusion). In both cases, a rule specifies the creation of new shape from an existing shape. This shape is called the *parent*, and the resulting, or created shapes, the *children*.

To facilitate generalization, many rules in the proposed system do not use named entities or indices to specify important aspects in an input shape (e.g. a face

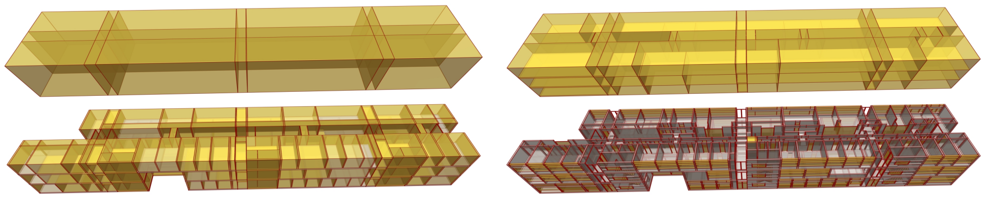


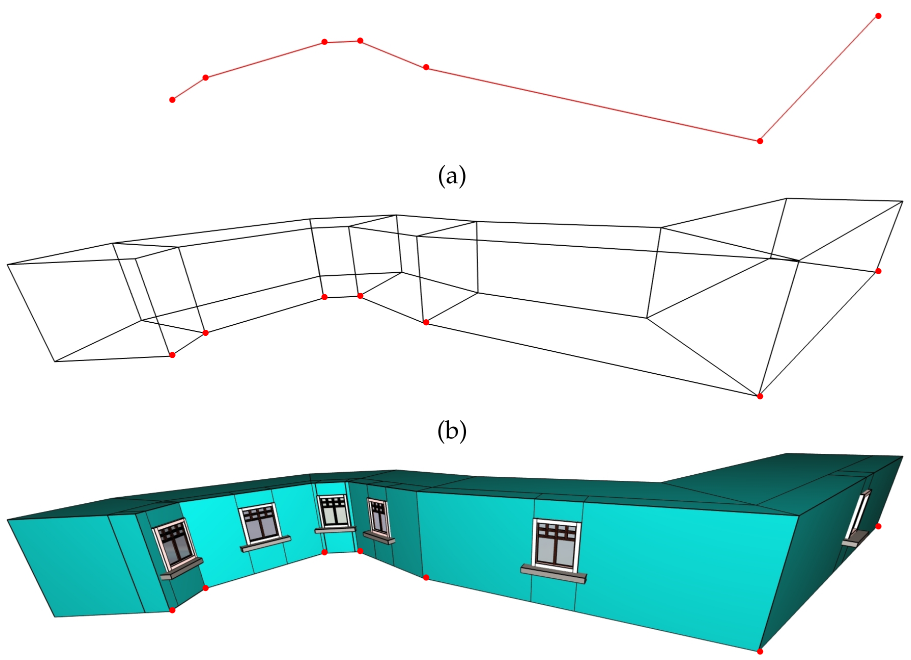
Figure 3.1: A coarse to fine modeling approach by consecutive subdivision is well suited for expression in rule based modeling, as the hierarchical refinement steps are modeled by respective rules, e.g. the refinement of a building by splitting a box into smaller parts to create the structure of a building [81].

for extrusion), but rather use search directions with respect to a local coordinate system. Furthermore, attributes can be attached to the shapes to influence the behavior of rules that will be applied in a later stage of the evaluation hierarchy.

Notation

The descriptions in this chapter use the following notation symbols:

- $\lfloor x \rceil$ denotes rounding a value x to its nearest integer
- $\lfloor x \rfloor$ is the largest integer $\not> x$
- $\lceil x \rceil$ is the smallest integer $\not< x$
- $x \in \mathbb{R}$ denote scalar values along an axis in the standard coordinate system
- $\hat{x} \in \mathbb{R}$ denote scalar values along an axis in the plane grid coordinate system (see Section 3.4)
- integer values are denoted as $x^* \in \mathbb{Z}_0$
- the function $B(x*) \rightarrow \mathbb{N}^+$ denotes the number of bits that is needed to store the value of an integer x^* (the result of a computation using finite length integer values).
- $c = \langle \vec{a}, \vec{b} \rangle$ of two $\mathcal{D}$-dimensional vectors specifies the dot product $c = \sum_{i=1}^{\mathcal{D}} a_i \cdot b_i$


(a)

(b)

(c)

Figure 3.2: The *grow* (or construction) paradigm constructs new shapes from existing shapes, e.g. starting from a polygon on the ground (a), a prism is constructed for each line segment, with bisectors between the segments (b). The resulting shapes are subdivided and some windows are placed. Note that the window sill is also modeled using a growing paradigm: extrusion.

3.2 Shapes: Half Spaces and Convex Polyhedra

Efficient representation of arbitrary shaped surfaces is still a challenge for generative systems, or as Mueller et al. denoted it, the *Problem of complex surfaces* [135]: The outcome of a rule should accordingly reflect the input shape of the rule. Arbitrary shapes might be composed of complex surfaces with complicated topology, where no general method of simple rule production is known.

Many procedural modeling systems deal with this problem by restricting its shape representation to a discrete set of basic shapes; each basic shape has to be handled individually in each rule. The solution of Müller et al. [135] is to instantiate new shapes from surface elements (vertices, edges and faces) of a shape with a complex surface, and allow rule adaption according to overlap tests (e.g. to not create a window in the region of two overlapping wall segments). The problem with this approach is that it yields many overlapping shapes, which is sufficient for visualizing purposes, but it puts severe limits to its usage if the system should

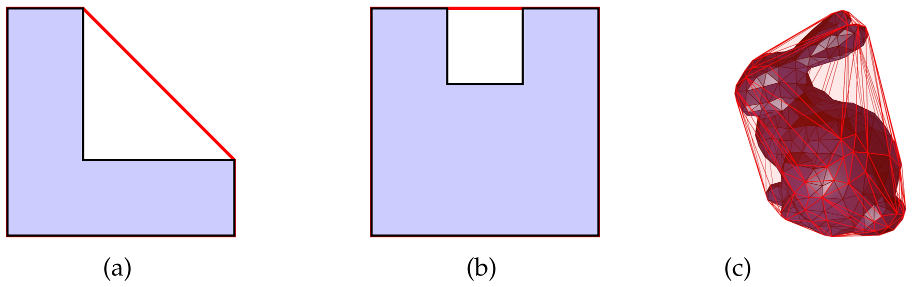


Figure 3.3: The convex hull of a shape is the smallest enclosing convex set that fully contains the shape. Some examples of shapes and their convex hulls (red) are given in two dimensions (a), (b) and three dimensions (c).

perform geometric reasoning about the result of such overlapping shapes, e.g. a ledge that runs along a façade that is composed of several segments from different rules.

In order to overcome the limitations of this surface-oriented approach, I define both the basic shapes and the basic partition rules to be *volume-oriented*, using one consistent representation. The reasoning about shapes in common split grammar systems is also volumetric: shape query operations are mostly carried out on the bounding box of a shape, along principal axis directions. In order to achieve versatility but still maintain enough flexibility to achieve general rules that adapt to a wide range of shapes, the rules of the proposed system query the convex hull of a shape (see Figure 3.3), instead the bounding box. Therefore, rules can be defined to operate on a wider range of shapes, in contrast to have a specific rule for each basic shape, as long as the shape is convex. Non-convex shapes have to be partitioned in convex parts.

The most elementary shape entity in the proposed system is a volumetric convex shape. This allows creating rules that operate on shape families without having to address the problem of complex surfaces. Therefore, the basic shape entity is a *convex polyhedron*. Shapes that cannot be expressed by a single convex polyhedron will be composed by a union of several polyhedra. I review the basic properties of half spaces and convex polyhedra necessary for the constructions in this section, for an extensive background on the theory of convex polyhedra in arbitrary dimensions I refer to the seminal work of Grünbaum et al. [65] and Aleksandrov [6].

The low-level construction entity is a planar half space; convex polyhedra will be composed of such half spaces. A half space can be described by an oriented hyperplane, see Figure 3.4 for an example of a two-dimensional half space.

Definition 1. *A half space in $\mathcal{D}$ dimensions is the set of points $p \in \mathbb{R}^{\mathcal{D}}$ that satisfy the equation*

$$\langle \vec{n}, p \rangle + d < 0 \tag{3.1}$$

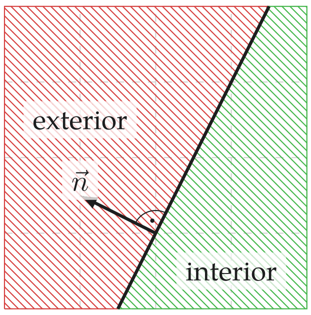


Figure 3.4: An oriented $\mathcal{D}$-dimensional hyperplane partitions $\mathbb{R}^{\mathcal{D}}$ in two regions (half-spaces). The outward pointing normal $\vec{n}$ specifies the orientation. This image shows an example in 2D.

the tuple $(\vec{n}, d)$ describes an oriented hyperplane in $\mathbb{R}^{\mathcal{D}}$, and $\|\vec{n}\| > 0$.

The $\mathcal{D}$-vector $\vec{n}$ corresponds to the normal direction of the hyperplane, and if $\|\vec{n}\| = 1$, the scalar d corresponds to the signed distance from the origin. (in which case the representation is called *hessian normal form*, or HNF).

Given such an oriented hyperplane $(\vec{n}, d)$, we can now classify any point $p \in \mathbb{R}^{\mathcal{D}}$ with respect to this plane in the following manner:

$$\mathcal{H}(\vec{n}, d) = sgn(\langle \vec{n}, p \rangle + d) = \begin{cases} -1 & \rightarrow \ominus \text{ interior} \\ 0 & \rightarrow \odot \text{ on} \\ +1 & \rightarrow \oplus \text{ exterior} \end{cases} \tag{3.2}$$

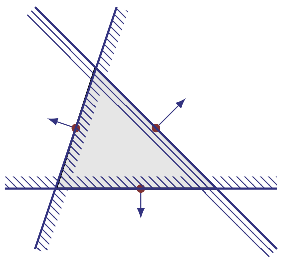
Such a function, as $\mathcal{H}$, that yields a (discrete) decision value with respect to a geometric configuration, is called a *geometric predicate*. When modeling with half spaces, it is often useful to swap interior and exterior half space, (*flipping*). This is achieved by inverting the inequality in Equation (3.1), and can easily be implemented by reversing the sign of the tuple: $(\vec{n}, d) \rightarrow (-\vec{n}, -d)$.

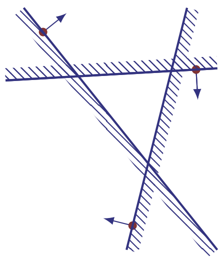
Definition 2. *A convex polyhedron P is defined by the intersection of k interior half spaces $H = \{(\vec{n_1}, d_1), (\vec{n_2}, d_2), ..., (\vec{n_k}, d_k)\}$, with $k \in \mathbb{N}_0$. The interior of P consists of all points p that satisfy*

$$\forall p \in P : \forall i \in \{1..n\} : \langle \vec{n_i}, p \rangle + d_i < 0 \tag{3.3}$$

*If $P = \varnothing$ the polyhedron is called **empty**. A polyhedron containing an empty set of half spaces (k=0) spans the whole space: $P = \mathbb{R}^{\mathcal{D}}$.*

The half spaces in H are also called the *defining half spaces* of the polyhedron. So far, we have defined the interior of a convex polyhedron. Another important concept is the *boundary* of a convex polyhedron:


(a) standard bounded


(b) empty

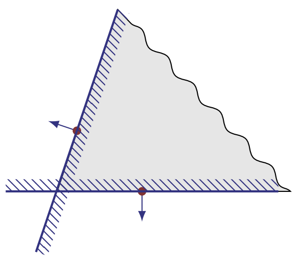

(c) unbounded

Figure 3.5: Several configurations may arise depending on the geometric configurations of the defining half spaces of a convex polyhedron: the standard case where a bounded intersection exists (a), the case where no intersection exists (b), and the case where an unbounded intersection exists (c).

Definition 3. *The points on the boundary of a convex polyhedron are the points that classify on ($\odot$) for at least one defining half space, and interior ($\ominus$) for all other defining half spaces of the polyhedron.*

Any defining half spaces that do not contribute to the boundary of a polyhedron are called *redundant*.

3.3 Rule based volumetric shape modeling

This section describes the basic entities and operations for the proposed approach of rule based shape modeling. A rule takes an input shape, the parent, applies an operation that may or may not create new output shapes, the children.

The Scope

In computer science, the term *scope* denotes part of a program where the binding of a named entity, a variable, is valid. Similarly, in the spirit of *CGA Shape* [133], the scope in the proposed generative modeling system is the part of the generative model that corresponds to the current context when evaluating a rule.

Each scope consists of the following attributes

- a convex boundary $\mathcal{CP}$
- a shape $\mathcal{S}$ (the geometry)
- a rigid transformation $\mathcal{T}$ that defines a local coordinate system, composed of a position p_s and orthogonal axis $\vec{x_s}$, $\vec{y_s}$ an $\vec{z_s}$.
- a set of named additional attributes $\mathcal{A}$

The set of named attributes correspond to properties that are necessary for visualization (e.g. visibility or material), or can be used to propagate semantic information to subsequent rules.

Rules

In this subsection, I describe a notation of the proposed rule system. The proposed system was much influenced by parametric set grammar systems [213]: A formal grammar is defined as the tuple

$$G = (N, T, P, S) \tag{3.4}$$

where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- P is the set of production rules,
- and S is the start symbol.
- The empty symbol is denoted by ϵ.

Each production rule $p \in P$ is of form $p : LHS \rightarrow RHS$, where LHS, the left-hand side, corresponds to one symbol in N (parent), and RHS, the right-hand side, corresponds to one or more symbols in $\{N \cup T \cup \epsilon\}$ (children). The children automatically inherit all attributes from their parent, production rules may add or alter attributes.

When evaluating production rules, each symbol on the left side is associated to exactly one scope; several production rules with the same symbol but different scopes may exist. Each symbol on the RHS of a rule may be associated to one or more scopes, depending on the operation. Assigning several scopes to one RHS label means producing that number of non-terminal symbols.

The production rule determines the method of shape transformation; conceptually, I distinguish between the following types of production rules that are grouped with respect to a "default" modification regarding the scope attributes:

instantiation : rules that create an instance of a shape given its parameters (e.g. create a prism). In this case $\mathcal{S}$ is ignored, i.e. the LHS shape is irrelevant.

construction : rules that create new shapes that depend on the shape $\mathcal{S}$ of LHS, i.e. refinement and construction operations.

attribution : rules that define new attributes other than $\mathcal{S}$, in this case the shape $\mathcal{S}$ from LHS is inherited, like the other attributes.

The general notation for a production rule p is:

$$p : \alpha \underset{op(p)}{\rightarrow} \beta_0, \beta_1, \ldots \tag{3.5}$$

where α is the non-terminal label, *op* denotes the modeling operation that changes or creates a shape using the parameters p. Optional parameters are enclosed with angle brackets, e.g. $\langle a \rangle$, and β denotes the RHS labels of this rule, also denoted as output labels.

The ability to generalize is limited in systems using a static list of output labels, as was also found to be a limitation for Krecklau and Kobbelt [103]. Their solution accepts non-terminal symbols as parameters in modeling rules. For example, a window structuring rule, i.e. splitting borders, insets and window sills, might be the same for a range of windows with different glass partitions. In my implementation I chose to allow for a right-hand symbol β to be either a label name or an anonymous function with no parameters that yields a label name. The evaluation scope of this function has access to the Scope attributes; storing a label name in an attribute allows to define an exchangeable label for an evaluation hierarchy, i.e. the sub-tree.

Modeling Operations

The modeling operations are grouped into low level operations, which provide the simplest basic mechanisms to manipulate shapes in the proposed systems, and higher level operations, which utilize the low level operations.

The system defines only two low level modeling operations: create half spaces and compose polyhedra from such half spaces. All volumetric modeling operations are expressed in terms of these two basic low level operations. Intersecting an existing polyhedron with another half space is simply expressed by adding the half space to the list of defining half spaces for this polyhedron.

The volumetric modeling operations can further seen as either constructing new volumes (e.g. placing a box, or extruding a face), or subdivision operations, where the geometry of a scope is replaced by non-overlapping smaller geometries. Therefore, most volumetric modeling operations are concerned with the way they construct the half spaces which constitute the resulting polyhedron (or polyhedra).

Specification of Half Spaces and Convex Polyhedra

There are several meaningful ways to specify a half space; the concrete method depends on the use case. The methods are depicted in Figure 3.6.

normal $\vec{n}$ and point p : The tuple $(\vec{n}, p)$ defines a half space with outward normal $\vec{n}$, where p is a point on the plane ($\odot$). The implicit form is easily derived from case ($\odot$) in Equation (3.2).

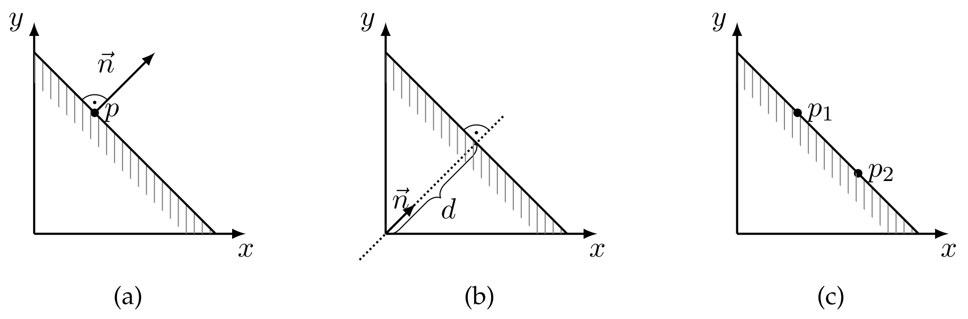


Figure 3.6: Three different ways to specify a half space in 2D: by point p and normal direction $\vec{n}$ (a), by normal direction $\vec{n}$ and distance to origin d (a), or by two plane defining points p_1 and p_2 (three points in 3D) (a). Note that the order of points is important to define the orientation of the half space in the last case.

normal $\vec{n}$ and distance to origin d : $(\vec{n}, d)$, in this case the implicit form is already given.

three points p_1, p_2, p_3 : Three ordered points define a triangle, from which a plane description can easily be retrieved. The normal vector is orthogonal to two triangle edges, which can easily be calculated using the cross product:

$$\vec{n} = (p_2 - p_1) \times (p_3 - p_1) \tag{3.6}$$

Assuming the points are given in counter clockwise direction. The implicit form is retrieved as in the first case, using $\vec{n}$ and any one of the three points.

A convex polyhedron P is simply specified by a list of n bounding half spaces $\{h_1 \cdots h_n\}$, see also Equation (3.3), in which case the rule is an instantiating rule.

Parametric Basic Shapes

While any planar bounded convex shape can serve as scope in the proposed system, using parameterized versions of specific shapes simplifies the modeling task. Only the type of shape, e.g. a box, and a few parameters, e.g. position, orientation or size have to be specified in these instantiating rules.

The **platonic solids** are a special group of three-dimensional convex polyhedra. Each vertex is adjacent to the same number of faces, and all faces are congruent. It can be easily shown that only 5 of such polyhedra exist, in fact, the proof is a classical textbook example. The five platonic solids are the Tetrahedron, the Octahedron, the Cube, the Dodecahedron and the Icosahedron (see Figure 3.7).

The rule for instantiating such a basic shape, e.g. a cube, is

$$\alpha \underset{\texttt{shape-cube}(\vec{c},\vec{u},s)}{\longrightarrow} \beta \tag{3.7}$$

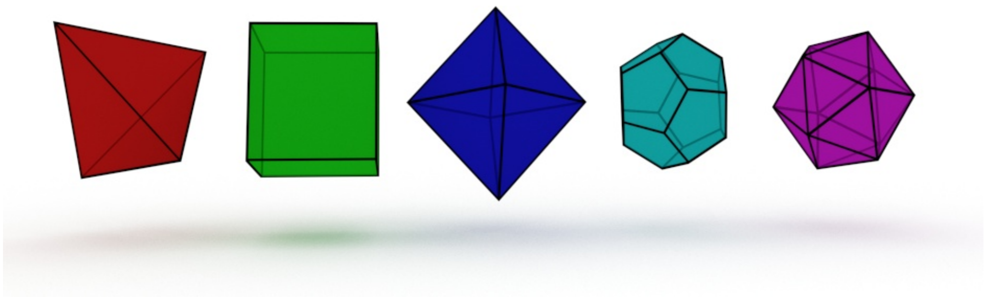


Figure 3.7: Only five convex polyhedra exist that have congruent faces and the identical number of faces that meet in each vertex, these polyhedra are called the *platonic solids*.

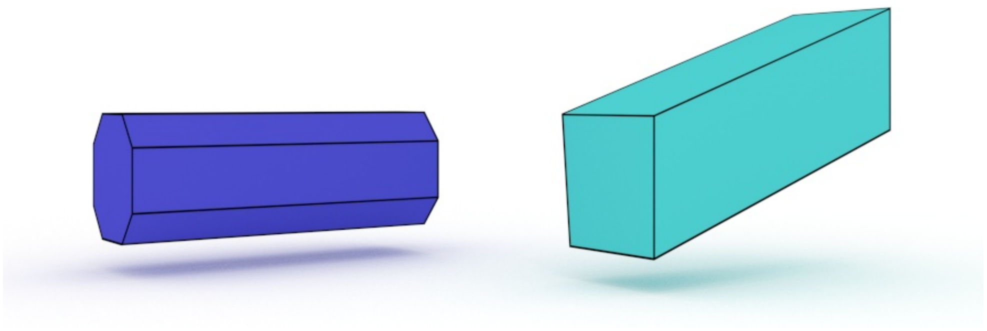


Figure 3.8: Another type of basic shapes in the proposed system are prisms, which are defined via extrusion of a convex polygon. Note that the extrusion direction need not be orthogonal to the polygon, as is demonstrated on the right side.

The `shape-` rules instantiate a new basic shape by constructing their according bounding planes (a static arrangement for the platonic solids), using the parameters center $\vec{c}$, the up vector $\vec{u}$ and a scale factor s. These rules inherit the shape of scope α.

General **prisms** are polyhedra that contain two congruent faces, with all remaining faces being parallelograms. The `shape-prism` operation constructs a subset of general prisms, namely prisms with two convex congruent faces. Its parameters are the polygon points $[p_0, p_1, \cdots, p_i]$, the extrude direction $\vec{u}$, and the extrude length l (see also Figure 3.8.

$$\alpha \underset{\texttt{shape-prism}([p_0,p_1,\cdots,p_i],\vec{u},l)}{\longrightarrow} \beta \tag{3.8}$$

Cylinders are just special cases of prisms with polygon points describing a circle. But any curved surfaces or round objects, like cylinders or spheres, have to be approximated by several planar segments.

In most cases it is sufficient to combine these basic shapes to create more complex shapes, special, i.e. domain specific, shapes can be added to the system by defining a new rule that composes bounding planes in the desired manner.

Convex Splits

Convex splits are a generalized volumetric refinement operation that belong to the construction category. A *convex split* partitions a shape into non-overlapping convex parts (a convex partition).

Generalized Interval Splits

The `split-axis` and `repeat-axis` operations using absolute and relative sizes were inspired from CGA shape, [135] but have been generalized to arbitrary convex splits. Both split operations are defined using an arbitrary direction (axis) $\vec{a}$, and a list of absolute and relative parts i, which is called the *split interval*: $I = [i_0 \cdots i_n]$ as the split is eventually calculated using an one-dimensional interval that is split into n sub-intervals. Example: $[1, \tilde{1}]$ denotes a split into two parts, relative sizes are specified using the $\sim$ symbol.

$$\alpha \underset{\texttt{split-axis}(\vec{a},[i_0 \cdots i_n])}{\longrightarrow} \beta_0 \cdots \beta_n \tag{3.9}$$

The `split-axis` operation partitions a shape into n parts, as specified in the split interval, using split planes orthogonal to the split direction $\vec{a}$. Absolute values describe the length of the corresponding sub-interval. The length of sub-intervals of relative values is determined from the remaining available space of the original interval, see Figure 3.9.

$$\alpha \underset{\texttt{repeat-axis}(\vec{a},l_m)}{\longrightarrow} \beta \tag{3.10}$$

The `repeat-axis` operation subdivides a shape along the split direction $\vec{a}$ into as many parts as possible, with each part having a minimum length l_m when being projected along the split direction. The actual number n of parts depends on the size of the interval upon rule application. All parts are assigned to the same output label β.

In a conventional box grammar system, the intervals are measured along the bounding box of the shape in principal coordinate directions only. The generalized interval split determines the spatial scope by projecting the convex hull on an axis defined by the search direction, which leads to the interval, see Figure 3.10. As a consequence, box grammar splits can be carried out by simply using the main coordinate axis as search directions.

Frame Split The operation `split-frame` is another versatile convex split operation. In principle, it corresponds to an offsetting operation, that partitions a shape into an inner and outer (border) parts, see Figure 3.11. Such offsetting operations can be solved via skeletons, e.g. the *straight skeleton* algorithm presented by Aichholzer et al. [4]. The straight skeleton is however sensitive to small input

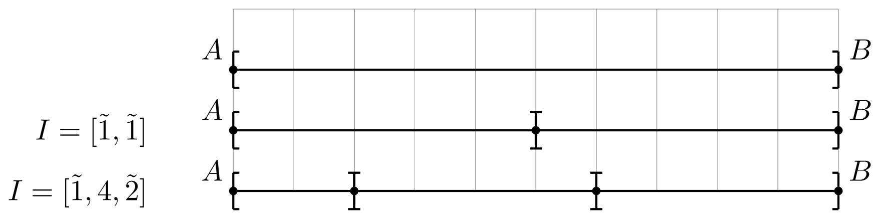


Figure 3.9: Example of several interval splits using `split-axis` on the interval $[A, B]$ (upper line): the split interval $[\tilde{1}, \tilde{1}]$ partitions the interval into two equally sized parts (middle line). The interval $[\tilde{1}, 4, \tilde{2}]$ partitions the interval into three parts, with the middle part having a length of 4 units, and the left and right parts have a length ratio of 1:2, in this case 2 and 4 units.

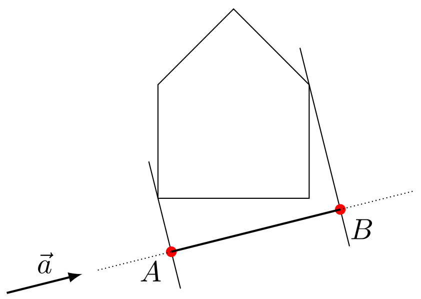


Figure 3.10: The `split-axis` and `repeat-axis` operations calculate the extend of the split interval $[A, B]$ by orthogonally projecting the convex hull of the shape on a given axis, the search direction $\vec{a}$.

segments, and a numerically stable implementation is demanding. Therefore, I devised the `split-frame` operation that yields similar output to a skeleton based offset by simulating the offset by an outward extrusion: Given a convex shape, the shape is shrunk by translating all boundary planes by a given offset in their negative normal direction, creating the inner part. Now, for each edge on the inner polyhedron, a bisector plane is created. The outer parts are created using the corresponding bounding planes of inner and outer polyhedra, as well as the adjacent bisector planes.

$$\alpha \xrightarrow[\texttt{split-frame}(d, \langle lbl \rangle)]{} \beta_i, \beta_f \tag{3.11}$$

`split-frame` takes the parameter d, the inward displacement distance, and $\langle lbl \rangle$, a set of labels that indicate faces that should not get displaced, i.e. these $\langle lbl \rangle$, a set of labels that indicate faces that should not get displaced, i.e. these faces will not be part of the frame. The rule assigns the created scopes to two output labels, the *inner* scopes β_i and the *frame* scopes β_f.

Bevel Split. The `split-bevel` operation mimics a chamfering; bevel is a well-known operation available in many 3D modeling programs. It constructs

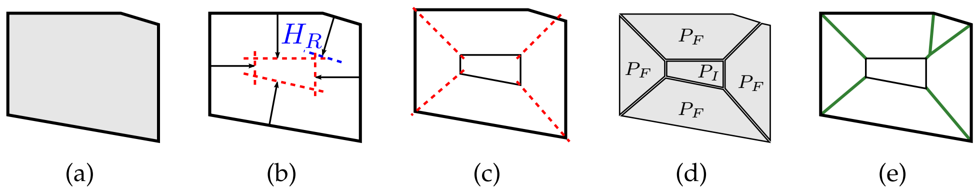


Figure 3.11: Two-dimensional demonstration of the `split-frame` operation: The operation decomposes a convex shape into an inner part and parts for selected bounding planes. The boundary planes of an input polyhedron (a) are offset to create the inner polyhedron (b). Note that the plane H_R does not contribute to the boundary of the inner polyhedron. A bisector plane is created for each vertex of the inner polyhedron (or corresponding edge for 3D polyhedra), as can be seen as red dotted lines in (c). The final partition consists of the inner polyhedron P_I and a polyhedron P_F for each boundary plane of the inner polyhedron (d). By comparison, offsetting using the straight skeleton [4] produces a different partition (e).

a bevel plane for each edge that is adjacent to a list of given planes, identified by labels $\langle lbl \rangle$. The bevel planes are constructed for each edge using a normal direction $\vec{n}$, and a point p, where the point p is the edge midpoint and $\vec{n}$ is the sum of the normals of the two adjacent faces. The number of beveled edges is e. The bevel planes are moved in opposite normal direction by a given distance d:

$$\alpha \underset{\texttt{split-bevel}(d,\langle lbl \rangle)}{\longrightarrow} \beta_i, \beta_{b0} \cdots \beta_{be-1} \tag{3.12}$$

The inner part β_i is constructed by adding bevel planes to the input scope, a bevel part β_b is constructed for each beveled edge, such that the union of all bevel parts and the inner part yields the input shape.

Radial Split. The `split-radial` operation corresponds to a radial partition of a scope in slices with identical angles. Its parameters are the center point of the radial split p, the radial axis $\vec{r}$, a vector in the first split plane $\vec{d}$, and the number of slices s. The operation constructs s split planes, and creates each slices by intersecting the input scope with two bounding split planes that form the slice wedge. The normal of the first split plane corresponds to the direction $\vec{r} \times \vec{d}$. See also Figure 3.12 for an example of a radial split with 5 slices. To facilitate a versatile usage, the local coordinate system of each scope is oriented with respect to the slice bounding split planes, such that the $\vec{z}$-axis points to the bisector between these split planes, and the $\vec{y}$-axis points to the radial axis $\vec{r}$.

$$\alpha \underset{\texttt{split-radial}(p,\vec{r},\vec{d},s)}{\longrightarrow} \beta_0, \beta_1, \cdots, \beta_{s-1} \tag{3.13}$$

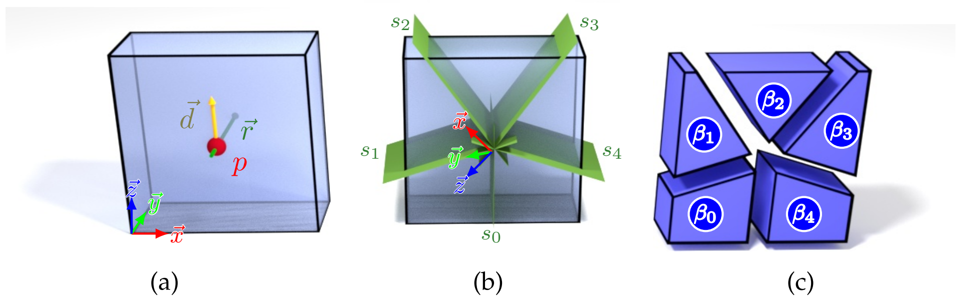


Figure 3.12: The `split-radial` operation partitions a convex shape into pieces, bounded by split planes that are oriented radially around a given point p and a rotation axis $\vec{r}$, as well as a starting direction $\vec{d}$, as seen in (a). Given the number of slices s, The operation constructs split planes s_0 to s_4 (b) and creates the slices (b). Note that the local coordinate system of each slice is oriented to give a standard orientation within each slice, with z pointing to the bisector of the bounding split planes of the slice, as is shown for slice β_0 in (b).

Extrude

Extrusion is a well-known principle in CAD modeling, and can be seen as the generation of a shape by sweeping a profile along a straight line. The term extrusion is also used in process technology to describe the creation process of solids with a fixed cross-sectional profile. Typically, extruding creates a new shape. Thus, extrusions correspond to the growing metaphor.

Move Plane Technically not really an extrusion, this method is described here because it serves a similar purpose for modeling.

$$\alpha \xrightarrow[\texttt{move-plane}(\vec{s},d)]{} \beta \tag{3.14}$$

The direction vector $\vec{s}$ specifies a search direction. From the input shape α, the bounding plane having the normal vector that is most most similar to $\vec{s}$ will be translated by the distance d in its normal direction.

Extrude Face

This operation creates a new polyhedron, via extrusion of the face polygon of a corresponding bounding plane by a given direction.

$$\alpha \xrightarrow[\texttt{extrude-face}(\vec{s},d)]{} \beta_0, \beta_1 \tag{3.15}$$

Again, $\vec{s}$ specifies a search direction, and the face of the polyhedron with its normal most similar to $\vec{s}$ will be selected. β_0 is identical to α, as α was consumed by this rule, and β_1 corresponds to the extruded polyhedron. The bounding planes are constructed from the flipped face plane, the translated face plane, and a

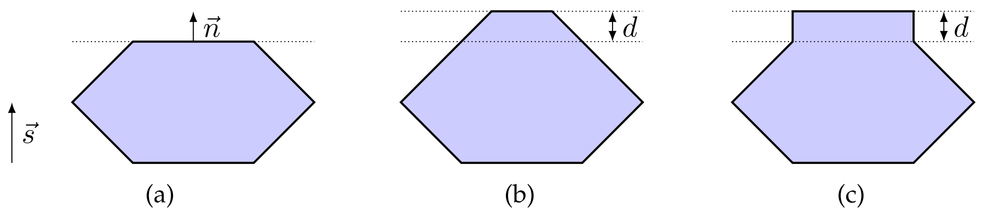


Figure 3.13: The difference between `move-plane` and `extrude-face`, when being applied to a shape: The face with the direction of its normal vector $\vec{n}$ being most similar to the extrude direction $\vec{s}$ is selected (a). With `move-plane`, the face bounding plane is translated by d units in normal direction (b), whereas `extrude-face` will introduce new bounding planes orthogonal to the face plane (c). Note that depending on the geometric configuration, it is possible that `move-plane` (b) might also make the moved bounding plane redundant or the polyhedron empty.

plane for each face polygon edge that goes through the edge and has a normal vector orthogonal to the face normal.

See also Figure 3.13 for a graphical interpretation of these operations.

Merging Scopes

The operation `merge-scope` takes several input scopes and merges them into one new scope, whose geometry is the union of the input geometries. The scope boundary is calculated as the convex hull of the resulting shape [13], thus allowing to build a non-convex shape from convex primitives.

Generic Boolean Operations

Naturally, Boolean combination of volumetric entities constitute a powerful tool for geometric modeling. Splits can be seen as some constrained Boolean operations; each split corresponds to the intersection with a split volume.

As convex polyhedra are already defined as intersection of half spaces in the proposed system, the other Boolean operations union and difference can be expressed using only intersection and complement using De Morgan's laws. The complement of a half space is easily obtained by swapping exterior and interior as described in Section 3.2.

This method to perform Boolean operations is related to BSP tree merging [137], as described in Section 2.3; each convex polyhedron can be seen as a degenerated BSP tree with one inside leaf. The implemented method does not use a BSP representation, but treats a shape as a flat list of convex polyhedra, i.e. only the *inside* - leaves, and utilizes the BSP tree leaf merging operations to combine polyhedra. For example, subtracting a convex polyhedron b from a convex poly-

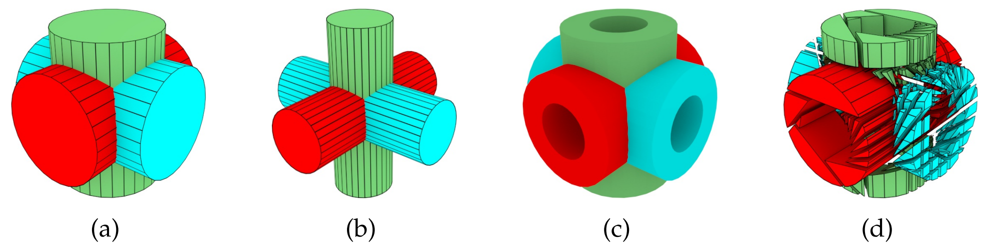


Figure 3.14: Boolean operations: A configuration of "hole" cylinders (b) is subtracted from a set of outer cylinders (a), which yields the configuration (c). This highly nonconvex object is represented by a union of non-overlapping convex polyhedra, which are here visualized using an offset operation in (d), where each bounding plane of a convex part has been slightly displaced inward. The partition is similar to binary space partitioning.

hedron a means to intersect a with all flipped bounding half spaces of b. Similar to the improved BSP merging method presented by Lysenko et al. [121], merging stops when an intersection is empty.

The result is expressed by a set of non-overlapping polyhedra; their union represents the shape after an operation.

3.4 Robust Evaluation

This section concerns itself with the aspects of robust and performant implementation of the proposed system. Parts of this section were published in [111].

An implementation of a system utilizing the modeling operations mentioned in the foregoing Section 3.3 has to deal with the robustness problem (as described in Section 2.6), as it will have to calculate intersections of half spaces and construct new shapes from the result.

In the proposed system, shapes correspond to convex polyhedra, which can be represented by the intersection of half spaces. The shape is therefore fully represented by the list of half spaces. For any geometric reasoning, the boundary surface of the polyhedron is relevant, therefore the system has to perform a conversion from a list of half spaces to a boundary representation: a manifold mesh. This is known in the literature as *vertex enumeration* of a convex polyhedron represented by half spaces.

Due to the nature of this rule based modeling system, namely constructing new polyhedra from existing polyhedra, an incremental algorithm is preferred, as it allows caching of intermediate results between rules. Because of this, and the fact that polyhedra in this system are always in either 2 or 3 dimensions, it was chosen to not use vertex enumeration algorithms (e.g. based on Motzkin's double description method [60]) but represent the polyhedron by a mesh and implement a plane clipping algorithm.

Quantized Plane Based Representation

Plane based representations of meshes store the geometric information of a mesh into faces (plane equations) instead of vertices (point coordinates).

Sugihara et al. [179] pointed out that such a representation allows to robustly perform Boolean operations on polyhedra as there need no constructions to be made, the geometric information of the resulting mesh consists of face planes of the input meshes only. Vertex positions are calculated only when they are needed, e.g. for visualization.

This led to the choice for using a plane based mesh representation for the intersection algorithm, with integral plane coefficients. The choice for integral coefficients was motivated as the intersection points of hyperplanes in this representation correspond to rational values. Therefore, the geometry kernel could be implemented using only integer operations, and kept exact by using static filtering techniques [58], which require a lot less implementation and computational effort than floating-point filtering techniques [169]. Using limited length integers for plane coefficients reduces the number of possible planes in this representation to a discrete set (see also Figure 3.15).

Notation: The proposed system uses two coordinate systems, world coordinates and plane grid coordinates. World coordinates are written without special notation (e.g. x), plane grid coordinates are denoted with a hat (e.g. $\hat{x}$), and are not necessarily integer. Coefficients and variables in integer representation are denoted by a star, e.g. a^*. A half space in this representation is therefore noted by $(\vec{n}^*, d^*)$. For the static analysis, the function $B(x)$ is used to denote an upper bound for the number of bits necessary to store a value (or intermediate value) in the given calculation tree. Given a half space $(\vec{n}^*, d^*)$ in this representation, the constants $B_{\vec{n}^*}$ and B_{d^*} represent the number of bits that are used to represent plane coefficients of defining planes in the proposed system, which is a value that can be chosen to balance between evaluation speed and accuracy.

$$\begin{gathered}\vec{n}^* = \begin{bmatrix} a^* \\ b^* \\ c^* \end{bmatrix} \subset \mathbb{Z}^3 : a^*, b^*, c^* \in \{-(2^{B_{\vec{n}^*}-1}) \dots 2^{B_{\vec{n}^*}-1} - 1\} \\ d^* \in \mathbb{Z} : d^* \in \{-(2^{B_{d^*}-1}) \dots 2^{B_{d^*}-1} - 1\}\end{gathered} \tag{3.16}$$

Statically Filtered Geometric Predicates

It is well known that many geometric intersection algorithms can be expressed via few geometric predicates and topological modifications based on the results of these predicates. Typically, these predicates are expressed by signs of 4x4 Matrix determinants (e.g. for point-plane classification).

In order to guarantee a correct result of such a predicate, static filtering analysis [58] determines the necessary number of bits for a calculated result. If all calculations are carried out using large enough integers, the result will be exact, and therefore correct. In contrast, adaptive filtering for geometric predicates [169] is typically directly applied to floating-point representations. Creating an adaptive filter implementation of a predicate by hand is a tedious task, therefore formal methods for automatic generation of geometric predicates have been developed [131]. The proposed system uses statically filtered integer predicates, because the static filter analysis just determines the necessary number of bits to guarantee that the direct implementation of predicates yields a correct result. Therefore, no additional overhead filter code needs to be generated, and no additional run-time penalty will be introduced, which is an important factor for the evaluation of a possibly complex generative model.

The necessary number of bits for the following predicates depends on the constants $B_{\vec{n}^*}$ and B_{d^*} that limit the resolution of a plane in integer representation. Thus, by choosing appropriate values for these constants allows balancing between *resolution*, i.e. the possible number of representable planes and intersection points, and *efficiency*, i.e. all calculations can be carried out in machine precision.

Intersection Points

To increase performance, polyhedra vertices (which correspond to intersection points of three planes) are cached in an exact intermediate representation in the proposed system. Point - plane classification is carried out on this representation instead of always calculating 4x4 determinants, which reduces the total number of arithmetic computations: Intersection points, which are rational expressions, are represented as points in homogeneous coordinates [171]. An intersection point p of three planes represented by the tuples $(\vec{n_1}, d_1)$, $(\vec{n_2}, d_2)$ and $(\vec{n_3}, d_3)$ can be determined by this simple system of linear equations:

$$A^* = \begin{bmatrix} \vec{n_1^*}^T \\ \vec{n_2^*}^T \\ \vec{n_3^*}^T \end{bmatrix}, \vec{d^*} = \begin{bmatrix} d_1^* \\ d_2^* \\ d_3^* \end{bmatrix}, A^* \cdot \hat{p} + \vec{d^*} = 0 \tag{3.17}$$

which yields the point p in homogeneous coordinates with D being the homogeneous factor:

$$\hat{p} = \begin{bmatrix} adj(A^*) \cdot (-\vec{d^*}) \\ det(A^*) \end{bmatrix} = \begin{bmatrix} p_N^* \\ D^* \end{bmatrix} \tag{3.18}$$

with $adj(A)$ being the adjugate of the 3x3 matrix A (i.e. the transposed cofactor matrix). As each element of $adj(A)$ is calculated by a 2x2 minor of elements with size $B_{\vec{n}^*}$, the number of necessary bits for storing an element $\vec{p_N^*}$ are:

$$B(p_N^*) = 2 \cdot B_{\vec{n}^*} + B_{d^*} + 3 \tag{3.19}$$

The determinant D^* can be expressed by the triple product $\langle \vec{n_1}, (\vec{n_2} \times \vec{n_3}) \rangle$, therefore we obtain

$$B(D^*) = 3 \cdot B_{\vec{n}^*} + 3 \tag{3.20}$$

Point - Plane Classification

The implementation of Equation (3.2) of an intersection point $\hat{p}$ with respect to a half space $h = (\vec{n}^*, d^*)$ corresponds to the evaluation of the sign of $P_{PH}(\hat{p}, h) = \langle \vec{n}^*, \hat{p} \rangle + d^*$. As $\hat{p} = [x^*, y^*, z^*, D^*]^T = [p_N^*, D^*]^T$ is given in homogeneous coordinates, we can apply a few simple transformations to remove the division of the homogeneous factor from the predicate:

$$\begin{aligned} P_{PH}(\hat{p}, h) &= sgn(\langle \vec{n}^*, \hat{p} \rangle + d^*) \\ &= sgn(D^*) \cdot sgn(\langle \vec{n}^*, p_N^* \rangle + d^* \cdot D^*) = \begin{cases} -1 & \rightarrow \ominus \text{ interior} \\ 0 & \rightarrow \odot \text{ on} \\ +1 & \rightarrow \oplus \text{ exterior} \end{cases} \end{aligned} \tag{3.21}$$

The number of necessary bits to calculate this predicate is

$$B(P_{PH}) = 3 \cdot B_{\vec{n}^*} + B_{d^*} + 6 \tag{3.22}$$

Plane Indexing

In this section I will describe an exact and efficient method for maintaining a strict weak order between planes stored as $(\vec{n}^*, d^*)$-tuples using integer plane coefficients. This allows us to implement plane indexing using generic data structures (containers) and algorithms. However, the $(\vec{n}^*, d^*)$-representation is not canonical, therefore it has to be normalized. The idea here is that we develop an element-wise normalized comparison method without actually performing normalization on $(\vec{n}^*, d^*)$-tuples. Using this comparison we can order tuples using a lexicographical ordering scheme: Given a tuple $(\vec{n}^*, d^*)$, we write the HNF of $\langle \vec{n}^*, p \rangle + d^* = 0$ with $p = [x, y, z]^T$ and $\vec{n}^* = [a^*, b^*, c^*]^T$:

$$\frac{a^*}{\|\vec{n}\|} \cdot x + \frac{b^*}{\|\vec{n}\|} \cdot y + \frac{c^*}{\|\vec{n}\|} \cdot z + \frac{d^*}{\|\vec{n}\|} = 0 \tag{3.23}$$

In this form, lexicographical ordering of two planes is achieved by comparing respective coefficients, e.g. the first coefficient a of two planes:

$$\frac{a_1^*}{\|\vec{n_1}\|} < \frac{a_2^*}{\|\vec{n_2}\|} \tag{3.24}$$

As planes are required to be valid ($\|\vec{n}\| > 0$), we can multiply by the denominators. To evaluate this predicate exactly with respect to the closure of the used data type (integers), we reformulate this equation so that it contains no square roots. Therefore, we introduce the new quadratic coefficients $\tilde{a}$:

$$\tilde{a} = \begin{cases} a^2 \text{ iff } a \geq 0 \\ -a^2 \text{ iff } a < 0 \end{cases} \tag{3.25}$$

The plane index comparison predicate P_{PI} for one coefficient is then written

$$\tilde{a}_1 \cdot (a_2^{*2} + b_2^{*2} + c_2^{*2}) < \tilde{a}_2 \cdot (a_1^{*2} + b_1^{*2} + c_1^{*2}) \tag{3.26}$$

For lexicographical ordering, a $<$-operator for $(\vec{n}, d)$-tuples would first compare coefficient a of two planes using Equation (3.26), unless the left and right side of this equation are equal, in which case coefficient b is used, etc.

Vertex Indexing

It is also highly desirable to index the cached polyhedron vertices in rational representation efficiently. Similar to the plane indexing method presented in the last Subsection, a simple lexicographical ordering method is derived:

The comparison between the x-coordinate of two rational points in homogeneous coordinates $\hat{p}_1 = [x_1^*, y_1^*, z_1^*, D_1^*]^T$ and $\hat{p}_2 = [x_2^*, y_2*, z_2^*, D_2^*]^T$ is

$$\frac{x_1}{D_1} < \frac{x_2}{D_2} \tag{3.27}$$

Again, we multiply the denominators in order to remove the divisions, however, the signs of the denominators have to be taken into account, which yields:

$$x_1 \cdot D_2 \diamond x_2 \cdot D_1 \tag{3.28}$$

with the comparison operator $\diamond$:

$$\diamond = \begin{cases} < \text{ iff } (D_1 < 0 \wedge D_2 \geq 0) \vee (D_1 \geq 0 \wedge D_2 < 0) \\ > \text{ iff } (D_1 < 0 \wedge D_2 < 0) \vee (D_1 \geq 0 \wedge D_2 \geq 0) \end{cases} \tag{3.29}$$

This leads to the lexicographical comparison predicate P_{VI} for each component of a vertex in rational representation, as has been described in the foregoing subsection (the plane indexing approach). The necessary bits for this predicate are given by

$$B(PVI) = B(p_N) + B(D) = 5 \cdot B_{\vec{n}^*} + B_{d^*} + 6 \tag{3.30}$$

Geometric Rounding / Plane Quantization

In this section, I discuss methods for converting planes from world coordinates to the proposed representation.

The choice to represent planes in the system using fixed length integer coefficients leads to a discrete set of representable planes using these coefficients (see also Figure 3.15), the *plane grid*. Note that this grid is different from discrete hyperplanes as found in the literature [8] in that the intersection points are rationals and therefore the coordinates of intersection points are not limited to integral numbers. See Figure 3.15 for a visualization of a plane grid with low precision. The surface of any shape bounded by planes in this representation is a subset of the arrangement of the plane grid.

In the proposed system, I utilize a plane based representation for polyhedral meshes, which means that faces of such polyhedra are represented by planes from the set of representable planes of the plane grid. However, direct specification of the integral plane coefficients is not intuitive for procedural modeling. Therefore, it is necessary to convert from a more convenient description to the plane grid. Furthermore, the same applies when importing arbitrary geometric data.

Conversion from World Coordinates to Plane Grid

In principle, the plane grid is associated to an axis aligned bounding box (AABB) in world coordinates, i.e. translation and scale (see also Figure 3.16). This means that a plane grid of planes specified by $(\vec{n}^*, d^*)$-tuples, with is defined in respect to a world coordinate system in $\mathbb{R}^3$ (see also Equation (3.16)).

$$\begin{aligned} P_{\mathcal{O}} \in \mathbb{R}^3 &: \text{the plane grid origin in world coordinates} \\ w &: \text{the width of the plane grid bounding box in world coordinates} \\ \hat{w} &: \text{the width of the plane grid bounding box in plane grid coordinates} \\ B(\vec{n}) \in \mathbb{N}^+ &: \text{max bits for component of } \vec{n} \\ B(d) \in \mathbb{N}^+ &: \text{max bits for component } d \end{aligned} \tag{3.31}$$

The conversion from a point p world coordinates to a point $\hat{p}$ in plane grid coordinates is a simple translation and scaling operation, and is expressed by

$$\hat{p} = \frac{(p - P_{\mathcal{O}}) \cdot \hat{w}}{w} \tag{3.32}$$

For an efficient implementation it is practicable to choose w and $\hat{w}$ such that both are a power of two.

A half space described by a point $p = [p_x, p_y, p_z]^T \in \mathbb{R}^3$ and a normal vector $\vec{n} = [n_x, n_y, n_z]^T \in \mathbb{R}^3$, $\|n\| = 1$ as shown in Figure 3.6a, given in world coordinates, is converted to the plane grid by choosing a *near* plane from the plane grid

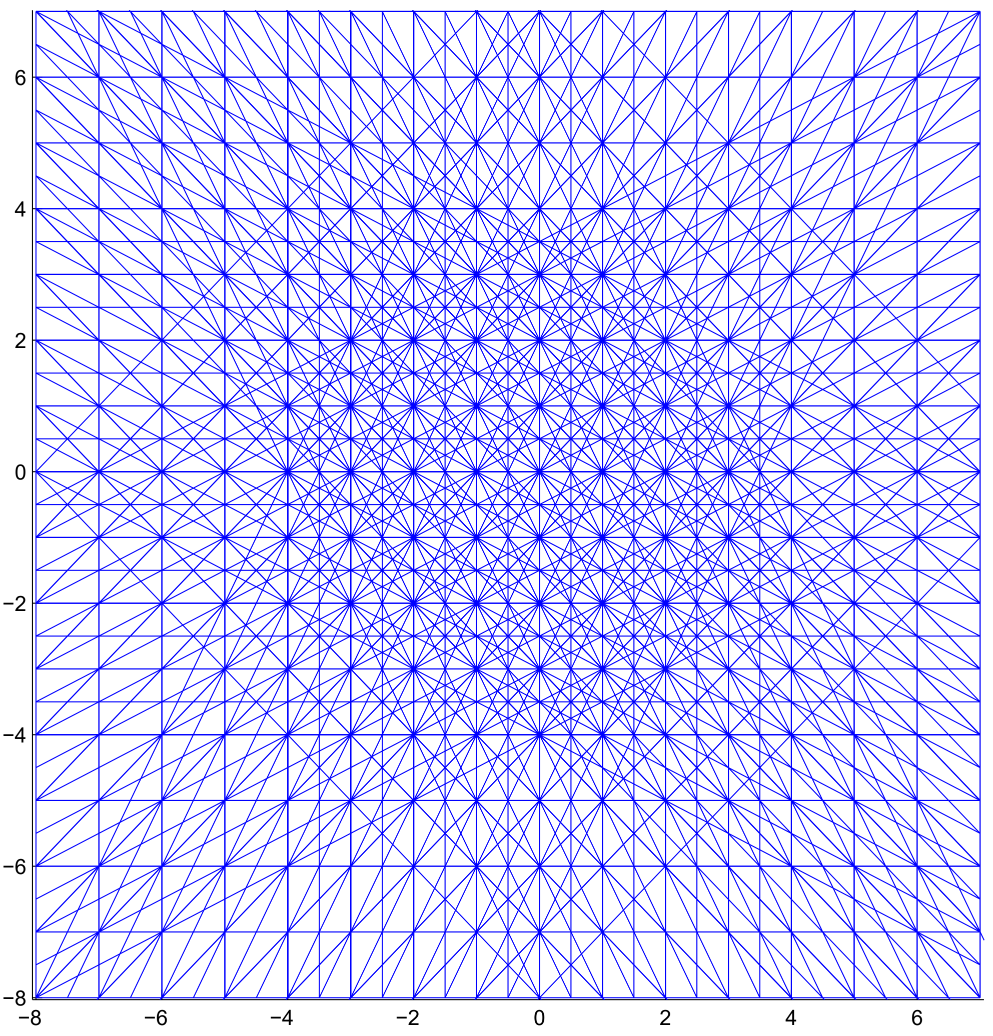
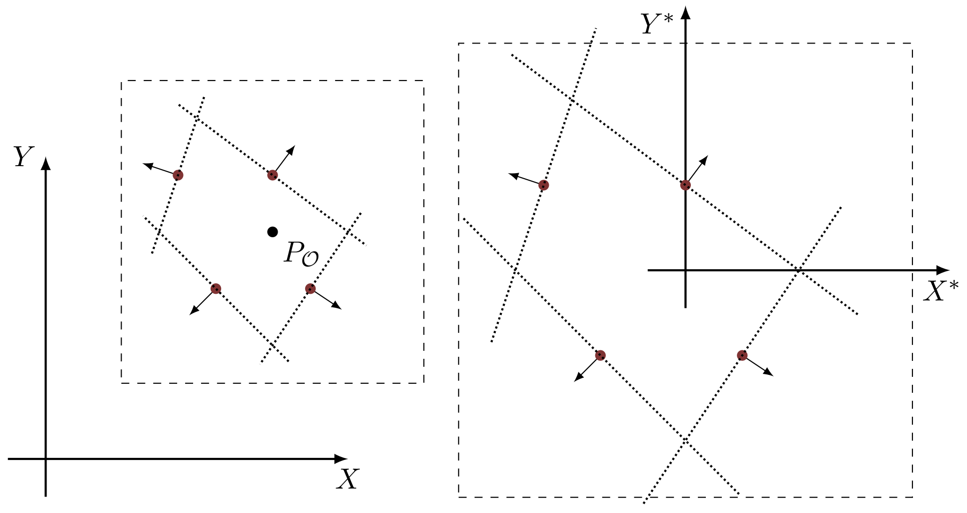


Figure 3.15: The discretization of plane coefficients leads to a discrete set of possible planes, the *plane grid*, when using a fixed length description for the plane coefficients $(\vec{n}, d)$. This image shows a visualization of all possible planes having coefficients of $\vec{n} \in \{-2, .., 1\}$ (2 bit) and $d \in \{-8, .., 7\}$ (4 bit), all coefficients $\in \mathbb{Z}$. It can be seen that even with equidistant planes, the intersection points are not evenly spaced; furthermore, there is an area of higher resolution near the center. By varying the number of bits the resolution can be balanced between evaluation speed and precision.



(a) (b)

Figure 3.16: As the plane based representation of the proposed system yields highest resolution around the origin, a transformation from world coordinates (a) to the plane grid (a) is introduced. The plane grid is defined by its origin point $P_{\mathcal{O}}$ and the size of the bounding box (dashed lines) in world and plane grid coordinates. Inside the plane grid bounding box, intersections are guaranteed to be carried out robustly, as they are exact.

to the point in plane grid coordinates. p and $\vec{n}$ are stored as floating-point values. The coefficients (n^*, d^*) of the fixed size integer representation the following way:

The **normal component** $\vec{n}^*$ corresponds to a vector in integral coordinates. World coordinates are stored as a floating-point number f. The IEEE754 standard [90] defines the most commonly used binary representation of floating-point numbers.

Any floating-point number can be expressed as

$$f = m \cdot 2^e \tag{3.33}$$

where m is called the mantissa, and e the exponent. Thus, the floating-point number can be seen as the following fraction, assuming that $e > 0$:

$$f = \frac{m}{2^{-e}} \tag{3.34}$$

The exact integer representation of $\vec{n}$ is found by a scaling of $\vec{n}$ such that all elements of $\vec{n}$ are integer, assuming without loss of generality that $e < 0$:

$$\begin{aligned} \vec{n} &= \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \frac{m_x}{2^{-e_x}} \\ \frac{m_y}{2^{-e_y}} \\ \frac{m_z}{2^{-e_z}} \end{bmatrix} \\ \vec{n}^{*1} = \vec{n} \cdot 2^{-e_x} \cdot 2^{-e_y} \cdot 2^{-e_z} &= \begin{bmatrix} m_x \cdot 2^{-e_y} \cdot 2^{-e_z} \\ m_y \cdot 2^{-e_x} \cdot 2^{-e_z} \\ m_z \cdot 2^{-e_x} \cdot 2^{-e_y} \end{bmatrix} \end{aligned} \tag{3.35}$$

if $0.9e \leq 0$, that coefficient is already integer in the form of Equation (3.33), therefore the nominator does not need to be multiplied.

This vector can be shortened by dividing all components of $\vec{n}^{*1}$ by their greatest common divisor (gcd), if it exists. If this vector fits into the chosen plane grid ($B(\vec{n}^{*1}) \leq B(\vec{n})$), an exact corresponding normal vector has been found. Otherwise, a suitable representative vector in the plane grid has to be chosen. In my implementation i chose to round to the nearest point of $\vec{n}$ scaled to the length of half the plane grid boundary.

$$\vec{n}^* = \lfloor 2^{B(\vec{n})-1} \cdot \vec{n} \rceil \tag{3.36}$$

The **distance component** d^* is simply obtained from the implicit plane equation via rounding:

$$\langle \vec{n}^*, p \rangle + d = 0 \to d^* = \lfloor -\langle \vec{n}^*, p \rangle \rceil \tag{3.37}$$

Constructing Orthogonal and Bisector Planes

For operations like `extrude-face` or `split-frame`, new planes have to be constructed. Thus, the newly constructed planes may not be representable in the plane grid, and need to be *rounded*. This introduces a slight deviation of the normal vector of the plane. Therefore, an appropriate plane from the representable set has to be selected such that the geometric rounding error is as small as possible.

For the `extrude-face` operation, a new plane has to be constructed that is orthogonal to a given face, i.e. its defining half space, and that goes through an edge, which is defined as the intersection of an adjacent half space with the face half space, compare also in Figure 3.13. Therefore, the normal vector of the orthogonal plane can be constructed by orthogonal projection of the normal vector of the adjacent half space, (see also Figure 3.17):

Given two half spaces A and B, the normal vector $\vec{n}_C$ for half space C is constructed by orthogonal projection of $\vec{n}_B$ onto $\vec{n}_B$, similar to Gram-Schmidt Orthogonalization [10]:

$$\vec{n}_C^* = \vec{n}_B^* - \vec{n}_A^* \cdot \frac{\langle \vec{n}_B^*, \vec{n}_A^* \rangle}{\langle \vec{n}_A^*, \vec{n}_A^* \rangle} \tag{3.38}$$

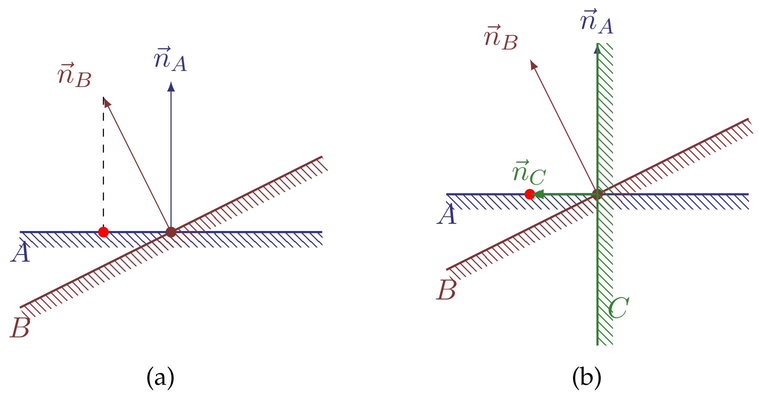


Figure 3.17: Construction of an orthogonal plane C from a plane A and an edge that lies in A and is defined by the intersection of A and B: The normal vector $\vec{n}_C$ of C is constructed by orthogonal projection of the normal vector $\vec{n}_B$ of B onto A.

which is rewritten to remove the fraction:

$$\vec{n}_C^* \cdot \langle \vec{n}_A^*, \vec{n}_A^* \rangle = \vec{n}_B^* \cdot \langle \vec{n}_A^*, \vec{n}_A^* \rangle - \vec{n}_A^* \cdot \langle \vec{n}_B^*, \vec{n}_A^* \rangle \tag{3.39}$$

Thus, the result of Equation (3.39) yields a scaled version of $\vec{n}_C^*$, which can be shortened by dividing each vector element through their greatest common divisor. If the resulting vector does not fit into the plane grid, it has to be rounded (similar to Equation (3.36)). After this step, the distance component d_C^* is determined like in Equation (3.37), using a point on the edge as plane defining point, which yields the tuple $(\vec{n}_C^*, d_C^*)$.

Edge Bisectors. The `split-frame` operation requires the construction of bisector planes (see also Figure 3.11). The bisector planes will be constructed with respect of two defining half spaces of a convex polyhedron whose intersection forms an edge, see Figure 3.18.

The bisector normal direction is derived by the following considerations: All points p in the bisector plane C with respect to two non-parallel half spaces A and B yield the property that the signed euclidean distance to half spaces A and B are equal, assuming without loss of generality that the orthogonal projection of p onto the edge, which is defined by the intersection of A and B, is the origin.

$$d_1 = \frac{-\langle \vec{n}_A^*, p \rangle}{\|\vec{n}_A^*\|} = d_2 = \frac{-\langle \vec{n}_B^*, p \rangle}{\|\vec{n}_B^*\|} \tag{3.40}$$

As C intersects the origin as well, this equation can be rewritten into implicit form, and determine the direction of the normal component of C:

$$\vec{n}_C = \vec{n}_A^* \cdot \|\vec{n}_B^*\| - \vec{n}_B^* \cdot \|\vec{n}_A^*\| \tag{3.41}$$

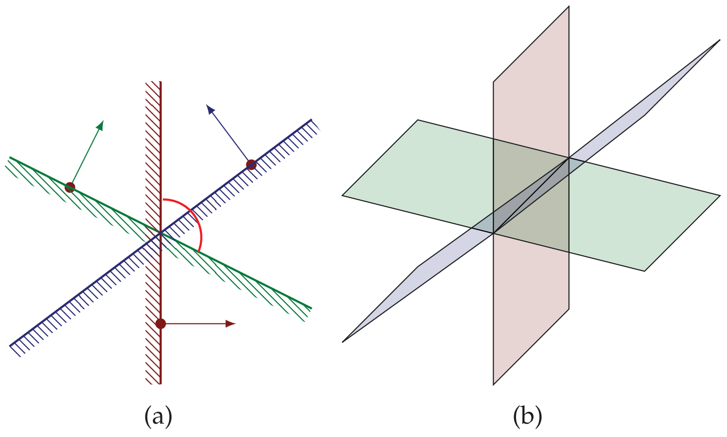


Figure 3.18: The `split-frame` operation requires the construction bisector planes for edges of a convex polyhedron: A bisector plane C (blue) is constructed from two edge-defining planes A (green) and B (red).

unfortunately, the multiplication terms with vector norms prevent a direct evaluation in integers for most cases, therefore, my choice of implementation converts integer plane normals into a normalized representation $\vec{n}_A^\circ$ and $\vec{n}_B^\circ$ using floating-point, calculating $\vec{n}_A^\circ - \vec{n}_B^\circ$ and - converting (rounding) the normal back to the integer representation using the method described in Equation (3.36). Finally, the distance component of C is determined using a point of the edge and Equation (3.37).

Note that bisector and some orthogonal plane constructions may need to be rounded, and thus can lead to micro fragments.

Boundary Evaluation

Up to now, I have described which points belong to the interior and the boundary of a polyhedron, and shape rules that construct half spaces and define convex polyhedra via sets of half spaces. At some point, the boundary, or surface, of such convex polyhedra has to be evaluated. This boundary evaluation can be seen as an incremental construction: a consecutive intersection of half space interiors, or, consecutive clipping of an object that is large enough to represent the shape with planes, similar to classical Sutherland-Hodgman style polygon clipping [181].

This approach fits best into our generative modeling application, as the majority of operations consists of consecutive splits of convex polyhedra. A split is easily expressed by the intersection of an object with the interior and the exterior of a given half space. The bounding box of the plane grid is chosen as the initial polyhedron representing the available space. Thus, the only operations needed for surface evaluation are the intersection of a convex polyhedron with the interior of a half space, and flipping the interior and exterior part of a half space (see also Equation (3.2)).

Convex Polyhedron - Half space Intersection

The clipping algorithm follows the classical approach of modification of the predicates. It evaluates the boundary B of a convex polyhedron defined by the half spaces $H = \{(\vec{n_1}, d_1), (\vec{n_2}, d_2), ..., (\vec{n_k}, d_k)\}$. If the intersection clips parts of the polyhedron away, due to the convexity, exactly one new face, the *horizon* will be introduced. The algorithm proceeds through the following stages, see also Figure 3.19 for an accompanying example.

1. **Initialization** The initial polyhedron represents the whole space. Therefore, the half-edge data structure representing the boundary B is initialized with the predefined topology of the bounding box of the plane grid.
2. **Vertex Classification** For each intersection of a half space $h \in H$, the vertices of B are classified into $\ominus$, $\odot$ or $\oplus$ with respect to h (see Equation (3.21)).
3. **Insert Horizon** Corresponding horizon vertices and edges are inserted.
4. **Remove Clipped part** The vertices, edges and faces in the exterior half space (marked $\oplus$) are removed.

Vertex positions are stored as homogeneous coordinates, representing exact intersection points, see Equation (3.18). Any additional half space intersection (e.g. a split) on B can be carried out efficiently by repeating steps 2-4.

Mesh export

The presented approach evaluates the surface of one convex polyhedron. Nonconvex or more complex objects can be expressed by a union of several polyhedra. Exporting the surface of each polyhedra of such a union yields a lot of excessive geometry inside the union. As all polyhedra in the system are bounded per definition, their union is therefore also bounded and watertight. This union can be calculated using the Boolean approach described in Section 3.3: a naive approach to calculate this union goes as follows: the surface of the union of a set of polyhedra can be found by subtracting all other polyhedra from each polyhedron of the set. In practice, I implemented the surface extraction via a 2D polygonal difference operations for each face of the polyhedron, using the polygon clipping technique proposed by Bernstein et al. [23]. Naivly subtracting all faces of all polyhedra for each output face is time-consuming; this can be speed up with a standard spatial search structure to determine intersecting polyhedra before subtraction. I used a variant of axis aligned bounding box (AABB) trees [203] to efficiently test if two polyhedra intersect before calculating their intersection.

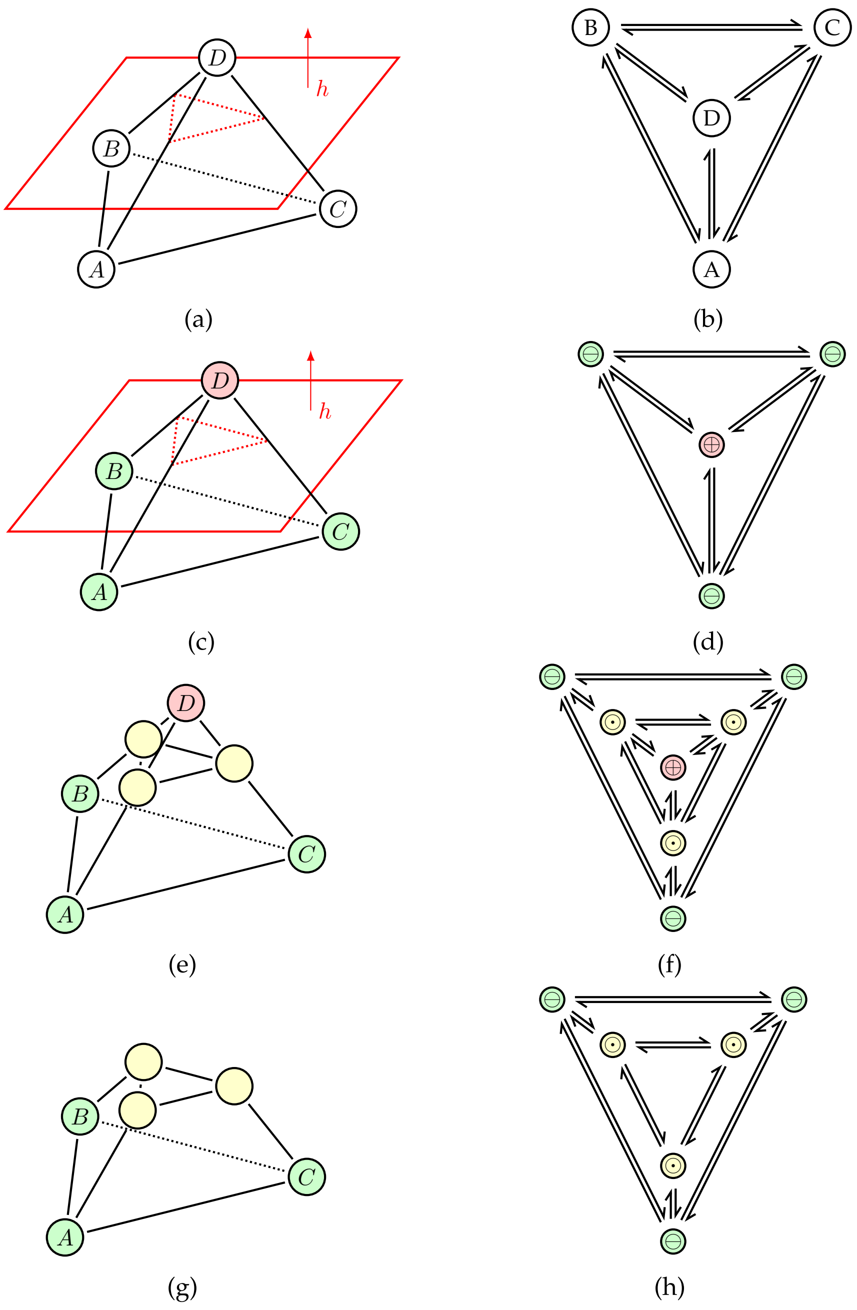


Figure 3.19: Clipping of a tetrahedron with a half space: First, the vertices of the boundary representation (b) are classified with respect to the clipping half space h (d). Then, horizon vertices and edges are inserted (f), and the parts residing in the exterior of h are removed (h).

Geometric Queries

To facilitate reusability, the rules that compose parametric elements should be created in a way that makes them applicable in a wide range of situations, i.e. to a wide range of shapes. Furthermore, application should be robust. If the shape does not correspond to assumptions that are made by the rule, e.g. that the input polyhedron is a box, the rule should still produce results - as good as possible.

In parametric CAD and similar systems, such non-robustness is often tied to the *persistent naming problem* [124], which states that "referenced entities must then be named in a persistent way in order to be able to reevaluate the model in a consistent manner. In particular, when a reevaluation leads to topological modifications, references between entities used during the design process are frequently reevaluated in an erroneous way, giving results different from those expected".

Typically, identifying such references is done e.g. by using an index of a topological entity, such as a face or a vertex. In the proposed system, my solution for this problem is to express such references not by indices but as search directions, specified in the local coordinate system. One example is the `split-axis` operation, that calculates the extent of the split interval by searching for minima and maxima of projections of polyhedron vertices on search directions.

nearest vertex : If the polyhedron is topologically evaluated, i.e. the mesh representation exists, the nearest vertex with respect to a given plane $(\vec{n}, d)$can be found by directly querying the distances from mesh vertices using the calculated distance from Equation (3.1):

$$\arg\min_{\vec{v}} |\langle \vec{n}, \vec{v} \rangle + d| \tag{3.42}$$

the optimal vertex with the shortest distance to a given search direction $\vec{s}$ is simply found by

$$\arg\min_{\vec{v}} \langle \vec{s}, \vec{v} \rangle \tag{3.43}$$

query face : For many operations, e.g. `extrude-face`, one polyhedron face needs to be identified. Given a search direction $\vec{s}$ that is specified with respect to a local coordinate system, the face with its face normal $\vec{n}_F$ being most similar to $\vec{s}$ will be selected:

$$\arg\max_{\vec{n}_F} \langle \frac{\vec{n}_F}{\|\vec{n}_F\|}, \frac{\vec{s}}{\|\vec{s}\|} \rangle \tag{3.44}$$

3.5 Generative Architecture

The proposed system provides various parametric elements that can be combined to create a complex model, with a focus on classical architecture, as this domain exhibits much structural regularity. This chapter presents some higher level methods for modeling of architecture using procedural half spaces.

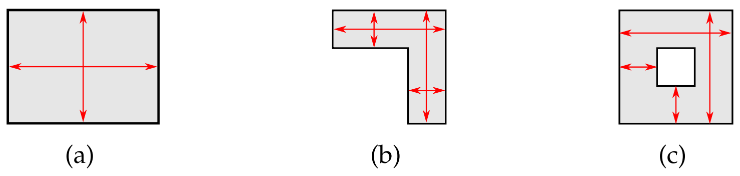


Figure 3.20: Examples of simple parametric footprints. The red arrows correspond to parameters that allow to alter the shape within its design space.

Modeling follows a coarse to fine approach, similar to established procedures in the community: first, the coarse outline or mass model of a building are defined, then the appropriate rules are applied to generate more and more detail.

Footprints and Mass Models

In order to model the coarse outline of a building, several methods are applicable. In some cases, it suffices to refine one of the basic shapes. Other cases can be modeled by specifying a 2D-polygon and create walls using extrusion. Some simple examples are shown in Figure 3.20. Another approach is by specifying several objects and combine them to create a more complicated shape; this approach also mimics an architects' way of creating an initial design using wood or clay. This approach is also known as mass modeling, and is found within many architectural modeling suites (e.g. Autodesk Revit Architecture). This approach is typically expressed using Boolean set operations, or constructive solid geometry (CSG). CGA shape addressed this problem using query operations that detect intersections at rule level and choose the appropriate rule for an intersection case, e.g. to suppress windows that would be intersecting with a wall of a neighboring building. The proposed system uses a different approach: intersection is geometrically evaluated using the CSG methods available with convex polyhedra, see also Section 3.3.

Basic Shape Refinement

In many cases, it is sufficient to instantiate and refine a basic shape (basic shapes have been introduced in Section 3.3). For more complicated shapes, either a polygonal description can be given that is extruded to a volumetric representation, or a set of basic shapes can be combined to yield a more complicated volumetric basic representation.

Structuring a Building

In general, the modeling paradigm of the proposed system corresponds to a coarse to fine strategy, i.e. a rule replaces a scope by smaller sub-scopes. In such a manner, a building can be structured by first creating its outer hull. Afterwards, it is partitioned into outer walls and floors. Then, windows are split into walls and indoor space is further subdivided into rooms, and so on.

Footprints: working with polygons

In the proposed system, footprints are described by a union of convex polygons, e.g. for floor slab creation, or a boundary polygon. In the general case, the former will be derived from the latter, either manually or with a convex partition algorithm [63]. From boundary polygon segments, walls can be extruded using bisector partition, using `split-frame` (3.11). Creating a floor slab is created by extruding the convex polygons, whose union represent the floor, into prisms using `shape-prism` (3.8). Both approaches can be seen in Figure 3.22.

After extrusion, the walls can be split into storeys and floor slabs.

Mass Modeling

Mass modeling is the method that is used by architects in an early stage of design, when a general concept is being developed. This is often done using clay or wood to create a physical representation of an early design concept. This terminology was also used by Pascal Müller et al. [135] for describing the coarse structure of a building, and is also used in commercial tools like Autodesk Revit [212].

As an example, we examine the mass model of a building that is shown in Figure 3.24: The building consists of three rectangular parts, each having four parameters: footprint width and length, building height (not shown in the figure), and rotation angle.

The three parts are combined into an united mass model, after that further refinement rules are applied until the final model is obtained, see Figure 3.23 for the final result.

After the creation of the general mass model structure, I split the coarse structure into walls and an inner part, using the `split-frame` operation. The walls and the inner part are then split into floors; the result after this step is a volume for each wall in each floor, volumes for floor slabs, and a volume that represents the inner part of the building.

Parametric Elements

In order to maximize the expressibility and the reusability, a generative modeling system should facilitate the creation of generic rules that can be applied in several situations. I give examples on the creation and design rationale of parametric elements for the domain of façade modeling of classical architecture.

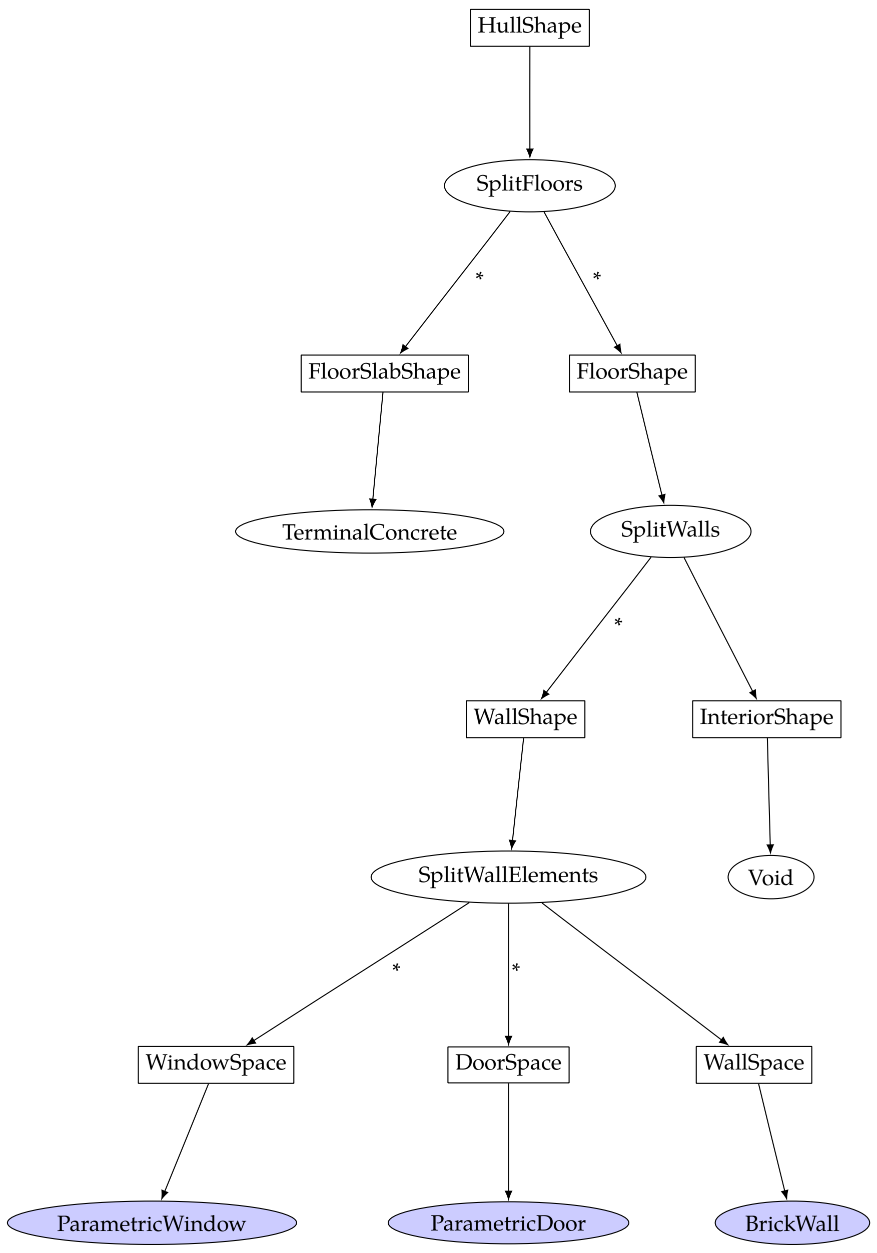
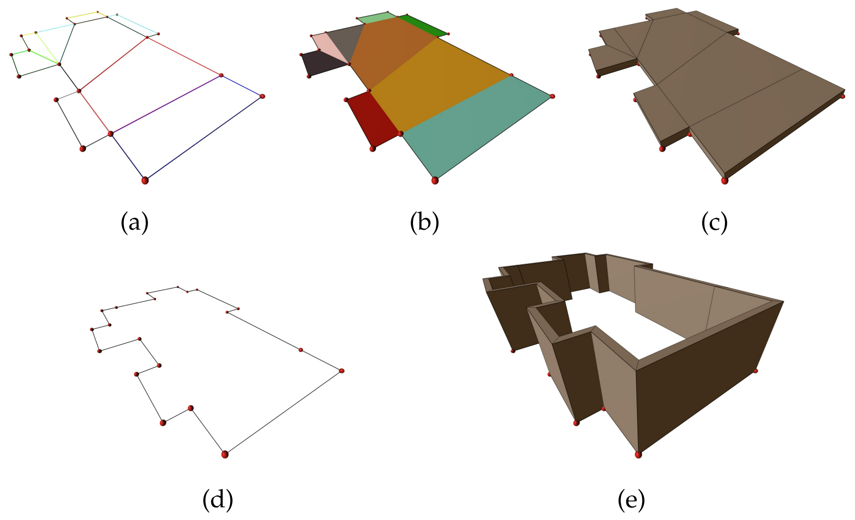


Figure 3.21: The *split graph* is a data flow representation of a concrete generative model in the proposed system. Rectangular nodes represent shapes, i.e. convex polyhedra, and elliptic nodes, such as `SplitWallElements`, represent rules. Parametric rules that correspond to a terminal element, e.g. a window, are highlighted by a blue background.



(a) (b) (c)

(d) (e)

Figure 3.22: Working with footprint polygons: A convex partition (b) of a footprint polygon (a) is used to extrude floor slabs (c). Furthermore, with an operation that creates a convex polygon using bisector partition similar to the `split-frame` operation, walls can be extruded (e) from the footprint boundary (e).

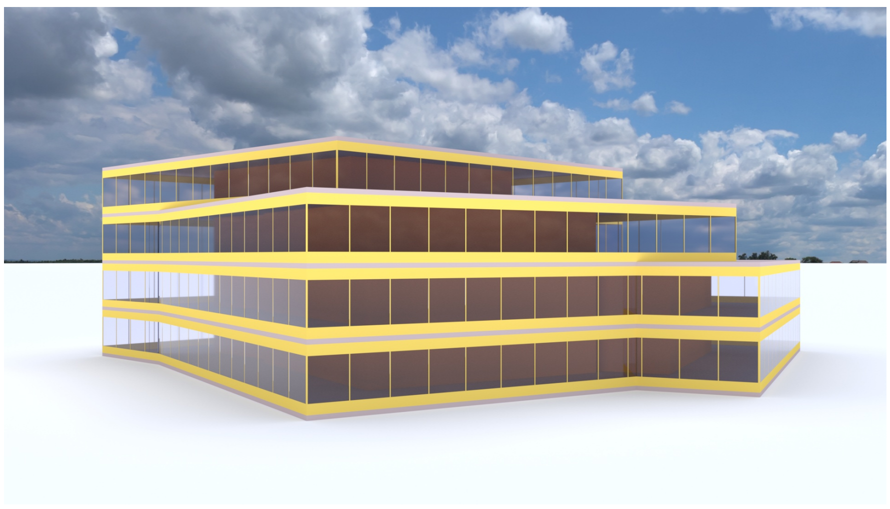


Figure 3.23: Rendering of an instance of the freeform office example.

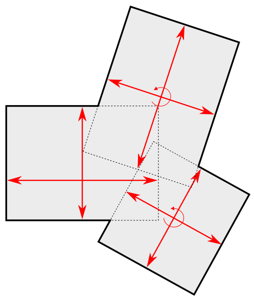


Figure 3.24: Examples of a simple parametric mass model.

Windows, Doors and Arches

Parametric elements, like windows, can be seen as terminals. I propose the following structure for such elements: Each parametric element (e.g. a rectangular window) can be configured by a set of parameters (e.g. the number of glass partitions). To facilitate their usage, the element should also provide default values for all parameters.

Windows. In the following, I will detail the progress of the creation of a parametric window: A modern rectangular window typically consists of an outer frame that is placed in the walls. A window wing (or wings) is mounted inside this frame, which consists of another framing that holds the window glass. This framing may be partitioned into various forms of subdivision.

The available space reserved for a window is then split into frame and glass parts using the split graph structure shown in Figure 3.25. This parametric window rule partitions the available space in outer and inner frame volumes, and an inner part that will be used for the glass partitions. The outer and inner frames are split using absolute measurements, the glass parts are defined using relative measurements and will therefore automatically adapt to the available space. The remaining inner part corresponds to the glass partition, the concrete type of partition is chosen by the rule that is written in an attribute which defaults to `termGlass`.

Doors are similar to windows, in terms of splitting structure, for the proposed generative modeling system. Simple doors can be expressed using a similar splitting strategy than the rectangular window, but the door tile has no space below the inner door rule, and terminal rules correspond to wood, mostly.

Arches, curved structural members spanning an opening or recess, typically to support the structure above the opening and distribute vertical pressure laterally. I will describe a rule that creates a one-centered semicircular arch: Given an input scope, the rule fits the largest inscribing circle, a remaining degree of freedom is resolved with respect to the up direction of the local coordinate system. The rule will then create split planes for the circular arc and partition the scope

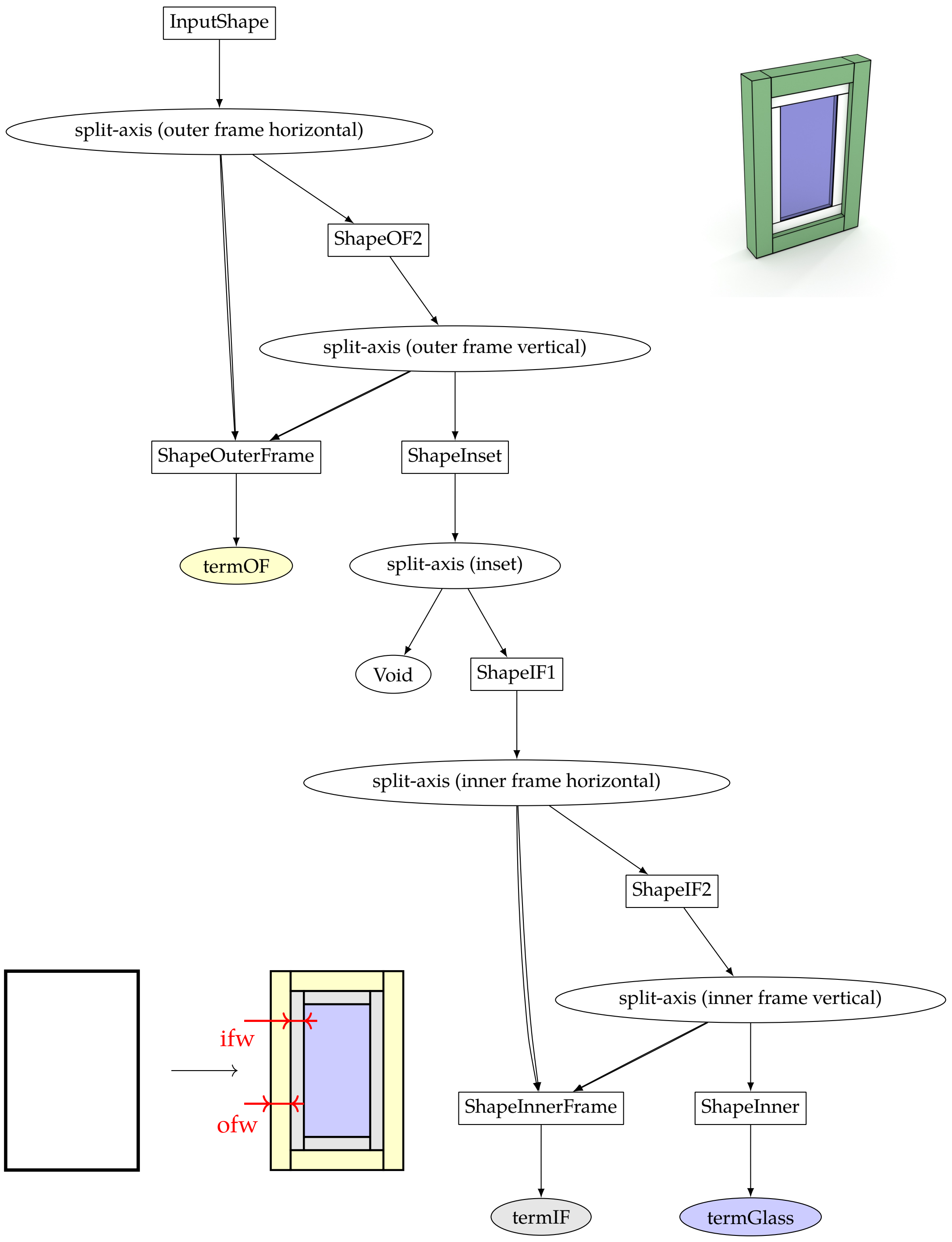

InputShape
split-axis (outer frame horizontal)
ShapeOF2
split-axis (outer frame vertical)
ShapeOuterFrame
ShapeInset
termOF
split-axis (inset)
Void
ShapeIF1
split-axis (inner frame horizontal)
ShapeIF2
split-axis (inner frame vertical)
ifw
ofw
ShapeInnerFrame
ShapeInner
termIF
termGlass

Figure 3.25: The split graph of the parametric window tile shows the application of several `split-axis` operations to split an outer and inner frame. The filled ellipses correspond to terminal rules for this parametric rule. By using attributes to represent the concrete rule name, behavior can be exchanged on per-application base of the parametric window terminal, e.g. by replacing the *termGlass* glass terminal rule by a 2x2 window partition. The attributes `ofw` and `ifw` correspond to the with of the outer and inner frames.

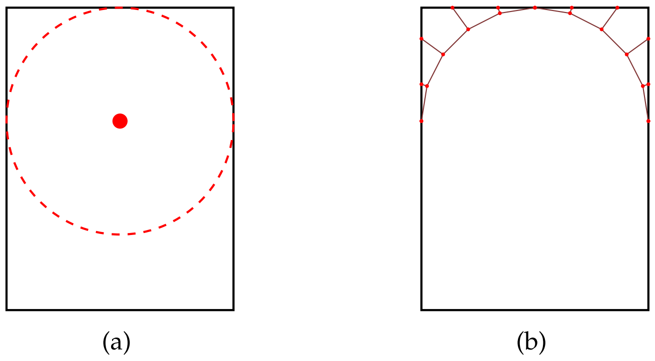


Figure 3.26: The *arch* rule will fit the largest circle in the bounding box of a shape, with respect to the local coordinate system upper $\vec{z}$ direction. The rule produces one (convex) inner part, and outer parts of the arch - whose union yields the nonconvex outer part.

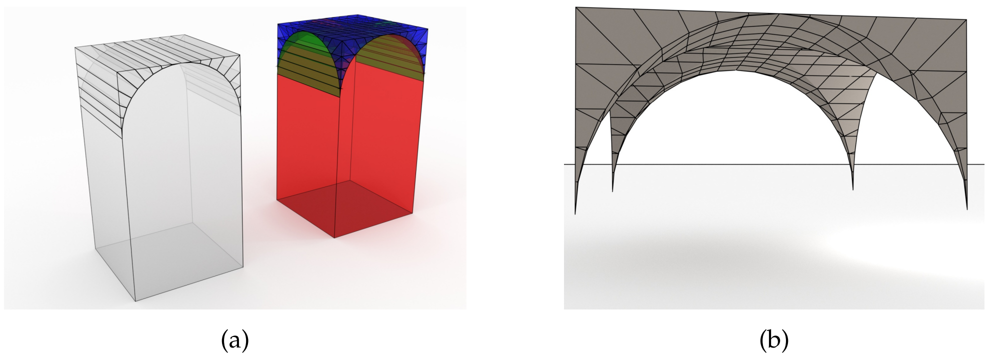


Figure 3.27: Consecutive application of the arch rule yields a cross vault: The first application of the arch rule partitions a rectangular scope into the part below and above the archline (a), applying the arch rule again to the upper part in the orthogonal direction yields a volumetric cross vault partition. The lower part from the first rule application is shown in red, the lower part from second rule appliation is shown in green, and the upper part from the second application is shown in blue. This part also corresponds to a cross vault structure, as is shown in the right picture (b).

into the inner arch part and the outer part, see also Figure 3.26.

Some other variants of rounded structures, e.g. holes can be made using a similar approach than with arches, and is not described in detail here.

Pillars and Columns have been realized by fitting the largest inscribing circle to the polyhedron face that is selected by an given up direction.

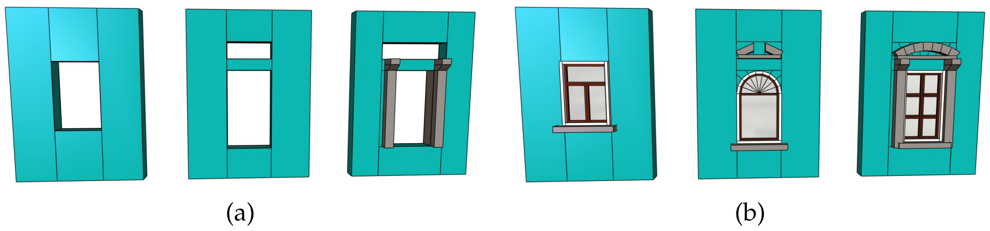


Figure 3.28: Structuring of often occuring façade layout stiles is done using a tile approach: the tile rules TileWindow, TileWindowGable and TileWindowPillarGable (a) define the general structure, a great variety of façade tiles can be quickly created by populating the tiles with parametric window and gable terminals (b).

Façade structuring

Visually examining neoclassical façades shows recurring patterns in the design of such façades. Many patterns can be explained using irregular sized rectangular grids [160]. This irregular grid can be created on a façade using split rules, which partitiones the façade into rectangular regions called *tiles* These tiles will be used to distinguish between different types of facade parts, e.g. parts with windows or doors. A façade tile rule will split the input shape into a wall and inner element parts. In these examples, I assume the following local coordinate system assumption: The up-axis is in Z-direction, the X-direction points to the right when looking frontal at the façade, and the Y-direction corresponds into that viewing direction.

A window tile rule set looks like the following, the attributes `att-ww` corresponds to the window width, and `att-wh` corresponds to the window height. The window will be horizontally centered, with the window sill placed 1.1m above the lower edge of the tile:

$$\begin{aligned} \textsc{TileWindow} &\xrightarrow[\texttt{split-axis}(\vec{X},[\tilde{1},\texttt{att-ww},\tilde{1}])]{} \textsc{Wall}, \textsc{WindowColumn}, \textsc{Wall} \\ \textsc{WindowColumn} &\xrightarrow[\texttt{split-axis}(\vec{Z},[1.1,\texttt{att-wh},\tilde{1}])]{} \textsc{Wall}, \textsc{WindowRule}, \textsc{Wall} \end{aligned} \tag{3.45}$$

3.6 FaMoS - Interactive Façade modeler

In the course of this thesis I also worked on a method to define rule sets interactively, instead of modifying the textual representation of a shape grammar. The modeler was presented at a workshop [107]. This system builds upon a shape grammar implementation of the proposed system using convex polyhedra in the generative modeling language (GML) [72]. The modeler itself was written using C# with windows forms.

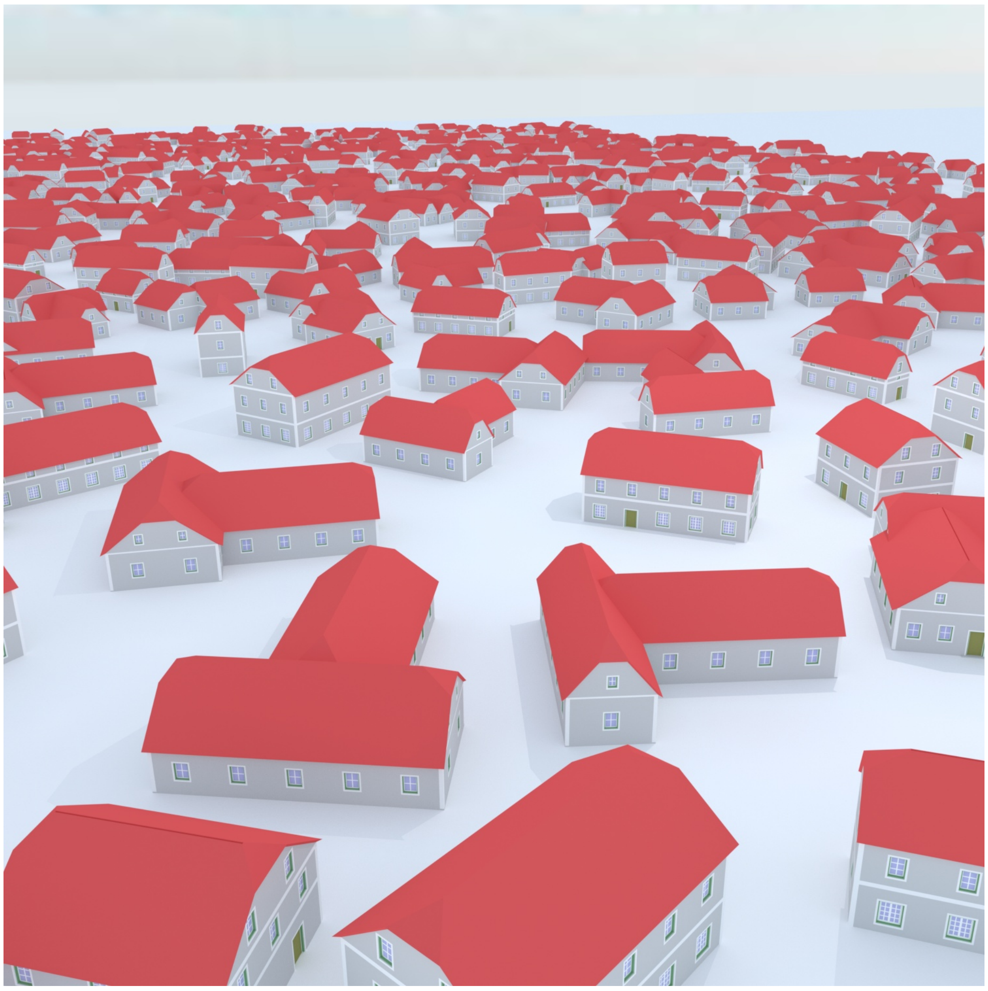


Figure 3.29: Variations of similar buildings can be generated using random intervals for values like width and height of a building. This randomly populated set of houses contains no two exactly similar houses - and is generated from two basic building layouts.

I abstracted the process of writing a rule set into a graphical user interface using the following requirements:

- interactively creating/manipulating split operations in the 3d view, similar to the system proposed by Markus Lipp et al. [119].
- grouping of several split operations into rules
- manipulating attributes

Each split partitions a scope into several child scopes. Grouping a number of consecutive splits into a rule yields a tree structure with one scope as input and an arbitrary number of child scopes, the leaves of the tree, as output. The interface is built upon manipulating such tree structures. The trees are transformed to GML code, which yields the geometry after evaluation.

Interface overview

The user interface (called *FaMoS*, the *Fa*cade *Mo*deling *S*ystem) consists of 5 main components that are shown in Figure 3.30. A Model consists of a collection of rules (split trees), which are grouped in referenceable rules and instance rules (which actually produce geometry). Manipulation and selection operations on the tree result in direct visual feedback in the 3D View.

Mapping user input to code. When designing a user interface that utilizes user input on the 3D view, the user actions have to be mapped to according code. This also means the persistent naming problem has to be addressed: the question of how to robustly identify entities in the model, on which future constructions depend, is not trivial.

Picking

Picking in the 3D view is well suited for intuitive interaction techniques. However, the pick operations that are carried out on actual 3D geometry have to be mapped back to operations that identify the picked object within a rule - which should be independent of geometry.

Therefore, vertices of a convex polyhedron are expressed within generated code as search directions in the scopes LCS, for finding the outermost vertex in that direction. This means that exactly the selected vertex will be picked if the shape is identical to the shape that was used for rule definition, when the shape is different, the next nearest vertex in that search direction will be used.

Split tree operations

Split trees can be seen as a graph where nodes correspond to operations and the edges correspond to scopes, as each node has one scope as input and zero, one

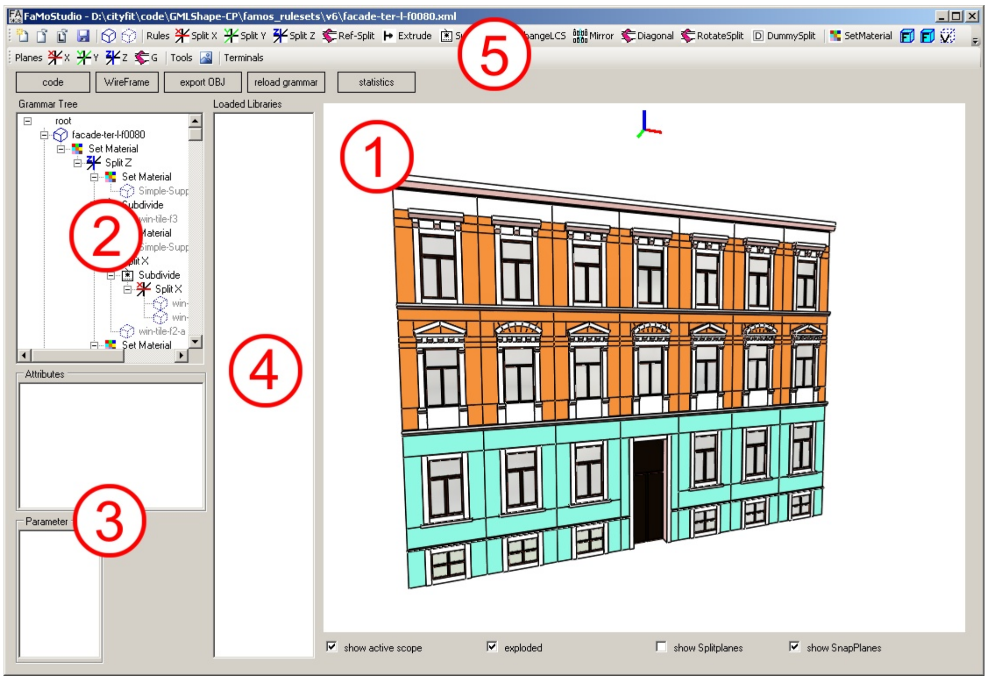


Figure 3.30: The main components of the user interface are: 3D-View (1), scope split hierarchy (2), attribute editor (3), a list of rules loaded from a library (4), and the toolbar buttons on top that provide access to the editing features (5).

or more child nodes as output. We conceptually group these operations into the following categories:

Interval Split. The generalized interval split defined in section 3.3 is used to partition the scope into child scopes. As axis parallel splits are often used, buttons for the X, Y and Z axis of the LCS are added for convenience. The number of children is inferred from the split interval.

Subdivide. This operation partitions a scope with straight splits along a split direction into equally sized parts. The number of children is not known beforehand, as the number depends on the size of the input scope. Therefore, subdivide has only one children node which is applied to all children when the rule is evaluated.

Extrude. Extrude allows to displace one of the planes that define the faces of a polyhedron along its normal direction. The user selects a face of the polyhedron the rule was defined on, the system generates code for selecting the face of the polyhedron with the most similar normal vector than the selected face.

Terminals. These types of operation consume the scope and have no child scopes. In the editor, 3 different types are used: F (*filled*), which fills a scope with a material (e.g. concrete), V (*void*), which corresponds to empty space and X, which marks the child as output node of the current rule. The terminal type AA allows

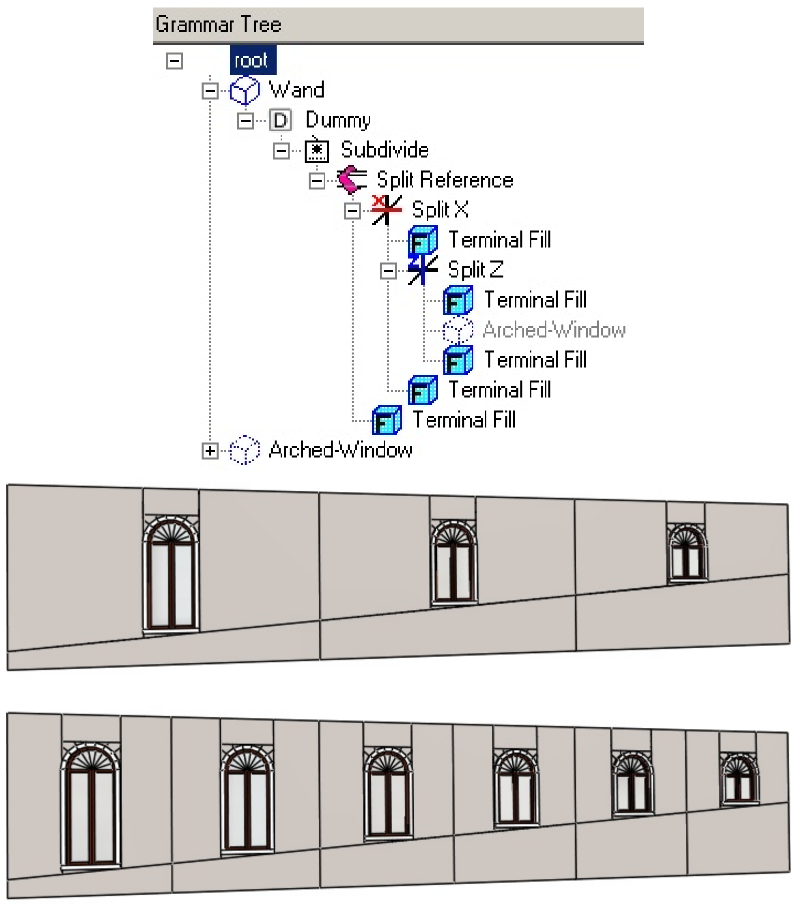


Figure 3.31: Modeling structure spanning multiple elements: The grammar tree (top) shows the steps used to create windows on a facade next to an inclined street. *Subdivide* is used to partition the scope into parts with similar width and each part is processed the same way (”window in wall”). By referencing a plane that was defined before the subdivide operation, we can create a parametrization that produces desired output when changing the width of the subdivided parts (bottom).

to call predefined parametric rules (e.g. such as shown in Figure 3.28).

Applying Rules. The terminal of the type corresponds to an output scope of this rule. In other words, X means "What happens to this part is to be specified later, each time the rule is applied". Applying a rule to scope yields as many children as X terminals inside that rule are defined. This allows defining more general rules with better reusability.

Modify attributes. The LCS and material attributes may be modified. E.g. the LCS may be changed by defining three points that correspond to the X direction, origin and Z direction of the new coordinate system. These points are identified via the vertex picking operation.

Scope dependent planes. A key feature for reusable rules was the abstraction of planes, that are dependent on the scopes shape, as attributes of that scope. Such attribute planes are inherited to children for later reference and use. These planes can either be defined using the interval split mechanism, or by picking 3 vertices of the input scope.

Reference Split. Scope dependent attribute planes can be reused for splits in any sub-level of the rule hierarchy, an example is shown in Figure 3.31.

Selection. Selection changes the visual appearance in the 3D view. If a node from a tree is selected, only this sub-tree is shown. The input scope of the node and its LCS is visualized in 3D; if the operation is a split its splitplanes are also visualized. Applying an operation replaces the currently selected sub-tree with that rule.

Drag & Drop. It was practical to use drag & drop for applying rules and operations. Dragging the root node of rule A to another node B replaces the sub-tree of B with an application call to A. Drag & Drop is also used to move or copy a node and its sub-tree.

Insert and delete. The insert operation inserts a *dummy* node (which performs no operation and yields the input scope as output), which can be replaced by any other operation. Deleting a node deletes also its dependencies (its sub-tree). Dummy nodes can also be used for attribute definitions if no operation is necessary.

Visualization and code generation

For visualization, the split tree is converted into a GML program that generates the 3D geometry. As noted in [81], the call graph of shape grammar rules defined as GML functions corresponds to a depth-first evaluation. In the same manner, the code is generated using a depth-first traversal of the split trees. Each referenceable rule tree is transformed into a GML function, and each instance rule is transformed into statements that will be executed when evaluating the generated code.

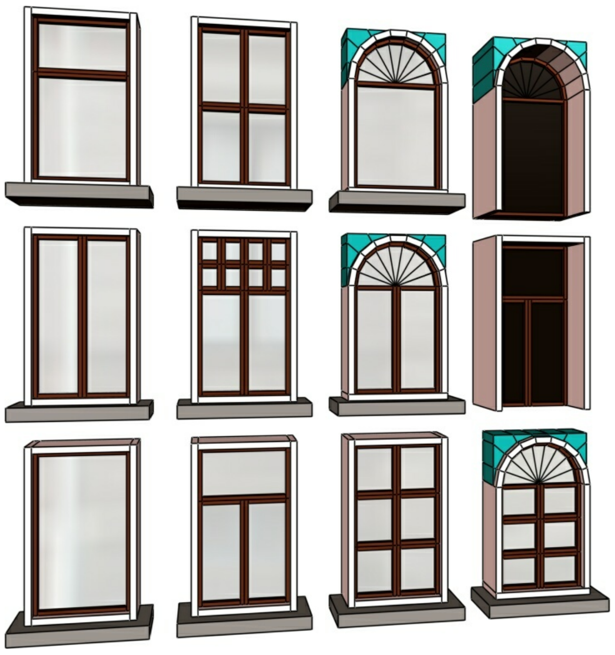


Figure 3.32: The parametric terminal rules can model a variety of different window and door styles.

Depending on the modeling operation and selection state, the generated code is altered to add visual aids for operations or selection (e.g. visualization of the currently selected volume). Callbacks are generated for pick-able objects in the 3D view, e.g. for selecting a face or a vertex.

The generated code is then sent to the GML interpreter plug-in [21] which parses and executes the code and renders the evaluated model in the 3D View.

Modeling procedure

The modeling prototype was evaluated by creating various models of façades based on real world examples. Some interesting cases were façades in neoclassical style (”Gründerzeit”), as those contain non-rectangular structures. We implemented the facade elements (doors, windows and gables) as scripted parametric terminal rules (cf. Fig. 3.32)

A façade is modeled in a top-down manner in the proposed modeler. The majority of façades that were modeled could be structured as follows: First, we split into horizontally aligned symmetric parts. Second, we vertically split into floors and ledges. Last, we split into horizontally aligned parts that contain façade terminal elements.

Reusing structure. We call the similarly structured façade elements **tiles.** Tiles are created with the parts to be filled in as X nodes. The tiles for a specific façade are created by assigning corresponding terminals, see also Figure 3.28 for some exemplary tiles.

3.7 Summary

In this Chapter, I first describe a novel shape grammar system using convex polyhedra in Section 3.2. This method allows to write shape rules that generalize to a larger group of shapes than box grammars. The next Section, (Section 3.3), describes common modeling and refinement operations that turned out to be useful in various scenarios. I then describe and discuss my proposed approach for fast and exact surface evaluation of convex polyhedra in the shape grammar system (Section 3.4). Afterwards, I describe a set of higher-level architectural construction rules that ease the construction of more complicated architectural objects. This method was used by the parse tree to model conversion described in the next Chapter (Section 4.1) and several others described in Chapter 5. Finally, I presented the standalone façade modeling application FaMoS (Section 3.6), that demonstrates direct interactive modeling using the proposed shape grammar system.

4 Inverse Generative Modeling

In this chapter I describe two approaches for utilizing generative descriptions in the context of shape reconstruction from measurements. Such model based reconstructions utilize an optimization step to create a final hypothesis that explains the measurements. A generative model can be used to describe a family of allowed or meaningful configurations, which can further be used to narrow down the search space of the optimization. The first approach encodes rules of classical architecture, i.e. repetition and symmetry, and uses methods from compiler construction to parse a given facade labeling, which yields an explanation of these labels favoring the architectural rules. It was published in *"Irregular lattices for complex shape grammar facade parsing"* [160] at the CVPR conference. The second approach utilizes scans of building interiors to create a hypothesis of electrical wiring below the walls, solely from the surface measurements. Under the assumption that the building was built using technical standards, a generative description that encodes these technical standards is used to restrict the search space to configurations that adhere to the norms and standards, from which a final hypothesis is extracted. The developed methods have been published amongst others in *"Automatic Texture and Orthophoto Generation from Registered Panoramic Views"* [106] and *"Data completion in building information management: electrical lines from range scans and photographs"* [105].

4.1 Inverse Generative Modeling of Building Façades

This Section describes a method that constructs a generative description of building façades from distance measurements and photographs. This work was done in collaboration with Hayko Riemenschneider and Wolfgang Thaller [160], and was published at CVPR [160]. My main contribution to this work was the implementation of parts of the parsing system and the transformation of parse trees to 3D models.

Pre-processing and Classification of Façade Elements

For the task of analyzing the structure of a façade, the first step is to partition the measurements into the relevant information for each façade. Since the structure

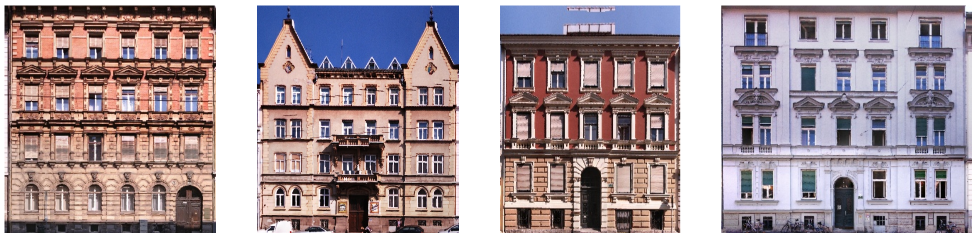


Figure 4.1: Ortho photos are extracted from the measurements. Classification of façade elements and structural parsing is carried out on this view of a facade.

is predominantly two-dimensional, a main plane for each façade is detected and an orthographic view of the façade, an ortho photo, is extracted.

In order to find positions of windows and doors, a computer vision algorithm has been trained to detect instances of windows and doors in the still pictures acquired by each camera. The window detection is difficult because of the intra-class appearance variations, and strong distortion and aspect ratio changes of the various camera orientations.

This classification approach is described in detail in [160] and [161]. The result is a pixel-wise probability function for each class; this function represents the probability of each pixel to belong to a semantic class, e.g. wall or window etc. - see also Figure 4.2. The merit function for the class terminal label is inspired by semantic scene labeling [170, 115]. A pixel-wise classifier is used on local image features giving the log-likelihood of each pixel x_i belonging to a class terminal $\psi \in \mathcal{L}$, as

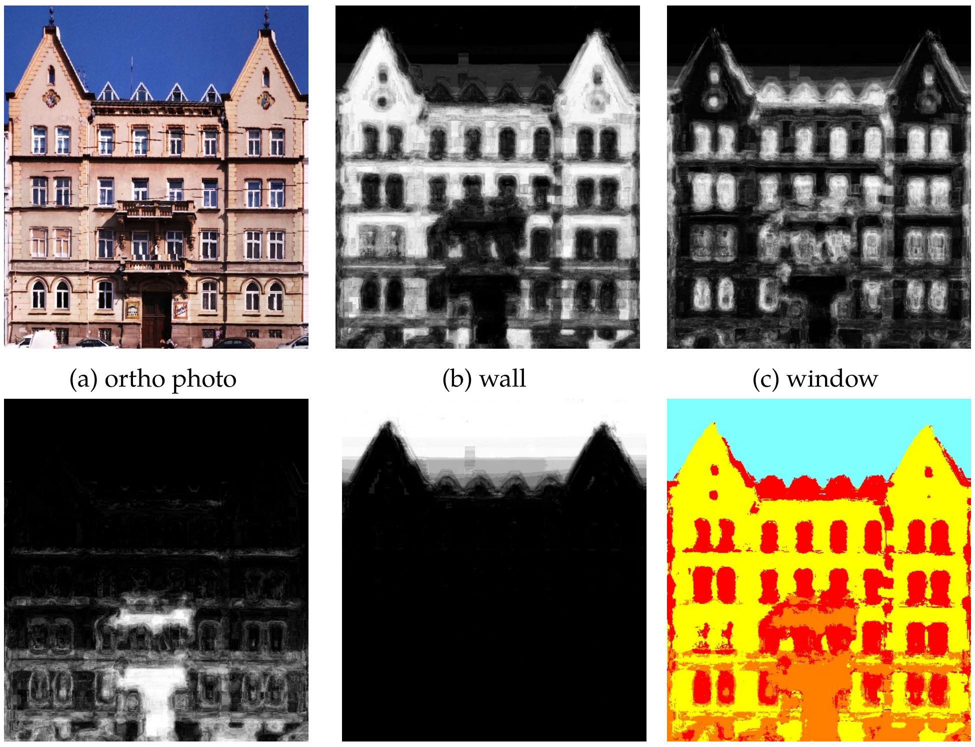
$$\Psi(x_i) = -log(P(\psi|i)) \tag{4.1}$$

Parsing Façade Structure

The approach uses a context-free grammar to partition rectified building facades into semantic image segmentations. The process involves training a local classifier for the desired terminal symbols and performing hierarchical splits to partition the image. This splitting process is guided by the low-level classifier probabilities and is designed to overcome the limitations of the local structure regularization.

Shape Grammar for Façade Parsing

The grammar for parsing the 2D facade structure and its relationship with the 2D image space are defined as follows. Following the definition of formal grammars in Equation (3.4), the two-dimensional split grammar $G = (N, T, P, S)$ is a context-free grammar, where the right-hand side of each rule consists either of



(a) ortho photo (b) wall (c) window

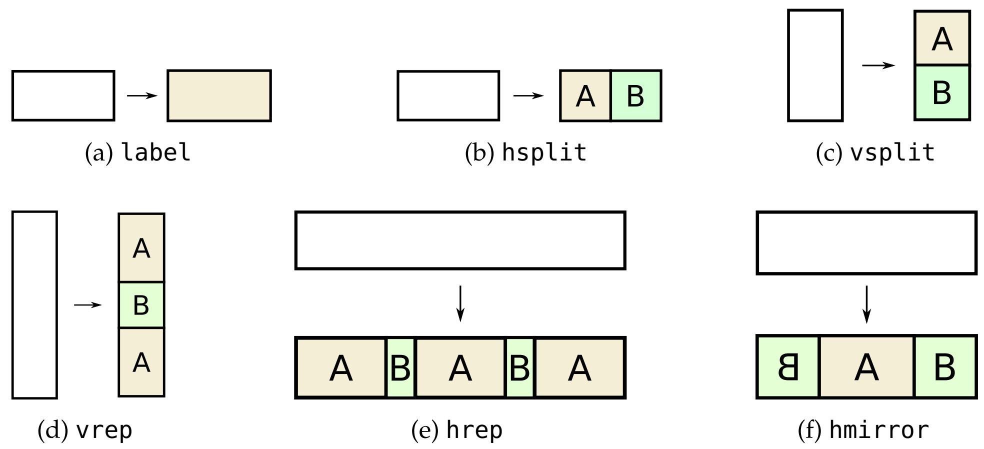
(d) door (e) sky (f) ML

Figure 4.2: The semantic labeling algorithm yields a probability value for each pixel of the input image (a) to belong to a specific semantic class. White corresponds to a high probability, and black corresponds to a low probability. The figures (b) – (e) show the probalbilities of their respecting semantic class, and the last image (f) shows a *maximum likelihood* (ML) labeling, where each pixel is assigned to the class with the highest probability value.

a single non-terminal symbol, or of a terminal symbol (an *operator*) followed by zero or more non-terminal symbols, where the number of non-terminal symbols matches the number of arguments expected by the operator.

A *parse tree* is an ordered tree whose interior nodes are labeled by non-terminal symbols, and whose leaves are labeled by terminal symbols (operators) from the grammar, and by an attribute a. The meaning of this attribute depends on the operator. The root of the tree is labeled by the grammar's starting symbol S. It follows from the restrictions we have imposed on the rules that for each interior node (non-terminal), its leftmost child is a leaf (an operator) and its other children are interior nodes (non-terminals) as well.

A parse tree is denoted by $d = op_a\, d_1 \ldots d_n$, and the set of all possible parse trees for a given grammar is denoted as $\mathcal{D}$. The number of arguments n is a constant for each operator op. The attribute value a is taken from a set $\mathcal{A}_{op}$ which



(a) `label` (b) `hsplit` (c) `vsplit` (d) `vrep` (e) `hrep` (f) `hmirror`

Figure 4.3: The 6 different split operators used by the grammar parsing system are the label operator, which assigns a terminal class to a given range (a), the horizontal (b) and vertical (c) split operators, vertical (d) and horizontal (e) alternating repeat operators, as well as the symmetric mirror around a middle column (f) operator. Note that alternating repeat and mirror operators include an additional size constraint: ranges associated to the same symbol (either `A` or `B` in the figure) have a similar size.

may be different for each operator. The sub-trees d_1 through d_n are themselves parse trees, but with different start symbols.

Operators and their Graphical Interpretation

A *range* is a rectangular area of an image that may or may not be mirrored around the y axis. Formally, a range r is a tuple (x_1, y_1, x_2, y_2) with $y_1 \leq y_2$. A point $p = (x, y)$ is written $p \in r$ iff $((x_1 \leq x < x_2) \vee (x_2 \leq x < x_1)) \wedge (y_1 \leq y < y_2)$. For the mirrored version of a range: $\bar{r} = (x_2, y_1, x_1, y_2)$.

To give a graphical interpretation of a parse tree d, a function $D(d, r)$ is defined which maps the parse tree to a labeling of the rectangular range r. This function can be defined recursively for each operator.

There is exactly one `label` operator for each label (wall, window, door, ...); a label operator takes no arguments and represents a rectangular image area belonging to that class. The remaining operators each split the range r they operate on into several sub-ranges r_i. The described method supports the standard horizontal and vertical split operators `hsplit` and `vsplit`, as well as horizontal and vertical *alternating repeat* and horizontal *mirror with center* operators. The mirror operator describes the common symmetry pattern `AB`$\bar{\texttt{A}}$. The alternating repeat operators are an indexed family of operators $alt(3)$, $alt(5)$, ... which describe patterns of the form `ABA`, `ABABA`, ..., respectively.

Each of these operators defines a set A_{op} of possible attribute values. For a standard split operator, the attribute value indicates the (relative) position of the split line between the two parts, for the mirror and the alternating repeat operators, it is the position of the split between the `A` and `B` parts. We could have defined a single alternating repeat operator whose attribute a also describes the number of repetitions. Our approach provides more flexibility in that it allows the grammar to express relations between the repetition counts in different parts of the facade. As we can establish reasonable upper bounds on the repetition counts, a variable repetition count can always be expressed by having one rule for each possible count.

The number m of sub-ranges is either equal to or greater than the number of arguments n. Each sub-range r_i is described by the sub-parse tree $d_{f(i)}$. The values of m, r_i and the function f may depend on the operator op, on the attribute a from the parse tree and range r.

$$D(op_a\, d_1 \dots d_n, r) = \bigcup_{i=1}^{m} D(d_{f(i)}, r_i) \tag{4.2}$$

where the set union operator is used to express the concatenation of labelings on adjacent ranges r_i.

Additionally, the method allows each non-terminal symbol in the grammar to be annotated with minimum and/or maximum sizes. Only parse trees are considered valid if the extent of the non-terminal falls within the allowed range.

Irregular Rectangular Lattice

A pixel-wise analysis of the whole façade image is not feasible, as the requirements for the parsing algorithm are too demanding in terms of run time and memory requirements. Therefore, to reduce the search space of the configuration, an irregular rectangular lattice is derived Such a lattice is a splitting of the orthogonal building frame into lattice tiles of varying width and height. Each tile, or several adjacent tiles, may represent a range of a terminal symbol, see also Figure 4.4. This lattice serves as an initialization for the inference process, and is defined by horizontal and vertical split lines. Each dimension solved independently, as the solution is not constrained to be regular. A split line is a transition which divides the image space into tiles, where neighboring tiles may belong to a different terminal label. Given the join distribution of the pixel-wise classifier for a tile, the resulting lattice marginalizes label transitions for each dimension. The actual lattice inference is described in detail in [160], and uses an energy minimization approach that is solved using graph cuts [25].

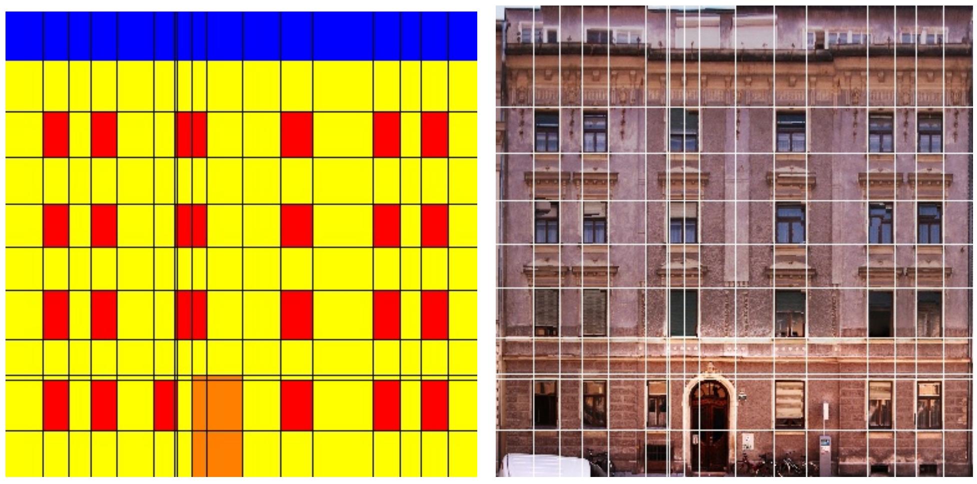


Figure 4.4: The irregular lattice (a) is a rectangular splitting of the image (b) into tiles of varying size, which depend on terminal symbols.

Grammar Matching

The task is now to match a grammar, a given set of production rules - that encode feasible facade structures - to a given lattice with associated class probabilities. The presented algorithm is similar to the standard Cocke-Younger-Kasami (CYK) parse algorithm for context-free grammars. It uses bottom-up parsing and dynamic programming. The modified CYK algorithm finds a solution by minimizing a cost function $c(d, I)$ over all possible labelings $d \in \mathcal{D}$:

$$d^* = \operatorname*{argmin}_{d \in \mathcal{D}} c(d, I), \tag{4.3}$$

where the set $\mathcal{D}$ denotes the set of all possible parse trees. Dynamic programming algorithms like CYK require the so-called *optimal substructure property*, i.e. it must be possible to efficiently calculate the optimal solution from the optimal solutions to its sub-problems. A sub-problem in our case means finding the optimal way of matching a given non-terminal against a given rectangular sub-range of the input. We write $q(I, r, S)$ to denote the optimal match for a non-terminal S on the sub-range r of input I.

Thus, given an operator *op* (that takes n arguments), an attribute value a and a range r, we can efficiently determine n sub-problems (r_i, S_i), such that

$$\min_{d_1 \ldots d_n} c(op_a\, d_1 \ldots d_n, I, r) = c(op_a\, d_1^* \ldots d_n^*, I, r), \tag{4.4}$$

where $d_i^* = q(I, r_i, S_i)$ are the optimal solutions for the sub-problems.

The optimal solution for (r, S) can thus be determined by calculating the costs for all rules applicable to S and for all possible values of a for the operator mentioned in each rule and choosing the minimum. This assumes that all sub-problems (r_i, S_i) have already been processed, which can easily be guaranteed by processing smaller ranges first.

Cost Function

A cost function is defined that implements a maximum a posteriori probability estimator: From the given grammar rule set, we get a prior probability distribution over all facades. The grammar is not yet stochastic, therefore this distribution is uniform for all segmentations that can be described by a parse tree, and zero for all “impossible” labelings. Thus, we maximize the posterior probability over the set of all parse trees $\mathcal{D}$ (rather than over the set of all possible labelings). The cost function is sought to be the log-likelihood of the parse tree:

$$c(d, I, r) = -\log P(D(d, r)|I). \tag{4.5}$$

In order to implement the estimator, the cost function has to fulfill the *optimal substructure condition* ((4.4)) and approximate this ”ideal” cost function. First, it is checked if any size constraints for the non-terminal symbol at the root of the current parse tree are violated. If so, the parse tree is assigned infinite cost. Otherwise, the matching proceeds according to the operator used at the root of the parse tree.

For label operators, c is calculated by summing up the pixel-wise merits Ψ for all pixels in the range r. For the other operators, we get (using Equation 4.2 and the assumption that the sub-trees d_i are statistically independent of each other):

$$\begin{aligned} c(op_a\, d_1 \ldots d_n, I, r) &= -\log \prod_{i=1}^{m} P(D(d_{f(i)}, r_i)|I) \\ &= \sum_{i=1}^{m} c(d_{f(i)}, I, r_i) \end{aligned} \tag{4.6}$$

In the case of the standard (vertical and horizontal) split operators, this results in $c(d_1, I, r_1) + c(d_2, I, r_2)$, which fulfills the optimal substructure condition (4.4).

In the general case, which includes the mirroring and alternating repetition operators, a cost function is acquired that violates the optimal substructure condition by matching the same sub-tree against multiple sub-ranges:

$$\begin{aligned} c(mir_a\, d_1 d_2, I, r) &= c(d_1, I, r_1) + c(d_2, I, r_2) \\ &\quad + c(d_1, I, \bar{r_3}) \end{aligned} \tag{4.7}$$

$$\begin{aligned} c(alt(m)_a\, d_1 d_2, I, r) &= \sum_{i=1,3\ldots}^{m} c(d_1, I, r_i) \\ &\quad + \sum_{i=2,4\ldots}^{m} c(d_2, I, r_i) \end{aligned} \tag{4.8}$$

Clearly, the d_i which minimize these costs are not necessarily the optimal solutions $d_i^* = q(I, r_i, S_i)$ on any of the sub-ranges. It is, however, a reasonable approximation to assume that they are.

Making that assumption still does not make the algorithm efficiently implementable. The cost of the optimal sub-solutions, $c(d_i^*, I, r_i)$, has already been calculated by previous iterations of the dynamic programming algorithm, but the costs $c(d_i^*, I, r_j)$ for $i \neq j$ have not. Therefore, these costs are estimated based on values that are more readily available, such as the costs $c(d_i^*, I, r_i)$ and $c(d_j^*, I, r_j)$, which are already pre-calculated.

Also, a dissimilarity estimate $\Delta(I, r_i, r_j)$ is introduced, which indicates the probability that two ranges should have the same labeling:

$$\Delta(I, r_i, r_j) := -logP(x_{r_i} = x_{r_j} | I) \tag{4.9}$$

This can be calculated from Ψ, again assuming statistical independence between the pixels.

The probability that two ranges can be explained by the same parse tree can be estimated using the probability that one of the ranges can be explained by the parse tree and the probability that the ranges have the same labeling:

$$\begin{aligned} & P(x_{r_i} = D(d_i^*, r_i) \wedge x_{r_j} = D(d_i^*, r_j) | I) \\ = {} & P(x_{r_i} = D(d_i^*, r_i) \wedge x_{r_i} = x_{r_j} | I) \\ \approx {} & P(x_{r_i} = D(d_i^*, r_i) | I) P(x_{r_i} = x_{r_j} | I) \end{aligned} \tag{4.10}$$

Taking this estimation together with the fact that $c(d_i^*, x, r_j) \geq c(d_j^*, x, r_j)$, the result is

$$c(d_i^*, x, r_j) \approx \max(c(d_j^*, x, r_j), \Delta(x, r_i, r_j)). \tag{4.11}$$

Inference on the Lattice

For the basic horizontal and vertical split operators, the irregular lattice amounts to limiting the choices for the attribute values a at the various parse tree nodes.

The result is equivalent to maximizing posterior probability under the assumption that each lattice tile has a homogeneous label.

For mirror and repeat operators, the situation is slightly more complicated. In an application of the mirror operator, if the sub-range r_1 is exactly representable on the lattice, the sub-range $\bar{r_3}$ that is its mirror image might not be, and vice versa. Likewise, the various sub-ranges r_i of a repetition operator will usually not fit exactly.

The sub-problem solutions d_i^* are therefore calculated on the closest range actually available in the lattice, and then used as an approximation for the exact range. This has the effect of adding the split lines required by the symmetries and repetitions to the output labeling, even if they have not been found during the lattice generation phase. The dissimilarity Δ is defined on ranges of different sizes to be the Δ between the smaller range and the corresponding sub-range in the center of the larger range, plus an extra penalty value proportional to the difference in areas.

A complete example of a parsed façade structure of the example façade shown in Figure 4.2 can be seen in Figure 4.5.

Parse Tree to Model Transformation

The parse tree corresponds to a coarse to fine refinement of façade structure, which partitions the façade into semantic elements by applying operators. Therefore, such a parse tree can be naturally converted back to program source code, similar to the code generation step in a conventional compiler pipeline.

In this subsection, I describe a transformation from façade parse trees to *GML*, the *G*enerative *M*odeling *L*anguage, a concatenative programming language with a syntax similar to postscript. The method utilizes the robust generative shape composition method described in Chapter 3. The general approach consists of the following code generation steps:

1. For each façade, a volume will be generated that corresponds to the maximum bounds of the façade space. Each volume will be refined starting with the root of the according parse tree.

2. Each parse tree is converted into textual generative representation of the operations that are applied to the façade bounding volume.

3. Each terminal symbol is associated to a predefined parametric rule such as the parametric windows or doors described in Section 3.5. The according rule application is generated for each terminal.

Coordinate System. The geometric operations are carried out in a local façade coordinate system that has the lower left point as origin. When looking from front

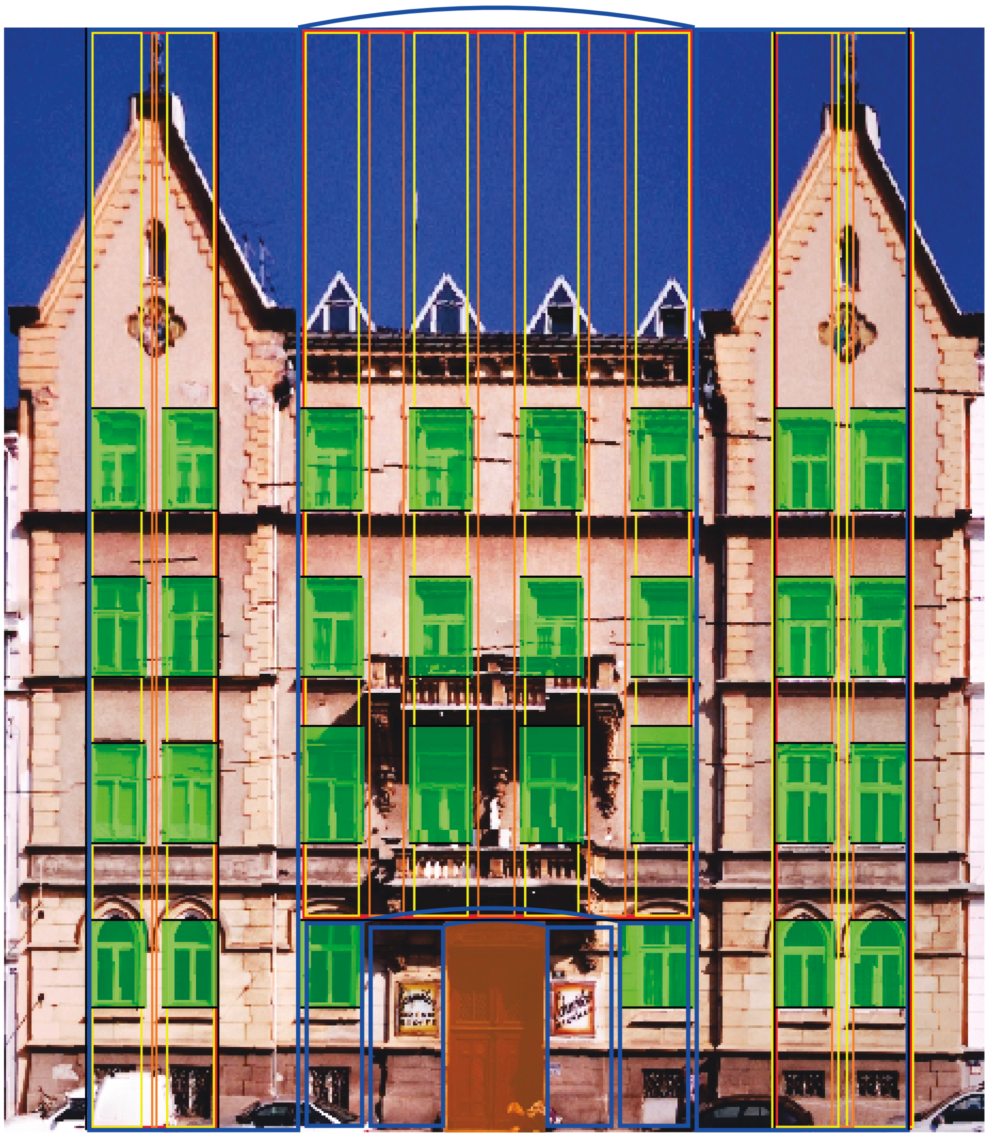


Figure 4.5: Parsed structure of a façade. Detected alternating repetitions of columns are shown as alternating rectangles of orange and yellow, symmetric side columns between a middle column are shown as blue rectangles (symmetric side columns) that are connected by a blue arc line. The result shows nested detected symmetries, such as repetitions inside a symmetric side column (left and right part). Also, the door has been parsed as a middle part that is surrounded by symmetric side columns.

at the façade, the x axis is pointing to the right, the y-axis points along the view direction and the z-axis points up. For each façade, a transformation is specified that positions the façade in a global world coordinate system.

Short GML Primer. The language GML is similar to Adobes PostScript, and was developed by Sven Havemann for generative mesh modeling; I will give a brief overview of the most important concepts, for more details see [70] for an extensive description of the language and the features. It is a stack oriented, concatenative [76] programming language. A GML program consists of a stream of tokens; operators take their arguments from the stack and push their result onto the stack. For example, the program `19 23 add` will first push `19` on the stack, then push `23` on the stack, the operator `add` will take the two arguments from the stack and push the result (`42`) on the stack. Variables and names are organized using key - value stores, so-called *dictionaries*. In addition to the standard stack, GML also maintains a dictionary stack that contains dictionaries only. A *name* in GML is just put on the stack when prefixed with a slash (e.g. `'/FOO'`), a name without prefix retrieves the key with that name from the first dictionary on the dictionary stack that contains the key. By creating a new dictionary on the dictionary stack, a program can create a new namespace for variable names. Functions or methods are just arrays of tokens, enclosed by curly brackets (`{, }`), and thus can be stored in variables as well. The operator `DEF` takes a name and a value and stores the key-value pair in the topmost dictionary on the dictionary stack.

Parse Tree to GML Transformation. I assume that operations on parse trees are side effect free and have a convex polyhedron as input and yield one or more convex polyhedra as output, and that the correct order of children nodes is maintained. Under this assumption, parse trees can easily be transformed to GML code using a standard tree traversal and generate code while traversing, as is standard in a conventional compiler tool chain. Code generation is especially easy as GML is a concatenative language and intermediate results are automatically put and taken from the stack, which means that call parameters are implicitly handled. In general, when converting parse trees between languages, the operations available in the source language have to be mapped to equivalent operations in the target language. If a directly equivalent operation is not available, code needs to be generated that expresses the original source operation via available target operations.

Basic GML operations. The available target operations in GML correspond to the modeling operations described in the previous chapter in Section 3.3.

The relevant operations for code generation of parse trees are:

planes – modeling operations are expressed in the local coordinate system described above, each basic orthogonal modeling axis (corresponding to the $\vec{x}$, $\vec{y}$ and $\vec{z}$-axis) is described by a function that retrieves the according plane in the global coordinate system for splitting. The names are `CPX`, `CPY` and

`CPZ` with the plane normal pointing in the according axis direction. Functions for inverse directions (to swap children order) have a `N` added to their name, e.g. `CPXN`. By using these names in rules for scope split operations, the symmetry operator can easily reuse a parse tree on a mirrored scope by switching the according plane pair. In my implementation this is realized by making e.g. `CPX` and `CPXN` functions that return the according plane from the current scope state. The mirror operation just reverses the direction of that plane of the scope state.

split – the implementation of the generalized interval split, described in the previous Section 3.3. The GML syntax is `split-interval`, and it expects its parameters on the stack, pushed in this order: a scope dictionary that contains all scope state, the absolute/relative distance interval, and a splitting plane (which will be moved along its normal direction). alternating splits are also built using the generalized interval split.

mirror – the `mirror-h` operator assumes the following parameters, pushed in order: a scope dictionary that contains all scope state, an absolute or relative distance interval that partitions the scope into exactly 3 parts. This operation will split along the `CPX` axis using the given interval, which yields 3 parts $AB\overline{A}$, where $\overline{A}$ is the horizontally mirrored version of A. The operator will switch the according scope state plane of $\overline{A}$ and leaves the scope on the stack in the order $B\overline{A}A$, ready for consumption by the following operations.

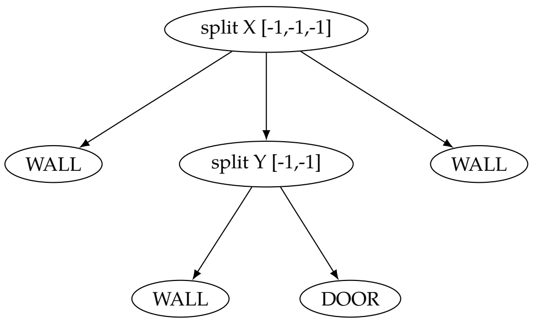
I give an example of a possible transformation of the three steps described above; the first part of the program is initialization and the definition of functions that correspond to terminals and their respective parametric façade elements:

```
ShapeGrammar.Tools.init      % prepare scope split methods
dict begin                   % create namespace for terminals

/t101 { 3 0 0.05 0.03 AA.window.rect } def        % rectangular window
/t102 { 9 2 0 0.05 0.03 AA.window.arched } def    % arch window
```

Initialization. This example code first initializes the shape grammar state and defines the appearance for terminal symbols, exemplary shown for the terminals `t101` and `t102`. Each terminal corresponds to a pre-trained class that was detected in the measurements, e.g. wall or different window styles, such as rounded or square windows. Each parametric terminal is assigned values for the parameters, e.g. the thickness of a frame in centimeters and so on. Then, each façade is initialized using its spatial extend, assuming a fixed façade depth (e.g. 1m):

```
(0,0,0) (23,1,25.9) 3 scope-box
```

This code creates a box-shaped convex polyhedron on the stack, corresponding to a façade with 23m width, 25.9m height and 1.0m depth. The depth is just



```
X [-1, -1, -1] split-interval
  WALL
  Y [-1, -1] split-interval
    WALL
    DOOR
  WALL
```

Figure 4.6: A parse tree (left) that contains operations in nodes and terminal symbols as leaves is converted to a GML program (right) via a pre-order depth-first tree traversal.

an arbitrary fixed constant at this point, in future work the façade parse tree could also be applied to the side wall scope which is part of a larger volumetric complete house model.

Parse tree to GML Code. The corresponding GML code to a façade parse tree is then generated by a pre-order depth-first tree traversal, see also Figure 4.6 for a simple example. Code for split operations is simply generated when visiting the according split node, each split in the proposed grammar consumes one scope and leaves two scopes on the stack; the code generation for alternating repetition and mirror is slightly more elaborate.

The parse tree node for alternate repeat operators (`hrep` and `vrep`) contains only two children A and B with their respective column split widths, as described in Section 4.1. The generated code for e.g. the sequence $ABABA$ will first perform a single split with the width of A, and generate the code for the A sub-tree that is applied to this first A in the sequence. Then, a loop is created that splits the remaining scope in BA blocks; each block is then split to separate B and A. Then, the code for each sub-tree B and A is created inside the loop.

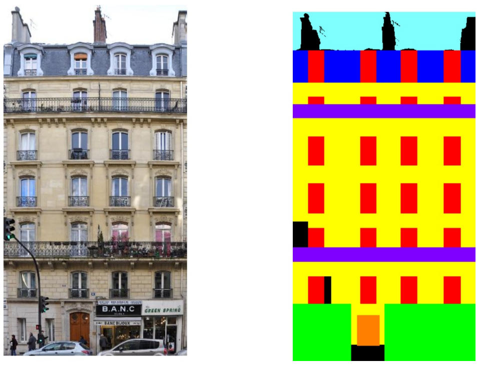
The `hmirror` operator (see also Figure 4.3) also contains two children S and M (for side and middle columns). For this node, code is generated that applies the GML method mirror-h, then generate a code block for the sub-tree of S that is looped two times for the left and right column, and generate the code block for the sub-tree of M for the middle column.

The complete code of the transformed parse tree for the façade shown in Figure 4.5 is shown in the appendix Section 8.3.

Experimental Evaluation

An experimental evaluation was carried out using a 30 image data set of Hausmannian style against the approach of Teboul et al. [186, 185] and a data set of

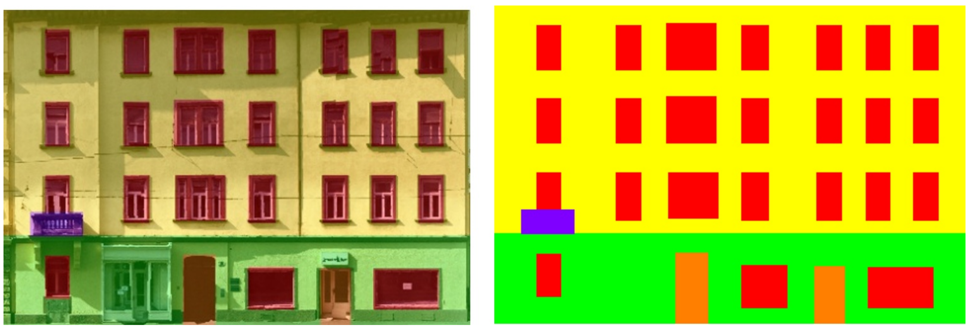
50 neoclassical façades of Graz was evaluated against manually labelled ground truth.



(a)

	MAP	[186]	[185]	Our
Window	29	81	81	68
Wall	63	83	84	87
Balcony	42	72	63	69
Door	90	71	84	56
Roof	62	80	86	83
Sky	95	94	94	95
Shop	26	95	97	97
Average	58	82	84	80

(b)



(c)

Method	Window	Wall	Door	Sky	Global	Class	IoU
MAP	60	66	57	80	66	65	43
Our	60	84	41	91	78	69	58

(d)

Figure 4.7: An experimental evaluation was carried out using two data sets: the *Paris2010* data set [186, 185], an example can be seen in (a), and the *Graz50* data set, of which an example is shown in (c). The proposed method produces comparable results to the state of the art (b) and increases the classification result of a simple MAP combination of the detection results (d).

4.2 Inverse Generative Modeling of Electrical Wiring in Building Interiors

The second approach concerning inverse generative modeling I present in this thesis is the creation of a hypothesis of the placement of electrical wiring inside buildings from unstructured data. This information can be useful for renovation or rebuilding projects, or in the case of urban mining situations. Urban mining denotes the method of detecting, removing and handling re-usable resources when a building is deconstructed. The estimation of electrical wiring is a valuable asset for such situations [152]. The information acquired by the proposed method can be used to determine the amount of resources, in this case copper, that could be salvaged.

Typically, such information can be acquired by using metal detectors, but the scanning and documentation process is demanding in terms of resources and time. On the other hand, a domain expert that considers technical standards and construction practices is able to give an educated guess of the placement of electrical wiring just by looking at a wall. The proposed system mimics such expert knowledge using a 2D grammar that creates electrical installation zones, given the coarse reconstructed room geometry as input.

The technique was evaluated by building a complete pipeline that creates a hypothesis from measured unstructured data. Our prototype implementation is called **RISE** - short for **R**eveal **I**nvisible **S**tructur**E**s.

Overview

The emphasis that was put into the system was to be as automated as possible, while being general and broadly applicable. The main idea is to train a system to detect the appearance of observable end points of electrical wiring, i.e. sockets or switches, and create a hypothesis of electrical wiring under the assumption that the installation follows regulations or technical standards that describe preferred for zones the installation of electrical wiring. However, technical standards may change over time, or differ between countries, therefore this information is represented by an exchangeable set of rules.

The complete task of wiring hypothesis has been subdivided into smaller subproblems, which leads to a pipeline architecture. The complete pipeline is shown in Figure 4.8. The following subsections describe the most important parts of the proposed approach. The input to the pipeline consists of distance measurements (point clouds in E57 format [83]) and color information (panoramic images), and is described in more detail in Section 4.2: a coarse geometric description, a floor plan representation, is extracted from the point clouds in the data pre-processing stage, see Section 4.2. Furthermore, an image per wall is created, that contains the color information of this wall (an ortho photo) of known scale. In these ima-

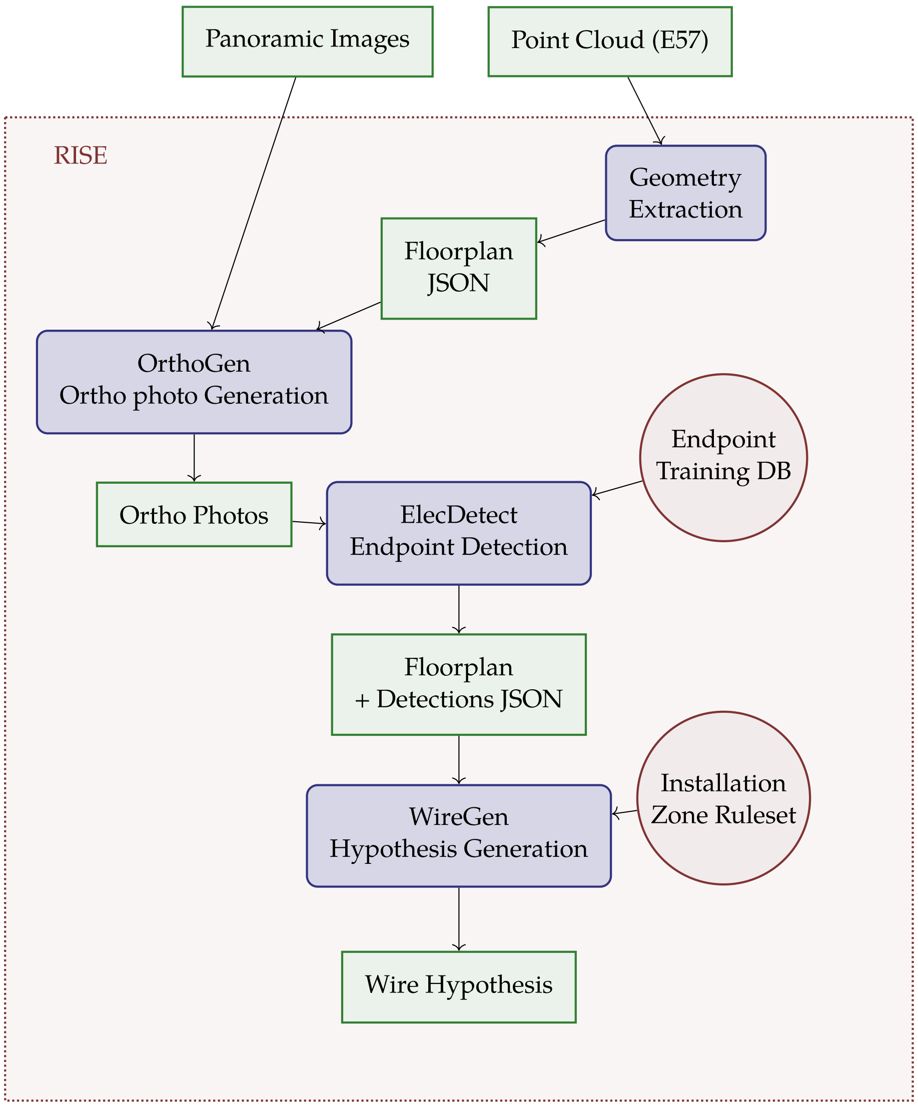

Panoramic Images
Point Cloud (E57)
RISE
Geometry Extraction
Floorplan JSON
OrthoGen Ortho photo Generation
Endpoint Training DB
Ortho Photos
ElecDetect Endpoint Detection
Floorplan + Detections JSON
Installation Zone Ruleset
WireGen Hypothesis Generation
Wire Hypothesis

Figure 4.8: Overall Architecture of the RISE pipeline. Modules (processes) correspond to blue nodes with rounded corners, and data (e.g. intermediate results) corresponds to green nodes. Circles depict data that represents prior knowledge.

ges, instances of visible endpoints are detected using methods of computer vision. Section 4.2 describes the method of encoding the prior knowledge from technical standards in an installation zone grammar, and the optimization approach that synthesizes a probable wiring, given the floor layout, detected endpoints and installation zones.

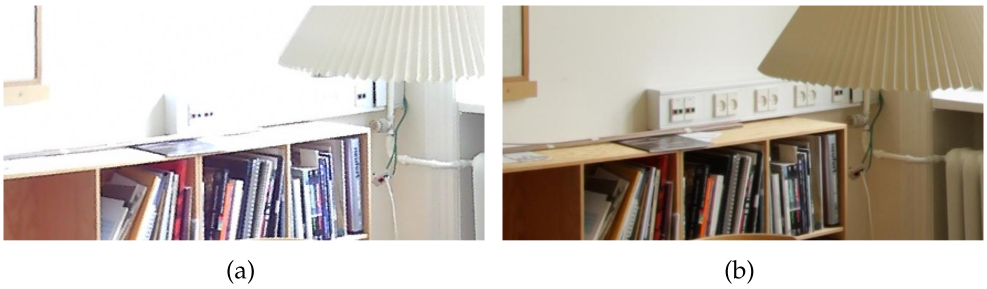


Figure 4.9: The 3D scanning unit used for acquisition did not capture important features that are necessary for classification. Furthermore, the images are easily over- or underexposed when lighting conditions are not optimal (a). Therefore, a manually taken image had to be used that yields higher resolution (b).

Data Acquisition

The data that was used to evaluate the pipeline has been acquired using a Faro Focus 3D [1] scanning device in E57 format. This scanner also acquires color information by stitching a spherical panoramic image, i.e. an equirectangular projection of a surrounding sphere, where each pixel corresponds to one distance measurement; this image can be used to assign colors to points to acquire a colored point cloud. The initial idea was to utilize this color information for the detection of sockets and switches, but it turned out that the quality was too low for sufficient recognition of endpoints, as is shown in Figure 4.9. Therefore, a high resolution panoramic image (HRPI) was acquired at the scanning position. A Nodal Ninja 3 panoramic head was mounted on the scanner's tripod for the acquisition, the HRPIs were stitched from pictures taken with a Canon 500D DSLR with a fish eye lens, see also Figure 4.10.

Data Pre-processing

The data is processed to reconstruct a floor plan representation using the method of Ochmann et al., which creates a BIM model from unstructured point cloud data, and was developed in the DURAARK project. First, the point cloud acquired from multiple scans is segmented into rooms [142], afterwards, a floor plan representation is generated from the segmented point cloud [141], see also Figure 4.11.

The result that is processed further in the RISE pipeline is an augmented floor plan representation that contains the connectivity of rooms, walls, and openings inside walls, like doors and windows. This representation has the underlying assumption that a building structure can be represented by a stack of two-dimensional structures, this typically corresponds to the floors of a building. Such

[1]`http://www.faro.com/`

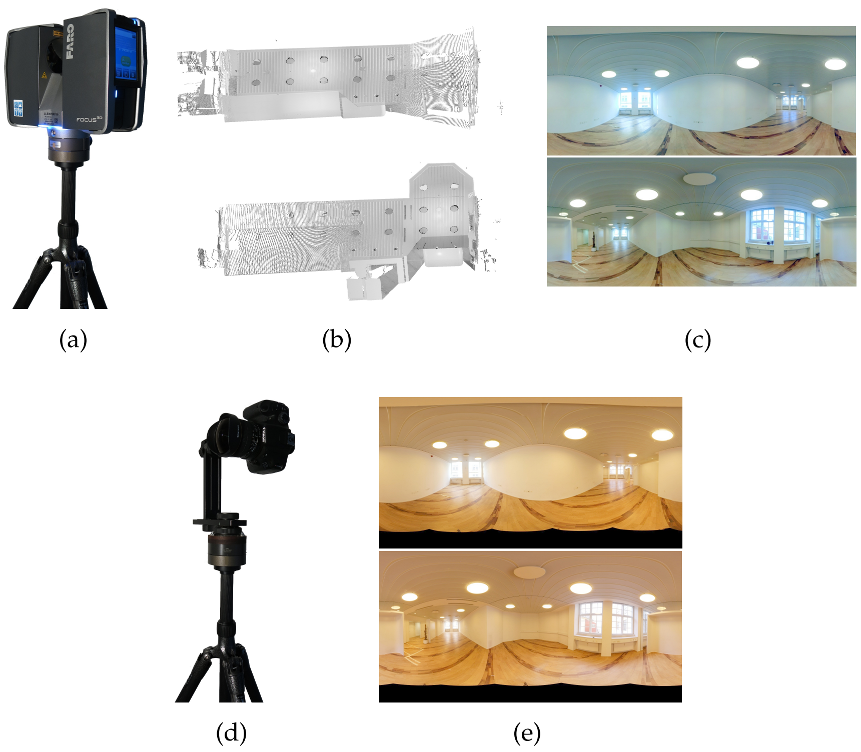


Figure 4.10: A Faro Focus 3D scanning device (a) was used to acquire a point cloud that consists of two scans (b). The scanner also acquires color information per scan (c). Additional high resolution panoramic images (e) were acquired using a panoramic head and a digital camera (d).

a two-dimensional structure has the basic underlying structure of an arrangement [3]. The RISE pipeline represents this arrangement structure by JSON format that contains a list of walls and a lists of elements that are associated to the walls, such as openings, or detections. See Figure 4.12 for an explanation of the floor plan encoding, each wall is specified in relation to a global world coordinate system. The elements that are associated to a wall are encoded in a two-dimensional wall coordinate system, as can be seen in Figure 4.13.

Pre-processing and Classification

OrthoGen

The OrthoGen module is responsible to extract the measured color information for each wall, the ortho photo, using the acquired panoramic images and the above described floor plan format, see also Figure 4.15.

Each panoramic image corresponds to a sphere that captures incoming light at the scanning position. The image uses the pose of the scanner; Manually acqui-

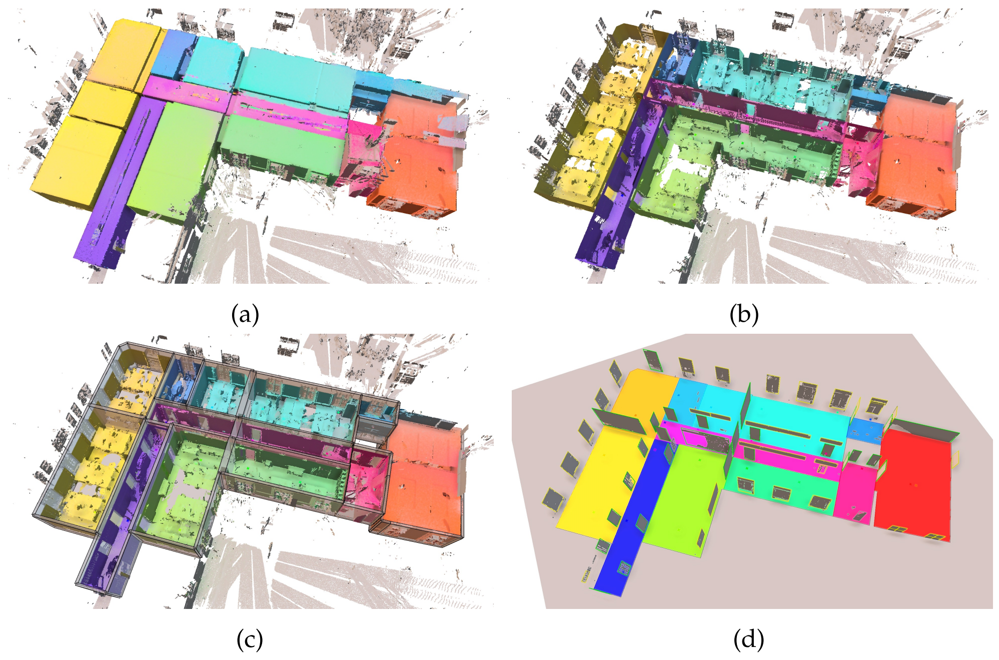
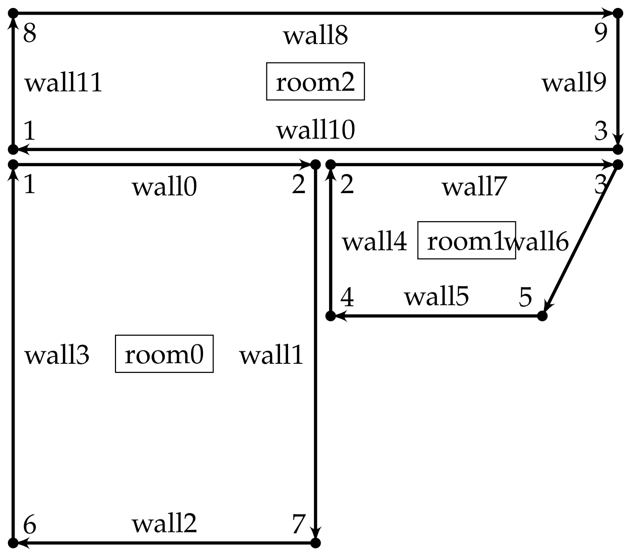


Figure 4.11: The proposed pipeline uses the approach presented by Ochmann et al. [141] to create an augmented floor plan representation from unstructured data. A point cloud is acquired using a laser scanning device (a) (the ceiling points have been removed for the images (b)-(d) for better visibility). The point clouds are segmented in rooms (b), as is shown by different colors. Then, the position of walls, and their height and thickness are detected (c). For each wall, openings, such as windows or doors, are detected and stored in the resulting IFC model and the augmented floor plan representation.

red images may, although taken at the scanning position, additionally need to be aligned along the azimuth (up) axis. Typically, the solution for such an image registration task is based on feature detection and matching [182], but in this constrained case a much simpler method was sufficient. First, both images are converted to a normalized gray-scale version; Normalization is carried out by subtracting the gray-scale mean and divide by the standard gray-scale deviation of the image. Both images are re-scaled to the same size. The alignment is resolved by an exhaustive test of all possible alignments at every horizontal pixel position. The minimal sum of absolute differences (SAD) determines the final alignment.

The color information is then projected on the proxy geometry, similar to texture mapping. Therefore, an error naturally arises where the proxy geometry does not correspond to the real geometry, e.g. fine details or furniture that are not represented by the proxy geometry. Typically, this error does not pose a problem for the proposed pipeline, as the resulting images are used to detect the appearances

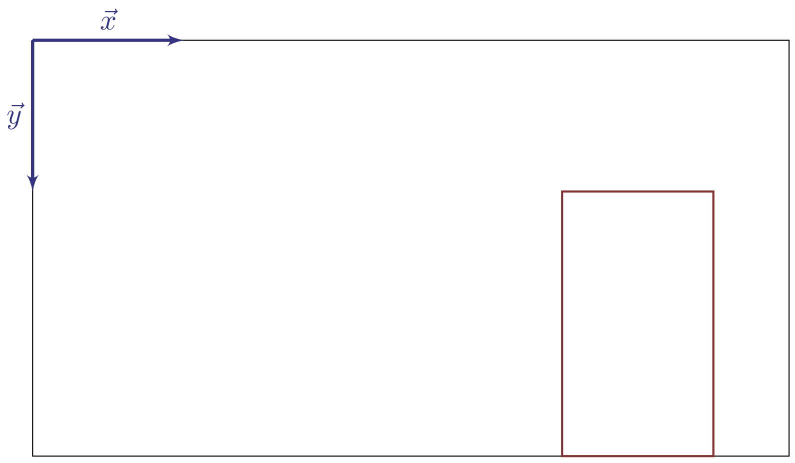


(a) The floor plan representation encodes rooms as oriented wall cycles, in counterclockwise order. For example, room0 consists of the wall cycle wall0-wall1-wall2-wall3. T-Vertices are encoded as *crosslinks*, e.g. between wall1 and wall5.

```
{
    "label": "WALL",
    "attributes": {
        "id": "wall1",
        "left": 0,
        "top": 0,
        "width": 6156.679,
        "height": 3182.134,
        "thickness": 200,
        "origin": [ 7887.29, 1989.9, 11840.8 ],
        "x": [ −0.787, 0.617, 0 ],
        "y": [ 0, 0, −1 ],
        "roomid": "room0"
    },
    "left": 2,
    "right": 7,
    "crosslink": [ "wall5" ]
}
```

(b) The wall cycle is encoded by specifying the id of the adjacent corner points, when looking at the wall from inside the room.

Figure 4.12: Description of the floor plan exchange format.



(a) Positional information of a wall element (e.g. a door) is encoded by an axis aligned bounding box with attribute names *left*,*right*,*width*,*height* that is related to a wall coordinate system with origin on top left, as seen from inside the room (all measurements in mm).

```
{
    "label": "DOOR",
    "attributes": {
        "left": 3856.68,
        "top": 432.13,
        "width": 1250,
        "height": 2650,
        "wallid": "wall0"
    }
},
```

(b) Each element is represented by a category label and a set of attributes that describe the location and context of the element.

Figure 4.13: Encoding of elements that are associated to a wall.

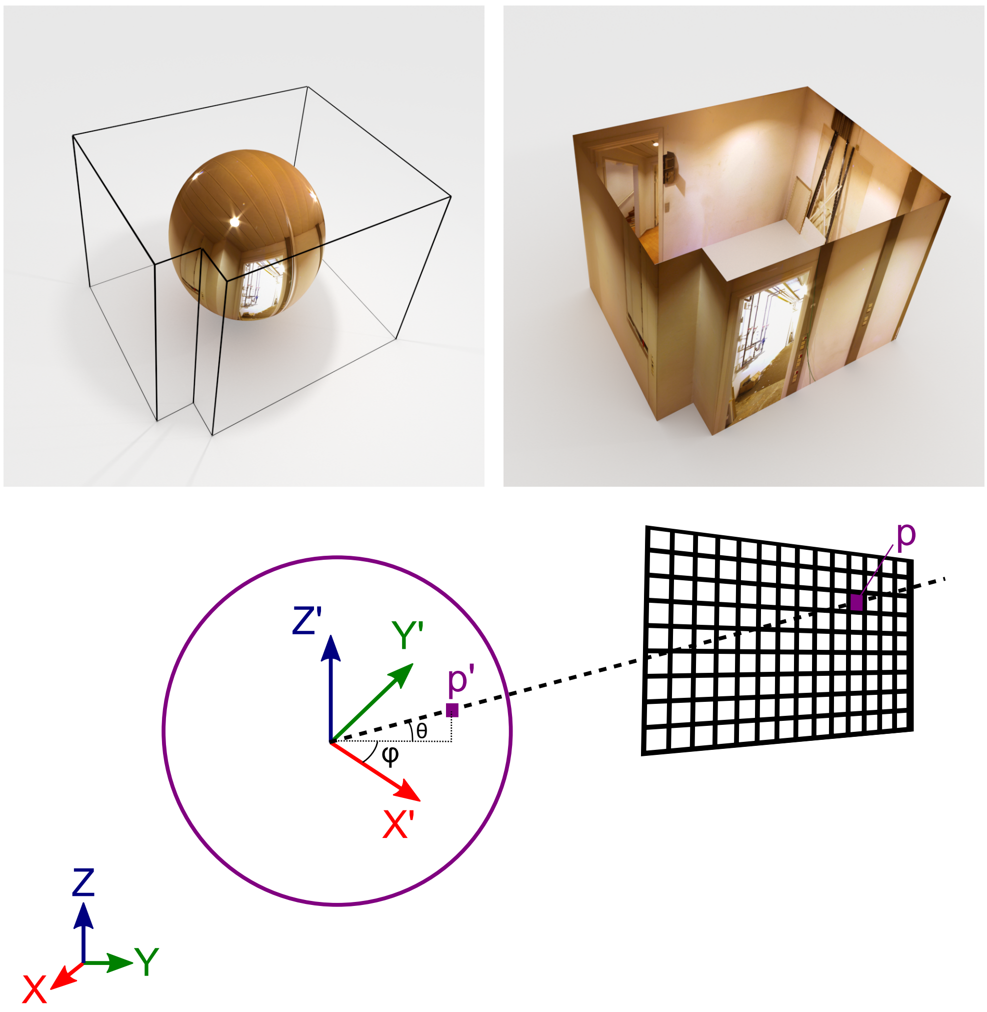


Figure 4.14: Ortho photo Projection: A rectangular patch in 3D space is sampled at a desired resolution, e.g. 1 pixel/mm. Each pixel p is transformed into the local coordinate frame (X', Y', Z') of the panoramic sphere in spherical coordinate angles azimuth ϕ and elevation θ to determine the color value in the panoramic image.

of sockets and switches, which are more likely to reside in the walls main surface plane.

Each pixel of the ortho photo corresponds to a 3D position of the wall rectangle in world coordinates. The projection is carried out by using a simple projection approach which is shown in Figure 4.14: For each pixel p of the patch, the system creates a ray from the corresponding 3D position to the center of the panoramic sphere. The intersection p' of the ray and the sphere is then transformed into the local spherical coordinate system of the panoramic sphere, which yields the azimuth angle ϕ and the elevation angle θ. These angles are used to acquire the color value of this pixel from the panoramic photograph. The ortho photos are sampled such that the distance between two neighboring pixels corresponds to 1mm. If a room consists of several scans, the sphere with shorter distance to the pixel world coordinates is chosen. The complete approach is described in [106].

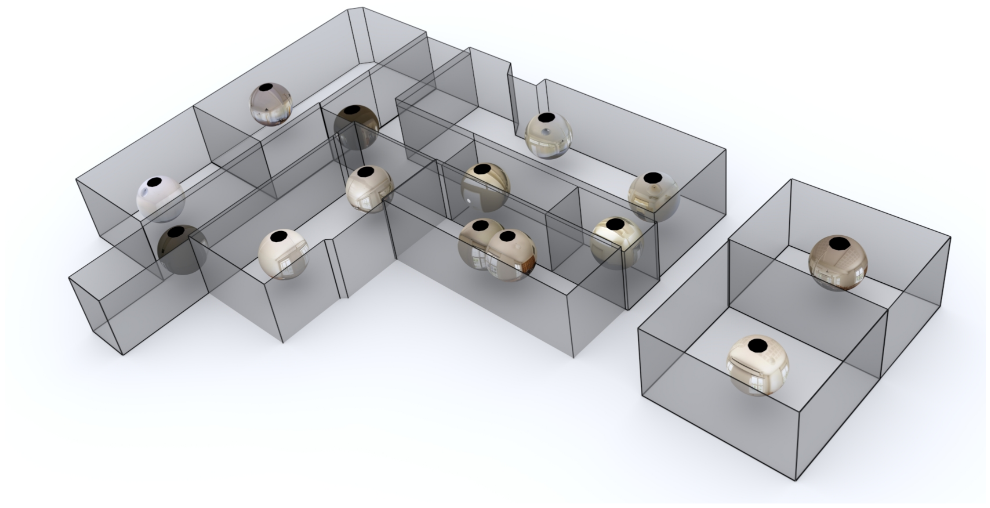
The ortho photo generation has also been used in contexts where a detailed floor plan description was not available, but instead a rough proxy geometry consisting of an indexed face set of either triangles or quads. Input geometry, even if coarse, commonly contains many small features or triangles. Therefore, the idea is to group as many as possible together. My method synthesizes rectangular patches by a fine-to coarse clustering approach using the mean shift algorithm [36]. It works in two principal steps: first, all similarly oriented triangles are grouped. Second, similarly oriented triangles that lie in the same plane are grouped. From these groups, the smallest rectangular patch is found within the plane of each triangle group. All patches are coherently oriented regarding the up direction.

The color information of each patch is obtained by projection: each patch is sampled for every pixel position, with a desired resolution, e.g. 1mm/pixel. For each pixel, the nearest scan – that is not occluded by the input geometry – is chosen. The color is obtained by intersecting the panoramic sphere with the ray from the pixel patch position in world coordinates to the center of scanner's position. Thus, an image for each patch is acquired. Note that due to this simple projection approach, the projected color will only correspond to the real value if the measured surface point lies in the detected plane of the patch, otherwise the projection will be distorted. This is especially noticeable with smaller details that were measured but are not present in the proxy geometry, such as furniture. To use the patches as texture information for the proxy geometry, I perform a coordinate system conversion from the world coordinate system into patch coordinates, where patch coordinate X and Y axis correspond to the image X and Y axis, and Z corresponds to the normal direction of the patch. I project each triangle into its corresponding patch cluster plane, and obtain texture coordinates directly from patch coordinates, normalized to the patch size in world coordinates. An example of ortho photo extraction can be seen in Figure 4.16 and Figure 4.17.

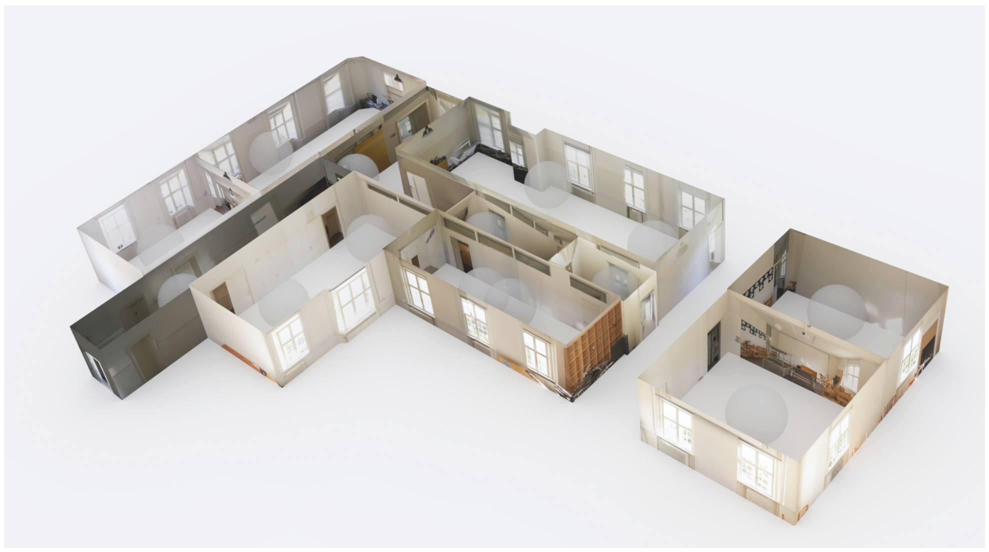
ElecDetect

The automatic detection of sockets gained early attention in the field of robotics, in order to build self-charging robots [32], [127]. The chosen approach follows a classical object detection methodology from computer vision: For each pixel p of an ortho photo, a small image patch P centered around p is analyzed to determine the detection probability of an object class for this pixel.

Instances of power sockets and light switches can be challenging to classify:

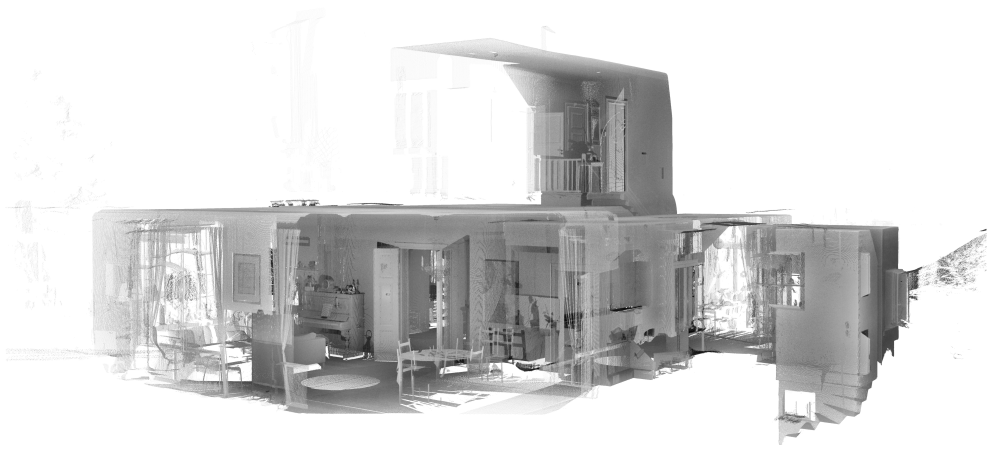


(a)

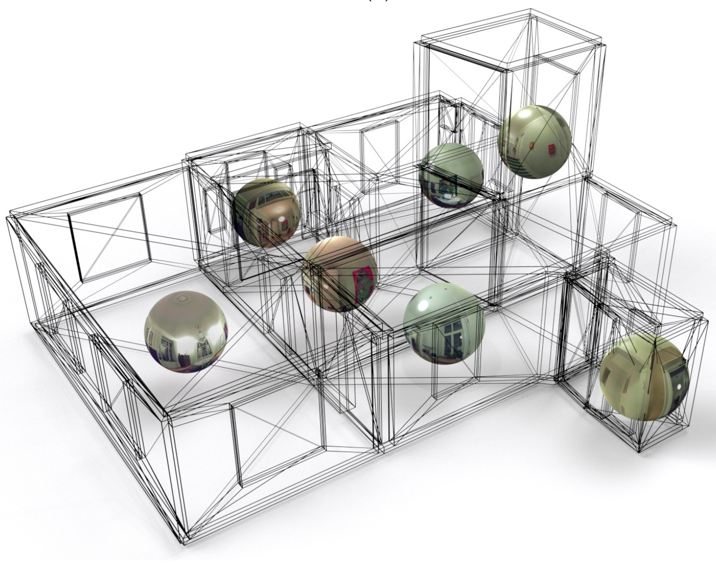


(b)

Figure 4.15: The ortho photo generation module projects the acquired color information, represented by the spheres in (a), on the coarse geometry that has been reconstructed from the point cloud (b).

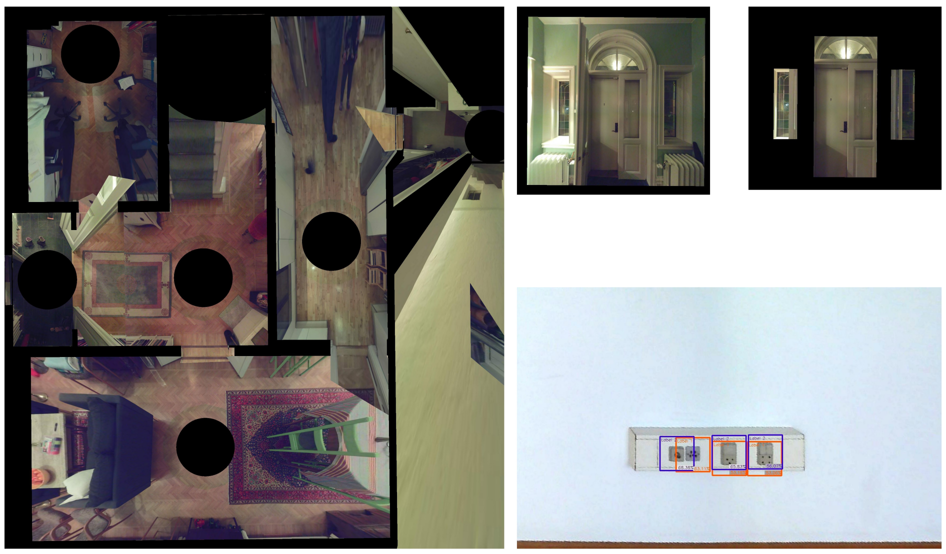


(a)

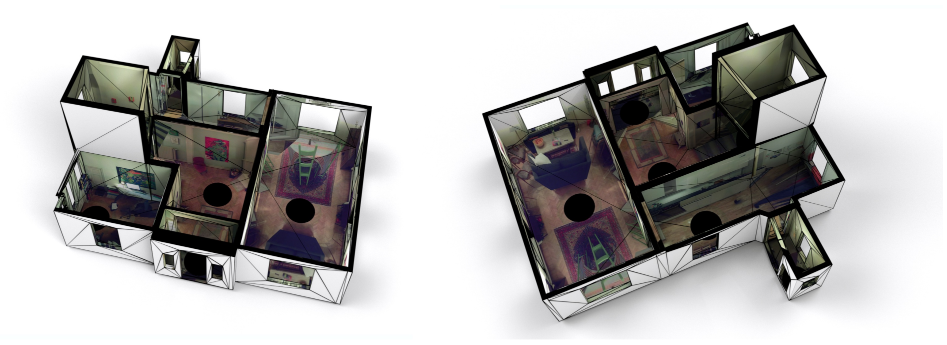


(b)

Figure 4.16: Another example of ortho photo extraction: the input consists of a point cloud that was fused from 7 scans, with a total of 76.98 million points (a). The front facing wall has been removed from this point cloud to facilitate a better view of the scene. A coarse proxy geometry has been extracted, which serves as input for the ortho photo generation, together with the spherical panoramas acquired at the scanning positions (b). Scan data was acquired by the architectural office Fojab in Malmø, Sweden, property of Paul Pierce.



(a)



(b)

Figure 4.17: This figure shows some exemplary ortho photos that were extracted using the proposed approach, black pixels were not visible from any scanning position. The left patch shows the ground plane, the patches on the upper right correspond to wall and window planes. The patch on the bottom right highlights detected sockets (a). The colored patches generated by the method have been used as texture data for the proxy geometry. The figure (b) shows two different views from top of the scanned model, the top floor slab has been removed for a better visual experience.

These objects are typically designed to be unobtrusive and thus are un-textured, but there exists a wide variety of different shapes or colors. The appearance of sockets and switches is generally defined by their silhouettes and mostly homogeneous colored regions of varying colors, e.g. for different brands.

For the classification task, the information inside a patch P is reduced to a discriminative numerical representation, typically a vector with less than 200 dimensions. This representation is called a *descriptor*. For the task of power sockets and switches detection, an approach that utilizes both gradient and color information proved to be the most discriminative. The descriptor contains

- a histogram of oriented gradients (HoG) [39], that models the distribution of intensity gradients, or edge directions. The patch is sub-sampled in a regular grid, and for each cell a histogram of gradient directions is created.
- color information is encoded by separating the color channels, and calculating differences in mean intensity of randomly sized and positioned subregions inside the patch, similar to Haar-like features [205].
- an additional descriptor part was developed that also encodes gradient information, but differently than the HoG descriptor: For each patch pixel where the local gradient magnitude is larger than a threshold, the gradient vector is normalized (scaled to length 1). Then, the gradient vector is projected onto a set of four fixed unit vectors, each angularly separated by 45°. This projection uniquely and continuously describes the vector's direction. By considering only absolute values of these projections, the descriptor becomes invariant to contrary object- and wall intensity values. The final descriptor entries are built using Haar-like differences of random sub-regions, similar to the color descriptor.

For training, a self-annotated database of 41 power sockets and 13 light switches was used. From each annotation, 10 training patches were generated by applying a random jitter, yielding a total number of 540 positive training samples. In order to create negative training data, 726 image patches were randomly extracted from unmarked regions of the annotated set. A random forest classifier [26] was used to retrieve the probability for the object classes power socket, switch and background. A non-maxima suppression is applied after classification on the probability maps to yield the final detection result, see Figure 4.18 for an example classification.

Hypothesis creation

At this stage of the pipeline, the floor plan information has been enriched with detected sockets and switches, associated to a wall in the same manner as openings,

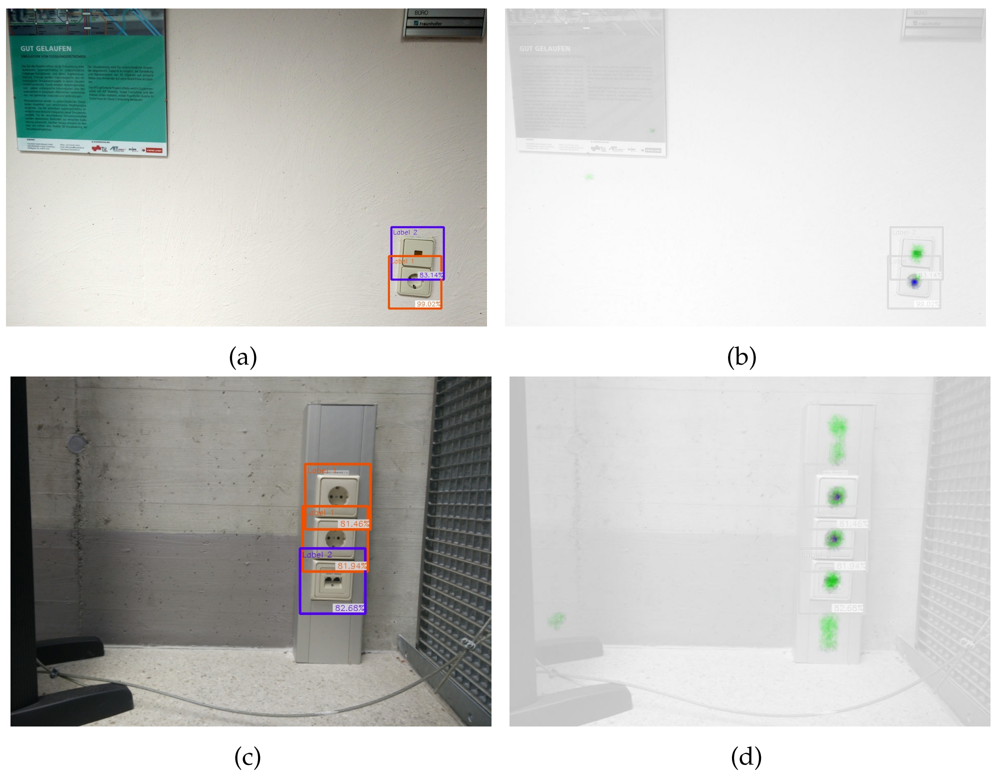


Figure 4.18: A machine learning algorithm is used for detection of sockets and switches. The detected elements are shown as colored rectangles in (a) and (c). The corresponding probability maps are shown in (b) and (d), where white is probability zero and color intensity encodes the probability for class 1 (sockets, blue) and class 2 (switches, green).

see also Figure 4.13. In order to create a hypothesis of probable wiring beneath the surface, additional prior information has to be taken into account. A domain expert might be able to give an educated guess about the position of electrical wiring, based on two key observations:

explicit knowledge that is directly observable, e.g. the locations of sockets and switches that have already been acquired in the pipeline, and

implicit knowledge e.g. "best practice" techniques, that apply to the layout of electrical wiring. This implicit knowledge may also be influenced by mandatory technical standards, e.g. for buildings in the public domain.

As technical standards may differ between countries or change over time, a general approach is desirable that is able to cope with such changes. Therefore,

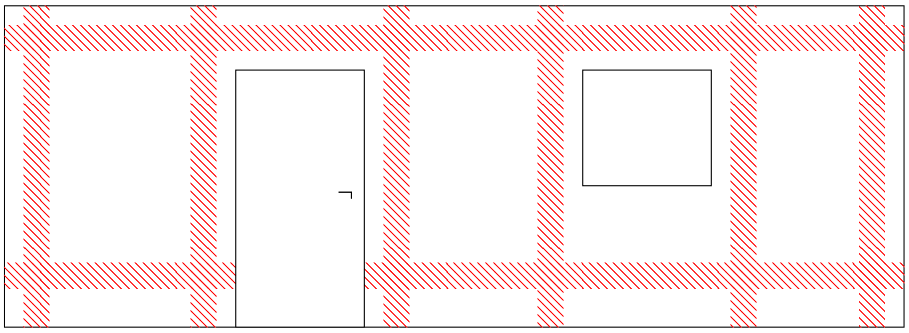


Figure 4.19: A schematic representation of installation zones defined by the German technical standard DIN18015 [44]. The placement of installation zones is influenced by neighboring walls, or forbidden zones, such as windows or doors.

this prior knowledge is encoded as an exchangeable set of rules, that is inspired from formal languages.

Installation Zones

Technical standards define zones for preferred routing of electrical installations. for example the "Deutsche Industrienorm" (DIN), the German industry standard specification, specifies so-called *installation zones* as the preferred method to decide the actual position of cable routing in walls in residential areas, as described in DIN 18015 [44]. Several additions to this standard exist, for example for rooms with bathtubs or showers as described in DIN VDE 0100-701 [45]. A similar approach is defined in the "Stærkstrømsbekendtg relsen", the technical standard in Denmark [56]. The key observation was that these zones are defined horizontally or vertically by distance intervals from walls, or "forbidden zones" such as windows or doors. See also Figure 4.19 for an example.

Electrical Wiring Hypothesis

An overview of the hypothesis creation is shown in Figure 4.20. For each wall in the floor plan description, the system creates a set of installation zones using an installation zone grammar, that utilizes the location of walls, openings and detections as starting rule. Furthermore, a power root has to be specified, e.g. if the location of the main fuse box on the floor is known. The Evaluation of this rule yields the set of separated horizontal and vertical installation zones for each wall. The line arrangement of these zones corresponds to a graph that contains all possible routes of electrical wiring. From this graph, invalid elements, i.e. lines that cross forbidden zones, are removed and unconnected detections are connected via additional routes. Furthermore, given the wall and room adjacency from the floor plan representation, zones associated to adjacent rooms are connected as

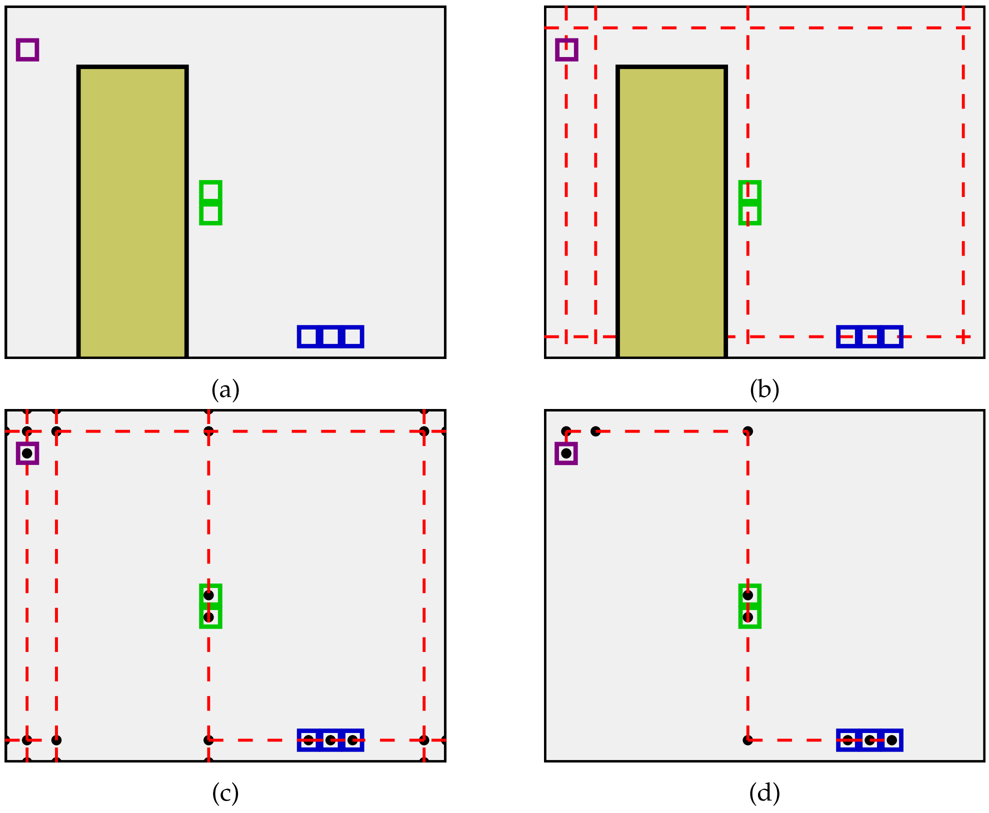


Figure 4.20: The Hypothesis generation begins with the starting symbol that specifies walls, openings (yellow), detections (green and blue), and roots (purple) as seen in (a). Applying the generative rule set, installation zones are created (b). From these zones, a graph is created that connects all given end points (c), and a final hypothesis is extracted that connects all endpoints to a root and minimizes the total wire length (d)

well. To create the final hypothesis, a sub-graph is extracted that connects all detections to a power root, under the assumption that the overall length of power lines, which corresponds to material costs, is minimized using a discrete optimization approach.

Installation Zone Grammar

As has been observed from the technical standards, the zones are placed with respect to wall boundaries, or forbidden zones. We define the installation zone grammar as an attribute grammar $G_{IZ} = (N, T, P, S)$ that consists of the set of non-terminal symbols N, the set of terminal symbols T, the set of production rules P and the start symbol S. Non-terminal symbols `NT` $\in N$ are written in uppercase letters, e.g. `WINDOW` and terminals `t` $\in T$ are written in lowercase, e.g.

hzone. A production rule is written

$$\texttt{NT}\,\{\text{constraints}\} \rightarrow \langle \texttt{t}|\texttt{NT}\rangle\,\{\text{attributes}\} \tag{4.12}$$

The left side of a production rule contains exactly one non-terminal, and an optional constraint that yields a Boolean expression, the rule will be matched only if the constraint evaluates to true. The right side can consist of any number of non-terminal and terminal symbols, together with attribute definitions in curly brackets. An attribute corresponds to a key-value pair. All symbols on the right-hand side inherit all attributes from the left-hand side, afterwards additional attribute definitions are carried out.

The starting production rule produces non-terminal symbols according to the elements that influence installation zone placement (walls, doors, windows, detections). All positional information is stored in attributes, with respect to wall coordinates (see also Figure 4.13). The terminal symbols of this grammar correspond to horizontal and vertical installation zones, as well as forbidden zones, e.g. doors or windows.

Given such a starting rule that corresponds to the information given in the floor plan description, the production rules are evaluated until the list of non-terminal symbols is consumed and only terminal symbols are left. The terminals with a special meaning for the optimization system are hzone, vzone and root, other terminals are treated as an endpoint if it contains the attribute endpoint that is set to *true*. Using this mechanism, users can manually add points that should be contained in the routing hypothesis, e.g. if there are known positions of wiring that could not be detected by the automatic endpoint detection.

This grammar is context free, but it turned out that this approach fails in cases where there is strong evidence for an installation area that is not given solely by the rules of the technical standard, but when the detected endpoints contain large horizontal or vertical structures, e.g. an array of sockets. Therefore, the grammar description has been extended to also contain rules in the form

$$\texttt{NT1} : \texttt{NT2}\,\{\text{constraints}\} \rightarrow \langle \texttt{t}|\texttt{NT}\rangle\,\{\text{attributes}\} \tag{4.13}$$

where the left-hand side consists of pairs of non-terminals, written "NT1 : NT2", with an optional condition statement. Upon evaluation, the production evaluation system creates for these pair rules all possible pairings of concrete instances of non-terminals. These rules match if their condition statement evaluates to *true*. Pair rules are always prioritized until no pair of non-terminals matches, afterwards context free rules are processed.

Using these pair rule system, rules can be defined that match two neighboring elements to a HGROUP: elements are defined to be horizontally adjacent if the bounding box of their union satisfies the constraint that the height of the union corresponds roughly to the height of an element, and the width of the union corresponds roughly to the sum of the widths plus a spacing. Additionally, rules are defined that extend a `HGROUP` by matching the group with an element. A context free rule that matches if the width of a `HGROUP` exceeds a specific threshold will create an additional `hzone`, as is suggested from the horizontal arrangement of elements.

Wire Routing Hypothesis

A wire routing corresponds to a set of possibly connected straight wire segments beneath a wall surface, and is represented by a graph $G_W = (V, E)$, that contains vertices $v \in V$ and edges $e \in E$, where a vertex is associated to a position in wall coordinates, and an edge connects two vertices v_0 and v_1. Vertices can also be associated to an endpoint, such as sockets and switches, or a power root.

The graph of all possible wire routes is created from the list of terminal symbols after grammar evaluation, by building the line arrangement formed by the horizontal and vertical installation zones. Any edges that intersect forbidden zones, e.g. windows or doors, are removed. Terminals that are marked as endpoints are connected to the graph, distinguished by three different cases: terminals near vertices are directly associated to this vertex, terminals near edges will split the edge and introduce an additional vertex on the edge, any remaining terminals will create a new vertex and connected with either a horizontal or vertical edge to the edge or vertex of the graph with the smallest distance. Finally, graphs that correspond to adjacent walls are connected at neighboring nodes using zero length edges.

The final hypothesis is extracted from this graph under the assumption that the wiring was carried out minimizing material costs. The problem of finding a hypothesis sub-graph $G_H \subseteq G_W$, where $G_H = (V_H, E_H)$ is a forest subject to

$$\forall v \in V : v \in V_H \text{ if v is endpoint} \tag{4.14}$$

with the cost function

$$\sum_{v_0, v_1 \text{ of } e \in E_H} d(v_0, v_1) \tag{4.15}$$

with the penalty d being the euclidean distance.

This problem is also called the minimum Steiner tree problem in graphs, which is of practical importance in several areas, e.g. chip design or shortest-length connection of households to a power grid. This problem is also known to be NP-complete, as shown by Hwang and Richards [88], even within an approximation factor of ≈1.129 which was shown by Kaklamanis et al. [94]. For small graphs, e.g. single or few-room data sets, this can be evaluated exhaustively to find the global optimum, for larger data sets a faster, approximative algorithm may be needed. Our current solution implements a local search algorithm that is similar to

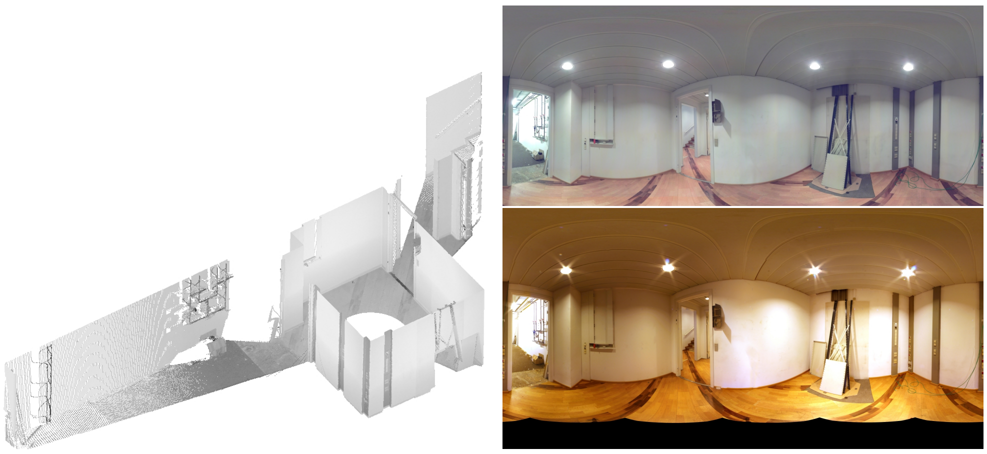

(a) (b)

Figure 4.21: The single room test data set consists of a point cloud with 11 million points (a), additionally to the panoramic image acquired by the scanner ((b) top), a high resolution panorama was acquired ((b) bottom).

"A Fast Algorithm for Steiner Trees" from Kou et al. [100]; this algorithm subsequently grows the final graph by sorting endpoints by 3D euclidean distance and subsequently connecting each endpoint to the graph.

Export and Visualization

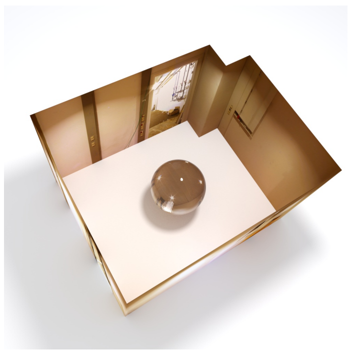
For result visualization, the floor plan representation and the hypothesis graph are converted to a 3D model in X3D format [30], which can be visualized in a web browser using *x3dom* [18].

Wiring Hypothesis Evaluation

After pre-processing, a floor plan representation in the proposed format was derived, and ortho images for each wall (see Figure 4.22b). The endpoint detection yielded the results seen in Figure 4.23a. The grammar evaluation provided the hypothesis of installation zones and endpoint groupings, depicted by the yellow rectangles in Figure 4.23b. Finally, the hypothesis extraction delivered the wire hypothesis shown in Figure 4.23c.

The pipeline was evaluated on several data sets, in this section I show results on a smaller single room data set, further examples can be seen in the D7.4 report of the DURAARK project [54]. The input measurements acquired from this room were a point cloud scan and panoramic images as can be seen in Figure 4.21.

The evaluation was carried out on a Core i7-4930K with 3.40GHz and 6 cores. The ortho photo generation took 2.4s, the endpoint detection 575.2s, and the wire


(a)

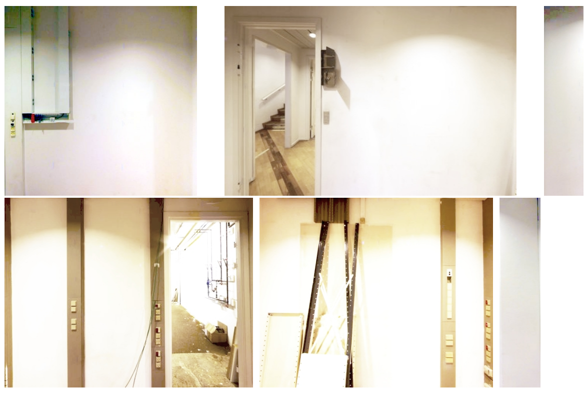
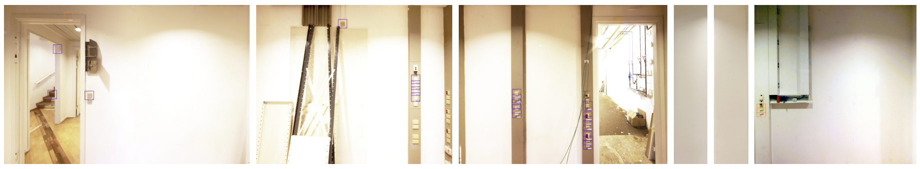
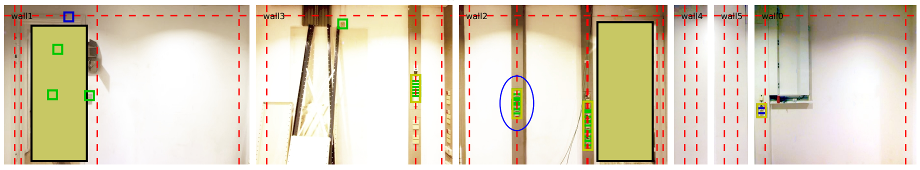
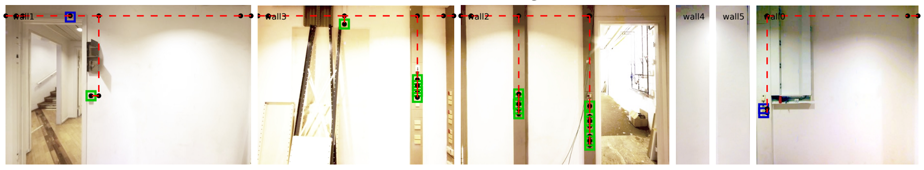

(b)

Figure 4.22: (b) shows the results of the ortho photo projection for the panoramic color sphere of the single room data set, which is shown as a rendering in (a).


(a) Endpoint detection



(b) Installation zones after grammar evaluation



(c) Final Hypothesis

Figure 4.23: The hypothesis extraction for the single room data set: Utilizing the detected endpoints (a), The evaluation of the grammar yields the installation zones, their center lines depicted by the dashed red lines. Note that zones are also generated for context sensitive information from vertically grouped endpoints, e.g. as shown by the ellipse in the middle of wall2 in (b). The final hypothesis is shown in (c).

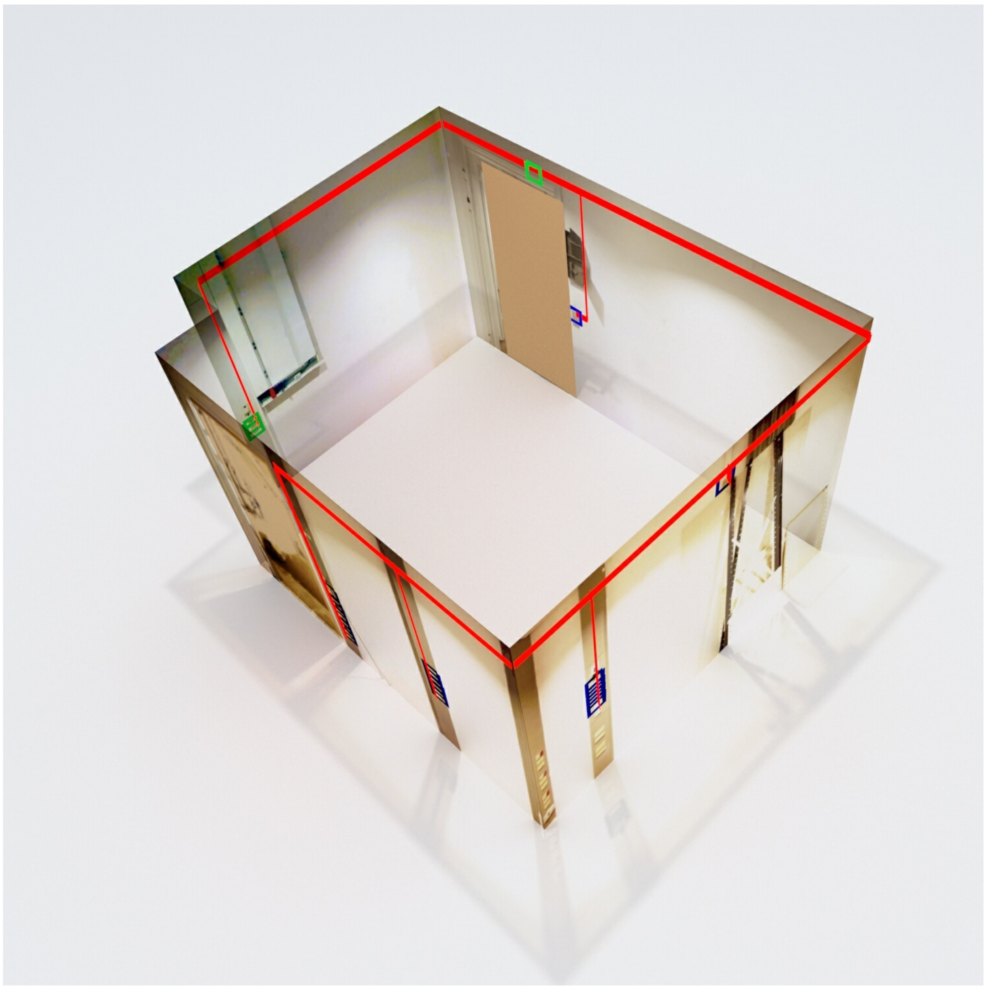


Figure 4.24: The resulting hypothesis of electrical wiring, re-projected into 3D.

hypothesis (inclusive grammar evaluation) 0.44s. The comparatively long runtime of the endpoint detection stems from the exhaustive sliding window evaluation of each wall. This could be further improved by using an interest point detector that reduces the search space of endpoint candidates.

4.3 Summary

In this Chapter, I presented two main contributions of this thesis: the application of rule-based modeling in the context of data completion for measured data. Both methods utilize a generative description to reduce the search space for an optimization step.

In Section 4.1, I describe the first method, a novel technique to reconstruct the two-dimensional structure of a building façade from an ortho photo, using a predefined grammar that encodes the possible buildings. Inspired by syntactic pattern recognition methodologies and compiler construction tools, the presented algorithm finds a parse tree given pixel-wise classifications of the relevant terminal symbols (e.g. walls, windows or doors) and a textual representation of the grammar rules. In Section 4.1, I describe a method to automatically convert these parse trees into a generative 3D model using the shape grammar system described in Section 3.3.

The following Section 4.2 describes a novel method in the context of indoor reconstruction: a first automatic pipeline to geometrically and semantically enrich measured data acquired by indoor terrestrial laser scanning. The method utilizes a two-dimensional shape grammar that encodes the technical standards for placement of electrical wiring. It produces a hypothesis of electrical wiring, given a reconstructed room layout and the detected position of observable endpoints such as sockets or switches, and a textual representation of the grammar rules.

Both approaches have the advantage that the domain knowledge is not hardcoded in the reconstruction or optimization algorithm, but is declaratively specified in a text file, which allows to replace or alter the domain knowledge and thus makes the method broader applicable than an algorithm that is tailored to a specific domain.

The following Chapter 5 showcases collaborative works that utilize the proposed shape grammar system from Chapter 3 and the inverse generative modeling approaches described in this chapter.

5 Applications and Results

This chapter presented a selected list of applications that demonstrate the contributions in the fields of forward and inverse generative modeling. These projects were mostly collaborative works.

5.1 Generative Forward Modeling of Building Facades

The first iteration of the shape grammar system was built on a simplified version that used only rectangular boxes as scope shapes. It was used to reconstruct the university building of the computer science faculty of Graz University of Technology, Inffeldgasse 16, using a generative approach [81], as this building is built of rectangular structures, see also Figure 5.1. The systems structure is similar to other rule-based systems for shape creation, the result is a list of rectangular scopes with different attributes, e.g. wall, window or office space. A description on this semantic level allows not only to visualize the final model, but to create visualizations for various application contexts. For example, visualizing a model where "empty" office spaces and connecting hallways are visualized as filled volumes to show all walkable office space in a building, for more examples I refer to the publication [81].

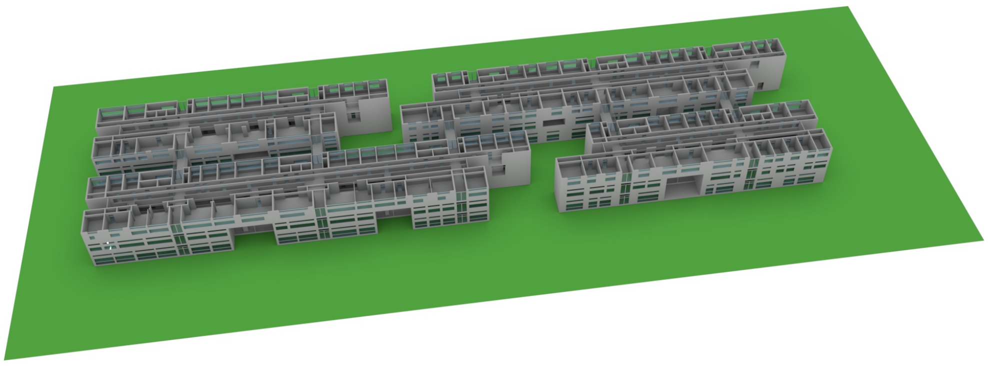


Figure 5.1: The university building was modeled using a box shape grammar approach (modeled by Bernhard Hohmann).

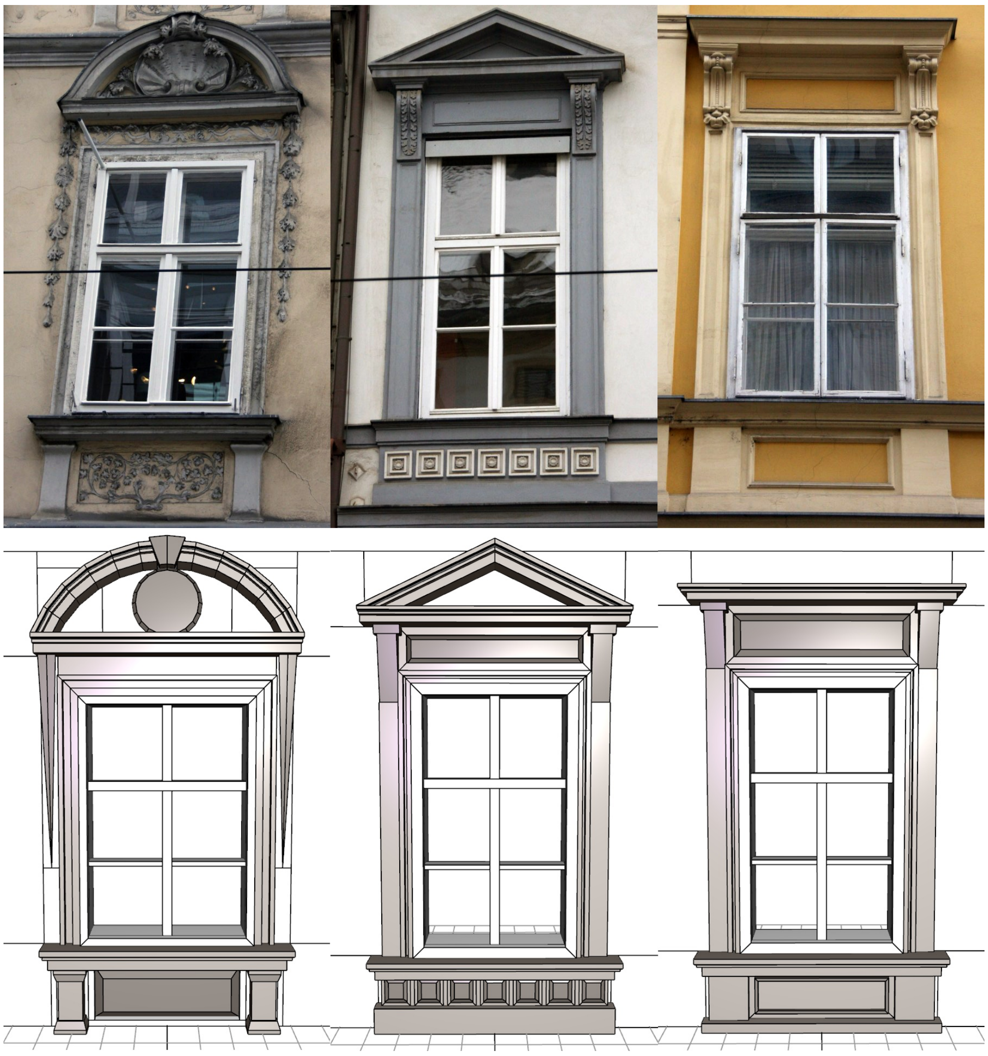


Figure 5.2: Different window styles using the shape grammar system based on convex polyhedra (modeled by Rene Zmugg).

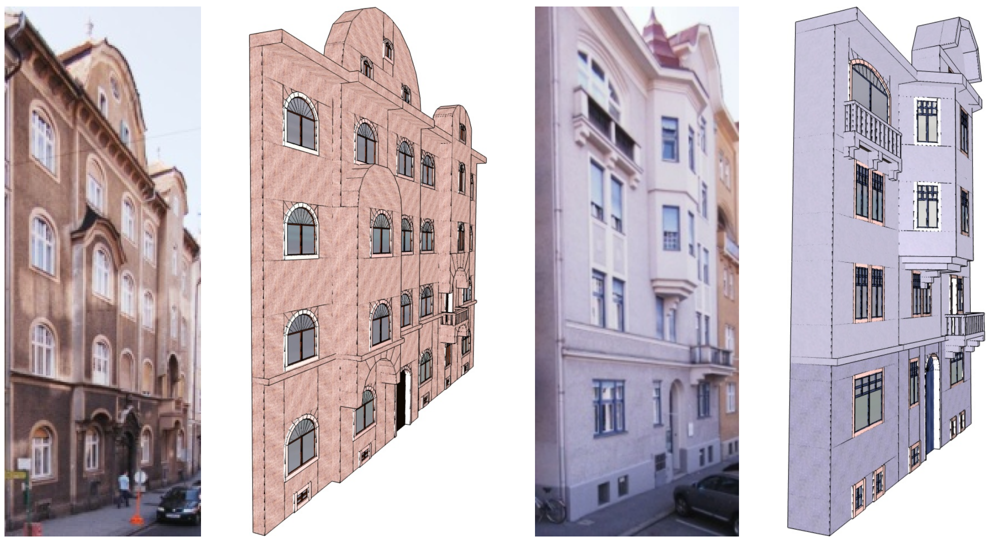


Figure 5.3: Examples of façades modeled from photos. Each façade is modeled using split rules presented in Section 3.5 (modeled by Bernhard Hohmann).

5.2 Inverse Generative Modeling: The CITYFIT project

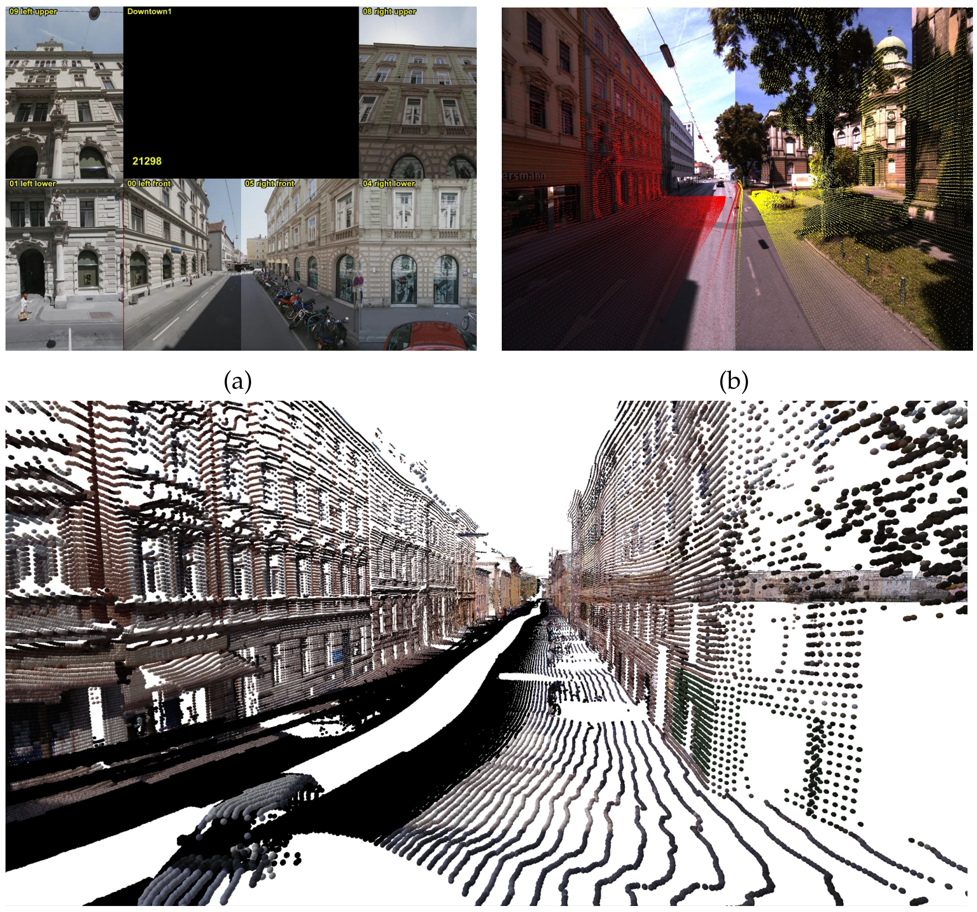
Most of the methodologies developed in this thesis have been developed in the course of the CITYFIT project. CITYFIT's goal was to synthesize a shape grammar that, when evaluated, creates a clean, CAD-quality reconstruction of a building, that fits given highly redundant input imagery (road side photographs) and distance measurements (a LIDAR scanner mounted on a vehicle).

Several buildings were modeled using an early version of the shape grammar based on convex polyhedra. These buildings consist of a manually created set of rules, utilizing the structural rules presented in Section 3.5. Examples of two buildings are shown in Figure 5.3.

Based on the convex polyhedra grammar, parametric elements such as windows or doors have also been analyzed using the *generative fact labeling method*, as was published in [191], see also Figure 5.2.

Data Acquisition

Two data sets were acquired by Microsoft Photogrammetry for evaluating the reconstruction approach. A car with 9 mounted cameras acquired images with at least 20% overlap on the side cameras. Additionally, two LIDAR scanners were mounted that took measurements in a half circle, orthogonal to the moving direction, to the left and the right of the car. See Figure 5.4 for an example of the input data. All data was registered and fused by Microsoft Photogrammetry, the result were all calibrated images from each camera, with an according time stamp,



(c)

Figure 5.4: The first part of the input data are photographs mounted on a car, as shown in (a). The second part of the input data were range measurements, acquired by two LIDAR devices that measure distance around a half-circle to the left and right of the car. The LIDAR points have been projected into a registered picture (b). The points are separated into the stream from the LIDAR scanner mounted on the left (red) and the one mounted on the right (yellow) of the vehicle. Using this approach, a color can be assigned to each LIDAR point (c).

as well as the external camera parameters. For each LIDAR scan, i.e. one line of 180 points with 1° spacing, a time stamp was also recorded, which allows to easily associate relevant points to single pictures.

Reconstruction

The façade localization evaluated on the acquired data set resulted in the separation of 240 individual buildings. The data set consists of 27195 highly overlapping

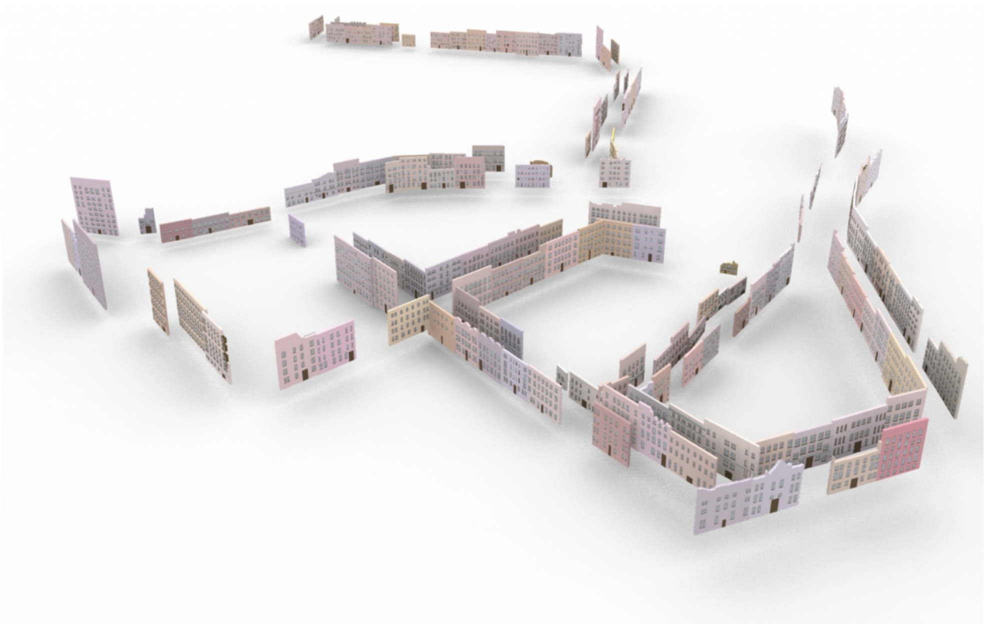


Figure 5.5: The resulted generative reconstruction from scan data consists of a small part of the city of Graz with 240 recognized façades. Note that the generative description represents only the front façades, without streets or other surroundings.

images which was acquired on a driving length of about 3,6km. Furthermore, 18 Million 3D points have been acquired using LIDAR. The pre-processing yielded 63,7 MByte ortho photo image data, and 10,4 MByte grid data, as well as 355,2 MByte raw detection probability data. The data set was processed with the parsing method presented in Section 4.1. After parsing, the converted generative description, the GML source code, is **2,06 MByte** in size, with an additional 400 KByte needed for the shape grammar library. After evaluation, the resulting geometry was exported to an 35 MByte OBJ file. The final 3D geometry consists of 104.387 convex polyhedra. See also Figure 5.5 for a rendering of the whole façade geometry.

5.3 Inverse Generative Modeling in the DURAARK Project

With the ever-increasing availability of cheap and accurate 3D acquisition devices, like laser range scanners, more and more legacy buildings are becoming digitized as 3D point clouds. Additionally, point clouds are increasingly used to document the change of a building state of over its lifetime and its deviation from the original 3D BIM model. Overall, the building industry is moving into a practice which engages various digital representations and data-sets to describe a building, such as: 3D BIM and point clouds, technical annotations in PDFs, legacy 2D drawings and information stored in databases, possibly on the web, such as digital libraries of standards or building parts, GIS and environmental data and the general web based knowledge, which is represented in sources such as DBpedia[1]. Taking into account these issues, as well as the ever-increasing importance of 3D data in architecture and construction, a series of demands become more and more pressing:

- the demand for digital and semantically rich 3D documentation of existing building stock, which will e.g. allow querying and finding information for decisions regarding policy making and renovation.
- the creation of links and relations between the many data-sets, which allows for powerful searches, that combine geometric data (such as the amount and orientation of windows) with descriptive metadata (such as technical data describing acoustic properties) and environmental data (such as the volume of nearby streets).
- the automatic comparison and potential update of data-sets, which allows e.g. to query for changes in a building over the last years.
- and finally means to guarantee long term access to the building information, in order to secure the investment and knowledge that stakeholders aggregate in building data.

What is therefore needed in AEC are technologies and processes to capture existing building stock, as well as sustainable long-term preservation systems tailored to the domain of AEC. The small and medium-sized enterprises (SMEs) of which the European building market is largely composed (across the EU-27, there are currently around 3.1 million construction enterprises, 72.1% of which are SMEs, generating a turnover of EUR 1.5 billion in the EU-27 in 2013 [57]) can hardly initiate the necessary research efforts on their own, yet they would benefit from such research. Several research initiatives focusing on 3D long-term preservation in non- SME-dominated fields, like aerospace, defense, and the automotive industry have been undertaken, see [28] for an overview. Though partially adaptable, the solutions created in these projects cannot be directly applied to architectural 3D data due to several reasons:

- The largely fragmented nature of the industry and the large spectrum of involved sub-domains have led to heterogeneous and inconsistent metadata schemes and ontologies for the description of building elements and their properties in highly enriched BIM models.

[1]`http://wiki.dbpedia.org/`

- The huge stock of legacy buildings, which is represented either by unstructured point clouds or by low- level legacy 3D CAD models from the pre-BIM era, requires elaborated methods of architecturally meaningful semantic enrichment, otherwise, targeted retrieval in the long-term archive is not possible.

- With the advent of digital means of planning in contemporary AEC/FM, additional challenges surfaced. Among them, the lack of interoperability between the large variety of domain-specific computer applications used in building projects is a well-understood problem [61]. In an industry with fast iterations of release cycles for CAD and BIM software packages, reliable preservation policies have to cover hundreds of proprietary data formats. These formats are quickly outdated, often not backwards compatible and each has a complex underlying information model.

This particular problem has been identified in earlier work on digital preservation of building information like PROBADO [20] and FACADE [158]. The use of vendor-neutral, inter-operable, and self-documenting data models has also been employed in long-term data protection (LDP) projects of other engineering domains such as Automotive and Aerospace [28].

Collaborators from web and computer science, knowledge management and architecture initiated hence the European FP7 research project DURAARK (Durable Architectural Knowledge), in order to tackle the challenges which come along with the new data driven practices in AEC [110]. For this, the project developed methods and tools for a highly automated semantic enrichment and long-term preservation of architectural knowledge and data. In this context, semantic enrichment means automated and semi-automated techniques to bridge the semantic gap between the many representations, which are used in AEC. These are lacking semantic information about the context of a structure or the model, which would facilitate non-ambiguous interpretation. Lacking information may as well include detailed material properties, vendor information, provenance data or information about legal, infrastructural and environmental context. One focus of DURAARK was the semantic enrichment of data with particular linked data. Given the continuous evolution of Web data and vocabularies, targeted methods for crawling and archiving Web data in a scalable manner are considered an inherent part of the semantic enrichment approaches developed by DURAARK [16]. The other focus, which is the focus of this chapter, was set on the geometric enrichment of architectural data, especially BIM and point clouds from laser scanning. The two representations constitute opposite extremes along the axes of semantic richness and geometric compactness. BIM models on the one hand include a compact description of explicit geometry, that often conveys design intent and includes attributions and classifications of elements like doors or walls. They also contain vast amounts of textual metadata, which facilitates interpreting, navigating and browsing such data. They represent the built reality in a highly abstract manner. Point clouds on the other hand are able to capture the present state of a building in high precision and can convey information about its usage and spatial atmosphere. Point clouds are however computationally heavy and cannot be processed easily by existing tools in AEC. In order to support many common use cases in the building profession and the digital preservation of buildings, these two opposed forms of representation have to be mapped and transferred in order to allow a comprehensive overview of the referred physical artifact. This mapping of information can be thought of as a movement along the semantic richness and geometric compactness axis which form fundamental building blocks of the DURAARK project: Geometric Enrichment describes the effort of adding geometric details to existing "as-built" models or deriving explicit geometries from measured data. Semantic Enrichment describes the contextualization and attribution of the underlying models from various resources including linked data. Synchronizing both representations by bridging the semantic and the geometric gap enables seamless preservation of heterogeneous architectural data. Two standardized file formats were instrumental for the techniques that were developed in Section 4.2: The Industry Foundation Classes (IFC) [92] is an ISO standardized data model for buildings based on the standard for the exchange of product model data (STEP) [91], which is used across numerous engineering domains [154] and the E57 file format to capture point cloud data structures [83]. For an in-depth look at the suitability of IFC as an archival format, I refer the reader to [118].

In order to investigate the necessary technologies and processes and to demonstrate the benefits for the profession the DURAARK consortium developed a web based prototype for a long-term preservation system — the DURAARK workbench. It includes workflows for data producers in the preparation of their 3D models for the submission to a digital preservation system and the semantic and geometric enrichment of data.

The DURAARK Workbench

The workbench is a graphical web application that runs in a browser, either locally on the stakeholder's computer, or as a web service in a private or public cloud. See Figure 5.6 for a screenshot.

The functionality of the workbench is organized into two workflows: the *Pre-ingest* and the *Retrieval* workflow. The pre-ingest workflow prepares the 3D data files to be added to a digital preservation system. Its steps handle the selection of input 3D data files, the extraction, manipulation and enrichment of metadata, and the creation of files containing additional geometric information. After the pre-ingest workflow is finished, all necessary data files, enriched metadata and geometric enrichment files are available to create a data package. The package is

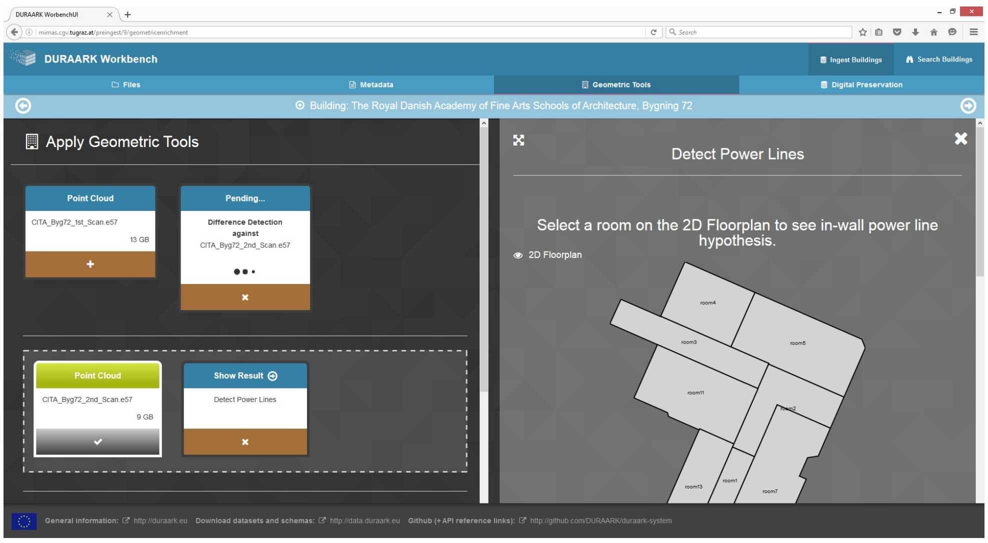

DURAARK Workbench
Ingest Buildings
Search Buildings
Files
Metadata
Geometric Tools
Digital Preservation
Building: The Royal Danish Academy of Fine Arts Schools of Architecture, Bygning 72
Apply Geometric Tools
Point Cloud
CITA_Byg72_1st_Scan.e57
13 GB
Pending...
Difference Detection against CITA_Byg72_2nd_Scan.e57
Point Cloud
CITA_Byg72_2nd_Scan.e57
9 GB
Show Result
Detect Power Lines
Detect Power Lines
Select a room on the 2D Floorplan to see in-wall power line hypothesis.
2D Floorplan
room4
room3
room5
room11
room2
room13
room1
room7
General information: http://duraark.eu
Download datasets and schemas: http://data.duraark.eu
Github (+ API reference links): http://github.com/DURAARK/duraark-system

Figure 5.6: The DURAARK workbench.

then either transferred to a long-term archival system like *ExLibris Rosetta* [2] or it can be downloaded locally for further processing.

The retrieval workflow allows to use the workbench as a flexible search system for architectural data. An internal knowledge graph contains a continuously growing information pool of buildings and their surrounding context, e.g., geographic, historic or legal context. The pre-ingest workflow directly adds or updates data to the knowledge graph, e.g., by using the focused crawling component to add information topics related to a building. Additionally, for each building instance ingested into the knowledge graph, a background task is automatically enriching the metadata with additional information. Together with properties extracted as part of the geometric enrichment, carried out in the pre-ingest step, a stakeholder is given fine-grained control on the archive search, allowing a great set of usage scenarios. Archive queries can reach from simple queries like "list all buildings which are located in *Berlin*" to more sophisticated ones where multiple properties are used, e.g., "list all buildings which have the architectural style of *Art Deco*, are designed by the architect *Ludwig Hoffman* and are located in *Berlin*". The search system uses SPARQL[3] as query language. SPARQL queries are very flexible and allow fine-grained control how to combine properties into a search query.

The electrical wiring reconstruction method presented in Section 4.2 has been integrated into the geometric enrichment pipeline of the DURAARK workbench.

[2]http://www.exlibrisgroup.com/category/DigitalHeritage/
[3]SPARQL: https://www.w3.org/TR/rdf-sparql-query/

5.4 GANDIS: Forward Modeling of Generative Buildings

In this section I present results from the GANDIS project, which attempts to close the gap between early design and construction-ready planning using a novel method: *procedural building templates*. It was a collaborative work together with a group of architects from the company ORTLOS engineering.

An important drawback of the conventional design process is that some implications of the initial design become apparent only after construction planning. The energy footprint of a building, for instance, depends on the A/V ratio (surface to volume), the glass proportion, the number of floors, but also of the wall insulation standard. So the requirement was to create an interactive planning tool that allows for computing characteristic key values from a set of prototypical parametric buildings. Even if the prototype buildings are only roughly similar to the building to be planned, the tool allows judging the impact of changes in the building on the energy footprint. This allows answering questions such as: How compact should the building be, what happens if we create this overhang, and how expensive is it to compensate energy loss by better insulation on that wall, etc.

We consider parametric building templates to be useful for many applications other than energy design. The examples created in this case study are simple, but once the approach is clear, more targeted and elaborate building models can be created using the same approach. The objective of this paper is to illustrate the process of developing parametric building models, in the hope that others can learn from it. We seek to provide an enabling technology for interactively changeable complex procedural building templates.

In the beginning, we had to assess the variability of the buildings. Thus, the process starts from a set of example buildings provided by architects, see Figure 5.7 for an example. These reference models provided a helpful guideline in the discussion that led to the abstraction of the buildings - they span the *design space* that is to be parameterized. To define this space unambiguously, however, required many discussions, e.g., on the parameter minima and maxima, or to clarify which special cases should be prevented to guarantee that the building created remains valid. The next step was to develop first simple buildings, and then to successively extend them to create more complex ones. This inductive process is marked by continuous refactoring in order to find a set of re-usable parametric sub-constructions (doors, windows, floor plan processing). So our experience was that developing the first few models took most of the time, while later the tool set and thus, the design space, were powerful enough to create more elaborate models faster.

In total, two parametric models have been developed for each of the four basic shapes, as shown in Figure 5.8 using the proposed shape modeling system, as described in Chapter 3. The first three basic shapes, *rectangular*, *L-shaped* and

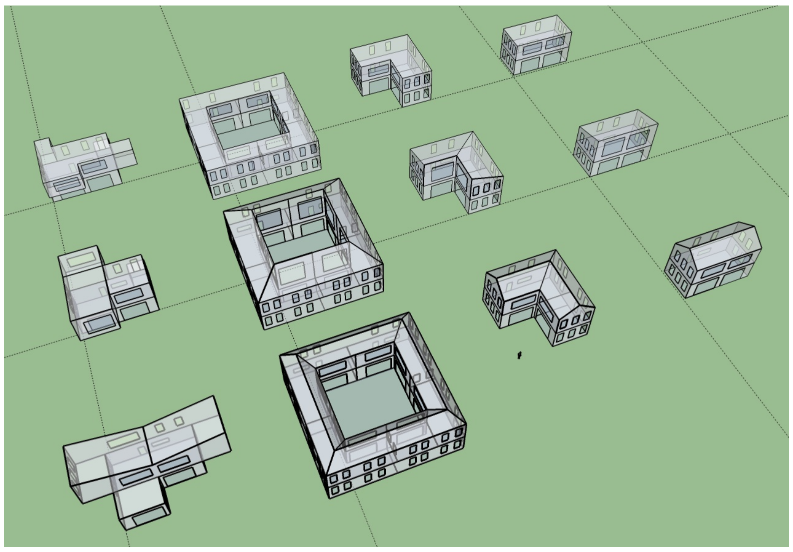


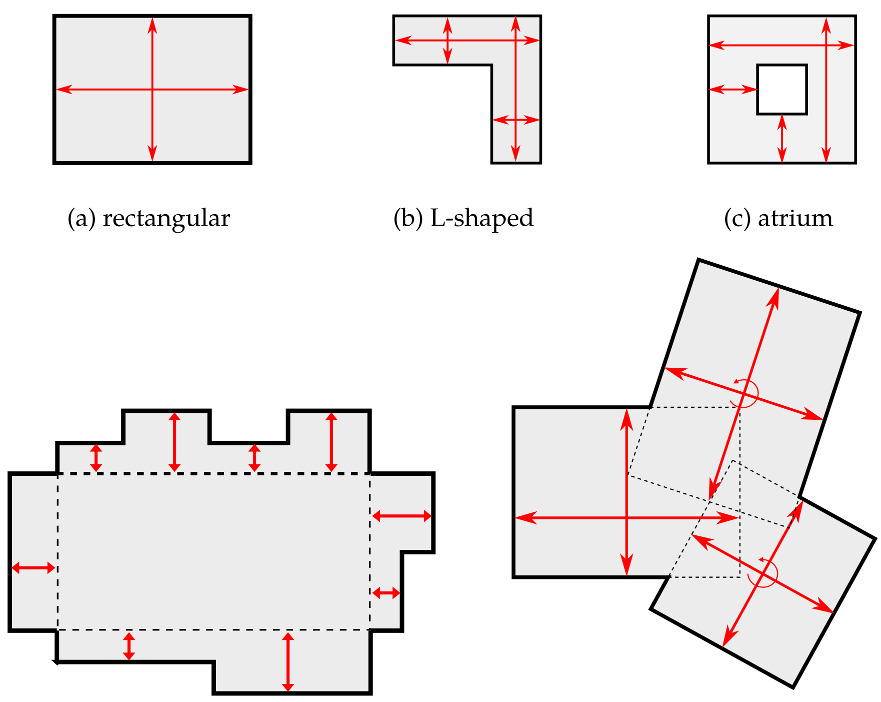
Figure 5.7: The design space of the "residential" buildings group was explored by architects by creating static reference models with conventional modeling software, Google SketchUp. These models served as guideline for further generative abstraction. Models by ORTLOS engineering.

atrium, are intended to approximate the more accurate models that will be created later in the design process. The fourth "shape" is called *free-form* and is intended not as a stand-in for any specific building but rather as an example that the user can manipulate in order to learn more about energy efficiency. The exact set of parameters varies between the different models. Common to all the models is the "number of floors" parameter; the rectangular buildings have length and width parameters, while the other types require more parameters to define their shape. Additionally, the residential building models allow a choice of three different roof shapes. The *Office* series of templates was created from the *Residential* series merely by replacing the façade decoration rule, see Figure 5.9 and Figure 5.10.

Extracting Measurements from Geometry

The main objective of the GANDIS project is to assess the energy efficiency of a procedurally generated parametric building. In fact only a few key quantities are required for the approximate energy efficiency calculation.

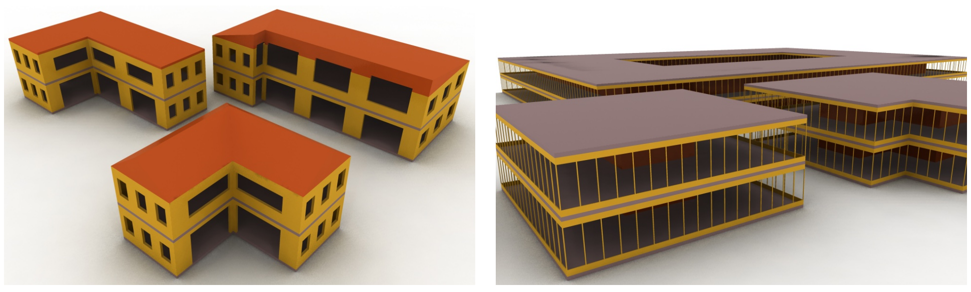
Some of them, such as the exterior area and the enclosed volume of the building, can be directly calculated from the parameters. It is much easier to multiply length, width and height of a box than to calculate the volume of a box-shaped building from the volumetric CP model. The most interesting challenge was to find out the exact amount of glass area on each of the façades of the building.



(a) rectangular (b) L-shaped (c) atrium

(d) freeform residential (e) freeform office

Figure 5.8: The extracted parameterized design space consists of simple rectangular, L-shaped or atrium base shapes, or more complex freeform residential and freeform office configurations. The degrees of freedom that were implemented as parameters for the generative description are shown in red.



(a) residential façade (b) office façade

Figure 5.9: The residential (left) office (right) façades share the same parametric basic shape template, detail structure is applied using different façade partition rules. The residential buildings allow a choice of different roof shapes.

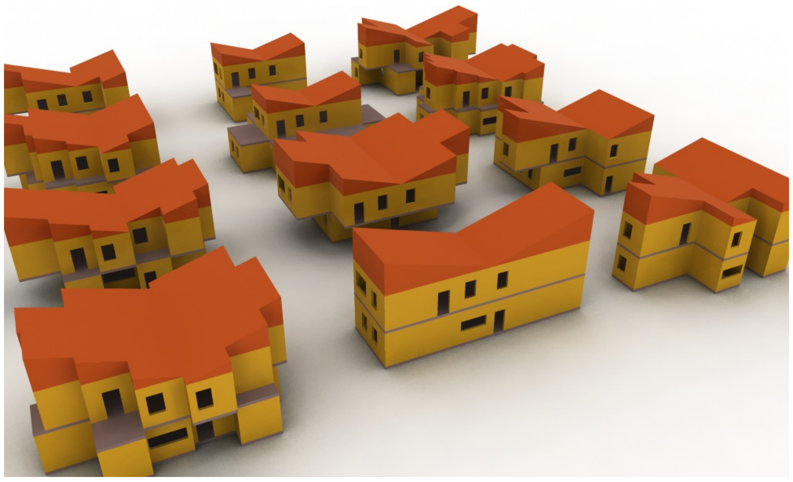


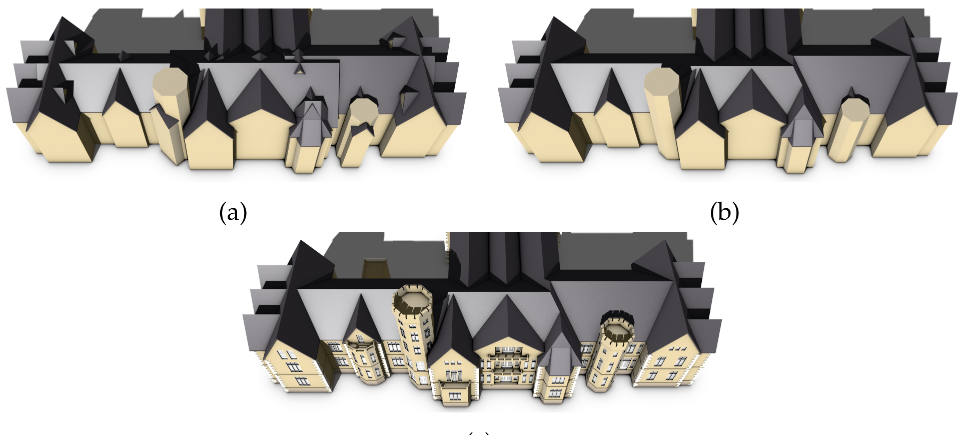
Figure 5.10: Various instances of the parameterized residential freeform building.

The area occupied by windows depends on the particular grammar used to define the façades. Different grammar rules behave differently when the wall length changes; therefore, we want to calculate the window areas just when building the wall geometry, as this is possible then with little extra effort. The method of choice is to collect all "glass" terminals that were produced for each outer wall, and to calculate the total area of CP faces that point in the query direction *after* building evaluation.

5.5 Forward Modeling of Constructive Roof Geometry

The work of Johannes Edelsbrunner et al. [52] utilizes the library on convex polyhedra in the context of generative building creation for virtual worlds. Due to the complexity and vastness of such worlds, manual creation is often very demanding in resources. A generative approach can help to reduce these costs by allowing to specify building by a higher level approach, a set of primitives, and perform the geometric detail evaluation of the combined model to the system. The domain of roof generation is especially challenging, as existing fully automatic roof generation algorithms might not yield desired results, and complete manual specification can get very tedious due to complex geometric configurations.

The *Constructive Roof Geometry* system allows to quickly create a complex building by combination of convex base parts; the roof for each part is generated automatically. To correctly combine the parts, the correct situation from a discrete set of trimming rules is applied to a combined model, which is also specified in the abstract building model description. See Figure 5.11c for an example of a model without trimming (Figure 5.11a) and with applied trimming rules (Figure 5.11b). The resulting model can be further refined and detailed by using the standard



(a) (b) (c)

Figure 5.11: Constructive Roof Geometry addresses the problem that arises for roof configurations when a building is described generatively using parametrizable parts. (a) shows that a simple union can lead to unwanted protruding parts. The method presented in [52] addresses this problem by introducing automatic roof trimming for solid building primitives (b). Applying further refinement using parametric rules yields a detailed result (c). Modeled by Johannes Edelsbrunner.

shape grammar system based on convex polyhedra.

The system proposes an abstract building model that is in which a compound *structure* is composed of several *solids* and their trimming influence to each other. Each solid is composed by several sides, with additional trimming information. For a more extensive description of the proposed roof language I refer to [52], especially the graph shown in Fig. 14 and its definition in Section 4: "Specification of the Building".

5.6 GMLCompositor: A User Interface for Generative Forward Modeling

Wolfgang Thaller built a method for interactive visual editing of generative models, based on the generative modeling language GML and an underlying data flow representation named *code graphs* [187], [189]. A code graph is a hyper-graph in which nodes contain values and hyper-edges - which connect an arbitrary number of input nodes to an arbitrary number of output nodes - correspond to functions or operations. The user interface is called *GMLCompositor* - the interface can be seen in Figure 5.12. The system allows visual editing and construction of generative shape representations using various types of shapes and respective modeling operations. The modeling system based on convex polyhedra described in

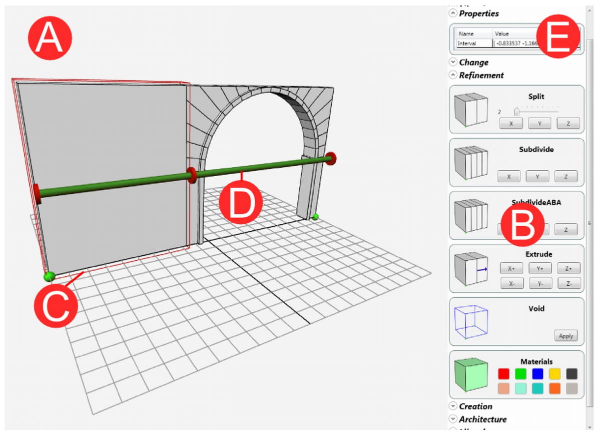


Figure 5.12: The **GMLCompositor** developed by Wolfgang Thaller et al. [187, 189] is a tool for direct interactive visual editing of generative models. It consists of a 3D view, where the current evaluation result of the generative description can be inspected (A), a toolbar that contains various modeling operations (B), visible parts of the 3D model can be selected and are highlighted in red (C), depending on the generating code context, interactive 3D widgets are displayed that allow direct manipulation of modeling parameters (D). These parameters can also be edited in a context dependent property box (E). The modeler supports interactive modeling of the proposed system using shape grammar rules based on convex polyhedra.

Chapter 3 is one of them.

The direct visual editing approach was evaluated on a case study, in which parts of the Louvre Palace located in Paris were reconstructed [215]. The system also allows *linking* of different model parts to have the same rule applied to each member of a group of linked scopes. In a coarse-to-fine methodology, coarse outlines of buildings have been modeled based on digitized old floor plans; extruded walls or structures are gradually refined using rule application, see Figure 5.13.

The system allows two create a arbitrarily oriented *box* like volumetric shape as a starting point for modeling. These boxes were aligned with the coarse outlines of buildings as a starting point. At any point in the modeling hierarchy, the user may assign either *void* to the volumetric shape, which makes the shape invisible, or assign a *fill material*, which effectively renders the shape in the final model.

The available refinement operations are:

- the *interval split*, which is similar to the operation described in Section 3.3 that partitions a volumetric shapes into smaller sub-volumes.

- the *extrude* operation, see also Section 3.3, which allows to extend a volumetric shape in a given direction.

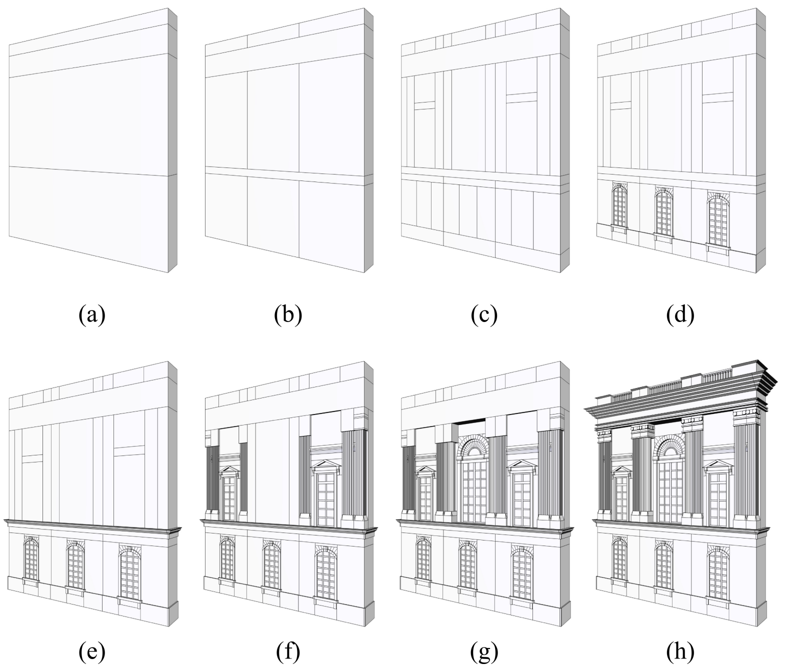
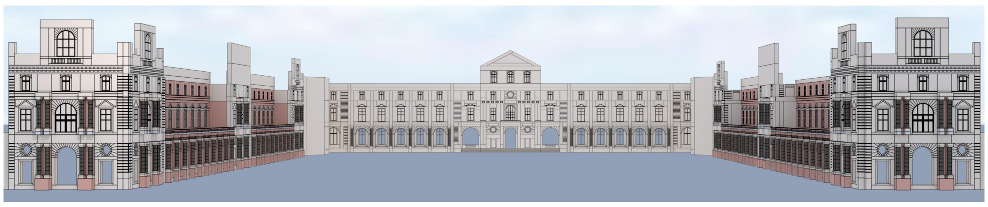


Figure 5.13: A building part, in this case a part of the outer east wing pavilion of the Louvre Palace, is reconstructed by gradual refinement using split and extrude operations.

- the *diagonal split* diagonally partitions a volumetric shape, which is assumed to be of rectangular shape along one of its diagonals; there are 6 possible cases.
- the *repeat* operation splits a volumetric shape into equally sized parts, the parts are linked, which means that the same rule applies to them.
- the *repeatABA* operation splits a volumetric shape in an alternating sequence of two differently sized parts A and B, e.g. for automatic distribution of pillars and gaps.
- the *merge* operation merges several volumetric shapes into one shape.
- the *link* operation creates a group of object that are treated in the same way, i.e. the same rule is applied to each object in the group.
- the *specialize* operation creates an exception rule for a linked group, i.e. the group is partitioned into two disjoined link groups.
- the *mirror* operation mirrors the orientation inside a volumetric object by changing its local coordinate system.



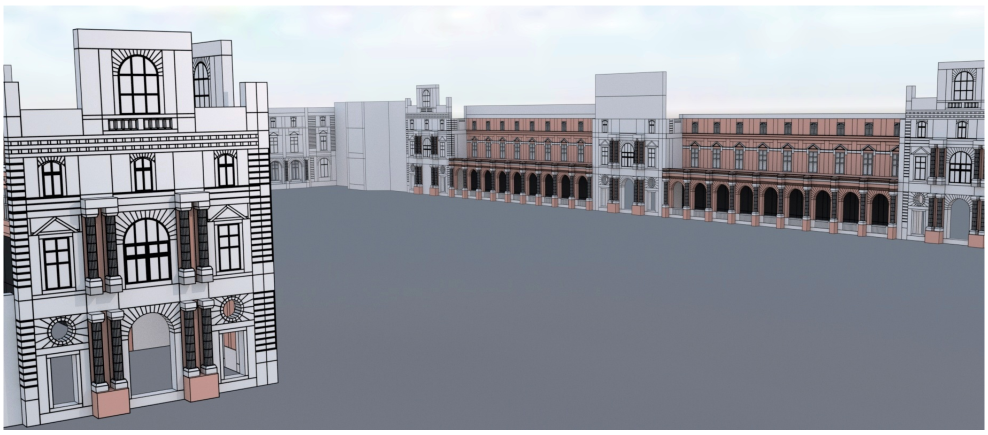


Figure 5.14: Parts of the Louvre Palace have been reconstructed in a case study presented in [215], using the proposed rule-based system based on convex polyhedra. These images show an intermediate step, also highlighting the convex partition of the resulting model (modeled by Martin Pszeida).

- the *rotate* operation rotates just the local coordinate system of a volumetric shape, without changing the shape itself, i.e. the rotation effects rules which are applied afterwards.

In addition to the above low-level modeling operations, the system also includes a set of pre-modeled parametric assets, such as doors, windows, arches, columns and other architectural elements which can be inserted into a volumetric shape. The distinction between the low-level modeling operations and the parametric assets is to some degree arbitrary; both use a given volumetric shape and yield a partition or parts for later use. Such parametric partitioning assets can be very versatile and may be used in several different situations. For example, a round hole operation can be applied subsequently to produce a window with borders, or rotated to produce a round pillar.

Due to time constraints, the reconstruction was finished only partially. The result was exported and rendered using Blender, which is shown in Figure 5.14.

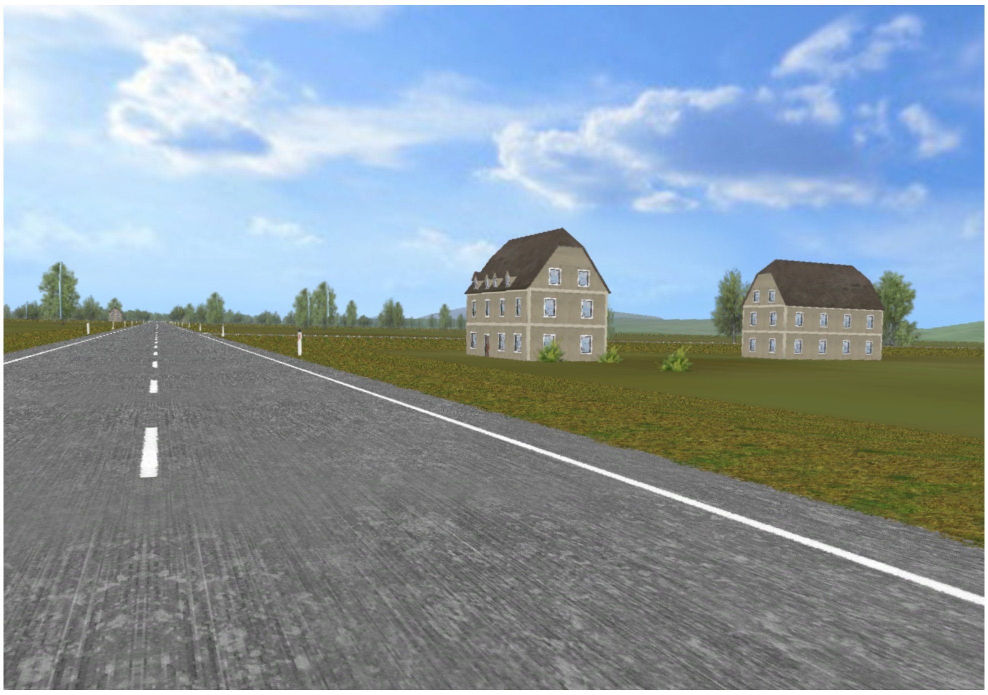


Figure 5.15: Generative modeled buildings have been used to populate a rural scenario for a driving simulation. The generative approach allowed for automatic creation of several levels of detail, which is a crucial performance improvement for large outdoor scenes (scenario by Volker Settgast).

5.7 Generative Forward Modeling of Parametric Houses for Driving Simulations

Using the aforementioned structuring, I implemented a simple procedural model of a parametric house that was used in the context of environment creation for a driving simulation. While not many parametric instances were used - to improve performance by re-instancing - the generative approach was used to create several levels of detail for a building. A screenshot can be seen in Figure 5.15.

The building is structured in the following way: A starting shape is split into floor slabs and floor spaces using a subdivide operation. Then the floor spaces are split into wall and inner building space using a frame split operation. Each wall is then subdivided along the wall axis, and split into tiles that correspond either to windows or doors if the current floor is the first floor. The roof is created by extruding the top floor and then applying a gable split operation. The dormers are just smaller scaled variants of a building split with gabled roof, similar to the structuring presented in [51]. Two buildings have been created that use the same set of parametric window and door terminals.

5.8 Summary

In this chapter, I reviewed some application scenarios in which the presented methods for forward and inverse generative modeling have been applied. These projects have mostly been collaborative work, and demonstrate the practical application of the proposed shape grammar system, and demonstrate a scope of application for the proposed inverse methods.

6 Discussion

In this Chapter I reflect on the findings obtained with the development of the proposed generative methods. It is divided in two parts. The first part discusses the rule based generative modeling system using half spaces. The second part reviews the presented inverse modeling approaches.

6.1 Shape Grammars on Convex Polyhedra

The proposed system improves the state-of-the-art by utilizing a more general shape representation to create generative rules (Chapter 3). Geometric queries were developed with a focus to facilitate reusability (see Section 3.4), which allows creating more general rules that can be applied to a broader family of shapes. The system was designed with a focus on robustness and evaluation efficiency (see Section 3.4) and was successfully applied in various projects and applications, as was shown in Chapter 5.

The rule based system presented in Section 3.3 is essentially a context free shape grammar. Thus, the evaluation corresponds to a hierarchy, a tree. The root of the tree corresponds to the starting symbol, non-terminal symbols of the grammar are tree nodes, and the grammars' terminal symbols are the leaves of the tree. Creating a rule based representation of a real world object, for instance a façade, the symmetries present in the object are rule candidates. However, such symmetries may not always correspond to a tree structure – which is a limitation of context free grammars. An example can be observed in the façade shown in Figure 6.1: the symmetries of a rectangular façade may be floors (vertical partitioning) and similar columns (horizontal partitioning). Let's assume we split the building first into columns and then into floors, we need to replicate the floor split for each column with a different non-terminal rule. Depending on the desired model and underlying shape grammar system, this can be addressed by enforcing constraints over several rules by using attributes, or adding special rules that enforce the constraints directly, e.g. a "grid" split in the above example, which partitions a scope into grid elements at once.

A similar difficulty is to define a general rule that influence geometry that is generated in a lower part of the rule evaluation hierarchy - e.g. the creation of a

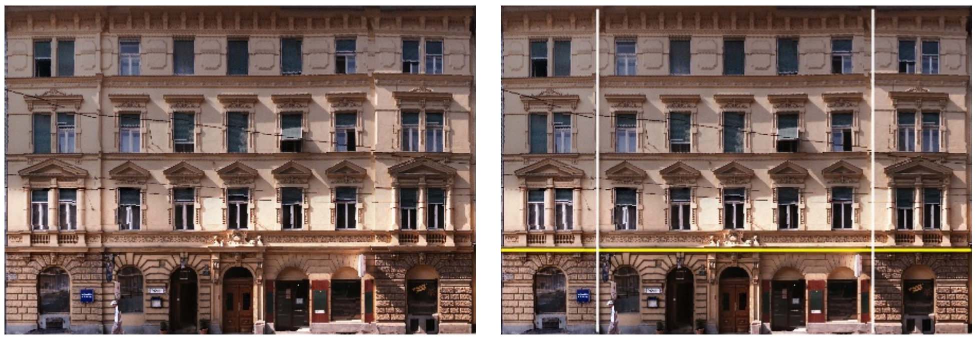

(a) (b)

Figure 6.1: The operations of a grammar should match the inherent structure of the desired generative model: using only horizontal and vertical splits yields an ambiguous situation for the façade shown in (a). In this example, the façade is split into three columns horizontally and into ground floor and above vertically (a). The split parameters for the second split need to be replicated for each scope after the first split. This needs to be addressed if the final model should contain both constraints, e.g. if the ground floor height and column width should be adjustable parameters.

ledge that goes horizontally along a façade, crossing windows or balconies. The abstraction *ledge* suits into the floor partitioning part of a façade, as ledges often visually display floor transitions and correspond to the same partition direction. But, ledges also go around protruding elements or columns, which are usually defined in a lower hierarchy than the floor split. The current solution to these problems is propagating attributes to scopes that mark them for later processing, and perform the ledge operation in a separate step after evaluation of the hierarchy.

Having a more general geometric representation, (the convex polyhedra system presented in Section 3.2) and rules that automatically adapt to this shape has advantages over shape grammar systems that utilize a set of basic shapes. These systems need to duplicate each split or repeat rule for each basic shape. When working with more complex shapes, non-convex objects need to be expressed by a union of convex parts, and this union may lead to dependencies on explicit shape configurations; identification of parts via search directions may lead to ambiguous situations in non-convex objects. Moreover, the convex partitioning is done manually by the expert that creates the rules, because it is not yet clear how to automatically find a convex partition that corresponds to the desired abstraction hierarchy.

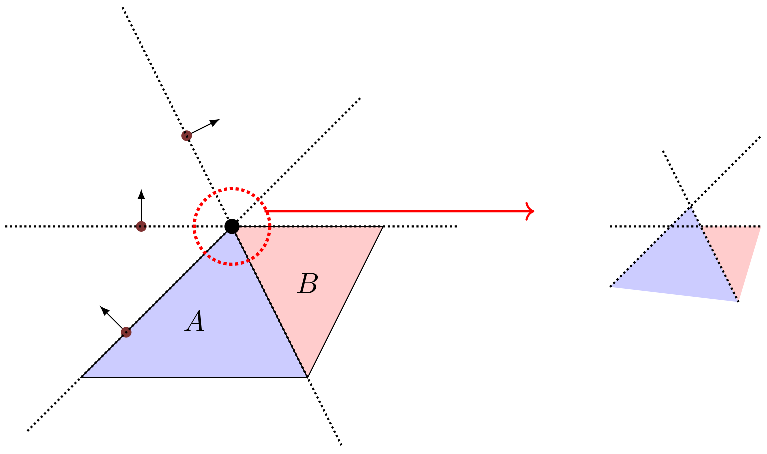


Figure 6.2: An example of geometric rounding artifacts that arise when constraints are present, such as adjacent polyhedra A (blue) and B (red): The encircled vertex is shared from both A and B, but the three involved bounding planes might not exactly intersect in one vertex, due to limitations of numerical representation.

Advantages and Limitations of Generative Half Space Construction

The decision to use a quantized plane-based representation was caused by the fact that numerical representations in computers use a fixed size of bits. Intermediate results need to be rounded which leads to problems, as was reviewed in Section 2.6. Thus, by using quantized plane-based representation, the rounding step is carried out when computing coefficients for a new plane in the system (Section 3.4), special construction operations were presented for orthogonal and bisector planes. By quantizing plane coefficients and using static filters we can choose how to value the trade-off between evaluation speed and model resolution (or precision) by varying the number of bits used for quantized plane representation. In this representation, the vertices of a convex polyhedra are then evaluated robustly (exact) and efficiently.

While the presented construction operations are tailored to produce representable planes, in some cases rounding is necessary. This can lead to microfragments in the generated geometry, but still evaluates robustly. An example of a situation that introduces rounding is the case of two adjacent polyhedra that should share vertices. Such constraints result in situations where four or more planes should exactly intersect in one vertex. See Figure 6.2 that demonstrates the problem on a two-dimensional example.

These artifacts are not a problem for visualization and rendering, which was the main initial design goal for the system. Nevertheless, the perturbations introduced by the quantization should be removed when exporting geometry to other systems, and thus introduce an additional post-processing step.

6.2 Discussion on Inverse Generative Modeling

Inverse Generative Modeling derives a generative model from observations or coarse descriptions. Within this thesis, I examined two inverse approaches that leverage generative descriptions to express prior knowledge of a model domain – architectural design elements of building façades and technical standards for the placement of indoor electrical wirings. This prior knowledge is used to restrict the solution search space.

The first approach utilizes a two-dimensional split grammar to parse the structure of façade ortho photos; the parse trees are converted to a generative model that evaluates to a 3D model of the façade. The second approach utilizes a rule based description of technical standards to create a hypothesis of electrical wiring below walls, given 3D scans and photographs of indoor office buildings.

Façade Reconstruction

The presented reconstruction approach uses principles from compiler construction to parse the structure of a façade. Observations are given as results of pixelwise object classifiers, e.g. the probability of a pixel belonging to the wall, window or door class. A grammar was developed that favors rules from classical architectural design: mirror symmetries and repetitions. The proposed parsing algorithm finds a hierarchical description of concrete application of these rules, the parse tree that corresponds to the description with global lowest costs. The parsing algorithm utilizes dynamic programming, which means the problem must satisfy the optimal substructure property – which is satisfied for context free grammars. An irregular rectangular grid was used to reduce the complexity of the search results, and thus the computational effort. This grid groups areas of similar classifier results; grid lines are candidates for split lines of the grammar.

In Section 4.1, I presented a method to automatically translate the parse trees into a generative model. This generative model utilizes the half space modeling approach presented in Chapter 3 to construct a 3D model of the parsed façade, which can then be used for visualization or further modeling. See Figure 5.5 for an automatic reconstruction result for a part of the city of Graz.

While the parsing or optimization step is independent of a concrete rule set, this rule set needs to be created by a domain expert. This gives more control to specifying the intended domain knowledge, in contrast to a method that fully automatically learns a model. The cost function was designed to prefer structural symmetry recognition, such as repetition of windows or symmetric columns of similar façade structure. A strong benefit of the proposed method is the ability to parse optical measurements to any given grammar – which allows to exchange the prior knowledge, without the need to re-implement parsing and optimization.

Electrical Wiring Hypothesis

The second inverse modeling approach was designed to fit into a first fully automatic pipeline that creates a hypothesis of electrical indoor wiring from terrestrial laser scans of office rooms. It answers the question "Under the assumption that the constructions in this building followed the given technical standards, what is the most probable position of electrical wiring, given observable endpoints (sockets and switches) and the room layout."

The room layout was extracted from terrestrial laser scans. Moreover, color information was extracted from the scans to yield rectified pictures of walls; Sockets and switches were detected on these pictures using computer vision object detection methods. The wall segments of the floor plans as well as the detections are transformed into start symbols for a non-context free attributed two-dimensional grammar that encodes the technical standards prior. The evaluation of this grammar yields the position of installation zones for the observed room layout. These zones are connected to a graph that represents all possible wiring positions. From this graph, a final hypothesis of electrical wiring for the observed situation is extracted. The final layout connects all endpoints and minimizes total wire length – which corresponds to the assumption that the building was planned cost-effectively by reducing the amount of needed copper.

In a discussion with domain experts, two points of special interest emerged. First, it was noted that the intermediate result of all possible installation zones was also of interest. It could be used for planning electrical installations for a re-building project for example, and thus would be an interesting add-on in an AEC planning tool. Second, it was noted that such a tool could also be valuable for estimating the wire costs in the planning stage of a building, which is a tedious counting and estimating task. While the final hypothesis generated by the proposed method may not match the real cable runs, the overall length of the generated hypothesis is a good approximation of the needed amount.

The method depends currently on several assumptions, e.g. that the real installation always complies with technical standards, and that the total wiring length is minimized which might not always be possible or the most feasible approach in practice. Such additional known constraints can nevertheless be easily integrated in the system, by adding more root connection nodes to the graph before final hypothesis extraction.

7 Conclusion

In this thesis, I presented contributions to the field of forward and inverse generative modeling, with the main application to extract geometric and semantic shape information from real-world measurements. In this chapter, I summarize the findings, and give a brief outlook on future work. I conclude with some open questions that surfaced while developing the methods presented in this thesis.

7.1 Contributions

I presented solutions utilizing generative modeling in the context of building outdoor and indoor reconstruction. The main contributions are:

- A new generative forward modeling system, based on shape grammars, for planar bounded objects that utilizes a novel shape representation – convex polyhedra – for geometric queries. The system provides improved generalizability of rules that allow adapting rules to a larger family of shapes. I presented an analysis to achieve an implementation that balances between evaluation speed and robustness. The system was applied and evaluated in various applications for forward and inverse generative modeling.

- I presented a novel method to extract two-dimensional building structure, using a shape grammar as façade prior. The method builds on the results of a pixel-wise classifier that identifies terminal symbols, such as walls, windows or doors. Inspired by compiler construction methods, the method finds an optimal hierarchy of rule applications – the parse tree. This means that the search space for finding façade structure is constrained by all possible words of the façade shape grammar prior. In addition, I presented a method to automatically convert such parse trees into a generative forward model that constructs the given façade. Such a generative representation has several advantages in comparison to a static 3D model: First of all, generative description is much smaller in file size than the geometry that results from evaluation; Furthermore, parts of the generated model, such as window or door types can easily be replaced by replacing the according rules. Due to the structural representation, it is even possible to exchange an arbitrary part of the hierarchy, such as symmetric parts. Structural elements such as windows or doors are not static geometry, but parametric rules that adapt to the space they are applied to.

- The third contribution is a system that creates a hypothesis of electrical wiring in indoor situations, given a pre-processed terrestrial laser scan, and a shape grammar that encodes the technical standards that apply to the building – the placement of electrical installation zones. The hypothesis was integrated into a first fully automatic pipeline that pre-processes the laser scan to extract augmented floor plans first. Then, color information from the laser scanner is projected on wall segments and observable endpoints of electrical wiring, such as sockets or switches, are detected using methods of computer vision. I presented a context-sensitive attribute grammar that produces installation zones from specific arrangements of detected endpoints, and a procedure to extract a probable wiring configuration from the graph that is created from the arrangement of all installation zones. While the hypothesis may not be perfect and can only be really verified by actually opening up walls or by using special detection devices, the proposed method helps by providing an initial layout that helps stakeholders to visualize how an installation would look if it adheres to technical standards. Furthermore, the total wire length of the extracted final graph can be used as a good estimate for the value of wiring inside walls of old buildings, or to estimate wiring material costs for planning of new buildings.

7.2 Outlook and Future Work

In the following section I will give a brief outlook on improvements and future developments of the proposed methods.

shape grammars on convex polyhedra : The representable objects are currently constrained to planar surfaces due to the planar half space kernel. A more general representation of arbitrary general half spaces $\mathbb{R}^3 \to \mathbb{R} : f(x) \leq 0$, not just those bounded by linear inequalities, are an interesting direction of future research. The surface extraction step for mesh export described in Section 3.4 is a post processing step for export after evaluation of a model. This step is needed as the internal geometric representation does not directly represent adjacency information of neighboring polyhedra. Therefore, an interesting area of research would be the integration of volumetric mesh representations, e.g. a cell complex of convex cells, to represent the context of volumetric shape configurations with adjacency.

façade reconstruction : A natural choice for future work is the extension of the proposed approach from single façade parsing to parsing of complete buildings. At this point it is not clear if a context-free grammar is suitable to represent general building structure, and how to extend the presented two-dimensional approach to 3D point clouds or similar representations. Moreover, an interesting direction of future research would be the extraction of such grammars from examples using a machine learning approach, while still being able to steer some high-level constraints that let the system capture the desired design intent. In the context for façade grammars as described in this thesis, the question arises if such a method could also learn the shape operations (symmetries) or if the application is limited to the learning of parameters or weights for rule application of given grammars.

indoor wiring hypothesis : The next logical step for the presented approach includes the integration of manual, visual editing of intermediate steps of the pipeline. A visual editor of grammar rules could greatly improve the user experience for domain experts modifying the grammar. Another interesting point of research is the integration of existing BIM data into the reconstruction approach; the electrical installation zone grammar set could be chosen automatically using the semantic relationships that are known about a building, e.g. time of construction, country, etc.

Open Questions

I conclude with a selection of the broader, general questions that arose in the process of developing the proposed methods:

"What is a suitable vocabulary for general specification of design spaces": A *design space* is a family of objects, e.g. the variations of a producible product. Most works utilize coding a generative representation of such a design space in an imperative scripting language – which means the control flow is explicitly specified. An interesting direction for future research would be to look at methods from declarative programming and see if it is possible to find a usable vocabulary that can represent such design spaces. My conjecture is that the constraints that restrict the degree of freedom, the *shape dependency* and functional or other requirements, may indeed lead to such vocabularies.

"Is there a general hierarchical high-level description for generative 3D shape models" The term generative model is broad as it contains any generative algorithmic description. It ranges from scripting languages to more constrained descriptions, such as context-free grammars. The latter seems to be too constrained to capture the desired degrees of freedom for a forward generative model – thus, the question arises if there is a general, hierarchical structural high-level description that is able to describe continuous and discrete variations in a model. The answer to this question may also be related to the foregoing question about design spaces.

"What are better low level shape representations for generative models": Using convex polyhedra still has limitations, as non-convex shapes must be represented by union of convex elements. In this context, looking for better shape representations and corresponding suitable geometric queries to facilitate reusable, robust rules is a preferred direction for future research. This is related to the persistent naming problem from parametric CAD – how to name geometric entities and parts robustly such that they can be identified, even if the underlying model or shape changes.

8 Appendix

8.1 Selected Publications

Ulrich Krispel, Christoph Schinko, and Torsten Ullrich. A Survey of Algorithmic Shapes. *Remote Sensing*, 7(10):12763, 2015

Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. Shape Grammars on Convex Polyhedra. *Computers & Graphics*, 37:707–717, 2013

Hayko Riemenschneider, Ulrich Krispel, Wolfgang Thaller, Michael Donoser, Sven Havemann, Dieter W. Fellner, and Horst Bischof. Irregular Lattices for Complex Shape Grammar Facade Parsing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1640–1647, 2012

Ulrich Krispel, Henrik Leander Evers, Martin Tamke, and Torsten Ullrich. Data completion in building information management: electrical lines from range scans and photographs. *Visualization in Engineering*, 5(1):4, March 2017

8.2 All Publications

All participated publications that have been written prior or in the course of this thesis:

1. Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner. Distance Calculation Between a Point and a Subdivision Surface. In *VMV*, pages 161–170, 2007

2. Torsten Ullrich, Ulrich Krispel, and Dieter W. Fellner. Compilation of Procedural Models. *Proceeding of the 13th International Conference on 3D Web Technology*, 13:75–81, 2008

3. Bernhard Hohmann, Ulrich Krispel, Sven Havemann, and Dieter W. Fellner. CityFit: High-Quality Urban Reconstructions by Fitting Shape Grammars to

Images and Derived Textured Point Clouds. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 3:61–68, 2009

4. Bernhard Hohmann, Sven Havemann, Ulrich Krispel, and Dieter W. Fellner. A GML Shape Grammar for Semantically Enriched 3D Building Models. *Computers & Graphics*, 34(4):322–334, 2010. Procedural Methods in Computer Graphics; Illustrative Visualization

5. Ulrich Krispel, Sven Havemann, and Dieter W. Fellner. FaMoS – A Visual Editor for Hierachical Volumetric Modeling. In *Tagungsband 05. Kongress Multimediatechnik Wismar*, pages 1–6, 2010

6. Wolfgang Thaller, Ulrich Krispel, Sven Havemann, Ivan Redi, Andrea Redi, and Dieter Fellner. Developing Parametric Building Models - the GANDIS Use Case. In Fabio Remondino and Sabry El-Hakim, editors, *Proceedings of the 4th ISPRS International Workshop 3D-ARCH 2011*. ISPRS, 2011

7. Hayko Riemenschneider, Ulrich Krispel, Wolfgang Thaller, Michael Donoser, Sven Havemann, Dieter W. Fellner, and Horst Bischof. Irregular Lattices for Complex Shape Grammar Facade Parsing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1640–1647, 2012

8. René Berndt, Christoph Schinko, Ulrich Krispel, Volker Settgast, Sven Havemann, Eva Eggeling, and Dieter W. Fellner. Ring's Anatomy – Parametric Design of Wedding Rings. In *CONTENT 2012*, pages 72–78. Xpert Publishing Services, Wilmington, USA, 2012

9. W. Thaller, U. Krispel, S. Havemann, and D. Fellner. Implicit Nested Repetition in Dataflow for Procedural Modeling. *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 3:45–50, 2012

10. René Zmugg, Ulrich Krispel, Wolfgang Thaller, Sven Havemann, Martin Pszeida, and Dieter W. Fellner. A New Approach for Interactive Procedural Modelling in Cultural Heritage. In *Proceedings of the 40th Conference of Computer Applications and Quantitative Methods in Archaeology*, 2012. to appear

11. Wolfgang Thaller, René Zmugg, Ulrich Krispel, Martin Posch, Sven Havemann, and W. Fellner Dieter. Creating Procedural Window Building Blocks using the Generative Fact Labeling Method. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 5:235–242, 2013

12. Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. Shape Grammars on Convex Polyhedra. *Computers & Graphics*, 37:707–717, 2013

13. René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. Deformation-Aware Split Grammars for Architectural Models. *Proceedings of the International Conference on Cyberworlds*, 11:4–11, 2013

14. René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. Procedural Architecture using Deformation-Aware Split Grammars. *The Visual Computer*, 12:1–11, 2013

15. Johannes Edelsbrunner, Ulrich Krispel, Sven Havemann, Alexei Sourin, and Dieter W. Fellner. Constructive Roof Geometry. *Proceedings of the International Conference on Cyberworlds*, 12:63–70, 2014

16. Ulrich Krispel, Christoph Schinko, and Torsten Ullrich. The Rules Behind – Tutorial on Generative Modeling. *Proceedings of Symposium on Geometry Processing / Graduate School*, 12:21–249, 2014

17. Ulrich Krispel, Torsten Ullrich, and Dieter W. Fellner. Fast and Exact Plane-based Representation for Polygonal Meshes. In Katherine Blashki and Yingcai Xiao, editors, *Proceedings of the 8th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing 2014,*, Lisbon, Portugal, 2014. International Association for Development of the Information Society, IADIS Press

18. Ulrich Krispel, Henrik Leander Evers, Martin Tamke, Robert Viehauser, and Dieter W. Fellner. Automatic Texture and Orthophoto Generation from Registered Panoramic Views. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:131–137, 2015

19. Christoph Schinko, Ulrich Krispel, Torsten Ullrich, and Dieter W. Fellner. Built by Algorithms - State of the Art Report on Procedural Modeling. In *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, volume XL-5/W4, pages 469–479. International Society for Photogrammetry and Remote Sensing, 2015

20. Ulrich Krispel, Christoph Schinko, and Torsten Ullrich. A Survey of Algorithmic Shapes. *Remote Sensing*, 7(10):12763, 2015

21. Ulrich Krispel, Henrik Leander Evers, Martin Tamke, and Torsten Ullrich. An Automatic Hypothesis of Electrical Lines from Range Scans and Photographs. In Nobuyoshi Yabuki and Koji Makanae, editors, *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, pages 815–822, July 2016

22. Ulrich Krispel, Henrik Leander Evers, Martin Tamke, and Torsten Ullrich. Data completion in building information management: electrical lines from range scans and photographs. *Visualization in Engineering*, 5(1):4, March 2017

8.3 Parsing Example

Example Grammar

This is an example grammar for general buildings up to 7 floors with symmetric columns of similar window splits and a mirrored column structures around a center structure with an optional cellar windows in the ground floor and a door in the center structure.

```
S ::= sized(WALL,0,0,200,9999) | F | sized(WALL,0,0,200,9999)
F ::= F(1)
F ::= F(2)
F ::= F(3)
F ::= F(4)
F ::= F(5)
F ::= F(6)
F ::= F(7)


F(n) ::= F1(n,0)
F(n) ::= F1(n,1)

F1(n,cellar) ::= CENTERREGULAR(n,cellar)
F1(n,cellar) ::= (REGULAR(WINDOW_COLUMN(n) - WINDOW_ROW(1,cellar)) | WALL) $ CENTERREGULAR(n,cellar)


REGULAR(x) ::= x
REGULAR(x) ::= halt(x, WALL)

CENTERREGULAR(n,cellar) ::= CENTERREGULAR(n,1,cellar)
CENTERREGULAR(n,cellar) ::= CENTERREGULAR(n,2,cellar)
CENTERREGULAR(n,cellar) ::= CENTERREGULAR(n,3,cellar)
CENTERREGULAR(n,cellar) ::= CENTERREGULAR(n,4,cellar)
CENTERREGULAR(n,cellar) ::= CENTERREGULAR(n,5,cellar)


CENTERREGULAR(n, 1, cellar) ::= WINDOW_COLUMN(n) - GROUNDFLOOR(1,cellar)
CENTERREGULAR(n,m >= 2, cellar) ::= halt(WINDOW_COLUMN(n), WALL, 2*m - 1) - GROUNDFLOOR(m,cellar)

GROUNDFLOOR(m, cellar) ::= WINDOW_ROW(m, cellar)
GROUNDFLOOR(m >= 3, cellar) ::= WINDOW_ROW((m-1) / 2, cellar) $ DOORTILE
GROUNDFLOOR(m < 3, cellar) ::= DOORTILE
```

```


DOORTILE ::= WALL $ DOOR2
DOORTILE {0, 0.7} ::= WALL | DOOR2 | WALL | (WIN - WALL) | WALL
DOORTILE {0, 0.7} ::= WALL | (WIN - WALL) | WALL | DOOR2 | WALL

DOOR2 ::= DOOR
DOOR2 ::= WALL - DOOR

WINDOW_ROW(1, 0) ::= WIN - WALL
WINDOW_ROW(1, 1) ::= WIN - WALL - CELLARWIN
WINDOW_ROW(m > 1, cellar) ::= halt(WINDOW_ROW(1,cellar), WALL, 2*m - 1)

WINDOW_COLUMN(0) ::= WALL
WINDOW_COLUMN(n > 0) ::= WALL - WIN - WINDOW_COLUMN(n-1)

WALL ::= t0
WIN ::= t1
CELLARWIN ::= sized(t1, 6,6,20,30)
DOOR ::= t3
```

Example Parse Tree

Automatically generated GML code of the parse tree of the façade shown in Figure 4.5:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <Library name="facade_0_0053403_0053679_grid">
    <item id="main" exec="1">usereg
   Userlib /ShapeGrammar known not {
      Userlib begin "Shape-Grammar-CP-05.xgml" dictfromxgml edef end
      ShapeGrammar.Tools.init
   } if
Userlib begin "Shape-Grammar-CP-05.xgml" dictfromxgml edef end

ShapeGrammar.Tools.init

dict begin

% reset terminal assignment
[ /t0 /t1 /t2 /t3 /t4 /t5 /t101 /t102 /t103 /t201 /t202 /t203 ] { { 4
    set-material F } def } forall

% assign specific parametric rules to terminals
/t0   { F } def
/t1 { 3 0 0.05 0.03 AA.window.rect } def
/t2 { -0.03 -0.4 0.05 0 3 0.2 0.05 8 41 41 12 AA.door.rect-g } def

/t101 { 3 0 0.05 0.03 AA.window.rect } def
/t102 { 9 2 0 0.05 0.03 AA.window.arched } def
```

```
/t103 { -0.03 -0.2 0.0   0 10 4 0.6 0.01 0.03   8 8 41 46 AA.door.arched
    -g } def
/t201 { -0.03 -0.4 0.05 0 3 0.2 0.05 8 41 41 12 AA.door.rect-g } def
/t202 { -0.03 -0.4 0.05 0 10 2 1.0 0.2 0.03 8 3 41 12 AA.door.arched-g }
     def
/t203 { -0.03 -1.0 0.0   0 10 0 0.6 0.01 0.03   8 8 41 2 AA.door.arched-
    g } def

facade_0_0053403_0053679_grid.construct
 end
(11.5,-30,12.95) (0,-1,0) (0,0,1) (1,0,0) !d !c !b !a :a dup :b sub :c
    opengl-camera    </item>
    <item id="construct" exec="1">
% code generated from parse tree
usereg (0,0,0) (23,1,25.9) 3 scope-box
[ -35 -425 ] CPXN split-interval
   [ -386 -39 ] CPXN split-interval
         t0
            [-101 -185 -101 ] mirror-h 2 {
               [ -38 -63 ] CPXN split-interval
                  [ -28 -35 ] CPXN split-interval
                  -1 CPXN subdivide 1 {
                  [ -7 -28 ] CPXN split-interval
                     [ -99 -419 ] CPZ split-interval
                        [ -192 -227 ] CPZ split-interval
                           [ -48 -179 ] CPZ split-interval
                              t0
                                 t1
                           [ -114 -78 ] CPZ split-interval
                              [ -47 -31 ] CPZ split-interval
                                 t0
                                    t1
                              [ -36 -78 ] CPZ split-interval
                                 [ -47 -31 ] CPZ split-interval
                                    t0
                                       t1
                                    t0
                        [ -58 -41 ] CPZ split-interval
                              t1
                           t0
                     t0
                  } repeat
                     [ -99 -419 ] CPZ split-interval
                        [ -192 -227 ] CPZ split-interval
                           [ -48 -179 ] CPZ split-interval
                              t0
                                 t1
                           [ -114 -78 ] CPZ split-interval
                              [ -47 -31 ] CPZ split-interval
                                 t0
                                    t1
                  [ -36 -78 ] CPZ split-interval
                     [ -47 -31 ] CPZ split-interval
                        t0
                           t1
                        t0
            [ -58 -41 ] CPZ split-interval
                  t1
               t0
      t0
} repeat
      [ -99 -419 ] CPZ split-interval
         [ -30 -155 ] CPXN split-interval
         -3 CPXN subdivide 3 {
         [ -21 -30 ] CPXN split-interval
            [ -192 -227 ] CPZ split-interval
               [ -48 -179 ] CPZ split-interval
                  t0
                     t1
               [ -114 -78 ] CPZ split-interval
                  [ -47 -31 ] CPZ split-interval
                     t0
                        t1
                  [ -36 -78 ] CPZ split-interval
                     [ -55 -23 ] CPZ split-interval
                        t0
                           t1
                        t0
            t0
         } repeat
            [ -192 -227 ] CPZ split-interval
               [ -48 -179 ] CPZ split-interval
                  t0
                     t1
               [ -114 -78 ] CPZ split-interval
                  [ -47 -31 ] CPZ split-interval
                     t0
                        t1
                  [ -36 -78 ] CPZ split-interval
                     [ -55 -23 ] CPZ split-interval
                        t0
                           t1
                        t0
            [-30 -119 -30 ] mirror-h 2 {
               [ -58 -41 ] CPZ split-interval
                     t1
                  t0
            } repeat
               [-38 -47 -38 ] mirror-h 2 {
                  t0
               } repeat
                     t3
      t0
    </item>
</Library>
```

8.4 Example Grammar for Electrical Installation Zones

Example Grammar for installation zone reconstruction based on the technical standard DIN18015 [44]. Each rule may specify a condition; the right hand side (rhs) of a rule emits one or more terminal or non-terminal labels with attribute specifications. Non context-free rules are specified by "NT:NT", *BB* corresponds to a bounding box operator.

```
{
  "WALL": [{
    "rhs": [{
        "label": "vzone",
        "attributes": {
          "pos": "left+150",
          "wallid": "id"
        }
      },
      {
        "label": "vzone",
        "attributes": {
          "pos": "left+width-150",
          "wallid": "id"
        }
      },
      {
        "label": "hzone",
        "attributes": {
          "pos": "top+150",
          "wallid": "id"
        }
      },
      {
        "label": "hzone",
        "attributes": {
          "pos": "top+height-150",
          "wallid": "id"
        }
      },
      {
        "label": "wall",
        "attributes": {
          "zone_width": "200"
        }
      }
    ]
```

```
}],

"DOOR": [{
  "rhs": [{
      "label": "vzone",
      "attributes": {
        "pos": "left-150"
      }
    },
    {
      "label": "vzone",
      "attributes": {
        "pos": "left+width+150"
      }
    },
    {
      "label": "door"
    }
  ]
}],

"WINDOW": [{
  "rhs": [{
      "label": "vzone",
      "attributes": {
        "pos": "left-150"
      }
    },
    {
      "label": "vzone",
      "attributes": {
        "pos": "left+width+150"
      }
    },
    {
      "label": "window"
    }
  ]
}],

"SOCKET": [{
  "rhs": [{
    "label": "socket",
    "attributes": {
      "endpoint": "true"
    }
  }]
}],

"SWITCH": [{
  "rhs": [{
      "label": "switch",
      "attributes": {
        "endpoint": "true"
      }
    }]
  }],

  "ROOT": [{
    "rhs": [{
      "label": "root",
      "attributes": {
        "endpoint": "true",
        "root": "true"
      }
    }]
  }],

  "VGROUP": [{
      "condition": "height > 350",
      "rhs": [{
          "label": "vzone",
          "attributes": {
            "pos": "left+width/2"
          }
        },
        {
          "label": "vgroup"
        }
      ]
    },
    {
      "rhs": [{
        "label": "vgroup"
      }]
    }
  ],


  "SWITCH:SWITCH": [{
    "vars": {
      "BBA": "BB(A)",
      "BBB": "BB(B)",
      "group": "BB(A).insertBB(BBB)"
    },
    "condition": "(DIFF(group.width(),BBA.width())<5)&&
    (DIFF(group.width(),BBB.width())<5) &&
    (group.height() < (BBA.height()+BBB.height()+50))&&
    (A.wallid==B.wallid)",
    "rhs": [{
        "label": "VGROUP",
        "attributes": {
          "left": "group.left()",
          "top": "group.top()",
          "width": "group.width()",
          "height": "group.height()",
          "wallid": "A.wallid"
        }
      },
      {
        "label": "switch",
        "attributes": {
          "left": "A.left",
          "top": "A.top",
          "width": "A.width",
          "height": "A.height",
          "wallid": "A.wallid",
          "endpoint": "true"
        }
      },
      {
        "label": "switch",
        "attributes": {
          "left": "B.left",
          "top": "B.top",
          "width": "B.width",
          "height": "B.height",
          "wallid": "B.wallid",
          "endpoint": "true"
        }
      }
    ]
  }],

  "SOCKET:SOCKET": [{
    "vars": {
      "BBA": "BB(A)",
      "BBB": "BB(B)",
      "group": "BB(A).insertBB(BBB)"
    },
    "condition": "(DIFF(group.width(),BBA.width())<5)&&
     (DIFF(group.width(),BBB.width())<5) &&
     (group.height() < (BBA.height()+BBB.height()+50))&&
     (A.wallid==B.wallid)",
    "rhs": [{
        "label": "VGROUP",
        "attributes": {
          "left": "group.left()",
          "top": "group.top()",
          "width": "group.width()",
          "height": "group.height()",
          "wallid": "A.wallid"
        }
      },
      {
        "label": "socket",
        "attributes": {
          "left": "A.left",
          "top": "A.top",
          "width": "A.width",
          "height": "A.height",
          "wallid": "A.wallid",
          "endpoint": "true"
        }
      },
      {
        "label": "socket",
        "attributes": {
          "left": "B.left",
          "top": "B.top",
          "width": "B.width",
          "height": "B.height",
          "wallid": "B.wallid",
          "endpoint": "true"
        }
      }
    ]
  }],

  "VGROUP:SWITCH": [{
    "vars": {
      "BBA": "BB(A)",
      "BBB": "BB(B)",
      "group": "BB(A).insertBB(BBB)"
    },
    "condition": "(DIFF(group.width(),BBA.width())<5)&&
     (DIFF(group.width(),BBB.width())<5) &&
     (group.height() < (BBA.height()+BBB.height()+50))&&
     (A.wallid==B.wallid)",
    "rhs": [{
        "label": "VGROUP",
        "attributes": {
          "left": "group.left()",
          "top": "group.top()",
          "width": "group.width()",
          "height": "group.height()",
          "wallid": "A.wallid"
        }
      },
      {
        "label": "switch",
        "attributes": {
          "left": "B.left",
          "top": "B.top",
      "width": "B.width",
      "height": "B.height",
      "wallid": "B.wallid",
      "endpoint": "true"
     }
    }
   ]
  }],

  "VGROUP:SOCKET": [{
   "vars": {
    "BBA": "BB(A)",
    "BBB": "BB(B)",
    "group": "BB(A).insertBB(BBB)"
   },
   "condition": "(DIFF(group.width(),BBA.width())<5)&&
   (DIFF(group.width(),BBB.width())<5) &&
   (group.height() < (BBA.height()+BBB.height()+50))&&
   (A.wallid==B.wallid)",
   "rhs": [{
     "label": "VGROUP",
     "attributes": {
      "left": "group.left()",
      "top": "group.top()",
      "width": "group.width()",
      "height": "group.height()",
      "wallid": "A.wallid"
     }
    },
    {
     "label": "socket",
     "attributes": {
      "left": "B.left",
      "top": "B.top",
      "width": "B.width",
      "height": "B.height",
      "wallid": "B.wallid",
      "endpoint": "true"
     }
    }
   ]
  }],

  "VGROUP:VGROUP": [{
   "vars": {
    "BBA": "BB(A)",
    "BBB": "BB(B)",
    "group": "BB(A).insertBB(BBB)"
   },
   "condition": "(DIFF(group.width(),BBA.width())<5)&&
   (DIFF(group.width(),BBB.width())<5) &&
    (group.height() < (BBA.height()+BBB.height()+50))&&
    (A.wallid==B.wallid)",
    "rhs": [{
      "label": "VGROUP",
      "attributes": {
        "left": "group.left()",
        "top": "group.top()",
        "width": "group.width()",
        "height": "group.height()",
        "wallid": "A.wallid"
      }
    }]
  }]

}
```

Bibliography

[1] Behzad Abbasnejad and Hashem Izadi Moud. BIM and Basic Challenges Associated with its Definitions, Interpretations and Expectations. *International Journal of Engineering Research and Applications*, 3, 2013.

[2] Sepehr Abrishami, Jack Steven Goulding, Farzad Pour Rahimian, and Abdulkadir Ganah. Integration of BIM and Generative Design to exploit AEC Conceptual Design Innovation. *ITcon Special Issue BIM Cloud-Based Technology in the AEC Sector: Present Status and Future Trends*, 19:350–359, 2013.

[3] Pankaj K. Agarwal and Micha Sharir. Arrangements and Their Applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, 1998.

[4] Oswin Aichholzer and Franz Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In Jin-Yi Cai and ChakKuen Wong, editors, *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 117–126. Springer Berlin Heidelberg, 1996.

[5] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[6] Aleksandr Danilovich Aleksandrov. *Convex Polyhedra*. Number XII in Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2005.

[7] Daniel G. Aliaga, İlke Demir, Bedrich Benes, and Michael Wand. Inverse Procedural Modeling of 3D Models for Virtual Worlds. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, pages 16:1–16:316, New York, NY, USA, 2016. ACM.

[8] Eric Andres, Raj Acharya, and Claudio Sibata. Discrete Analytical Hyperplanes. *Graphical Models and Image Processing*, 59(5):302–309, 1997.

[9] Marios C. Angelides and Harry Agius. *Handbook of Digital Games*. John Wiley & Sons, 2014.

[10] Tilo Arens, Frank Hettlich, Christian Karpfinger, Ulrich Kockelkorn, Klaus Lichtenegger, and Hellmuth Stachel. *Mathematik*. Spektrum Akademischer Verlag, 1st edition, February 2008.

[11] Marco Attene. Direct Repair of Self-intersecting Meshes. *Graphical Models*, 76(6):658–668, 2014.

[12] Phillipa Avery, Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen. Computational Intelligence and Tower Defence Games. *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 13:1084–1091, 2011.

[13] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996.

[14] Michael F. Barnsley. *Fractals Everywhere*. Academic Press, Cambridge, Massachusetts, 2nd edition, 1993.

[15] Bruce G. Baumgart. Winged Edge Polyhedron Representation. Technical Report CS-320, Computer Science Department, School of Humanities and Sciences, Stanford University, 1972.

[16] Jakob Beetz, Ina Blümel, Stefan Dietze, B. Fetahui, Ujwal Gadiraju, Martin Hecher, Thomas Krijnen, Michelle Lindlar, Martin Tamke, Raoul Wessel, and R. Yu. *Lecture Notes in Computer Science, Edited Volume Sander Münster*, chapter DURAARK: Enrichment and Preservation of Architectural Knowledge. Springer LNCS, 2015.

[17] Johannes Behr, Patrick Dähne, Yvonne Jung, and Sabine Webel. Beyond the Web Browser – X3D and Immersive VR. *IEEE Virtual Reality Tutorial and Workshop Proceedings*, 28:5–9, 2007.

[18] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3DOM: A DOM-based HTML5/X3D Integration Model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[19] Heinzgerd Bendels, Dieter W. Fellner, and Sven Havemann. Modellierung der Grundlagen — Erweiterbare Datenstrukturen zur Modellierung und Visualisierung polygonaler Welten. In *Modeling – Virtual Worlds – Distributed Graphics*, pages 149–158. infix, 1995.

[20] René Berndt, Ina Blümel, Michael Clausen, David Damm, Jürgen Diet, Dieter W. Fellner, Christian Fremerey, Reinhard Klein, Maximilian Scherer, Tobias Schreck, Irina Sens, Verena Thomas, and Raoul Wessel. The PROBADO

Project – Approach and Lessons Learned in Building a Digital Library System for Heterogeneous Non-textual Documents. *Proceedings of the 14th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Springer*, 6273:376–383, 2010.

[21] René Berndt, Dieter W. Fellner, and Sven Havemann. Generative 3D Models: a Key to More Information within less Bandwidth at Higher Quality. *Proceeding of the 10th International Conference on 3D Web Technology*, 1:111–121, 2005.

[22] René Berndt, Christoph Schinko, Ulrich Krispel, Volker Settgast, Sven Havemann, Eva Eggeling, and Dieter W. Fellner. Ring's Anatomy – Parametric Design of Wedding Rings. In *CONTENT 2012*, pages 72–78. Xpert Publishing Services, Wilmington, USA, 2012.

[23] Gilbert Bernstein and Don Fussell. Fast, Exact, Linear Booleans. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1269–1278, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[24] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, and Bruno Lévy. Geometric Modeling Based on Polygonal Meshes. In *Eurographics Tutorial*, 2008.

[25] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[26] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[27] Frank Breuel, René Bernd, Torsten Ullrich, Eva Eggeling, and Dieter W. Fellner. Mate in 3D – Publishing Interactive Content in PDF3D. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 15:110–119, 2011.

[28] Jörg Brunsmann, Wolfgang Wilkes, Gunter Schlageter, and Matthias Hemmje. State-of-the-art of Long-term Preservation in Product Lifecycle Management. *International Journal on Digital Libraries*, 12(1):27–39, 2012.

[29] Don Brutzman. The virtual reality modeling language and Java. *Communications of the ACM*, 41(6):57–64, 1998.

[30] Don Brutzman and Leonard Daly. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[31] Suzanne F. Buchele and Richard H. Crawford. Three-dimensional Halfspace Constructive Solid Geometry Tree Construction from Implicit Boundary Representations. In *Proceedings of the Eighth ACM Symposium on Solid*

Modeling and Applications, SM '03, pages 135–144, New York, NY, USA, 2003. ACM.

[32] Luis Bustamante and Jason Gu. Localization of Electrical Outlet for a Mobile Robot Using Visual Servoing. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 1211–1214. IEEE, April 2007.

[33] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed Edges - A Scalable Representation for Triangle Meshes. *Journal of Graphics Tools*, 3(4):1–11, December 1998.

[34] Robert Chitham. *The Classical Orders of Architecture*. Elsevier, 2nd edition, 2005.

[35] Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[36] Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[37] João Comba and Bruce Naylor. *Conversion of Binary Space Partitioning Trees to Boundary Representation*, chapter Geometric Modeling: Theory and Practice, pages 286–301. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[38] Kate Compton and Michael Mateas. Procedural Level Design for Platform Games. *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, 2:109–111, 2006.

[39] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society*, volume 1, pages 886–893. IEEE, 2005.

[40] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2 edition, 2000.

[41] Oliver Deussen and Bernd Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer, 2005.

[42] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. SpiderGL: a JavaScript 3D graphics library for next-generation WWW. *Proceedings of the 15th International Conference on Web 3D Technology*, 15:165–174, 2010.

[43] Edsger W. Dijkstra. The Humble Programmer. *Communications of the ACM*, 15(10):859–866, October 1972.

[44] DIN 18015-1 - Electrical Installations in Residential Buildings. Beuth Verlag, September 2013.

[45] Low-voltage Electrical Installations - Part 7-701: Requirements for Special Installations or Locations - Locations Containing a Bath or Shower (IEC 60364-7-701:2006, modified); German Implementation HD 60364-7-701:2007, October 2008.

[46] C. Dore and M. Murphy. Semi-Automatic Modelling of Building Façades with Shape Grammars Using Historic Building Information Modelling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W1:57–64, 2013.

[47] C. Dore, M. Murphy, S. McCarthy, F. Brechin, C. Casidy, and E. Dirix. Structural Simulations and Conservation Analysis - Historic Building Information Model (HBIM). *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:351–357, 2015.

[48] Chuck Eastman, Paul Teicholz, Rafael Sacks, and Kathleen Liston. *BIM Handbook*. Wiley, 2 edition, 2011.

[49] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steve Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 2002.

[50] Bruce Eckel. *Thinking in C++: Introduction to Standard C++, Practical Programming*. Prentice Hall, 2003.

[51] Johannes Edelsbrunner, Ulrich Krispel, Sven Havemann, Alexei Sourin, and Dieter W. Fellner. Constructive Roof Geometry. *Proceedings of the International Conference on Cyberworlds*, 12:63–70, 2014.

[52] Johannes Edelsbrunner, Ulrich Krispel, Sven Havemann, Alexei Sourin, and Dieter W. Fellner. Constructive Roofs from Solid Building Primitives. *Lecture Notes in Computer Science*, 9550:17–40, 2016.

[53] J. R. Edmonds. A Combinatorial Representation for Oriented Polyhedral Surfaces. *Notices Amer. Math. Soc.*, 7:641, 1960.

[54] Dag Fjeld Edvardsen, Jakob Beetz, Henrik Leander, Thomas Krijnen, Martin Hecher, Martin Tamke, Mateusz Zwierzycki, Michael Panitz, Raoul Wessel, Richard Vock, Sebastian Ochmann, and Ujwal Gadiraju. DURAARK Deliverable D7.4 - Evaluation. `http://duraark.eu/wp-content/uploads/2016/02/DURAARK_D7.4.pdf`, January 2016.

[55] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 341–346, New York, NY, USA, 2001. ACM.

[56] Erhvervs-og Vækstministeriet. Stærkstrømsbekendtgørelsen, EK nr 9146 af 22/03/2006 Gældende (SB - Afsnit 6A) Offentliggørelsesdato, March 2006.

[57] Eurostat. Construction by Employment Size Class (NACE Rev. 2, F). `http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=sbs_sc_con_r2&lang=en`, 2013. [Online; accessed 10.06.2016].

[58] Steven Fortune and Christopher J. Van Wyk. Static Analysis Yields Efficient Exact Integer Arithmetic for Computational Geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[59] Oleg Fryazinov, Alexander Pasko, and Valery Adzhiev. BSP-Fields: An Exact Representation of Polygonal Objects by Differentiable Scalar Fields Based on Binary Space Partitioning. *Computer-Aided Design*, 43(3):265–277, 2011.

[60] Komei Fukuda and Alain Prodon. Double Description Method Revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111, London, UK, 1996. Springer-Verlag.

[61] Michael P. Gallaher, Alan C. O'Connor, Jr John L. Dettbarn, and Linda T. Gilday. Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry. `http://fire.nist.gov/bfrlpubs/build04/art022.html`, August 2004.

[62] Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. Snap Rounding Line Segments Efficiently in Two and Three Dimensions. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, pages 284–293, New York, NY, USA, 1997. ACM.

[63] Daniel H. Greene. The Decomposition of Polygons into Convex Parts. *Computational Geometry*, Adv. Comput. Res.(1):235–259, 1983.

[64] Marcus Gross and Hanspeter Fister, editors. *Point-Based Graphics*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 1st edition, June 2007. ISBN-13: 978-0123706041.

[65] Branko Grünbaum, Volker Kaibel, Vcitor Klee, and Günter M. Ziegler. *Convex Polytopes*. Graduate texts in mathematics. Springer-Verlag New York, Inc., 2 edition, 2003.

[66] John L. Gustavson. *The End of Error: Unum Computing*. Chapman and Hall, 1st edition, 2015.

[67] Jacques Hadamard. Sur les problèmes aux dérivés partielles et leur signification physique. *Princeton University Bulletin*, 13:49–52, 1902.

[68] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[69] Sven Havemann. Generative Mesh Modeling. *PhD-Thesis, Technische Universität Braunschweig, Germany*, 1:1–303, 2005.

[70] Sven Havemann and Dieter W. Fellner. Generative Mesh Modeling. *Technical Report TUBS-CG-2003-01*, 1:1–11, 2003.

[71] Sven Havemann and Dieter W. Fellner. Generative Parametric Design of Gothic Window Tracery. *Proceedings of the 5th International Symposium on Virtual Reality, Archeology, and Cultural Heritage*, 1:193–201, 2004.

[72] Sven Havemann, Volker Settgast, René Berndt, and Øyvid Eide. The Arrigo Showcase Reloaded – towards a sustainable link between 3D and semantics. *Proceedings of the 9th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, 9:125–132, 2008.

[73] Daniel Heckenberg, Joseph Hegarty, and Jean Pascal leBlanc. RepTile: How to Skin a Dinosaur. In *ACM SIGGRAPH 2014 Talks*, SIGGRAPH '14, pages 31:1–31:1, New York, NY, USA, 2014. ACM.

[74] J. L. Heiberg, editor. *Euclid's Elements of Geometry*. Fitzpatrick, Richard, 2007.

[75] Peter Henderson. Functional Geometry. *Higher Order Symbol. Comput.*, 15(4):349–365, December 2002.

[76] Dominikus Herzberg and Tim Reichert. Concatenative Programming - An Overlooked Paradigm in Functional Programming. In Boris Shishkov, José Cordeiro, and Alpesh Ranchordas, editors, *ICSOFT 2009 - Proceedings of the 4th International Conference on Software and Data Technologies, Volume 1, Sofia, Bulgaria, July 26-29, 2009*, pages 257–263. INSTICC Press, 2009.

[77] N. Hichri, C. Stefani, L. De Luca, and P. Veron. Review of the "as-built BIM" Approaches. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W1:107–112, 2013.

[78] Younis Hijazi, Hans Hagen, Charles D. Hansen, and Kenneth I. Joy. Why Interval Arithmetic is so Useful. In *Visualization of Large and Unstructured Data Sets: Second workshop of the DFG's International Research Training Group*

"Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling, and Engineering", September 9-11, 2007 Kaiserslautern, Germany, pages 148–163, 2007.

[79] Christoph M. Hoffmann. The Problems of Accuracy and Robustness in Geometric Computation. *Computer*, 22(3):31–40, March 1989.

[80] Christoph M. Hoffmann. Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1(2):143–155, 2001.

[81] Bernhard Hohmann, Sven Havemann, Ulrich Krispel, and Dieter W. Fellner. A GML Shape Grammar for Semantically Enriched 3D Building Models. *Computers & Graphics*, 34(4):322–334, 2010. Procedural Methods in Computer Graphics; Illustrative Visualization.

[82] Bernhard Hohmann, Ulrich Krispel, Sven Havemann, and Dieter W. Fellner. CityFit: High-Quality Urban Reconstructions by Fitting Shape Grammars to Images and Derived Textured Point Clouds. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 3:61–68, 2009.

[83] Daniel Huber. The ASTM E57 File Format for 3D Imaging Data Exchange. In *Proceedings of the SPIE Vol. 7864A, Electronics Imaging Science and Technology Conference (IS&T), 3D Imaging Metrology*, volume 7864A, January 2011.

[84] J.-F. Hullo, G. Thibault, and C. Boucheny. Advances in Multi-Sensor Scanning and Visualization of Complex Plants: The Utmost Case of a Reactor Building. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:163–169, 2015.

[85] Robin Hunicke and Vernell Chapman. AI for Dynamic Difficulty Adjustment in Games. *Proceedings of the Challenges in Game AI Workshop / Conference on Artificial Intelligence*, 19:91–96, 2004.

[86] Ali R. Hurson and Krishna M. Kavi. *Dataflow Computers: Their History and Future*, chapter 1, pages 1–12. John Wiley & Sons, Inc., 2007.

[87] John Hutchinson. Fractals and Self-Similarity. *Indiana Univ. Math. J.*, 30:713–747, 1981.

[88] F. K. Hwang and Dana S. Richards. Steiner Tree Problems. *Networks*, 22(1):55–89, 1992.

[89] Laurent Hyafil and Ronald L. Rivest. Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[90] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, NY, USA, August 1985.

[91] ISO 10303-1:1994 Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles, 1994.

[92] ISO 16739:2013 Industry Foundation Classes, Release 2x, Platform Specication (IFC2x Platform), 2013.

[93] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. Polymorph: A Model for Dynamic Level Generation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 6:138–143, 2010.

[94] C. Kaklamanis, Miroslav Chlebík, and Janka Chlebíková. The Steiner Tree Problem on Graphs: Inapproximability Results. *Theoretical Computer Science*, 406(3):207–214, 2008.

[95] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In Konrad Polthier and Alla Sheffer, editors, *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[96] Tom Kelly and Peter Wonka. Interactive Architectural Modeling with Procedural Extrusions. *ACM Transactions on Graphics*, 30:141–15, 2011.

[97] Lutz Kettner. Using Generic Programming for Designing a Data Structure for Polyhedral Surfaces. *Computational Geometry: Theory and Applications*, 13(1):65–90, May 1999.

[98] Patrick Knöbelreiter, René Berndt, Torsten Ullrich, and Dieter W. Fellner. Automatic fly-through Camera Animations for 3D Architectural Repositories. *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (GRAPP 2014)*, 9:335–341, 2014.

[99] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 4$^{\text{th}}$ edition, 2007.

[100] L. Kou, G. Markowsky, and L. Berman. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15(2):141–145, 1981.

[101] Lars Krecklau, Janis Born, and Leif Kobbelt. View-Dependent Realtime Rendering of Procedural Facades with High Geometric Detail. *Comput. Graph. Forum*, 32(2):479–488, 2013.

[102] Lars Krecklau and Leif Kobbelt. Procedural Modeling of Interconnected Structures. *Computer Graphics Forum*, 30(2):335–344, 2011.

[103] Lars Krecklau, Darko Pavic, and Leif Kobbelt. Generalized Use of Non-Terminal Symbols for Procedural Modeling. *Computer Graphics Forum*, 29(8):2291–2303, 2010.

[104] Ulrich Krispel, Henrik Leander Evers, Martin Tamke, and Torsten Ullrich. An Automatic Hypothesis of Electrical Lines from Range Scans and Photographs. In Nobuyoshi Yabuki and Koji Makanae, editors, *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, pages 815–822, July 2016.

[105] Ulrich Krispel, Henrik Leander Evers, Martin Tamke, and Torsten Ullrich. Data completion in building information management: electrical lines from range scans and photographs. *Visualization in Engineering*, 5(1):4, March 2017.

[106] Ulrich Krispel, Henrik Leander Evers, Martin Tamke, Robert Viehauser, and Dieter W. Fellner. Automatic Texture and Orthophoto Generation from Registered Panoramic Views. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:131–137, 2015.

[107] Ulrich Krispel, Sven Havemann, and Dieter W. Fellner. FaMoS – A Visual Editor for Hierachical Volumetric Modeling. In *Tagungsband 05. Kongress Multimediatechnik Wismar*, pages 1–6, 2010.

[108] Ulrich Krispel, Christoph Schinko, and Torsten Ullrich. The Rules Behind – Tutorial on Generative Modeling. *Proceedings of Symposium on Geometry Processing / Graduate School*, 12:21–249, 2014.

[109] Ulrich Krispel, Christoph Schinko, and Torsten Ullrich. A Survey of Algorithmic Shapes. *Remote Sensing*, 7(10):12763, 2015.

[110] Ulrich Krispel, Martin Tamke, Martin Hecher, Jakob Beetz, Stefan Dietze, and Dieter W. Fellner. *Geometric Enrichment of Digital Building Data in the DURAARK Project*, chapter European Project Space on Intelligent Technologies, Software engineering, Computer Vision, Grahics, Optics and Photonics. SCITEPRESS, 2016.

[111] Ulrich Krispel, Torsten Ullrich, and Dieter W. Fellner. Fast and Exact Plane-based Representation for Polygonal Meshes. In Katherine Blashki and Yingcai Xiao, editors, *Proceedings of the 8th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing 2014,*, Lisbon, Portugal, 2014. International Association for Development of the Information Society, IADIS Press.

[112] Zhengzheng Kuang, Bin Chan, Yizhou Yu, and Wenping Wang. A Compact Random-access Representation for Urban Modeling and Rendering. *ACM Trans. Graph.*, 32(6):172:1–172:12, November 2013.

[113] Daniel Ladenhauf, René Berndt, Eva Eggeling, Torsten Ullrich, Kurt Battisti, and Markus Gratzl-Michlmair. From Building Information Models to Simplified Geometries for Energy Performance Simulation. *Proceeding of the International Academic Conference on Places and Technologies*, 1:669–676, 2014.

[114] Daniel Ladenhauf, René Berndt, Ulrich Krispel, Eva Eggeling, Torsten Ullrich, Kurt Battisti, and Markus Gratzl-Michlmair. Geometry Simplification According to Semantic Constraints. *Computer Science – Research and Development*, 11:to appear, 2014.

[115] L. Ladický, C. Russell, P. Kohli, and P. H. S. Torr. Associative Hierarchical CRFs for Object Class Image Segmentation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 739–746, September 2009.

[116] John Lee. *Introduction to Topological Manifolds*, volume 940 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 2nd edition, 2011.

[117] Pascal Lienhardt. Topological Models for Boundary Representation: A Comparison with N-dimensional Generalized Maps. *Computer-Aided Design*, 23(1):59–82, February 1991.

[118] Michelle Lindlar. Building Information Modeling – A Game Changer for Interoperability and a Chance for Digital Preservation of Architectural Data? In Serena Coates, Ross King, Steve Knight, Christopher (Cal) Lee, Peter McKinney, Erin O'Meara, and David Pearson, editors, *Proceedings of the 11th International Conference on Digital Preservation*. IPres, 2014.

[119] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Transactions on Graphics*, 27(3):1–10, 2008.

[120] Markus Lipp, Peter Wonka, and Michael Wimmer. Parallel Generation of Multiple L-Systems. *Computers & Graphics*, 34:585–593, 2010.

[121] Mikola Lysenko, Roshan D'Souza, and Ching-Kuan Shene. Improved Binary Space Partition Merging. *Computer-Aided Design*, 40(12):1113–1120, 2008.

[122] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Co., 1982.

[123] Martti Mäntylä. *An Introduction to Solid Modeling*. Principles of Computer Science Series. Computer Science Press, Inc., New York, NY, USA, 1987.

[124] David Marcheix and Guy Pierra. A Survey of the Persistent Naming Problem. *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 7:13–22, 2002.

[125] George E. Martin. *Geometric Constructions*. Springer, 1998.

[126] Jean-Eudes Marvie, Cyprien Buron, Pascal Gautron, Patrice Hirtzlin, and Gaël Sourimant. GPU Shape Grammars. *Computer Graphics Forum*, 31(7pt1):2087–2095, September 2012.

[127] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein, Marius Muja, Victor Eruhimov, Tully Foote, et al. Autonomous Door Opening and Plugging in With a Personal Robot. In *International Conference on Robotics and Automation (ICRA), 2010 IEEE*, pages 729–736. IEEE, 2010.

[128] Guillaume Melquiond and Sylvain Pion. Formally Certified Floating-point Filters for Homogeneous Geometric Predicates. *Theoretical Informatics and Applications*, 41(1):57–69, 2007.

[129] Paul Merell. *Model Synthesis*. PhD thesis, University of North Carolina at Chapel Hill, 2009.

[130] Paul Merrell and Dinesh Manocha. Model Synthesis: A General Procedural Modeling Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 17:715–728, 2010.

[131] Andreas Meyer and Sylvain Pion. FPG: A Code Generator for Fast and Certified Geometric Predicates. In *Real Numbers and Computers*, pages 47–60, Santiago de Compostela, Espagne, 2008.

[132] William J. Mitchell. *The Logic of Architecture: Design, Computation, and Cognition*. MIT Press, 1990.

[133] Pascal Müller, Tijl Vereenooghe, Andreas Ulmer, and Luc Van Gool. Automatic Reconstruction of Roman Housing Architecture. *Recording, Modeling and Visualization of Cultural Heritage*, 1:287–298, 2006.

[134] Pascal Müller, Tijl Vereenooghe, Peter Wonka, Iken Paap, and Luc Van Gool. Procedural 3D Reconstruction of Puuc Buildings in Xkipche. *Proceedings of Eurographics Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, 1:139–146, 2006.

[135] Pascal Müller, Peter Wonka, Simon Haegler, Ulmer Andreas, and Luc Van Gool. Procedural Modeling of Buildings. *Proceedings of 2006 ACM Siggraph*, 25(3):614–623, 2006.

[136] National Institute of Building Sciences. Frequently Asked Questions About the National BIM Standard, 2014.

[137] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP Trees Yields Polyhedral Set Operations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 115–124, New York, NY, USA, 1990. ACM.

[138] W. Nef. *Beiträge zur Theorie der Polyeder: mit Anwendungen in der Computergraphik*. Beiträge zur Mathematik, Informatik und Nachrichtentechnik. Lang, 1978.

[139] Mark J. Nelson, Calvin Ashmore, and Michael Mateas. Authoring an Interactive Narrative with Declarative Optimization-Based Drama Management. *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, 2:127–129, 2006.

[140] NVidia. NVidia CUDA C Programming Guide, 2010.

[141] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. Automatic Reconstruction of Parametric Building Models from Indoor Point Clouds. *Computers & Graphics*, 54:94–103, 2016. Special Issue on CAD/-Graphics 2015.

[142] Sebastian Ochmann, Richard Vock, Raoul Wessel, Martin Tamke, and Reinhard Klein. Automatic Generation of Structural Building Descriptions from 3D Point Cloud Scans. In *GRAPP 2014 - International Conference on Computer Graphics Theory and Applications*. SCITEPRESS, January 2014.

[143] Review Board OpenGL Architecture. *OpenGL Reference Manual*. Addison-Wesley Publishing Company, 1993.

[144] Erich Ossa. *Topologie : eine anschauliche Einführung in die geometrischen und algebraischen Grundlagen*. Aufbaukurs Mathematik. Studium. Vieweg + Teubner, Wiesbaden, 2., überarb. aufl edition, 2009.

[145] John K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer Magazine*, 31(3):23–30, 1998.

[146] Mine Özkar and Sotirios Kotsopoulos. Introduction to Shape Grammars. *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 2008 (course notes)*, 36:1–175, 2008.

[147] Yogi Parish and Pascal Müller. Procedural Modeling of Cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:301–308, 2001.

[148] Terence Parr. *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Pragmatic Bookshelf, 2010.

[149] Alexander Pasko and Valery Adzhiev. Function-based Shape Modeling: Mathematical Framework and Specialized Language. *Lecture Notes in Computer Science*, 2930:132–160, 2004.

[150] Gustavo Patow. User-Friendly Graph Editing for Procedural Buildings. *Computer Graphics and Applications, IEEE*, PP(99):1, 2012.

[151] Christopher Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:54–67, 2010.

[152] P. J. Petkova and U. Rüppel. A Graph-based Prediction Method for Electrical Wiring in Old Residential Buildings as a Part of BIM for Urban Mining Purposes. *eWork and eBusiness in Architecture, Engineering and Construction*, pages 109–113, 2014.

[153] Tomas Polgar. *FREAX: The Brief History of the Computer Demoscene*. CSW-Verlag, Winnenden, 2005.

[154] Michael J. Pratt. Introduction to ISO 10303—the STEP Standard for Product Data Exchange. *Journal of Computing and Information Science in Engineering*, 1(1):102–103, 2001.

[155] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[156] R. Quattrini, E. S. Malinverni, P. Clini, R. Nespeca, and E. Orlietti. From TLS to HBIM. High Quality Semantically-aware 3D Modeling of Complex Architecture. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:367–374, 2015.

[157] Casey Reas, Ben Fry, and John Maeda. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[158] William Reilly. Crossing the Curatorial Chasm - Lessons from the FACADE Project. In *4th International Conference on Open Repositories*, 2009.

[159] Dirk Reiners, Gerrit Voss, and Johannes Behr. OpenSG: Basic Concepts. *Proceedings of OpenSG Symposium 2002*, 1:1–7, 2002.

[160] Hayko Riemenschneider, Ulrich Krispel, Wolfgang Thaller, Michael Donoser, Sven Havemann, Dieter W. Fellner, and Horst Bischof. Irregular Lattices for Complex Shape Grammar Facade Parsing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1640–1647, 2012.

[161] Hayko Riemenschneider, Sabine Sternig, Michael Donoser, Peter M. Roth, and Horst Bischof. *Hough Regions for Joining Instance Localization and Segmentation*, pages 258–271. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[162] N. Ripperda. Grammar Based Facade Reconstruction using RjMCMC. In *Photogrammetrie Fernerkundung Geoinformation (PFG)*, pages 83–92, 2008.

[163] Aitor Santamaria-Ibirika, Xabier Cantero, Sergio Huerta, Igor Santos, and Pablo G. Bringas. Procedural Playable Cave Systems based on Voronoi Diagram and Delaunay Triangulation. *Proceedings of the International Conference on Cyberworlds*, 12:15–22, 2014.

[164] Christoph Schinko, Ulrich Krispel, Torsten Ullrich, and Dieter W. Fellner. Built by Algorithms - State of the Art Report on Procedural Modeling. In *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, volume XL-5/W4, pages 469–479. International Society for Photogrammetry and Remote Sensing, 2015.

[165] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. Scripting Technology for Generative Modeling. *International Journal On Advances in Software*, 4:308–326, 2011.

[166] Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. Minimally Invasive Interpreter Construction – How to reuse a compiler to build an interpreter. *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 3:38–44, 2012.

[167] Michail I. Schlesinger, Savchynskyy Bogdan, and M.A. Anochina. Grammar Approach to Printed Notes Recogition. In *Control Systems and Computers*, 2003.

[168] Dan Sheerin. Procedural Tentacle Bundles in "Edge of Tomorrow". In *ACM SIGGRAPH 2014 Talks*, SIGGRAPH '14, pages 33:1–33:1, New York, NY, USA, 2014. ACM.

[169] Jonathan Richard Shewchuk. Robust Adaptive Floating-Point Geometric Predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery, May 1996.

[170] J. Shotton, M. Johnson, and R. Cipolla. Semantic Texton Forests for Image Categorization and Segmentation. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[171] Vaclav Skala and Vit Ondracka. A Precision of Computation in the Projective Space. In *Proceedings of the 15th WSEAS international conference on Computers*, pages 35–40, Stevens Point, Wisconsin, USA, 2011. World Scientific and Engineering Academy and Society (WSEAS).

[172] John M. Snyder and James T. Kajiya. Generative Modeling: A Symbolic System for Geometric Modeling. *Proceedings of 1992 ACM Siggraph*, 1:369–378, 1992.

[173] Celestino Soddu and Enrica Colabella. The 19th Generative Art Conference. In *Proceedings of the 19th Generative Art Conference*, 2016.

[174] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. Inverse Procedural Modelling of Trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

[175] Ondrej Stava, Bedrich Benes, Radomir Mech, Daniel G. Aliaga, and Peter Kristof. Inverse Procedural Modeling by Automatic Generation of L-systems. *Proceedings of EUROGRAPHICS, Computer Graphics Forum*, 29:665–674, 2010.

[176] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Jörg Müller, Wonka Peter, and Dieter Schmalstieg. Parallel Generation of Architecture on the GPU. *Computer Graphics Forum*, 33:73–82, 2014.

[177] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Peter Wonka, and Dieter Schmalstieg. On-the-fly Generation and Rendering of Infinite Cities on the GPU. *Comput. Graph. Forum*, 33(2):105–114, 2014.

[178] George Stiny and James Gips. Shape Grammars and the Generative Specification of Painting and Sculpture. *Best computer papers of 1971*, 1:125–135, 1971.

[179] K. Sugihara and M. Iri. A Solid Modelling System Free from Topological Inconsistency. *J. Inf. Process.*, 12(4):380–393, April 1990.

[180] Kokichi Sugihara. Robust Geometric Computation Based on Topological Consistency. In Vassil Alexandrov, Jack Dongarra, Benjoe Juliano, Renė Renner, and C. Tan, editors, *Computational Science - ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*, pages 12–26. Springer Berlin / Heidelberg, 2001.

[181] Ivan E. Sutherland and Gary W. Hodgman. Reentrant Polygon Clipping. *Communications of the ACM*, 17(1):32–42, January 1974.

[182] Richard Szeliski. Image Alignment and Stitching: A Tutorial. Technical Report MSR-TR-2004-92, Microsoft Research, October 2004.

[183] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomir Mech, and Vladlen Koltun. Metropolis Procedural Modeling. *ACM Transactions on Graphics*, 30:111–14, 2011.

[184] Martin Tamke, Ina Blümel, Sebastian Ochmann, Richard Vock, and Raoul Wessel. From Point Clouds to Definitions of Architectural Space - Potentials of Automated Extraction of Semantic Information from Point Clouds for the Building Profession. In Emine Mine Thompson, editor, *Proceedings of the 32nd eCAADe Conference*, volume 2, pages 557–566, 2014.

[185] O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios. Shape Grammar Parsing via Reinforcement Learning. In *CVPR 2011*, pages 2273–2280, June 2011.

[186] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios. Segmentation of Building Facades Using Procedural Shape Priors. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3105–3112, June 2010.

[187] W. Thaller, U. Krispel, S. Havemann, and D. Fellner. Implicit Nested Repetition in Dataflow for Procedural Modeling. *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 3:45–50, 2012.

[188] Wolfgang Thaller, Ulrich Krispel, Sven Havemann, Ivan Redi, Andrea Redi, and Dieter Fellner. Developing Parametric Building Models - the GANDIS Use Case. In Fabio Remondino and Sabry El-Hakim, editors, *Proceedings of the 4th ISPRS International Workshop 3D-ARCH 2011*. ISPRS, 2011.

[189] Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. A Graph-Based Language for Direct Manipulation of Procedural Models. *International Journal on Advances in Software*, 6:225–236, 2013.

[190] Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. Shape Grammars on Convex Polyhedra. *Computers & Graphics*, 37:707–717, 2013.

[191] Wolfgang Thaller, René Zmugg, Ulrich Krispel, Martin Posch, Sven Havemann, and W. Fellner Dieter. Creating Procedural Window Building

Blocks using the Generative Fact Labeling Method. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 5:235–242, 2013.

[192] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.7 edition, 2015.

[193] Robert F. Tobler, Stefan Maierhofer, and Alexander Wilkie. A Multiresolution Mesh Generation Approach for Procedural Definition of Complex Geometry. *Proceedings of the Shape Modeling International*, 6:35–44, 2002.

[194] Robert F. Tobler, Stefan Maierhofer, and Alexander Wilkie. Mesh-Based Parametrized L-Systems and Generalized Subdivision for Generating Complex Geometry. *International Journal of Shape Modeling*, 8:173–191, 2002.

[195] Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Making Racing Fun Through Player Modeling and Track Evolution. *Proceedings of the Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 6:61–70, 2006.

[196] Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Towards Automatic Personalised Content Creation for Racing Games. *IEEE Symposium on Computational Intelligence and Games*, 11:252–259, 2007.

[197] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:172–186, 2011.

[198] W. T. Tutte. Combinatorial Oriented Maps. *Can. J. Math*, XXXI(5):986–1004, 1979.

[199] Torsten Ullrich. Reconstructive Geometry. *PhD-Thesis, Technische Universität Graz, Austria*, 1:1–322, 2011.

[200] Torsten Ullrich, Ulrich Krispel, and Dieter W. Fellner. Compilation of Procedural Models. *Proceeding of the 13th International Conference on 3D Web Technology*, 13:75–81, 2008.

[201] Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner. Distance Calculation Between a Point and a Subdivision Surface. In *VMV*, pages 161–170, 2007.

[202] Torsten Ullrich, Nelson Silva, Eva Eggeling, and Dieter W. Fellner. Generative Modeling and Numerical Optimization for Energy Efficient Buildings. *Proceedings of IEEE / OCG Energy Informatics*, 2:123–128, 2013.

[203] Gino van den Bergen. Efficient Collision Detection of Complex Deformable Models Using AABB Trees. *J. Graph. Tools*, 2(4):1–13, January 1998.

[204] George Vaněček. BREP-INDEX: A Multidimensional Space Partitioning Tree. *International Journal of Computational Geometry & Applications*, 01(03):243–261, 1991.

[205] Paul Viola and Michael Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[206] Gerrit Voß, Johannes Behr, Dirk Reiners, and Marcus Roth. A Multi-thread Safe Foundation for Scene Graphs and its Extension to Clusters. *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, 4:33–37, 2002.

[207] Charlie C.L. Wang and Dinesh Manocha. Efficient Boundary Extraction of BSP Solids Based on Clipping Operations. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2012.

[208] Kevin Weiler. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[209] Eric Weisstein. *MathWorld – A Wolfram Web Resource*. Wolfram Research, 2009.

[210] Tim Weyrich, Jason Lawrence, Hendrik P. A. Lensch, Szymon Rusinkiewicz, and Todd Zickler. Principles of Appearance Acquisition and Representation. *Found. Trends. Comput. Graph. Vis.*, 4(2):75–191, February 2009.

[211] Robin J Wilson. *Introduction to Graph Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[212] Eric Wing. *Autodesk Revit Architecture 2016: No Experience Required*. Sybex, 1st edition edition, 2015. ISBN-13: 978-1119059530.

[213] Peter Wonka, Michael Wimmer, Francois Sillion, and William Ribarsky. Instant Architecture. *International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2003*, 22(3):669–677, 2003.

[214] C.K. Yap. Robust Geometric Computation. In O'Rourke J. Goodman J. E., editor, *Handbook of Discrete and Computational Geometry*, pages 927–952. Chapman and Hall/CRC, 2004.

[215] René Zmugg, Ulrich Krispel, Wolfgang Thaller, Sven Havemann, Martin Pszeida, and Dieter W. Fellner. A New Approach for Interactive Procedural Modelling in Cultural Heritage. In *Proceedings of the 40th Conference of Computer Applications and Quantitative Methods in Archaeology*, 2012. to appear.

[216] René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. Deformation-Aware Split Grammars for Architectural Models. *Proceedings of the International Conference on Cyberworlds*, 11:4–11, 2013.

[217] René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. Procedural Architecture using Deformation-Aware Split Grammars. *The Visual Computer*, 12:1–11, 2013.